基于深度学习的颌面骨龄分类软件源程序

```
import colorsys
import os
import time
import warnings

import numpy as np
import torch
import torch.backends.cudnn as cudnn
from PIL import Image, ImageDraw, ImageFont

from nets.ssd import SSD300
from utils.anchors import get_anchors
from utils.utils import (cvtColor, get_classes, preprocess_input, resize_image,
                         show_config)
from utils.utils_bbox import BBoxUtility

warnings.filterwarnings("ignore")

#--------------------------------------------#
#   使用自己训练好的模型预测需要修改 3 个参数
#   model_path、backbone 和 classes_path 都需要修改！
#   如果出现 shape 不匹配
#   一定要注意训练时的 config 里面的 num_classes、
#   model_path 和 classes_path 参数的修改
#--------------------------------------------#
class SSD(object):
    _defaults = {
        #--------------------------------------------------------------------------#
        #   使用自己训练好的模型进行预测一定要修改 model_path 和 classes_path！
        #   model_path 指向 logs 文件夹下的权值文件，classes_path 指向 model_data 下的 txt
        #
        #   训练好后 logs 文件夹下存在多个权值文件，选择验证集损失较低的即可。
        #   验证集损失较低不代表 mAP 较高，仅代表该权值在验证集上泛化性能较好。
        #   如果出现 shape 不匹配，同时要注意训练时的 model_path 和 classes_path 参数的
修改
        #--------------------------------------------------------------------------#
        "model_path"        : 'logs/best_epoch_weights.pth',
        "classes_path"      : 'model_data/classes.txt',
        #---------------------------------------------------------------------#
        #   用于预测的图像大小，和 train 时使用同一个即可
        #---------------------------------------------------------------------#
        "input_shape"       : [300, 300],
        #-------------------------------#
        #   主干网络的选择
        #   vgg 或者 mobilenetv2
        #-------------------------------#
        "backbone"          : "vgg",
        #---------------------------------------------------------------------#
        #   只有得分大于置信度的预测框会被保留下来
        #---------------------------------------------------------------------#
```

```
        "confidence"        : 0.5,
        #---------------------------------------------------------------------#
        #   非极大抑制所用到的 nms_iou 大小
        #---------------------------------------------------------------------#
        "nms_iou"           : 0.45,
        #---------------------------------------------------------------------#
        #   用于指定先验框的大小
        #---------------------------------------------------------------------#
        'anchors_size'      : [30, 60, 111, 162, 213, 264, 315],
        #---------------------------------------------------------------------#
        #   该变量用于控制是否使用 letterbox_image 对输入图像进行不失真的 resize，
        #   在多次测试后，发现关闭 letterbox_image 直接 resize 的效果更好
        #---------------------------------------------------------------------#
        "letterbox_image"   : False,
        #-------------------------------#
        #   是否使用 Cuda
        #   没有 GPU 可以设置成 False
        #-------------------------------#
        "cuda"              : True,
    }

    @classmethod
    def get_defaults(cls, n):
        if n in cls._defaults:
            return cls._defaults[n]
        else:
            return "Unrecognized attribute name '" + n + "'"

    #---------------------------------------------------#
    #   初始化 ssd
    #---------------------------------------------------#
    def __init__(self, **kwargs):
        self.__dict__.update(self._defaults)
        for name, value in kwargs.items():
            setattr(self, name, value)
        #---------------------------------------------------#
        #   计算总的类的数量
        #---------------------------------------------------#
        self.class_names, self.num_classes  = get_classes(self.classes_path)
        self.anchors                        = torch.from_numpy(get_anchors(self.input_shape, sel
f.anchors_size, self.backbone)).type(torch.FloatTensor)
        if self.cuda:
            self.anchors = self.anchors.cuda()
        self.num_classes                    = self.num_classes + 1

        #---------------------------------------------------#
        #   画框设置不同的颜色
        #---------------------------------------------------#
        hsv_tuples = [(x / self.num_classes, 1., 1.) for x in range(self.num_classes)]
        self.colors = list(map(lambda x: colorsys.hsv_to_rgb(*x), hsv_tuples))
```

```
        self.colors = list(map(lambda x: (int(x[0] * 255), int(x[1] * 255), int(x[2] * 255)), s
elf.colors))

        self.bbox_util = BBoxUtility(self.num_classes)
        self.generate()

        show_config(**self._defaults)

    #---------------------------------------------------#
    #   载入模型
    #---------------------------------------------------#
    def generate(self, onnx=False):
        #-------------------------------#
        #   载入模型与权值
        #-------------------------------#
        self.net    = SSD300(self.num_classes, self.backbone)
        device      = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.net.load_state_dict(torch.load(self.model_path, map_location=device))
        self.net    = self.net.eval()
        print('{} model, anchors, and classes loaded.'.format(self.model_path))
        if not onnx:
            if self.cuda:
                self.net = torch.nn.DataParallel(self.net)
                self.net = self.net.cuda()

    #---------------------------------------------------#
    #   检测图片
    #---------------------------------------------------#
    def detect_image(self, image,save_name, crop = False, count = False):
        outcome = 1
        position_list = []
        #---------------------------------------------------#
        #   计算输入图片的高和宽
        #---------------------------------------------------#
        image_shape = np.array(np.shape(image)[0:2])
        #---------------------------------------------------------#
        #   在这里将图像转换成 RGB 图像，防止灰度图在预测时报错。
        #   代码仅仅支持 RGB 图像的预测，所有其它类型的图像都会转化成 RGB
        #---------------------------------------------------------#
        image       = cvtColor(image)
        #---------------------------------------------------------#
        #   给图像增加灰条，实现不失真的 resize
        #   也可以直接 resize 进行识别
        #---------------------------------------------------------#
        image_data  = resize_image(image, (self.input_shape[1], self.input_shape[0]), self.letter
box_image)
        #---------------------------------------------------------#
        #   添加上 batch_size 维度，图片预处理，归一化。
        #---------------------------------------------------------#
        image_data = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, dtyp
```

```
e='float32')), (2, 0, 1)), 0)

        with torch.no_grad():
            #---------------------------------------------------#
            #   转化成 torch 的形式
            #---------------------------------------------------#
            images = torch.from_numpy(image_data).type(torch.FloatTensor)
            if self.cuda:
                images = images.cuda()
            #---------------------------------------------------------#
            #   将图像输入网络当中进行预测！
            #---------------------------------------------------------#
            outputs      = self.net(images)
            #-----------------------------------------------------------#
            #   将预测结果进行解码
            #-----------------------------------------------------------#
            results      = self.bbox_util.decode_box(outputs, self.anchors, image_shape, self.inp
ut_shape, self.letterbox_image,
                                                     nms_iou = self.nms_iou, confidence = self.con
fidence)
            #--------------------------------------#
            #   如果没有检测到物体，则返回原图
            #--------------------------------------#
            if len(results[0]) <= 0:
                outcome = 0
                print("no good")
                return image,outcome,position_list

            top_label   = np.array(results[0][:, 4], dtype = 'int32')
            top_conf    = results[0][:, 5]
            top_boxes   = results[0][:, :4]
        #---------------------------------------------------------#
        #   设置字体与边框厚度
        #---------------------------------------------------------#
        font = ImageFont.truetype(font='model_data/simhei.ttf', size=np.floor(3e-2 * np.shape(i
mage)[1] + 0.5).astype('int32'))
        thickness = max((np.shape(image)[0] + np.shape(image)[1]) // self.input_shape[0], 1)
        #---------------------------------------------------------#
        #   计数
        #---------------------------------------------------------#
        # if count:
        #     print("top_label:", top_label)
        #     classes_nums    = np.zeros([self.num_classes])
        #     for i in range(self.num_classes):
        #         num = np.sum(top_label == i)
        #         if num > 0:
        #             print(self.class_names[i], " : ", num)
        #         classes_nums[i] = num
        #     print("classes_nums:", classes_nums)
        #---------------------------------------------------------#
```

```
        #   是否进行目标的裁剪
        #---------------------------------------------------------#
        if crop:
            for i, c in list(enumerate(top_boxes)):
                top, left, bottom, right = top_boxes[i]
                top      = max(0, np.floor(top).astype('int32'))
                left     = max(0, np.floor(left).astype('int32'))
                bottom   = min(image.size[1], np.floor(bottom).astype('int32'))
                right    = min(image.size[0], np.floor(right).astype('int32'))
                position_list.append(left)
                position_list.append(top)
                position_list.append(right)
                position_list.append(bottom)
                if(len(position_list) > 0):
                    print("1\n")
                # print(save_name)
                # dir_save_path = "img_crop"
                # if not os.path.exists(dir_save_path):
                #     os.makedirs(dir_save_path)
                # crop_image = image.crop([left, top, right, bottom])
                # crop_image.save(os.path.join(dir_save_path, "crop_" + save_name), quality=9
5, subsampling=0)
                # print("save crop_" + save_name + " to " + dir_save_path)
        #---------------------------------------------------------#
        #   图像绘制
        #---------------------------------------------------------#
        for i, c in list(enumerate(top_label)):
            predicted_class = self.class_names[int(c)]
            box             = top_boxes[i]
            score           = top_conf[i]

            top, left, bottom, right = box

            top      = max(0, np.floor(top).astype('int32'))
            left     = max(0, np.floor(left).astype('int32'))
            bottom   = min(image.size[1], np.floor(bottom).astype('int32'))
            right    = min(image.size[0], np.floor(right).astype('int32'))

            label = '{} {:.2f}'.format(predicted_class, score)
            draw = ImageDraw.Draw(image)
            label_size = draw.textsize(label, font)
            label = label.encode('utf-8')
            print(label, top, left, bottom, right)

            if top - label_size[1] >= 0:
                text_origin = np.array([left, top - label_size[1]])
            else:
                text_origin = np.array([left, top + 1])

            for i in range(thickness):
```

```
                draw.rectangle([left + i, top + i, right - i, bottom - i], outline=self.colors[c])
            draw.rectangle([tuple(text_origin), tuple(text_origin + label_size)], fill=self.colors
[c])
            draw.text(text_origin, str(label,'UTF-8'), fill=(0, 0, 0), font=font)
            del draw

        return image,outcome,position_list

    def get_FPS(self, image, test_interval):
        #---------------------------------------------------#
        #   计算输入图片的高和宽
        #---------------------------------------------------#
        image_shape = np.array(np.shape(image)[0:2])
        #---------------------------------------------------------#
        #   在这里将图像转换成 RGB 图像，防止灰度图在预测时报错。
        #   代码仅仅支持 RGB 图像的预测，所有其它类型的图像都会转化成 RGB
        #---------------------------------------------------------#
        image       = cvtColor(image)
        #---------------------------------------------------------#
        #   给图像增加灰条，实现不失真的 resize
        #   也可以直接 resize 进行识别
        #---------------------------------------------------------#
        image_data  = resize_image(image, (self.input_shape[1], self.input_shape[0]), self.letter
box_image)
        #---------------------------------------------------------#
        #   添加上 batch_size 维度，图片预处理，归一化。
        #---------------------------------------------------------#
        image_data = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, dtyp
e='float32')), (2, 0, 1)), 0)

        with torch.no_grad():
            #---------------------------------------------------#
            #   转化成 torch 的形式
            #---------------------------------------------------#
            images = torch.from_numpy(image_data).type(torch.FloatTensor)
            if self.cuda:
                images = images.cuda()
            #---------------------------------------------------------#
            #   将图像输入网络当中进行预测！
            #---------------------------------------------------------#
            outputs     = self.net(images)
            #-----------------------------------------------------------#
            #   将预测结果进行解码
            #-----------------------------------------------------------#
            results     = self.bbox_util.decode_box(outputs, self.anchors, image_shape, self.inp
ut_shape, self.letterbox_image,
                                                    nms_iou = self.nms_iou, confidence = self.con
fidence)

        t1 = time.time()
```

```
        for _ in range(test_interval):
            with torch.no_grad():
                #---------------------------------------------------------#
                #   将图像输入网络当中进行预测！
                #---------------------------------------------------------#
                outputs     = self.net(images)
                #---------------------------------------------------------#
                #   将预测结果进行解码
                #---------------------------------------------------------#
                results     = self.bbox_util.decode_box(outputs, self.anchors, image_shape, self.input_shape, self.letterbox_image,
                                                        nms_iou = self.nms_iou, confidence = self.confidence)

        t2 = time.time()
        tact_time = (t2 - t1) / test_interval
        return tact_time

    def convert_to_onnx(self, simplify, model_path):
        import onnx
        self.generate(onnx=True)

        im                  = torch.zeros(1, 3, *self.input_shape).to('cpu')  # image size(1, 3, 512, 512) BCHW
        input_layer_names   = ["images"]
        output_layer_names  = ["output"]

        # Export the model
        print(f'Starting export with onnx {onnx.__version__}.')
