图像分割技术实现

[前言 2](#_Toc68953444)

[一.图像映射 2](#_Toc68953445)

[1.1.将采集的待标注图像拷贝至OirImage; 2](#_Toc68953446)

[1.2.运行\_01WarpMain.py实现图像的映射 2](#_Toc68953447)

[1.3.在WarpedImg文件夹中得到映射后的图像 2](#_Toc68953448)

[二.标注图像，生成训练数据 2](#_Toc68953449)

[2.1.完成图像标注 3](#_Toc68953450)

[2.1.1.配置labelme环境 3](#_Toc68953451)

[2.1.2.使用labelme进行标注 3](#_Toc68953452)

[2.2.转换标签数据 3](#_Toc68953453)

[2.2.1.标注图像和标签拷贝至OriLabelDataset； 3](#_Toc68953454)

[2.2.2.运行\_02GemerateDataset.py，得到适用于神经网络训练的标签数据. 3](#_Toc68953455)

[2.2.3整理训练数据集 3](#_Toc68953456)

[三.数据集封装 2](#_Toc68953457)

[3.1.数据增强与数据标准化 2](#_Toc68953458)

[3.1.1.数据增强 2](#_Toc68953459)

[3.1.2数据加载 4](#_Toc68953460)

[3.1.3封装dataset 5](#_Toc68953461)

[四.搭建模型 5](#_Toc68953462)

[4.1.相关基础 6](#_Toc68953463)

[4.1.1.解读层作用 6](#_Toc68953464)

[4.2.封装网络模块 7](#_Toc68953465)

[4.3.组网 7](#_Toc68953466)

[4.4.设计loss函数 8](#_Toc68953467)

[五.配置模型准备训练 9](#_Toc68953468)

[5.1. \_03Training.py代码解读 9](#_Toc68953469)

[六.模型训练与检测 10](#_Toc68953470)

[6.1.训练网络与调参 10](#_Toc68953471)

[6.2．模型效果检测 12](#_Toc68953472)

[6.2.1.查看网络在具体图片的效果 12](#_Toc68953473)

[6.2.2.查看网络整体效果 12](#_Toc68953474)

[6.2.3.运行时间 15](#_Toc68953475)

# 前言

由于硬件配置原因，本小组使用CPU进行深度学习的训练，对于部分代码有一定的修改。对组委会审核造成的不便，我们深表歉意。

# 一.图像映射

## 1.1.将采集的待标注图像拷贝至OirImage;

## 1.2.运行\_01WarpMain.py实现图像的映射

（1）先通过相机的标定（见交通灯识别模块介绍）获得相机的内参矩阵k，畸变矩阵dis。变换矩阵H通过cv2.getPerspectiveTransform()函数获得。

