附录 A　关于本论文模板的相关说明

此论文仅作模板示例用，有关内容的正确与否不做研究，参考文献引用部分也仅做演示用，请勿引用或直接抄录模板里的示例内容！否则后果自负。

本科毕业论文规范见厦门大学教务处文件：

https://jwc.xmu.edu.cn/2016/0506/c2160a173611/page.htm。

本论文模板编写参考书籍：《LaTex 入门》（刘海洋著）。

附录 B　附录代码示例

注意！代码中最好不要出现中文（包括注释），如果确实需要的，请自行搜索如何兼容中文。示例：

模型训练的代码如下：

```
"""
File: train.py
Author: San Zhang
Date: 2022-04-12
Description: Use the MindSpore framework to train a ResNet50-based cat and
dog classification neural network model.
"""

import os
import stat
import numpy as np
import matplotlib.pyplot as plt
import mindspore.nn as nn
import mindspore.dataset as ds
import mindspore.dataset.vision.c_transforms as CV
import mindspore.dataset.transforms.c_transforms as C
from mindspore import dtype as mstype
from mindspore.train.callback import TimeMonitor, Callback
from mindspore import Model, Tensor, context, save_checkpoint, \
    load_checkpoint, load_param_into_net
from resnet import resnet50

# Set use CPU/GPU/Ascend
context.set_context(mode=context.GRAPH_MODE, device_target="CPU")

# Set data path
train_data_path = 'dataset/train'
val_data_path = 'dataset/val'


def create_dataset(data_path, batch_size=100, repeat_num=1):
    data_set = ds.ImageFolderDataset(data_path, num_parallel_workers=2,
                                     shuffle=True)

    image_size = [224, 224]
    mean = [0.485 * 255, 0.456 * 255, 0.406 * 255]
    std = [0.229 * 255, 0.224 * 255, 0.225 * 255]
    trans = [
        CV.Decode(),
        CV.Resize(image_size),
        CV.Normalize(mean=mean, std=std),
        CV.HWC2CHW()
    ]
```

```
44
45      type_cast_op = C.TypeCast(mstype.int32)
46      data_set = data_set.map(operations=trans, input_columns="image",
47                              num_parallel_workers=2)
48      data_set = data_set.map(operations=type_cast_op, input_columns="label",
49                              num_parallel_workers=2)
50      data_set = data_set.batch(batch_size, drop_remainder=True)
51      data_set = data_set.repeat(repeat_num)
52
53      return data_set
54
55
56  train_ds = create_dataset(train_data_path)
57
58
59  def apply_eval(eval_param):
60      eval_model = eval_param['model']
61      eval_ds = eval_param['dataset']
62      metrics_name = eval_param['metrics_name']
63      res = eval_model.eval(eval_ds)
64      return res[metrics_name]
65
66
67  class EvalCallBack(Callback):
68
69      def __init__(self, eval_function, eval_param_dict, interval=1,
70                   eval_start_epoch=1, save_best_ckpt=True,
71                   ckpt_directory="./", besk_ckpt_name="best.ckpt",
72                   metrics_name="acc"):
73          super(EvalCallBack, self).__init__()
74          self.eval_param_dict = eval_param_dict
75          self.eval_function = eval_function
76          self.eval_start_epoch = eval_start_epoch
77          if interval < 1:
78              raise ValueError("interval should >= 1.")
79          self.interval = interval
80          self.save_best_ckpt = save_best_ckpt
81          self.best_res = 0
82          self.best_epoch = 0
83          if not os.path.isdir(ckpt_directory):
84              os.makedirs(ckpt_directory)
85          self.best_ckpt_path = os.path.join(ckpt_directory,
86                                             besk_ckpt_name)
87          self.metrics_name = metrics_name
88
89      def remove_ckpoint_file(self, file_name):
90          os.chmod(file_name, stat.S_IWRITE)
91          os.remove(file_name)
92
```

```
    def epoch_end(self, run_context):
        cb_params = run_context.original_args()
        cur_epoch = cb_params.cur_epoch_num
        loss_epoch = cb_params.net_outputs
        if cur_epoch >= self.eval_start_epoch and \
                (cur_epoch - self.eval_start_epoch) % self.interval == 0:
            res = self.eval_function(self.eval_param_dict)
            print('Epoch {}/{}'.format(cur_epoch, num_epochs))
            print('-' * 10)
            print('train Loss: {}'.format(loss_epoch))
            print('val Acc: {}'.format(res))
            if res >= self.best_res:
                self.best_res = res
                self.best_epoch = cur_epoch
                if self.save_best_ckpt:
                    if os.path.exists(self.best_ckpt_path):
                        self.remove_ckpoint_file(self.best_ckpt_path)
                    save_checkpoint(cb_params.train_network,
                                    self.best_ckpt_path)

    def end(self, run_context):
        print("End training, the best {0} is: {1}, "
              "the best {0} epoch is {2}".format(self.metrics_name,
                                                 self.best_res,
                                                 self.best_epoch),
              flush=True)


def visualize_model(best_ckpt_path,val_ds):
    net = resnet50(2)
    param_dict = load_checkpoint(best_ckpt_path)
    load_param_into_net(net,param_dict)
    loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True,reduction='mean')
    model = Model(net, loss,metrics={"Accuracy":nn.Accuracy()})
    data = next(val_ds.create_dict_iterator())
    images = data["image"].asnumpy()
    labels = data["label"].asnumpy()
    class_name = {0:"cat",1:"dog"}
    output = model.predict(Tensor(data['image']))
    pred = np.argmax(output.asnumpy(), axis=1)

    plt.figure(figsize=(12,5))
    for i in range(24):
        plt.subplot(3,8,i+1)
        color = 'blue' if pred[i] == labels[i] else 'red'
        plt.title('pre:{}'.format(class_name[pred[i]]), color=color)
        picture_show = np.transpose(images[i],(1,2,0))
        picture_show = picture_show/np.amax(picture_show)
        picture_show = np.clip(picture_show, 0, 1)
```

```
        plt.imshow(picture_show)
        plt.axis('off')
    plt.show()


def filter_checkpoint_parameter_by_list(origin_dict, param_filter):
    for key in list(origin_dict.keys()):
        for name in param_filter:
            if name in key:
                print("Delete parameter from checkpoint: ", key)
                del origin_dict[key]
                break


# Define Network
net = resnet50(2)
num_epochs=5

# Define optimizer and loss function
opt = nn.Momentum(params=net.trainable_params(),
                  learning_rate=0.1, momentum=0.9)
loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# Instantiate model
model = Model(net, loss, opt, metrics={"Accuracy": nn.Accuracy()})

# Load traing dataset and evaluation dataset
train_ds = create_dataset(train_data_path)
val_ds = create_dataset(val_data_path)

# Instantiate the callback class
eval_param_dict = {"model": model,"dataset": val_ds,
                   "metrics_name": "Accuracy"}
eval_cb = EvalCallBack(apply_eval, eval_param_dict,)

# Training model
model.train(num_epochs,train_ds,
            callbacks=[eval_cb, TimeMonitor()], dataset_sink_mode=False)

visualize_model('best.ckpt', val_ds)
```