        torch.onnx.export(self.net,
                          im,
                          f                 = model_path,
                          verbose           = False,
                          opset_version     = 12,
                          training          = torch.onnx.TrainingMode.EVAL,
                          do_constant_folding = True,
                          input_names       = input_layer_names,
                          output_names      = output_layer_names,
                          dynamic_axes      = None)

        # Checks
        model_onnx = onnx.load(model_path)  # load onnx model
        onnx.checker.check_model(model_onnx)  # check onnx model

        # Simplify onnx
        if simplify:
            import onnxsim
            print(f'Simplifying with onnx-simplifier {onnxsim.__version__}.')
            model_onnx, check = onnxsim.simplify(
                model_onnx,
```

```
                dynamic_input_shape=False,
                input_shapes=None)
            assert check, 'assert check failed'
            onnx.save(model_onnx, model_path)

        print('Onnx model save as {}'.format(model_path))

    def get_map_txt(self, image_id, image, class_names, map_out_path):
        f = open(os.path.join(map_out_path, "detection-results/"+image_id+".txt"),"w")
        #---------------------------------------------------#
        #   计算输入图片的高和宽
        #---------------------------------------------------#
        image_shape = np.array(np.shape(image)[0:2])
        #---------------------------------------------------------#
        #   在这里将图像转换成 RGB 图像，防止灰度图在预测时报错。
        #   代码仅仅支持 RGB 图像的预测，所有其它类型的图像都会转化成 RGB
        #---------------------------------------------------------#
        image       = cvtColor(image)
        #---------------------------------------------------------#
        #   给图像增加灰条，实现不失真的 resize
        #   也可以直接 resize 进行识别
        #---------------------------------------------------------#
        image_data  = resize_image(image, (self.input_shape[1], self.input_shape[0]), self.letterbox_image)
        #---------------------------------------------------------#
        #   添加上 batch_size 维度，图片预处理，归一化。
        #---------------------------------------------------------#
        image_data = np.expand_dims(np.transpose(preprocess_input(np.array(image_data, dtype='float32')), (2, 0, 1)), 0)

        with torch.no_grad():
            #---------------------------------------------------#
            #   转化成 torch 的形式
            #---------------------------------------------------#
            images = torch.from_numpy(image_data).type(torch.FloatTensor)
            if self.cuda:
                images = images.cuda()
            #---------------------------------------------------------#
            #   将图像输入网络当中进行预测！
            #---------------------------------------------------------#
            outputs     = self.net(images)
            #-----------------------------------------------------------#
            #   将预测结果进行解码
            #-----------------------------------------------------------#
            results     = self.bbox_util.decode_box(outputs, self.anchors, image_shape, self.input_shape, self.letterbox_image,
                                                    nms_iou = self.nms_iou, confidence = self.confidence)
            #--------------------------------------#
            #   如果没有检测到物体，则返回原图
```

```
            #---------------------------------------#
            if len(results[0]) <= 0:
                return

            top_label   = np.array(results[0][:, 4], dtype = 'int32')
            top_conf    = results[0][:, 5]
            top_boxes   = results[0][:, :4]

        for i, c in list(enumerate(top_label)):
            predicted_class = self.class_names[int(c)]
            box             = top_boxes[i]
            score           = str(top_conf[i])

            top, left, bottom, right = box
            if predicted_class not in class_names:
                continue

            f.write("%s %s %s %s %s %s\n" % (predicted_class, score[:6], str(int(left)), str(i
nt(top)), str(int(right)),str(int(bottom))))

        f.close()
        return


class L2Norm(nn.Module):
    def __init__(self,n_channels, scale):
        super(L2Norm,self).__init__()
        self.n_channels = n_channels
        self.gamma      = scale or None
        self.eps        = 1e-10
        self.weight     = nn.Parameter(torch.Tensor(self.n_channels))
        self.reset_parameters()

    def reset_parameters(self):
        init.constant_(self.weight,self.gamma)

    def forward(self, x):
        norm    = x.pow(2).sum(dim=1, keepdim=True).sqrt()+self.eps
        #x /= norm
        x       = torch.div(x,norm)
        out     = self.weight.unsqueeze(0).unsqueeze(2).unsqueeze(3).expand_as(x) * x
        return out

def add_extras(in_channels, backbone_name):
    layers = []
    if backbone_name == 'vgg':
        # Block 6
        # 19,19,1024 -> 19,19,256 -> 10,10,512
        layers += [nn.Conv2d(in_channels, 256, kernel_size=1, stride=1)]
        layers += [nn.Conv2d(256, 512, kernel_size=3, stride=2, padding=1)]
```

```
501
502             # Block 7
503             # 10,10,512 -> 10,10,128 -> 5,5,256
504             layers += [nn.Conv2d(512, 128, kernel_size=1, stride=1)]
505             layers += [nn.Conv2d(128, 256, kernel_size=3, stride=2, padding=1)]
506
507             # Block 8
508             # 5,5,256 -> 5,5,128 -> 3,3,256
509             layers += [nn.Conv2d(256, 128, kernel_size=1, stride=1)]
510             layers += [nn.Conv2d(128, 256, kernel_size=3, stride=1)]
511
512             # Block 9
513             # 3,3,256 -> 3,3,128 -> 1,1,256
514             layers += [nn.Conv2d(256, 128, kernel_size=1, stride=1)]
515             layers += [nn.Conv2d(128, 256, kernel_size=3, stride=1)]
516         else:
517             layers += [InvertedResidual(in_channels, 512, stride=2, expand_ratio=0.2)]
518             layers += [InvertedResidual(512, 256, stride=2, expand_ratio=0.25)]
519             layers += [InvertedResidual(256, 256, stride=2, expand_ratio=0.5)]
520             layers += [InvertedResidual(256, 64, stride=2, expand_ratio=0.25)]
521
522         return nn.ModuleList(layers)
523
524     class SSD300(nn.Module):
525         def __init__(self, num_classes, backbone_name, pretrained = False):
526             super(SSD300, self).__init__()
527             self.num_classes     = num_classes
528             if backbone_name     == "vgg":
529                 self.vgg         = add_vgg(pretrained)
530                 self.extras      = add_extras(1024, backbone_name)
531                 self.L2Norm      = L2Norm(512, 20)
532                 mbox             = [4, 6, 6, 6, 4, 4]
533
534                 loc_layers       = []
535                 conf_layers      = []
536                 backbone_source  = [21, -2]
537                 #---------------------------------------------------#
538                 #   在 add_vgg 获得的特征层里
539                 #   第 21 层和-2 层可以用来进行回归预测和分类预测。
540                 #   分别是 conv4-3(38,38,512)和 conv7(19,19,1024)的输出
541                 #---------------------------------------------------#
542                 for k, v in enumerate(backbone_source):
543                     loc_layers  += [nn.Conv2d(self.vgg[v].out_channels, mbox[k] * 4, kernel
544     _size = 3, padding = 1)]
545                     conf_layers += [nn.Conv2d(self.vgg[v].out_channels, mbox[k] * num_clas
546     ses, kernel_size = 3, padding = 1)]
547                 #-----------------------------------------------------------#
548                 #   在 add_extras 获得的特征层里
549                 #   第 1 层、第 3 层、第 5 层、第 7 层可以用来进行回归预测和分类预测。
550                 #   shape 分别为(10,10,512), (5,5,256), (3,3,256), (1,1,256)
```

```
                #-------------------------------------------------------------#
                for k, v in enumerate(self.extras[1::2], 2):
                    loc_layers  += [nn.Conv2d(v.out_channels, mbox[k] * 4, kernel_size = 3,
 padding = 1)]
                    conf_layers += [nn.Conv2d(v.out_channels, mbox[k] * num_classes, kerne
l_size = 3, padding = 1)]
            else:
                self.mobilenet  = mobilenet_v2(pretrained).features
                self.extras     = add_extras(1280, backbone_name)
                self.L2Norm     = L2Norm(96, 20)
                mbox            = [6, 6, 6, 6, 6, 6]

                loc_layers      = []
                conf_layers     = []
                backbone_source = [13, -1]
                for k, v in enumerate(backbone_source):
                    loc_layers  += [nn.Conv2d(self.mobilenet[v].out_channels, mbox[k] * 4,
kernel_size = 3, padding = 1)]
                    conf_layers += [nn.Conv2d(self.mobilenet[v].out_channels, mbox[k] * nu
m_classes, kernel_size = 3, padding = 1)]
                for k, v in enumerate(self.extras, 2):
                    loc_layers  += [nn.Conv2d(v.out_channels, mbox[k] * 4, kernel_size = 3,
 padding = 1)]
                    conf_layers += [nn.Conv2d(v.out_channels, mbox[k] * num_classes, kerne
l_size = 3, padding = 1)]

            self.loc            = nn.ModuleList(loc_layers)
            self.conf           = nn.ModuleList(conf_layers)
            self.backbone_name  = backbone_name

        def forward(self, x):
            #---------------------------#
            #   x 是 300,300,3
            #---------------------------#
            sources = list()
            loc     = list()
            conf    = list()

            #---------------------------#
            #   获得 conv4_3 的内容
            #   shape 为 38,38,512
            #---------------------------#
            if self.backbone_name == "vgg":
                for k in range(23):
                    x = self.vgg[k](x)
            else:
                for k in range(14):
                    x = self.mobilenet[k](x)
            #---------------------------#
            #   conv4_3 的内容
```

```
        #   需要进行 L2 标准化
        #-------------------------#
        s = self.L2Norm(x)
        sources.append(s)

        #-------------------------#
        #   获得 conv7 的内容
        #   shape 为 19,19,1024
        #-------------------------#
        if self.backbone_name == "vgg":
            for k in range(23, len(self.vgg)):
                x = self.vgg[k](x)
        else:
            for k in range(14, len(self.mobilenet)):
                x = self.mobilenet[k](x)

        sources.append(x)
        #-------------------------------------------------------------#
        #   在 add_extras 获得的特征层里
        #   第 1 层、第 3 层、第 5 层、第 7 层可以用来进行回归预测和分类预测。
        #   shape 分别为(10,10,512), (5,5,256), (3,3,256), (1,1,256)
        #-------------------------------------------------------------#
        for k, v in enumerate(self.extras):
            x = F.relu(v(x), inplace=True)
            if self.backbone_name == "vgg":
                if k % 2 == 1:
                    sources.append(x)
            else:
                sources.append(x)

        #-------------------------------------------------------------#
        #   为获得的 6 个有效特征层添加回归预测和分类预测
        #-------------------------------------------------------------#
        for (x, l, c) in zip(sources, self.loc, self.conf):
            loc.append(l(x).permute(0, 2, 3, 1).contiguous())
            conf.append(c(x).permute(0, 2, 3, 1).contiguous())

        #-------------------------------------------------------------#
        #   进行 reshape 方便堆叠
        #-------------------------------------------------------------#
        loc     = torch.cat([o.view(o.size(0), -1) for o in loc], 1)
        conf    = torch.cat([o.view(o.size(0), -1) for o in conf], 1)
        #-------------------------------------------------------------#
        #   loc 会 reshape 到 batch_size, num_anchors, 4
        #   conf 会 reshap 到 batch_size, num_anchors, self.num_classes
        #-------------------------------------------------------------#
        output = (
            loc.view(loc.size(0), -1, 4),
            conf.view(conf.size(0), -1, self.num_classes),
        )
```

```
651             return output
652
653
654         base = [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'C', 512, 512, 512, 'M',
655                 512, 512, 512]
656
657         def vgg(pretrained = False):
658             layers = []
659             in_channels = 3
660             for v in base:
661                 if v == 'M':
662                     layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
663                 elif v == 'C':
664                     layers += [nn.MaxPool2d(kernel_size=2, stride=2, ceil_mode=True)]
665                 else:
666                     conv2d = nn.Conv2d(in_channels, v, kernel_size=3, padding=1)