（2）调用cv2.undistort()函数实现图片的去除畸变

（3）调用cv2.warpPerspective(）完成透视变换

（4）输出图像存储在WarpedImg文件夹

## 1.3.在WarpedImg文件夹中得到映射后的图像

# 二.标注图像，生成训练数据

## 2.1.完成图像标注

### 2.1.1.配置labelme环境

下载安装anaconda，通过Prompt指令激活labelme环境。具体指令见图1-1。

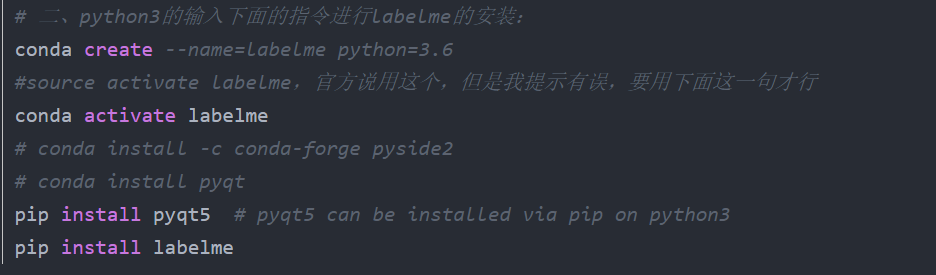


图2-1激活labelme环境

### 2.1.2.使用labelme进行标注

通过open读取文件，选择create polygons对想要的区域进行编辑，生成对应的json文件。

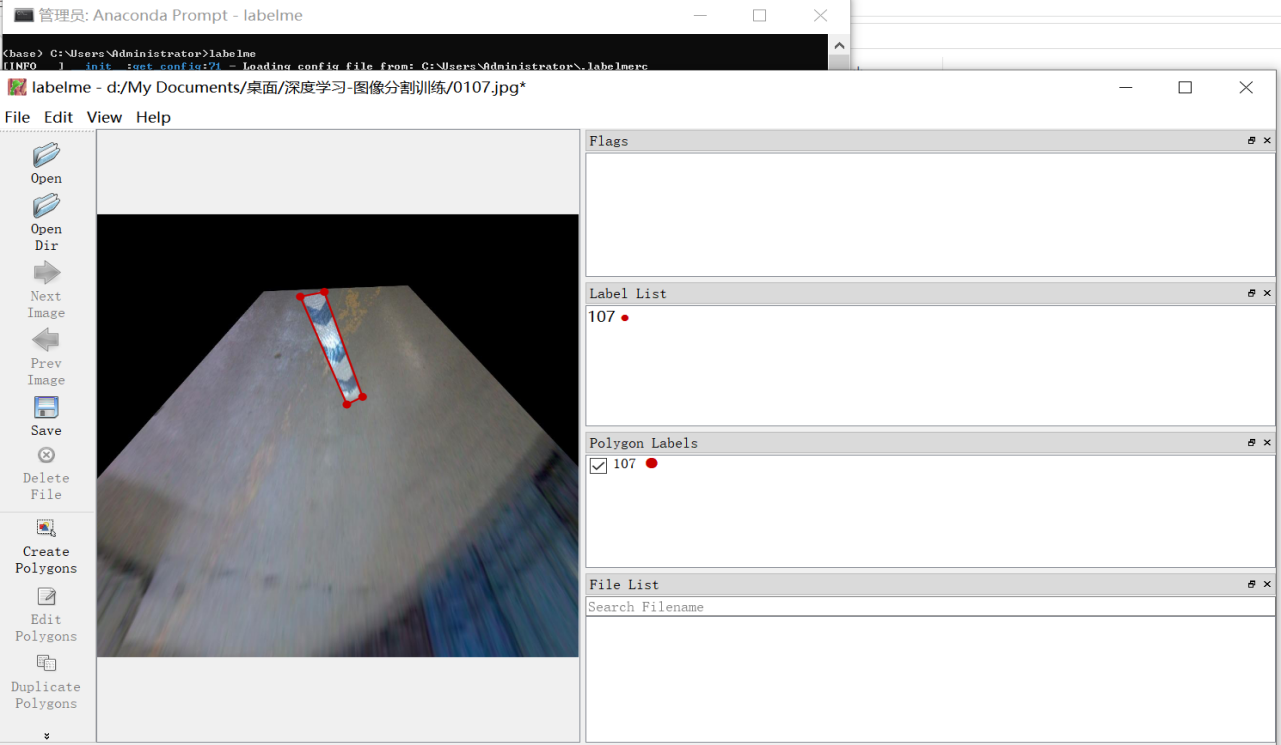


图2-2标注车道线

## 2.2.转换标签数据

### 2.2.1.标注图像和标签拷贝至OriLabelDataset；

### 2.2.2.运行\_02GemerateDataset.py，得到适用于神经网络训练的标签数据.

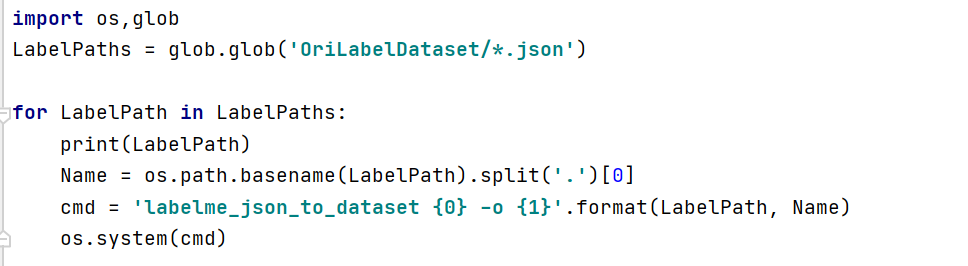


图2-3\_02GemerateDataset.py

代码解释：

使用glob.glob()获取“.json”结尾文件名，调用os.system()函数，对目录下所有标签执行labelme\_json\_to\_dataset.exe，将json文件转化为可用于训练的数据。

