PaddleX
1. Know PaddleX Quickly

1  Quick start within 10 minutes  3
2  Quick installation  7
3  Data Annotation, Transformation and Division  9
4  Data Format Description  17
5  Model Training  27
6  Loading a model for inference  33
7  Training parameter adjustment  39
8  Model Saving  43
9  Model Clipping  45
10 Model Quantification  49
11 Deployment model export  51
12 Lightweight service-oriented deployments  53
13 CPU / GPU (encryption) deployment  59
14 Nvidia Jetson Development Board  85
15 Android platform  91
16 Raspberry Pi Deployment  99
17 OpenVINO Deployment  107
PaddleX is an Entire Process Development Toolkit for Deep Learning based on the core frameworks, development kits and tool components of the PaddlePaddle. It has three major characteristics of “whole process connection”, “integration of industrial practice” and “ease of use and integration”.

- Official Website: http://www.paddlepaddle.org.cn/paddle/paddlex
- GitHub: https://github.com/PaddlePaddle/PaddleX
- Official QQ Chat Group: 1045148026
- GitHub Issue: http://www.github.com/PaddlePaddle/PaddleX/issues

1. Note: The user manual may be compatible with some formats when printed as PDF;
2. Note: This document is continuously updated at http://paddlex.readthedocs.io/.
1. Know PaddleX Quickly
Quick start within 10 minutes

This document shows how to perform training on a small dataset through PaddleX. This example is synchronized to AIStudio. You can directly [experience this model training online](https://aistudio.baidu.com/aistudio/projectdetail/450220).

The codes of this example are derived from Github [tutorials/train/classification/mobilenetv3_small_ssld.py](https://github.com/PaddlePaddle/PaddleX/blob/develop/tutorials/train/image_classification/mobilenetv3_small_ssld.py). You can download and run them locally.

All model trainings in PaddleX follow the following three steps to quickly finish the development of training codes.

**Note**: The transforms, datasets and training parameters of different models are quite different. For more model trainings, you can get more model training codes directly from the tutorial. [Model training tutorial](train/index.html)

Other usages of PaddleX

- Use VisualDL to view an index change during training
- Load a model saved during training and perform inference

1 Install PaddleX

For the installation-related process and problems, refer to the PaddleX [installation document]. (.*/install.md)

```
pip install paddlex -i https://mirror.baidu.com/pypi/simple
```
2 Prepare a vegetable classification dataset

```
wget https://bj.bcebos.com/paddlex/datasets/vegetables_cls.tar.gz
tar xzvf vegetables_cls.tar.gz
```

3 Define a training/validation image processing flow transforms

Model data processing flows must be respectively defined during training and validation because data enhancement operations are added during training. RandomCrop and RandomHorizontalFlip data enhancement methods are added in train_transforms', as shown in the following codes. For more methods, refer to the [data enhancement document](apis/transforms/augment.md).

```
from paddlex.cls import transforms
train_transforms = transforms.Compose([  
    transforms.RandomCrop(crop_size=224),  
    transforms.RandomHorizontalFlip(),  
    transforms.Normalize()  
])
eval_transforms = transforms.Compose([  
    transforms.ResizeByShort(short_size=256),  
    transforms.CenterCrop(crop_size=224),  
    transforms.Normalize()  
])
```

4 Define a dataset and load an image classification dataset

Define a dataset. `pdx.datasets. ImageNet` indicates reading a classification dataset in ImageNet format

- `pdx.datasets. ImageNet API description`
- `ImageNet data format description`

```
train_dataset = pdx.datasets.ImageNet(  
    data_dir='vegetables_cls',  
    file_list='vegetables_cls/train_list.txt',  
    label_list='vegetables_cls/labels.txt',  
    transforms=train_transforms,  
    shuffle=True)  
eval_dataset = pdx.datasets.ImageNet(  
    data_dir='vegetables_cls',  
    file_list='vegetables_cls/val_list.txt',  
    label_list='vegetables_cls/labels.txt',  
    transforms=eval_transforms)
```

5 Start training using the MobileNetV3_small_ssld model
In this document, the MobileNetV3 pre-training model obtained by Baidu based on the distillation method is used. The model structure is the same as MobileNetV3, but the precision is higher. PaddleX has more than 20 built-in classification models. For the details of more classification models, refer to the [PaddleX model library](appendix/model_zoo.md).

```python
num_classes = len(train_dataset.labels)
model = pdx.cls.MobileNetV3_small_ssld(num_classes=num_classes)

model.train(num_epochs=20,
            train_dataset=train_dataset,
            train_batch_size=32,
            eval_dataset=eval_dataset,
            lr_decay_epochs=[4, 6, 8],
            save_dir='output/mobilenetv3_small_ssld',
            use_vdl=True)
```

6 View a change in training indexes using VisualDL during training

Model indexes on both the training and validation sets are outputted to a command terminal in the form of standard output stream during training. When you set `use_vdl=True`, indexes are also sent to the `vdl_log` folder in the `save_dir` directory in VisualDL format. Run the following command in the terminal to start `visualdl` and view a visual index change.

```
visualdl --logdir output/mobilenetv3_small_ssld --port 8001
```

After the service is started, open https://0.0.0.0:8001 or https://localhost:8001 on the browser.

If you use the AIStudio platform for training, you cannot start visualdl using this method. Refer to the AIStudio VisualDL start tutorial

7 Load a model saved during training and perform inference

A model is saved every certain number of rounds during training. The round with the best evaluation on the validation set is saved in the `best_model` folder in the `save_dir` directory. The following method is used to load a model and perform inference.

- `load_model` API description
- `predict` API description for a classification model

```python
import paddlex as pdx
model = pdx.load_model('output/mobilenetv3_small_ssld/best_model')
result = model.predict('vegetables_cls/bocai/100.jpg')
print("Predict Result:", result)
```

The inference results are outputted as follows:
Predict Result: [
    {
        'score': 0.9999393,
        'category': 'bocai',
        'category_id': 0
    }
]

More tutorials

- 1 Object detection model training
- 2 Semantic segmentation model training
- 3 Instance segmentation model training
- 4 If a model is too large and you want to have a small model, try to prune it.
Quick installation

By default, the following installation process supposes that you have installed paddlepaddle-gpu or paddlepaddle (version greater than or equal to 1.8.1)**. For the paddlepaddle installation method, refer to the official website of [PaddlePaddle](https://www.paddlepaddle.org.cn/install/quick).

### 2.1 pip installation

Note that the pycocotools installation in Windows is special. Refer to the following installation command in Windows

```
pip install paddlex -i https://mirror.baidu.com/pypi/simple
```

### 2.2 Anaconda installation

Anaconda is an open source Python released version which contains more than 180 science packages and their dependencies such as conda and Python. By creating multiple independent Python environments, the use of Anaconda can avoid conflict due to too many different version dependencies installed in your Python environment.

- Refer to the PaddleX document on Anaconda installation
2.3 Code installation

The github codes will be constantly updated with the development progress

```bash
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX
git checkout develop
python setup.py install
```

2.4 pycocotools installation problems

For the PaddleX dependency pycocotools package. If the pycocotools installation fails, install pycocotools by referring to the following method

2.4.1 Windows system

- During installation in Windows, the message Microsoft Visual C++ 14.0 is required may be displayed, resulting in installation error. [Click to download and install VC build] tools and then execute the following pip command (https://go.microsoft.com/fwlink/?LinkId=691126)

  Note: After the installation is complete, you must reopen a new terminal command window

```bash
pip install cython
pip install git+https://gitee.com/jiangjiajun/philferriere-cocoapi.git
    #subdirectory=PythonAPI
```

2.4.2 Linux/Mac system

- Directly use pip to install the following two dependencies in the Linux/Mac system

```bash
pip install cython
pip install pycocotools
```
3.1 Image Classification

Image classification and annotation is the most basic and simplest annotation task. Users only need to put the images belonging to the same category in the same folder, such as the directory structure shown below,

```
MyDataset/  # Image classification dataset root
|---dog/   # All pictures in the current folder belong to dog category
    |   |---d1.jpg
    |   |---d2.jpg
    |   |---...
    |   |---...
    |
    |---...
    |
    |---snake/  # All pictures in the current folder belong to snake category
    |   |---s1.jpg
    |   |---s2.jpg
    |   |---...
    |   |---...
```
3.1.1 Data partition

When training the model, we need to divide the training set, verification set and test set, so we need to divide the above data. We can divide the data set into 70% training set, 20% verification set and 10% test set by using paddlex command

```
paddlex --split_dataset --format ImageNet --dataset_dir MyDataset --val_value 0.2 --test_value 0.1
```

labels.txt, train_list.txt, val_list.txt, test_list.txt are generated from the divided data set, and then the training can be carried out directly.

Note: if you use PaddleX visual client for model training, the data set partition function is integrated in the client, and there is no need to use command partition by yourself

- Image Classification Task Training Example Code

3.2 Object detection

3.2.1 Introduction

Currently, PaddleX provides FasterRCNN and YOLOv3 detection structures and various backbone models to meet the requirements of developers for different scenarios and performances.

- **Box MMAP**: Model test precision on the COCO dataset
- **Inference speed**: Inference time for a single image (preprocessing and postprocessing excluded)
- “-“ indicates that the indexes are not updated temporarily

3.2.2 Start training

Save and run codes locally (The code downloading links are located in the table above) and **codes automatically download training data and start training**. If codes are saved as yolov3_mobilenetv1.py, execute the following command to start training:

```
python yolov3_mobilenetv1.py
```

3.2.3 Related document

- **Important** Adjust training parameters according to your machine environment and data, adjust training parameters? Understand the role of training parameters in PaddleX first. [——>Portal](../appendix/parameters.md)
• **[Useful]** There are no machine resources? Use a free AIStudio GPU resource: online training model. [——>>Portal](https://aistudio.baidu.com/aistudio/projectdetail/450925)

• **[Extension]** For more object detection models, refer to the *PaddleX model library* and the *API operation document*.

### 3.3 Instance Segmentation

Labelme annotation tool is recommended for instance segmentation data annotation. If you have not previously installed labelme, please refer to *Labelme installation and startup* for the installation of labelme.

**Note:** LabelMe is not friendly to Chinese support, so please do not appear Chinese characters in the following path and file name!

#### 3.3.1 Preparation

1. Store the collected images in the **JPEGImages** folder, for example, in D:\MyDataset\JPEGImages

2. Create a folder **Annotations** corresponding to the image folder to store annotated JSON files, such as D:\MyDataset\Annotations

3. Open LabelMe, click the “Open Dir” button, select the folder where the image to be labeled is opened, and the “File List” dialog box will display the absolute path corresponding to all images, and then you can start to traverse each image and label

#### 3.3.2 Target edge annotation

1. Open polygon annotation tool (right-click menu > Create Polygon) to circle the outline of the target by dot, and write the corresponding label in the pop-up dialog box (Click when the label already exists. Please note that the label should not be used in Chinese) , Specifically, as shown below, when the box is marked incorrectly, you can click “Edit Polygons” on the left, and then click the label box to modify it by dragging, or click “Delete Polygon” to delete it.
2. Click “Save” on the right to save the annotation results to the Annotations directory created in

3.3.3 Format conversion

LabelMe annotated data needs to be converted to MSCOCO format before it can be used for instance segmentation task training. Create the save directory `D:\dataset_seg`. After installing paddlex in Python environment, use the following command

```
paddlex --data_conversion --source labelme --to MSCOCO \  
    --pics D:\MyDataset\JPEGImages \  
    --annotations D:\MyDataset\Annotations \  
    --save_dir D:\dataset_coco
```

3.3.4 Dataset partition

After data conversion, in order to train, the data needs to be divided into training set, verification set and test set. After installing paddlex, the data can be divided into 70% training set, 20% verification set and 10% test set by using the following command

```
paddlex --split_dataset --format COCO --dataset_dir D:\MyDataset --val_value 0.2 --test_ \  
    --value 0.1
```

If you execute the above command line, `train.json`, `val.json`, `test.json` will be generated under `D:\MyDataset`, which will store training sample information, verification sample information and test sample information respectively.
Note: if you use PaddleX visual client for model training, the data set partition function is integrated in the client, and there is no need to use command partition by yourself

- Instance segmentation task training example code

### 3.4 Semantic Segmentation

LabelMe annotation tool is recommended for semantic data annotation. If you have not previously installed LabelMe, please refer to [LabelMe installation and startup](#) for labelme installation. The annotation of semantic segmentation is similar to instance segmentation, and the process is as follows

**Note:** LabelMe is not friendly to Chinese support, so please do not appear Chinese characters in the following path and file name!

#### 3.4.1 Preparation

1. Store the collected images in the `JPEGImages` folder, for example, in `D:\MyDataset\JPEGImages`

2. Create a folder `Annotations` corresponding to the image folder to store annotated JSON files, such as `D:\MyDataset\Annotations`

3. Open LabelMe, click the “Open Dir” button, select the folder where the image to be labeled is opened, and the “File List” dialog box will display the absolute path corresponding to all images, and then you can start to traverse each image and label

#### 3.4.2 Target edge annotation

1. Open polygon annotation tool (right-click menu > Create Polygon) to circle the outline of the target by dot, and write the corresponding label in the pop-up dialog box (Click when the label already exists. Please note that the label should not be used in Chinese). Specifically, as shown below, when the box is marked incorrectly, you can click “Edit Polygons” on the left, and then click the label box to modify it by dragging, or click “Delete Polygon” to delete it.
2. Click “Save” on the right to save the annotation results to the Annotations directory created in

### 3.4.3 Format conversion

LabelMe annotated data needs to be converted to SEG format before it can be used for semantic segmentation task training. Create the save directory `D:\dataset_seg`. After installing PaddleX in Python environment, use the following command

```bash
paddlex --data_conversion --source labelme --to SEG \n    --pics D:\MyDataset\JPEGImages \n    --annotations D:\MyDataset\Annotations \n    --save_dir D:\dataset_seg
```

### 3.4.4 Dataset partition

After data conversion, in order to train, the data needs to be divided into training set, verification set and test set. After installing paddlex, the data can be divided into 70% training set, 20% verification set and 10% test set by using the following command

```bash
paddlex --split_dataset --format SEG --dataset_dir D:\MyDataset --val_value 0.2 --test_ \n    --value 0.1
```

If you execute the above command line, `train_list.txt`, `val_list.txt`, and `test_list.txt` will be generated under `D:\MyDataset`, which will store training sample information, verification sample information and test sample information respectively.
Note: if you use PaddleX visual client for model training, the data set partition function is integrated in the client, and there is no need to use command partition by yourself

- Semantic segmentation task training example code

### 3.5 Labelme Installation and Startup

LabelMe is an open source annotation tool, which can be used to label object detection, instance segmentation and semantic segmentation.

#### 3.5.1 1. Install Anaconda

Anaconda is recommended to install Python dependencies. Experienced developers can skip this step. For installation of anaconda, please refer to Document.

After installing Anaconda and creating the environment, proceed to the next steps.

#### 3.5.2 2. Install Labelme

After entering Python environment, execute the following command:

```bash
conda activate my_paddlex
conda install pyqt
pip install labelme
```

#### 3.5.3 3. Start Labelme

Enter the Python environment where LabelMe is installed and execute the following command to start LabelMe:

```bash
conda activate my_paddlex
labelme
```
4.1 PaddleClas Image Classification

4.1.1 Data folder structure

In PaddleX, image classification supports ImageNet dataset format. The dataset directory `data_dir` contains multiple folders, and the images in each folder belong to the same category. The folder name is the category name (note that the path should not contain Chinese characters and spaces). The structure example is as follows:

```
-MyDataset/ # Image classification dataset root directory
  |--dog/ # All pictures in the current folder belong to the dog category.
  |   |--d1.jpg
  |   |--d2.jpg
  |   |--...
  |   |--...
  |   |
  |   |
  |   |--snake/ # All pictures in the current folder belongs to the snake category.
  |     |--s1.jpg
  |     |--s2.jpg
  |     |--...
```
4.1.2 Divide the training set and validation sets

To facilitate training, prepare `train_list.txt`, `val_list.txt` and `labels.txt` files in the `MyDataset` directory, indicating training set list, validation set list and category labels list, respectively. Click to download the image classification example dataset.

Note: You can also use PaddleX’s own tool to randomly divide the dataset. After the dataset is organized in the above format, run the following commands to quickly complete the random division of the dataset, where `val_value` indicates the ratio of the validation set, and `test_value` indicates the ratio of the test set (which can be 0), and the remaining ratio is used for the training set.****

```
paddlex --split_dataset --format ImageNet --dataset_dir MyDataset --val_value 0.2 --test_value 0.1
```

**labels.txt**

labels.txt: lists all the categories. The corresponding line number of the category represents the id of the category during the training of the model (the line number starts counting from 0), for example, labels.txt has the following content:

```
dog cat snake
```

There are three categories in the dataset, namely dog, cat and snake. In the model training, the category id of dog is 0, cat is 1, and so on.

**train_list.txt**

train_list.txt lists the collection of images used for training. The corresponding category ids are as follows (example):

```
dog/d1.jpg 0 dog/d2.jpg 0 cat/c1.jpg 1 . . . . . . . . . . . . . . snake/s1.jpg 2
```

The first column is the relative path to `MyDataset`, and the second column is the category id of the corresponding category for the image.

**val_list.txt**

val_list lists the image integration used for validation. The corresponding category id is in the same format as `train_list.txt`.  
4.1.3 PaddleX dataset loading

Sample codes are as follows:

```python
import paddlex as pdx from paddlex.cls import transforms
eval_transforms = transforms.Compose([transforms.ResizeByShort(short_size=256), transforms.CenterCrop(crop_size=224), transforms.Normalize()])
train_dataset = pdx.datasets.ImageNet(data_dir='/MyDataset', file_list='MyDataset/train_list.txt', label_list='MyDataset/labels.txt', transforms=train_transforms)
eval_dataset = pdx.datasets.ImageNet(data_dir='/MyDataset', file_list='MyDataset/eval_list.txt', label_list='MyDataset/labels.txt', transforms=eval_transforms)
```

4.2 PaddleDetection Object Detection

4.2.1 Dataset folder structure

In PaddleX, the object detection supports the PascalVOC dataset format. It is recommended that users organize the dataset in the following way: The original images are placed in the same directory, for example, JPEGImages. The marked xml files with the same name are placed in the same directory, for example, Annotations.

```
MyDataset/ # Object detection dataset root directory
|--JPEGImages/ # The directory where the original image files are located. |--1.jpg |--2.jpg |--. . . |--. . . |
|--Annotations/ # Mark the directory where the file is located. |--1.xml |--2.xml |--. . . |
```

4.2.2 Divide the training set and validation sets

To facilitate training, prepare train_list.txt, val_list.txt, and labels.txt files in the MyDataset directory, indicating training set list, validation set list, and category labels list, respectively. Click to download the object detection example dataset.

Note: You can also use PaddleX’s own tool to randomly divide the dataset. After the dataset is organized in the above format, run the following commands to quickly complete the random division of the dataset, where val_value indicates the ratio of the validation set, and test_value indicates the ratio of the test set (which can be 0), and the remaining ratio is used for the training set. ****
The first column is the relative path of the original image relative to MyDataset, and the second column is the relative path of the labeled file relative to MyDataset.

**val_list.txt**

val_list lists the image integration used for validation. Its corresponding annotation file has the same format as val_list.txt.

### 4.2.3 PaddleX dataset loading

Example codes are as follows:

```python
import paddlex as pdx from paddlex.det import transforms

train_transforms = transforms.Compose([transforms.RandomHorizontalFlip(), transforms.Normalize(), transforms.ResizeByShort(short_size=800, max_size=1333), transforms.Padding(coarsest_stride=32)])

train_dataset = pdx.datasets.VOCDetection( data_dir='./MyDataset', file_list='./MyDataset/train_list.txt', label_list='./MyDataset/labels.txt', transforms=train_transforms)

eval_transforms = transforms.Compose([transforms.Normalize(), transforms.ResizeByShort(short_size=800, max_size=1333), transforms.Padding(coarsest_stride=32)])

eval_dataset = pdx.datasets.VOCDetection( data_dir='./MyDataset', file_list='./MyDataset/val_list.txt', label_list='./MyDataset/labels.txt', transforms=eval_transforms)
```
4.3 Instance Segmentation MSCOCO

4.3.1 Dataset folder structure

In PaddleX, the instance segmentation supports the MSCOCO dataset format (MSCOCO format can also be used for the object detection). It is recommended to organize the datasets in the following way: original images are in the same directory (for example, JPEGImages), and the annotations files (for example, annotations.json) are in the same level directory as JPEGImages. The instance structure is as follows:

```
MyDataset/ # Instances segmentation dataset root directory |--JPEGImages/ # The directory where the original image files are located. |--1.jpg |--2.jpg |--... |--.../ |--.../|--annotations.json # Directory of annotation files
```

4.3.2 Divide the training set and validation sets

In PaddleX, to distinguish between the training set and the validation set, different json files are used to indicate the segmentation of data in the same level directory as MyDataset, for example, train.json and val.json. Click to download the instance segmentation and instance dataset

Note: You can also use PaddleX’ s own tool to randomly divide the dataset. After the dataset is organized in the above format, run the following commands to quickly complete the random division of the dataset, where val_value indicates the ratio of the validation set, and test_value indicates the ratio of the test set (which can be 0), and the remaining ratio is used for the training set. 

```
paddlex --split_dataset --format COCO --dataset_dir MyDataset --val_value 0.2 --test_value 0.1
```

MSCOCO data annotation files are in the JSON format. Users can use the annotation tools such as Labelme, Wizard Annotation Assistant or EasyData to annotate. For details, see the data annotation tool.

4.3.3 PaddleX loads dataset

Example codes are as follows:

```python
```
4.4 Semantic Segmentation Seg

4.4.1 Dataset folder structure

In PaddleX, the annotation files are png files. It is recommended to organize the datasets in the following way: the original images are in the same directory (for example, JPEGImages), and the annotations files with the same png are in the same directory, for example, Annotations. The instance is as follows:

MyDataset/ # Semantic segmentation dataset root directory  
|--JPEGImages/ # The directory where the original image files are located. 
   |--1.jpg 
   |--2.jpg 
   |   . . .
   |   . . .
   |--Annotations/ # Mark the directory where the file is located. 
      |--1.png 
      |--2.png 
      |   . . .
      |   . . .

Semantically segmented annotated images, for example, 1.png, Png: single-channel image. The pixel label category starts from 0 in the ascending order (0 means background), for example, 0, 1, 2, 3 means four categories. The annotation category can be up to 255 categories (where the pixel value 255 are not involved in training and evaluation).

4.4.2 Divide the training set and validation sets

To facilitate training, prepare train_list.txt, val_list.txt and labels.txt files in the MyDataset directory, indicating training set list, validation set list and category labels list, respectively. Click to download the semantic segmentation sample dataset

Note: You can also use PaddleX’s own tool to randomly divide the dataset. After the dataset is organized in the above format, run the following commands to quickly complete the random division of the dataset, where val_value indicates the ratio of the validation set, and test_value indicates the ratio of the test set (which can be 0), and the remaining ratio is used for the training set.

```
paddlex --split_dataset --format Seg --dataset_dir MyDataset --val_value 0.2 --test_value 0.1
```

**labels.txt**

labels.txt: lists all the categories. The corresponding line number of the category represents the id of the category during the training of the model (the line number starts counting from 0), for example, labels.txt has the following content:
Indicates that there are 3 segmentation categories in the detection dataset, namely, background, human, and car. In the model training, the category id corresponding to background is 0, human corresponds to 1, and so on. If you don’t know the specific category label, you can directly enter labels. txt by marking 0, 1, 2 line by line...

**train_list.txt **

train_list.txt lists the collection of images used for training. The corresponding annotation files are as follows (example):

```
JPEGImages/1.jpg Annotations/1.png JPEGImages/2.jpg Annotations/2.png . . . . .
```

The first column is the relative path of the original image relative to MyDataset, and the second column is the relative path of the labeled file relative to MyDataset```

**val_list.txt **

val_list lists the image integration used for validation. Its corresponding annotation file has the same format as val_list.txt.

### 4.4.3 PaddleX dataset loading

Example codes are as follows:

```python
import paddlex as pdx from paddlex.seg import transforms
eval_transforms = transforms.Compose([ transforms.ResizeByLong(long_size=512), transforms.Padding(target_size=512), transforms.Normalize() ])
train_dataset = pdx.datasets.SegDataset( data_dir='/MyDataset', file_list='/MyDataset/train_list.txt', label_list='/MyDataset/labels.txt', transforms=train_transforms)
eval_dataset = pdx.datasets.SegDataset( data_dir='/MyDataset', file_list='/MyDataset/val_list.txt', label_list='MyDataset/labels.txt', transforms=eval_transforms)
```

### 4.5 Plot Detection ChangeDet

#### 4.5.1 Dataset folder structure

In PaddleX, the annotation files are png files**.** It is recommended that users organize the dataset in the following way: The original landscape maps of the same plot at different periods are placed in the
same directory, such as JPEGImages. The marked png files with the same name are placed in the same directory, such as Annotations.

MyDataset/ # Semantic segmentation dataset root directory --JPEGImages/ # The directory where the original image files are located, containing images of the same object in both early stage and late stage |--1_1.jpg |--1_2.jpg |--2_1.jpg |--2_2.jpg |--. . . |--Annotations/ # Mark the directory where the file is located. |--1.png |--2.png |--. . . Original landscape images of the same plot at different times, such as 1_1.jpg and 1_2.jpg, which can be RGB color images, grayscale maps, or multi-channel images in tiff format. Semantically segmented annotated images, for example, 1.png, It is the single channel image. Pixel annotation categories should start from 0 in the ascending order (0 means background), for example, 0, 1, 2, 3 mean four categories. There are up to 255 categories (the pixel 255 is not involved in training and evaluation).

### 4.5.2 Divide the training set and validation sets

**To facilitate training, prepare train_list.txt , val_list.txt and labels.txt files in the MyDataset directory, indicating training set list, validation set list and category labels list, respectively.**

**labels.txt**

labels.txt: lists all the categories. The corresponding line number of the category represents the id of the category during the training of the model (the line number starts counting from 0), for example, labels.txt has the following content:

unchanged changed

Indicates that there are two segmentation categories in the detection dataset, namely, unchanged and changed. In the model training, the category id corresponding to unchanged is 0, changed is 1, and so on. If you don’t know the specific category label, you can directly enter labels.txt one by one, 0, 1, 2... 序列即可。

**train_list.txt**

train_list.txt lists the collection of images used for training. The corresponding annotation files are as follows (example):

JPEGImages/1_1.jpg JPEGImages/1_2.jpg Annotations/1.png JPEGImages/2_1.jpg JPEGImages/2_2.jpg Annotations/2.png . . . .

The first and second columns correspond to the relative paths of the original image relative to MyDataset for different periods of the same plot, and the third column is the relative path of the labeled file relative to MyDataset.
**val_list.txt**

val_list lists the image integration used for validation. Its corresponding annotation file has the same format as val_list.txt.

### 4.5.3 PaddleX dataset loading

sample code (computing)
PaddleX 集成了 PaddleClas、PaddleDetection 和 PaddleSeg 三大 CV 工具套件中的工业领域应用成熟的模型，并提供了统一易用的 API 使用接口，帮助用户快速完成视觉领域的图像分类、目标检测、实例分割和语义分割模型的训练。

5.1 Image classification

5.1.1 Introduction

PaddleX 提供了超过 20 个图像分类模型以满足开发者的不同场景需求。

- **Top1 precision**: 模型在 ImageNet 数据集上的测试精度
- **Inference speed**: 单张图像的推断时间（预处理和后处理除外）
- “o” 表示未更新的指数

5.1.2 Start training

保存并运行本地代码（代码下载链接位于表格上方）并自动下载训练数据并开始训练。如果代码保存为 `mobilenetv3_small_ssld.py`，执行以下命令开始训练：
5.1.3 Related document

- [Important] Adjust training parameters according to your machine environment and data, adjust training parameters? Understand the role of training parameters in PaddleX first. [——>>Portal] (../appendix/parameters.md)

- [Useful] There are no machine resources? Use a free AIStudio GPU resource: online training model. [——>>Portal] (https://aistudio.baidu.com/aistudio/projectdetail/450925)

- [Extension] For more image classification models, refer to the PaddleX model library and the API operation document.

5.2 Object detection

5.2.1 Introduction

Currently, PaddleX provides FasterRCNN and YOLOv3 detection structures and various backbone models to meet the requirements of developers for different scenarios and performances.

- Box MMAP: Model test precision on the COCO dataset
- Inference speed: Inference time for a single image (preprocessing and postprocessing excluded)
- “- “indicates that the indexes are not updated temporarily

5.2.2 Start training

Save and run codes locally (The code downloading links are located in the table above) and codes automatically download training data and start training. If codes are saved as yolov3_mobilenetv1.py, execute the following command to start training:

```
python yolov3_mobilenetv1.py
```

5.2.3 Related document

- [Important] Adjust training parameters according to your machine environment and data, adjust training parameters? Understand the role of training parameters in PaddleX first. [——>>Portal] (../appendix/parameters.md)
5.3 Instance segmentation

5.3.1 Introduction

Currently, PaddleX provides a MaskRCNN with instance segmentation model structure and various backbone models to meet the requirements of developers for different scenarios and performance.

- **Box MMAP/Seg MMAP**: Model test precision on the COCO dataset
- **Inference speed**: Inference time for a single image (preprocessing and postprocessing excluded)
- "—" indicates that the indexes are not updated temporarily

5.3.2 Start training

Save and run codes locally (The code downloading links are located in the table above) and codes automatically download training data and start training. If codes are saved as `mask_rcnn_r50_fpn.py`, execute the following command to start training:

```
python mask_rcnn_r50_fpn.py
```

5.3.3 Related document

- [Important] Adjust training parameters according to your machine environment and data, adjust training parameters? Understand the role of training parameters in PaddleX first. ——>>>Portal
- [Useful] There are no machine resources? Use a free AIIstudio GPU resource: online training model. ——>>>Portal
- [Extension] For more instance segmentation models, refer to the [PaddleX model library] (../appendix/model_zoo.md) and the API operation document.
5.4 Semantic segmentation

5.4.1 Introduction

Currently, PaddleX provides DeepLabv3p, UNet, HRNet and FastSCNN with semantic segmentation structures and various backbone models to meet the requirements of developers for different scenarios and performance.

- **mIoU**: Model test precision on the CityScape dataset
- **Inference speed**: Inference time for a single image (preprocessing and postprocessing excluded)
- “o” “indicates that the indexes are not updated temporarily

5.4.2 Start training

Save and run codes locally (The code downloading links are located in the table above) and codes automatically download training data. Start training. If codes are saved as `deeplabv3p_mobilenetv2_x0.25.py`, execute the following command to start training:

```
python deeplabv3p_mobilenetv2_x0.25.py
```

5.4.3 Related document

- **Important**: Adjust training parameters according to your machine environment and data, adjust training parameters? Understand the role of training parameters in PaddleX first. ——>>[Portal] (../appendix/parameters.md)
- **Useful**: There are no machine resources? Use a free AIStudio GPU resource: online training model. ——>>[Portal] (https://aistudio.baidu.com/aistudio/projectdetail/450925)
- **Extension**: For more semantic segmentation models, refer to the PaddleX model library and the API operation document.

5.5 VisualDL visual training index

In the process of training a model using PaddleX, the training and evaluation indexes are directly output to the standard output stream. At the same time, the indexes in the training process can be visualized through VisualDL by setting the `use_vdl` parameter to `True` when the `train` function is called, as shown in the following codes.
model = paddlex.cls. ResNet50(num_classes=1000)
model.train(num_epochs=120, train_dataset=train_dataset,
    train_batch_size=32, eval_dataset=eval_dataset,
    log_interval_steps=10, save_interval_epochs=10,
    save_dir='./output', use_vdl=True)

The vdl_log directory is generated under save_dir during model training. VisualDL is started by executing the following command at the command line terminal.

```
visualdl --logdir=output/vdl_log --port=8008
```

Open http://0.0.0.0:8008 on the browser and you can directly view the indexes which dynamically change with training iteration. (0.0.0.0 indicates starting the IP address of the server where VisualDL is located. The local machine uses 0.0.0.0.)

The example graph of performing visualization using VisualDL during the training of classification models is shown below.

Change trend of Loss and the corresponding Top1 accuracy rate in each step during training:

Change trend of learning rate lr and the corresponding Top5 accuracy rate in each step during training:

Top1 accuracy rate and Top5 accuracy rate of the model on the val-
idation dataset every time the model is saved during training:

![Graph 1](image1.png)

![Graph 2](image2.png)
CHAPTER 6

Loading a model for inference

PaddleX may use `paddlex.load_model` API to load models (including models saved during training, exported deployment models, quantitative models and pruned models) for inference. PaddleX also has a series of built-in visualization tool functions to help you check model effects easily.

**Note:** A model loaded by the `paddlex.load_model` API is used for model inference only. If you want to continue the training on the basis of this model, you can use this model as a pre-training model. The specific procedure is to specify the `pretrain_weights` parameter in the train function as a pre-training model path in the training codes.

### 6.1 Image classification

[Click to download] the model in the following example codes (https://bj.bcebos.com/paddlex/models/mobilenetv3_small_ssld_imagenet.tar.gz)

```python
import paddlex as pdx

test_jpg = 'mobilenetv3_small_ssld_imagenet/test.jpg'
model = pdx.load_model('mobilenetv3_small_ssld_imagenet')
result = model.predict(test_jpg)
print("Predict Result: ", result)
```

The results are outputted as follows:

```
Predict Result: [{'category_id': 549, 'category': 'envelope', 'score': 0.29062933}]
```

The test picture is as follows:
- Classification model predict API [Description document] (../apis/models/classification.html#predict)

6.2 Object detection

[Click to download] the model in the following example codes (https://bj.bcebos.com/paddlex/models/yolov3_mobilenetv1_coco.tar.gz)

```python
import paddlex as pdx
test_jpg = 'yolov3_mobilenetv1_coco/test.jpg'
model = pdx.load_model('yolov3_mobilenetv1_coco')

# The predict API does not filter low-confidence recognition results. You shall perform filtering by score value according to requirements
result = model.predict(test_jpg)

# The visualization results are stored in ./visualized_test.jpg, as shown in the following figure
pdx.det.visualize(test_jpg, result, threshold=0.3, save_dir='/')
```

- YOLOv3 model predict API [Description document] (../apis/models/detection.html#predict)
- Visualization paddx.det.visualize API [Description document] (../apis/visualize.html#paddlex-det-visualize)

Note: For results obtained by calling the predict API for object detection and instance segmentation models, you must filter low-confidence results. In the paddlex. det.visualize API, we provide a threshold for filtering. Any result of which the confidence is smaller than this value will be filtered and will not be visualized.
6.3 Instance segmentation

[Click to download] the model in the following example codes](https://bj.bcebos.com/paddlex/models/mask_r50_fpn_coco.tar.gz)

```python
import paddlex as pdx

test_jpg = 'mask_r50_fpn_coco/test.jpg'
model = pdx.load_model('mask_r50_fpn_coco')

# The predict API does not filter low-confidence recognition results. You shall perform filtering by score value according to requirements
result = model.predict(test_jpg)

# The visualization results are stored in ./visualized_test.jpg, as shown in the following figure
pdx.det.visualize(test_jpg, result, threshold=0.5, save_dir='./')
```

- MaskRCNN model predict API [Description document] (../apis/models/instance_segmentation.html#predict)
- Visualization pdx.det.visualize API [Description document] (../apis/visualize.html#paddlex-det-visualize)
Note: For results obtained by calling the `predict` API for object detection and instance segmentation models, you must filter low-confidence results. In the `paddlex.det.visualize` API, we provide a threshold for filtering. Any result of which the confidence is smaller than this value will be filtered and will not be visualized.

6.4 Semantic segmentation

[Click to download] the model in the following example codes (https://bj.bcebos.com/paddlex/models/deeplabv3p_mobilenetv2_voc.tar.gz)

```python
import paddlex as pdx
test_jpg = './deeplabv3p_mobilenetv2_voc/test.jpg'
model = pdx.load_model('./deeplabv3p_mobilenetv2_voc')
result = model.predict(test_jpg)
# The visualization results are stored in './visualized_test.jpg', as shown in the following figure on the right (The figure on the left is the original figure)
pdx.seg.visualize(test_jpg, result, weight=0.0, save_dir='./')
```

In the above-mentioned example codes, the inference results of semantic segmentation can be visualized by
calling `paddlex.seg.visualize` and the results of the visualization are saved under `save_dir`, as shown in the following figure. The `weight` parameter is used to adjust a weight during the mixed presentation of inference results and original figure results. Only the visualization of inference results mask is shown at 0.0. Only the visualization of the original figure is shown at 1.0.

6.5 Downloading of trained models on public datasets

PaddleX provides some trained models on public datasets. After directly downloading them, you can load and use them by referring to this document.

The `load_model` API of PaddleX can meet your general requirements for model investigation. For inference deployments with higher performance, refer to the following documents

- `Server Python deployment`
- `Server C++ deployment`
CHAPTER 7

Training parameter adjustment

In all the training interfaces of PaddleX, the built-in parameters are based on the better parameters under the corresponding batch_size of the single GPU card. Users train the model on their own data. When a parameter needs to be adjusted, refer to the following modes (in case of lack of rich parameter adjustment experiences):

7.1 1. Adjustment of num_epochs

num_epochs: The total number of rounds of training iterations of the model (the model goes through all the samples of the training set once is an epoch). The user can set to a larger value to determine whether the model converges or not based on the performance of the model’s iterations on the validation set, and then terminate the training early. In addition, you can also use the early_stop strategy in the train interface. The model automatically determines whether the model converges and aborts automatically.

7.2 2. batch_size and learning_rate

- Batch Size refers to the number of samples used to compute the model forward once (that is, one step) during training.
- If you are training with multiple cards, the batch_size is divided equally among the cards (so, you need to divide the batch size by the number of cards).
- Batch Size: It is highly related to the video/memory. The higher the batch_size, the more video/memory is consumed.
• PaddleX configures the default batch size (by default, for single GPU card) in each train interface. If the system prompts insufficient GPU memory in the training, you should set BatchSize to a smaller value accordingly.

• If the user adjusts the batch size, the user should also adjust other parameters, especially the default learning_rate value in the train interface. For example, in YOLOv3 model, the default training_batch_size is 8 and learning_rate is 0.000125. When you train the model on a card No.2, you can set training_batch_size to 16, and then learning_rate to 0.000125 \times 2 = 0.00025''''

7.3 3. warmup_steps and warmup_start_lr

In the model training, the pre-training model is usually used. For example, use backbone’s pre-training weights on the ImageNet dataset during training in the detection model. However, due to the large difference between own data and the ImageNet dataset in the training, the training may have problems at first for the large step. In this case, you can set learning rate to a smaller value, and then grow slowly to a proper learning rate. Warmup_steps and warmup_start_lr are used for this purpose. When the model starts training, the learning rate starts from warmup_start_lr and grows linearly to the set learning rate after iterations of warmup_steps and the batch data.'''