667                     layers += [conv2d, nn.ReLU(inplace=True)]
668                     in_channels = v
669             # 19, 19, 512 -> 19, 19, 512
670             pool5 = nn.MaxPool2d(kernel_size=3, stride=1, padding=1)
671             # 19, 19, 512 -> 19, 19, 1024
672             conv6 = nn.Conv2d(512, 1024, kernel_size=3, padding=6, dilation=6)
673             # 19, 19, 1024 -> 19, 19, 1024
674             conv7 = nn.Conv2d(1024, 1024, kernel_size=1)
675             layers += [pool5, conv6,
676                        nn.ReLU(inplace=True), conv7, nn.ReLU(inplace=True)]
677
678             model = nn.ModuleList(layers)
679             if pretrained:
680                 state_dict = load_state_dict_from_url("https://download.pytorch.org/models/vgg1
681     6-397923af.pth", model_dir="./model_data")
682                 state_dict = {k.replace('features.', '') : v for k, v in state_dict.items()}
683                 model.load_state_dict(state_dict, strict = False)
684             return model
685
686         if __name__ == "__main__":
687             net = vgg()
688             for i, layer in enumerate(net):
689                 print(i, layer)
690
691
692         class MultiboxLoss(nn.Module):
693             def __init__(self, num_classes, alpha=1.0, neg_pos_ratio=3.0,
694                          background_label_id=0, negatives_for_hard=100.0):
695                 self.num_classes = num_classes
696                 self.alpha = alpha
697                 self.neg_pos_ratio = neg_pos_ratio
698                 if background_label_id != 0:
699                     raise Exception('Only 0 as background label id is supported')
700                 self.background_label_id = background_label_id
```

```
701             self.negatives_for_hard = torch.FloatTensor([negatives_for_hard])[0]
702
703         def _l1_smooth_loss(self, y_true, y_pred):
704             abs_loss = torch.abs(y_true - y_pred)
705             sq_loss = 0.5 * (y_true - y_pred)**2
706             l1_loss = torch.where(abs_loss < 1.0, sq_loss, abs_loss - 0.5)
707             return torch.sum(l1_loss, -1)
708
709         def _softmax_loss(self, y_true, y_pred):
710             y_pred = torch.clamp(y_pred, min = 1e-7)
711             softmax_loss = -torch.sum(y_true * torch.log(y_pred),
712                                       axis=-1)
713             return softmax_loss
714
715         def forward(self, y_true, y_pred):
716             # ---------------------------------------------- #
717             #   y_true batch_size, 8732, 4 + self.num_classes + 1
718             #   y_pred batch_size, 8732, 4 + self.num_classes
719             # ---------------------------------------------- #
720             num_boxes       = y_true.size()[1]
721             y_pred          = torch.cat([y_pred[0], nn.Softmax(-1)(y_pred[1])], dim = -
722 1)
723
724             # ---------------------------------------------- #
725             #   分类的 loss
726             #   batch_size,8732,21 -> batch_size,8732
727             # ---------------------------------------------- #
728             conf_loss = self._softmax_loss(y_true[:, :, 4:-1], y_pred[:, :, 4:])
729
730             # ---------------------------------------------- #
731             #   框的位置的 loss
732             #   batch_size,8732,4 -> batch_size,8732
733             # ---------------------------------------------- #
734             loc_loss = self._l1_smooth_loss(y_true[:, :, :4],
735                                             y_pred[:, :, :4])
736
737             # ---------------------------------------------- #
738             #   获取所有的正标签的 loss
739             # ---------------------------------------------- #
740             pos_loc_loss = torch.sum(loc_loss * y_true[:, :, -1],
741                                      axis=1)
742             pos_conf_loss = torch.sum(conf_loss * y_true[:, :, -1],
743                                       axis=1)
744
745             # ---------------------------------------------- #
746             #   每一张图的正样本的个数
747             #   num_pos         [batch_size,]
748             # ---------------------------------------------- #
749             num_pos = torch.sum(y_true[:, :, -1], axis=-1)
750
```

```
751                 # ---------------------------------------------- #
752                 #   每一张图的负样本的个数
753                 #   num_neg         [batch_size,]
754                 # ---------------------------------------------- #
755                 num_neg = torch.min(self.neg_pos_ratio * num_pos, num_boxes - num_pos)
756                 # 找到了哪些值是大于 0 的
757                 pos_num_neg_mask = num_neg > 0
758                 # ---------------------------------------------- #
759                 #   如果所有的图，正样本的数量均为 0
760                 #   那么则默认选取 100 个先验框作为负样本
761                 # ---------------------------------------------- #
762                 has_min = torch.sum(pos_num_neg_mask)
763
764                 # ---------------------------------------------- #
765                 #   从这里往后，与视频中看到的代码有些许不同。
766                 #   由于以前的负样本选取方式存在一些问题，
767                 #   我对该部分代码进行重构。
768                 #   求整个 batch 应该的负样本数量总和
769                 # ---------------------------------------------- #
770                 num_neg_batch = torch.sum(num_neg) if has_min > 0 else self.negatives_for_
771     hard
772
773                 # ---------------------------------------------- #
774                 #   对预测结果进行判断，如果该先验框没有包含物体
775                 #   那么它的不属于背景的预测概率过大的话
776                 #   就是难分类样本
777                 # ---------------------------------------------- #
778                 confs_start = 4 + self.background_label_id + 1
779                 confs_end   = confs_start + self.num_classes - 1
780
781                 # ---------------------------------------------- #
782                 #   batch_size,8732
783                 #   把不是背景的概率求和，求和后的概率越大
784                 #   代表越难分类。
785                 # ---------------------------------------------- #
786                 max_confs = torch.sum(y_pred[:, :, confs_start:confs_end], dim=2)
787
788                 # ---------------------------------------------- #
789                 #   只有没有包含物体的先验框才得到保留
790                 #   我们在整个 batch 里面选取最难分类的 num_neg_batch 个
791                 #   先验框作为负样本。
792                 # ---------------------------------------------- #
793                 max_confs   = (max_confs * (1 - y_true[:, :, -1])).view([-1])
794
795                 _, indices  = torch.topk(max_confs, k = int(num_neg_batch.cpu().numpy().tolis
796     t()))
797
798                 neg_conf_loss = torch.gather(conf_loss.view([-1]), 0, indices)
799
800                 # 进行归一化
```

```
                num_pos        = torch.where(num_pos != 0, num_pos, torch.ones_like(num_po
s))
                total_loss  = torch.sum(pos_conf_loss) + torch.sum(neg_conf_loss) + torch.su
m(self.alpha * pos_loc_loss)
                total_loss  = total_loss / torch.sum(num_pos)
                return total_loss

    def weights_init(net, init_type='normal', init_gain=0.02):
        def init_func(m):
            classname = m.__class__.__name__
            if hasattr(m, 'weight') and classname.find('Conv') != -1:
                if init_type == 'normal':
                    torch.nn.init.normal_(m.weight.data, 0.0, init_gain)
                elif init_type == 'xavier':
                    torch.nn.init.xavier_normal_(m.weight.data, gain=init_gain)
                elif init_type == 'kaiming':
                    torch.nn.init.kaiming_normal_(m.weight.data, a=0, mode='fan_in')
                elif init_type == 'orthogonal':
                    torch.nn.init.orthogonal_(m.weight.data, gain=init_gain)
                else:
                    raise NotImplementedError('initialization method [%s] is not implem
ented' % init_type)
            elif classname.find('BatchNorm2d') != -1:
                torch.nn.init.normal_(m.weight.data, 1.0, 0.02)
                torch.nn.init.constant_(m.bias.data, 0.0)
        print('initialize network with %s type' % init_type)
        net.apply(init_func)

    def get_lr_scheduler(lr_decay_type, lr, min_lr, total_iters, warmup_iters_ratio = 0.05, w
armup_lr_ratio = 0.1, no_aug_iter_ratio = 0.05, step_num = 10):
        def yolox_warm_cos_lr(lr, min_lr, total_iters, warmup_total_iters, warmup_lr_start,
no_aug_iter, iters):
            if iters <= warmup_total_iters:
                # lr = (lr - warmup_lr_start) * iters / float(warmup_total_iters) + warmu
p_lr_start
                lr = (lr - warmup_lr_start) * pow(iters / float(warmup_total_iters), 2) +
warmup_lr_start
            elif iters >= total_iters - no_aug_iter:
                lr = min_lr
            else:
                lr = min_lr + 0.5 * (lr - min_lr) * (
                    1.0 + math.cos(math.pi* (iters - warmup_total_iters) / (total_iters -
warmup_total_iters - no_aug_iter))
                )
            return lr

        def step_lr(lr, decay_rate, step_size, iters):
            if step_size < 1:
                raise ValueError("step_size must above 1.")
            n           = iters // step_size
```

```
851                 out_lr  = lr * decay_rate ** n
852                 return out_lr
853 
854             if lr_decay_type == "cos":
855                 warmup_total_iters  = min(max(warmup_iters_ratio * total_iters, 1), 3)
856                 warmup_lr_start     = max(warmup_lr_ratio * lr, 1e-6)
857                 no_aug_iter         = min(max(no_aug_iter_ratio * total_iters, 1), 15)
858                 func = partial(yolox_warm_cos_lr ,lr, min_lr, total_iters, warmup_total_iters,
859 warmup_lr_start, no_aug_iter)
860             else:
861                 decay_rate  = (min_lr / lr) ** (1 / (step_num - 1))
862                 step_size   = total_iters / step_num
863                 func = partial(step_lr, lr, decay_rate, step_size)
864 
865             return func
866 
867         def set_optimizer_lr(optimizer, lr_scheduler_func, epoch):
868             lr = lr_scheduler_func(epoch)
869             for param_group in optimizer.param_groups:
870                 param_group['lr'] = lr
871 
872         def _make_divisible(v, divisor, min_value=None):
873             if min_value is None:
874                 min_value = divisor
875             new_v = max(min_value, int(v + divisor / 2) // divisor * divisor)
876             if new_v < 0.9 * v:
877                 new_v += divisor
878             return new_v
879 
880         class ConvBNReLU(nn.Sequential):
881             def __init__(self, in_planes, out_planes, kernel_size=3, stride=1, groups=1):
882                 padding = (kernel_size - 1) // 2
883                 super(ConvBNReLU, self).__init__(
884                     nn.Conv2d(in_planes, out_planes, kernel_size, stride, padding, groups=gro
885 ups, bias=False),
886                     nn.BatchNorm2d(out_planes),
887                     nn.ReLU6(inplace=True)
888                 )
889                 self.out_channels = out_planes
890 
891         class InvertedResidual(nn.Module):
892             def __init__(self, inp, oup, stride, expand_ratio):
893                 super(InvertedResidual, self).__init__()
894                 self.stride = stride
895                 assert stride in [1, 2]
896 
897                 hidden_dim = int(round(inp * expand_ratio))
898                 self.use_res_connect = self.stride == 1 and inp == oup
899 
900                 layers = []
```

```
901                 if expand_ratio != 1:
902                     layers.append(ConvBNReLU(inp, hidden_dim, kernel_size=1))
903                 layers.extend([
904                     ConvBNReLU(hidden_dim, hidden_dim, stride=stride, groups=hidden_di
905 m),
906                     nn.Conv2d(hidden_dim, oup, 1, 1, 0, bias=False),
907                     nn.BatchNorm2d(oup),
908                 ])
909                 self.conv = nn.Sequential(*layers)
910 
911                 self.out_channels = oup
912 
913             def forward(self, x):
914                 if self.use_res_connect:
915                     return x + self.conv(x)
916                 else:
917                     return self.conv(x)
918 
919         class MobileNetV2(nn.Module):
920             def __init__(self, num_classes=1000, width_mult=1.0, inverted_residual_setting=Non
921 e, round_nearest=8):
922                 super(MobileNetV2, self).__init__()
923                 block = InvertedResidual
924                 input_channel = 32
925                 last_channel = 1280
926 
927                 if inverted_residual_setting is None:
928                     inverted_residual_setting = [
929                         [1, 16, 1, 1],
930                         [6, 24, 2, 2],