补充：需要调整编译环境为anaconda配置的python环境或者设置绝对路径，否则无法找到exe文件

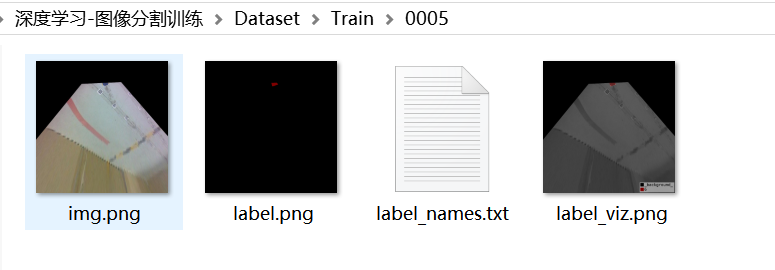


图2-4生成训练文件

### 2.2.3整理训练数据集

(1)将生成数据拷贝至Dataset\Train下作为训练集数据

(2)Val文件夹下需要随机拷贝数个文件作为测试集

# 三.数据集封装

## 3.1.数据增强与数据标准化

### 3.1.1.数据增强

调用transforms.Compose()函数整合以下变化增强函数：

**A．**对于TrainImg有：

（1）transforms.RandomAffine()-保持中心不变的随机仿射变换。度数范围10度，水平和垂直平移的最大绝对分数为0.1，随机缩放比例scale为0.5到2，shear=10意味着在 x 轴上随机选择 (-10, 10) 之间的角度进行错切，在 y 轴上不错切。

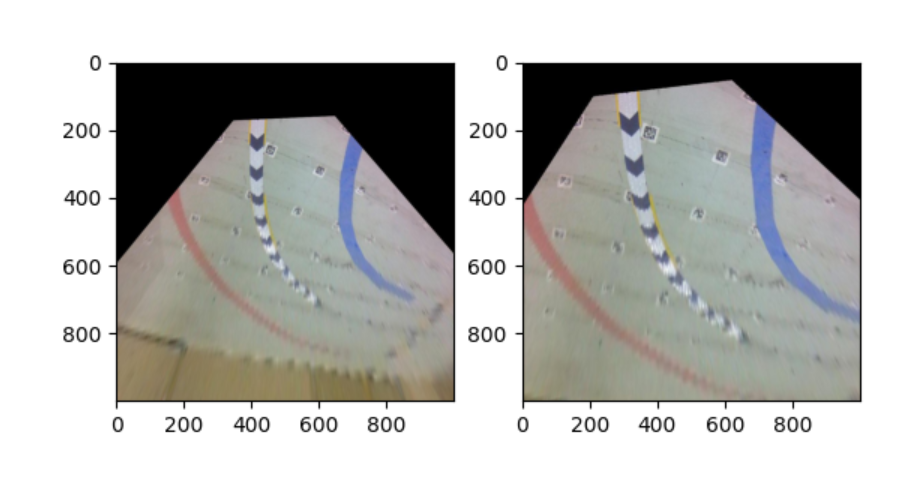


图3-1transforms.RandomAffine()展示

（2）transforms.RandomHorizontalFlip()-依概率p水平翻转,默认概率0.5

（3） transforms.RandomResizedCrop(InputImgSize, scale=，interpolation=),--size随机长宽比裁剪，每条边的预期输出大小为InputImgSize=(128,128)；scale为裁剪图片的大小为原来图像的x倍，本次设置为1倍；interpolation插值采用双线性插值法，扩充像素。添加插值的参数是因为考虑到有的图片大小小于我们设定的规格，需要进行扩充。

（4）transforms.ToTensor(),把一个取值范围是[0,255]的Image 转换成 Tensor，shape为(H,W,C)的numpy.ndarray，转换成形状为[C,H,W]，取值范围是[0,1.0]的Tensor。由于训练需要的通道次序与默认的通道次序不一致，因此这一变化是必须的。