For example, in the train interface of YOLOv3, the default train_batch_size is 8, learning_rate is 0.000125, warmup_steps is 1000, and warmup_start_lr is 0.0. With this parameter configuration, after the model starts training, the learning rate grows linearly from 0.0 to a set value 0.000125 after the first 1000 steps (each step uses one batch of data, that is, 8 samples), the learning rate grows linearly from 0.0 to a set 0.000125.

7.4 4. lr_decay_epochs and lr_decay_gamma

lr_decay_epochs is used to allow the learning rate to decay progressively later in the model training; it is typically a list such as 6, 8, and 10, indicating that the learning rate decays once at the 6th epoch, again at the 8th epoch, and again at the 10th epoch.[Each learning rate decays as the previous learning rate \times lr_decay_gamma.]

For example, the train interface of YOLOv3 has a default num_epochs of 270, learning_rate of 0.000125, [lr_decay_epochs] of 213, 240, and lr_decay_gamma of 0.1. In this parameter configuration, after the model starts training, for the first 213 epochs, the learning rate used for training is 0.000125, for 213-240 epochs, the learning rate in training is 0.000125x0.1=0.0000125, and for more than 240 epochs, the learning rate is 0.000125x0.1x0.1=0.00000125.
7.5 5. Constraints on parameter setting

Based on these several parameters, it is understood that the change of the learning rate includes WarmUp and Decay.

- Warmup: with training iterations, the learning rate grows linearly from a low value to a set value. The unit is step.
- Decay phase: with training iterations, the learning rate gradually decays, that is, each decay is 0.1 of the previous one. The unit is epoch.
- The relationship between step and epoch: 1 epoch is composed of several steps. For example, there are 800 images in the training sample, and the train_batch_size is 8, and each epoch should use these 800 images to train the model once, and each epoch contains 800//8=100 steps in total.

In PaddleX, the constraint warmup must end before Decay, so each parameter needs to meet the following conditions.

\[
\text{warmup\_steps} \leq \text{lr\_decay\_epochs}[0] \times \text{num\_steps\_each\_epoch}
\]

where \(\text{num\_steps\_each\_epoch}\) is calculated as follows,

\[
\text{num\_steps\_each\_epoch} = \frac{\text{num\_samples\_in\_train\_dataset}}\text{train\_batch\_size}
\]

Therefore, if you are prompted “warmup\_steps should be less than” in the start of the training, it means that you need to adjust your parameters according to the above formula: lr\_decay\_epoch or warmup\_steps.

7.6 6. How to use multi-GPU cards for training

Configure environment variables in front of import paddlex. Codes are as follows:

```python
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0'  # Training with GPU card 0  # Note
# that either paddle or paddlex needs to be imported after setting environment variables.
# import paddle as pd
```

```python
import os
os.environ['CUDA_VISIBLE_DEVICES'] = ''  # Use the CPU for training without using the GPU import paddlex as pdx
```

```python
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '0,1,3'  # Simultaneous training with GPU cards 0, 1, and 3 import paddlex as pdx
```
7.7 Related Model Interface

- Image classification model train interface
- Object detection FasterRCNN train interface
- Object detection YOLOv3 train interface
- Instance segmentation MaskRCNN train interface
- Semantic segmentation train interface
8.1 Training process preservation

PaddleX in the training process, according to the `train` function interface, the `save_interval_epoch` parameter setting, save the model every multiple epochs. The model directory contains `model.pdparams`, `model.yml` files.

In the training process, you can save the model as a `pretrain_weights` for retraining later, or use `paddlex.load_model` interface to load the model for prediction and evaluation.

8.2 Deployment model export

In the training process, if you need to export the model for deployment (see the section on multi-platform deployment in the PaddleX document), the exported model format is the same, the model directory contains `__model__.`, `__params__.` and `model.yml` files.

The model can be used in the `paddle.deploy.Predictor` interface if you need to use prediction in Python. You can also use the `paddlex.load_model` interface.

Model export reference: 部署模型导出

Summary: If the model file contains `model.pdparams`, it means the model is saved in the training process. When deploying, you need to export it. The model directory contains `__model__.`, `__params__.` and `model.yml` files.
8.3 Model deployment file description

- __model__: 保存了模型的网络结构信息
- __params__: 保存了模型网络中的参数权重
- model.yml: 在 PaddleX 中，将模型的预处理、后处理，以及类别相关信息均存储在此文件中

8.4 The model is exported to ONNX Model

PaddleX 作为开放开源的套件，其中的大部分模型均支持导出为 ONNX 协议，满足开发者多样性的需求。

需要注意的是 ONNX 存在多个 OpSet 版本，下表为 PaddleX 各模型支持导出的 ONNX 协议版本。

8.4.1 How to export

- 1. 参考文档部署模型导出，将训练保存的模型导出为部署模型
- 1. 安装 paddle2onnx pip install paddle2onnx，转换命令如下，通过-opset_version 指定版本
   (9/10/11)，转换使用方法参考 Paddle2ONNX 说明
- 附: Paddle2ONNX 参阅 https://github.com/PaddlePaddle/paddle2onnx
模型裁剪可以更好地满足在端侧、移动端上部署场景下的性能需求，可以有效降低模型的体积，以及计算量，加速预测性能。PaddleX 集成了 PaddleSlim 的基于敏感度的通道裁剪算法，用户可以在 PaddleX 的训练代码里轻松使用起来。

在本文档中展示了分类模型的裁剪过程，文档中代码以及更多其它模型的的裁剪代码可在 Github 中的 tutorials/slim/prune 目录获取。

### 9.1 Usage methods

模型裁剪相对比我们普通训练一个模型，步骤会多出两步

- 1. 采用正常的方式训练一个模型
- 2. 对模型的参数进行敏感度分析
- 3. 根据第 2 步得到的敏感度信息，对模型进行裁剪，并以第 1 步训练好的模型作为预训练权重，继续进行训练

具体我们以图像分类模型 MobileNetV2 为例，本示例中所有代码均可在 Github 的 [tutorials/slim/prune/image_classification] 中获得。

### 9.1.1 First step Training model normally

此步骤中采用正常的代码进行模型训练，在获取本示例代码后，直接执行如下命令即可
在训练完成后，我们以 `output/mobilenetv2/best_model` 保存的模型，继续接下来的步骤

### 9.1.2 Step two Parameter sensitivity analysis

此步骤中，我们需要加载第一步训练保存的模型，并通过不断地遍历参数，分析各参数裁剪后在验证数据集上的精度损失，以此判断各参数的敏感度。敏感度分析的代码很简单，用户可直接查看 `params_analysis.py`。在命令行终端执行如下命令开始参数分析。

```python
python params_analysis.py
```

在此步骤中，我们会得到保存的 `mobilenetv2.sensi.data` 文件，这个文件保存了模型中每个参数的敏感度，在后续的裁剪训练中，会根据此文件中保存的信息，对各个参数进行裁剪。同时，我们也可以对这个文件进行可视化分析，判断 `eval_metric_loss` 的大小设置与模型被裁剪比例的关系。（`eval_metric_loss` 的说明见第三步）

模型裁剪比例可视化分析代码见 `slim_visualize.py`，执行如下命令即可

```python
python slim_visualize.py
```

可视化结果如下，该图表明，当我们将 `eval_metric_loss` 设为 0.05 时，模型将被裁剪掉 65%；将 `eval_metric_loss` 设为 0.10，模型将被裁剪掉 68.0%。因此在实际使用时，我们可以根据自己的需求，去设置 `eval_metric_loss` 控制裁剪比例。

### 9.1.3 Step three Model cutting training

在前两步，我们得到了正常训练保存的模型 `output/mobilenetv2/best_model` 和基于该保存模型得到的参数敏感度信息文件 `mobilenetv2.sensi.data`，接下来则是进行模型裁剪训练。模型裁剪训练的代码在第一步基本一致，唯一区别在最后的 `train` 函数中，我们修改了 `pretrain_weights`, `save_dir`, `sensitivities_file` 和 `eval_metric_loss` 四个参数，如下所示

```python
model.train(
    num_epoch=10,
    train_dataset=train_dataset,
    train_batch_size=32,
    eval_dataset=eval_dataset,
    lr_decay_epochs=[4, 6, 8],
    learning_rate=0.025,
    pretrain_weights='output/mobilenetv2/best_model',
    save_dir='output/mobilenetv2_prune',
    sensitivities_file='./mobilenetv2.sensi.data',
)
```

(下页继续)
具体代码见 tutorials/slim/prune/image_classification/mobilenetv2_prune_train.py，执行如下命令即可

```python
python mobilenetv2_prune_train.py
```

其中修改的 4 个参数函数如下

- `pretrain_weights`: 预训练权重，在裁剪训练中，将其指定为第一步正常训练得到的模型路径
- `save_dir`: 裁剪训练过程中，模型保存的新路径
- `sensitivities_file`: 第二步中分析得到的各参数敏感度信息文件
- `eval_metric_loss`: 可用于控制模型最终被裁剪的比例，见第二步中的可视化说明

### 9.2 Clipping Effect

在本示例的数据集上，经过裁剪训练后，模型的效果对比如下，其中预测速度不包括图像的预处理和结果的后处理。从表中可以看到，对于本示例中的简单数据集，模型裁剪掉 68% 后，模型准确度没有降低，在 CPU 的单张图片预测用时减少了 37%
Model Quantification

Model quantification is the process of transforming a continuous floating-point model into a discrete low-bit model. It can increase the computation speed at runtime and reduce the model size on mobile or edge devices.

10.1 Usage method

PaddleX provides a quantification API that can be used directly. This code can be found in the GitHub repository tutorials/slim/quant/image_classification.

```python
import paddlex as pdx

model = pdx.load_model('mobilenetv2_vegetables')

# 加载数据集用于量化
dataset = pdx.datasets.ImageNet(
    data_dir='vegetables_cls',
    file_list='vegetables_cls/train_list.txt',
    label_list='vegetables_cls/labels.txt',
    transforms=model.test_transforms)
```

(下页继续)
```python
pdx.slim.export_quant_model(model, dataset,
    batch_size=4,
    batch_num=5,
    save_dir='./quant_mobilenet',
    cache_dir='./tmp')
```

在获取此示例代码后，执行如下命令即可完成量化和 PaddleLite 的模型导出

```bash
# 将 mobilenetv2 模型量化保存
python mobilenetv2_quant.py
# 将量化后的模型导出为 PaddleLite 部署格式
python paddlelite_export.py
```

## 10.2 Quantitative effect

在本示例中，我们可以看到模型量化后的服务端部署模型格式 `server_mobilenet` 和 `quant_mobilenet` 两个目录中，模型参数大小并无变化。但在使用 PaddleLite 导出后，`mobilenetv2.nb` 和 `mobilenetv2_quant.nb` 大小分别为 8.8M, 2.7M，压缩至原来的 31%。
When deploying models on the server, you need to export the model saved during training to a model in the inference format. The exported inference format model includes three files: __model__, __params__ and model.yml, which represent the network structure, model weights, and model configuration file (including data preprocessing parameters) respectively.

**Check your model folder.** If it contains model.pdparams, model.pdmodel and model.yml files, you need to export the model in the following process.

After installing PaddleX, run the following command to export the model in a command line terminal. You can directly download the DUDU sorting model to test the process: xiaoduxiong_epoch_12.tar.gz.

```
paddlex --export_inference --model_dir= ./xiaoduxiong_epoch_12 --save_dir= ./inference_ _model
```

When using TensorRT for prediction, you need to fix the input size of the model. You can prepare the input size \([w,h]\) by using `--fixed_input_shape`.

**Note:**
- Keep the fixed input size for the classification model to be the same as the input size for training;
- In the setting of \([w,h]\), w and h are separated by a comma, and no other characters such as spaces are allowed.

```
paddlex --export_inference --model_dir= ./xiaoduxiong_epoch_12 --save_dir= ./inference_ _model --fixed_input_shape=[640,960]
```
12.1 Introduction

With PaddleHub-Serving, PaddleX’s Inference Model can be rapidly deployed to provide the online prediction ability.

For more information on PaddleHub-Serving, refer to [PaddleHub-Serving]. (https://github.com/PaddlePaddle/PaddleHub/blob/develop/docs/tutorial/serving.md)

Note: To deploy in this way, you need to make sure that the version of PaddleHub in the Python environment is later than 1.8.0. You can run `pip show paddlehub` to confirm the version information.

Next, follow the steps to convert an image classification model MobileNetV3_small_ssl into a pre-training model for PaddleHub, and use PaddleHub-Serving to implement one-key deployment.

12.2 Model Deployment

12.2.1 Preparation for deployment model

The format of the deployment model is three files __model__, __params__ and model.yml contained in the directory. For the format of the files, refer to the [deployment model export file]. (.export_model.md)
12.2.2 2 Model conversion

First, convert PaddleX’s Inference Model to PaddleHub’s pre-training model. Run `hub convert` to implement the one-key conversion. The command is described as follows:

```bash
$ hub convert --model_dir XXXX \
   --module_name XXXX \
   --module_version XXXX \
   --output_dir XXXX
```

Parameters:

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--model_dir/-m</td>
<td>PaddleX Inference Model directory</td>
<td>Generate the name of the pre-training model. The default value is 1.0.0</td>
</tr>
<tr>
<td>--module_name/-n</td>
<td></td>
<td>Generate the name of the pre-training model.</td>
</tr>
<tr>
<td>--module_version/-v</td>
<td></td>
<td>Version of generating the pre-training model. The default value is 1.0.0</td>
</tr>
<tr>
<td>--output_dir/-o</td>
<td></td>
<td>Directory of storing the generated pre-trained model. The default name is <code>{module_name}_{timestamp}</code></td>
</tr>
</tbody>
</table>

Therefore, you only need to run a command to complete the conversion of the pre-training model.

```bash
hub convert --model_dir mobilenetv3_small_ssld_imagenet_hub --module_name mobilenetv3_small_ssld_imagenet_hub
```

After the conversion is complete the prompted information is as follows:

```bash
$ The converted module is stored in `MobileNetV3_small_ssld_hub_1596077881.868501`.
```

After the prompt, a pre-training model of PaddleHub in the output directory is obtained.

12.2.3 3 Model installation

In the model conversion step, a pre-trained model compression package in the .tar.gz format. Before deploying, you need to install it locally by running the command `hub install`. The description is as follows:

```bash
$ hub install ${MODULE}
```

${MODULE} is the path of the pre-training model file to be installed.

Run `hub install`.

```bash
hub install MobileNetV3_small_ssld_hub_1596077881.868501/mobilenetv3_small_ssld_imagenet_hub.tar.gz
```

After a successful installation, the following message is displayed:
$$\text{Successfully installed mobilenetv3_small_ssd_imagenet_hub}$$

### 12.2.4 Model Deployment

You can run `hub serving` to deploy the model through one-key. The description is as follows:

```bash
$ hub serving start --modules/-m [Module1==Version1, Module2==Version2, ...] \n  --port/-p XXXX \n  --config/-c XXXX
```

**Parameters:**

Therefore, only one line of code is needed to deploy the model.

```bash
$ hub serving start -m mobilenetv3_small_ssd_imagenet_hub
```

After the model is loaded, this pre-training model is now deployed on the machine.

You can perform more configurations by using the configuration file. The format of the configuration file is as follows:

```json
{
  "modules_info": {
    "mobilenetv3_small_ssd_imagenet_hub": {
      "init_args": {
        "version": "1.0.0"
      },
      "predict_args": {
        "batch_size": 1,
        "use_gpu": false
      }
    }
  },
  "port": 8866
}
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Use</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>modules_info</td>
<td>PaddleHub Serving pre-installed models, listed as a dictionary list, key is the model name. Where: <code>init_args</code> is the parameter to be entered when the model is loaded, equivalent to <code>paddlehub.Module(**init_args)</code></td>
<td>predict_args is the parameter entered when the model is predicted. For example, in <code>mobilenetv3_small_ssd_imagenet_hub</code>, it is equivalent to <code>mobilenetv3_small_ssd_imagenet_hub.batch_predict(**predict_args)</code></td>
</tr>
</tbody>
</table>
12.2.5 5 Test

While the model is installed in the second step, a client request example is generated and stored in the model installation directory. By default, it is `${HUB_HOME}/.paddlehub/modules`. In this example, the client example `serving_client_demo.py` can be found in `~/.paddlehub/modules/mobilenetv3_small_ssld_imagenet_hub`. The codes are as follows:

```python
# coding: utf8
import requests
import json
import cv2
import base64

def cv2_to_base64(image):
    data = cv2.imencode('.jpg', image)[1]
    return base64.b64encode(data.tobytes()).decode('utf8')

if __name__ == '__main__':
    # Get the base64 encoding format of the image
    img1 = cv2_to_base64(cv2.imread("IMAGE_PATH1"))
    img2 = cv2_to_base64(cv2.imread("IMAGE_PATH2"))
    data = {'images':[img1, img2]}
    # Specify content-type
    headers = {'Content-type':"application/json"}
    # Send an HTTP request
    url = "http://127.0.0.1:8866/predict/mobilenetv3_small_ssld_imagenet_hub"
    r = requests.post(url=url, headers=headers, data=json.dumps(data))

    # Print the prediction result
    print(r.json()['results'])
```

The following test images are used.
After changing IMAGE_PATH1 in the code to the path of the image where you want to make the prediction, run the following command line:

```
python ~/.paddlehub/module/MobileNetV3_small_ssld_hub/serving_client_demo.py
```

The following prediction results can be received:

```
[[{'category': 'envelope', 'category_id': 549, 'score': 0.2141510397195816}]]
```

The one-key deployment of the PaddleX model is completed.
13.1 Python deployment

PaddleX has been integrated with a high performance prediction interface based on Python. After installing PaddleX, you can refer to the following code example to make predictions.

13.1.1 Export the prediction model

You can refer to Model Export to export the model in an inference format.

13.1.2 Inference deployment

For the prediction interface, refer to paddlex.deploy.

Click to download the test image xiaoduxiong_test_image.tar.gz

- Single image prediction

```python
import paddlex as pdx
predictor = pdx.deploy.Predictor('./inference_model')
result = predictor.predict(image='xiaoduxiong_test_image/JPEGImages/WeChatIMG110.jpeg')
```

- Batch image prediction
import paddlex as pdx
predictor = pdx.deploy.Predictor('./inference_model')
image_list = ['xiaoduxiong_test_image/JPEGImages/WeChatIMG110.jpeg',
              'xiaoduxiong_test_image/JPEGImages/WeChatIMG111.jpeg']
result = predictor.batch_predict(image_list=image_list)

- Video Stream Prediction

```
import cv2
import paddlex as pdx
predictor = pdx.deploy.Predictor('./inference_model')
cap = cv2.VideoCapture(0)
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        result = predictor.predict(frame)
        vis_img = pdx.det.visualize(frame, result, threshold=0.6, save_dir=None)
        cv2.imshow('Xiaoduxiong', vis_img)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        break
cap.release()
```

Description about the prediction speed: The prediction speed of the first few images after loading the model is slow, because the initialization of the video card and memory is involved in the start-up. Generally, the prediction speed after predicting 20-30 images is stable.

13.1.3 Prediction performance comparison

Test environment

- CUDA 9.0
- CUDNN 7.5
- PaddlePaddle 1.71
- GPU: Tesla P40
- AnalysisPredictor: a Python high-performance prediction method is used.
- Executor: A common Python prediction method of the PaddlePaddle is used.
- Batch Size: It is 1, time consumption unit is ms/image. Only model runtime is calculated, excluding data pre-processing and post-processing.

Performance comparison

## 13.2 C++ Deployment

### 13.2.1 Windows platform deployments

**Description**

On the Windows platform, use the Visual Studio 2019 Community for testing. Since 2017, Microsoft Visual Studio has supported the direct management of CMake cross-platform compilation projects. But it did not provide stable and complete support until 2019. If you want to use CMake to manage project compilation and build, Visual Studio 2019 is recommended.

**Pre-conditions**

- Visual Studio 2019
- CUDA 9.0 / CUDA 10.0, CUDNN 7+ (required only if using GPU version prediction library)
- CMake 3.0+

Make sure that the above basic software is installed on your system. Here the VS2019 Community Edition is used.

All the examples below are shown in the working directory: D:\projects.

**Step1: Download the PaddleX prediction code.**

```bash
d: mkdir projects
cd projects
git clone https://github.com/PaddlePaddle/PaddleX.git
```

*Note: The C++ prediction code is in PaddleX\deploy\cppdirectory, which does not depend on any other directory in PaddleX.*

**Step2: Download PaddlePaddle C++ Prediction Library: paddle_inference.**

PaddlePaddle C++ prediction Library provides the compiled prediction libraries for the use of GPU or not, whether or not support TensorRT, and different CUDA versions. At present, PaddleX depends on Paddle
1.8.4. The download link of Paddle prediction library based on Paddle 1.8.4 is as follows:

Select the download as required. If the above version does not meet your needs, go to the [download list of C++ prediction library](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/advanced_guide/inference_deployment/inference/windows_cpp_inference.html) and choose a suitable version.

After unzipping the prediction library, the directory is located in (for example, D:\projects\fluid_inference\) contains the following:

```
paddle\  # paddle core library and header files
  |  
third_party\  # Third-party dependency library and header files
  |  
version.txt  # Version and compilation information
```

**Step3: Install and configure the OpenCV**

1. Download version 3.4.6 for Windows from the OpenCV website. The [website for downloading](https://bj.bcebos.com/paddleseg/deploy/opencv-3.4.6-vc14_vc15.exe)

2. Run the downloaded executable file and decompress the OpenCV to the specified directory, for example, D:\projects\opencv

3. Configure the environment variables:
   - My Computer->Properties->Advanced System Settings->Environmental Variables
   - Find Path in the system variables (if not, create one yourself and double-click to edit it.
   - Add a new file. Fill in the opencv path and save it. For example, D:\projects\opencv\build\x64\vc14\bin
Step 4: Compile CMake directly with Visual Studio 2019.

1. Open Visual Studio 2019 Community and click Continue. No code is required.

2. Choose: File -> Open -> CMake
Choose the path where the C++ prediction code is located (for example, D:\projects\PaddleX\deploy\cpp), and open CMakeList.txt:
3. Choose Project->CMake Settings

4. Click Browse to set the compiling options to specify the paths to the CUDA, OpenCV, and Paddle prediction libraries, respectively.
Meaning of the dependency library path (with * means it is only specified when using the GPU version prediction library. For the CUDA library version, it should be aligned with the Paddle prediction library as much as possible. For example, if the Paddle prediction library is compiled with versions 9.0 and 10.0, the PaddleX prediction codes are compiled ** without using the CUDA libraries of V9.2, 10.1):

Note:

1. If you are using the CPU the prediction library, de-select WITH_GPU.

2. If you are using the openblas version, de-select WITH_MKL.

3. In the compiling in the windows environment, YAML is downloaded automatically. If you can’t access the external network, you can download yamldcpp.zip manually. After downloading the YAML file, you don’t need to decompress it, just change the website in the URL https://bj.bcebos.com/paddlex/deploy/deps/yaml-cpp.zip to the path of the downloaded file in cmake/yaml.cmake.

4. If you use the model encryption function, you need to download the Windows Prediction Model Encryption Tool manually. For example, decompress it to D:/projects. The directory after decompression is D:/projects/paddlex-encryption. When compiling, select WITH_EBNCRYPTION and fill in D:/projects/paddlex-encryption in ENCRTPYPTION_DIR.
After the settings are complete, click Save to generate CMake cache to load the variables.

5. Choose Generate->Generate All
Step 5: Prediction and visualization

Before loading the model, make sure that the files in your model directory should include model.yml, __model__, and __params__. If this condition is not met, refer to the Deploy Model Export to export your model to the deployment format.

The above compiled executable files in Visual Studio 2019 are in the out\build\x64-Release directory. Run cmd to go to the directory:

```
D:
cd D:\projects\PaddleX\deploy\cpp\out\build\x64-Release
```

- After successful compilation, the entry program of the image prediction demo is paddlex_inference\detector.exe, paddlex_inference\classifier.exe, paddlex_inference\segmenter.exe, users can choose according to their own model types. Its main command parameters are described as follows:

- After the successful compilation, the entry program for the video prediction demo is paddlex_inference\video_detector.exe, paddlex_inference\video_classifier.exe, paddlex_inference\video_segmenter.exe. Users can choose according to their model type. The main command parameters are described as follows:

Note: If the GUI is unavailable in the system, you should not set show_result to 1. When using a camera for prediction, press ESC to disable the camera and launch the prediction program.

Example

You can use the inference_model and test pictures exported from the DUDU recognition model to make predictions. For example, export to D:\projects. The model path is D:\projects\inference_model.
Description about the prediction speed: The prediction speed of the first few images after loading the model is slow, because the initialization of the video card and memory is involved in the start-up. Generally, the prediction speed after predicting 20-30 images is stable.

Example 1: (Use the unencrypted model to predict a single image)

Test image without GPU: D:\images\xiaoduxiong.jpeg

```
.\paddlex_inference\detector.exe --model_dir=D:\projects\inference_model --image=D:\images\xiaoduxiong.jpeg --save_dir=output
```

The image file visual predictions are saved in the directory where the save_dir parameter is set.

Example 2: (Use the unencrypted model to predict the image list)

Use GPU to predict multiple images D:\images\image_list.txt, the format of the content of image_list.txt is as follows:

```
D:\images\xiaoduxiong1.jpeg
D:\images\xiaoduxiong2.jpeg
...
D:\images\xiaoduxiongN.jpeg
```

```
.\paddlex_inference\detector.exe --model_dir=D:\projects\inference_model --image_list=D:\images\image_list.txt --use_gpu=1 --save_dir=output --batch_size=2 --thread_num=2
```

The image file visual predictions are saved in the directory where the save_dir parameter is set.

Example 3: (Use the encrypted model to predict a single picture)

If the model is not encrypted, please refer to the encrypted PaddleX model to encrypt the model. For example, the directory where the encrypted model is located at D:\projects\encrypted_inference_model.

```
.\paddlex_inference\detector.exe --model_dir=D:\projects\encrypted_inference_model --image=D:\images\xiaoduxiong.jpeg --save_dir=output --key=kLAl1qOs5uRbFt0/RRrIDTW2+tf5b5vUIaHGF8lJ1c=
```

--key: pass in the key output from the encryption tool, for example, kLAl1qOs5uRbFt0/RRrIDTW2+tf5b5vUIaHGF8lJ1c=, the image file visual prediction result will be saved in the directory where the save_dir parameter is set.
Example 4: (Use an unencrypted model to enable camera prediction)

```
./paddlex_inference/video_detector.exe --model_dir=D:/projects/inference_model --use_camera=1 --use_gpu=1 --save_dir=output
```

When `save_result` is set to 1, the visual prediction results are saved in the directory where the `save_dir` parameter is set in the video file format.

Example 5: (Use an unencrypted model to make predictions on video files)

```
./paddlex_inference/video_detector.exe --model_dir=D:/projects/inference_model --video_path=D:/projects/video_test.mp4 --use_gpu=1 --show_result=1 --save_dir=output
```

When `save_result` is set to 1, the visual prediction results are saved in the directory where the `save_dir` parameter is set in the video file format. If the GUI is available in the system, view the visual prediction results on the screen by setting `show_result` to 1.

13.2.2 Linux platform deployment

Description

This document is tested with GCC 4.8.5 and GCC 4.9.4 in the Linux. To compile it with a later G++ version, you need to recompile the Paddle Prediction Library. Refer to Compiling Paddle Prediction Library from source code.

Pre-conditions

- G++ 4.8.2 ~ 4.9.4
- CUDA 9.0 / CUDA 10.0, CUDNN 7+ (required only if using GPU version prediction library)
- CMake 3.0+

Make sure that the above basic software is installed on your system. **All the following examples are in the working directory `/root/projects/`.**

Step1: Download the code.

`git clone https://github.com/PaddlePaddle/PaddleX.git`

**Note:** The C++ prediction code is in `/root/projects/PaddleX/deploy/cpp` directory. This directory does not depend on any other directory under PaddleX.

The PaddlePaddle C++ prediction library provides different pre-compiled versions for different CPUs, CUDAs, and whether to support TensorRT. At present, PaddleX depends on the Paddle 1.8.4 version. The following provides a number of different versions of Paddle prediction libraries:

For more and later versions, you can download as required: [C++ prediction library download list](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/advanced_guide/inference_deployment/inference/build_and_install_lib_cn.html)

The directory `/root/projects/fluid_inference` after downloading and decompression contains the following contents:

```
fluid_inference  paddle  # paddle core library and header files
|                  third_party  # third-party dependency library and header files
|                  version.txt  # Version and compilation information
```

**Note:** Except for `nv-jetson-cuda10-cudnn7.5-trt5`, other packages in the pre-compiled versions are based on GCC 4.8.5. There may be a ABI compatibility problem in the use of the later version of GCC. It is recommended to downgrade or [compile the prediction library yourself](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/advanced_guide/inference_deployment/inference/build_and_install_lib_cn.html#id12).

Step3: Compile

The command to compile `cmake` is in `scripts/build.sh`. Modify the main parameters as required. The main contents are described as follows:

```
# Whether GPU is used (i.e., whether CUDA is used)
WITH_GPU=OFF
# Use MKL or openblas
WITH_MKL=ON
# Whether or not to integrate TensorRT (only WITH_GPU=ON available)
WITH_TENSORRT=OFF
# TensorRT path. If you need to integrate TensorRT, change the path to the actual path you installed. TENSORRT_DIR=/root/projects/TensorRT/
# Paddle prediction library path. Change the path to the actual prediction library you installed.
PADDLE_DIR=/root/projects/fluid_inference
# Whether or not Paddle's prediction library is compiled using a static library
# When TensorRT is used, Paddle's prediction library is usually a dynamic library
WITH_STATIC_LIB=OFF
```

(下页继续)
# CUDA's lib path
CUDA_LIB=/usr/local/cuda/lib64

# CUDNN's lib path
CUDNN_LIB=/usr/local/cuda/lib64

# Whether to load the encrypted model
WITH_ENCRYPTION=ON

# Path to the encryption tool. It may not be modified if you use the customized pre-
# compiling version.
sh $(pwd)/scripts/bootstrap.sh # Download the pre-compiling version of the encryption tool

ENCRYPTION_DIR=$(pwd)/paddlex-encryption

# OPENCV path. It may not be modified if you use the customized pre-compiling version.
sh $(pwd)/scripts/bootstrap.sh # Download the pre-compiled version of opencv

OPENCV_DIR=$(pwd)/deps/opencv3gcc4.8/

# You should not modify the following:
rm -rf build
mkdir -p build
cd build
cmake .. \
  -DWITH_GPU=${WITH_GPU} \
  -DWITH_MKL=${WITH_MKL} \
  -DWITH_TENSORRT=${WITH_TENSORRT} \
  -DWITH_ENCRYPTION=${WITH_ENCRYPTION} \
  -DTENSORRT_DIR=${TENSORRT_DIR} \
  -DPADDLE_DIR=${PADDLE_DIR} \
  -DWITH_STATIC_LIB=${WITH_STATIC_LIB} \
  -DCUDA_LIB=${CUDA_LIB} \
  -DCUDNN_LIB=${CUDNN_LIB} \
  -DENCRYPTION_DIR=${ENCRYPTION_DIR} \
  -DOPENCV_DIR=${OPENCV_DIR}
make

**Note: In the compiling in the Linux, the OPENCV, PaddleX-Encryption, and YAML are automatically downloaded. If the access to the Internet is unavailable in the compiling environment, you can download manually:

- opencv3.4.6gcc4.8ffmpeg.tar.gz
- paddlex-encryption.zip

Chapter 13. CPU / GPU (encryption) deployment
Download opencv3gcc4.8.tar.bz2. Then, unzip it. Specify OPENE_DIR in script/build.sh as the path to unzip.

Download paddlex-encryption.zip. Then unzip it. Specify ENCRYPTION_DIR in script/build.sh as the path to unzip.

After downloading the yaml-cpp.zip file, you don’t need to decompress it. In cmake/yaml.cmake, change the website in the URL https://bj.bcebos.com/paddlex/deploy/deps/yaml-cpp.zip to the path of the downloading file.

After setting the main parameters of the modified script, run the build script.

```
sh ./scripts/build.sh
```

Step4: Prediction and visualization

**Before loading the model, make sure that the files in your model directory should include model.yml, __model__, and __params__. If this condition is not met, refer to the Model Export to Inference document to export your model to the deployment format.**

- After successful compilation, the executable programs for the image prediction demo are build/demo/detector, build/demo/classifier, and build/demo/segmenter. Users can choose according to their model type. The main command parameters are as follows:

- After successful compilation, the executable programs of the video prediction demo are build/demo/video_detector, build/demo/video_classifier, and build/demo/video_segmenter. Users can choose according to the model type. The main command parameters are as follows:

Note: If the GUI is unavailable in the system, you should not set show_result to 1. When using a camera for prediction, press ESC to disable the camera and launch the prediction program.

Example

Predictions can be made using the inference_model and test images exported from the [DUDU recognition model](../../export_model.md), to export to /root/projects. The model path is /root/projects/inference_model.

Description about the prediction speed: The prediction speed of the first few images after loading the model is slow, because the initialization of the video card and memory is involved in the start-up. Generally, the prediction speed after predicting 20-30 images is stable.

Example 1:

Not using GPU test images: /root/projects/images/xiaoduxiong.jpeg
The image file visual predictions are saved in the directory where the save_dir parameter is set.

Example 2:
Using the GPU to predict multiple images /root/projects/image_list.txt. The content of image_list.txt is in the following format:

```
```

The image file visual predictions are saved in the directory where the save_dir parameter is set.

Example 3:
Using the camera prediction:

```
./build/demo/video_detector --model_dir=/root/projects/inference_model --use_camera=1 --use_gpu=1 --save_dir=output --save_result=1
```

When save_result is set to 1, the visual prediction results are saved in the directory where the save_dir parameter is set in the video file format.

Example 4:
Predicting the video file:

```
./build/demo/video_detector --model_dir=/root/projects/inference_model --video_path=/path/to/video_file --use_gpu=1 --save_dir=output --save_result=1
```

When save_result is set to 1, the visual prediction results are saved in the directory where the save_dir parameter is set in the video file format. If the GUI is available in the system, view the visual prediction results on the screen by setting show_result to 1.

13.2.3 C++ code interface description

Chapter 13. CPU / GPU (encryption) deployment
Header file

include/paddlex/paddlex.h

Class PaddleX::Model

Model class is used to load models trained by PaddleX.

Complete model loading

```
PaddleX::Model::Init(const std::string& model_dir,
    bool use_gpu = false,
    bool use_trt = false,
    bool use_mkl = true,
    bool mkl_thread_num = 4,
    int gpu_id = 0,
    std::string key = "",
    bool use_ir_optim = true)
```

Parameters

- model_dir: path to the model directory
- use_gpu: whether to use GPU for prediction
- use_trt: whether or not to TensorRT
- use_mkl: whether or not to use MKLDNN to accelerate the predicted performance of the model on the CPU
- mkl_thread_num: the number of threads when MKLDNN is used
- gpu_id: ID of the GPU
- key: model decryption key. It is used when the encrypted PaddleX model is loaded.
- use_ir_optim: whether to speed up the model after image optimization

Returned value

- Returns true or false, indicating whether the model is loaded successfully.

Model prediction inference

Single image prediction of category model
PaddleX::Model::predict(const cv::Mat & im, ClsResult* result)

Batch prediction for multiple pictures in category model

PaddleX::Model::predict(const std::vector<cv::Mat>& im_batch, std::vector<ClsResult>* results)

Object detection/instance segmentation model single picture prediction

PaddleX::Model::predict(const cv::Mat & im, DetResult* result)

Object detection/instance segmentation model batch prediction for multiple images

PaddleX::Model::predict(const std::vector<cv::Mat>& im_batch, std::vector<DetResult>* results)

Single image prediction for semantic segmentation model

PaddleX::Model::predict(const cv::Mat & im, SegResult* result)

Multiple image batch prediction for semantic segmentation model

PaddleX::Model::predict(const std::vector<cv::Mat>& im_batch, std::vector<SegResult>* results)

The return value of each interface is true or false, indicating whether the prediction is successful or not.

In the prediction, the cv::Mat structure should be passed in, and the structure should be identical to the one loaded by the following code

```cpp
cv::Mat im = cv::imread('test.jpg', 1);
```

In the use of batch prediction, it should be noted that all the data in the incoming vector is predicted as a batch. Therefore, the larger the vector, the higher the GPU video memory required.

In the prediction, the ClsResult/DetResult/SegResult structure is passed in at the same time to store the prediction result of the model. The description of each structure is as follows:

```cpp
// Prediction results of category models
class ClsResult {
public:
  int category_id; // Category id
  std::string category; // Category label
  float score; // Prediction confidence level
};
```
std::string type = "cls";
}

// Prediction results of object detection/instance segmentation model
class DetResult {
public:
    std::vector<Box> boxes; // Each object box in the prediction result
    int mask_resolution; std::string type = "det";
}

// Prediction results of semantic segmentation model
class SegResult : public BaseResult {
public:
    Mask<int64_t> label_map; // Category of each pixel in the prediction segmentation
    Mask<float> score_map; // Confidence level of each pixel in the prediction segmentation
    std::string type = "seg";
}

struct Box {
    int category_id; // Category id
    std::string category; // Category label
    float score; // Confidence level
    std::vector<float> coordinate; // 4 element values, indicating xmin, ymin, width, height
    Mask<int> mask; // In instance segmentation, it represents the segmentation result of inside Box
}

struct Mask {
    std::vector<T> data; // the label map or score map in the segmentation
    std::vector<int> shape; // represents the shape of the segmented graph.
}

Visualization of predicted results

Object detection/instance segmentation result visualization

PaddleX::Visualize(const cv::Mat& img, // original image)
                    const DetResult& result, // prediction result

PaddleX

const std::map<int, std::string>& labels // each class of info <id, label_name>

Returns the cv::Mat structure, that is, the result of visualization

Visualization of semantic segmentation results

PaddleX::Visualize(const cv::Mat& img, // original image
                   const SegResult& result, // prediction result
                   const std::map<int, std::string>& labels // each class of info <id, label_name>
                   )

Returns the cv::Mat structure, that is, the result of visualization

Code example:

- Image classification PaddleX/deploy/cpp/demo/classifier.cpp
- Object detection/Instance segmentation PaddleX/deploy/cpp/demo/detector.cpp
- Semantic segmentation PaddleX/deploy/cpp/demo/segmenter.cpp

13.3 Model encryption deployment

PaddleX provides a lightweight model encryption deployment solution to encrypt inference models through PaddleX’s built-in model encryption tool. The Prediction Deployment SDK supports direct loading of cryptographic models and implements the inference to enhance the security of AI model deployment.

The encryption solution currently supports Windows and Linux systems.

13.3.1 1 Introduction to the solution

1.1 Introduction

(1) Selection of encryption algorithms and supported libraries

The OpenSSL library is generally used to support the encryption and decryption of data. The OpenSSL provides a large number of encryption and decryption algorithms, including symmetric encryption algorithms (for example, AES) and asymmetric encryption algorithms (for example, RSA).
The two algorithms are used in different scenarios. The asymmetric encryption algorithm is generally used in digital signature and key negotiation scenarios. The symmetric encryption algorithm is generally used in pure data encryption scenarios, and features better performance. The symmetric encryption algorithm is used in the encryption process of the model.

In the following model encryption scenario, AES symmetric encryption algorithm is used based on the development of a C/C++ library. In order to quickly determine whether the decryption is successful before and after encryption and decryption, AES-GCM encryption and decryption mode is used, with key data of a length of 256 bits.

(2) General steps to perform model protection:

1. With the consideration of loading data from memory after decrypting the encrypted model file, the combine mode is used to generate the model file and parameter file.

2. The project is integrated with OpenSSL in the use of the static library.

3. Implement the AES algorithm interface, with the help of the EVP interface provided by OpenSSL, and specify the algorithm type in the EVP interface. The algorithm uses AES in the symmetric encryption and decryption algorithm. The encryption and decryption mode uses AES-GCM, the key length is 256 bits. For the implementation of AES-GCM, refer to the official example to encapsulate the interface: [AES-GCM implementation](https://wiki.openssl.org/index.php/EVP_Authenticated_Encryption_and_Decryption).

4. Use the OpenSSL library to implement SHA256 digest algorithm. The following part is useful (optional). For the SHA256 hash algorithm, refer to the example provided by OpenSSL: [OpenSSL info Digest example](https://wiki.openssl.org/index.php/EVP_Message_Digests)

5. In the model encryption, the data content of the model file and params file is encrypted directly and then saved to a new file. In order to distinguish and iterate the new file, header information is added in addition to the encrypted data. For example, use a fixed magic number as the file header to determine the file type; write the version number to show the difference to facilitate iterations later; store the encrypted key after SHA256 calculation in order to determine whether the same key is used when
decrypting. These three parts constitute the header information of the currently encrypted file. The encrypted file contains the header information + key text information.

6. In the model decryption, the relevant encrypted data is read into the memory according to the encrypted file. The memory data is decrypted using AES algorithm. Note that when decrypting, you need to use the same encryption algorithm and encryption mode as the encryption, as well as the data and length of the key. Otherwise, it causes data error after decryption.

7. For the C/C++ library integrating the model prediction, the specific use of prediction usually involves paddle:: AnalysisConfig and paddle:: Predictor. In order to load the decrypted model plaintext data directly from the memory data (to avoid creating temporary files after the model is decrypted), the model loading function of AnalysisConfig needs to be replaced from SetModel to SetModelBuffer. This method can realize that model data is loaded from memory.

Note that in this solution, the key is integrated in the code of the upper-level prediction service. Therefore, the security strength of the model is equal to the strength of the code against reverse debugging. In order to protect the security of the key and model, the developer also needs to harden and protect the application. Common application hardening methods include code obfuscation, binary file shelling, and so on. Alternatively, change the encryption mechanism to AES white box encryption technology to protect the key. There are a large number of commercial and open source products available in the technology field. The details are not described here.

1.2 Encryption tools

For the Linux-based PaddleX model encryption tool, the compiling script automatically downloads this version of the encryption tool. You can also choose to download manually.

For the Windows-based PaddleX model encryption tool, this version of the encryption tool needs to be downloaded manually. If you already downloaded the tool during the compilation of C++ prediction codes using Visual Studio 2019, you do not need to repeat the download here.

The Linux encryption tool contains the following:

```
paddlex-encryption
  include # header file: paddle_model_decrypt.h (decrypt) and paddle_model_encrypt.h
| (encrypt)
| lib # libpmodel-encrypt.so and libpmodel-decrypt.so dynamic libraries
| tool # paddle_encrypt_tool
```

The Windows encryption tool contains the following:

```
paddlex-encryption
  include # header file: paddle_model_decrypt.h (decrypt) and paddle_model_encrypt.h
| (encrypt) (下页继续)
```
1.3 Encrypting the PaddleX model

After encrypting the model, the encryption tool generates a random key (used for AES encryption and decryption). It is used for decryption when the encryption is deployed later.

The key is composed of 32-byte key + 16-byte iv. Note that the key is base64 encoded to extend the range of key selection.

**Linux platform:**

```
# Assume that the model is under /root/projects
/paddlex-encryption/tool/paddle_encrypt_tool -model_dir /root/projects/paddlex_inference_ 
   -model -save_dir /root/projects/paddlex_encrypted_model
```

**Windows platform:**

```
# Assume that the model is under D:/projects
\paddlex-encryption\tool\paddle_encrypt_tool.exe -model_dir D:\projects\paddlex_ 
   -inference_model -save_dir D:\projects\paddlex_encrypted_model
```

- **model_dir:** Specify the inference model path (refer to Export inference model to export the model to an inference format model). You can use the `inference_model` exported from the export of the DUDU recognition model. After the encryption is completed, the encrypted model is saved to the specified `-save_dir`, including the files `__model__.encrypted`, `__params__.encrypted` and `model.yml`. The key is generated at the same time. The command output is shown in the following figure. The key is `kLA11qOs5uRbFt0/RrIDTW2+t0f5bFvUIaHGF8lJ1c=

**Output:** Encryption key:

```
  kLA11qOs5uRbFt0/RrIDTW2+t0f5bFvUIaHGF8lJ1c=
Success, Encrypt __model__, __params__ to paddle_encrypt_model(dir) success!
```

13.3.2 2 PaddleX C++ Encryption Deployment

2.1 Using the Linux platform

Refer to the Linux platform compilation guide to compile C++ deployment code. After successful compiling, the executables for the predicted demo are `build/demo/detector`, `build/demo/classifier`, and `build/
PaddleX

demo/segmenter. You can choose according to their model types. The main command parameters are described below.

Example

Predictions can be made using a test image [exported from the DUDU recognition model] (../export_model.md).

Example 1:

Not using GPU test images: /root/projects/images/xiaoduxiong.jpeg

```bash
./build/demo/detector --model_dir=/root/projects/paddlex_encrypted_model --image=/root/.../projects/xiaoduxiong.jpeg --save_dir=output --key=kLA1lqOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=
```

--key: pass in the key output from the encryption tool, for example, kLA1lqOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=, the image file visual prediction result will be saved in the directory where the save_dir parameter is set.

Example 2:

Using the GPU to predict multiple images /root/projects/image_list.txt. The content of image_list.txt is in the following format:

```
/root/projects/images/xiaoduxiong1.jpeg
/root/projects/images/xiaoduxiong2.jpeg

/root/projects/images/xiaoduxiongn.jpeg
```

```bash
./build/demo/detector --model_dir=/root/projects/models/paddlex_encrypted_model --image_list=/root/projects/images_list.txt --use_gpu=1 --save_dir=output --key=kLA1lqOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=
```

--key: pass in the key output from the encryption tool, for example, kLA1lqOs5uRbFt0/RrIDTZW2+tOf5bzvUIaHGF8lJ1c=, the image file visual prediction result will be saved in the directory where the save_dir parameter is set.

2.2 Using Windows platform

Refer to the [compilation guide of Windows platform] (cpp/windows.md). You need to download the Windows PaddleX encryption tool compression package, decompress it, and check WITH_ENCRYPTION in
the CMake settings based on the compilation process of the compilation guide. Set ENCRYPTION_DIR
to the directory after the encryption tool package is decompressed, and then carry out the compiling.
The parameters are consistent with those in the Linux prediction deployment. The entry programs
for the prediction demo are paddlex_inference\detector.exe, paddlex_inference\classifier.exe, and pad-
dlex_inference\segmenter.exe.

Example

Predictions can be made using a test image exported from the DUDU recognition model.

Example 1:

Test a single image without using the GPU. For example, the image is D:\images\xiaoduxiong.jpeg, and
the model directory after encryption is D:\projects\paddlex_encrypted_model.

```bash
.
paddlex_inference\detector.exe --model_dir=D:\projects\paddlex_encrypted_model --
'image=D:\images\xiaoduxiong.jpeg --save_dir=output --key=kLA1q0s5uRbFt0/
'RrIDTZW2+tOf5bzwUIaHGF81J1c=
```

--key: pass in the key output from the encryption tool, for example, kLA1q0s5uRbFt0/
RrIDTZW2+tOf5bzwUIaHGF81J1c=, the image file visual prediction result will be saved in the directory where
the save_dir parameter is set.

Example 2:

Using the GPU to predict the image list, for example, the image list is D:\projects\image_list.txt
and the content of image_list.txt is as follows.

```text
D:\projects\images\xiaoduxiong1.jpeg
D:\projects\images\xiaoduxiong2.jpeg
...
D:\projects\images\xiaoduxiong1n.jpeg
```

Model directory after encryption: D:\projects\paddlex_encrypted_model

```bash
.
paddlex_inference\detector.exe --model_dir=D:\projects\paddlex_encrypted_model --image_ 
--list=D:\projects\image_list.txt --use_gpu=1 --save_dir=output --key=kLA1q0s5uRbFt0/
'RrIDTZW2+tOf5bzwUIaHGF81J1c=
```

--key: pass in the key output from the encryption tool, for example, kLA1q0s5uRbFt0/
RrIDTZW2+tOf5bzwUIaHGF81J1c=, the image file visual prediction result will be saved in the directory
where the save_dir parameter is set.

13.3. Model encryption deployment
14.1 Description

This document describes the test with GCC 7.4 on the Linux platform based on Nvidia Jetpack 4.4. If you want to use a different G++ version, you need to recompile the Paddle prediction library. For details, see the compiling of [NVIDIA Jetson embedded hardware prediction library source codes](https://www.paddlepaddle.org.cn/documentation/docs/zh/develop/advanced_guide/inference_deployment/inference/build_and_install_lib_cn.html#id12).

14.2 Pre-conditions

- G++ 7.4
- CUDA 10.0 / CUDNN 8 (required only if using the prediction library in GPU version)
- CMake 3.0+

Make sure that the above basic software is installed on your system. **All the following examples are in the working directory /root/projects/**.

14.2.1 Step1: Download the code

`git clone https://github.com/PaddlePaddle/PaddleX.git`

**Note:** The C++ prediction code is in PaddleX/deploy/cpp, which does not depend on any other directories inPaddleX.
14.2.2 Step2: Download PaddlePaddle C++ prediction library: paddle_inference.

PaddlePaddle currently provides a C++ prediction library for Nvidia Jetson based on version 1.6.2.

The directory `/root/projects/fluid_inference` after downloading and decompression contains the following contents:

```plaintext
fluid_inference
   paddle # paddle core library and header files
| third_party # third-party dependency library and header files
| version.txt # version and compilation information
```

14.2.3 Step3: Compile

The command to compile `cmake` is in `scripts/jetson_build.sh`. Modify the main parameters as required. Its main content is described as follows:

```bash
# Whether GPU is used (i.e., whether CUDA is used)
WITH_GPU=OFF
# Use MKL or openblas
WITH_MKL=OFF
# Whether or not to integrate TensorRT (only WITH_GPU=ON available)
WITH_TENSORRT=OFF
# TensorRT path. If you need to integrate TensorRT, change the path to the actual TensorRT you installed.
TENSORRT_DIR=/root/projects/TensorRT/
# Paddle prediction library path. Change the path to the actual prediction library you installed.
PADDLE_DIR=/root/projects/fluid_inference
# Whether or not Paddle's prediction library is compiled using a static library
# When TensorRT is used, Paddle's prediction library is usually a dynamic library
WITH_STATIC_LIB=OFF
# CUDA's lib path
CUDA_LIB=/usr/local/cuda/lib64
# CUDNN's lib path
CUDNN_LIB=/usr/local/cuda/lib64

# You should not modify the following:
rm -rf build
```

(下页继续)
Step 4: Prediction and visualization

Before loading the model, make sure that the files in your model directory should include `model.yml`, `__model__`, and `__params__`. If this condition is not met, refer to the Model Export to Inference document to export your model to the deployment format.

- After successful compilation, the executable programs for the image prediction demo are `build/demo/detector`, `build/demo/classifier`, and `build/demo/segmenter`. Users can choose according to their model type. The main command parameters are as follows:

- After successful compilation, the executable programs of the video prediction demo are `build/demo/video_detector`, `build/demo/video_classifier`, and `build/demo/video_segmenter`. Users can choose according to the model type. The main command parameters are as follows:

Note: If the GUI is unavailable in the system, you should not set `show_result` to 1. When using a camera for prediction, press ESC to disable the camera and launch the prediction program.
14.3 Example

Predictions can be made using the `inference_model` and test images exported from the DUDU recognition model, to export to `/root/projects`. The model path is `/root/projects/inference_model`.

**Example 1:**

Not using GPU test images: `/root/projects/images/xiaoduxiong.jpeg`

```bash
./build/demo/detector --model_dir=/root/projects/inference_model --image=/root/projects/images/xiaoduxiong.jpeg --save_dir=output
```

The image file visual predictions are saved in the directory where the `save_dir` parameter is set.

**Example 2:**

Using the GPU to predict multiple images `/root/projects/image_list.txt`. The content of `image_list.txt` is in the following format:

```none
/root/projects/images/xiaoduxiong1.jpeg
/root/projects/images/xiaoduxiong2.jpeg
...
/root/projects/images/xiaoduxiongn.jpeg
```

```bash
./build/demo/detector --model_dir=/root/projects/inference_model --image_list=/root/projects/images_list.txt --use_gpu=1 --save_dir=output --batch_size=2 --thread_num=2
```

The image file visual predictions are saved in the directory where the `save_dir` parameter is set.

**Example 3:**

Using the camera prediction:

```bash
./build/demo/video_detector --model_dir=/root/projects/inference_model --use_camera=1 --use_gpu=1 --save_dir=output --save_result=1
```

When `save_result` is set to 1, the visual predictions are saved in the directory where the `save_dir` parameter is set in the video file format.

**Example 4:**

Predicting the video file:

```bash
./build/demo/video_detector --model_dir=/root/projects/inference_model --video_path=/path/to/video_file --use_gpu=1 --save_dir=output --show_result=1 --save_result=1
```
When `save_result` is set to 1, the visual predictions are saved in the directory where the `save_dir` parameter is set in the video file format. If the GUI is available in the system, view the visual prediction results on the screen by setting `show_result` to 1.
The Android deployment of PaddleX is based on Paddle Lite, the deployment process is as follows: export the trained model as reference model, then optimize the model, and finally use the Paddle Lite prediction library to perform the deployment. For the detailed introduction and use of Paddle Lite, see the Paddle Lite document.

PaddleX → Inference Model → Paddle Lite Opt → Paddle Lite Inference

Introduction:

1. Describes how to export PaddleX to an inference model.
2. Optimizes the model using Paddle Lite’s OPT module.
3. Describes the Android demo based on PaddleX Android SDK and how to quickly deploy the trained model.
4. Describes the PaddleX Android SDK and Secondary Development

15.1 1. Export the PaddleX model to an inference model.

Refer to the export inference model to export the model to an inference format.

15.2 2. Optimize the inference model to a Paddle Lite model

Two methods are available to optimize the Paddle model to the Paddle Lite model:
1. The python script optimization model features ease-of-use. Currently, the latest version is PaddleLite version 2.6.1.

1. The bin file optimization model (Linux) supports the develop version (Commit Id: 11cbd50e). Only bin file optimization can be used in the deployment of the semantic segmentation DeepLab model and Unet model.

### 15.2.1 2.1 Using python script optimization model

```bash
pip install paddlelite
python export_lite.py --model_dir /path/to/inference_model --save_file /path/to/lite_\n--model_name --place place/to/run
```

For the `export_lite.py` script, download it by accessing the github: https://github.com/PaddlePaddle/PaddleX/blob/develop/deploy/lite/export_lite.py

### 15.2.2 2.3 Optimizing models with bin files (Linux)

First download and decompress: Model Optimizer opt

```bash
./opt --model_file=<model_path> \ 
--param_file=<param_path> \ 
--valid_targets=arm \ 
--optimize_out_type=naive_buffer \ 
--optimize_out=model_output_name
```

For detailed usage and parameter meaning, refer to [Using the opt conversion model] (https://paddle-lite.readthedocs.io/zh/latest/user_guides/opt/opt_bin.html)

### 15.3 3 Mobile (Android) Demo

PaddleX provides an Android demo based on PaddleX Android SDK, located in /PaddleX/deploy/lite/android/demo. This demo is preset with the MobilenetV2 model parameters. Users can directly import the demo into Android Studio and run the experience. The user can replace the preset Mobilenetv2 model parameters with other detection or segmentation models exported by PaddleX for prediction.

#### 15.3.1 3.1 Requirements

- Android Studio 3.4
- Android phone or development panel
3.2 Category Demo

3.2.1 Importing the project and running

- Start the Android Studio and click “Open an existing Android Studio project” in the “Welcome to Android Studio” window. In the pop-up path selection window, access the /PaddleX/deploy/lite/android/demo directory. At the bottom right corner, click “Open” to import the project.

- Connect an Android phone or development panel through a USB port.

- After loading the project, choose Run->Run ‘App’. In the pop-up “Select Deployment Target” window, select the connected Android device and click “OK”.

- Upon successful operation, the Android device loads an App called PaddleX Demo. By default, a test image is uploaded. In addition, the prediction is supported by taking photos and selecting photos from the gallery.

Note: In the project construction process, the system remotely downloads the Mobilenetv2 model, yml configuration file, test pictures, and PaddleX Android SDK.

3.3 Deploy a customized model

The demo also supports user-defined models to perform predictions. This can help users quickly verify the trained models. First, you have prepared the Lite model (.nb file) and yml configuration file as described in Step 1 to Step 2 (Note: specify --place=arm when exporting the Lite model). In the project view of Android Studio:

- Copy the .nb file to the /src/main/assets/model/ directory, and modify the MODEL_PATH_DEFAULT in the /src/main/res/values/strings.xml file according to the name of the .nb file.

- Copy the .yml file to the /src/main/assets/config/ directory, and modify the YAML_PATH_DEFAULT in the /src/main/res/values/strings.xml file according to the name of the .yml file.

- You can replace the test image as required, copy the image to the /src/main/assets/images/ directory, and modify the IMAGE_PATH_DEFAULT in the file /src/main/res/values/strings.xml according to the name of the image file.

- After importing the project, click the Run->Run ‘App’ button in the menu bar. In the displayed “Select Deployment Target” window, select the connected Android device, and then click “OK”.

4. PaddleX Android SDK and secondary development

PaddleX Android SDK is an Android AI reasoning tool developed by PaddleX based on Paddle Lite. It uses the Yaml configuration file exported by PaddleX as an interface to realize image preprocessing, post-processing, and visualization for different models. Developers can integrate it into the services. The SDK
mainly includes Paddle Lite inference engine layer, Paddle Lite interface layer, and PaddleX service layer from bottom to top.

- The Paddle Lite inference engine layer is a binary package compiled on Android. It only involves the execution of the Kernel and can be deployed separately to support extremely lightweight deployment.
- The Paddle Lite interface layer encapsulates the underlying C++ inference library with a Java interface.
- The PaddleX service layer encapsulates the pre-processing, inference and post-processing, and visualization of the PaddleX export model. It supports the detection, segmentation, and classification models exported by PaddleX.

15.4.1 4.1 SDK installation

First, download and decompress the PaddleX Android SDK, to obtain the paddlex.aar file, copy it to the android project directory app/libs/, and then add dependencies for the build.gradle of the APP:

```groovy
dependencies {
    implementation fileTree(include: ['*.jar', '*.aar'], dir: 'libs')
}
```

15.4.2 4.2 SDK example

```java
import com.baidu.paddlex.Predictor;
import com.baidu.paddlex.config.ConfigParser;
import com.baidu.paddlex.postprocess.DetResult;
import com.baidu.paddlex.postprocess.SegResult;
import com.baidu.paddlex.postprocess.ClsResult;
```
```java
import com.baidu.paddlex.visual.Visualize;

// Predictor
Predictor predictor = new Predictor();
// model config
ConfigParser configParser = new ConfigParser();
// Visualize
Visualize visualize = new Visualize();
// image to predict
Mat predictMat;

// initialize
configParser.init(context, model_path, yaml_path, cpu_thread_num, cpu_power_mode);
visualize.init(configParser.getNumClasses());
predictor.init(context, configParser);

// run model
if (predictImage != null && predictor.isLoaded()) {
    predictor.setInputMat(predictMat);
    runModel();
}

// get result & visualize
if (configParser.getModelType().equalsIgnoreCase("Segmenter")) {
    SegResult segResult = predictor.getSegResult();
    Mat visualizeMat = visualize.draw(segResult, predictMat, predictor.getImageBlob());
} else if (configParser.getModelType().equalsIgnoreCase("Detector")) {
    DetResult detResult = predictor.getDetResult();
    Mat visualizeMat = visualize.draw(detResult, predictMat);
} else if (configParser.getModelType().equalsIgnoreCase("Classifier")) {
   ClsResult clsResult = predictor.getClsResult();
}
```

### 15.4.3 4.3 Result member variables

**Note:** All the member variables of Result are obtained by way of java bean.
Fields

- **type** (String|static): The value is “cls”.
- **categoryId** (int): category ID.
- **category** (String): category name.
- **score** (float): prediction confidence.

```java
com.baidu.paddle.postprocess.DetResult
```

Nested classes

The box result predicted by the *DetResult.Box*.

Fields

- **type** (String|static): The value is “det”.
- **boxes** (List<DetResult.Box>): The box result predicted by the model.

```java
com.baidu.paddle.postprocess.DetResult.Box
```

Fields

- **categoryId** (int): category ID.
- **category** (String): category name.
- **score** (float): the confidence of the prediction box.
- **coordinate** (float[4]): The coordinate value of the prediction box {xmin, ymin, xmax, ymax}.

```java
com.baidu.paddle.postprocess.SegResult
```

Nested classes

- **SegResult.Mask**: The mask result predicted by the model.

Fields

- **type** (String|static): The value is “Seg”.
- **mask** (SegResult.Mask): The mask result predicted by the model.
Fields

- **scoreData** (float[]): the confidence of the model prediction in each category. The length is: 1 * numClass * H * W
- **scoreShape** (long[4]): shape information of scoreData, [1, numClass, H, W]
- **labelData** (long[]): The label with the highest model prediction confidence, the length is: 1 * H * W * 1
- **labelShape** (long[4]): shape information of labelData, [1, H, W, 1]

15.4.4 4.4 SDK secondary development

- Open Android Studio to create a new project (or load an existing project). Choose File->New->Import Module, and import the project /PaddleX/deploy/lite/android/sdk. The Project view is added with a module named sdk.
- Add dependency in build.grade of the APP:

```java
dependencies {
    implementation project(':sdk')
}
```

- The source code is located in sdk/main/java/. After modifying the source code for secondary development, choose Build->Run ‘sdk’ button in the menu bar to compile and generate aar. The file is located in the sdk/build/outputs/aar/path.
16.1 Raspberry

PaddleX supports the prediction deployment on the raspberry through both Paddle-Lite and the OpenVINO-based Neural Compute Stick (NCS2).

16.1.1 Hardware environment configuration

For a Raspberry without installing the system yet, you need to perform the system installation and environment configuration to initialize the hardware environment. The required software and hardware are as follows:

- **Hardware:** micro SD, monitor, keyboard, mouse
- **Software:** Raspbian OS

**Step1: System installation**

- Format micro SD card as FAT. In the Windows and Mac systems, the SD Memory Card Formatter tool is recommended. In the Linux system, refer to NOOBS For Raspberry Pi.
- Download the NOOBS Raspbian OS [download link] (https://www.raspberrypi.org/downloads/). Copy the decompressed file to SD. After the SD is inserted, the Raspberry is powered on. The system is installed automatically.
Step2: Environment configuration

- Start the VNC and SSH services: start the LX Terminal. Enter the following command, and select Interfacing Option. Then, select P2 SSH and P3 VNC to start the SSH and VNC respectively. After the startup, the Raspberry is connected through SSH or VNC.

```
sudo raspi-config
```

- Replace source: The official Raspberry source is very slow; therefore, it is recommended to check the official website of the domestic source Raspberry software. After the replacement, run the following:

```
sudo apt-get update
sudo apt-get upgrade
```

16.1.2 Paddle-Lite deployment

The Paddle-Lite-based deployment currently support PaddleX classification, segmentation and detection models. For the detection model, only YOLOV3 is supported.

Deployment process include: PaddleX model conversion and post-conversion model deployment

**Note:** For the PaddleX installation, refer to PaddleX. For the details of the Paddle-Lite, refer to Paddle-Lite. Make sure that the above basic software is installed on your system and that you have configured your environment accordingly. The following examples are based on the /root/projects/ directory.

16.1.3 Paddle-Lite model conversion

Convert the PaddleX model to Paddle-Lite model. For details, see Paddle-Lite Model Conversions.

16.1.4 Paddle-Lite prediction

**Step1 Download the PaddleX prediction code.**

```
mkdir -p /root/projects
cd /root/projects
git clone https://github.com/PaddlePaddle/PaddleX.git
```

**Note:** The C++ prediction code is in PaddleX/deploy/raspberry directory. This directory does not depend on any other directory under PaddleX. For the prediction deployment in the Python, refer to [Python prediction deployment] (./python.md).
Step2: Download the Paddle-Lite pre-compiling library.

Provide the Paddle-Lite pre-compiling library under ArmLinux corresponding to the downloaded opt tool: [Paddle-Lite (ArmLinux) pre-compiling library](https://bj.bcebos.com/paddlex/deploy/lite/inference_lite_2.6.1_armlinux.tar.bz2). The pre-compiling library is recommended. If you compile it yourself, enter the following command in the LX terminal on the Raspberry.

```
  git clone https://github.com/PaddlePaddle/Paddle-Lite.git
  cd Paddle-Lite
  sudo ./lite/tools/build.sh --arm_os=armlinux --arm_abi=armv7hf --arm_lang=gcc --build_ --extra=ON full_publish
```

Path of pre-compiling library: /build-lite.armlinux.armv7hf.gcc/inference_lite_lib.armlinux.armv7hf/cxx

Note: The prediction library version needs to be the same as the opt version. For more Paddle-Lite compiling contents, refer to Paddle-Lite Compiling. For more pre-compiling Paddle-Lite prediction library, refer to Paddle-Lite Release Note.

Step3 Software dependencies

Pre-compiling packages or one-key compilation of dependent software are provided. Users do not need to separately download or compile a third party dependent software. If you need to compile a third-party dependency software yourself, refer to:

- gflags: For compiling, refer to the Compiling Documents.
- opencv: For compiling, refer to [Compiling Documents](https://docs.opencv.org/master/d7/d9f/tutorial_linux_install.html).

Step4: Compile

Compile `cmake` in `scripts/build.sh`. Modify LITE_DIR to Paddle-Lite prediction library directory. If you compile a third-party dependency software, modify the main parameters as required in Step 1. The main content is described as follows:

```
# Path to the Paddle-Lite pre-compiling library
LITE_DIR=/path/to/Paddle-Lite/inference/lib
# Path to the gflags pre-compiling library
GFLAGS_DIR=$(pwd)/deps/gflags
# Path to the opencv pre-compiling library
OPENCV_DIR=$(pwd)/deps/opencv/
```

Run the `build` script:
Step3: Prediction

After successful compilation, the prediction executable program for the classification task is `classifier`, the prediction executable for the segmentation task is `segmenter`, and the prediction executable program for the detection task is `detector`. The main command parameters are as follows:

**Example**

**Example 1:** Single image classification task Test image `/path/to/test_img.jpeg`

```
./build/classifier --model_dir=/path/to/nb_model
--image=/path/to/test_img.jpeg --cfg_file=/path/to/PaddleX_model.yml --thread_num=4
```

**Example 2:** Multi-image segmentation task Prediction of multiple images: `/path/to/image_list.txt`. The format of the `image_list.txt` content is as follows:

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

```
./build/segmenter --model_dir=/path/to/models/nb_model --image_list=/root/projects/images_list.txt
--config_file=/path/to/PaddleX_model.yml --save_dir ./output --thread_num=4
```

**16.1.5 Performance test**

**Test environment:**

Hardware: Raspberry Pi 3 Model B System: raspbian OS Software: paddle-lite 2.6.1

**Test results**

Unit: ms. The `num` parameter indicates the number of threads used under paddle-lite.

From the test results, it is recommended that users use MobileNetV1-V3 and ShuffleNetV2, and other small networks on Raspberry.
16.1.6 NCS2 deployment

Raspberry supports the running of PaddleX model prediction on NCS2 through OpenVINO. Currently, only PaddleX classification network is supported. The NCS2-based method includes two steps: Paddle model converted to OpenVINO IR and deployment of IR on NCS2 for prediction.

- For model conversion, refer to: PaddleX model converted to OpenVINO IR. OpenVINO on raspbian OS does not support model conversion. You need to convert FP16 IR on the host side first.
- For the prediction deployment, refer to the VPU deployment in raspbian OS in [OpenVINO deployment](./openvino/linux.md).

16.2 Inference Deployment

This document describes how to use the Python Paddle-Lite for PaddleX model prediction deployment on Raspberry. You can install the Python Paddle-Lite prediction library according to the following command. If the installation fails, download the whl file to install Paddle-Lite _2.6.0_python. For more versions, see the Paddle-Lite Release Note.

```
python -m pip install paddlelite
```

Before deployment, you need to convert PaddleX model to Paddle-Lite nb model. For details, see Paddle-Lite model conversion. **Note:** If the Python Prediction Library 2.6.0 is used, download version 2.6.0 of the opt conversion tool to convert the model.

16.2.1 Pre-conditions

- Python 3.6+
- Paddle-Lite_python 2.6.0+

Make sure that the above basic software is installed on your system. **All the following examples are in the working directory /root/projects/**.

16.2.2 Inference deployment

Run the demo.py file in the /root/projects/PaddleX/deploy/raspberry/python directory to perform the prediction. The command parameters are described as follows:

**Note:** The Python API of the Paddle-lite doesn’t support the input of int64 data yet; therefore, Raspberry doesn’t support the deployment of YoloV3 in python. If it is required, use C++ codes to deploy YoloV3.
Example

Example 1: test images /path/to/test_img.jpeg

```
cd /root/projects/python

python demo.py --model_dir /path/to/openvino_model --img /path/to/test_img.jpeg --cfg_file /path/to/PaddleX_model.yml --thread_num 4
```

Example 2:

Prediction of multiple images: /path/to/image_list.txt. The format of the image_list.txt content is as follows:

```
/path/to/images/test_img1.jpeg /path/to/images/test_img2.jpeg ... /path/to/images/test_imgn.jpeg
```

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

```
cd /root/projects/python

python demo.py --model_dir /path/to/models/openvino_model --image_list /root/projects/ --images_list.txt --cfg_file=/path/to/PaddleX_model.yml --thread_num 4
```

16.3 Paddle-Lite model conversion

The PaddleX model is converted to Paddle-Lite nb model. The model conversion mainly includes PaddleX to inference model and inference model to Paddle-Lite nb model.

16.3.1 Step1: Export the inference model

Before converting PaddleX model to Paddle-Lite model, you need to export the PaddleX model to inference format first. The exported model includes three file names: `model`, `params`, and `model.yml`. For more details, refer to the Inference Model Export.
16.3.2 Step2: Export the Paddle-Lite model

The Paddle-Lite model needs to be converted through the Paddle-Lite opt tool. Download and decompress the model optimization tool opt (2.6.1-linux) and run it on Linux:

```
./opt --model_file=<model_path> \
   --param_file=<param_path> \
   --valid_targets=arm \
   --optimize_out_type=naive_buffer \
   --optimize_out=model_output_name
```

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>--model_file</td>
<td>Export the network structure file contained in the inference model: the path where <strong>model</strong> is located.</td>
<td></td>
</tr>
<tr>
<td>--param_file</td>
<td>Export the parameter file contained in the inference model: the path where <strong>params</strong> is located.</td>
<td></td>
</tr>
<tr>
<td>--valid_targets</td>
<td>Specify the model executable backend. Here it is specified as arm.</td>
<td></td>
</tr>
<tr>
<td>--optimize_out_type</td>
<td>Output model type. Currently supports two types: protobuf and naive_buffer, where naive_buffer is a more lightweight serialization/deserialization. Here it is specified as naive_buffer.</td>
<td></td>
</tr>
<tr>
<td>--optimize_out</td>
<td>Model output name</td>
<td></td>
</tr>
</tbody>
</table>

If the python Paddle-Lite is installed, it can also be converted in the following method:

```
./paddle_lite_opt --model_file=<model_path> \
   --param_file=<param_path> \
   --valid_targets=arm \
   --optimize_out_type=naive_buffer \
   --optimize_out=model_output_name
```

For more detailed instructions and parameter meanings, refer to Using the Opt Conversion Model. For more opt pre-compiling versions, see the Paddle-Lite Release Note.

**Note:** The opt version needs to be consistent with the prediction library version. If you want to use the 2.6.0 prediction library, download the opt conversion model of version 2.6.0 from the Release Note.
17.1 Introduction to OpenVINO deployment

PaddleX supports the prediction acceleration of the trained Paddle model through OpenVINO. For details and installation process, refer to OpenVINO document. This document is based on OpenVINO 2020.4 and 2021.1. **Note:** Resize-11 is supported starting from OpenVINO 2021.1 because the PaddleX segmentation model uses ReSize-11 Op. Make sure to download OpenVINO 2021.1+.

17.1.1 Deployment support

The following table lists the support status of using OpenVINO for acceleration by PaddleX in different environments

**Note:** Raspbian OS is the Raspberry OS. The detection model supports only YOLOv3

17.1.2 Deployment process

The PaddleX to OpenVINO deployment process has the following two steps:

- **Model conversion:** Convert Paddle’s model to OpenVINO’s Inference Engine.
- **Prediction Deployment:** Prediction with using Inference Engine**
17.1.3 Model conversion

For model conversion, refer to the Model Conversion document. Note: Since the methods of converting OpenVINO model are the same under different hardware and software platforms, details on how to convert the model are omitted in subsequent documents.

17.1.4 Inference deployment

The methods of deploying OpenVINO to implement predictions are not completely identical in different hardware and software. For details, refer to:

Linux: introduces the prediction acceleration by using OpenVINO when PaddleX operates on Linux or Raspbian OS with C++ programming language and hardware platform is CPU or VPU.

Windows: introduces the prediction acceleration by using OpenVINO when PaddleX operates on Windows OS with C++ programming language and hardware platform is CPU or VPU

Python: introduces the prediction acceleration by using OpenVINO when PaddleX operates in Python

17.2 Windows platform

17.2.1 Description

On the Windows platform, use the Visual Studio 2019 Community for testing. Since 2017, Microsoft Visual Studio has supported the direct management of CMake cross-platform compilation projects. But it did not provide stable and complete support until 2019. If you want to use CMake to manage project compilation and build, Visual Studio 2019 is recommended.

17.2.2 Pre-conditions

- Visual Studio 2019
- OpenVINO 2021.1+
- CMake 3.0+

Note: For PaddleX installation, refer to [PaddleX]. For OpenVINO installation, refer to [OpenVINO-Windows] (https://docs.openvinotoolkit.org/latest/opencv_doc_install_guide_installing_openvino_windows.html)

Note: After installing OpenVINO, you need to manually add the OpenVINO directory to the system environment variable. Otherwise, the dll may not be found when you run the program. For example, if you install OpenVINO without changing the OpenVINO installation directory, the process is as follows:

- My Computer->Properties->Advanced System Settings->Environmental Variables
  - Find Path in the system variables (if not, create one yourself and double-click to edit it.)
To create a new one, fill in the following paths for OpenVINO respectively, and save it:

- C:\Program Files (x86)\IntelSWTools\openvino\inference_engine\bin\intel64\Release
- C:\Program Files (x86)\IntelSWTools\openvino\inference_engine\external\tbb\bin
- C:\Program Files (x86)\IntelSWTools\openvino\deployment_tools\ngraph\lib

Make sure that you have installed the above basic software and configured your system accordingly. All the examples below are based on the working directory D:\projects.

17.2.3 Inference deployment

This document provides prediction deployment methods under C++. To perform prediction deployment under python, see python prediction deployment.

**Step1: Download the PaddleX prediction code.**

```bash
mkdir projects
cd projects
git clone https://github.com/PaddlePaddle/PaddleX.git
```

**Note:** The C++ prediction code is in the PaddleX\deploy\openvino directory. The directory does not depend on any other directory in PaddleX.

**Step2: Software dependencies**

Pre-compiled libraries for dependent software are provided:

- **gflags**
- opencv Download the pre-compiled libraries for the two links above. If you need to download them yourself, please refer to:
- **gflags:** download address
- opencv:** download address** After downloading opencv, you need to configure the environment variables as follows:
  - My Computer->Properties->Advanced System Settings->Environmental Variables
  - Find Path in the system variables (if not, create one yourself) and double-click to edit it.
  - Add a new file. Fill in the opencv path and save it. For example, D:\projects\opencv\build\x64\vc14\bin

17.2. Windows platform
**Step 3: Compile CMake directly by using Visual Studio 2019**

1. Open Visual Studio 2019 Community and click **Continue**, but no code is required

2. Choose **File**->**Open**->**CMake** to select the path where the C++ prediction code is located (for example, D:\projects\PaddleX\deploy\openvino), and open **CMakeList.txt**.

3. Choose **Project**->**CMake Settings**

4. Click **Browse** to set the compiling options, and specify the paths to **OpenVINO**, **Gflags**, **NGRAPH**, and **OPENCV** respectively.

**After the settings are complete**, click **Save** to generate the CMake cache to load the variables.

5. Choose **Generate**->**Generate All**

**Step 5: Prediction**

The above compiled executable files in **Visual Studio 2019** are in the **out\build\x64-Release** directory. Run **cmd** to go to the directory:

```
D:
\cd D:\projects\PaddleX\deploy\openvino\out\build\x64-Release
```

- After successful compilation, the entry program for the image prediction demo is **detector.exe**, **classifier.exe**, and **segmenter.exe**. You can choose according to the model types. Its main command parameters are described as follows:

**Example**

**Example 1**: Classification task prediction for a single image under the CPU Test image /path/to/test_img.jpeg

```
./ classifier. exe --model_dir=/path/to/openvino_model --image=/path/to/test_img.jpeg --
cfg_file=/path/to/PaddleX_model.yml
```

**Example 2**: Detection task prediction of multiple images under CPU and saving of the prediction visualization results Prediction of multiple images: /path/to/image_list.txt. The format of the image_list.txt content is as follows:

```
/ path/to/images/test_img1.jpeg
/ path/to/images/test_img2.jpeg
...
/ path/to/images/test_imgN.jpeg
```

---

**Chapter 17. OpenVINO Deployment**
Example 3: Classification task prediction for a single image under the VPU Test image /path/to/test_img.jpeg

```bash
./classifier.exe --model_dir=/path/to/openvino_model --image=/path/to/test_img.jpeg --cfg_file=/path/to/PaddleX_model.yml --device=MYRIAD
```

### 17.3 Linux platform

#### 17.3.1 Pre-conditions

- **OS**: Ubuntu, Raspbian OS
- **GCC**: 5.4.0
- **CMake**: 3.0+
- **PaddleX**: 1.0+
- **OpenVINO**: 2021.1+
- **Hardware platform**: CPU, VPU

**Note**: For PaddleX installation, see [PaddleX description](#). For the installation of OpenVINO, see [OpenVINO-Linux](#) or [OpenVINO-Raspbian](#) description according to the corresponding systems.

Make sure that the above basic software is installed on your system and that you have configured your environment accordingly. **The following examples are based on the /root/projects/ directory.**

#### 17.3.2 Inference deployment

This document provides prediction deployment methods under c++. To perform prediction deployment under python, see [python prediction deployment](#).

**Step1 Download the PaddleX prediction code.**

```bash
mkdir -p /root/projects
cd /root/projects
git clone https://github.com/PaddlePaddle/PaddleX.git
```

**Note**: The C++ prediction code is in PaddleX/deploy/openvino. The directory does not depend on any other directory in PaddleX.
Step2 Software dependencies

For the compiled scripts in Step3, the pre-compiled package of a third-party dependent software is installed by pressing one key. Users do not need to download or compile these dependent software separately. If you need to compile a third-party dependency software yourself, refer to:

- opencv: For compiling, refer to [Compiling Documents] (https://docs.opencv.org/master/d7/d9f/tutorial_linux_install.html).

Step3: Compile

The command to compile cmake is in scripts/build.sh. If the compiling is performed in Raspberry (Raspbian OS), modify the ARCH parameters x86 to armv7. If you compile your own third-party dependency software, modify the main parameters as required, according to the software compiled in Step 1. The main content is described as follows:

```bash
# Path of the openvino pre-compiling library
OPENVINO_DIR=$INTEL_OPENVINO_DIR/inference_engine
# Path to the gflags pre-compiling library
GFLAGS_DIR=$(pwd)/deps/gflags
# Path to the ngraph lib pre-compiling library
NGRAPH_LIB=$INTEL_OPENVINO_DIR/deployment_tools/ngraph/lib
# Path to the opencv pre-compiling library
OPENCV_DIR=$(pwd)/deps/opencv/
# cpu architecture (x86 or armv7)
ARCH=x86
```

Run the build script:

```
sh . /scripts/build.sh
```

Step4: Prediction

After successful compilation, the prediction executable program for the classification task is classifier, the prediction executable program for the detection task is detector, and the prediction executable program for the segmentation task is segmenter. The main command parameters are described as follows:

Example

Example 1: Classification task prediction for a single image under CPU in Linux Test image: /path/to/test_img.jpeg
Example 2: The Linux system performs multiple image detection task predictions under the CPU, and saves the prediction visualization results Predicted multiple images /path/to/image_list.txt, image_list.txt content in the following format.

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

Example 3: Raspbian OS Single image classification task prediction under VPU Test image: /path/to/test_img.jpeg

```
./build/classifier --model_dir=/path/to/openvino_model --image=/path/to/test_img.jpeg --cfg_file=/path/to/PaddleX_model.yml --device=MYRIAD
```

## 17.3.3 Performance Test

**Test 1:** The performance of OpenVINO acceleration on PaddleX deployments was tested at the server CPU.

- CPU: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
- OpenVINO: 2020.4
- PaddleX: using Paddle prediction library (1.8), to enable the mkldnn acceleration and start the multithreading.
- The model is from PaddleX tutorials, the Batch Size is 1, the time consumption unit is ms/image. Only the model running time is calculated, not including the pre-processing and post-processing of the data, 20 images warmup, 100 images for testing the performance.

**Test 2:** Inserting a VPU architecture Neural Compute Stick (NCS2) into a PC to accelerate through Openvino.

- CPU: Intel(R) Core(TM) i5-4300U 1.90GHz
- VPU: Movidius Neural Compute Stick2
- OpenVINO: 2020.4
The model is from PaddleX tutorials, the Batch Size is 1, the time consumption unit is ms/image. Only the model running time is calculated, not including the pre-processing and post-processing of the data, 20 images warmup, 100 images for testing the performance.

Test 3: Inserting a VPU architecture neural computation stick (NCS2) on the Raspberry 3B to accelerate through Openvino.

- CPU: ARM Cortex-A72 1.2GHz 64bit
- VPU: Movidius Neural Compute Stick2
- OpenVINO 2020.4

The model is from PaddleX tutorials, the Batch Size is 1, the time consumption unit is ms/image. Only the model running time is calculated, not including the pre-processing and post-processing of the data, 20 images warmup, 100 images for testing the performance.

17.4 Inference deployment

The document describes OpenVINO-based prediction deployment in python. Before deployment, you need to convert the paddle model to OpenVINO’s Inference Engine. For details, see Model Conversion. Currently, the classification, detection, and segmentation model of PaddleX is supported on CPU hardware; the classification model of PaddleX is supported on VPU.

17.4.1 Pre-conditions

- Python 3.6+
- OpenVINO 2021.1

Note: For OpenVINO installation, refer to OpenVINO description.

Make sure that the above basic software is installed on your system. All the following examples are in the working directory /root/projects/.

17.4.2 Inference deployment

Running the demo.py file in the /root/projects/PaddleX/deploy/openvino/python directory can make predictions with the following command parameters.

Example

Example 1: test images: /path/to/test_img.jpeg
Example 2:
Prediction of multiple images: /path/to/image_list.txt. The format of the image_list.txt content is as follows:

```
/path/to/images/test_img1.jpeg
/path/to/images/test_img2.jpeg
...
/path/to/images/test_imgn.jpeg
```

```
cd /root/projects/python
python demo.py --model_dir /path/to/openvino_model --img /path/to/test_img.jpeg --cfg_file /path/to/PaddleX_model.yml
```

17.5 OpenVINO model conversion

The document describes how to convert Paddle models to Inference Engine of the OpenVINO.

17.5.1 Environment dependence

- ONNX 1.6.0+
- PaddleX 1.2+
- OpenVINO 2021.1+

Note: For PaddleX installation, refer to PaddleX document. For OpenVINO installation, refer to OpenVINO document. For ONNX, install V1.6.0 or later; otherwise, the conversion error may occur.

Make sure that the above basic software is installed on your system. All the following examples are in the working directory /root/projects/.

17.5.2 Export an inference model

Before converting paddle model to opencvino, you need to export the paddle model to inference format first. The exported model includes model, params, and model.yml. The export command is as follows:
paddlex --export_inference --model_dir=/path/to/paddle_model --save_dir=./inference_ --model --fixed_input_shape=[w,h]

Note: If you need to convert the OpenVINO model to export the inference model, make sure to specify the --fixed_input_shape parameter to fix the input size of the model, and the input size of the model should be the same as the OpenVINO model during training.

17.5.3 Export an OpenVINO model

mkdir -p /root/projects
cd /root/projects
git clone https://github.com/PaddlePaddle/PaddleX.git
cd PaddleX/deploy/openvino/python

python converter.py --model_dir /path/to/inference_model --save_dir /path/to/openvino_ --model --fixed_input_shape [w,h]

After the conversion is successful, three files with suffixes .xml, .bin and .mapping appear under save_dir. The conversion parameters are described as follows:

Note:

- Because OpenVINO supports the ONNX resize-11 OP from version 2021.1, make sure to download OpenVINO 2021.1+.

- In the deployment of YOLOv3 through OpenVINO, due to the OpenVINO’s limitation support for ONNX OPs, the special processing is performed to the last layer of multiclass_nms to export the ONNX model when the Paddle model of YOLOv3 is exported. The final output Box results include the background category (the Paddle model does not include it). Here, in the deployment codes of OpenVINO, the background category is filtered through post-processing.
This case implements the detection and automatic reading of traditional mechanical pointer meters based on PaddleX, to open up the meter data and pre-training model, and provide the deployment guide for server-side on Windows-based systems and jetson embedded devices on Linux-based systems.

18.1 Reading flow

Meter readings are completed in three steps:

- Step 1: Detect the meter in the image by using the object detection model.
- Step 2: Use the semantic segmentation model to segment the pointer and scales of each meter.
- Step 3: Calculate the reading of each meter based on the relative position of the pointer and the predicted range.
• **Meter detection:** Since there is no meter with a small area in this case, the object detection model chooses YOLOv3 that has better performance. With the consideration that this case is mainly deployed on devices with GPUs, DarkNet53 with higher precision is chosen for the backbone network.

• **Scale and pointer segmentation:** With the consideration that the scale and pointer are in fine regions, the semantic segmentation model chooses DeepLapv3 that has better performance.

• **Post-processing of readings:** 1. The semantically segmented prediction class map is subjected to an image etching operation for the purpose of scale segmentation. 2. The ring-shaped dial is expanded into a rectangular image, and a one-dimensional scale array and a one-dimensional pointer array are generated based on the class information in the image. 3. The mean value of the scale array is calculated, to use the mean value of the scale array for the binary operation. 4. The position of the
pointer relative to the scale is located, to determine the type of dial according to the number of the scales to obtain the range of the dial, to multiply the relative position of the pointer and the range to get the reading of the dial.

### 18.2 Metering data and pre-training models

This case opens up meter test images for experiencing the full flow of prediction inference for meter readings. It also opens up the meter detection dataset, and pointer and scale segmentation dataset to allow users to use these datasets for experiencing the training model.

The case opens up pre-trained detection and semantic segmentation models, which can be used to quickly experience the full flow of the meter reading, or deployed directly on server-side or jetson embedded devices to perform the inference prediction.

### 18.3 Quick experience of dial readings

You can use the pre-trained model provided in this case to quickly experience the full flow of automatic prediction of meter readings. If you do not need a pre-trained model, you can go to the model training to restart the training model.

#### 18.3.1 Pre-dependence

- Paddle paddle >= 1.8.0
- Python >= 3.5
- PaddleX >= 1.0.0

For installation related issues, refer to [PaddleX Installation](../install.md).

#### 18.3.2 Test meter readings

Step 1. Download PaddleX source code:

```
git clone https://github.com/PaddlePaddle/PaddleX
```

Step 2. The prediction execution file is located in PaddleX/examples/meter_reader/. Access the directory:

```
cd PaddleX/examples/meter_reader/
```

The prediction execution file is reader_infer.py, and its main parameters are described as follows:

Step 3. Prediction
If the GPU is used, the GPU card number is specified (for example, card 0):

```
export CUDA_VISIBLE_DEVICES=0
```

If the GPU is not used, CUDA_VISIBLE_DEVICES is set to null:

```
export CUDA_VISIBLE_DEVICES=
```

- Prediction of a single picture

```
python reader_infer.py --detector_dir /path/to/det_inference_model --segmenter_dir /path/to/seg_inference_model --image /path/to/meter_test/20190822_168.jpg --save_dir ./output --use_erode
```

- Prediction of multiple pictures

```
python reader_infer.py --detector_dir /path/to/det_inference_model --segmenter_dir /path/to/seg_inference_model --image_dir /path/to/meter_test --save_dir ./output --use_erode
```

- Start the camera for prediction

```
python reader_infer.py --detector_dir /path/to/det_inference_model --segmenter_dir /path/to/seg_inference_model --save_dir ./output --use_erode --use_camera
```

### 18.4 Inference deployment

#### 18.4.1 Server-side security deployment of Windows systems

**c++ deployment**

Step 1. Download PaddleX source code:

```
git clone https://github.com/PaddlePaddle/PaddleX
```

Step 2. Copy the `meter_reader` folder and `CMakeList.txt` from `PaddleX\examples\meter_reader\deploy\cpp` to the `PaddleX\deploy\cpp` directory. Make a backup of the original `CMakeList.txt` in `PaddleX\deploy\cpp` before copying.

Step 3. Compile the C++ prediction code according to Step2 to Step4 in the [Windows platform deployment] (.\deploy\server\cpp\windows.md).

Step 4. After successful compilation, the executable file is in the `out\build\x64-Release` directory. Run `cmd` and switch to that directory.
cd PaddleX\deploy\cpp\out\build\x64-Release

The prediction program is paddle_inference\meter_reader.exe. Its main command parameters are described below.

Step 5. Inference prediction:

The model for the deployment inference should be in inference format. The pre-training models provided in this case are in inference format. For a re-trained model, you need to refer to the Deployment Model Export to export the model to inference format.

• Use unencrypted models to make predictions on a single picture.

```
.
paddle_inference\meter_reader.exe --det_model_dir=path\to\det_inference_model --seg_model_dir=path\to\seg_inference_model --image=path\to\meter_test\20190822_168.jpg --use_gpu=1 --use_erode=1 --save_dir=output
```

• Use unencrypted models to make predictions about image lists. The format of the image_list.txt content is as follows (it is not provided yet due to the different absolute paths, and can be generated by the user as required):

```
\path\to\images\1.jpg
\path\to\images\2.jpg
...
\path\to\images\n.jpg
```

```
.
paddle_inference\meter_reader.exe --det_model_dir=path\to\det_inference_model --seg_model_dir=path\to\seg_inference_model --image_list=path\to\meter_test\image_list.txt --use_gpu=1 --use_erode=1 --save_dir=output
```

• Use unencrypted models to start the camera to make predictions

```
.
paddle_inference\meter_reader.exe --det_model_dir=path\to\det_inference_model --seg_model_dir=path\to\seg_inference_model --use_camera=1 --use_gpu=1 --use_erode=1 --save_dir=output
```

• Prediction of a single image using an encrypted model

If the model is not encrypted, please refer to the [encrypted PaddleX model] to encrypt the model. For example, the directory where the encrypted detection model is located is \path\to\encrypted_det_inference_model, and the key is yEBLDiB0d1lj+5EsNnrABhfDuQKdcreYcHncq4d6bxO=; after encryption, the directory where the segmentation model is located is \path\to\encrypted_seg_inference_model, and the key is DbVS64I9pFRo5XmQ8MN2kSGsfEr4FKAG6O9OOUrRsY=

18.4. Inference deployment
18.4.2 Security deployment of jetson embedded devices for Linux systems

c++ deployment

Step 1. Download PaddleX source code:

```bash
git clone https://github.com/PaddlePaddle/PaddleX
```

Step 2. Copy `meter_reader` folder and `CMakeList.txt` from `PaddleX/examples/meter_reader/deploy/cpp` to `PaddleX/deploy/cpp` directory. You can make a backup of the original `CMakeList.txt` in `PaddleX/deploy/cpp` before copying.

Step 3. Follow Step 2 to Step 3 in the *Deployment of Nvidia Jetson development panel* to compile the C++ prediction codes.

Step 4. After successful compilation, the executable program is `build/meter_reader/meter_reader`. Main command parameters are as follows:

Step 5. Inference prediction:

The model for the deployment inference should be in inference format. The pre-training models provided in this case are in inference format. For a re-trained model, you need to refer to the *Deployment Model Export* to export the model to inference format.

- Use unencrypted models to make predictions on a single picture.

```
./build/meter_reader/meter_reader --det_model_dir=/path/to/det_inference_model --seg_model_dir=/path/to/seg_inference_model --image=/path/to/meter_test/20190822_168.jpg --use_gpu=1 --use_erode=1 --save_dir=output
```

- Use unencrypted models to make predictions about image lists. The format of the image_list.txt content is as follows (it is not provided yet due to the different absolute paths, and can be generated by the user as required):

```
\path/to/images/1.jpg
\path/to/images/2.jpg
...
\path/to/images/n.jpg
```
18.5 Model training

18.5.1 Pre-dependence

- Paddle paddle >= 1.8.0
- Python >= 3.5
- PaddleX >= 1.0.0

For installation related issues, refer to [PaddleX Installation] (../install.md)

18.5.2 Training

- Training of dial detection

```
python /path/to/PaddleX/examples/meter_reader/train_detection.py
```

- Training of pointer and scale segmentation

```
python /path/to/PaddleX/examples/meter_reader/train_segmentation.py
```

Run the above script to train the detection model and segmentation model in this case. If you don’t need the data and model parameters of this case, you can change the data, select the appropriate model and adjust the training parameters.
CHAPTER 19

Portrait segmentation model

This tutorial implements portrait segmentation based on the PaddleX core segmentation model, opens up the pre-training model and test data, supports video streaming portrait segmentation, and provides a full application guide for model Fine-tune to Paddle Lite mobile and Nvidia Jeston embedded device deployment.

19.1 Pre-training models and test data

19.1.1 Pre-training model

This case opens up two models trained on large-scale portrait datasets for both server-end and mobile-end scenarios. The models can be used to quickly experience video stream portrait segmentation, deployed to mobile or embedded devices for real-time portrait segmentation, or used to complete model Fine-tuning.

- Checkpoint Parameter is a model weight for Fine-tuning scenarios, containing `__params__` model parameter and `model.yaml`-based model configuration information.

- Inference Model and Quant Inference Model are prediction deployment models, containing `__model__` computational graph structure, `__params__` model parameter, and `model.yaml`-based model configuration information.

- The Inference Model is for server-end CPU and GPU prediction deployment. The Quant Inference Model is the quantized version for end-end device deployments through Paddle Lite.

The storage size and inference duration of the pre-training model are as follows: The operating environment of mobile model: CPU: Snapdragon 855, RAM: 6GB, image size: 192*192
Execute the following script to download all pre-training models:

- Download PaddleX source code:
  
  ```
git clone https://github.com/PaddlePaddle/PaddleX
  ```

- The codes for downloading the pre-training model is located in `PaddleX/examples/human_segmentation`. Access the directory:
  
  ```
cd PaddleX/examples/human_segmentation
  ```

- Run the download:
  
  ```
python pretrain_weights/download_pretrain_weights.py
  ```

### 19.1.2 Test data

`supervise.ly` has released the **Supervise Persons** dataset for portrait segmentation dataset. In this case, a small portion of the data is randomly extracted and converted into a format that can be loaded directly by PaddleX. Run the following codes to download the data and `video_test.mp4` (portrait test video) shot from the mobile phone front cameras.

- The code to download the test data is located in `PaddleX/examples/human_segmentation`. Access the directory and execute the download:
  
  ```
python data/download_data.py
  ```

### 19.2 Quick experience of video streaming portrait segmentation

#### 19.2.1 pre-dependence

- Paddle paddle >= 1.8.0
- Python >= 3.5
- PaddleX >= 1.0.0

For installation related issues, refer to `[PaddleX Installation]`. (../../docs/install.md)

- Download PaddleX source code:
  
  ```
git clone https://github.com/PaddlePaddle/PaddleX
  ```

- The executable files for both the video stream portrait segmentation and background replacement are located in `PaddleX/examples/human_segmentation`. Access the directory:
19.2.2 Light flow tracking-assisted video streaming portrait segmentation

In this case, the prediction results of the DIS (Dense Inverse Search-based method) light flow tracking algorithm are merged with the segmentation results of PaddleX to improve the effect of the video stream portrait segmentation. Run the following code for experience. The codes are in PaddleX/examples/human_segmentation.

- Real-time segmentation by computer camera

```python
python video_infer.py --model_dir pretrain_weights/humanseg_mobile_inference
```

- Segmentation of offline portrait videos

```python
python video_infer.py --model_dir pretrain_weights/humanseg_mobile_inference --video_path data/video_test.mp4
```

The results of the video segmentation are as follows.

19.2.3 Portrait background replacement

This case also implements the portrait background replacement function, to replace the background image of the portrait according to the selected background. The background can be a picture, or a video. The code for portrait background replacement is located in PaddleX/examples/human_segmentation. Access this directory and run it:

- Replace background in real time through a computer camera. Transmit the background video via ‘-background_video_path’

```python
python bg_replace.py --model_dir pretrain_weights/humanseg_mobile_inference --background_image_path data/background.jpg
```

- Perform background replacement for portrait video. Transmit the background video through ‘-background_video_path’

```python
python bg_replace.py --model_dir pretrain_weights/humanseg_mobile_inference --video_path data/video_test.mp4 --background_image_path data/background.jpg
```

- Background replacement for a single image

```python
python bg_replace.py --model_dir pretrain_weights/humanseg_mobile_inference --image_path data/human_image.jpg --background_image_path data/background.jpg
```
The result of the background replacement is as follows:

**Note:**

- Video segmentation processing takes a few minutes, please be patient.
- The provided model is suitable for vertical screen shooting scene of mobile phone camera, the effect in horizontal screen is slightly poor.

### 19.3 Model Fine-tune

#### 19.3.1 Pre-dependence

- Paddle paddle $\geq 1.8.0$
- Python $\geq 3.5$
- PaddleX $\geq 1.0.0$

For installation related issues, refer to [PaddleX Installation]. (../../docs/install.md)

- Download PaddleX source code:

  ```bash
  git clone https://github.com/PaddlePaddle/PaddleX
  ```

- Execution files for portrait segmentation training, evaluation, prediction, model export, and offline quantification are located in PaddleX/examples/human_segmentation. Access the directory:

  ```bash
  cd PaddleX/examples/human_segmentation
  ```

#### 19.3.2 Model training

Run the following command to perform model training based on the pre-training model. Make sure that the selected model structure `model_type` and model parameter `pretrain_weights` are matched. If you do not need the test data provided in this case, you can replace the data, select a suitable model and adjust the training parameters.

```bash
# Specify the GPU card number (take card 0 as an example)
export CUDA_VISIBLE_DEVICES=0
# Specify CUDA_VISIBLE_DEVICES to be null if the GPU is not used.
# export CUDA_VISIBLE_DEVICES=
python train.py --model_type HumanSegMobile \\
--save_dir output/ \\
--data_dir data/mini_supervisely \\
```

(下页继续)
Meaning of the parameters:

- **--model_type**: model type, options are: HumanSegServer and HumanSegMobile
- **--save_dir**: model save path
- **--data_dir**: data set path
- **--train_list**: training set list path
- **--val_list**: validation set list path
- **--pretrain_weights**: pretraining model path
- **--batch_size**: batch size
- **--learning_rate**: initial learning rate
- **--num_epochs**: number of training rounds
- **--image_shape**: network input image size (w, h)

For more help of command lines, run the following command:

```bash
go train.py -help
```

**Note**: You can use different models for quick try by replacing **--model_type** variable and the corresponding **--pretrain_weights**.

### 19.3.3 Evaluate

Evaluate the model precision on the validation set by running the following commands:

```bash
go eval.py --model_dir output/best_model \
--data_dir data/mini_supervisely \
--val_list data/mini_supervisely/val.txt \
--image_shape 192 192
```

Meaning of the parameters:

- **--model_dir**: model path
• --data_dir: data set path
• --val_list: validation set list path
• --image_shape: network input image size (w, h)

19.3.4 Prediction

Use the following command to predict the test set. By default, the prediction visualization results are saved in the /output/result/ folder.

```python
python infer.py --model_dir output/best_model \ 
--data_dir data/mini_supervisely \ 
--test_list data/mini_supervisely/test.txt \ 
--save_dir output/result \ 
--image_shape 192 192
```

Meaning of the parameters:

• --model_dir: model path
• --data_dir: data set path
• --test_list: test set list path
• --image_shape: network input image size (w, h)

19.3.5 Model export

The model deployed on the server needs to be exported to an inference format model first. The exported model consists of three file names, __model__.txt, __params__.txt, and model.yml, which are respectively model network structure, model weights and the model configuration file (including data preprocessing parameters). After installing PaddleX, use the following command at the command line terminal to export the model:

```python
paddlex --export_inference --model_dir output/best_model \ 
--save_dir output/export
```

Meaning of the parameters:

• --model_dir: model path
• --save_dir: storage path of exported models
### 19.3.6 Offline quantification

```bash
python quant_offline.py --model_dir output/best_model \
--data_dir data/mini_supervisely \
--quant_list data/mini_supervisely/val.txt \
--save_dir output/quant_offline \
--image_shape 192 192
```

Meaning of the parameters:

- **--model_dir**: path of models to be quantified
- **--data_dir**: data set path
- **--quant_list**: path of quantification dataset list. Generally, it is selected as training set or validation set.
- **--save_dir**: storage path of quantification model
- **--image_shape**: network input image size (w, h)

### 19.4 Inference deployment

#### 19.4.1 Paddle Lite mobile-end deployment

This case deploys the portrait segmentation model on the mobile end. The deployment process is as follows. See [Paddle Lite Mobile Deployment](https://bj.bcebos.com/paddlex/deploy/lite/model_optimize_tool_11cbd50e.tar.gz) for the general mobile deployment process.

1. **Export the PaddleX model to an inference model**

In this case, we use the humanseg_mobile_quant pre-training model, which is already an inference model, so we don’t need to perform the model export step. If the pre-training model is not used, execute the `model export` in the previous `Model Training` to export your own trained model to the inference format.

2. **Optimize the inference model to a Paddle Lite model**

Download and decompress the [Model Optimizer opt](https://bj.bcebos.com/paddlex/deploy/lite/model_optimize_tool_11cbd50e.tar.gz). Go to the path where the [Model Optimizer opt] is located, and execute the following command:

```bash
./opt --model_file=<model_path> \ 
--param_file=<param_path> \ 
--valid_targets=arm \ 
```

(下页继续)
For more detailed usage and parameter meanings, refer to: [Using the opt transformation model](https://paddle-lite.readthedocs.io/zh/latest/user_guides/opt/opt_bin.html)

3. Mobile end prediction

PaddleX provides an Android demo based on the PaddleX Android SDK for users to experience image class, object detection, instance segmentation and semantic segmentation, and the demo is located at PaddleX/deploy/lite/android/demo. Users can copy models, configuration files and test images to the demo for prediction.

3.1 Pre-dependencies

- Android Studio 3.4
- Android phone or development panel

3.2 Copying models, configuration files and test images

- Copy the Lite model (.nb file) to PaddleX/deploy/lite/android/demo/app/src/main/assets/model/ directory, and modify MODEL_PATH_DEFAULT in the PaddleX/deploy/lite/android/demo/app/src/main/res/values/strings.xml according to the name of the .nb file.
- Copy the configuration file (.yaml file) to the PaddleX/deploy/lite/android/demo/app/src/main/assets/config/ directory, and modify YAML_PATH_DEFAULT in the file PaddleX/deploy/lite/android/demo/app/src/main/res/values/strings.xml according to the name of the .yaml file.
- Copy the test image to the PaddleX/deploy/lite/android/demo/app/src/main/assets/images/ directory, and modify IMAGE_PATH_DEFAULT in the file PaddleX/deploy/lite/android/demo/app/src/main/res/values/strings.xml according to the name of the image file.

3.3 Import the project and run

- Start Android Studio. Click “Open an existing Android Studio project” in the “Welcome to Android Studio” window. Access the PaddleX/deploy/lite/android/demo directory in the pop-up path selection window. Click the “Open” button in the bottom right corner to import the project.
- Connect an Android phone or development panel through a USB port.
- After the project is compiled, choose Run-> Run ‘App’ button on the menu bar. Select the connected Android device in the pop-up “Select Deployment Target” window, and then click the “OK” button.
• After successful running, the Android device loads a PaddleX Demo App. By default, a test image is loaded. It also supports the prediction by taking photos and selecting photos from the gallery.

The test image and its segmentation results are as follows:

19.4.2 Nvidia Jetson embedded device deployment

c++ deployment

Step 1. Download the PaddleX source code

```
git clone https://github.com/PaddlePaddle/PaddleX
```

Step 2. Copy the `human_segmenter.cpp` and `CMakeList.txt` from PaddleX/examples/
human_segmentation/deploy/cpp to PaddleX/deploy/cpp directory. You can make a backup of the original CMakeList.txt in PaddleX/deploy/cpp before copying.

Step 3. Follow Step2 to Step3 in the [Deployment of Nvidia Jetson Development Panel] (../deploy/nvidia-jetson.md) to compile the C++ prediction codes.

Step 4. After the compilation is successful, run the executable program build/human_segmenter. The main command parameters are described as follows:

Step 5. Inference prediction

The model used to deploy inference should be in the inference format. In this case, use the humanseg_server_inference pre-training model, which is already an inference model. It is not necessary to perform the model export step. If you do not use a pre-training model, perform the model export in Chapter 2 Model Training to export your own training model as the inference format.

- Use unencrypted models to make predictions on a single picture.

The image to be tested is located in the test data provided in this case and can be replaced with its own image.

```
./build/human_segmenter --model_dir=/path/to/humanseg_server_inference --image=/path/to/
data/mini_supervisely/Images/pexels-photo-63776.png --use_gpu=1 --save_dir=output
```

- Use unencrypted models to start the camera to make predictions

```
./build/human_segmenter --model_dir=/path/to/humanseg_server_inference --use_camera=1 --save_result=1 --use_gpu=1 --save_dir=output
```

- Use an unencrypted model to make predictions on video files

The video files to be tested are located in the test data provided in this case and can be replaced with their own video file.

```
./build/human_segmenter --model_dir=/path/to/humanseg_server_inference --video_path=/
/path/to/data/mini_supervisely/video_test.mp4 --save_result=1 --use_gpu=1 --save_
_dir=output
```
RGB remote sensing image segmentation

This case is the implementation of remote sensing image segmentation based on PaddleX, and provides a prediction method in a sliding window to avoid the occurrence of insufficient display memory in the direct prediction of large-size images. In addition, the degree of overlapping between the sliding windows can be configured. This can eliminate cracks in the final prediction results at the window splices.

20.1 Pre-dependence

- Paddle paddle >= 1.8.4
- Python >= 3.5
- PaddleX >= 1.1.4

For installation related issues, refer to [PaddleX Installation]. (../install.md)

Download PaddleX source code:

```
git clone https://github.com/PaddlePaddle/PaddleX
```

All scripts for the case are located in PaddleX/examples/remote_sensing/. Access the directory:

```
cd PaddleX/examples/remote_sensing/
```
20.2 Data preparation

In this case, the high-definition remote sensing images provided by the 2015 CCF Big Data competition is used, containing five RGB images with labels with a maximum image size of 7969 × 7939 and a minimum size of 4011 × 2470. The dataset is labeled with a total of 5 categories of objects: background (labeled 0), vegetation (labeled 1), buildings (labeled 2), bodies of water (labeled 3), and roads (labeled 4).

In this case, the first four images are categorized into the training set and the fifth image is used as the validation set. In order to increase the batch size during training, the first four images are cut with sliding window (1024, 1024) and step (512, 512). In addition to the original four large images, there are 688 images in the training set in total. In order to avoid the occurrence of insufficient display memory in the validation of large-size images during training, the 5th image is cut in the sliding window (769, 769) and step (769, 769) for the validation set to obtain 40 sub-images.

Run the following script to download the original dataset and complete the cut of the dataset.

```python
python prepare_data.py
```

20.3 Model training

For the split model, the Deeplabv3 model whose Backbone is set to MobileNetv3_large_ssld and the model features high performance and high precision. Run the following script to carry out the model training:

```python
python train.py
```

You can also skip the model training step and directly download the pre-trained model for subsequent model prediction and evaluation.

```bash
wget https://bj.bcebos.com/paddlex/examples/remote_sensing/models/ccf_remote_model.tar.gz
tar -xvf ccf_remote_model.tar.gz
```

20.4 Model predictions

The direct prediction of large size images can lead to insufficient video memory. In order to avoid such a problem, this case provides a sliding window prediction interface, which supports both overlapping and non-overlapping methods.

- Non-overlapping sliding window prediction

The images under each window are predicted separately by sliding the windows at a fixed size on input images. Finally, the prediction results of each window are stitched together into the prediction result of the
input images. Since the prediction effect of the edge part of each window is worse than that of the middle part, there may be obvious cracks in each window splice.

For the API of the prediction method, see [overlap_tile_predict](https://paddlex.readthedocs.io/zh_CN/develop/apis/models/semantic_segmentation.html#overlap-tile-predict). The parameter `pad_size` must be set to `[0, 0]` during use.

- Overlapping sliding window prediction

In the Unet paper, the author proposes an Overlap-tile prediction strategy with overlapping sliding window to eliminate the cracking sensation at the splice. In the prediction in each sliding window, a certain area is expanded around the expanded window, such as the blue part of the area in the figure below. Only the middle part of the window is predicted in the splice, for example, the yellow part area in the figure below. The pixels under the expanded area of the window located at the edge of the input image are obtained by mirroring the pixels at the edge.

The API for this prediction method is described in `overlap_tile_predict`.

Compared to the non-overlapping sliding window prediction, the overlapping sliding window prediction strategy improves the model precision from 80.58% to 81.52%, and eliminates the cracking sensation in the prediction visualization. See the effect comparison of the two prediction methods.
Run the following script for prediction by using overlapping sliding windows:

```
python predict.py
```

### 20.5 Model evaluation

During the training process, the precision of the model in the validation set is evaluated every 10 iteration rounds. The original large size image is cut into small slices in advance. Therefore, the prediction method by using the overlapping big image slicing small image is used. The optimal model precision mIoU is 80.58%. Run the following script to re-evaluate the model precision of the original large size image in the prediction method by using the overlapping big image slicing small image. At this time, mIoU is 81.52%.

```
python eval.py
```
CHAPTER 21

Multi-channel remote sensing image segmentation

Remote sensing image segmentation is an important application scene in the field of image segmentation, and is widely used in land surveying and mapping, environmental monitoring, urban construction and other fields. The targets of remote sensing image segmentation are diversified, such as snow, crops, roads, buildings, water sources and other features, as well as air targets (for example, cloud).

This case implements multi-channel remote sensing image segmentation based on PaddleX, covering data analysis, model training, model prediction and other processes, and aims to help users solve the multi-channel remote sensing image segmentation problem by using the deep learning technology.

21.1 Pre-dependence

- Paddle paddle $\geq 1.8.4$
- Python $\geq 3.5$
- PaddleX $\geq 1.1.4$

For installation related issues, refer to [PaddleX Installation]. (../../install.md)

**In addition, you need to install gdal.** There may be an error in the installation of gdal by using pip. It is recommended to use conda to install gdal.

```
conda install gdal
```

Download PaddleX source code:
git clone https://github.com/PaddlePaddle/PaddleX

All scripts for the case are located in PaddleX/examples/channel_remote_sensing/. Access the directory.

cd PaddleX/examples/channel_remote_sensing/

# 21.2 Data preparation

Remote sensing images are available in a variety of formats, and the formats of the data produced by different sensors may vary. PaddleX is now compatible with the following four formats for image reading:

- tif
- png
- img
- npy

The annotation map must be a single-channel image in png format, the pixel value is the corresponding category, and the pixel annotation category needs to be incremented from 0. For example, 0, 1, 2, and 3 indicate that there are 4 categories, and 255 is used to specify pixels that are not involved in training and evaluation, and the maximum number of label categories is 256.

This case study uses the L8 SPARCS public dataset for segmentation of cloud and snow. The dataset contains 80 satellite images, covering 10 bands. The original annotated images contain 7 classes, that is, cloud, cloud shadow, shadow over water, snow/ice, water, land, and flooded. Since flooded and shadow over water account for only 1.8% and 0.24%, flooded is merged into land and shadow over water is merged into shadow. After the merge, there are five classes in total.

The corresponding table of value, class, and color:
Run the following command to download and decompress the class-merged dataset.

```
mkdir dataset && cd dataset
wget https://paddleseg.bj.bcebos.com/dataset/remote_sensing_seg.zip
unzip remote_sensing_seg.zip
cd ..
```

The `data` directory stores remote sensing images, the `data_vis` directory stores color composite preview images, and the `mask` directory stores labeled images.

### 21.3 Data analysis

Remote sensing images are often composed of many wavelengths, and the distribution of data in different wavelengths may vary greatly, for example, the visible and thermal infrared wavelengths are differently distributed. In order to better understand the distribution of the data to optimize the training effect of the model, the data needs to be analyzed.

Statistical analysis is performed on the training set with reference to the [document data], to determine the truncation range of image pixels, and to make statistics of the mean and variance of the truncated data. (./analysis.md)
21.4 Model training

In this case, the UNet semantic segmentation model is selected to complete the segmentation of clouds and snow. Perform the following steps to complete the model training. The optimal precision of the model \( \text{miou} \) is 78.38%.

- Set the GPU card number.

```bash
export CUDA_VISIBLE_DEVICES=0
```

- Run the following script to start training:

```bash
python train.py --data_dir dataset/remote_sensing_seg \
--train_file_list dataset/remote_sensing_seg/train.txt \
--eval_file_list dataset/remote_sensing_seg/val.txt \
--label_list dataset/remote_sensing_seg/labels.txt \
--save_dir saved_model/remote_sensing_unet \
--num_classes 5 \
--channel 10 \
--lr 0.01 \
--clip_min_value 7172 6561 5777 5103 4291 4000 4000 4232 6934 7199 \ 
--clip_max_value 50000 50000 50000 50000 50000 40000 30000 18000 40000 36000 \ 
--mean 0.15163569 0.15142828 0.15574491 0.1716084 0.2799778 0.27652043 0.28195933 0.\!07853807 0.56333154 0.5477584 \ 
--std 0.09301891 0.09818967 0.09831126 0.1057784 0.2799778 0.27652043 0.28195933 0.\!02637859 0.0675052 0.06168227 \ 
--num_epochs 500 \ 
--train_batch_size 3
```

It is also possible to skip the model training step and download pre-training models for direct model prediction.