931                         [6, 32, 3, 2],
932                         [6, 64, 4, 2],
933                         [6, 96, 3, 1],
934                         [6, 160, 3, 2],
935                         [6, 320, 1, 1],
936                     ]
937 
938                 if len(inverted_residual_setting) == 0 or len(inverted_residual_setting[0]) != 4:
939                     raise ValueError("inverted_residual_setting should be non-empty "
940                                      "or a 4-element list, got {}".format(inverted_residual_
941 setting))
942 
943                 input_channel = _make_divisible(input_channel * width_mult, round_nearest)
944                 self.last_channel = _make_divisible(last_channel * max(1.0, width_mult), roun
945 d_nearest)
946                 features = [ConvBNReLU(3, input_channel, stride=2)]
947                 for t, c, n, s in inverted_residual_setting:
948                     output_channel = _make_divisible(c * width_mult, round_nearest)
949                     for i in range(n):
950                         stride = s if i == 0 else 1
```

```
                        features.append(block(input_channel, output_channel, stride, expand_r
atio=t))
                        input_channel = output_channel
                features.append(ConvBNReLU(input_channel, self.last_channel, kernel_size=1))
                self.features = nn.Sequential(*features)

                self.classifier = nn.Sequential(
                    nn.Dropout(0.2),
                    nn.Linear(self.last_channel, num_classes),
                )

                for m in self.modules():
                    if isinstance(m, nn.Conv2d):
                        nn.init.kaiming_normal_(m.weight, mode='fan_out')
                        if m.bias is not None:
                            nn.init.zeros_(m.bias)
                    elif isinstance(m, nn.BatchNorm2d):
                        nn.init.ones_(m.weight)
                        nn.init.zeros_(m.bias)
                    elif isinstance(m, nn.Linear):
                        nn.init.normal_(m.weight, 0, 0.01)
                        nn.init.zeros_(m.bias)

            def forward(self, x):
                x = self.features(x)
                x = x.mean([2, 3])
                x = self.classifier(x)
                return x

        def mobilenet_v2(pretrained=False, progress=True, **kwargs):
            model = MobileNetV2(**kwargs)
            if pretrained:
                state_dict = load_state_dict_from_url('https://download.pytorch.org/models/mobil
enet_v2-b0353104.pth', model_dir="./model_data", progress=progress)
                model.load_state_dict(state_dict)
            del model.classifier
            return model

        if __name__ == "__main__":
            net = mobilenet_v2()
            for i, layer in enumerate(net.features):
                print(i, layer)



    def cvtColor(image):
        if len(np.shape(image)) == 3 and np.shape(image)[2] == 3:
            return image
        else:
            image = image.convert('RGB')
```

```
            return image

#---------------------------------------------------#
#   对输入图像进行 resize
#---------------------------------------------------#
def resize_image(image, size, letterbox_image):
    iw, ih  = image.size
    w, h    = size
    if letterbox_image:
        scale   = min(w/iw, h/ih)
        nw      = int(iw*scale)
        nh      = int(ih*scale)

        image   = image.resize((nw,nh), Image.BICUBIC)
        new_image = Image.new('RGB', size, (128,128,128))
        new_image.paste(image, ((w-nw)//2, (h-nh)//2))
    else:
        new_image = image.resize((w, h), Image.BICUBIC)
    return new_image

#---------------------------------------------------#
#   获得类
#---------------------------------------------------#
def get_classes(classes_path):
    with open(classes_path, encoding='utf-8') as f:
        class_names = f.readlines()
    class_names = [c.strip() for c in class_names]
    return class_names, len(class_names)

#---------------------------------------------------#
#   获得学习率
#---------------------------------------------------#
def preprocess_input(inputs):
    MEANS = (104, 117, 123)
    return inputs - MEANS

#---------------------------------------------------#
#   获得学习率
#---------------------------------------------------#
def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

def show_config(**kwargs):
    print('Configurations:')
    print('-' * 70)
    print('|%25s | %40s|' % ('keys', 'values'))
    print('-' * 70)
    for key, value in kwargs.items():
        print('|%25s | %40s|' % (str(key), str(value)))
```

```
1051        print('-' * 70)
1052
1053    def download_weights(backbone, model_dir="./model_data"):
1054        import os
1055        from torch.hub import load_state_dict_from_url
1056
1057        download_urls = {
1058            'vgg'               : 'https://download.pytorch.org/models/vgg16-397923af.pth',
1059            'mobilenetv2'       : 'https://download.pytorch.org/models/mobilenet_v2-b0353104.pth'
1060        }
1061        url = download_urls[backbone]
1062
1063        if not os.path.exists(model_dir):
1064            os.makedirs(model_dir)
1065        load_state_dict_from_url(url, model_dir)
1066
1067
1068
1069    if __name__ == "__main__":
1070        #-------------------------------#
1071        #   Cuda    是否使用 Cuda
1072        #           没有 GPU 可以设置成 False
1073        #-------------------------------#
1074        Cuda = True
1075        #---------------------------------------------------------------------#
1076        #   distributed     用于指定是否使用单机多卡分布式运行
1077        #                   终端指令仅支持 Ubuntu。CUDA_VISIBLE_DEVICES 用于在 Ub
1078    untu 下指定显卡。
1079        #                   Windows 系统下默认使用 DP 模式调用所有显卡，不支持 DDP。
1080        #   DP 模式：
1081        #       设置            distributed = False
1082        #       在终端中输入    CUDA_VISIBLE_DEVICES=0,1 python train.py
1083        #   DDP 模式：
1084        #       设置            distributed = True
1085        #       在终端中输入    CUDA_VISIBLE_DEVICES=0,1 python -m torch.distribute
1086    d.launch --nproc_per_node=2 train.py
1087        #---------------------------------------------------------------------#
1088        distributed     = False
1089        #---------------------------------------------------------------------#
1090        #   sync_bn     是否使用 sync_bn，DDP 模式多卡可用
1091        #---------------------------------------------------------------------#
1092        sync_bn         = False
1093        #---------------------------------------------------------------------#
1094        #   fp16        是否使用混合精度训练
1095        #               可减少约一半的显存、需要 pytorch1.7.1 以上
1096        #---------------------------------------------------------------------#
1097        fp16            = False
1098        #---------------------------------------------------------------------#
1099        #   classes_path    指向 model_data 下的 txt，与自己训练的数据集相关
1100        #                   训练前一定要修改 classes_path，使其对应自己的数据集
```

```
1101        #-------------------------------------------------------------------#
1102        classes_path      = 'model_data/classes.txt'
1103        #-------------------------------------------------------------------------------------------------------
1104    -------------#
1105        #     权值文件的下载请看 README，可以通过网盘下载。模型的 预训练权重 对不同
1106    数据集是通用的，因为特征是通用的。
1107        #     模型的 预训练权重 比较重要的部分是 主干特征提取网络的权值部分，用于进行
1108    特征提取。
1109        #     预训练权重对于 99%的情况都必须要用，不用的话主干部分的权值太过随机，特
1110    征提取效果不明显，网络训练的结果也不会好
1111        #
1112        #     如果训练过程中存在中断训练的操作，可以将 model_path 设置成 logs 文件夹下的
1113    权值文件，将已经训练了一部分的权值再次载入。
1114        #     同时修改下方的 冻结阶段 或者 解冻阶段 的参数，来保证模型 epoch 的连续性。
1115        #
1116        #     当 model_path = "的时候不加载整个模型的权值。
1117        #
1118        #     此处使用的是整个模型的权重，因此是在 train.py 进行加载的，下面的 pretrain 不
1119    影响此处的权值加载。
1120        #     如果想要让模型从主干的预训练权值开始训练，则设置 model_path = "，下面的 p
1121    retrain = True，此时仅加载主干。
1122        #     如果想要让模型从 0 开始训练，则设置 model_path = "，下面的 pretrain = Fasle，
1123    Freeze_Train = Fasle，此时从 0 开始训练，且没有冻结主干的过程。
1124        #     一般来讲，从 0 开始训练效果会很差，因为权值太过随机，特征提取效果不明显。
1125        #
1126        #     网络一般不从 0 开始训练，至少会使用主干部分的权值，有些论文提到可以不用预
1127    训练，主要原因是他们 数据集较大 且 调参能力优秀。
1128        #     如果一定要训练网络的主干部分，可以了解 imagenet 数据集，首先训练分类模型，
1129    分类模型的 主干部分 和该模型通用，基于此进行训练。
1130        #-------------------------------------------------------------------------------------------------------
1131    -------------#
1132        model_path        = 'model_data/ep180-loss1.927-val_loss1.786.pth'
1133        #-----------------------------------------------------#
1134        #     input_shape       输入的 shape 大小
1135        #-----------------------------------------------------#
1136        input_shape       = [300, 300]
1137        #-----------------------------------------------------#
1138        #     vgg 或者 mobilenetv2
1139        #-----------------------------------------------------#
1140        backbone          = "vgg"
1141        #-------------------------------------------------------------------------------------------------------
1142    -------------#
1143        #     pretrained        是否使用主干网络的预训练权重，此处使用的是主干的权重，因此
1144    是在模型构建的时候进行加载的。
1145        #                       如果设置了 model_path，则主干的权值无需加载，pretrained 的
1146    值无意义。
1147        #                       如果不设置 model_path，pretrained = True，此时仅加载主干开
1148    始训练。
1149        #                       如果不设置 model_path，pretrained = False，Freeze_Train = Fa
1150    sle，此时从 0 开始训练，且没有冻结主干的过程。
```

```
    #---------------------------------------------------------------------------------------------------------------
-------------#
    pretrained          = True
    #-------------------------------------------------------#
    #   可用于设定先验框的大小，默认的 anchors_size
    #   是根据 voc 数据集设定的，大多数情况下都是通用的！
    #   如果想要检测小物体，可以修改 anchors_size
    #   一般调小浅层先验框的大小就行了！因为浅层负责小物体检测！
    #   比如 anchors_size = [21, 45, 99, 153, 207, 261, 315]  [30, 60, 111, 162, 213, 26
4, 315]
    #-------------------------------------------------------#
    anchors_size        = [21, 45, 99, 153, 207, 261, 315]

    #---------------------------------------------------------------------------------------------------------------
-------------#
    #   训练分为两个阶段，分别是冻结阶段和解冻阶段。设置冻结阶段是为了满足机器性
能不足的同学的训练需求。
    #   冻结训练需要的显存较小，显卡非常差的情况下，可设置 Freeze_Epoch 等于 UnFr
eeze_Epoch，此时仅仅进行冻结训练。
    #
    #   在此提供若干参数设置建议，各位训练者根据自己的需求进行灵活调整：
    #   （一）从整个模型的预训练权重开始训练：
    #       Adam：
    #           Init_Epoch = 0，Freeze_Epoch = 50，UnFreeze_Epoch = 100，Freeze_T
rain = True，optimizer_type = 'adam'，Init_lr = 6e-4，weight_decay = 0。（冻结）
    #           Init_Epoch = 0，UnFreeze_Epoch = 100，Freeze_Train = False，optimiz
er_type = 'adam'，Init_lr = 6e-4，weight_decay = 0。（不冻结）
    #       SGD：
    #           Init_Epoch = 0，Freeze_Epoch = 50，UnFreeze_Epoch = 200，Freeze_T
rain = True，optimizer_type = 'sgd'，Init_lr = 2e-3，weight_decay = 5e-4。（冻结）
    #           Init_Epoch = 0，UnFreeze_Epoch = 200，Freeze_Train = False，optimiz
er_type = 'sgd'，Init_lr = 2e-3，weight_decay = 5e-4。（不冻结）
    #       其中：UnFreeze_Epoch 可以在 100-300 之间调整。
    #   （二）从主干网络的预训练权重开始训练：
    #       Adam：
    #           Init_Epoch = 0，Freeze_Epoch = 50，UnFreeze_Epoch = 100，Freeze_T
rain = True，optimizer_type = 'adam'，Init_lr = 6e-4，weight_decay = 0。（冻结）
    #           Init_Epoch = 0，UnFreeze_Epoch = 100，Freeze_Train = False，optimiz
er_type = 'adam'，Init_lr = 6e-4，weight_decay = 0。（不冻结）
    #       SGD：
    #           Init_Epoch = 0，Freeze_Epoch = 50，UnFreeze_Epoch = 200，Freeze_T
rain = True，optimizer_type = 'sgd'，Init_lr = 2e-3，weight_decay = 5e-4。（冻结）
    #           Init_Epoch = 0，UnFreeze_Epoch = 200，Freeze_Train = False，optimiz
er_type = 'sgd'，Init_lr = 2e-3，weight_decay = 5e-4。（不冻结）
    #       其中：由于从主干网络的预训练权重开始训练，主干的权值不一定适合目标检
测，需要更多的训练跳出局部最优解。
    #           UnFreeze_Epoch 可以在 200-300 之间调整，YOLOV5 和 YOLOX 均推
荐使用 300。
    #           Adam 相较于 SGD 收敛的快一些。因此 UnFreeze_Epoch 理论上可以小
一点，但依然推荐更多的 Epoch。
```

```
    #   （三）batch_size 的设置：
    #       在显卡能够接受的范围内，以大为好。显存不足与数据集大小无关，提示显存不足（OOM 或者 CUDA out of memory）请调小 batch_size。
    #       受到 BatchNorm 层影响，batch_size 最小为 2，不能为 1。
    #       正常情况下 Freeze_batch_size 建议为 Unfreeze_batch_size 的 1-2 倍。不建议设置的差距过大，因为关系到学习率的自动调整。
    #------------------------------------------------------------------------------------------------------------------#
    #-------------------------------------------------------------#
    #   冻结阶段训练参数
    #   此时模型的主干被冻结了，特征提取网络不发生改变
    #   占用的显存较小，仅对网络进行微调
    #   Init_Epoch          模型当前开始的训练世代，其值可以大于 Freeze_Epoch，如设置：
    #                       Init_Epoch = 60、Freeze_Epoch = 50、UnFreeze_Epoch = 100
    #                       会跳过冻结阶段，直接从 60 代开始，并调整对应的学习率。
    #                       （断点续练时使用）
    #   Freeze_Epoch        模型冻结训练的 Freeze_Epoch
    #                       (当 Freeze_Train=False 时失效)
    #   Freeze_batch_size   模型冻结训练的 batch_size
    #                       (当 Freeze_Train=False 时失效)
    #-------------------------------------------------------------#
    Init_Epoch          = 0
    Freeze_Epoch        = 50
    Freeze_batch_size   = 16
    #-------------------------------------------------------------#
    #   解冻阶段训练参数
    #   此时模型的主干不被冻结了，特征提取网络会发生改变
    #   占用的显存较大，网络所有的参数都会发生改变
    #   UnFreeze_Epoch          模型总共训练的 epoch
    #                           SGD 需要更长的时间收敛，因此设置较大的 UnFreeze_Epoch
    #                           Adam 可以使用相对较小的 UnFreeze_Epoch
    #   Unfreeze_batch_size     模型在解冻后的 batch_size
    #-------------------------------------------------------------#
    UnFreeze_Epoch      = 200
    Unfreeze_batch_size = 32
    #-------------------------------------------------------------#
    #   Freeze_Train    是否进行冻结训练
    #                   默认先冻结主干训练后解冻训练。
    #                   如果设置 Freeze_Train=False，建议使用优化器为 sgd
    #-------------------------------------------------------------#
    Freeze_Train        = True

    #-------------------------------------------------------------#
    #   其它训练参数：学习率、优化器、学习率下降有关
    #-------------------------------------------------------------#
    #-------------------------------------------------------------#
    #   Init_lr         模型的最大学习率
```

```
1251        #                           当使用 Adam 优化器时建议设置   Init_lr=6e-4
1252        #                           当使用 SGD 优化器时建议设置    Init_lr=2e-3
1253        #   Min_lr                  模型的最小学习率，默认为最大学习率的 0.01
1254        #------------------------------------------------------------------#
1255        Init_lr                 = 2e-3
1256        Min_lr                  = Init_lr * 0.01
1257        #------------------------------------------------------------------#
1258        #   optimizer_type      使用到的优化器种类，可选的有 adam、sgd
1259        #                           当使用 Adam 优化器时建议设置   Init_lr=6e-4
1260        #                           当使用 SGD 优化器时建议设置    Init_lr=2e-3
1261        #   momentum                优化器内部使用到的 momentum 参数
1262        #   weight_decay        权值衰减，可防止过拟合
1263        #                           adam 会导致 weight_decay 错误，使用 adam 时建议设置为 0。
1264        #------------------------------------------------------------------#
1265        optimizer_type          = "sgd"
1266        momentum                = 0.937
1267        weight_decay            = 5e-4
1268        #------------------------------------------------------------------#
1269        #   lr_decay_type       使用到的学习率下降方式，可选的有'step'、'cos'
1270        #------------------------------------------------------------------#
1271        lr_decay_type           = 'cos'
1272        #------------------------------------------------------------------#
1273        #   save_period         多少个 epoch 保存一次权值
1274        #------------------------------------------------------------------#
1275        save_period             = 10
1276        #------------------------------------------------------------------#
1277        #   save_dir            权值与日志文件保存的文件夹
1278        #------------------------------------------------------------------#
1279        save_dir                = 'logs'
1280        #------------------------------------------------------------------#
1281        #   eval_flag           是否在训练时进行评估，评估对象为验证集
1282        #                           安装 pycocotools 库后，评估体验更佳。
1283        #   eval_period         代表多少个 epoch 评估一次，不建议频繁的评估
1284        #                           评估需要消耗较多的时间，频繁评估会导致训练非常慢
1285        #   此处获得的 mAP 会与 get_map.py 获得的会有所不同，原因有二：
1286        #   （一）此处获得的 mAP 为验证集的 mAP。
1287        #   （二）此处设置评估参数较为保守，目的是加快评估速度。
1288        #------------------------------------------------------------------#
1289        eval_flag               = True
1290        eval_period             = 10
1291        #------------------------------------------------------------------#
1292        #   num_workers         用于设置是否使用多线程读取数据，1 代表关闭多线程
1293        #                           开启后会加快数据读取速度，但是会占用更多内存
1294        #                           keras 里开启多线程有些时候速度反而慢了许多
1295        #                           在 IO 为瓶颈的时候再开启多线程，即 GPU 运算速度远大于读取
1296    图片的速度。
1297        #------------------------------------------------------------------#
1298        num_workers             = 2
1299
1300        #------------------------------------------------------#
```

```
1301        #   train_annotation_path   训练图片路径和标签
1302        #   val_annotation_path     验证图片路径和标签
1303        #-------------------------------------------------------#
1304        train_annotation_path   = '2007_train.txt'
1305        val_annotation_path     = '2007_val.txt'
1306
1307        #-------------------------------------------------------#
1308        #   设置用到的显卡
1309        #-------------------------------------------------------#
1310        ngpus_per_node  = torch.cuda.device_count()
1311        if distributed:
1312            dist.init_process_group(backend="nccl")
1313            local_rank  = int(os.environ["LOCAL_RANK"])
1314            rank        = int(os.environ["RANK"])
1315            device      = torch.device("cuda", local_rank)
1316            if local_rank == 0:
1317                print(f"[{os.getpid()}] (rank = {rank}, local_rank = {local_rank}) training...")
1318                print("Gpu Device Count : ", ngpus_per_node)
1319        else:
1320            device          = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
1321            local_rank      = 0
1322
1323        if pretrained:
1324            if distributed:
1325                if local_rank == 0:
1326                    download_weights(backbone)
1327                dist.barrier()
1328            else:
1329                download_weights(backbone)
1330
1331        #-------------------------------------------------------#
1332        #   获取 classes 和 anchor
1333        #-------------------------------------------------------#
1334        class_names, num_classes = get_classes(classes_path)
1335        num_classes += 1
1336        anchors = get_anchors(input_shape, anchors_size, backbone)
1337
1338        model = SSD300(num_classes, backbone, pretrained)
1339        if not pretrained:
1340            weights_init(model)
1341        if model_path != '':
1342            #-------------------------------------------------------#
1343            #   权值文件请看 README，百度网盘下载
1344            #-------------------------------------------------------#
1345            if local_rank == 0:
1346                print('Load weights {}.'.format(model_path))
1347
1348            #-------------------------------------------------------#
1349            #   根据预训练权重的 Key 和模型的 Key 进行加载
1350            #-------------------------------------------------------#
```

```
            model_dict      = model.state_dict()
            pretrained_dict = torch.load(model_path, map_location = device)
            load_key, no_load_key, temp_dict = [], [], {}
            for k, v in pretrained_dict.items():
                if k in model_dict.keys() and np.shape(model_dict[k]) == np.shape(v):
                    temp_dict[k] = v
                    load_key.append(k)
                else:
                    no_load_key.append(k)
            model_dict.update(temp_dict)
            model.load_state_dict(model_dict)
            #------------------------------------------------------#
            #   显示没有匹配上的 Key
            #------------------------------------------------------#
            if local_rank == 0:
                print("\nSuccessful Load Key:", str(load_key)[:500], "……\nSuccessful Load Key Num:", len(load_key))
                print("\nFail To Load Key:", str(no_load_key)[:500], "……\nFail To Load Key num:", len(no_load_key))
                print("\n\033[1;33;44m 温馨提示，head 部分没有载入是正常现象，Backbone 部分没有载入是错误的。\033[0m")

        #---------------------#
        #   获得损失函数
        #---------------------#
        criterion       = MultiboxLoss(num_classes, neg_pos_ratio=3.0)
        #---------------------#
        #   记录 Loss
        #---------------------#
        if local_rank == 0:
            time_str        = datetime.datetime.strftime(datetime.datetime.now(),'%Y_%m_%d_%H_%M_%S')
            log_dir         = os.path.join(save_dir, "loss_" + str(time_str))
            loss_history    = LossHistory(log_dir, model, input_shape=input_shape)
        else:
            loss_history    = None

        #------------------------------------------------------------------#
        #   torch 1.2 不支持 amp，建议使用 torch 1.7.1 及以上正确使用 fp16
        #   因此 torch1.2 这里显示"could not be resolve"
        #------------------------------------------------------------------#
        if fp16:
            from torch.cuda.amp import GradScaler as GradScaler
            scaler = GradScaler()
        else:
            scaler = None

        model_train     = model.train()
        #----------------------------#
        #   多卡同步 Bn
```

```
1401        #----------------------------#
1402        if sync_bn and ngpus_per_node > 1 and distributed:
1403            model_train = torch.nn.SyncBatchNorm.convert_sync_batchnorm(model_train)
1404        elif sync_bn:
1405            print("Sync_bn is not support in one gpu or not distributed.")
1406
1407        if Cuda:
1408            if distributed:
1409                #----------------------------#
1410                #   多卡平行运行
1411                #----------------------------#
1412                model_train = model_train.cuda(local_rank)
1413                model_train = torch.nn.parallel.DistributedDataParallel(model_train, device_ids=
1414    [local_rank], find_unused_parameters=True)
1415            else:
1416                model_train = torch.nn.DataParallel(model)
1417                cudnn.benchmark = True
1418                model_train = model_train.cuda()
1419
1420        #----------------------------#
1421        #   读取数据集对应的 txt
1422        #----------------------------#
1423        with open(train_annotation_path, encoding='utf-8') as f:
1424            train_lines = f.readlines()
1425        with open(val_annotation_path, encoding='utf-8') as f:
1426            val_lines   = f.readlines()
1427        num_train   = len(train_lines)
1428        num_val     = len(val_lines)
1429
1430        if local_rank == 0:
1431            show_config(
1432                classes_path = classes_path, model_path = model_path, input_shape = input_s
1433    hape, \
1434                Init_Epoch = Init_Epoch, Freeze_Epoch = Freeze_Epoch, UnFreeze_Epoch =
1435    UnFreeze_Epoch, Freeze_batch_size = Freeze_batch_size, Unfreeze_batch_size = Unfreeze_b
1436    atch_size, Freeze_Train = Freeze_Train, \
1437                Init_lr = Init_lr, Min_lr = Min_lr, optimizer_type = optimizer_type, momentu
1438    m = momentum, lr_decay_type = lr_decay_type, \
1439                save_period = save_period, save_dir = save_dir, num_workers = num_worker
1440    s, num_train = num_train, num_val = num_val
1441            )
1442            #------------------------------------------------------#
1443            #   总训练世代指的是遍历全部数据的总次数
1444            #   总训练步长指的是梯度下降的总次数
1445            #   每个训练世代包含若干训练步长，每个训练步长进行一次梯度下降。
1446            #   此处仅建议最低训练世代，上不封顶，计算时只考虑了解冻部分
1447            #------------------------------------------------------#
1448            wanted_step = 5e4 if optimizer_type == "sgd" else 1.5e4
1449            total_step  = num_train // Unfreeze_batch_size * UnFreeze_Epoch
1450            if total_step <= wanted_step:
```

```
                if num_train // Unfreeze_batch_size == 0:
                    raise ValueError('数据集过小，无法进行训练，请扩充数据集。')
                wanted_epoch = wanted_step // (num_train // Unfreeze_batch_size) + 1
                print("\n\033[1;33;44m[Warning] 使用%s 优化器时，建议将训练总步长设置到%d 以上。\033[0m"%(optimizer_type, wanted_step))
                print("\033[1;33;44m[Warning] 本次运行的总训练数据量为%d，Unfreeze_batch_size 为%d，共训练%d 个 Epoch，计算出总训练步长为%d。\033[0m"%(num_train, Unfreeze_batch_size, UnFreeze_Epoch, total_step))
                print("\033[1;33;44m[Warning] 由于总训练步长为%d，小于建议总步长%d，建议设置总世代为%d。\033[0m"%(total_step, wanted_step, wanted_epoch))

        #------------------------------------------------------#
        #   主干特征提取网络特征通用，冻结训练可以加快训练速度
        #   也可以在训练初期防止权值被破坏。
        #   Init_Epoch 为起始世代
        #   Freeze_Epoch 为冻结训练的世代
        #   UnFreeze_Epoch 总训练世代
        #   提示 OOM 或者显存不足请调小 Batch_size
        #------------------------------------------------------#
        if True:
            UnFreeze_flag = False
            #------------------------------------#
            #   冻结一定部分训练
            #------------------------------------#