```bash
wget https://bj.bcebos.com/paddlex/examples/multi-channel_remote_sensing/models/l8sparcs_\!remote_model.tar.gz
```

```
tar -xvf l8sparcs_remote_model.tar.gz
```

21.5 Model predictions

Run the following script to predict the remote sensing image and visualize the prediction result, and also visualize the corresponding label file to compare the prediction effect.
export CUDA_VISIBLE_DEVICES=0
python predict.py

The visualization effect is as follows:

The corresponding table of value, class, and color:
CHAPTER 22

Plot change detection

This case implements the plot change detection based on PaddleX, that is, two images of the same plot in the early stage and late stage are stitched together and then input to the semantic segmentation network for predicting the change of the area. In the training phase, a variety of data enhancement strategies such as random scaling size, rotation, pruning, color space perturbation, horizontal flipping, and vertical flipping are used. In the validation and prediction phases, sliding window prediction is used to avoid insufficiency of display memory when large-size images are directly predicted.

22.1 pre-dependence

- Paddle paddle >= 1.8.4
- Python >= 3.5
- PaddleX >= 1.2.2

For installation related issues, refer to [PaddleX Installation]. (../install.md)

Download PaddleX source code:

```
git clone https://github.com/PaddlePaddle/PaddleX
```

All scripts of this case are located in PaddleX/examples/change_detection/. Access the directory:

```
cd PaddleX/examples/change_detection/
```
22.2 Data preparation

This case uses the [Google Dataset] (https://github.com/daifeng2016/Change-Detection-Dataset-for-High-Resolution-Satellite-Imagery) developed by Daifeng Peng, etc. The dataset covers changes in houses and buildings in partial areas in Guangzhou from 2006 to 2019 for analyzing the urbanization process. There are 20 pairs of high-resolution images in red, green and blue bands with a spatial resolution of 0.55m, and image sizes range from 1006x1168 to 4936x5224.

Google Dataset only marks whether the houses and buildings are changed; therefore, this case is a two-class change detection task, which can be extended to the multi-class change detection by modifying the number of classes according to actual needs.

In this case, 15 images are categorized into the training set and 5 images are categorized into the validation set. Since the size of the images is too large, direct training may cause the problem of insufficient display memory, the training images are divided according to the sliding window (1024, 1024) and step (512, 512). There are 743 pictures in the training set after the division. Divide the validation pictures based on a sliding window of (769, 769) and a step of (769, 769) to get 108 sub-pictures for validation during training.

Run the following script to download the original dataset and complete the cut of the dataset.

```python
python prepare_data.py
```

Data after division is as follows:
Note:

- PaddleX uses the gdal library to read tiff images. For the installation of gdal, see the reference document. For RGB pictures in tiff, if you don’t want to install gdal, you need to convert pictures to formats such as jpeg, bmp, or png.

- The label file should be a single-channel image in png format, and the labels start counting from 0, the label 255 means the category is not involved in the calculation. For example, in this case, 0 means unchanged class, 1 means changed class.

### 22.3 Model training

Due to the small amount of data, choose the UNet model for the segmentation model, because it features both shallow details and deep semantic information. Run the following script to carry out the model training:

```
python train.py
```

You can change the number of GPU cards and setting value of `train_batch_size` in the training script according to the actual video memory size, and then adjust the `learning_rate` according to the adjustment ratio of `train_batch_size`. For example, when `train_batch_size` decreases from 16 to 8, `learning_rate` decreases from 0.1 to 0.05.
PaddleX decreases from 0.1 to 0.05. In addition, the learning rate corresponding to the optimal precision obtained on different datasets may vary. You can try to adjust it.

It is also possible to skip the model training step and directly download the pre-training model for subsequent model evaluation and prediction.

```bash
wget https://bj.bcebos.com/paddlex/examples/change_detection/models/google_change_det_-model.tar.gz tar -xvf google_change_detection_model.tar.gz
```

## 22.4 Model evaluation

During the training process, the precision of the model in the validation set is evaluated every 10 iteration rounds. The original large size images are sliced into smaller blocks in advance. Therefore, it means that a non-overlapping sliding-window prediction method is used. The optimal model precision:

The category corresponds to unchanged and changed respectively.

Run the following script to re-evaluate the model precision of the original large size image by using overlapping sliding window predictions. The model precision at that time is:

```bash
python eval.py
```

For the sliding window prediction interface, see API description. For the existing usage scenarios, refer to RGB remote sensing segmentation Case. The tile_size, pad_size and batch_size of the evaluation script can be modified according to the actual memory size.

## 22.5 Model predictions

Run the following script to predict the validation set by using an overlapping sliding prediction window. The tile_size, pad_size and batch_size of the evaluation script can be modified according to the actual memory size.

```bash
python predict.py
```

The result of the prediction visualization is shown in the following figure:
CHAPTER 23

Industrial Quality Inspection

23.1 1. GPU solutions

23.1.1 1.1 Dataset introduction

23.1.2 1.2 Precision optimization

23.1.3 1.3 Performance optimization
(1) Reduce the number of channels in the FPN section

将 FPN 部分的通道数量由原本的 256 减少至 64，使用方式在定义模型 FasterRCNN 类时设置参数 fpn_num_channels 为 64 即可，需要重新对模型进行训练。

(2) Reduce the number of candidate frames in the test phase

将测试阶段 RPN 部分非极大值抑制计算的候选框数量由原本的 6000 减少至 500，将 RPN 部分非极大值抑制后保留的候选框数量由原本的 1000 减少至 300。使用方式在定义模型 FasterRCNN 类时设置参数 test_pre_nms_top_n 为 500, test_post_nms_top_n 为 300。

采用 Fluid C++ 预测引擎在 Tesla P40 上测试模型的推理时间（输入数据拷贝至 GPU 的时间，计算时间，数据拷贝至 CPU 的时间），输入大小设置为 800x1333，加速前后推理时间如下表所示：

23.1.4 1.4 Final plan

本案例面向 GPU 端的最终方案是选择二阶段检测模型 FasterRCNN，其骨干网络选择加入了可变形卷积（DCN）的 ResNet50_vd，训练时使用 SSLD 蒸馏方案训练得到的 ResNet50_vd 预训练模型，FPN 部分的通道数量设置为 64，使用复核过的数据集，训练阶段数据增强策略采用 RandomHorizontalFlip, RandomDistort, RandomCrop，并加入背景图片。测试阶段的 RPN 部分非极大值抑制计算的候选框数量由原本的 6000 减少至 500，做完非极大值抑制后保留的候选框数量由原本的 1000 减少至 300。模型在验证集上的 VOC mAP 为 87.72%。

在 Tesla P40 的 Linux 系统下，对于输入大小是 800 x 1333 的模型，图像预处理时长为 30ms/image，模型的推理时间为 46.08ms/image，包括输入数据拷贝至 GPU 的时间，计算时间，数据拷贝至 CPU 的时间。

具体的训练和部署流程点击文档 GPU 端最终解决方案进行查看。

23.2 2. CPU Solution

为了实现高效的模型推理，面向 CPU 端的模型选择精度和效率皆优的单阶段检测模型 YOLOv3，骨干网络选择基于 PaddleClas 中 SSLD 蒸馏方案训练得到的 MobileNetv3_large。训练完成后，对模型做剪裁操作，以提升模型的性能。模型在验证集上的 VOC mAP 为 79.02%。

部署阶段，借助 OpenVINO 预测引擎完成在 Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz Windows 系统下高效推理。对于输入大小是 608 x 608 的模型，图像预处理时长为 38.69 ms/image，模型的推理时间为 34.50ms/image。

23.2.1 Model training

环境前置依赖、下载 PaddleX 源码、下载数据集与 GPU 端是一样的，可点击文档 GPU 端最终解决方案查看，在此不做赘述。

如果不想再次训练模型，可以直接下载已经训练好的模型完成后面的模型测试和部署推理。
wget https://bj.bcebos.com/paddlex/examples/industrial_quality_inspection/models/yolov3_
  --mobilenetv3_large_pruned.tar.gz
tar xvf yolov3_mobilenetv3_large_pruned.tar.gz

运行以下代码进行模型训练，代码会自动下载数据集，如若事先下载了数据集，需将下载和解压铝材缺陷检测数据集的相关行注释掉。代码中默认使用 0,1,2,3,4,5,6,7 号 GPU 训练，可根据实际情况设置卡号并调整 batch_size 和 learning_rate。

```python
python train_yolov3.py
```

### 23.2.2 Model clipping

运行以下代码，分析在不同的精度损失下模型各层的剪裁比例：

```python
python params_analysis.py
```

设置可允许的精度损失为 0.05，对模型进行剪裁，剪裁后需要重新训练模型：

```python
python train_pruned_yolov3.py
```

分析预测错误的原因，统计图片级召回率和误检率，模型测试这些步骤与 GPU 端是一样的，可点击文档 GPU 端最终解决方案查看。在此不做赘述。

### 23.2.3 Reasoning deployment

本案例采用 C++ 部署方式，通过 OpenVINO 将模型部署在 Intel(R) Core(TM) i9-9820X CPU @ 3.30GHz 的 Windows 系统下，具体的部署流程请参考文档 PaddleX 模型多端安全部署/OpenVINO 部署。
PaddleX 可视化客户端基于 PaddleX 开发的可视化深度学习模型训练套件，目前支持训练视觉领域的图像分类、目标检测、实例分割和语义分割四大任务，同时支持模型裁剪、模型量化两种方式压缩模型。开发者以点选、键入的方式快速体验深度学习模型开发的全流程。可以作为您提升深度学习模型开发效率的工具。

PaddleX GUI 当前提供 Windows，Mac，Ubuntu 三种版本一键绿色安装的方式。请至飞桨官网：https://www.paddlepaddle.org.cn/paddle/paddleX 下载您需要的版本。

### 24.1 Functions

PaddleX 可视化客户端是 PaddleX API 的衍生品，它在集成 API 功能的基础上，额外提供了可视化分析、评估等附加功能，致力于为开发者带来极致顺畅的开发体验。其拥有以下独特的功能：

#### 24.1.1 Get through the whole process

PaddleX GUI 覆盖深度学习模型开发必经的 数据处理、超参数配置、模型训练及优化、模型发布全流程，无需开发一行代码，即可得到高性深度学习推理模型。

#### 24.1.2 Intelligent analysis of datasets

详细的数据结构说明，并提供 数据标签自动校验。支持 可视化数据预览、数据分布图表展示、一键数据集切分等实用功能
24.1.3 Auto reference recommendation

集成飞桨团队长时期产业实践经验，根据用户选择的模型类别、骨架网络等，提供多种针对性优化的 预训练模型，并提供推荐超参数配置，可一键开启多种优化策略。

24.1.4 Visual model evaluation

集成可视化分析工具：VisualDL，以线性图表的形式展示 acc、lr 等关键参数在训练过程中的变化趋势。提供混淆矩阵等实用方法，帮助快速定位问题，加速调参。模型评估报告一键导出，方便项目复盘分析。

24.1.5 Model tailoring and quantification

一键启动模型裁剪、量化，在不同阶段为开发者提供模型优化的策略，满足不同环境对模型性能的需求。

24.1.6 Pre-training model management

可对历史训练模型进行保存及管理，未进行裁剪的模型可以保存为预训练模型，在后续任务中使用。

24.1.7 Visual model testing

客户端直接展示模型预测效果，无需上线即可进行效果评估。

24.1.8 Model multiterminal deployment

点选式选择模型发布平台、格式，一键导出预测模型，并匹配完善的模型预测部署说明文档，贴心助力产业端到端项目落地。
PaddleX GUI is a core module to improve project development efficiency. Developers can quickly complete the whole development of deep learning models. We sincerely invite you to download and try out the PaddleX GUI visualization front-end at our [official website] (https://www.paddlepaddle.org.cn/paddle/paddleX), and hope to get your valuable comments or contribution to the open source project.

### 25.1 Recommended Installation Environment

- **Operating System**
  - Windows7/8/10 (Windows 10 recommended);
  - Mac OS 10.13+;
  - Ubuntu 18.04+;

  ***Note: The processor must use an x86_64 architecture and support MKL.***

- **Training Hardware:**
  - **GPU** (Only the Windows and Linux systems): NVIDIA GPUs which support the CUDA, such as GTX 1070+ or better GPUs, are recommended; Windows system X86_64 drive version >=411.31; Linux system X86_64 drive version >=410.48; 8G or more GPU display memory
  - **CPU:** Currently, PaddleX allows you to train with a local CPU, but a GPU is recommended for a better development experience.
  - **Memory:** 8G or more is recommended
– **Hard disk space**: 1T or more SSD remaining space is recommended (not mandatory)

***Note: PaddleX only supports single-card models in the Mac OS system. The Windows system currently does not support NCCL.***
PaddleX GUI tutorial

Notes: If your system is Mac OS 10.15.5 and later, after double-clicking the client icon, you need to execute 
sudo xattr -r -d com.apple. quarantine/Users/username/PaddleX in Terminal and wait a few seconds to 
start the client, where /Users/username/PaddleX is a folder path where you save PaddleX.

**Step 1: Prepare data**

Before starting model training, you need to annotate data in the corresponding format according to different 
task types. Currently, PaddleX supports [image classification], [object detection], [semantic segmentation] 
and [instance segmentation] task types. For the data processing methods of different types of tasks, view 

**Step 2: Import a dataset**

After the data annotation is complete, you need to rename data and annotation documents according to 
different tasks and save them in the correct file.

Create a dataset on the client, select a task type that matches data as well as a path corresponding to the 
dataset, and import the dataset.
After an imported dataset is selected, the client automatically checks whether data and annotation documents are compliant. After the check is successful, you can proportionally divide datasets into training sets, validation sets and test sets according to actual requirements.

You can preview your annotated dataset in the [Data Analysis] module according to the rules. Double-click a single image to zoom in on it.
Step 3: Create a project

After the data import is complete, you can click [New Project] to create a project.

You can select a task type for the project according to actual task requirements. Note that the dataset used also has a task type attribute. Both of them need to match each other.
Step 4: Project development

Data selection: After the project creation is complete, you need to select a dataset which has been loaded into the client and checked. Click Next to enter the parameter configuration page.

Parameter configuration: Mainly divided into three parts including model parameters, training parameters and optimization policies. You can select a model structure, a backbone network and the corresponding training parameters and optimization policies according to actual requirements to optimize the task effects.
After the parameter configuration is complete, click Start Training to start training the model and perform effect evaluation.

**Training visualization:** You can view any parameter change, log details, and the current optimal training indexes of training and validation sets through VisualDL during training. You can suspend the model training process at any time by clicking “Suspend Training” during training.
After the model training is complete, you can choose to enter [Model Pruning Analysis] or directly enter [Model Evaluation].

**Model pruning**: If you want to reduce the model size and computation to improve the model inference performance on the device, you can use the model pruning policies provided by PaddleX. The pruning process is to analyze the sensitivity information of convolutional layers of a model, perform pruning in
different proportions according to the impact of parameters on the model effects, and perform fine-tuned training to obtain the final pruned model.

Model evaluation: You can view the trained model effects in the model evaluation page. Evaluation methods include confusion matrix, precision and recall rate.
You can also select a [Test Dataset] reserved during [Dataset Splitting] or import one or more images from a local folder to test a trained model. Based on the test results, you can decide whether to save the trained model as a pre-training model and enter the model release page, or go back to the previous step to adjust the parameter configuration and perform re-training.

### Step 5: Model release

After you are satisfied with the model effects, you can choose to release the model as a required version according to actual production environment requirements.
If you have any questions or suggestions, do not hesitate to provide your feedback in the form of an issue or join PaddleX’s official QQ group (1045148026)

1. Why is the training speed so slow?

PaddleX makes a calculation using your local hardware entirely and deep learning tasks have a high requirement for computing power. We have adapted the CPU hardware so that you can quickly experience development with PaddleX, but we strongly recommend using a GPU to improve the training
2. Can I deploy PaddleX on a server or cloud platform?

PaddleX GUI is a client that adapts to local stand-alone installation and cannot be directly deployed on the server. You can directly use a PaddleX API or the PaddlePaddle core framework for its deployment on the server. If you wish to use public computing power, it is strongly recommended that you try to use EasyDL or AI Studio in the PaddlePaddle product series for development.

3. Does PaddleX support EasyData annotated data?

Yes, it does. PaddleX can read EasyData annotated data smoothly. However, the current version of PaddleX GUI does not support direct import of the EasyData data format. By referring to the related document, you can convert datasets and import them to PaddleX GUI for subsequent development. In addition, we are working on the function that PaddleX GUI can directly import the EasyData data format.

4. Why is the model pruning analysis so time-consuming?

The model pruning analysis process is to analyze the sensitivity information of convolutional layers of a model and then perform pruning in different proportions according to the impact of parameters on the model effects. This process needs to be repeated several times until the FLOPS meet the requirements. Finally, fine-tuned training is performed to obtain the final pruned model. Therefore, it is time-consuming. For the principle of model pruning, refer to the document Pruning Principle Introduction.

5. How to call backend codes?

The PaddleX team has compiled the related API document for your learning and use. For details, refer to the PaddleX API Description Document.

6. How to use PaddleX in an offline environment?

PaddleX allows users to train models in a local offline environment, but if you want to use the pre-training models trained on a standard dataset that the PaddleX team has prepared for you, you need to download them in an online environment. You can refer to the complete document on training models without any networking and see how to quickly you can download all pre-training models with one click.

7. Do you have any industry application cases or implemented engineering instances?

Yes, we have. PaddleX offers a wealth of industry application cases and complete example projects. Refer to the [PaddleX Industry Casebook](https://paddlex.readthedocs.io/zh_CN/develop/examples/index.html)

If you have any questions or suggestions, do not hesitate to provide your feedback in the form of an issue or join PaddleX’s official QQ group (1045148026)
飞桨 PaddleX交流群
群号：1045148026

扫一扫二维码，加入群聊。
27.1 Data Processing and Enhancement

Transforms provides data preprocessing and data enhancement interface for PaddleX model training.

27.1.1 paddlex.cls.transforms

This section describes the operations for image classification tasks. The *Compose* class can be used to combine image preprocessing/augmenter operations.

**Compose**

```
paddlex.cls.transforms.Compose(transforms)
```

The input data is operated by the data preprocessing/augmenter operator. **Usage Example**

**Parameters**

- `transforms` (list): Data preprocessing/data augmenter list.

**Normalize**

```
paddlex.cls.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
```
Standardize the image.

1. Images are normalized to range \([0, 1.0]\).
2. The image is subtracted from the mean and divided by the standard deviation.

**Parameters**

- **mean** (list): The mean value of the image data set. Default values are 0.485, 0.456, and 0.406.
- **std** (list): Standard deviation of the image data set. Default values are 0.229, 0.224, 0.225.

### ResizeByShort

```python
paddlex.cls.transforms.ResizeByShort(short_size=256, max_size=-1)
```

Resizes the image according to the short edge of the image.

1. Get the length of the long and short edges of the image.
2. According to the ratio of short side and short_size, calculate the target length of the long side. At this time, the resize ratio of height and width is `short_size/`original short side length`
3. If `max_size`>0, adjust the resize ratio: if the target length of the long side is `max_size`, the resize ratio of height and width is `max_size/the length of the long edge of the original image`
4. Resize the image according to the resize ratio.

**Parameters**

- **short_size** (int): The target length of the short side of the resized image. The default value is 256.
- **max_size** (int): Maximal limit of the length of the long side target. The default value is -1.

### CenterCrop

```python
paddlex.cls.transforms.CenterCrop(crop_size=224)
```

Diffusely prune a square with `crop_size` at the center of the image.

1. Calculates the start point of the pruning.
2. Prune the image.
Parameters

- **crop_size** (int): The length of the target edge to be pruned. The default value is 224.

RandomCrop

```
paddlex.cls.transforms.RandomCrop(crop_size=224, lower_scale=0.08, lower_ratio=3./4, upper_ratio=4./3)
```

Random pruning of images, data augmenter operations for the model training.

1. Calculate the height and width of random pruning according to lower_scale, lower_ratio and upper_ratio.
2. Pick the starting point of the random pruning according to the height and width of the random pruning.
3. Prune the image.
4. Resize the pruning image to crop_size*crop_size.

Parameters

- **crop_size** (int): The length of the target edge to be resized after random cropping. The default value is 224.
- **lower_scale** (float): Minimum limit of the ratio of the crop area to the original area. The default value is 0.08.
- **lower_ratio** (float): The minimum limit of the width change scale. The default value is 3./4.
- **upper_ratio** (float): Minimum limit for the width change ratio. The default value is 4./3.

RandomHorizontalFlip

```
paddlex.cls.transforms.RandomHorizontalFlip(prob=0.5)
```

Flip the image horizontally at random with a certain probability. It is the data augmenter operation during model training.

Parameters

- **prob** (float): The probability of a random level flip. The default value is 0.5.
RandomVerticalFlip

```python
paddlex.cls.transforms. RandomVerticalFlip(prob=0.5)
```

Vertically flip the image at random with a certain probability. It is the data augmenter operation in the model training.

**Parameters**

- `prob` (float): probability of a random vertical flip. The default value is 0.5.

RandomRotate

```python
paddlex.cls.transforms. RandomRotate(rotate_range=30, prob=0.5)
```

Rotate the image with probability in [-rotate_range, rotaterange] angle range. It is the data augmenter operation in the model training.

**Parameters**

- `rotate_range` (int): The range of the rotation degree. The default value is 30.
- `prob` (float): The probability of random rotation. The default value is 0.5.

RandomDistort

```python
paddlex.cls.transforms. RandomDistort(brightness_range=0.9, brightness_prob=0.5,
contrast_range=0.9, contrast_prob=0.5, saturation_range=0.9, saturation_prob=0.5, hue_
range=18, hue_prob=0.5)
```

Random pixel content transformation of the image with a certain probability. It is the data augmenter operation in the model training.

1. Randomize the operation order of the transformations.
2. Perform a random pixel content transformation in the range[-range, range] with a certain probability on the image in the order shown in Step 1.

[Note] This data augmenter must be used before the Normalize.
Parameters

- **brightness_range** (float): the range of the brightness factor. The default value is 0.9.
- **brightness_prob** (float): The probability that the brightness is adjusted randomly. The default value is 0.5.
- **contrast_range** (float): The range of the contrast factor. The default value is 0.9.
- **contrast_prob** (float): The probability that the contrast is adjusted randomly. The default value is 0.5.
- **saturation_range** (float): The range of the saturation factor. The default value is 0.9.
- **saturation_prob** (float): The probability that the saturation is adjusted randomly. The default value is 0.5.
- **hue_range** (int): The range of the hue factor. The default value is 18.
- **hue_prob** (float): The probability that the hue is adjusted randomly. The default value is 0.5.

27.1.2 paddlex.det.transforms

This section describes the operation of data of the object detection/instance segmentation tasks. The `Compose` class can be used to combine image preprocessing/augmenter operations.

**Compose**

```python
paddlex.det.transforms.Compose(transforms)
```

The input data is operated by the data preprocessing/augmenter operator. **Usage Example**

Parameters

- **transforms** (list): Data preprocessing/data augmenter list.

**Normalize**

```python
paddlex.det.transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
```

Standardize the image.

1. Normalizes the image to the interval $[0, 1.0]$.
2. The image is subtracted from the mean and divided by the standard deviation.
Parameters

- **mean** (list): The mean value of the image data set. Default values are 0.485, 0.456, and 0.406.
- **std** (list): Standard deviation of the image dataset. Default values are 0.229, 0.224, 0.225.

### ResizeByShort

```
paddlex.det.transforms.ResizeByShort(short_size=800, max_size=1333)
```

Resizes the image according to the short edge of the image.

1. Get the length of the long and short edges of the image.
2. According to the ratio of short side and short_size, calculate the target length of the long side. At this time, the resize ratio of height and width is short_size/original short side length.
3. If max_size>0, adjust the resize ratio: if the target length of the long side is > max_size, the resize ratio of height and width is max_size/the length of the long edge of the original image.
4. Resize the image according to the resize ratio.

Parameters

- **short_size** (int): The length of the short side object. The default value is 800.
- **max_size** (int): Maximal limit of the length of the long side target. The default value is 1333.

### Padding

```
paddlex.det.transforms.Padding(coarsest_stride=1)
```

Multiples of padding the length and width of the image to the coarsest_stride. If the input image is [300], 640, and the coarsest_stride is 32, the rightmost and the bottom of the image is padded with 0, and the final output image is [[320], 640], because 300 is not a multiple of 32.

1. Returns directly if coarsest_stride is 1.
2. Calculate the difference between the width and the height and the nearest coarsest_stride multiple
3. Based on the calculated difference, padding is performed on the rightmost and lowest part of the image.

Parameters

- **coarsest_stride** (int): the length and width of the filled image is a multiple of this parameter. The default value is 1.
Resize

\[
paddlex.det.transforms.\text{Resize}(\text{target\_size}=608, \text{interp}='\text{LINEAR}')
\]

Resizes the image (resize).

- When the target size (target\_size) type is int, resize the image to [target\_size, target\_size] according to the interpolation method.
- When the target size (target\_size) type is list or tuple, resize the image to target\_size according to the interpolation method. [Note] When the interpolation method is “RANDOM”, one of the interpolation methods is randomly selected for resize. It is the data augmenter operation during model training.

Parameters

- **target\_size** (int/list/tuple): target length of short side. Default value is 608.
  
  The default value is “LINEAR”.

RandomHorizontalFlip

\[
paddlex.det.transforms.\text{RandomHorizontalFlip}(\text{prob}=0.5)
\]

Flip the image horizontally at random with a certain probability. It is the data augmenter operation during model training.

Parameters

- **prob** (float): The probability of a random level flip. The default value is 0.5.

RandomDistort

\[
paddlex.det.transforms.\text{RandomDistort}(\text{brightness\_range}=0.5, \text{brightness\_prob}=0.5, \text{contrast\_range}=0.5, \text{contrast\_prob}=0.5, \text{saturation\_range}=0.5, \text{saturation\_prob}=0.5, \text{hue\_range}=18, \text{hue\_prob}=0.5)
\]

Random pixel content transformation of the image with a certain probability. It is the data augmenter operation in the model training.

1. Randomize the operation order of the transformations.
2. Perform a random pixel content transformation in the range [-range, range] with a certain probability on the image in the order shown in Step 1.

[Note] This data augmenter must be used before the Normalize.

**Parameters**

- **brightness_range** (float): the range of the brightness factor. The default value is 0.5.
- **brightness_prob** (float): The probability that the brightness is adjusted randomly. The default value is 0.5.
- **contrast_range** (float): The range of the contrast factor. The default value is 0.5.
- **contrast_prob** (float): The probability of randomly adjusting the contrast. The default value is 0.5.
- **saturation_range** (float): The range of the saturation factor. The default value is 0.5.
- **saturation_prob** (float): The probability of randomly adjusting the saturation. The default value is 0.5.
- **hue_range** (int): The range of the hue factor. The default value is 18.
- **hue_prob** (float): The probability of randomly adjusting the hue. The default value is 0.5.

**MixupImage**

```python
def paddlex.det.transforms.MixupImage(alpha=1.5, beta=1.5, mixup_epoch=-1):
```

Perform mixup operations on images. It is the data augmenter operation during model training. Currently, only the YOLOv3 model supports this transform. When the mixup field does not exist in label_info, return directly. Otherwise, perform the following operations:

1. The random factor is extracted from the random beta distribution.

2. The processing varies with different scenarios.

   - When factor >= 1.0, remove the mixup field in label_info and return it directly.

   - When the factor <= 0.0, return the mixup field in label_info directly. The field is removed from label_info.

   - For the rest, perform the following operations: (1) multiply the original image by the factor, multiply the mixup image by (1-factor), and superimpose the two results. (2) Splice the original image label box and the mixup image label box. (3) Splice original image label box category and mixup image label box category. (4) Multiply the original image label box mixing score by the factor, and multiply the mixup image label box mixing score by (1-factor), and superimpose the 2 results.

3. Update the augment_shape information in im_info.
Parameters

- **alpha** (float): The lower limit of the random beta distribution. The default value is 1.5.
- **beta** (float): The upper limit of the random beta distribution. The default value is 1.5.
- **mixup_epoch** (int): Use mixup augmentation in the previous mixup_epoch round. This policy does not take effect when this parameter is -1. The default value is -1.

RandomExpand Class

```python
paddlex.det.transforms.RandomExpand(ratio=4.0, prob=0.5, fill_value=[123.675, 116.28, 103.53])
```

Randomly expand the image. It is the data augmenter operation during model training.

1. Randomly select the expansion ratio (expansion is performed only when the expansion ratio is greater than 1).
2. Calculate the size of the expanded image.
3. Initialize the image whose pixel value is the input fill-in value, and paste the original image randomly on this image.
4. Compute the position coordinates of the expanded real label box from the original image pasted position.
5. Compute the position of the real segmentation area after expansion based on the original image pasted position.

Parameters

- **ratio** (float): The maximum ratio of image expansion. The default value is 4.0.
- **prob** (float): Probability of random expansion. The default value is 0.5.
- **fill_value** (list): The initial fill-in value of the expanded image (0-255). The default value is 123.675, 116.28, 103.53.

[Note] This data augmenter must be used before the data augmenter of Resize and ResizeByShort.

RandomCrop

```python
paddlex.det.transforms.RandomCrop(aspect_ratio=[.5, 2.], thresholds=[.0, .1, .3, .5, .7, .9], scaling=[.3, 1.], num_attempts=50, allow_no_crop=True, cover_all_box=False)
```

Random crop image. It is the data augmenter operation during model training.

27.1. Data Processing and Enhancement
1. If `allow_no_crop` is True, add ‘no_crop’ to the thresholds.

2. Randomly disrupt the thresholds.

3. Traverse the elements in the thresholds: (1) If the current thresh is ‘no_crop’, return the original image and label information. (2) Randomly retrieve the values of aspect_ratio and scaling, and calculate the height, width and start point of the candidate cropping area. (3) Calculate the IoU of the real label box and the candidate cropping area. If all IoUs of the real label box are less than thresh, go to step 3. (4) If the `cover_all_box` is True and the IoU of the real label box is less than thresh, go to step 3. (5) Filter out the real label boxes located in the candidate cropping area. If the number of valid boxes is 0, go to step 3. Otherwise, go to step 4.

4. Convert the position coordinates of the valid true label box relative to the candidate cropping region.

5. Convert the position coordinates of the valid segmentation region relative to the candidate crop region.

[Note] This data augmenter must be used before the data augmenter of Resize and ResizeByShort.

**Parameters**

- **aspect_ratio** (list): the range of the scaling of the cropping short edge, in the form of min and max. The default values are [.5, 2.].

- **thresholds** (list): the list of IoU thresholds to determine whether the cropped candidate region is valid. The default values are [.0, .1, .3, .5, .7, .9].

- **scaling** (list): the range of the cropping area relative to the original area, in the form of min and max. The default values are [.3, 1.].

- **num_attempts** (int): The number of attempts before giving up on finding a valid crop area. The default value is 50.

- **allow_no_crop** (bool): Whether to allow no cropping. It is true by default.

- **cover_all_box** (bool): whether or not require all real label boxes to be in the crop area. It is false by default.

### 27.1.3 paddlex.seg.transforms

This section describes the operation on data used for segmentation tasks. The `Compose` class can be used to combine image preprocessing/augmenter operations.

**Compose**

```
paddlex.seg.transforms.Compose(transforms)
```

The input data is operated on according to the data preprocessing/data augmenter list. Usage Example
 Parameters

- **transforms** (list): Data preprocessing/data augmenter list.

RandomHorizontalFlip

```python
paddlex.seg.transforms.RandomHorizontalFlip(prob=0.5)
```

Flip the image horizontally with a certain probability. It is the data augmenter operation during model training.

 Parameters

- **prob** (float): The probability of a random level flip. It is 0.5 by default.

RandomVerticalFlip

```python
paddlex.seg.transforms.RandomVerticalFlip(prob=0.1)
```

Flip the image vertically with a certain probability. It is the data augmenter operation during model training.

 Parameters

- **prob** (float): probability of a random vertical flip. The default value is 0.1.

Resize

```python
paddlex.seg.transforms.Resize(target_size, interp='LINEAR')
```

Resizes the image (resize).

- When the target size (target_size) type is int, resize the image to [target_size, target_size] according to the interpolation method.
- When the target size (target_size) type is list or tuple, resize the image to target_size. The input for target_size should be [w, h] or (w, h) according to the interpolation method.

 Parameters

- **target_size** (int|list|tuple): target size
• **interp** (str): resize interpolation. It is corresponding to opencv interpolation. The available values are ‘NEAREST’, ‘LINEAR’, ‘CUBIC’, ‘AREA’, ‘LANCZOS4’ and the default value is “LINEAR”.

**ResizeByLong**

```python
paddlex.seg.transforms. ResizeByLong(long_size)
```

Resize the long side of the image to a fixed value and scale the short side proportionally.

**Parameters**

- **long_size** (int): Size of the long side of the image after resize.

**ResizeRangeScaling**

```python
paddlex.seg.transforms. ResizeRangeScaling(min_value=400, max_value=600)
```

Randomly resize the long side of the image to the specified range. Scale the short side proportionally. Perform the data augumenter operation during model training.

**Parameters**

- **min_value** (int): The minimum value after the long side is resized. The default value is 400.
- **max_value** (int): The maximum value after resizing the long side of the image. Default value is 600.

**ResizeStepScaling**

```python
paddlex.seg.transforms. ResizeStepScaling(min_scale_factor=0.75, max_scale_factor=1.25, scale_step_size=0.25)
```

Resize the image by a scale in scale_step_size. It varies randomly in the range \([\text{min\_scale\_factor}, \text{max\_scale\_factor}]\). It is the data augumenter operation during model training.

**Parameters**

- **min_scale_factor** (float), resize the minimum scale. The default value is 0.75.
- **max_scale_factor** (float), maximal resize scale. The default value is 1.25.
- **scale_step_size** (float), resize scale range interval. The default value is 0.25.
Normalize

```python
paddlex.seg.transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5], min_val=[0, 0, 0], max_val=[255.0, 255.0, 255.0])
```

Standardize the image.

1. Pixel value minus min_val
2. Pixel value divided by (max_val - min_val), normalized to interval [0, 1.0]
3. The image is subtracted from the mean and divided by the standard deviation.

Parameters

- **mean** (list): The mean value of the image data set. Default values are 0.5, 0.5, 0.5. The length should be the same as the number of image channels.
- **std** (list): Standard deviation of the image dataset. Default values are 0.5, 0.5, 0.5. The length should be the same as the number of image channels.
- **min_val** (list): Minimum value of the image dataset. The default value is 0, 0, 0. The length should be the same as the number of image channels.
- **max_val** (list): The maximum value of the image dataset. The default value is 255, 255, 255. The length should be the same as the number of image channels.

Padding

```python
paddlex.seg.transforms.Padding(target_size, im_padding_value=[127.5, 127.5, 127.5], label_padding_value=255)
```

Perform padding on an image or label image with padding direction right and down. Padding is applied to the image or label image according to the provided value.

Parameters

- **target_size** (int|list|tuple): Size of the image after padding.
- **im_padding_value** (list): The value of the padding of the image. The default value is 127.5, 127.5, 127.5. The length should be the same as the number of image channels.
- **label_padding_value** (int): The value of the label image padding. The default value is 255 (this parameter only needs to be set during training).
**RandomPaddingCrop**

```
paddlex.seg.transforms. RandomPaddingCrop(crop_size=512, im_padding_value=[127.5, 127.5, 127.5], label_padding_value=255)
```

Random cropping of images and labeled maps. Perform the padding operations when the desired crop size is larger than the original. It is the data augmenter operation during model training.

**Parameters**

- `crop_size`(int|list|tuple): The size of the crop image. The default value is 512.
- `im_padding_value` (list): The value of the padding of the image. The default value is 127.5, 127.5, 127.5. The length should be the same as the number of image channels.
- `label_padding_value` (int): The value of the label image padding. The default value is 255.

**RandomBlur**

```
paddlex.seg.transforms. RandomBlur(prob=0.1)
```

Gaussian blurring of the image with a certain probability. It is the data augmenter operation during model training.

**Parameters**

- `prob` (float): Probability of image blurring. It is 0.1 by default.

**RandomRotate**

```
paddlex.seg.transforms. RandomRotate(rotate_range=15, im_padding_value=[127.5, 127.5, 127.5], label_padding_value=255)
```

Random rotation of images. It is the data augmenter operation during model training. At present, it supports multi-channel RGB images. For example, it supports image data of multiple RGB images after concatenate along the channel axis, but it does not support image data with the number of channels not multiples of 3.

Random rotation of images within the rotation range `[-rotate_range, rotate_range]`. It is synchronized when labeled images exist. You can apply the corresponding padding to the rotated and labeled images.
Parameters

- **rotate_range** (float): The maximum rotation angle. The default value is 15 degrees.

- **im_padding_value** (list): The value of the padding of the image. The default value is $127.5, 127.5, 127.5$. The length should be the same as the number of image channels.

- **label_padding_value** (int): The value of the label image padding. The default value is 255.

**RandomScaleAspect**

```python
paddlex.seg.transforms.RandomScaleAspect(min_scale=0.5, aspect_ratio=0.33)
```

Crop and resize back to the original size image and labeled image. It is the data augmenter operation during model training.

Cropping and resizing the image according to the area ratio and aspect ratio. When there is a labeled image, the operation is performed simultaneously.

**Parameters**

- **min_scale** (float): the area ratio of the crop image to the original image, the value is 0 or 1. If the value is 0, return to the original image. The default value is 0.5.

- **aspect_ratio** (float): the aspect ratio range of the crop image. It is a non-negative value. When it is 0, return to the original image. The default value is 0.33.

**RandomDistort**

```python
paddlex.seg.transforms.RandomDistort(brightness_range=0.5, brightness_prob=0.5, contrast_range=0.5, contrast_prob=0.5, saturation_range=0.5, saturation_prob=0.5, hue_range=18, hue_prob=0.5)
```

Random pixel content transformation of the image with a certain probability. It is the data augmenter operation in the model training. At present, it supports multi-channel RGB images. For example, it supports image data of multiple RGB images after concatenate along the channel axis, but it does not support image data with the number of channels not multiples of 3.

1 Randomize the operation order of the transformations. 2 Perform a random pixel content transformation in the range $[-\text{range}, \text{range}]$ with a certain probability on the image in the order shown in Step 1.

[Note] This data augmenter must be used before the Normalize.
Parameters

- **brightness_range** (float): the range of the brightness factor. The default value is 0.5.
- **brightness_prob** (float): The probability that the brightness is adjusted randomly. The default value is 0.5.
- **contrast_range** (float): The range of the contrast factor. The default value is 0.5.
- **contrast_prob** (float): The probability of randomly adjusting the contrast. The default value is 0.5.
- **saturation_range** (float): The range of the saturation factor. The default value is 0.5.
- **saturation_prob** (float): The probability of randomly adjusting the saturation. The default value is 0.5.
- **hue_range** (int): The range of the hue factor. The default value is 18.
- **hue_prob** (float): The probability of randomly adjusting the hue. The default value is 0.5.

Clip

```python
paddlex.seg.transforms.Clip(min_val=[0, 0, 0], max_val=[255.0, 255.0, 255.0])
```

Clip data that is out of range on the image.

Parameters

- **min_val** (list): the lower limit of the clip, any value smaller than min_val is set to min_val. The default value is 0.
- **max_val** (list): The upper limit of the crop, any value greater than max_val will be set to max_val. The default value is 255.0.

27.1.4 Data augmenter and imgaug support

Data augmenter operations can be used to increase the diversity of the training samples during model training, thereby improving the generalization of the models.

PaddleX Built-in Augmenter Operations

PaddleX has some common data augmenter operations built in for image classification, object detection, instance segmentation, and semantic segmentation. See the following table.
Currently, PaddleX is adapted to the image augment library imgaug. You can directly construct the transforms in PaddleX by calling imgaug. The methods are as follows:

```python
import paddlex as pdx from paddlex.cls import transforms import imgaug.augmenters as iaa,

# train_transforms = transforms.Compose([ # Randomly blur the image by selecting the value in [0.0 3.0]. iaa.blur.GaussianBlur(sigma=(0.0, 3.0)), transforms.
# RandomCrop(crop_size=224), transforms.Normalize() ])
```

In addition to the above usage, the Compose interface also supports imgaug’s Someof, Sometimes, Sequential, Oneof, and so on. Developers are free to combine these methods to create an augmenter process. Since imgaug’s training logic for labeling information (object detection frame and instance segmentation mask) is different from that in PaddleX models, only the pixel-level augmenter method is supported in detection and segmentation, (that is, the size and orientation of the image are not changed during augmentation).

It should be noted that the basic methods of imgaug, such as `imgaug.augmenters.blur`, are only image processing operations, without the settings of probability. In the CV model training, augment operations are often applied to samples with a certain probability. Therefore, you can combine imgaug's Someof, Sometimes, Sequential, and Oneof operations to achieve this. See the following codes:

- **Someof**: executes some of the methods in the list of defined augmenter methods
- **Sometimes**: defines a list of augmenter methods executed in certain probability.
- **Sequential**: executes the defined list of augmenter methods in order.

```python
image imgaug.augmenters as iaa from paddlex.cls import transforms # Blurring of image
samples with the probability of 0.6 img_augmenters = iaa.Sometimes(0.6, [ iaa.blur.GaussianBlur(sigma=(0.0, 3.0)) ])
train_transforms = transforms.Compose([img
augmenters, transforms.RandomCrop(crop_size=224), transforms.Normalize() ])
```

### 27.2 Dataset reading

#### 27.2.1 paddlex.datasets. ImageNet

Used for image classification models

```python
paddlex.datasets.ImageNet(data_dir, file_list, label_list, transforms=None, num_workers='auto', buffer_size=8, parallel_method='process', shuffle=False)
```
Read a classification dataset in ImageNet format and process samples accordingly. For the introduction to the ImageNet dataset format, see the following document: [Dataset Format Description] (../data/format/classification.md)

Example: [Code file](https://github.com/PaddlePaddle/PaddleX/blob/develop/tutorials/train/image_classification/mobilenetv2.py)

**Parameters**

- **data_dir** (str): Directory path where the dataset is located.
- **file_list** (str): Describes a file path to a dataset image file and category ID (Each line of path in the text is a relative path relative to data_dir).
- **label_list** (str): Describes a category information file path contained in the dataset.
- **transforms** (paddlex.cls.transforms): Preprocessing/enhancement operator for each sample in the dataset. For the details, see paddlex.cls.transforms
- **num_workers** (int|str): The number of threads or processes during the preprocessing of samples in the dataset. It is ‘auto’ by default. When it is set to ‘auto’, num_workers is set according to the actual number of CPU cores of the system. If half of the number of CPU cores is greater than 8, num_workers is 8, otherwise num_workers is a half of the number of CPU cores.
- **buffer_size** (int): Queue cache length during the preprocessing of samples in the dataset. The unit is the number of samples. It is 8 by default.
- **parallel_method** (str): Parallel processing method during the preprocessing of samples in the dataset. Two methods including ‘thread’ thread and ‘process’ process are supported. It is ‘process’ by default (Thread is mandatory in Windows and Mac and this parameter is invalid).
- **shuffle** (bool): Whether to disrupt the order of the samples in the dataset. It is false by default.

### 27.2.2 paddlex.datasets. VOCDetection

Used for object detection models

```python
paddlex.datasets.VOCDetection(data_dir, file_list, label_list, transforms=None, num_workers='auto', buffer_size=100, parallel_method='process', shuffle=False)
```

Read a detection dataset in PascalVOC format and process samples accordingly. For the introduction to the PascalVOC dataset format, see the following document: [Dataset Format Description] (../data/format/detection.md)

Example: [Code file](https://github.com/PaddlePaddle/PaddleX/blob/develop/tutorials/train/detection/mobilenetv2.py)
- data_dir (str): Directory path where the dataset is located.
- file_list (str): Describes a file path to a dataset image file and the corresponding annotation file (Each line of path in the text is a relative path relative to data_dir).
- label_list (str): Describes a category information file path contained in the dataset.
- transforms (paddlex.det.transforms): Preprocessing/enhancement operator for each sample in the dataset. For the details, see paddlex.det.transforms
- num_workers (int|str): The number of threads or processes during the preprocessing of samples in the dataset. It is ‘auto’ by default. When it is set to ‘auto’, num_workers is set according to the actual number of CPU cores of the system. If half of the number of CPU cores is greater than 8, num_workers is 8, otherwise num_workers is a half of the number of CPU cores.
- buffer_size (int): Queue cache length during the preprocessing of samples in the dataset. The unit is the number of samples. It is 100 by default.
- parallel_method (str): Parallel processing method during the preprocessing of samples in the dataset. Two methods including ‘thread’ thread and ‘process’ process are supported. It is ‘process’ by default (Thread is mandatory in Windows and Mac and this parameter is invalid).
- shuffle (bool): Whether to disrupt the order of the samples in the dataset. It is false by default.

27.2.3 paddlex.datasets. CocoDetection

Used for instance segmentation/object detection models

```python
paddlex.datasets.CocoDetection(data_dir, ann_file, transforms=None, num_workers='auto',
                                buffer_size=100, parallel_method='process', shuffle=False)
```

Read a detection dataset in MSCOCO format and process samples accordingly. A dataset in this format can also be applied to the training of instance segmentation models. For the introduction to the MSCOCO dataset format, see the following document: Dataset Format Description

Example: Code file

Parameters

- data_dir (str): Directory path where the dataset is located.
- ann_file (str): Dataset annotation file as an independent file in json format.
- transforms (paddlex.det.transforms): Preprocessing/enhancement operator for each sample in the dataset. For the details, see [paddlex.det.transforms] (.transforms/det_transforms.md).
• **num_workers** (int|str): The number of threads or processes during the preprocessing of samples in the dataset. It is ‘auto’ by default. When it is set to ‘auto’, ‘num_workers’ is set according to the actual number of CPU cores of the system. If half of the number of CPU cores is greater than 8, ‘num_workers’ is 8, otherwise ‘num_workers’ is a half of the number of CPU cores.

• **buffer_size** (int): Queue cache length during the preprocessing of samples in the dataset. The unit is the number of samples. It is 100 by default.

• **parallel_method** (str): Parallel processing method during the preprocessing of samples in the dataset. Two methods including ‘thread’ thread and ‘process’ process are supported. It is ‘process’ by default (Thread is mandatory in Windows and Mac and this parameter is invalid).

• **shuffle** (bool): Whether to disrupt the order of the samples in the dataset. It is false by default.

### 27.2.4 paddlex.datasets. SegDataset

Used for semantic segmentation models

```python
paddlex.datasets.SegDataset(data_dir, file_list, label_list, transforms=None, num_workers='auto', buffer_size=100, parallel_method='process', shuffle=False)
```

Read a semantic segmentation task dataset and process samples accordingly. For the introduction to the semantic segmentation task dataset format, see the following document: [Dataset Format Description](#)

Example: Code file

#### Parameters

• **data_dir** (str): Directory path where the dataset is located.

• **file_list** (str): Describes a file path to a dataset image file and the corresponding annotation file (Each line of path in the text is a relative path relative to data_dir).

• **label_list** (str): Describes a category information file path contained in the dataset.

• **transforms** (paddlex.seg.transforms): Preprocessing/enhancement operator for each sample in the dataset. For the details, see paddlex.seg.transforms.

• **num_workers** (int|str): The number of threads or processes during the preprocessing of samples in the dataset. It is ‘auto’ by default. When it is set to ‘auto’, num_workers is set according to the actual number of CPU cores of the system. If half of the number of CPU cores is greater than 8, num_workers is 8, otherwise num_workers is a half of the number of CPU cores.
• **buffer_size** (int): Queue cache length during the preprocessing of samples in the dataset. The unit is the number of samples. It is 100 by default.

• **parallel_method** (str): Parallel processing method during the preprocessing of samples in the dataset. Two methods including ‘thread’ thread and ‘process’ process are supported. It is ‘process’ by default (Thread is mandatory in Windows and Mac and this parameter is invalid).

• **shuffle** (bool): Whether to disrupt the order of the samples in the dataset. It is false by default.

### 27.2.5 paddlex.datasets. EasyDataCls

Used for image classification models

```python
paddlex.datasets.EasyDataCls(data_dir, file_list, label_list, transforms=None, num_workers='auto', buffer_size=8, parallel_method='process', shuffle=False)
```

Read an annotation image classification dataset on the EasyData platform and process samples accordingly.

**Parameters**

• **data_dir** (str): Directory path where the dataset is located.

• **file_list** (str): Describes a file path to a dataset image file and the corresponding annotation file (Each line of path in the text is a relative path relative to data_dir).

• **label_list** (str): Describes a category information file path contained in the dataset.

• **transforms** (paddlex.seg.transforms): Preprocessing/enhancement operator for each sample in the dataset. For the details, see paddlex.cls.transforms.

• **num_workers** (int|str): The number of threads or processes during the preprocessing of samples in the dataset. It is ‘auto’ by default. When it is set to ‘auto’, num_workers is set according to the actual number of CPU cores of the system. If half of the number of CPU cores is greater than 8, num_workers is 8, otherwise num_workers is a half of the number of CPU cores.

• **buffer_size** (int): Queue cache length during the preprocessing of samples in the dataset. The unit is the number of samples. It is 8 by default.

• **parallel_method** (str): Parallel processing method during the preprocessing of samples in the dataset. Two methods including ‘thread’ thread and ‘process’ process are supported. It is ‘process’ by default (Thread is mandatory in Windows and Mac and this parameter is invalid).

• **shuffle** (bool): Whether to disrupt the order of the samples in the dataset. It is false by default.
27.2.6 paddle.datasets. EasyDataDet

Used for object detection/instance segmentation models

```python
paddle.datasets.EasyDataCls(data_dir, file_list, label_list, transforms=None, num_workers='auto', buffer_size=8, parallel_method='process', shuffle=False)
```

Read a dataset in EasyData object detection/instance segmentation format and process samples accordingly. A dataset in this format can also be applied to the training of instance segmentation models.

**Parameters**

- **data_dir** (str): Directory path where the dataset is located.
- **file_list** (str): Describes a file path to a dataset image file and the corresponding annotation file (Each line of path in the text is a relative path relative to `data_dir`).
- **label_list** (str): Describes a category information file path contained in the dataset.
- **transforms** (paddlex.seg.transforms): Preprocessing/enhancement operator for each sample in the dataset. For the details, see [paddlex.det.transforms] (./transforms/det_transforms.md)
- **num_workers** (int|str): The number of threads or processes during the preprocessing of samples in the dataset. It is ‘auto’ by default. When it is set to ‘auto’, `num_workers` is set according to the actual number of CPU cores of the system. If half of the number of CPU cores is greater than 8, `num_workers` is 8, otherwise `num_workers` is a half of the number of CPU cores.
- **buffer_size** (int): Queue cache length during the preprocessing of samples in the dataset. The unit is the number of samples. It is 100 by default.
- **parallel_method** (str): Parallel processing method during the preprocessing of samples in the dataset. Two methods including ‘thread’ thread and ‘process’ process are supported. It is ‘process’ by default (Thread is mandatory in Windows and Mac and this parameter is invalid).
- **shuffle** (bool): Whether to disrupt the order of the samples in the dataset. It is false by default.

27.2.7 paddle.datasets. EasyDataSeg

Used for semantic segmentation models

```python
paddle.datasets.EasyDataSeg(data_dir, file_list, label_list, transforms=None, num_workers='auto', buffer_size=100, parallel_method='process', shuffle=False)
```
Read an EasyData semantic segmentation task dataset and process samples accordingly.

**Parameters**

- **data_dir** (str): Directory path where the dataset is located.
- **file_list** (str): Describes a file path to a dataset image file and the corresponding annotation file (Each line of path in the text is a relative path relative to `data_dir`).
- **label_list** (str): Describes a category information file path contained in the dataset.
- **transforms** (paddlex.det.transforms): Preprocessing/enhancement operator for each sample in the dataset. For the details, see [paddlex.seg.transforms](./transforms/seg_transforms.md).
- **num_workers** (int|str): The number of threads or processes during the preprocessing of samples in the dataset. It is ‘auto’ by default. When it is set to ‘auto’, `num_workers` is set according to the actual number of CPU cores of the system. If half of the number of CPU cores is greater than 8, `num_workers` is 8, otherwise `num_workers` is a half of the number of CPU cores.
- **buffer_size** (int): Queue cache length during the preprocessing of samples in the dataset. The unit is the number of samples. It is 100 by default.
- **parallel_method** (str): Parallel processing method during the preprocessing of samples in the dataset. Two methods including ‘thread’ thread and ‘process’ process are supported. It is ‘process’ by default (Thread is mandatory in Windows and Mac and this parameter is invalid).
- **shuffle** (bool): Whether to disrupt the order of the samples in the dataset. It is false by default.

### 27.2.8 paddlex.datasets. ChangeDetDataset

Used for semantic segmentation models for change detection

```python
paddlex.datasets.ChangeDetDataset(data_dir, file_list, label_list, transforms=None, num_workers='auto', buffer_size=100, parallel_method='process', shuffle=False)
```

Read a semantic segmentation dataset for change detection and process samples accordingly. For an introduction to the change detection dataset format, see the following document: [Dataset Format Description](../data/format/change_det.md)

Example: Code file

**Parameters**

- **data_dir** (str): Directory path where the dataset is located.
- **file_list** (str): Describes a file path to dataset image 1 and 2 files and the corresponding annotation file (Each line of path in the text is a relative path relative to `data_dir`).

- **label_list** (str): Describes a category information file path contained in the dataset.

- **transforms** (paddlex.seg.transforms): Preprocessing/enhancement operator for each sample in the dataset. For the details, see paddlex.seg.transforms.

- **num_workers** (int|str): The number of threads or processes during the preprocessing of samples in the dataset. It is ‘auto’ by default. When it is set to ‘auto’, `num_workers` is set according to the actual number of CPU cores of the system. If half of the number of CPU cores is greater than 8, `num_workers` is 8, otherwise `num_workers` is a half of the number of CPU cores.

- **buffer_size** (int): Queue cache length during the preprocessing of samples in the dataset. The unit is the number of samples. It is 100 by default.

- **parallel_method** (str): Parallel processing method during the preprocessing of samples in the dataset. Two methods including ‘thread’ thread and ‘process’ process are supported. It is ‘process’ by default (Thread is mandatory in Windows and Mac and this parameter is invalid).

- **shuffle** (bool): Whether to disrupt the order of the samples in the dataset. It is false by default.

### 27.3 Dataset tools

#### 27.3.1 Dataset analysis

**paddlex.datasets.analysis.Seg**

```
paddlex.datasets.analysis.Seg(data_dir, file_list, label_list)
```

Construct the analyzer of statistical analysis semantic classification dataset.

**Parameters**

- **data_dir** (str): The directory path where the dataset is located.

- **file_list** (str): Describes the file path of the image file and category id of the dataset (the path of each line in the text is the relative path of the relative `data_dir`).

- **label_list** (str): Describes the path of the category information file contained in the dataset.
The analysis interface of SEG analyzer completes the analysis and statistics of the following information:

- Number of images
- Maximum and minimum size of image
- Number of image channels
- The minimum and maximum values of each channel of the image
- Pixel value distribution of each channel of the image
- Mean and variance of normalized image channels
- Mark the number and proportion of each category in the diagram

Code Example

Sample Example

cal_clipped_mean_std

cal_clipped_mean_std(self, clip_min_value, clip_max_value, data_info_file)

SEG analyzer is used to calculate the mean and variance of image after truncation.

Parameters

- `clip_min_value` (list): The lower limit of truncation and the values less than min_val are set as min_val.
- `clip_max_value` (list): The upper limit of truncation and the value greater than max_val is set as max_val.
- `data_info_file` (str): The path of the analysis result file (named train_information.pkl) saved in the analysis() interface.

Code Example

Calculation Results Example

27.3.2 Dataset generation
paddlex.det.paste_objects

paddlex.det.paste_objects(templates, background, save_dir='dataset_clone')

Paste the target object on the background image to generate a new image and annotation file

**Parameters**

- **templates** (list|tuple): The target objects on multiple images can be pasted on the same background image at the same time, so templates is a list, in which each element is a dict, which represents the target object of a picture. The target object of an image has two keywords `image` and `annos`. The key value of `image` is the path of the image, or it is an array of decoded array format (H, W, C) of uint8 and BGR format. There can be multiple target objects on the image, so the key value of `annos` is a list. Each element in the list is a dict, which represents the information of a target object. The dict contains two keywords `polygon` and `category`, where `polygon` represents the edge coordinates of the target object, such as [[0, 0], [0, 1], [1, 1], [1, 0]], and `category` represents the category of the target object, such as ‘dog’.

- **background** (dict): Background images can have true values, so background is a dict, which contains the keywords `image` and `annos`. The key value of `image` is the path of the background image, or it is an array of decoded array format (H, W, C) with uint8 type and BGR format. If there is no true value on the background image, the key value of `annos` is an empty list [], if there is one, the key value of `annos` is a list composed of multiple dicts. Each dict represents the information of an object, including the keywords `bbox` and `category`. The key value of `bbox` is the coordinates of the upper left corner and the lower right corner of the object frame, i.e. [x1, Y1, X2, Y2], and `category` represents the category of the target object, such as ‘dog’.

- **save_dir** (str): Storage directory for new pictures and their annotation files. The default value is `dataset_clone`.

**Code Example**

```python
import paddlex as pdx
templates = [{'image': 'dataset/JPEGImages/budaodian-10.jpg',
              'annos': [{'polygon': [[146, 169], [909, 169], [909, 489], [146, 489]],
                        'category': 'lou_di'},
                       {'polygon': [[146, 169], [909, 169], [909, 489], [146, 489]],
                        'category': 'lou_di'}]}]
background = {'image': 'dataset/JPEGImages/budaodian-12.jpg', 'annos': []}
pdx.det.paste_objects(templates, background, save_dir='dataset_clone')
```
27.4 Visual Model Set

Padlex currently supports four visual task solutions, including image classification, object detection, instance segmentation and semantic segmentation. For each visual task, paddlex provides one or more models, which users can select according to their needs and application scenarios.

27.4.1 Image Classification

paddlex.cls. ResNet50

```python
paddlex.cls. ResNet50(num_classes=1000)
```

Build a ResNet50 classifier and implement its training, evaluation and prediction.

Parameters

- **num_classes** (int): Number of classes. It is 1000 by default.

```python
train(self, num_epochs, train_dataset, train_batch_size=64, eval_dataset=None, save_interval_epochs=1, log_interval_steps=2, save_dir='output', pretrain_weights='IMAGENET', optimizer=None, learning_rate=0.025, warmup_steps=0, warmup_start_lr=0.0, lr_decay_epochs=[30, 60, 90], lr_decay_gamma=0.1, use_vdl=False, sensitivities_file=None, eval_metric_loss=0.05, early_stop=False, early_stop_patience=5, resume_checkpoint=None)
```

Parameters

- **num_epochs** (int): Number of training iteration epochs. - **train_dataset** (paddle.datasets): Training data reader. - **train_batch_size** (int): Training data batch size. It is also a validation data batch size. It is 64 by default. - **eval_dataset** (paddle.datasets): Validation data reader. - **save_interval_epochs** (int): Model saving interval (unit: number of iteration epochs). It is 1 by default. - **log_interval_steps** (int): Training log output interval (unit: number of iteration steps). It is 2 by default. - **save_dir** (str): Path where models are saved. - **pretrain_weights** (str): If it is a path, a pre-training model under the path is loaded. If it is a string, a model weight pre-trained on ImageNet image data is automatically downloaded. If it is None, no pre-training model is used. It is ‘IMAGENET’ by default.

- **optimizer** (paddle.fluid.optimizer): Optimizer. When this parameter is None, a default optimizer is used: fluid.layers.piecewise_decay attenuation policy, fluid.optimizer. Momentum optimization method. - **learning_rate** (float): Initial learning rate of the default optimizer. It is 0.025 by default.
• warmup_steps (int): Number of warmup steps of the default optimizer. The learning rate will be within a set number of steps and linearly increase to a set learning_rate from warmup_start_lr. It is 0 by default.

• warmup_start_lr (float): Warmup starting learning rate of the default optimizer. It is 0.0 by default.

• lr_decay_epochs (list): Number of learning rate attenuation epochs of the default optimizer. It is [30, 60, 90] by default.

• lr_decay_gamma (float): Attenuation rate of learning rate of the default optimizer. It is 0.1 by default.

• use_vdl (bool): Whether to use VisualDL for visualization. It is false by default.

• sensitivities_file (str): If it is a path, sensitivity information under the path is loaded to perform pruning. If it is a string ‘DEFAULT’, sensitivity information obtained from ImageNet image data is automatically downloaded to perform pruning. If it is None, no pruning is performed. It is None by default.

• eval_metric_loss (float): Tolerable precision loss. It is 0.05 by default.

• early_stop (bool): Whether to use a policy for early termination of training. It is false by default.

• early_stop_patience (int): When a policy for early termination of training is used, training is terminated if the validation set precision continuously decreases or remains unchanged within early_stop_patience epochs. It is 5 by default.

• resume_checkpoint (str): When training is resumed, specify a model path saved during the last training. If it is None, training is not resumed. It is None by default.

evaluate

evaluate(self, eval_dataset, batch_size=1, epoch_id=None, return_details=False)

Parameters

• eval_dataset (paddlex.datasets): Validation data reader.

• batch_size (int): Validation data batch size. It is 1 by default.

• epoch_id (int): Number of training epochs of the current evaluation model.

• return_details (bool): Whether to return detailed information. It is false by default.

Returned value

• dict: When return_details is false, dict is returned, containing keywords: ‘acc1’ and ‘acc5’ which indicate the accuracy of the maximum value and the accuracy of top 5
maximum values respectively.

- **tuple** (metrics, eval_details): When ‘return_details’ is true, the return of dict is increased, containing keywords: ‘true_labels’ and ‘pred_scores’ which indicate the true class ID and the prediction score of each class respectively.

### predict

```python
predict(self, img_file, transforms=None, topk=1)
```

Classification model prediction API. Note that the image processing flow during prediction can be saved in `ResNet50.test_transforms` and `ResNet50.eval_transforms` during model saving only when eval_dataset is defined during training. If eval_dataset is not defined during training, when the `predict` API for prediction is called, you need to redefine and pass test_transforms to the predict API.

**Parameters**

- **img_file** (str|np.ndarray): Path or numpy array of the predicted image (HWC arrangement, BGR format).
- **transforms** (paddlex.cls.transforms): Data preprocessing operation.
- **topk** (int): Top k maximum values during prediction.

**Returned value**

- **list**: All elements are dictionaries. Dictionary keywords include ‘category_id’, ‘category’ and ‘score’ which correspond to the prediction class ID, prediction class tag and prediction score respectively.

### batch_predict

```python
batch_predict(self, img_file_list, transforms=None, topk=1)
```

Classification model batch prediction API. Note that the image processing flow during prediction can be saved in `ResNet50.test_transforms` and `ResNet50.eval_transforms` during model saving only when eval_dataset is defined during training. If eval_dataset is not defined during training, when the `batch_predict` API for prediction is called, you need to redefine and pass test_transforms to the batch_predict API.

**Parameters**

- **img_file_list** (list|tuple): Images in the list (or tuple) are simultaneously predicted. Elements in the list may be image paths or numpy arrays (HWC arrangement, BGR format).
- **transforms** (paddlex.cls.transforms): Data preprocessing operation.
- **topk** (int): Top k maximum values during prediction.
Returned value

- list: Each element is a list which indicates prediction results of each image. All elements in the prediction list of images are dictionaries. Dictionary keywords include ‘category_id’, ‘category’ and ‘score’ which correspond to the prediction class ID, prediction class tag and prediction score respectively.

Other classification models

PaddleX provides a total of 22 classification models. All classification models provide the same train, evaluate and predict APIs as ‘ResNet50’. For model effects, refer to the model library.

27.4.2 Object Detection

paddlex.det. PPyOLO

```
paddlex.det. PPyOLO(num_classes=80, backbone='ResNet50_vd_ssld', with_dcn_v2=True, ,
   anchors=None, anchor_masks=None, use_coord_conv=True, use_iou_aware=True, use_spp=True,
   use_drop_block=True, scale_x_y=1.05, ignore_threshold=0.7, label_smooth=False, use_%
   iou_loss=True, use_matrix_nms=True, nms_score_threshold=0.01, nms_topk=1000, nms_keep_%
   topk=100, nms_iou_threshold=0.45, train_random_shapes=[320, 352, 384, 416, 448, 480, ,
   512, 544, 576, 608])
```

Build a PPyOLO detector. Note that num_classes does not need to include background class in PPyOLO. If an object includes humans and dogs, set num_classes to 2, which is different from FasterRCNN/MaskRCNN here

Parameters

- num_classes (int): Number of classes. It is 80 by default.
- backbone (str): PPyOLO backbone network in a value range of ‘ResNet50_vd_ssld’. [It is ‘ResNet50_vd_ssld’ by default.]
- with_dcn_v2 (bool): Whether Backbone uses DCNv2 structure. It is true by default.
- anchors (list|tuple): Width and height of the anchor box. When it is none, it indicates using the default [[10, 13], [16, 30], [33, 23], [30, 61], [62, 45], [59, 119], [116, 90], [156, 198], [373, 326]].
- anchor_masks (list|tuple): When the PPyOLO loss is calculated, the mask index of the anchor is used. When it is none, it indicates using the default [[6, 7, 8], [3, 4, 5], [0, 1, 2]].
- use_coord_conv (bool): Whether to use CoordConv. It is true by default.
- use_iou_aware (bool): Whether to use an IoU Aware branch. It is true by default.
• use_spp (bool): Whether to use Spatial Pyramid Pooling structure. It is true by default.
• use_drop_block (bool): Whether to use Drop Block. It is true by default.
• scale_x_y (float): Factor when the center is adjusted. It is 1.05 by default.
• use_iou_loss (bool): Whether to use IoU loss. It is true by default.
• use_matrix_nms (bool): Whether to use Matrix NMS. It is true by default.
• ignore_threshold (float): When the PPYOLO loss is calculated, the confidence of predicted boxes of which the IoU is greater than ignore_threshold is ignored. It is 0.7 by default.
• nms_score_threshold (float): Confidence score threshold of the detected box. Any box of which the confidence is smaller than the threshold shall be ignored. It is 0.01 by default.
• nms_topk (int): Maximum number of detected boxes reserved according to the confidence when NMS is performed. It is 1000 by default.
• nms_keep_topk (int): Total number of detected boxes to be reserved for each image after NMS is performed. It is 100 by default.
• nms_iou_threshold (float): IOU threshold used to eliminate detected boxes when NMS is performed. It is 0.45 by default.
• label_smooth (bool): Whether to use label smooth. It is false by default.
• train_random_shapes (list|tuple): Image size randomly selected from the list during training. It is 320, 352, 384, 416, 448, 480, 512, 544, 576, 608 by default.

```python
train(self, num_epochs, train_dataset, train_batch_size=8, eval_dataset=None, save_interval_epochs=20, log_interval_steps=2, save_dir='output', pretrain_weights='IMAGENET', optimizer=None, learning_rate=1.0/8000, warmup_steps=1000, warmup_start_lr=0.0, lr_decay_epochs=[213, 240], lr_decay_gamma=0.1, metric=None, use_vdl=False, sensitivities_file=None, eval_metric_loss=0.05, early_stop=False, early_stop_patience=5, resume_checkpoint=None, use_ema=True, ema_decay=0.9998)
```

PPYOLO model training API. The function has a built-in piecewise learning rate attenuation policy and a momentum optimizer.

Parameters

• num_epochs (int): Number of training iteration epochs.
• train_dataset (paddle.datasets): Training data reader.
• train_batch_size (int): Training data batch size. Currently, the detection supports only the single-card evaluation. The quotient of the training data batch size and the GPU quantity is a validation data batch size. It is 8 by default.
- **eval_dataset** (paddlex.datasets): Validation data reader. - **save_interval_epochs** (int): Model saving interval (unit: number of iteration epochs). It is 20 by default. - **log_interval_steps** (int): Training log output interval (unit: number of iterations). It is 2 by default. - **save_dir** (str): Path where models are saved. It is `output` by default. - **pretrain_weights** (str): If it is a path, a pre-training model under the path is loaded. If it is a string ‘IMAGENET’, a model weight pre-trained on ImageNet image data is automatically downloaded. If it is a string ‘COCO’, a model weight pre-trained on the COCO dataset is automatically downloaded. If it is none, no pre-training model is used. It is ‘IMAGENET’ by default. - **optimizer** (paddle.fluid.optimizer): Optimizer. When this parameter is none, a default optimizer is used: fluid.layers.piecewise_decay attenuation policy, fluid.optimizer. Momentum optimization method. - **learning_rate** (float): Learning rate of the default optimizer. It is 1.0/8000 by default. - **warmup_steps** (int): Number of steps to perform the warmup process by the default optimizer. It is 1000 by default. - **warmup_start_lr** (int): Initial learning rate of warmup of the default optimizer. It is 0.0 by default. - **lr_decay_epochs** (list): Number of learning rate attenuation epochs of the default optimizer. It is [213, 240] by default. - **lr_decay_gamma** (float): Attenuation rate of learning rate of the default optimizer. It is 0.1 by default. - **metric** (bool): Evaluation method during training in the value range of [ ‘COCO’, ‘VOC’ ] . It is None by default. - **use_vdl** (bool): Whether to use VisualDL for visualization. It is false by default. - **sensitivities_file** (str): If it is a path, sensitivity information under the path is loaded to perform pruning. If it is a string ‘DEFAULT’, sensitivity information obtained on PascalVOC data is automatically downloaded to perform pruning. If it is none, no pruning is performed. It is None by default. - **eval_metric_loss** (float): Tolerable precision loss. It is 0.05 by default. - **early_stop** (bool): Whether to use a policy for early termination of training. It is false by default. - **early_stop_patience** (int): When a policy for early termination of training is used, training is terminated if the validation set precision continuously decreases or remains unchanged within early_stop_patience epochs. It is 5 by default. - **resume_checkpoint** (str): When training is resumed, specify a model path saved during the last training. If it is None, training is not resumed. It is None by default. - **use_ema** (bool): Whether to use exponential attenuation to calculate a parameter sliding average value. It is true by default. - **ema_decay** (float): Exponential attenuation rate. It is 0.9998 by default.

### evaluate

```python
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, metric=None, return_details=False)
```

PPYOLO model evaluation API. The index `box_map` (when metric is set to ‘VOC’) or `box_mmap` (when metric is set to COCO) on the validation set is returned after the model is evaluated.

### Parameters
- **eval_dataset** (paddlex.datasets): Validation data reader.
- **batch_size** (int): Validation data batch size. It is 1 by default.
- **epoch_id** (int): Number of training epochs of the current evaluation model.
- **metric** (bool): Evaluation method during training in the value range of ['COCO', 'VOC']. It is none by default. It is automatically selected according to the dataset passed by you. If it is VOCDetection, **metric** is 'VOC'. If it is COCODetection, **metric** is 'COCO'. If it is a EasyData dataset, 'VOC' is also used.
- **return_details** (bool): Whether to return detailed information. It is false by default.

Returned value

- **tuple** (metrics, eval_details) | **dict** (metrics): When **return_details** is true, (metrics, eval_details) is returned. When **return_details** is false, metrics is returned. metrics is dict and contains keywords: 'bbox_mmap' or 'bbox_map' which respectively indicates that the results of the average value of average accuracy rates under each threshold take the results of the average value (mmAP) and the average value of average accuracy rates (mAP). eval_details is dict and contains two keywords: 'bbox' and 'gt'. The key value of the keyword 'bbox' is a list. Each element in the list represents an prediction result. An prediction result is a list consisting of an image ID, an predicted box class ID, predicted box coordinates and an predicted box score. The key value of the keyword 'gt' is information on the true annotated box.

**predict**

```python
predict(self, img_file, transforms=None)
```

PPYOLO model prediction API. Note that the image processing flow during prediction can be saved in YOLOv3.test_transforms' and YOLOv3.eval_transforms during model saving only when eval_dataset is defined during training. If eval_dataset is not defined during training, when the predict' API for prediction is called, you need to redefine and pass test_transforms to the predict API

Parameters

- **img_file** (str|np.ndarray): Path or numpy array of the predicted image (HWC arrangement, BGR format).
- **transforms** (paddlex.det.transforms): Data preprocessing operation.

Returned value

- **list**: List of prediction results. Each element in the list has a dict. The key includes 'bbox', 'category', 'category_id' and 'score' which indicate the box coordinate information, class, class ID and confidence of each predicted object respectively. The box coordinate
information is \([x_{\text{min}}, y_{\text{min}}, w, h]\), i.e. the x and y coordinates and the box width and height in the top left corner.

**batch_predict**

```python
batch_predict(self, img_file_list, transforms=None)
```

PPYOLO model batch prediction API. Note that the image processing flow during prediction can be saved in `YOLOv3.test_transforms` and `YOLOv3.eval_transforms` during model saving only when `eval_dataset` is defined during training. If `eval_dataset` is not defined during training, when the `batch_predict` API for prediction is called, you need to redefine and pass `test_transforms` to the `batch_predict` API.

**Parameters**

- **img_file_list** (str|np.ndarray): Images in the list (or tuple) are simultaneously predicted. Elements in the list are predicted image paths or numpy arrays (HWC arrangement, BGR format).
- **transforms** (paddlex.det.transforms): Data preprocessing operation.

**Returned value**

- **list**: Each element is a list which indicates prediction results of each image. Each element in the list of prediction results of each image has a dict. The key includes ‘bbox’, ‘category’, ‘category_id’ and ‘score’ which indicate the box coordinate information, class, class ID and confidence of each predicted object respectively. The box coordinate information is \([x_{\text{min}}, y_{\text{min}}, w, h]\), i.e. the x and y coordinates and the box width and height in the top left corner.

**paddlex.det. YOLOv3**

```python
paddlex.det.YOLOv3(num_classes=80, backbone='MobileNetV1', anchors=None, anchor_masks=None, ignore_threshold=0.7, nms_score_threshold=0.01, nms_topk=1000, nms_keep_topk=100, nms_iou_threshold=0.45, label_smooth=False, train_random_shapes=[320, 352, 384, 416, 448, 480, 512, 544, 576, 608])
```

Build a YOLOv3 detector. **Note that num_classes does not need to include background class in YOLOv3. If an object includes humans and dogs, set num_classes to 2, which is different from FasterRCNN/MaskRCNN here**

**Parameters**

- **num_classes** (int): Number of classes. It is 80 by default.
- **backbone** (str): YOLOv3 backbone network in a value range of ['DarkNet53', 'ResNet34', 'MobileNetV1' and 'MobileNetV3_large']. It is 'MobileNetV1' by default.

- **anchors** (list|tuple): Width and height of the anchor box. When it is none, it indicates using the default [[10, 13], [16, 30], [33, 23], [30, 61], [62, 45], [59, 119], [116, 90], [156, 198], [373, 326]].

- **anchor_masks** (list|tuple): When the YOLOv3 loss is calculated, the mask index of the anchor is used. When it is none, it indicates using the default [[6, 7, 8], [3, 4, 5], [0, 1, 2]].

- **ignore_threshold** (float): When the YOLOv3 loss is calculated, the confidence of predicted boxes of which the IoU is greater than `ignore_threshold` is ignored. It is 0.7 by default.

- **nms_score_threshold** (float): Confidence score threshold of the detected box. Any box of which the confidence is smaller than the threshold shall be ignored. It is 0.01 by default.

- **nms_topk** (int): Maximum number of detected boxes reserved according to the confidence when NMS is performed. It is 1000 by default.

- **nms_keep_topk** (int): Total number of detected boxes to be reserved for each image after NMS is performed. It is 100 by default.

- **nms_iou_threshold** (float): IOU threshold used to eliminate detected boxes when NMS is performed. It is 0.45 by default.

- **label_smooth** (bool): Whether to use label smooth. It is false by default.

- **train_random_shapes** (list|tuple): Image size randomly selected from the list during training. It is [320, 352, 384, 416, 448, 480, 512, 544, 576, 608] by default.