            if Freeze_Train:
                if backbone == "vgg":
                    for param in model.vgg[:28].parameters():
                        param.requires_grad = False
                else:
                    for param in model.mobilenet.parameters():
                        param.requires_grad = False

            #-------------------------------------------------------------------#
            #   如果不冻结训练的话，直接设置 batch_size 为 Unfreeze_batch_size
            #-------------------------------------------------------------------#
            batch_size = Freeze_batch_size if Freeze_Train else Unfreeze_batch_size

            #-------------------------------------------------------------------#
            #   判断当前 batch_size，自适应调整学习率
            #-------------------------------------------------------------------#
            nbs             = 64
            lr_limit_max    = 1e-3 if optimizer_type == 'adam' else 5e-2
            lr_limit_min    = 3e-4 if optimizer_type == 'adam' else 5e-5
            Init_lr_fit     = min(max(batch_size / nbs * Init_lr, lr_limit_min), lr_limit_max)
            Min_lr_fit      = min(max(batch_size / nbs * Min_lr, lr_limit_min * 1e-2), lr_limit_max * 1e-2)

            #---------------------------------------#
            #   根据 optimizer_type 选择优化器
            #---------------------------------------#
```

```
        optimizer = {
            'adam'  : optim.Adam(model.parameters(), Init_lr_fit, betas = (momentum, 0.999), weight_decay = weight_decay),
            'sgd'   : optim.SGD(model.parameters(), Init_lr_fit, momentum = momentum, nesterov=True, weight_decay = weight_decay)
        }[optimizer_type]

        #---------------------------------------#
        #   获得学习率下降的公式
        #---------------------------------------#
        lr_scheduler_func = get_lr_scheduler(lr_decay_type, Init_lr_fit, Min_lr_fit, UnFreeze_Epoch)

        #---------------------------------------#
        #   判断每一个世代的长度
        #---------------------------------------#
        batch_size = 4
        epoch_step      = num_train // batch_size
        epoch_step_val  = num_val // batch_size
        print(f"num_train is {num_train}")
        print(f"batch size is {batch_size}")
        print(f"num_val is {num_val}")
        if epoch_step == 0 or epoch_step_val == 0:
            raise ValueError("数据集过小，无法继续进行训练，请扩充数据集。")

        train_dataset   = SSDDataset(train_lines, input_shape, anchors, batch_size, num_classes, train = True)
        val_dataset     = SSDDataset(val_lines, input_shape, anchors, batch_size, num_classes, train = False)

        if distributed:
            train_sampler   = torch.utils.data.distributed.DistributedSampler(train_dataset, shuffle=True,)
            val_sampler     = torch.utils.data.distributed.DistributedSampler(val_dataset, shuffle=False,)
            batch_size      = batch_size // ngpus_per_node
            shuffle         = False
        else:
            train_sampler   = None
            val_sampler     = None
            shuffle         = True

        gen             = DataLoader(train_dataset, shuffle = shuffle, batch_size = batch_size, num_workers = num_workers, pin_memory=True,
                                     drop_last=True, collate_fn=ssd_dataset_collate, sampler=train_sampler)
        gen_val         = DataLoader(val_dataset  , shuffle = shuffle, batch_size = batch_size, num_workers = num_workers, pin_memory=True,
                                     drop_last=True, collate_fn=ssd_dataset_collate, sampler=val_sampler)
```

```
1551
1552            #----------------------#
1553            #    记录 eval 的 map 曲线
1554            #----------------------#
1555            if local_rank == 0:
1556                eval_callback   = EvalCallback(model, input_shape, anchors, class_names, nu
1557    m_classes, val_lines, log_dir, Cuda, \
1558                                                eval_flag=eval_flag, period=eval_period)
1559            else:
1560                eval_callback   = None
1561
1562            #---------------------------------------#
1563            #    开始模型训练
1564            #---------------------------------------#
1565            for epoch in range(Init_Epoch, UnFreeze_Epoch):
1566                #---------------------------------------#
1567                #    如果模型有冻结学习部分
1568                #    则解冻，并设置参数
1569                #---------------------------------------#
1570                if epoch >= Freeze_Epoch and not UnFreeze_flag and Freeze_Train:
1571                    batch_size = Unfreeze_batch_size
1572
1573                    #-------------------------------------------------------------------#
1574                    #    判断当前 batch_size，自适应调整学习率
1575                    #-------------------------------------------------------------------#
1576                    nbs             = 64
1577                    lr_limit_max    = 1e-3 if optimizer_type == 'adam' else 5e-2
1578                    lr_limit_min    = 3e-4 if optimizer_type == 'adam' else 5e-5
1579                    Init_lr_fit     = min(max(batch_size / nbs * Init_lr, lr_limit_min), lr_lim
1580    it_max)
1581                    Min_lr_fit      = min(max(batch_size / nbs * Min_lr, lr_limit_min * 1e
1582    -2), lr_limit_max * 1e-2)
1583                    #---------------------------------------#
1584                    #    获得学习率下降的公式
1585                    #---------------------------------------#
1586                    lr_scheduler_func = get_lr_scheduler(lr_decay_type, Init_lr_fit, Min_lr_fit,
1587     UnFreeze_Epoch)
1588
1589                    if backbone == "vgg":
1590                        for param in model.vgg[:28].parameters():
1591                            param.requires_grad = True
1592                    else:
1593                        for param in model.mobilenet.parameters():
1594                            param.requires_grad = True
1595
1596                    epoch_step      = num_train // batch_size
1597                    epoch_step_val  = num_val // batch_size
1598
1599                    if epoch_step == 0 or epoch_step_val == 0:
1600                        raise ValueError("数据集过小，无法继续进行训练，请扩充数据集。")
```

```
                    if distributed:
                        batch_size = batch_size // ngpus_per_node

                    gen             = DataLoader(train_dataset, shuffle = shuffle, batch_size =
batch_size, num_workers = num_workers, pin_memory=True,
                                                drop_last=True, collate_fn=ssd_dataset_co
llate, sampler=train_sampler)
                    gen_val         = DataLoader(val_dataset   , shuffle = shuffle, batch_size =
 batch_size, num_workers = num_workers, pin_memory=True,
                                                drop_last=True, collate_fn=ssd_dataset_co
llate, sampler=val_sampler)

                    UnFreeze_flag = True

                if distributed:
                    train_sampler.set_epoch(epoch)

                set_optimizer_lr(optimizer, lr_scheduler_func, epoch)

                fit_one_epoch(model_train, model, criterion, loss_history, eval_callback, optimi
zer, epoch,
                        epoch_step, epoch_step_val, gen, gen_val, UnFreeze_Epoch, Cuda, f
p16, scaler, save_period, save_dir, local_rank)

                if distributed:
                    dist.barrier()

            if local_rank == 0:
                loss_history.writer.close()


import os
import random
import xml.etree.ElementTree as ET

import numpy as np

from utils.utils import get_classes

#----------------------------------------------------------------------------------------------------------
-----------#
#   annotation_mode 用于指定该文件运行时计算的内容
#   annotation_mode 为 0 代表整个标签处理过程，包括获得 VOCdevkit/VOC2007/ImageSet
s 里面的 txt 以及训练用的 2007_train.txt、2007_val.txt
#   annotation_mode 为 1 代表获得 VOCdevkit/VOC2007/ImageSets 里面的 txt
#   annotation_mode 为 2 代表获得训练用的 2007_train.txt、2007_val.txt
#----------------------------------------------------------------------------------------------------------
-----------#
annotation_mode     = 0
```

```
#-------------------------------------------------------------------#
#   必须要修改，用于生成 2007_train.txt、2007_val.txt 的目标信息
#   与训练和预测所用的 classes_path 一致即可
#   如果生成的 2007_train.txt 里面没有目标信息
#   那么就是因为 classes 没有设定正确
#   仅在 annotation_mode 为 0 和 2 的时候有效
#-------------------------------------------------------------------#
classes_path        = 'model_data/classes.txt'
#--------------------------------------------------------------------------------------------------------------
#   trainval_percent 用于指定(训练集+验证集)与测试集的比例，默认情况下 (训练集+验证集):测试集 = 9:1
#   train_percent 用于指定(训练集+验证集)中训练集与验证集的比例，默认情况下 训练集:验证集 = 9:1
#   仅在 annotation_mode 为 0 和 1 的时候有效
#--------------------------------------------------------------------------------------------------------------
trainval_percent    = 0.9
train_percent       = 0.9
#-------------------------------------------------------#
#   指向 VOC 数据集所在的文件夹
#   默认指向根目录下的 VOC 数据集
#-------------------------------------------------------#
VOCdevkit_path  = 'VOCdevkit'

VOCdevkit_sets  = [('2007', 'train'), ('2007', 'val')]
classes, _      = get_classes(classes_path)

#-------------------------------------------------------#
#   统计目标数量
#-------------------------------------------------------#
photo_nums  = np.zeros(len(VOCdevkit_sets))
nums        = np.zeros(len(classes))
def convert_annotation(year, image_id, list_file):
    in_file = open(os.path.join(VOCdevkit_path, 'VOC%s/Annotations/%s.xml'%(year, image_id)), encoding='utf-8')
    tree=ET.parse(in_file)
    root = tree.getroot()

    for obj in root.iter('object'):
        difficult = 0
        if obj.find('difficult')!=None:
            difficult = obj.find('difficult').text
        cls = obj.find('name').text
        if cls not in classes or int(difficult)==1:
            continue
        cls_id = classes.index(cls)
        xmlbox = obj.find('bndbox')
        b = (int(float(xmlbox.find('xmin').text)), int(float(xmlbox.find('ymin').text)), int(float(xmlbox.find('xmax').text)), int(float(xmlbox.find('ymax').text)))
```

```
            list_file.write(" " + ",".join([str(a) for a in b]) + ',' + str(cls_id))
            nums[classes.index(cls)] = nums[classes.index(cls)] + 1
if __name__ == "__main__":
    random.seed(0)
    if " " in os.path.abspath(VOCdevkit_path):
        raise ValueError("数据集存放的文件夹路径与图片名称中不可以存在空格，否则会影响正常的模型训练，请注意修改。")
    if annotation_mode == 0 or annotation_mode == 1:
        print("Generate txt in ImageSets.")
        xmlfilepath = os.path.join(VOCdevkit_path, 'VOC2007/Annotations')
        saveBasePath = os.path.join(VOCdevkit_path, 'VOC2007/ImageSets/Main')
        temp_xml = os.listdir(xmlfilepath)
        total_xml = []
        for xml in temp_xml:
            if xml.endswith(".xml"):
                total_xml.append(xml)
        num = len(total_xml)
        list = range(num)
        tv = int(num*trainval_percent)
        tr = int(tv*train_percent)
        trainval= random.sample(list,tv)
        train = random.sample(trainval,tr)
        print("train and val size",tv)
        print("train size",tr)
        ftrainval = open(os.path.join(saveBasePath,'trainval.txt'), 'w')
        ftest = open(os.path.join(saveBasePath,'test.txt'), 'w')
        ftrain = open(os.path.join(saveBasePath,'train.txt'), 'w')
        fval = open(os.path.join(saveBasePath,'val.txt'), 'w')
        for i in list:
            name=total_xml[i][:-4]+'\n'
            if i in trainval:
                ftrainval.write(name)
                if i in train:
                    ftrain.write(name)
                else:
                    fval.write(name)
            else:
                ftest.write(name)
        ftrainval.close()
        ftrain.close()
        fval.close()
        ftest.close()
        print("Generate txt in ImageSets done.")
```

```
    if annotation_mode == 0 or annotation_mode == 2:
        print("Generate 2007_train.txt and 2007_val.txt for train.")
        type_index = 0
        for year, image_set in VOCdevkit_sets:
            image_ids = open(os.path.join(VOCdevkit_path, 'VOC%s/ImageSets/Main/%s.tx
t'%(year, image_set)), encoding='utf-8').read().strip().split()
            list_file = open('%s_%s.txt'%(year, image_set), 'w', encoding='utf-8')
            for image_id in image_ids:
                list_file.write('%s/VOC%s/JPEGImages/%s.jpg'%(os.path.abspath(VOCdevki
t_path), year, image_id))

                convert_annotation(year, image_id, list_file)
                list_file.write('\n')
            photo_nums[type_index] = len(image_ids)
            type_index += 1
            list_file.close()
        print("Generate 2007_train.txt and 2007_val.txt for train done.")