```python
def train(self, num_epochs, train_dataset, train_batch_size=8, eval_dataset=None, save_interval_epochs=20, log_interval_steps=2, save_dir='output', pretrain_weights='IMAGENET', optimizer=None, learning_rate=1.0/8000, warmup_steps=1000, warmup_start_lr=0.0, lr_decay_epochs=[213, 240], lr_decay_gamma=0.1, metric=None, use_vdl=False, sensitivities_file=None, eval_metric_loss=0.05, early_stop=False, early_stop_patience=5, resume_checkpoint=None)
```

YOLOv3 model training API. The function has a built-in **piecewise** learning rate attenuation policy and a **momentum** optimizer.

**Parameters**

- **num_epochs** (int): Number of training iteration epochs.

- **train_dataset** (paddlex.datasets): Training data reader.

- **train_batch_size** (int): Training data batch size. Currently, the detection supports only the single-card evaluation. The quotient of the training data batch size and the GPU quantity is a validation data batch size. It is 8 by default.

- **eval_dataset** (paddlex.datasets): Validation data reader.

- **save_interval_epochs** (int): Model saving interval (unit: number of iteration epochs).
It is 20 by default.

- **log_interval_steps** (int): Training log output interval (unit: number of iterations). It is 2 by default.

- **save_dir** (str): Path where models are saved. It is ‘output’ by default.

- **pretrain_weights** (str): If it is a path, a pre-training model under the path is loaded. If it is a string ‘IMAGENET’, a model weight pre-trained on ImageNet image data is automatically downloaded. If it is a string ‘COCO’, a model weight pre-trained on the COCO dataset is automatically downloaded. If it is none, no pre-training model is used. It is ‘IMAGENET’ by default.

- **optimizer** (paddle.fluid.optimizer): Optimizer. When this parameter is none, a default optimizer is used: fluid.layers.piecewise_decay attenuation policy, fluid.optimizer. Momentum optimization method.

- **learning_rate** (float): Learning rate of the default optimizer. It is 1.0/8000 by default.

- **warmup_steps** (int): Number of steps to perform the warmup process by the default optimizer. It is 1000 by default.

- **warmup_start_lr** (int): Initial learning rate of warmup of the default optimizer. It is 0.0 by default.

- **lr_decay_epochs** (list): Number of learning rate attenuation epochs of the default optimizer. It is [213, 240] by default.

- **lr_decay_gamma** (float): Attenuation rate of learning rate of the default optimizer. It is 0.1 by default.

- **metric** (bool): Evaluation method during training in the value range of [‘COCO’, ‘VOC’]. It is None by default.

- **use_vdl** (bool): Whether to use VisualDL for visualization. It is false by default.

- **sensitivities_file** (str): If it is a path, sensitivity information under the path is loaded to perform pruning. If it is a string ‘DEFAULT’, sensitivity information obtained on PascalVOC data is automatically downloaded to perform pruning. If it is none, no pruning is performed. It is None by default.

- **eval_metric_loss** (float): Tolerable precision loss. It is 0.05 by default.

- **early_stop** (bool): Whether to use a policy for early termination of training. It is false by default.

- **early_stop_patience** (int): When a policy for early termination of training is used, training is terminated if the validation set precision continuously decreases or remains unchanged within early_stop_patience epochs. It is 5 by default.
resume_checkpoint (str): When training is resumed, specify a model path saved during the last training. If it is None, training is not resumed. It is None by default.

evaluate

evaluate(self, eval_dataset, batch_size=1, epoch_id=None, metric=None, return_details=False)

YOLOv3 model evaluation API. The index box_map (when metric is set to ‘VOC’) or box_mmap (when metric is set to COCO) on the validation set is returned after the model is evaluated.

Parameters

- **eval_dataset** (paddlex.datasets): Validation data reader. - **batch_size** (int): Validation data batch size. It is 1 by default. - **epoch_id** (int): Number of training epochs of the current evaluation model. - **metric** (bool): Evaluation method during training in the value range of [‘COCO’, ‘VOC’]. It is none by default. It is automatically selected according to the dataset passed by you. If it is VOCDetection, metric is [‘VOC’]. If it is COCODetection, metric is ‘COCO’ . If it is a EasyData dataset, ‘VOC’ is also used. - **return_details** (bool): Whether to return detailed information. It is false by default.

Returned value

- **tuple** (metrics, eval_details) | **dict** (metrics): When return_details is true, (metrics, eval_details) is returned. When return_details is false, metrics is returned. metrics is dict and contains keywords: ’bbox_mmap’ or bbox_map which respectively indicates that the results of the average value of average accuracy rates under each threshold take the results of the average value (mmAP) and the average value of average accuracy rates (mAP). eval_details is dict and contains two keywords: ’bbox’ and gt. The key value of the keyword bbox; is a list. Each element in the list represents an prediction result. An prediction result is a list consisting of an image ID, an predicted box class ID, predicted box coordinates and an predicted box score. The key value of the keyword gt is information on the true annotated box.

predict

predict(self, img_file, transforms=None)

YOLOv3 model prediction API. Note that the image processing flow during prediction can be saved in YOLOv3.test_transforms and YOLOv3.eval_transforms during model saving only when eval_dataset is defined during training. If eval_dataset is not defined during training,
when the **predict** API for prediction is called, you need to redefine and pass **test_transforms** to the predict API

**Parameters**

- **img_file** (str|np.ndarray): Path or numpy array of the predicted image (HWC arrangement, BGR format).
- **transforms** (paddlex.det.transforms): Data preprocessing operation.

**Returned value**

- **list**: List of prediction results. Each element in the list has a dict. The key includes ‘bbox’, ‘category’, ‘category_id’ and ‘score’ which indicate the box coordinate information, class, class ID and confidence of each predicted object respectively. The box coordinate information is [xmin, ymin, w, h], i.e. the x and y coordinates and the box width and height in the top left corner.

**batch_predict**

```python
batch_predict(self, img_file_list, transforms=None)
```

YOLOv3 model batch prediction API. Note that the image processing flow during prediction can be saved in YOLOv3.test_transforms and YOLOv3.eval_transforms during model saving only when eval_dataset is defined during training. If eval_dataset is not defined during training, when the **batch_predict** API for prediction is called, you need to redefine and pass **test_transforms** to the **batch_predict** API

**Parameters**

- **img_file_list** (str|np.ndarray): Images in the list (or tuple) are simultaneously predicted. Elements in the list are predicted image paths or numpy arrays (HWC arrangement, BGR format).
- **transforms** (paddlex.det.transforms): Data preprocessing operation.

**Returned value**

- **list**: Each element is a list which indicates prediction results of each image. Each element in the list of prediction results of each image has a dict. The key includes ‘bbox’, ‘category’, ‘category_id’ and ‘score’ which indicate the box coordinate information, class, class ID and confidence of each predicted object respectively. The box coordinate information is [xmin, ymin, w, h], i.e. the x and y coordinates and the box width and height in the top left corner.
paddle.det. FasterRCNN

```python
defaultdict(int): Number of classes including the background class. It is 81 by default.
- **aspect_ratios** (list): Optional value of the anchor aspect ratio. It is [0.5, 1.0, 2.0] by default. - **anchor_sizes** (list): Optional value of the anchor size. It is [32, 64, 128, 256, 512] by default.
- **with_dcn** (bool): Whether to use deformable convolution network v2 in the backbone. Default: False.
- **rpn_cls_loss** (str): The classification loss function for RPN in a value range of ['SigmoidCrossEntropy', 'SigmoidFocalLoss']. When there are many false positives in backgorund areas, 'SigmoidFocalLoss' with appropriate rpn_focal_loss_alpha and rpn_focal_loss_gamma settings may be a better option. Default: 'SigmoidCrossEntropy'.
- **rpn_focal_loss_alpha** (float): Hyper-parameter to balance the positive and negative examples where 'SigmoidFocalLoss' is set as the lassification loss function for RPN, Default: 0.25. If use 'SigmoidCrossEntropy', rpn_focal_loss_alpha has no effect.
- **rpn_focal_loss_gamma** (float): Hyper-parameter to balance the easy and hard examples where 'SigmoidFocalLoss' is set as the lassification loss function for RPN, Default: 2. If use 'SigmoidCrossEntropy', rpn_focal_loss_gamma has no effect.
- **rcnn_bbox_loss** (str): The location regression loss function for RCNN in a value range of ['SmoothL1Loss', 'CIoULoss']. Default: 'SmoothL1Loss'.
- **rcnn_nms** (str): The non-maximum suppression(NMS) method for RCNN, in a value range of ['MultiClassNMS', 'MultiClassSoftNMS', 'MultiClassCiouNMS']. Default:
PaddleX

MultiClassNMS . When MultiClassNMS is set, keep_top_k, nms_threshold and score_threshold can be set as 100, 0.5 and 0.05 respectively. When MultiClassSoftNMS is set, keep_top_k, score_threshold and softnms_sigma can be set as 300, 0.01 and 0.5 respectively. When MultiClassCionNMS is set, keep_top_k, score_threshold and nms_threshold can be set as 100, 0.05 and 0.5 respectively.

- **keep_top_k** (int): The Number of total bounding boxes to be kept per image after NMS step for RCNN. Default: 100.
- **nms_threshold** (float): The IoU threshold to filter out bounding boxes in NMS for RCNN. When rcnn_nms is set as MultiClassSoftNMS, nms_threshold has no effect. Default: 0.5.
- **score_threshold** (float): The confidence score threshold to filter out bounding boxes before nms. Default: 0.05.
- **softnms_sigma** (float): When rcnn_nms is set as MultiClassSoftNMS, softnms_sigma is used to adjust the confidence score of suppressed bounding boxes according to \( \text{score} = \text{score} \times \text{weights} \), \( \text{weights} = \exp\left(-\left(\frac{\text{iou} \times \text{iou}}{\text{softnms_sigma}}\right)\right) \). Default: 0.5.
- **bbox_assigner** (str): The method of sampling positive and negative examples during the training phase, in a value range of [ 'BBoxAssigner', 'LibraBBoxAssigner' ]. If the size of objects is a small portion of the image, LibraRCNN proposed a IoU-balanced sampling method to obtain more hard-negative examples, namely 'LibraBBoxAssigner'. Default: 'BBoxAssigner'.
- **fpn_num_channels** (int): The number of channels of feature maps in FPN2. Default: 56.
- **input_channel** (int): The number of channels of a input image. Default: 3.
- **rpn_batch_size_per_im** (int): Total number of training examples per image for RPN. Default: 256.
- **rpn_fg_fraction** (float): The fraction of positive examples in total train examples for RPN. Default: 0.5.
- **test_pre_nms_top_n** (int): The number of predicted bounding boxes fed into NMS step. If set as None, test_pre_nms_top_n will be set as 6000 with a FPN or 1000 with no FPN. Default: None.
- **test_post_nms_top_n** (int): The number of predicted bounding boxes kept after NMS step. Default: 1000.

```python
train(self, num_epochs, train_dataset, train_batch_size=2, eval_dataset=None, save_interval_epochs=1, log_interval_steps=2, save_dir='output', pretrain_weights='IMAGENET', optimizer=None, learning_rate=0.0025, warmup_steps=500, warmup_start_lr=1.0/1200, decay_epochs=[8, 11], lr_decay_gamma=0.1, metric=None, use_vdl=False, early_stop=False, early_stop_patience=5, resume_checkpoint=None)
```

Chapter 27. API Interface Description
FasterRCNN model training API. The function has a built-in **piecewise** learning rate attenuation policy and a **momentum** optimizer.

**Parameters**

- **num_epochs** (int): Number of training iteration epochs.
- **train_dataset** (paddlex.datasets): Training data reader.
- **train_batch_size** (int): Training data batch size. Currently, the detection supports only the single-card evaluation. The quotient of the training data batch size and the GPU quantity is a validation data batch size. It is 2 by default.
- **eval_dataset** (paddlex.datasets): Validation data reader.
- **save_interval_epochs** (int): Model saving interval (unit: number of iteration epochs). It is 1 by default.
- **log_interval_steps** (int): Training log output interval (unit: number of iterations). It is 2 by default.
- **save_dir** (str): Path where models are saved. It is `output` by default.
- **pretrain_weights** (str): If it is a path, a pre-training model under the path is loaded. If it is a string `IMAGENET`, a model weight pre-trained on ImageNet image data is automatically downloaded. If it is a string `COCO`, a model weight pre-trained on the COCO dataset is automatically downloaded (Note: A COCO pre-training model for ResNet18 is unavailable temporarily. If it is none, no pre-training model is used. It is `IMAGENET` by default.
- **optimizer** (paddle.fluid.optimizer): Optimizer. When this parameter is none, a default optimizer is used: fluid.layers.piecewise_decay attenuation policy, fluid.optimizer. Momentum optimization method.
- **learning_rate** (float): Initial learning rate of the default optimizer. It is 0.0025 by default.
- **warmup_steps** (int): Number of steps to perform the warmup process by the default optimizer. It is 500 by default.
- **warmup_start_lr** (int): Initial learning rate of warmup of the default optimizer. It is 1.0/1200 by default.
- **lr_decay_epochs** (list): Number of learning rate attenuation epochs of the default optimizer. It is [8, 11] by default.
- **lr_decay_gamma** (float): Attenuation rate of learning rate of the default optimizer. It is 0.1 by default.
- **metric** (bool): Evaluation method during training in the value range of [‘COCO’, ‘VOC’]. It is None by default.

- **use_vdl** (bool): Whether to use VisualDL for visualization. It is false by default.

- **early_stop** (float): Whether to use a policy for early termination of training. It is false by default.

- **early_stop_patience** (int): When a policy for early termination of training is used, training is terminated if the validation set precision continuously decreases or remains unchanged within `early_stop_patience` epochs. It is 5 by default.

- **resume_checkpoint** (str): When training is resumed, specify a model path saved during the last training. If it is None, training is not resumed. It is None by default.

### evaluate

```python
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, metric=None, return_details=False)
```

FasterRCNN model evaluation API. The index box_map (when metric is set to VOC) or box_mmap (when metric is set to COCO) on the validation set is returned after the model is evaluated.

**Parameters**

- **eval_dataset** (paddlex.datasets): Validation data reader.

- **batch_size** (int): Validation data batch size. It is 1 by default. Currently, it must be set to 1.

- **epoch_id** (int): Number of training epochs of the current evaluation model.

- **metric** (bool): Evaluation method during training in the value range of [‘COCO’, ‘VOC’]. It is none by default. It is automatically selected according to the dataset passed by you. If it is VOC Detection, ‘metric’ is ‘VOC’. If it is COCO Detection, ‘metric’ is ‘COCO’.

- **return_details** (bool): Whether to return detailed information. It is false by default.

**Returned value**

- **tuple** (metrics, eval_details) | **dict** (metrics): When ‘return_details’ is true, (metrics, eval_details) is returned. When ‘return_details’ is false, metrics is returned. metrics is dict and contains keywords: ‘bbox_mmap’ or ‘bbox_map’ which respectively indicates that the results of the average value of average accuracy rates under each threshold take the results of the average value (mMAP) and the average value of average accuracy rates (mAP). eval_details is dict and contains two keywords: ‘bbox’ and ‘gt’. The key value of the keyword bbox; is
a list. Each element in the list represents a prediction result. A prediction result is a list consisting of an image ID, a predicted box class ID, predicted box coordinates and a predicted box score. The key value of the keyword `gt` is information on the true annotated box.

**predict**

```python
predict(self, img_file, transforms=None)
```

FasterRCNN model prediction API. Note that the image processing flow during prediction can be saved in `FasterRCNN.test_transforms` and `FasterRCNN.eval_transforms` during model saving only when `eval_dataset` is defined during training. If `eval_dataset` is not defined during training, when the `predict` API for prediction is called, you need to redefine and pass `test_transforms` to the `predict` API.

**Parameters**

- **img_file** (str|np.ndarray): Path or numpy array of the predicted image (HWC arrangement, BGR format).

- **transforms** (paddlex.det.transforms): Data preprocessing operation.

**Returned value**

- **list**: List of prediction results. Each element in the list has a dict. The key includes `'bbox'`, `'category'`, `'category_id'` and `'score'` which indicate the box coordinate information, class, class ID and confidence of each predicted object respectively. The box coordinate information is `[xmin, ymin, w, h]`, i.e. the x and y coordinates and the box width and height in the top left corner.

**batch_predict**

```python
batch_predict(self, img_file_list, transforms=None)
```

FasterRCNN model batch prediction API. Note that the image processing flow during prediction can be saved in `FasterRCNN.test_transforms` and `FasterRCNN.eval_transforms` during model saving only when `eval_dataset` is defined during training. If `eval_dataset` is not defined during training, when the `batch_predict` API for prediction is called, you need to redefine and pass `test_transforms` to the `batch_predict` API.

**Parameters**

- **img_file_list** (list|tuple): Images in the list (or tuple) are simultaneously predicted. Elements in the list are predicted image paths or numpy arrays (HWC arrangement, BGR format).

- **transforms** (paddlex.det.transforms): Data preprocessing operation.
Returned value

- **list**: Each element is a list which indicates prediction results of each image. Each element in the list of prediction results of each image has a dict. The key includes ‘bbox’, ‘category’, ‘category_id’ and ‘score’ which indicate the box coordinate information, class, class ID and confidence of each predicted object respectively. The box coordinate information is [xmin, ymin, w, h], i.e. the x and y coordinates and the box width and height in the top left corner.

### 27.4.3 Instance Segmentation

**MaskRCNN**

```python
paddlex.det.MaskRCNN(num_classes=81, backbone='ResNet50', with_fpn=True, aspect_ratios=[0.5, 1.0, 2.0], anchor_sizes=[32, 64, 128, 256, 512], with_dcn=False, rpn_cls_loss='SigmoidCrossEntropy', rpn_focal_loss_alpha=0.25, rpn_focal_loss_gamma=2, rcnn_bbox_loss='SmoothL1Loss', rcnn_nms='MultiClassNMS', keep_top_k=100, nms_threshold=0.5, score_threshold=0.05, softnms_sigma=0.5, bbox_assigner='BBoxAssigner', fpn_num_channels=256, input_channel=3, rpn_batch_size_per_im=256, rpn_fg_fraction=0.5, test_pre_nms_top_n=None, test_post_nms_top_n=1000)
```

Build a MaskRCNN detector. **Note that num_classes needs to be set to number of classes+background class in MaskRCNN. If an object includes humans and dogs, set num_classes to 3 so that the background class is included**

**Parameters**

- **num_classes** (int): Number of classes including the background class. It is 81 by default.
- **with_fpn** (bool): Whether to use FPN structure. It is true by default.
- **aspect_ratios** (list): Optional value of the anchor aspect ratio. It is [0.5, 1.0, 2.0] by default.
- **anchor_sizes** (list): Optional value of the anchor size. It is [32, 64, 128, 256, 512] by default.
  - **with_dcn** (bool): Whether to use deformable convolution network v2 in the backbone. Default: False.
- **rpn_cls_loss** (str): The classification loss function for RPN in a value range of [ ‘SigmoidCrossEntropy’, ‘SigmoidFocalLoss’ ]. When there are many false positives in background areas, ‘SigmoidFocalLoss’ with appropriate rpn_focal_loss_alpha and rpn_focal_loss_gamma settings may be a better option. Default: ‘SigmoidCrossEntropy’.
• **rpn_focal_loss_alpha** (float): Hyper-parameter to balance the positive and negative examples where ‘SigmoidFocalLoss’ is set as the classification loss function for RPN, Default: 0.25. If use ‘SigmoidCrossEntropy’, **rpn_focal_loss_alpha** has no effect.

• **rpn_focal_loss_gamma** (float): Hyper-parameter to balance the easy and hard examples where ‘SigmoidFocalLoss’ is set as the classification loss function for RPN, Default: 2. If use ‘SigmoidCrossEntropy’, **rpn_focal_loss_gamma** has no effect.

• **rcnn_bbox_loss** (str): The location regression loss function for RCNN in a value range of [ ‘SmoothL1Loss’ , ‘CIoULoss’ ]. Default: ‘SmoothL1Loss’.

• **rcnn_nms** (str): The non-maximum suppression(NMS) method for RCNN, in a value range of [ ‘MultiClassNMS’ , ‘MultiClassSoftNMS’ , ‘MultiClassCiouNMS’ ]. Default: ‘MultiClassNMS’. When ‘MultiClassNMS’ is set, **keep_top_k** , **nms_threshold** and **score_threshold** can be set as 100, 0.5 and 0.05 respectively. When ‘MultiClassSoftNMS’ is set, **keep_top_k** , **score_threshold** and **softnms_sigma** can be set as 300, 0.01 and 0.5 respectively. When ‘MultiClassCiouNMS’ is set, **keep_top_k** , **score_threshold** and **nms_threshold** can be set as 100, 0.05 and 0.5 respectively.

• **keep_top_k** (int): The Number of total bounding boxes to be kept per image after NMS step for RCNN. Default: 100.

• **nms_threshold** (float): The IoU threshold to filter out bounding boxes in NMS for RCNN. When **rcnn_nms** is set as ‘MultiClassSoftNMS’, **nms_threshold** has no effect. Default: 0.5.

• **score_threshold** (float): The confidence score threshold to filter out bounding boxes before nms. Default: 0.05.

• **softnms_sigma** (float): When **rcnn_nms** is set as ‘MultiClassSoftNMS’, **softnms_sigma** is used to adjust the confidence score of suppressed bounding boxes according to score = score * weights, weights = exp(-(iou * iou) / softnms_sigma). Default: 0.5.

• **bbox_assigner** (str): The method of sampling positive and negative examples during the training phase, in a value range of [ ‘BBBoxAssigner’ , ‘LibraBBoxAssigner’ ]. If the size of objects is a small portion of the image, LibraRCNN proposed a IoU-balanced sampling method to obtain more hard-negative examples, namely ‘LibraBBoxAssigner’. Default: ‘BBBoxAssigner’.

• **fpn_num_channels** (int): The number of channels of feature maps in FPN2. Default: 56.

• **input_channel** (int): The number of channels of a input image. Default: 3.

• **rpn_batch_size_per_im** (int): Total number of training examples per image for RPN. Default: 256.

• **rpn_fg_fraction** (float): The fraction of positive examples in total train examples for RPN. Default: 0.5.
• **test_pre_nms_top_n** (int): The number of predicted bounding boxes fed into NMS step. If set as None, *test_pre_nms_top_n* will be set as 6000 with a FPN or 1000 with no FPN. Default: None.

• **test_post_nms_top_n** (int): The number of predicted bounding boxes kept after NMS step. Default: 1000.

### train

```python
train(self, num_epochs, train_dataset, train_batch_size=1, eval_dataset=None, save_interval_epochs=1, log_interval_steps=20, save_dir='output', pretrain_weights='IMAGENET', optimizer=None, learning_rate=1.0/800, warmup_steps=500, warmup_start_lr=1.0 / 2400, lr_decay_epochs=[8, 11], lr_decay_gamma=0.1, metric=None, use_vdl=False, early_stop=False, early_stop_patience=5, resume_checkpoint=None)
```

MaskRCNN model training API. The function has a built-in **piecewise** learning rate attenuation policy and a **momentum** optimizer.

**Parameters**

- **num_epochs** (int): Number of training iteration epochs.

- **train_dataset** (paddlex.datasets): Training data reader.

- **train_batch_size** (int): Training data batch size. Currently, the detection supports only the single-card evaluation. The quotient of the training data batch size and the GPU quantity is a validation data batch size. It is 1 by default.

- **eval_dataset** (paddlex.datasets): Validation data reader.

- **save_interval_epochs** (int): Model saving interval (unit: number of iteration epochs). It is 1 by default.

- **log_interval_steps** (int): Training log output interval (unit: number of iterations). It is 2 by default.

- **save_dir** (str): Path where models are saved. It is ‘output’ by default.

- **pretrain_weights** (str): If it is a path, a pre-training model under the path is loaded. If it is a string ‘IMAGENET’, a model weight pre-trained on ImageNet image data is automatically downloaded. If it is a string ‘COCO’, a model weight pre-trained on the COCO dataset is automatically downloaded (Note: A COCO pre-training model for ResNet18 and HRNet_W18 is unavailable temporarily. If it is none, no pre-training model is used. It is None by default.

- **optimizer** (paddle.fluid.optimizer): Optimizer. When this parameter is none, a default optimizer is used: fluid.layers.piecewise_decay attenuation policy, fluid.optimizer. Momentum optimization method.

- **learning_rate** (float): Initial learning rate of the default optimizer. It is 0.00125 by default.
• **warmup_steps** (int): Number of steps to perform the warmup process by the default optimizer. It is 500 by default.

• **warmup_start_lr** (int): Initial learning rate of warmup of the default optimizer. It is 1.0/2400 by default.

• **lr_decay_epochs** (list): Number of learning rate attenuation epochs of the default optimizer. It is 8, 11 by default.

• **lr_decay_gamma** (float): Attenuation rate of learning rate of the default optimizer. It is 0.1 by default.

• **metric** (bool): Evaluation method during training in the value range of ‘COCO’, ‘VOC’. [It is None by default.]

• **use_vdl** (bool): Whether to use VisualDL for visualization. It is false by default.

• **early_stop** (float): Whether to use a policy for early termination of training. It is false by default.

• **early_stop_patience** (int): When a policy for early termination of training is used, training is terminated if the validation set precision continuously decreases or remains unchanged within early_stop_patience epochs. It is 5 by default.

• **resume_checkpoint** (str): When training is resumed, specify a model path saved during the last training. If it is None, training is not resumed. It is None by default.

---

**evaluate**

```
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, metric=None, return_details=False)
```

MaskRCNN model evaluation API. The index box_mmap (when metric is set to COCO) on the validation set and the corresponding seg_mmap are returned after the model is evaluated.

**Parameters**

• **eval_dataset** (paddlex.datasets): Validation data reader.

• **batch_size** (int): Validation data batch size. It is 1 by default. Currently, it must be set to 1.

• **epoch_id** (int): Number of training epochs of the current evaluation model.

• **metric** (bool): Evaluation method during training in the value range of ‘COCO’, ‘VOC’. [It is none by default. It is automatically selected according to the dataset passed by you. If it is VOCDetection, metric is ‘VOC’. If it is COCODetection, metric is ‘COCO’].

• **return_details** (bool): Whether to return detailed information. It is false by default.

**Returned value**
predict

predict(self, img_file, transforms=None)

MaskRCNN model prediction API. Note that the image processing flow during prediction can be saved in MaskRCNN.test_transforms and MaskRCNN.eval_transforms during model saving only when eval_dataset is defined during training. If eval_dataset is not defined during training, when the predict API for prediction is called, you need to redefine and pass test_transforms to the predict API.

**Parameters**

- **img_file** (str|np.ndarray): Path or numpy array of the predicted image (HWC arrangement, BGR format).

- **transforms** (paddlex.det.transforms): Data preprocessing operation.

**Returned value**

- **list**: List of prediction results. Each element in the list has a dict. The key includes ‘bbox’, ‘mask’, ‘category’, ‘category_id’ and ‘score’ which indicate the box coordinate information, mask information, class, class ID and confidence of each predicted object respectively. The box coordinate information is [xmin, ymin, w, h], i.e. the x and y coordinates and the box width and height in the top left corner. The mask information is a binary image which has the same size as the original figure. The value 1 indicates that pixels belong to the prediction class. The value 0 indicates that pixels are a background.
MaskRCNN model batch prediction API. Note that the image processing flow during prediction can be saved in `MaskRCNN.test_transforms` and `MaskRCNN.eval_transforms` during model saving only when `eval_dataset` is defined during training. If `eval_dataset` is not defined during training, when the `batch_predict` API for prediction is called, you need to redefine and pass `test_transforms` to the `batch_predict` API.

**Parameters**

- `img_file_list` (list|tuple): Images in the list (or tuple) are simultaneously predicted. Elements in the list are predicted image paths or numpy arrays (HWC arrangement, BGR format).
- `transforms` (`paddlex.det.transforms`): Data preprocessing operation.

**Returned value**

- `list`: Each element is a list which indicates prediction results of each image. Each element in the list of prediction results of each image has a dict and contains keywords: `bbox`, `mask`, `category`, `category_id` and `score` which indicate the box coordinate information, mask information, class, class ID and confidence of each predicted object respectively. The box coordinate information is `xmin`, `ymin`, `w`, `h`, i.e. the x and y coordinates and the box width and height in the top left corner. The mask information is a binary image which has the same size as the original figure. The value 1 indicates that pixels belong to the prediction class. The value 0 indicates that pixels are a background.

### 27.4.4 Semantic Segmentation

```python
import paddlex as pdx

paddlex.seg.DeepLabv3p(num_classes=2, backbone='MobileNetV2_x1.0', output_stride=16,
                        aspp_with_sep_conv=True, decoder_use_sep_conv=True, encoder_with_aspp=True, enable_decoder=True, use_bce_loss=False, use_dice_loss=False, class_weight=None, ignore_index=255, pooling_crop_size=None, input_channel=3)
```

Build a DeepLabv3p segmenter.

**Parameters**

- `num_classes` (int): Number of classes.
- `backbone` (str): DeepLabv3+ backbone network to implement the calculation of characteristic images in a value range of ‘Xception65’, ‘Xception41’, ‘MobileNetV2_x0.25’, ‘MobileNetV2_x0.5’, ‘MobileNetV2_x1.0’,
MobileNetV2_x1.5', 'MobileNetV2_x2.0', 'MobileNetV3_large_x1_0_ssld'. It is 'MobileNetV2_x1.0' by default.

- **output_stride** (int): Downsampling multiple of the backbone output characteristic image relative to the input. It is generally 8 or 16. It is 16 by default.

- **aspp_with_sep_conv** (bool): Whether the decoder module uses separable convolutions. It is true by default.

- **decoder_use_sep_conv** (bool): Whether the decoder module uses separable convolutions. It is true by default.

- **encoder_with_aspp** (bool): Whether to use an ASPP module in the encoder phase. It is true by default.

- **enable_decoder** (bool): Whether to use a decoder module. It is true by default.

- **use_bce_loss** (bool): Whether to use bce loss as a network loss function. The bce loss function can be used for two kinds of segmentation only and may be used with dice loss. It is false by default.

- **use_dice_loss** (bool): Whether to use dice loss as a network loss function. The dice loss function can be used for two kinds of segmentation only and may be used with dice loss. When both `use_bce_loss` and `use_dice_loss` are false, the cross entropy loss function is used. It is false by default.

- **class_weight** (list/str): Weight of various losses of the cross entropy loss function. When `class_weight` is a list, the length shall be `num_classes`. When `class_weight` is str, weight. lower() shall be 'dynamic'. At this moment, the corresponding weight is automatically calculated according to the proportion of all classes of pixels in each round. The weight of each class is as follows: Proportion of each class * num_classes. When `class_weight` is the default none, the weight of each class is 1, i.e. the usually used cross entropy loss function.

- **ignore_index** (int): Value ignored on a label. A pixel of which the label is `ignore_index` does not participate in the calculation of the loss function. It is 255 by default.

- **pooling_crop_size** (int): When backbone is MobileNetV3_large_x1_0_ssld, this parameter must be set to a model input size during training in W, H[ format]. For example, if the model input size is [512, 512], pooling_crop_size shall be set to 512, 512[.]. This parameter is used when an image average is obtained in the encoder module. If it is none, an average is directly calculated. If it is a model input size, an average is obtained using the avg_pool operator. It is None by default.

- **input_channel** (int): Number of input image channels. It is 3 by default.

```
train(self, num_epochs, train_dataset, train_batch_size=2, eval_dataset=None, eval_batch_size=4, save_interval_epochs=1, log_interval_steps=2, save_dir='output', pretrained_weights='IMAGENET', optimizer=None, learning_rate=0.01, lr_decay_power=0.9, use_vdl=False, sensitivities_file=None, eval_metric_loss=0.05, early_stop=False, early_stop_patience=5, resume_checkpoint=None):
```
DeepLabv3p model training API. The function has a built-in polynomial learning rate attenuation policy and a momentum optimizer.

**Parameters**

- **num_epochs** (int): Number of training iteration epochs.
- **train_dataset** (paddlex.datasets): Training data reader.
- **train_batch_size** (int): Training data batch size. It is also a validation data batch size. It is 2 by default.
- **eval_dataset** (paddlex.datasets): Evaluation data reader.
- **save_interval_epochs** (int): Model saving interval (unit: number of iteration epochs). It is 1 by default.
- **log_interval_steps** (int): Training log output interval (unit: number of iterations). It is 2 by default.
- **save_dir** (str): Path where models are saved. It is `output` by default.
- **pretrain_weights** (str): If it is a path, a pre-training model under the path is loaded. If it is a string ‘IMAGENET’, a model weight pre-trained on ImageNet image data is automatically downloaded. If it is a string ‘COCO’, a model weight pre-trained on the COCO dataset is automatically downloaded (Note: A COCO pre-training model for Xception41, MobileNetV2_x0.25, MobileNetV2_x0.5, MobileNetV2_x1.5 and MobileNetV2_x2.0 is unavailable temporarily). If it is a string ‘CITYSCAPES’, a model weight pre-trained on the CITYSCAPES dataset is automatically downloaded (Note: A CITYSCAPES pre-training model for Xception41, MobileNetV2_x0.25, MobileNetV2_x0.5, MobileNetV2_x1.5 and MobileNetV2_x2.0 is unavailable temporarily). If it is none, no pre-training model is used. It is ‘IMAGENET’ by default.
- **optimizer** (paddle.fluid.optimizer): Optimizer. When this parameter is none, the following default optimizer is used: Use the fluid.optimizer. Momentum optimization and the polynomial learning rate attenuation policy.
- **learning_rate** (float): Initial learning rate of the default optimizer. It is 0.01 by default.
- **lr_decay_power** (float): Learning rate attenuation index of the default optimizer. It is 0.9 by default.
- **use_vdl** (bool): Whether to use VisualDL for visualization. It is false by default.
- **sensitivities_file** (str): If it is a path, sensitivity information under the path is loaded to perform pruning. If it is a string ‘DEFAULT’, sensitivity information obtained on Cityscapes image data is automatically downloaded to perform pruning. If it is None, no pruning is performed. It is None by default.
- **eval_metric_loss** (float): Tolerable precision loss. It is 0.05 by default.
• **early_stop** (bool): Whether to use a policy for early termination of training. It is false by default.

• **early_stop_patience** (int): When a policy for early termination of training is used, training is terminated if the validation set precision continuously decreases or remains unchanged within early_stop_patience epochs. It is 5 by default.

• **resume_checkpoint** (str): When training is resumed, specify a model path saved during the last training. If it is None, training is not resumed. It is None by default.

### evaluate

```python
evaluate(self, eval_dataset, batch_size=1, epoch_id=None, return_details=False):
```

DeepLabv3p model evaluation API.

**Parameters**

- **eval_dataset** (paddlex.datasets): Evaluation data reader.
- **batch_size** (int): Batch size during evaluation. It is 1 by default.
- **epoch_id** (int): Number of training epochs of the current evaluation model.
- **return_details** (bool): Whether to return detailed information. It is false by default.

**Returned value**

- **dict**: When return_details is false, dict is returned. The following keywords are contained: ‘miou’, ‘category_iou’, ‘macc’, ‘category_acc’ and ‘kappa’ which indicate the average IoU, the IoU of each class, the average accuracy rate, the accuracy rate of each class and the kappa coefficient respectively.

- **tuple** (metrics, eval_details): When return_details is true, the return of dict (eval_details) is added. The following keywords are contained: ‘confusion_matrix’ which indicates the evaluation confusion matrix.

### predict

```python
predict(self, img_file, transforms=None):
```

DeepLabv3p Model inference API. Note that the image processing flow during inference can be saved in DeepLabv3p.test_transforms and DeepLabv3p.eval_transforms during model saving only when eval_dataset is defined during training. If eval_dataset is not defined during training, when the predict API for prediction is called, you need to redefine and pass test_transforms to the predict API. 

**Parameters**
- **img_file** (str|np.ndarray): Path or numpy array of the predicted image (HWC arrangement, BGR format).
- **transforms** (paddlex.seg.transforms): Data preprocessing operation.

**Returned value**

- **dict**: It contains the keywords ‘label_map’ and ‘score_map’. ‘label_map’ stores an inference result grayscale image. A pixel value indicates the corresponding class. ‘score_map’ stores a probability of each class. shape = (h, w, num_classes).

### batch_predict

```python
batch_predict(self, img_file_list, transforms=None):
```

DeepLabv3p model batch inference API. Note that the image processing flow during inference can be saved in `DeepLabv3p.test_transforms` and `DeepLabv3p.eval_transforms` during model saving only when `eval_dataset` is defined during training. If `eval_dataset` is not defined during training, when the `batch_predict` API for prediction is called, you need to redefine and pass `test_transforms` to the `batch_predict` API.

**Parameters**

- **img_file_list** (list|tuple): Images in the list (or tuple) are simultaneously predicted. Elements in the list are predicted image paths or numpy arrays (HWC arrangement, BGR format).
- **transforms** (paddlex.seg.transforms): Data preprocessing operation.

**Returned value**

- **dict**: Each element is a list which indicates inference results of each image. The inference results of each image is expressed as a dictionary. It contains the keywords ‘label_map’ and ‘score_map’. ‘label_map’ stores an inference result grayscale image. A pixel value indicates the corresponding class. ‘score_map’ stores a probability of each class. shape = (h, w, num_classes).

### overlap_tile_predict

```python
overlap_tile_predict(self, img_file, tile_size=[512, 512], pad_size=[64, 64], batch_size=32, transforms=None)
```

Sliding inference API for the DeepLabv3p model. The overlapping and non-overlapping modes are supported.
**Non-overlapping sliding window inference:** Slide on the input image using a window of fixed size. Infer an image under each window. Splice inference results of each window into inference results of the input image. The parameter `pad_size` **must be set to [0, 0] during use.**

**Overlapping sliding window inference:** In Unet’s paper, the author proposed an overlap-tile strategy to eliminate a crack feeling at the splice. In the prediction in each sliding window, a certain area is expanded around the expanded window, such as the blue part of the area in the figure below. Only the middle part of the window is predicted in the splice, for example, the yellow part area in the figure below. The pixels under the expanded area of the window located at the edge of the input image are obtained by mirroring the pixels at the edge.

Note that the image processing flow during inference can be saved in `DeepLabv3p.test_transforms` and `DeepLabv3p.eval_transforms` during model saving only when `eval_dataset` is defined during training. If `eval_dataset` is not defined during training, when the `overlap_tile_predict` API for inference is called, you need to redefine and pass `test_transforms` to the `overlap_tile_predict` API.

**Parameters**

- `img_file` (str|np.ndarray): Path or numpy array of the predicted image (HWC arrangement, BGR format).
- `tile_size` (list|tuple): Sliding window size. This area is used to splice inference results. The format is (W, H). It is 512, 512 by default.
- `pad_size` (list|tuple): Size of the area where the sliding window extends towards its surrounding. The extended area is not used to splice prediction results. The format is (W, H). It is 64, 64 by default.
- `batch_size` (int): Batch size during the batch inference on the window. It is 32 by default.
- `transforms` (paddlex.seg.transforms): Data preprocessing operation.
Returned value

- **dict**: It contains the keywords ‘label_map’ and ‘score_map’. ‘label_map’ stores an inference result grayscale image. A pixel value indicates the corresponding class. ‘score_map’ stores a probability of each class. shape = (h, w, num_classes).

**paddlex.seg.UNet**

```python
paddlex.seg.UNet(num_classes=2, upsample_mode='bilinear', use_bce_loss=False, use_dice_loss=False, class_weight=None, ignore_index=255, input_channel=3)
```

Build a UNet segmenter.

**Parameters**

- **num_classes** (int): Number of classes.
- **upsample_mode** (str): Upsampling mode used during the UNet decoding. When the value is ‘bilinear’, a bilinear difference value is used to perform upsampling. When other options are input, a deconvolution is used to perform upsampling. It is ‘bilinear’ by default.
- **use_bce_loss** (bool): Whether to use bce loss as a network loss function. The bce loss function can be used for two kinds of segmentation only and may be used with dice loss. It is false by default.
- **use_dice_loss** (bool): Whether to use dice loss as a network loss function. The dice loss function can be used for two kinds of segmentation only and may be used with bce loss. When both use_bce_loss and use_dice_loss are false, the cross entropy loss function is used. It is false by default.
- **class_weight** (list/str): Weight of various losses of the cross entropy loss function. When class_weight is a list, the length shall be num_classes. When class_weight is str, weight.lower() shall be ‘dynamic’. At this moment, the corresponding weight is automatically calculated according to the proportion of all classes of pixels in each round. The weight of each class is as follows: Proportion of each class * num_classes. When class_weight is the default none, the weight of each class is 1, i.e. the usually used cross entropy loss function.
- **ignore_index** (int): Value ignored on a label. A pixel of which the label is ignore_index does not participate in the calculation of the loss function. It is 255 by default.
- **input_channel** (int): Number of input image channels. It is 3 by default.
- The description of the train API for training is the same as the train API of the DeepLabv3p model
- The description of the evaluate API for evaluation is the same as the evaluate API of the DeepLabv3p model
- The description of the predict API for inference is the same as the predict API of the DeepLabv3p model
• The description of the batch_predict API for batch prediction is the same as the predict API of the DeepLabv3p model

• The overlap_tile_predict API for sliding window prediction is the same as the poverlap_tile_predict API of the DeepLabv3p model

**paddlex.seg.HRNet**

```python
paddlex.seg.HRNet(num_classes=2, width=18, use_bce_loss=False, use_dice_loss=False,
...class_weight=None, ignore_index=255, input_channel=3)
```

Build an HRNet segmenter.

**Parameters**

- **num_classes** (int): Number of classes.
- **width** (int|str): Number of channels in the characteristic layer in a high-resolution branch. It is 18 by default. The optional values are 18, 30, 32, 40, 44, 60, 64, ‘18_small_v1’. ‘18_small_v1’ is the lightweight version of 18.[
- **use_bce_loss** (bool): Whether to use bce loss as a network loss function. The bce loss function can be used for two kinds of segmentation only and may be used with dice loss. It is false by default.
- **use_dice_loss** (bool): Whether to use dice loss as a network loss function. The dice loss function can be used for two kinds of segmentation only and may be used with bce loss. When both use_bce_loss and use_dice_loss are false, the cross entropy loss function is used. It is false by default.
- **class_weight** (list|str): Weight of various losses of the cross entropy loss function. When class_weight is a list, the length shall be num_classes. When class_weight is str, weight.lower() shall be ‘dynamic’. At this moment, the corresponding weight is automatically calculated according to the proportion of all classes of pixels in each round. The weight of each class is as follows: Proportion of each class * num_classes. When class_weight is the default none, the weight of each class is 1, i.e. the usually used cross entropy loss function.
- **ignore_index** (int): Value ignored on a label. A pixel of which the label is ignore_index does not participate in the calculation of the loss function. It is 255 by default.
- **input_channel** (int): Number of input image channels. It is 3 by default.

• The description of the train API for training is the same as the train API of the DeepLabv3p model

• The description of the evaluate API for evaluation is the same as the evaluate API of the DeepLabv3p model

• The description of the predict API for inference is the same as the predict API of the DeepLabv3p model
The description of the batch_predict API for batch prediction is the same as the predict API of the DeepLabv3p model.

The overlap_tile_predict API for sliding window inference is the same as the poverlap_tile_predict API of the DeepLabv3p model.

```
paddlex.seg.FastSCNN
```

Build a FastSCNN segmenter.

**Parameters**

- **num_classes** (int): Number of classes.
- **use_bce_loss** (bool): Whether to use bce loss as a network loss function. The bce loss function can be used for two kinds of segmentation only and may be used with dice loss. It is false by default.
- **use_dice_loss** (bool): Whether to use dice loss as a network loss function. The dice loss function can be used for two kinds of segmentation only and may be used with bce loss. When both use_bce_loss and use_dice_loss are false, the cross entropy loss function is used. It is false by default.
- **class_weight** (list/str): Weight of various losses of the cross entropy loss function. When class_weight is a list, the length shall be num_classes. When class_weight is str, weight.lower() shall be ‘dynamic’. At this moment, the corresponding weight is automatically calculated according to the proportion of all classes of pixels in each round. The weight of each class is as follows: Proportion of each class * num_classes. When class_weight is the default none, the weight of each class is 1, i.e. the usually used cross entropy loss function.
- **ignore_index** (int): Value ignored on a label. A pixel of which the label is ignore_index does not participate in the calculation of the loss function. It is 255 by default.
- **multi_loss_weight** (list): Loss weight on multiple branches. The default is to calculate a loss on one branch, i.e. the default is [1.0]. A loss on two or three branches can also be calculated and the weight is arranged in a sequence of [fusion_branch_weight, higher_branch_weight, lower_branch_weight]. fusion_branch_weight is the loss weight on the branch after the spatial detail branch and the global context branch are blended. higher_branch_weight is the loss weight on the spatial detail branch. lower_branch_weight is the loss weight on the global context branch. If higher_branch_weight and lower_branch_weight are not set, a loss on these two branches will not be calculated.
- **input_channel** (int): Number of input image channels. It is 3 by default.

The description of the train API for training is the same as the train API of the DeepLabv3p model.
• The description of the evaluate API for evaluation is the same as the evaluate API of the DeepLabv3p model
• The description of the predict API for inference is the same as the predict API of the DeepLabv3p model
• The description of the batch_predict API for batch prediction is the same as the predict API of the DeepLabv3p model
• The overlap_tile_predict API for sliding window inference is the same as the poverlap_tile_predict API of the DeepLabv3p model

27.5 Model compression

27.5.1 paddlex.slim.cal_params_sensitivities

Calculate parameter sensitivity

```
paddlex.slim.cal_params_sensitivities(model, save_file, eval_dataset, batch_size=8)
```

Calculate sensitivity of pruned parameters in the model on the validation set and save sensitivity information in the `save_file` file

1. Obtain the name of the pruned convolutions Kernel in the model.
2. Calculate sensitivity of pruned convolutions Kernel at different pruning rates.

[Note] Convolution sensitivity is a model precision loss after the model is pruned according to a pruning rate. Select an appropriate sensitivity and thus determine the list of parameters to be pruned for the final model and the pruning rate corresponding to each pruned parameter.

View an example

Parameters

• **model** (paddlex.cls.models/paddlex.det.models/paddlex.seg.models): Model loaded by paddlex.
• **save_file** (str): Storage path of the calculated parameter sensitivity file.
• **eval_dataset** (paddlex.datasets): Reader for evaluated datasets.
• **batch_size** (int): batch_size size during evaluation.

27.5.2 paddlex.slim.export_quant_model

Export a quantitative model
Export a quantitative model. This API implements the Post Quantization quantization method. An incoming test dataset is required. In addition, \texttt{batch\_size} and \texttt{batch\_num} need to be set. The calculation results of sample data of which the quantity is \texttt{batch\_size} * \texttt{batch\_num} are used as statistic information to complete the quantization of the model during quantization.

**Parameters**

- \texttt{model} (\texttt{paddlex.cls.models/paddlex.det.models/paddlex.seg.models}): Model loaded by paddlex.
- \texttt{test\_dataset} (\texttt{paddlex.dataset}): Test dataset.
- \texttt{batch\_size} (int): Batch data size during the forward calculation.
- \texttt{batch\_num} (int): Batch data quantity during the forward calculation.
- \texttt{save\_dir} (str): Directory where a quantized model is saved.
- \texttt{cache\_dir} (str): Temporary storage directory of statistic data during quantization.

**Usage example**

Click to download the \texttt{model} and \texttt{dataset} in the following example

```python
import paddlex as pdx
model = pdx.load_model('vegetables_mobilenet')
test_dataset = paddlex.datasets.ImageNet(
    data_dir='vegetables_cls',
    file_list='vegetables_cls/train_list.txt',
    label_list='vegetables_cls/labels.txt',
    transforms=model.eval_transforms)
pdx.slim.export_quant_model(model, test_dataset, save_dir='./quant Mobilenet')
```

### 27.6 Visualization of predicted results

PaddleX provides a series of visualization functions for model prediction and result analysis.

#### 27.6.1 paddlex.det.visualize

Visualization of prediction results for object detection/instance segmentation

```python
paddlex.det.visualize(image, result, threshold=0.5, save_dir='./', color=None)
```

Visualize a box and mask predicted by object detection/instance segmentation models in the original figure.
Parameters

- **image** (str|np.ndarray): File path or numpy array of the original figure (HWC arrangement, BGR format).
- **result** (str): Model prediction results.
- **threshold** (float): Score threshold. Any box of which the confidence is smaller than this threshold is filtered and is not visualized. It is 0.5 by default.
- **save_dir** (str): Path where visualized results are saved. If this parameter is None, it indicates that the path does not exist and that this function returns visualized results in the form of np.ndarray. If this parameter is set to a directory path, this function saves visualized results in this directory. It is ‘.’ by default.
- **color** (list|tuple|np.array): List of BGR color values of all categories, shape of the list is required to be N x 3 where N is the number of categories and color value should be within [0, 255] range. If set as None, the color values are generated automatically. Default: None.

Usage Example

Click to download the model in the following example

```python
import paddlex as px
model = px.load_model('xiaoduxiong_epoch_12')
result = model.predict('./xiaoduxiong_epoch_12/xiaoduxiong.jpeg')
px.det.visualize('./xiaoduxiong_epoch_12/xiaoduxiong.jpeg', result, save_dir='./')
# Prediction results are saved in ./visualize_xiaoduxiong.jpeg
```

27.6.2 paddlex.seg.visualize

Visualization of prediction results for semantic segmentation models

```python
paddlex.seg.visualize(image, result, weight=0.6, save_dir='./', color=None)
```

Visualize a mask predicted by semantic segmentation models in the original figure.

Parameters

- **image** (str|np.ndarray): File path or numpy array of the original figure (HWC arrangement, BGR format).
- **result** (str): Model prediction results.
- **weight** (float): Weight factor of mask visualized results and the original figure. The weight parameter indicates a weight of the original figure. It is 0.6 by default.

- **save_dir** (str): Path where visualized results are saved. If this parameter is None, it indicates that the path does not exist and that this function returns visualized results in the form of np.ndarray. If this parameter is set to a directory path, this function saves visualized results in this directory. It is `. /` by default.

- **color** (list): List of all classes of BGR color values. For example, it can be set to 255, 255, 255, 0, 0, 0[ in the case of two classes]. It is None by default, indicating that the list of colors generated by default is used.

**Usage example**

Click to download the model and test image in the following example

```python
import paddlex as pdx
model = pdx.load_model('cityscape_deeplab')
result = model.predict('city.png')
pdx.det.visualize('city.png', result, save_dir='. /')
# Prediction results are saved in . /visualize_city.png
```

### 27.6.3 paddlex.det.draw_pr_curve

Visualization of accuracy rate versus recall rate for object detection/instance segmentation

```python
paddlex.det.draw_pr_curve(eval_details_file=None, gt=None, pred_bbox=None, pred_mask=None, iou_thresh=0.5, save_dir='. /')
```

Visualize the relation of accuracy rate versus recall rate of each class in the evaluation results of the object detection/instance segmentation model as well as the relation of recall rate versus confidence threshold.

Note: The `eval_result.json` file is contained in any model directory saved by PaddleX during training. This file path is passed to the `eval_details_file` parameter. A PR curve graph of the corresponding model on the validation set can be obtained by setting `iou_threshold`.

**Parameters**

- **eval_details_file** (str): Path where model evaluation results including true value information and prediction results are saved. It is None by default.

- **gt** (list): True value information of the dataset. It is None by default.
• **pred_bbox** (list): Predicted box by the model on the dataset. It is None by default.

• **pred_mask** (list): Predicted mask by the model on the dataset. It is None by default.

• **iou_thresh** (float): IoU threshold when the predicted box or mask is determined to be genuine yang. It is 0.5 by default.

• **save_dir** (str): Path where visualized results are saved. It is `/` by default.

**Note:** **eval_details_file** has a higher priority. True value information and prediction results are extracted from **eval_details_file** for analysis as long as **eval_details_file** is not None. When **eval_details_file** is None, `gt`, pred_mask and pred_mask are used to make an analysis.

**Usage example**

Click to download the **model** and **dataset** in the following example

Method 1: Analyze an evaluation result file saved in the model folder during training, e.g. **eval_details.json** in the model ([https://bj.bcebos.com/paddlex/models/insect_epoch_270.zip](https://bj.bcebos.com/paddlex/models/insect_epoch_270.zip)).

```python
import paddlex as pdx
eval_details_file = 'insect_epoch_270/eval_details.json'
pdx.det.draw_pr_curve(eval_details_file, save_dir='./insect')
```

Method 2: Analyze evaluation results returned by the model evaluation function.

```python
import os
# Choose to use Card 0
os.environ['CUDA_VISIBLE_DEVICES'] = '0'

from paddlex.det import transforms
import paddlex as pdx

model = pdx.load_model('insect_epoch_270')
eval_dataset = pdx.datasets.VOCDetection(
    data_dir='insect_det',
    file_list='insect_det/val_list.txt',
    label_list='insect_det/labels.txt',
    transforms=model.eval_transforms)
metrics, evaluate_details = model.evaluate(eval_dataset, batch_size=8, return_...
details=True)

``gt`` = evaluate_details['gt']
``bbox`` = evaluate_details['bbox']
pdx.det.draw_pr_curve(gt=gt, pred_bbox=bbox, save_dir='./insect')
```
Relations of accuracy rate versus recall rate and recall rate versus confidence threshold of each class of the predicted box are visualized as follows:

27.6.4 `paddlex.slim.visualize`

Visualization analysis of model pruning proportions

```
paddlex.slim.visualize(model, sensitivities_file, save_dir='./')
```

Model pruning proportions can be analyzed under different `eval_metric_loss` parameters using this API. The vertical axis of visualized results shows `eval_metric_loss` parameter values and the horizontal axis shows pruning proportions of the corresponding models. `eval_metric_loss` (convolution sensitivity) is a model precision loss after the model is pruned according to a pruning rate.

**Parameters**

- `model` (paddlex.cv.models): Use a model loaded by PaddleX.
- `sensitivities_file` (str): Calculated parameter sensitivity information file of model parameters on the validation set.
• **save_dir**(str): Path where visualized results are saved. It is the current directory by default.

**Usage example**

Click to download the **model** and **sensitivities_file** in the example

```python
import paddlex as pdx
model = pdx.load_model('vegetablesmobilenet')
pdx.slim.visualize(model, 'mobilenetv2.sensitivities', save_dir='./')
# Visualized results are saved in ./sensitivities.png
```

27.6.5 paddlex.transforms.visualize

**Visualization of data preprocessing/enhancement process**

```python
paddlex.transforms.visualize(dataset,
    img_count=3,
    save_dir='vdl_output')
```

Visualize intermediate results of data preprocessing/enhancement. Intermediate results can be viewed using VisualDL:

2. Open https://0.0.0.0:8001 on the browser and click [Sample Data-Image] in the page. 0.0.0.0 indicates local access. In case of remote services, change it to the corresponding machine IP address

**Parameters**

- **dataset** (paddlex.datasets): Dataset reader.
- **img_count**(int): Number of images which require data preprocessing/enhancement. It is 3 by default.
- **save_dir**(str): Path where logs are saved. It is ‘vdl_output’ by default.

27.7 Model interpretability

Currently, PaddleX supports visual interpretation of image classification results and supports LIME and NormLIME interpretability algorithms.

27.7.1 paddlex.interpret.lime

**Visualization of LIME interpretability results**
The LIME algorithm is used to visualize the interpretability of model prediction results. LIME represents model-independent local interpretability and can interpret any model. The idea of LIME is as follows: By taking an input sample as a center and randomly taking a sample in the space near it, each sample obtains a new output through the original model, so a series of inputs and the corresponding outputs are obtained. LIME fits this mapping relation using a simple and interpretable model (such as a linear regression model) to get the weight of each input dimension to interpret the model.

**Note:** Currently, the visualization of interpretability results supports classification models only.

### Parameters

- **img_file** (str): Prediction image path.
- **model** (paddlex.cv.models): Model in paddlex.
- **num_samples** (int): Number of samples that LIME uses for linear learning models. It is 3000 by default.
- **batch_size** (int): Prediction data batch size. It is 50 by default.
- **save_dir** (str): Storage path of visualized interpretability results (saved as a png file) and intermediate files.

### Visualization effects

### Usage example

For the visualization process of prediction interpretability results, refer to [codes](https://github.com/PaddlePaddle/PaddleX/blob/develop/tutorials/interpret/lime.py).

**27.7.2 paddlex.interpret.normlime**

Visualization of NormLIME interpretability results
The NormLIME algorithm is used to visualize the interpretability of model prediction results. NormLIME uses a certain number of samples to make a global interpretation. A simplified method is used here because the NormLIME calculations are large: Use a certain number of test samples (Currently, all test samples are used by default) to perform feature extractions on each sample and map them to the same feature space. By using this feature as an input and the model output as an output, use linear regression to fit it to obtain a global input and output relation. When a test sample is interpreted, use NormLIME global interpretation to filter LIME results so that the final visualized results are more stable.

**Note:** Currently, the visualization of interpretability results supports classification models only.

**Parameters**

- **img_file** (str): Prediction image path.
- **model** (paddlex.cv.models): Model in paddlex.
- **dataset** (paddlex.datasets): Dataset reader. It is none by default.
- **num_samples** (int): Number of samples that LIME uses for linear learning models. It is 3000 by default.
- **batch_size** (int): Prediction data batch size. It is 50 by default.
- **save_dir** (str): Storage path of visualized interpretability results (saved as a png file) and intermediate files.
- **normlime_weights_file** (str): NormLIME initialization filename. If it does not exist, it is calculated once and saved in this path. If it exists, it is directly loaded.

**Note:** **dataset** reads a dataset. This dataset shall not be too large, otherwise the calculation time is long. However, all data categories shall be contained. Currently, the visualization of NormLIME interpretability results supports classification models only.

**Usage example**

For the visualization process of prediction interpretability results, refer to [codes](https://github.com/PaddlePaddle/PaddleX/blob/develop/tutorials/interpret/normlime.py).
28.1 Image classification model

The model accuracy rate in the table is based on the ImageNet dataset, and the symbol - in the table indicates that the metric has not been tested yet:

- The CPU evaluation environment is based on Snapdragon 855 (SD855).
- The GPU evaluation environment is based on a T4 machine with 500 runs in FP32+TensorRT configuration (removing warmup time for the first 10 runs).

28.1.1 Mobile Series

28.1.2 Other series

28.2 Object detection model

The model accuracy BoxAP in the table is obtained by testing the MSCOCO validation set through the evaluate() interface. The symbol - indicates that the relevant metrics have not been yet tested and the prediction time is obtained in the following environments:

- Test environment:
  - CUDA 9.0
  - CUDNN 7.5
- PaddlePaddle v1.6
- TensorRT-5.1.2.2
- GPU: Tesla V100

• Test mode:
  - To make it easier to compare the reasoning speed of different models, the input is the same size as the image: 3x640x640.
  - Batch Size=1
  - Remove the first 10 rounds of warmup time and test the average time in the unit of ms/image for 100 rounds, including the time to copy the input data to the GPU, computation time, and time to copy data to CPU.
  - Use the Fluid C++ prediction engine. Enable the FP32 TensorRT configuration.
  - Start the test. FLAGS_cudnn_exhaustive_search=True: search for the convolutional algorithm using the exhaustive method.

28.3 Instance segmentation model

The model precision BoxAP/ MaskAP in the table is obtained by testing the MSCOCO validation set through the evaluate() interface. The symbol - indicates that the relevant metrics have not been tested yet, and the prediction time is obtained in the following environment:

• Test environment:
  - CUDA 9.0
  - CUDNN 7.5
  - PaddlePaddle v1.6
  - TensorRT-5.1.2.2
  - GPU: Tesla V100

• Test mode:
  - To make it easier to compare the reasoning speed of different models, the input is the same size as the image: 3x640x640.
  - Batch Size=1
  - Remove the first 10 rounds of warmup time and test the average time in the unit of ms/image for 100 rounds, including the time to copy the input data to the GPU, computation time, and time to copy data to CPU.
  - Use the Fluid C++ prediction engine. Enable the FP32 TensorRT configuration.
– Start the test. FLAGS_cudnn_exhaustive_search=True: search for the convolutional algorithm using the exhaustive method.

### 28.4 Semantic segmentation model

The following metrics are tested on the MSCOCO validation set. The symbol - in the table indicates that the metrics have not been tested yet.

The following metrics are tested on the Cityscapes validation set. The symbol - in the table indicates that the metrics have not been tested yet.
PaddleX has logs and metrics that are fed back during model training and evaluation. This document describes the meanings of logs and metrics.

### 29.1 Generic training statistics

The output log information of all PaddleX models during training contains six common statistics that are used to assist users in model training, for example, training log for segmentation models. See the following:**

```
[TRAIN] Epoch=4/20, Step=62/66, loss=0.007226, lr=0.008215, time_each_step=0.41s, eta=0:9:44
[TRAIN] Epoch=4/20, Step=64/66, loss=0.012199, lr=0.008201, time_each_step=0.41s, eta=0:9:43
[TRAIN] Epoch=4/20, Step=66/66, loss=0.003387, lr=0.008187, time_each_step=0.41s, eta=0:9:42
[TRAIN] Epoch 4 finished, loss=0.007828, lr=0.008414.
```

The meaning of each field is as follows:

In addition to the above general fields, there are other fields in the logs of the different models. For the meanings of these fields, see the description of each task model.

### 29.2 Evaluate generic statistical information

All models in PaddleX are evaluated and saved at regular intervals during the training process according to the `save_interval_epochs` parameter set by the user. For example, the evaluation log for the classification model is shown in the figure below.****
The first line in the above figure indicates that the number of samples in the validation dataset is 240. It takes 8 iteration steps to evaluate all the validation data; the fifth line indicates that the second round of the model completes the saving operation; the sixth line indicates: In the current saving model, the second-round model has the optimal metrics in the validation set (see $\text{acc}_1$ for the classification task, and the value of $\text{acc}_1$ is 0.258333). The optimal model is saved in the $\text{best\_model}$ directory.

### 29.3 Classify specific statistic information

#### 29.3.1 Training log field

The training log for the classification task includes two specific fields $\text{acc}_1$ and $\text{acc}_5$, in addition to generic statistics.

Note: The accuracy is calculated for a single image: the prediction scores of the model on each category are sorted from in the descending order, the top k prediction categories are taken out. If these k prediction categories contain the true value, the image is considered to be correctly classified.

The $\text{acc}_1$ in Line 1 of the above figure represents the average top1 accuracy of the training samples participating in the current iterations, and the higher values indicates the better model. $\text{acc}_5$ represents the average top5 (topn if the number of categories n is less than 5) accuracy of the training samples participating in the current iteration of steps, and the higher value indicates the better model. The $\text{loss}$ in line 4 represents the average loss function value for the entire training set. $\text{acc}_1$ indicates the average top1 accuracy for the entire training set, and $\text{acc}_5$ indicates the average top5 accuracy for the entire training set.

#### 29.3.2 Evaluate the log field
acc1 in Line 3 of the above figure represents the average top1 accuracy of the entire validation set, and acc5 represents the average top5 accuracy of the entire validation set.

29.4 Detect specific statistical information

29.4.1 Training log field

YOLOv3

YOLOv3’s training log includes only training generic statistics (see Training Generic Statistics above).

```
[TRAIN] Epoch=1/270, Step=204/211, lr=2.5e-05, time_each_step=0.41s, eta=7:24:50
[TRAIN] Epoch=1/270, Step=206/211, loss=49.226215, lr=2.6e-05, time_each_step=0.41s, eta=7:22:34
[TRAIN] Epoch=1/270, Step=208/211, loss=66.17791, lr=2.6e-05, time_each_step=0.41s, eta=7:30:9
[TRAIN] Epoch=1/270, Step=210/211, loss=61.262436, lr=2.6e-05, time_each_step=0.42s, eta=7:35:28
[TRAIN] Epoch 1 finished, loss=59.357239, lr=1.3e-05.
```

The loss in Line 5 in the above figure represents the average loss function (loss) value for the entire training set.

FasterRCNN

FasterRCNN’s training log includes, in addition to generic statistics, loss_cls, loss_bbox, loss_rpn_cls, and loss_rpn_bbox. The meanings of the fields are as follows:

```
2020-04-26 17:22:46 (INFO) [TRAIN] Epoch=1/270, Step=204/211, loss=0.10885, loss_cls=0.379884, loss_bbox=0.272749, loss_rpn_cls=0.044654, loss_rpn_bbox=0.418138, lr=0.
2020-04-26 17:22:46 (INFO) [TRAIN] Epoch=1/270, Step=206/211, loss=0.10885, loss_cls=0.379884, loss_bbox=0.272749, loss_rpn_cls=0.044654, loss_rpn_bbox=0.418138, lr=0.
2020-04-26 17:22:46 (INFO) [TRAIN] Epoch=1/270, Step=208/211, loss=0.308001, loss_cls=0.308001, loss_bbox=0.246939, loss_rpn_cls=0.021446, loss_rpn_bbox=0.4324, lr=0.
```

In the above figure, loss, loss_cls, loss_bbox, loss_rpn_cls, and loss_rpn_bbox in Line 1 are all the loss values of the training samples participating in the current iteration step, while Line 7 is the loss function value for the entire training set.

MaskRCNN

MaskRCNN’s training log includes, in addition to generic statistics, loss_cls, loss_bbox, loss_mask, loss_rpn_cls, and loss_rpn_bbox. The meanings of the fields are as follows:

```
2020-04-26 17:31:16 (INFO) [TRAIN] Epoch=1/270, Step=216/211, loss=0.38725, loss_cls=0.379884, loss_bbox=0.272749, loss_mask=0.197976, loss_rpn_cls=0.020088, loss_rpn_bbox=0.418138, lr=0.
2020-04-26 17:31:16 (INFO) [TRAIN] Epoch=1/270, Step=218/211, loss=0.38725, loss_cls=0.379884, loss_bbox=0.272749, loss_mask=0.197976, loss_rpn_cls=0.020088, loss_rpn_bbox=0.418138, lr=0.
2020-04-26 17:31:16 (INFO) [TRAIN] Epoch=1/270, Step=220/211, loss=0.38725, loss_cls=0.379884, loss_bbox=0.272749, loss_mask=0.197976, loss_rpn_cls=0.020088, loss_rpn_bbox=0.418138, lr=0.
```

In the above figure, loss, loss_cls, loss_bbox, loss_mask, loss_rpn_cls, and loss_rpn_bbox in Line 1 are all loss values for the training samples participating in the current iteration step, and line 7 is the loss function value for the entire training set.
29.4.2 Evaluate the log field

Two evaluation standards can be used for detection: VOC evaluation standard and COCO evaluation standard.

**VOC evaluation standard**

![Image showing VOC evaluation example]

Note: map is the average value of the average accuracy, that is, the average of the area under the accuracy-recall curve for each category when Intersection Over Union (IoU) is set to 0.5.

Line 3 of the bbox_map in the above figure shows the average accuracy of the entire validation set for the detection task.

**COCO evaluation standard**

Note: for the COCO evaluation metrics, see COCO official website for details. PaddleX mainly feeds back mmAP. That is, AP at IoU=.50:.05:.95 metric is the average of the mAP at each IoU threshold.

The COCO-formatted dataset can be used to train not only object detection models, but also training instance segmentation models. In object detection, PaddleX mainly feeds the bbox_mmAP metric against the detection box. For the instance segmentation, it also includes the seg_mmAP metric against the Mask. As shown below, the first log screenshot shows the evaluation result of the object detection, and the second log screenshot shows the evaluation result of instance segmentation.

![Image showing COCO evaluation example]

The bbox_mmap marked by the red box in the above figure represents the average accuracy of the detection boxes for the entire validation set.
The bbox_mmap and seg_mmap marked in the red box in the above figure represent the average accuracy of the detection box averaged over the entire validation set, and the average accuracy of the Mask averaged over the entire validation set, respectively.

### 29.5 Segmentation of specific statistics

#### 29.5.1 Training log field

Training logs for semantic segmentation include only training generic statistics (see Training Generic Statistics above).
29.5.2 Evaluate the log field

The evaluation log for semantic partitioning includes the fields \texttt{miou}, \texttt{category_iou}, \texttt{macc}, \texttt{category_acc}, and \texttt{kappa}. The meanings are as follows:

```
2020-04-26 17:58:55 [INFO]  Start to evaluating(total_samples=76, total_steps=19)...  
2020-04-26 17:58:55 [INFO]  [VAL] Finished, [epoch=4, miou=0.901114, category_iou=[0.9961154 0.886131], macc=0.996177, category_acc=[0.99748952 0.9244862], kappa=0.89
```

There is a frequently occurring problem with deep learning: the model is currently a black box, and it is almost impossible to perceive its internal workings, and the reliability of the prediction results has been questioned. For this reason, PaddleX provides two algorithms to perform interpretable research on image classification prediction results: LIME and NormLIME.

### 30.1 LIME

The Local interpretable model-agnostic explanations (LIME) indicates a model-independent local interpretability. The main steps in its implementation are as follows.

1. Acquire the image’s superpixels.

2. Centered on the input sample, take random samples in the space around it. Each sample is a random mask of the superpixel in the sample (the weight of each sample is inversely proportional to the distance of that sample from the original sample).

3. Each sample has a new output through the prediction model. In this way, series of inputs $X$ and corresponding output $Y$ are obtained.

4. $X$ is converted to a superpixel feature $F$. A simple and interpretable $Model$ (here using Ridge regression) is used to fit the mapping between $F$ and $Y$.

5. The $Model$ obtains the weight of each input dimension of $F$ (each dimension represents a superpixel).
For the usage of LIME, see code examples and API introduction. The setting of the `num_samples` parameter is especially important. It indicates the number of random samples in Step 2. If it is set to a value, too small the stability of the interpretable result is affected. If it is set to a value, too large, it takes a long time in step 3. The parameter `batch_size` indicates that it takes longer time in step 3 if it is set too small, and the upper limit is determined by the computer configuration.

The visualization result of the final LIME interpretable algorithm is as follows: the green area represents the positive superpixels, the red area represents the negative superpixels, and “First n superpixels” represents the first n superpixels with higher weight (calculated from step 5).

### 30.2 NormLIME

NormLIME is an improvement on LIME, where the interpretation of LIME is local and a specific interpretation for the current sample. NormLIME is a global interpretation of the current sample using a certain number of samples, with a certain noise reduction effect. Its implementation steps are as follows:

1. Download the Kmeans model parameters and the first three layers of the ResNet50_vc network parameters. (The parameters of ResNet50_vc are the parameters of the network obtained by training on ImageNet; using ImageNet images as a dataset, each image extracts the average feature on the corresponding super pixel position and the feature on the center of mass from the third layer output of ResNet50_vc, and the Kmeans model is obtained through training here)

2. Calculate the weight information of normlime using the data in the test set (if no test set is available, use the validation set instead). For each image: (1) Get the image’s superpixel. (2) Use ResNet50_vc to obtain the feature of the third layer. Combine the prime and mean features $F$ for each superpixel location. (3) Use $F$ as input to the Kmeans model to calculate the clustering center for each superpixel location. (4) Use the trained classification model, and predict the label for that image. For all images: (1) Take a vector consisting of information about the clustering centers of each image (set to 1 if a cluster center appears on the way to stamping and 0 otherwise) as an input. The predicted label is the output. Construct the logic regression function $\text{regression}\_\text{func}$. (2) Based on the $\text{regression}\_\text{func},$
obtain the weights of each cluster center under different categories, and normalize the weights.

3. Use the Kmeans model to obtain the clustering center for each superpixel of the image to be visualized.

4. A new image is constructed by randomly masking the superpixels of the image to be visualized.

5. Predict the label for each constructed image using a prediction model.

6. According to the weight information of normlime, each superpixel is given a different weight. The highest weight is selected as the final weight to interpret the model.

For the usage of NormalLIME, refer to the code example and api description. The parameter num_samples is especially important, it indicates the number of random samples in step 2. If it is set too small, the stability of the interpretable result is affected. If it is set too large, it takes a long time in step 3. The parameter batch_size indicates that if it is set too small, it takes a long time in step 3, and the upper limit is decided in the machine configuration. The dataset is the data constructed by the test set or validation set.

The visualization results of the final NormLIME interpretable algorithm are as follows: the green area represents the positive superpixels, the red area represents the negative superpixels, and “First n superpixels” represents the first n superpixels with larger weight (calculated in step 5).
last row represents the result of multiplying the weights of LIME and NormLIME corresponding superpixels.
In the model training of PaddleX, the networking for downloading is required in the following two scenarios:

1 Training models: When the user is not configured with `pretrain_weights` for customizing the pre-trained model weights, at this time the PaddleX automatically performs the networking to download the pre-trained models on the standard dataset.

2 Model cropping training: When the user is not configured with `sensitivities_file` for customizing the parameter sensitivity information file to set `sensitivities_file` to DEFAULT, at this time the PaddleX automatically performs the networking to download the parameter sensitivity information file computed by the model on the standard dataset.

### 31.1 PaddleX Python API Offline Training

Download all the pre-trained models of PaddleX with by running the following codes (total size is 7.5G):

```python
from paddlex.cv.models.utils.pretrain_weights import image_pretrain from paddlex.cv.
models.utils.pretrain_weights import coco_pretrain from paddlex.cv.models.utils.
pretrain_weights import cityscapes_pretrain import paddlehub as hub save_dir = '/home/
work/paddlex_pretrain' for name, url in image_pretrain.items(): hub.download(name, save_.
dir) for name, url in coco_pretrain.items(): hub.download(name, save_dir) for name, url,
in cityscapes_pretrain.items(): hub.download(name, save_dir)
```

The user executes the above code on an internet-connected computer. All pre-trained models are downloaded.
to the specified `save_dir` (/home/work/paddlex_pretrain in the code example). When using the PaddleX training code through the Python code, you only need to import paddlex and configure the following parameters. The model first searches for the pre-trained model that has been downloaded in this directory during training.

```python
import paddle as pdx
dpx.pretrain_dir = '/home/work/paddlex_pretrain'
```

31.2 PaddleX GUI offline training

After PaddleX GUI is started, you need to set the workspace. Assume that the workspace is D:\Paddle_X_Workspace. To perform the offline training, you need to download all the following files manually (decompression is not required after downloading) to D:\Paddle_X_Workspace\pretrain\directory. After that, networking is no longer needed for training models.

31.2. PaddleX GUI offline training
**v1.3.0 2020.12.20**

- **Model Update**
  - Add to image classification model ResNet50_vd 100,000 classification pre-training models.
  - Add to object detection model FasterRCNN new model clipping support.
  - Add to target detection model multi-channel image training support.

- **Model Deployment Update**
  - Fix some bugs in OpenVINO deployment C++ code.
  - Add to Raspberry Pi deployment Arm V8 support.

- **Industrial Case Update**

- Add a industrial quality inspection case and provide industrial quality inspection solutions based on GPU and CPU deployment scenarios, as well as optimization strategies related to quality inspection
  Details link

- **Add RestFUL API module** Add RestFUL API module, through which developers can quickly develop training platform based on PaddleX.

- Add HTML Demo based on RestFUL API Details link

- Add Remote visual client based on RestFUL API Details link Add model deployment scheme through OpenVINODetails link

**v1.2.0 2020.09.07**
- **Model Update**
  - Add the most practical object detection model PP-YOLO in the industry. Deeply considering the double requirements for precision and speed in the industrial application, the COCO dataset precision is 45.2% and the Tesla V100 inference speed is 72.9 FPS. [Details link](#)
  - Add to FasterRCNN, MaskRCNN, YOLOv3, DeepLabv3p and other models a built-in COCO dataset pre-training model which applies to fine-tuned training of small datasets.
  - Add to object detection models FasterRCNN and MaskRCNN backbone HRNet_W18 which applies to application scenarios having high requirements for details inference. [Details link](#)
  - Add backbone MobileNetV3_large_ssld to the semantic segmentation model DeepLabv3p. The model volume is 9.3 MB and the Cityscapes dataset precision still is 73.28%. [Details link](#)

- **Model Deployment Update**
  - Add a model inference acceleration deployment solution via OpenVINO. Compared with the mkldnn acceleration library, the inference speed increases by about 1.5-2 times on the CPU. [Details link](#)
  - Add a model deployment solution on Raspberry Pi and further enrich an edge deployment solution. [Details link](#)
  - Optimize the data preprocessing and postprocessing code performance of PaddleLite Android deployment. The preprocessing speed increases by about 10 times and the postprocessing speed increases by about 4 times.
  - Optimize C++ deployment codes on the Paddle server and add parameters such as use_mkl. Compared with not starting mkldnn, the inference speed increases by about 10-50 times on the CPU.

- **Industrial Case Update**
  - Add a remote sensing segmentation case of large RGB images and provide a sliding window inference API, which can not only avoid the occurrence of insufficient GPU memory, but also eliminate the cracking feeling at the splice of the windows in the final inference results by configuring the degree of overlapping. [Details link](#)
  - Add a multi-channel remote sensing image segmentation case and bridge the whole process of data analysis, model training and model deployment of semantic segmentation tasks on any number of channels. [Details link](#)

- **Others**
  - Add the dataset splitting function which supports splitting ImageNet, PascalVOC, MSCOCO and semantic segmentation datasets with one click via command line. [Details link](#)

v1.1.0 2020.07.12

- **Model Update**
• Add semantic segmentation models HRNet and FastSCNN
• Add backbone HRNet to the object detection FasterRCNN and the instance segmentation MaskRCNN
• Add a COCO dataset pre-training model to the object detection/instance segmentation model
• Integrate X2Paddle. All PaddleX classification and semantic segmentation models support export as an ONNX protocol
• Model Deployment Update
  • Add the support for the Windows platform in model encryption
  • Add a Jetson and Paddle Lite model deployment and inference solution
  • Add batch inference in the C++ deployment codes and use OpenMP for parallel acceleration of preprocessing
• Add two PaddleX Industrial Cases
  • Portrait segmentation case
  • Industrial instrument reading case
• Add the data format conversion function which converts data annotated by LabelMe, Colabeler and the EasyData platform into a data format that PaddleX supports loading
• Update the PaddleX document by optimizing the document structure

v1.0.0 2020.05.20
• Add model C++ and Python deployment codes
• Add a model encryption deployment solution
• Add an OpenVINO deployment solution for classification models
• Add a model interpretability API

v0.1.8 2020.05.17
• Fix some code bugs
• Add the support for the data annotation format on the EasyData platform
• Support the pixel-level operator in the imgaug data enhancement library