        def printTable(List1, List2):
            for i in range(len(List1[0])):
                print("|", end=' ')
                for j in range(len(List1)):
                    print(List1[j][i].rjust(int(List2[j])), end=' ')
                    print("|", end=' ')
                print()

        str_nums = [str(int(x)) for x in nums]
        tableData = [
            classes, str_nums
        ]
        colWidths = [0]*len(tableData)
        len1 = 0
        for i in range(len(tableData)):
            for j in range(len(tableData[i])):
                if len(tableData[i][j]) > colWidths[i]:
                    colWidths[i] = len(tableData[i][j])
        printTable(tableData, colWidths)

        if photo_nums[0] <= 500:
            print("训练集数量小于 500，属于较小的数据量，请注意设置较大的训练世代
（Epoch）以满足足够的梯度下降次数（Step）。")

from torchvision import transforms
from PIL import Image,ImageOps
import torch
import numpy as np
import matplotlib.pyplot as plt
import os

```

```
import cv2
import numpy as np
import torch
import torchvision.transforms as transforms
from efficientnet_pytorch import EfficientNet
import torchvision
import torch.nn as nn
import torch.optim as optim
import torchvision.models as models
#得到将三张图片堆叠并转化为 tensor 格式后标准化的 img_tensor
def merge(pic1,pic2,pic3):
    # 读取灰度图片并调整大小

    img1 = cv2.imread(pic1, cv2.IMREAD_GRAYSCALE)
    # 检查图像是否正确加载
    if img1 is None:
        print("error picture 1 upload")
    else:
        img1 = cv2.resize(img1, (224, 224))
        print("successfully upload and resize picture 1")

    img2 = cv2.imread(pic2, cv2.IMREAD_GRAYSCALE)
    if img2 is None:
        print("error picture 2 upload")
    else:
        img2 = cv2.resize(img2, (224, 224))
        print("successfully upload and resize picture 2")

    img3 = cv2.imread(pic3, cv2.IMREAD_GRAYSCALE)
    if img3 is None :
        print("error picture 3 upload")
    else:
        img3 = cv2.resize(img3, (224, 224))
        print("successfully upload and resize picture 3")

    # 将三张灰度图片堆叠为一个 RGB 图像
    img = np.zeros((224, 224, 3), dtype=np.float32)
    img[..., 0] = img1.astype(np.float32) / 255.0
    img[..., 1] = img2.astype(np.float32) / 255.0
    img[..., 2] = img3.astype(np.float32) / 255.0

    # 转化为 tensor 格式并标准化
    img_tensor = transforms.ToTensor()(img)
    normalize = transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
    img_tensor = normalize(img_tensor)

    return img_tensor
```

```
def cv_imread(filepath):
    """
    功能相当于 cv2.imread
    :param filepath:读取的图片地址（包含名字），如 "C:\\待处理图片\\宝马.jpg"
    """
    cv_imr = cv2.imdecode(np.fromfile(filepath,dtype=np.uint8),cv2.IMREAD_GRAYSCALE)
    return cv_imr
    import matplotlib.pyplot as plt
    dir_save_path    = './datalogs/Totaldata/' #已经裁切好的图片路径
    rating_list = os.listdir(dir_save_path)
    number = 0
    for rating in rating_list:
        if(len(rating)<2): #选中 abcde 文件夹
            people_list = os.listdir(dir_save_path+rating)
            for people in people_list:
                pic_list = os.listdir(dir_save_path+rating+'/'+people)
                #if(len(pic_list) == 5):
                #      print(people)
                if(len(pic_list)>=4):
                    img = merge_pic4(dir_save_path+rating+'/'+people+'/'+pic_list[0],
                                     dir_save_path+rating+'/'+people+'/'+pic_list[1],
                                     dir_save_path+rating+'/'+people+'/'+pic_list[2],
                                     dir_save_path+rating+'/'+people+'/'+pic_list[3])
                    number += 1
                    img = Image.fromarray(img.numpy().transpose(1, 2, 0).astype('uint8'))
                    #将一个 PyTorch 图像张量转换为一个 Pillow 图像对象
                    #保存 PIL Image 对象到文件
                    folder_path = './hello/'+rating
                    if not os.path.exists(folder_path):
                        os.makedirs(folder_path)
                    img.save(os.path.join(folder_path, str(number)+'.png'))
    print(number)
    import os
    import glob
    import cv2
    import random
    from pathlib import Path

    #补边,这一步主要是为了将图片填充为正方形，防止直接 resize 导致图片变形
    def expend_img(img):
        '''
        :param img: 图片数据
        :return:
        '''
        fill_pix=[122,122,122] #填充色素，可自己设定
        h,w=img.shape[:2]
        if h>=w: #左右填充
            padd_width=int(h-w)//2
            padd_top,padd_bottom,padd_left,padd_right=0,0,padd_width,padd_width #各个
```

```
方向的填充像素
        elif h<w: #上下填充
            padd_high=int(w-h)//2
            padd_top,padd_bottom,padd_left,padd_right=padd_high,padd_high,0,0 #各个方向的填充像素
        new_img = cv2.copyMakeBorder(img,padd_top,padd_bottom,padd_left,padd_right,cv2.BORDER_CONSTANT, value=fill_pix)
        return new_img

    #切分训练集和测试集，并进行补边处理
    def split_train_test(img_dir,save_dir,train_val_num):
        '''
        :param img_dir: 原始图片路径，注意是所有类别所在文件夹的上一级目录
        :param save_dir: 保存图片路径
        :param train_val_num: 切分比例
        :return:
        '''
        img_dir_list=glob.glob(img_dir+os.sep+"*")#获取每个类别所在的路径（一个类别对应一个文件夹）
        #img_dir_list 放着 A,B,C,D,E 的分类文件名
        for class_dir in img_dir_list:
            class_name=class_dir.split(os.sep)[-1] #获取当前类别
            img_list=glob.glob(class_dir+os.sep+"*") #获取每个类别文件夹下的所有图片
            all_num=len(img_list) #获取总个数
            #从当前类别的图像列表中随机选取一部分图像作为训练集。
            train_list=random.sample(img_list,int(all_num*train_val_num)) #训练集图片所在路径
            save_train=save_dir+os.sep+"train"+os.sep+class_name
            save_val=save_dir+os.sep+"val"+os.sep+class_name
            os.makedirs(save_train,exist_ok=True)
            os.makedirs(save_val,exist_ok=True) #建立对应的文件夹

            print(class_name+" train num",len(train_list))
            print(class_name+" test num",all_num-len(train_list))
            #保存切分好的数据集

            for imgpath in img_list:
                imgname=Path(imgpath).name #获取文件名
                if imgpath in train_list:
                    img=cv_imread(imgpath)
                    new_img=expend_img(img)
                    cv2.imwrite(save_train+os.sep+imgname,new_img)
                else: #将除了训练集意外的数据均视为验证集
                    img = cv2.imread(imgpath)
                    new_img = expend_img(img)
                    cv2.imwrite(save_val + os.sep + imgname, new_img)

                    #将处理后的图像保存到对应的训练集或验证集文件夹中
        print("split train and test finished !")

```

```
1951        split_train_test('./hello','./resnet_datasets',0.9)
1952        import torch
1953        import torchvision
1954        import torchvision.transforms as transforms
1955        import torch.nn as nn
1956        import torch.optim as optim
1957        from efficientnet_pytorch import EfficientNet
1958        import torchvision.models as models
1959
1960        device="cuda" if torch.cuda.is_available() else "cpu"
1961        print("device is ",device)
1962        # 定义训练数据预处理
1963        transform_train = transforms.Compose(
1964            [transforms.RandomResizedCrop(224),
1965             transforms.RandomHorizontalFlip(),
1966             transforms.ToTensor(),
1967             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
1968
1969        # 定义测试数据预处理
1970        transform_test = transforms.Compose(
1971            [transforms.Resize(224),
1972             transforms.ToTensor(),
1973             transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
1974
1975        # 加载训练集和测试集
1976        trainset = torchvision.datasets.ImageFolder(root='./TRAIN/train', transform=transform_trai
1977    n)
1978        trainloader = torch.utils.data.DataLoader(trainset, batch_size=16,shuffle=True, num_worke
1979    rs=2)
1980
1981        testset = torchvision.datasets.ImageFolder(root='./TRAIN/val', transform=transform_test)
1982        testloader = torch.utils.data.DataLoader(testset, batch_size=16,shuffle=False, num_worker
1983    s=2)
1984
1985        # 定义 EfficientNet 模型
1986        #net = EfficientNet.from_name('efficientnet-b0')
1987        # net = models.resnet50(pretrained=True)
1988        # # num_ftrs = net._fc.in_features
1989        # # net._fc = nn.Linear(num_ftrs, 5)
1990        # net = net.to(device)
1991
1992        # 定义 ResNet50 模型
1993        net = models.resnet34(pretrained=True)
1994
1995        # 将最后一个全连接层的输出维度改为自定义维度
1996        num_ftrs = net.fc.in_features
1997        net.fc = nn.Linear(num_ftrs, 5)  # num_classes 表示自定义的类别数
1998        net = net.to(device)
1999        # # 定义训练数据预处理
2000        # transform_train = transforms.Compose(
```

```
2001        #         [transforms.RandomResizedCrop(224),
2002        #           transforms.RandomHorizontalFlip(),
2003        #           transforms.ToTensor()])
2004        #           #transforms.Normalize(mean=[0.485, 0.456, 0.406,0.5], std=[0.229, 0.224, 0.225,
2005    0.5])])
2006
2007        # # 定义测试数据预处理
2008        # transform_test = transforms.Compose(
2009        #         [transforms.Resize(224),
2010        #           transforms.ToTensor()])
2011        #         #transforms.Normalize(mean=[0.485, 0.456, 0.406,0.5], std=[0.229, 0.224, 0.225,0.
2012    5])])
2013
2014        # # 加载训练集和测试集
2015        # trainset = torchvision.datasets.ImageFolder(root='./HELLO/train', transform=transform_t
2016    rain)
2017        # trainloader = torch.utils.data.DataLoader(trainset, batch_size=16,shuffle=True, num_wo
2018    rkers=2)
2019
2020        # testset = torchvision.datasets.ImageFolder(root='./HELLO/val', transform=transform_tes
2021    t)
2022        # testloader = torch.utils.data.DataLoader(testset, batch_size=16,shuffle=False, num_work
2023    ers=2)
2024
2025        # # 定义 EfficientNet 模型
2026        # #net = EfficientNet.from_name('efficientnet-b0')
2027        # # net = models.resnet50(pretrained=True)
2028        # # # num_ftrs = net._fc.in_features
2029        # # # net._fc = nn.Linear(num_ftrs, 5)
2030        # # net = net.to(device)
2031
2032        # # 定义 ResNet50 模型
2033        # resnet50 = models.resnet50(pretrained=True)
2034
2035        # #将输入层改为 4 通道
2036        # net = nn.Sequential(*list(resnet50.children())[:-1])
2037        # net[0] = nn.Conv2d(4, 64, kernel_size=7, stride=2, padding=3, bias=False)
2038
2039        # # 将最后一个全连接层的输出维度改为自定义维度
2040        # new_fc_layer = nn.Linear(resnet50.fc.in_features, 5)
2041        # net.add_module('fc', new_fc_layer)
2042
2043        # 定义损失函数和优化器
2044        criterion = nn.CrossEntropyLoss()
2045        #optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
2046        optimizer = optim.Adam(net.parameters(), lr=0.001, weight_decay=0.0001)
2047        def lrfn(num_epoch, optimzer):
2048            lr_start = 0.00001  # 初始值
2049            max_lr = 0.0004  # 最大值
2050            lr_up_epoch = 10  # 学习率上升 10 个 epoch
```

```
            lr_sustain_epoch = 5  # 学习率保持不变
            lr_exp = .8  # 衰减因子
            if num_epoch < lr_up_epoch:  # 0-10 个 epoch 学习率线性增加
                lr = (max_lr - lr_start) / lr_up_epoch * num_epoch + lr_start
            elif num_epoch < lr_up_epoch + lr_sustain_epoch:  # 学习率保持不变
                lr = max_lr
            else:  # 指数下降
                lr = (max_lr - lr_start) * lr_exp ** (num_epoch - lr_up_epoch - lr_sustain_e
poch) + lr_start
            for param_group in optimzer.param_groups:
                param_group['lr'] = lr
            return optimzer
        optimizer = lrfn(10,optimizer)

        # 训练模型
        for epoch in range(500):
            running_loss = 0.0
            #print("----------Start Train Epoch %d----------" % (epoch + 1))
            for i, data in enumerate(trainloader,0):
                inputs, labels = data
                inputs, labels = inputs.to(device), labels.to(device)

                optimizer.zero_grad()

                outputs = net(inputs)
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()

                running_loss += loss.item()

            print('[%d] loss: %.3f' %
                  (epoch + 1, running_loss / len(trainloader)))

        print('Finished Training')

        # 测试模型
        correct = 0
        total = 0
        with torch.no_grad():
            for data in testloader:
                images, labels = data
                images, labels = images.to(device), labels.to(device)
                outputs = net(images)
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

        print('Accuracy of the network on the test images: %d %%' % (
            100 * correct / total))
```

```
2101
2102    if __name__ == "__main__":
2103        ssd = SSD()
2104        #----------------------------------------------------------------------#
2105        #   crop                指定了是否在单张图片预测后对目标进行截取
2106        #   count               指定了是否进行目标的计数
2107        #   crop、count 仅在 mode='predict'时有效
2108        #----------------------------------------------------------------------#
2109        crop            = True
2110        count           = True
2111        #----------------------------------------------------------------------#
2112        #   dir_origin_path     指定了用于检测的图片的文件夹路径
2113        #   dir_save_path       指定了检测完图片的保存路径
2114        #
2115        #   dir_origin_path 和 dir_save_path 仅在 mode='dir_predict'时有效
2116        #----------------------------------------------------------------------#
2117        dir_origin_path = "WaitPredictPic/"
2118        dir_save_path   = "PredictOutcome/"
2119
2120        success = 0 #记录总的成功预测的数量
2121        outcome = 0 #记录一张图片是否预测成功
2122        prePic_list = os.listdir(dir_origin_path)
2123        for prepic in prePic_list:
2124            img = Image.open(dir_origin_path+prepic)
2125            try:
2126                image = Image.open(dir_origin_path+prepic)
2127            except:
2128                print('Open Error! Try again!')
2129                continue
2130            else:
2131                r_image, outcome,position_list = ssd.detect_image(image,prepic, crop = crop,
2132    count=count)
2133                success += outcome
2134                r_image.save(dir_save_path+prepic)
2135        print(f"Accuracy: {success}/{len(prePic_list)}={success/len(prePic_list)*100}%")
2136
2137        #----------------------------------------------------------------------#
2138        #   将单个患者的图片预测裁剪                                            #
2139        #----------------------------------------------------------------------#
2140        import time
2141
2142        import cv2
2143        import numpy as np
2144        from PIL import Image
2145        import os
2146        from ssd import SSD
2147        import torch
2148        import merge
2149        import matplotlib.pyplot as plt
2150        import torchvision.transforms as transforms
```

```
    import torchvision.models
    import random
    #此函数将 dir_origin_path 文件中的图片检测并裁剪
    #然后将裁剪后的图片展示保存到 dir_save_path 文件中


    class in_image:
        def __init__(self, name,dir_origin_path,dir_save_path ):
            #初始化属性：图片原始路径裁剪后保存路径和图片分类
            self.name=name
            self.dir_origin_path = dir_origin_path
            self.dir_save_path=dir_save_path
            self.out_class ='roi'
            # 准备分类模型
            self.device="cuda" if torch.cuda.is_available() else "cpu"
            self.net = torchvision.models.resnet50(pretrained=False)
            model_weights_path = '/home/ubuntu/django_project/argon-dashboard-django/ap
ps/Maxillo_bone_detection/code/resnet_logs/resnet50_weights_60.pth'  # 替换为已经训练过的
模型权重文件路径
            num_ftrs = self.net.fc.in_features
            # 将最后一个全连接层的输出维度改为自定义维度
            self.net.fc = torch.nn.Linear(num_ftrs, 5)  # 假设有 5 个类别
            self.net = self.net.to(self.device)
            self.net.load_state_dict(torch.load(model_weights_path,map_location='cpu')) # 加
载自己的预训练权重
            self.net.eval()
        #裁剪图片
        #将 dir_origin_path 文件中的图片检测并裁剪
        #将裁剪后的图片展示保存到 dir_save_path 文件中

        def crop(self):
            print(f"begin to crop and save.....")
            ssd = SSD()
            crop            = True
            count           = True
            #这里是保存一个患者最原始的图片的文件夹，文件夹内有一个患者多个层面
的图片
            outcome = 0 #记录图片是否预测成功

            #检测并裁剪出 roi 部分
            #prePic_list = os.listdir(self.dir_origin_path)
            #for prepic in prePic_list:
            #    img = Image.open(self.dir_origin_path+'/'+prepic)
            try:
                image = Image.open(self.dir_origin_path)
            except:
                print('Open Error! Try again!')
                return -1
            r_image, outcome,position_list = ssd.detect_image(image,self.name,crop = crop,
count=count)
```

```
                #保存并展示 crop 后的图片
                if(outcome != 0):
                    crop_image = image.crop([position_list[0], position_list[1], position_list[2], position_list[3]])
                    crop_save_path=os.path.join(self.dir_save_path+'/crop_'+self.name+".jpg")
                    crop_image.save(crop_save_path, quality=95, subsampling=0)
                    #Image.show(crop_image)
                    print(f'successfully crop and save')
                #print(f"successfully crop and save all!")
                    return outcome,crop_save_path
                else:
                    return outcome,''


            #运行 resnet50 模型并返回分级内容
            #dir_save_path 文件中有患者各层面的已被检测并裁剪的图片
            #folder_path 中保存了患者 3 层面融合后的图片
            def merge_classify(self,dir_save_path):
                # 获取文件夹中所有图片文件的列表
                #image_files = [f for f in os.listdir(dir_save_path) if f.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp', '.tiff'))]
                # 检查是否有足够的图片文件
                #if len(image_files) < 3:
                #    print("Not enough image files in the folder.")
                #else:
                    # 从图片列表中随机选取 3 张图片
                #    pic_list = random.sample(image_files, 3)
                    #将三层面的图片进行融合
                #    img = merge.merge(dir_save_path+'/'+pic_list[0],
                #                      dir_save_path+'/'+pic_list[1],
                #                      dir_save_path+'/'+pic_list[2])
                #    img = Image.fromarray(img.numpy().transpose(1, 2, 0).astype('uint8'))
                    #保存 PIL Image 对象到文件 folder_path
                #    img.save(os.path.join(folder_path+'/'+'merge.png'))
                #    print("successfully merge and save!")
                    #图片预处理流水线
                    transform_test = transforms.Compose(
                    [transforms.Resize(224),
                    transforms.ToTensor(),
                    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])
                    image_path =dir_save_path#folder_path+'/'+'merge.png' 图片路径
                    imag = Image.open(image_path).convert('RGB')  # 确保图片是 RGB 格式
                    # 应用预处理流水线
                    image_tensor = transform_test(imag).unsqueeze(0)
                    #将数据移动到对应的设备
                    # 将输入张量移动到 GPU 上
                    image_tensor = image_tensor.to(self.device)
                    # 进行模型推理
                    with torch.no_grad():  # 不需要计算梯度，节省内存和计算资源
```

```
                        output = self.net(image_tensor)
                        # 找到概率最高的类别索引
                        _, predicted = torch.max(output.data, 1)
                    print("successfully classify!")
                    class_names=['A','B','C','D','E'] #定义 5 个类别
                    return class_names[predicted.item()]

    from django.db import models
    from django.contrib.auth.models import User
    from django.conf import settings
    from imagekit.models import ImageSpecField
    from imagekit.processors import ResizeToFill

    #患者信息模型
    class PatientInfo(models.Model):
        name=models.CharField(max_length=128)
        age=models.IntegerField(default=18)
        image_id=models.CharField(max_length=128,default="#")
        image_pre= models.ImageField(upload_to='pre/')
        image_crop=models.ImageField(upload_to='post/')
        image_class= models.CharField(max_length=1,default=" ")

    from django.urls import path, re_path
    from apps.home import views
    from django.conf.urls.static import static
    from django.conf import settings
    from django.views.static import serve
    urlpatterns = [

        # The home page
        path('', views.index, name='home'),
        path('info_submit', views.info_submit, name='info_submit'),
        path('display_information/<int:pk>/', views.display_information, name='display_infor
mation'),
        # Matches any html file
        #media 配置——配合 settings 中的 MEDIA_ROOT 的配置，就可以在浏览器的地址
栏访问 media 文件夹及里面的文件了
        re_path(r'apps/home/images/(?P<path>.*)$',serve,{'document_root':settings.MEDIA_R
OOT}),
        re_path(r'^.*\.*', views.pages, name='pages'),

    ]

    from django import forms
    from .models import PatientInfo
    class PatientForm(forms.ModelForm):
        name=forms.CharField(help_text="输入患者名称")
        age=forms.IntegerField(help_text="输入患者年龄",initial=0)
        image_id=forms.CharField(widget=forms.HiddenInput(),required=False)

```

```
2301            image_pre=forms.ImageField(help_text="请上传患者 CBCT 图像")
2302            image_crop=forms.ImageField(widget=forms.HiddenInput(),required=False)
2303            image_class=forms.CharField(widget=forms.HiddenInput(),required=False)
2304            class Meta:
2305                model = PatientInfo
2306                fields = ['name', 'age', 'image_pre',]
2307
2308        from django.core.files import File
2309        from io import BytesIO
2310        from django import template
2311        from django.contrib.auth.decorators import login_required
2312        from django.http import HttpResponse, HttpResponseRedirect
2313        from django.template import loader
2314        from django.urls import reverse
2315        from .models import PatientInfo
2316        from .forms import PatientForm
2317        from django.shortcuts import render,redirect,get_object_or_404
2318        import sys
2319        #windows 路径
2320        #sys.path.append(r"D:\\1Desktop\\大创\django_project\\argon-dashboard-django\\apps\\Maxi
2321    llo_bone_detection\\code\\")
2322        #sys.path.append(r"D:\\1Desktop\\大创\\django_project\\argon-dashboard-django\\apps\\Max
2323    illo_bone_detection\\code\\")
2324        #linux 路径
2325        sys.path.append('/home/ubuntu/django_project/argon-dashboard-django/apps/Maxillo_bone_
2326    detection/code/')
2327        import user_input_predict as my_model
2328
2329        @login_required(login_url="/login/")
2330        def index(request):
2331            context = {'segment': 'index'}
2332            """
2333            name='***'
2334            id='***'
2335            #查询数据库中是否对应的图片
2336            try:
2337                patient = PatientInfo.objects.get(name=name,image_id=id)
2338            except patient.DoesNotExist:
2339                # 处理找不到对象的情况
2340                print(f"No myImage instance found with name={name},image_id={id}")
2341            #如果找到了对象，获取图片的路径
2342            if patient:
2343                dir_origin_path = image_instance.image.path
2344            else:
2345                # 处理未找到对象时的情况，例如设置 dir_origin_path 为 None 或空字符串
2346                dir_origin_path = None
2347            图片路径
2348            dir_origin_path='./imgs/dir_origin_path'
2349            #图片裁剪后保存路径
2350            dir_save_path='./imgs/dir_save_path'#linux 路径,需改动
```

```
2351            #随机选取三张图片融合为一张保存路径
2352            folder_path='./imgs/folder_path'#linux 路径,需改动
2353            my_image=my_model.in_image(dir_origin_path,dir_save_path)
2354            PatientInfo.image_crop=my_image.crop()
2355            PatientInfo.image_class=my_image.merge_classify(my_image.dir_save_path,folder_pat
2356    h)
2357            # 保存实例到数据库
2358            PatientInfo.save()
2359            """
2360            html_template = loader.get_template('home/index.html')
2361            return HttpResponse(html_template.render(context, request))
2362
2363        def info_submit(request):
2364            context={'segment':'submit'}
2365            if request.method == 'POST':
2366                form = PatientForm(request.POST, request.FILES)
2367                if form.is_valid():
2368                    Submit_PatientInfo=form.save()
2369                    Submit_PatientInfo.image_id=Submit_PatientInfo.name
2370                    dir_origin_path=Submit_PatientInfo.image_pre.path
2371                    dir_save_path='apps/home/images/post'
2372                    my_image=my_model.in_image(Submit_PatientInfo.name,dir_origin_path,dir
2373    _save_path)
2374                    outcome,crop_image_path = my_image.crop()
2375                    if outcome:
2376                        # 打开裁剪后的图片文件
2377                        with open(crop_image_path, 'rb') as image_file:
2378                            # 使用 FieldFile 的 save 方法保存文件
2379                            Submit_PatientInfo.image_crop.save('crop_'+Submit_PatientInfo.na
2380    me+".jpg",File(image_file))
2381                            my_image.dir_save_path=Submit_PatientInfo.image_crop.path
2382                            Submit_PatientInfo.image_class=my_image.merge_classify(my_im
2383    age.dir_save_path)
2384                            Submit_PatientInfo.save()
2385                    else:
2386                        Submit_PatientInfo.image_class=' '
2387                        Submit_PatientInfo.save()
2388                    return redirect('display_information', pk=Submit_PatientInfo.pk)
2389            else:
2390                    form = PatientForm()
2391            return render(request,'home/submit.html',{'form':form})
2392
2393        def display_information(request,pk):
2394            patient_info = get_object_or_404(PatientInfo,pk=pk)
2395            print(patient_info)
2396            # 准备上下文数据
2397            context = {
2398                'patient_info': patient_info,
2399                # 添加其他你想要展示的字段...
2400            }
```

```
2401            return  render(request,'home/display_information.html',context)
2402
2403        @login_required(login_url="/login/")
2404        def  pages(request):
2405            context  =  {}
2406            #  All  resource  paths  end  in  .html.
2407            #  Pick  out  the  html  file  name  from  the  url.  And  load  that  template.
2408            try:
2409
2410                load_template  =  request.path.split('/')[-1]
2411
2412                if  load_template  ==  'admin':
2413                    return  HttpResponseRedirect(reverse('admin:index'))
2414                context['segment']  =  load_template
2415
2416                html_template  =  loader.get_template('home/'  +  load_template)
2417                return  HttpResponse(html_template.render(context,  request))
2418
2419            except  template.TemplateDoesNotExist:
2420
2421                html_template  =  loader.get_template('home/page-404.html')
2422                return  HttpResponse(html_template.render(context,  request))
2423
2424            except:
2425                html_template  =  loader.get_template('home/page-500.html')
2426                return  HttpResponse(html_template.render(context,  request))
2427
2428
```