（5）transforms.Normalize(mean=[0.46], std=[0.10])逐个通道对图像进行标准化（均值变为0，标准差变为1），执行这个标准化可以加快模型的收敛。output = (input - mean) / std

数据集平均值和方差需要我们另外计算。

**B．**LabelImg变化简单一些：

A.transforms.RandomAffine(),

B.transforms.RandomHorizontalFlip(),

C.transforms.RandomResizedCrop(),

D.transforms.ToTensor(),

**C．**验证集不参与到训练当中，只需要做一些基础的变化就可以，因此验证集的compose为

对于Img：

transforms.Resize(), transforms.ToTensor(), transforms.Normalize(),

对于label：

transforms.Resize(),

transforms.ToTensor(),

### 3.1.2数据加载

构建dataset子类PipeDataset传入训练图片数据

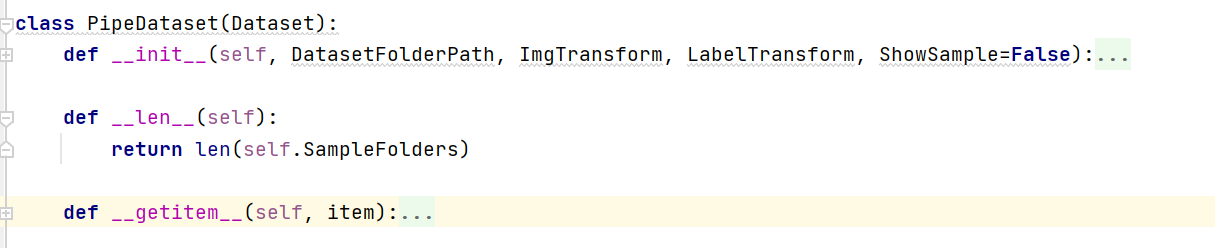


图3-2PipeDataset

代码解释：

def\_\_init\_\_(self,DatasetFolderPath,ImgTransform,LabelTransform,ShowSample=False)--在初始化中导入数据集地址，图片地址，标签地址，是否展示图片与标签。

def \_\_len\_\_(self)--获得数据集的图片数

def \_\_getitem\_\_(self, item)

第一步先获取图片的绝对路径

第二步利用 Image.open ()对图片进行读取，同时完成对应图片格式转换

第三步对图像进行transform，为了实现原图像与标签的变化一致，需要设置随机数种子。使用seed = np.random.randint()实现。

在对代码进行检查的时候发现标记与原图的变化不一致，具体情况如下图1-6：

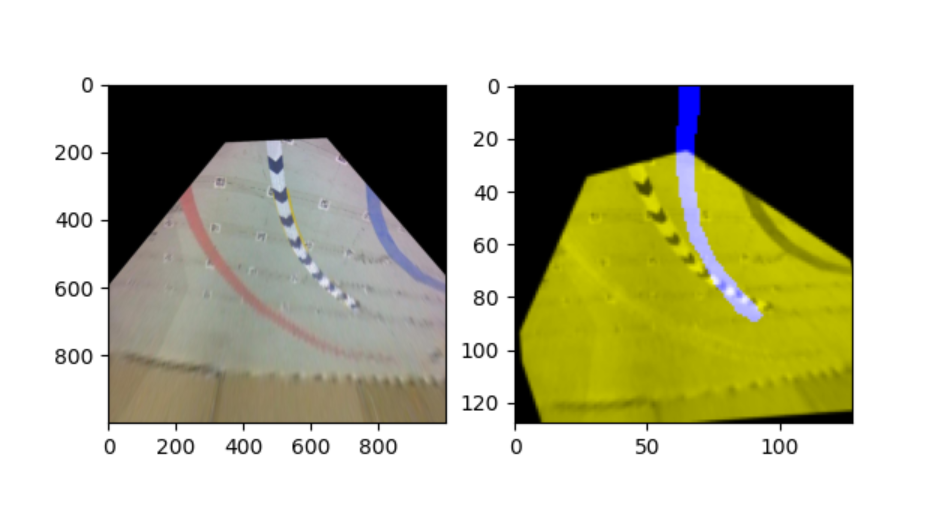


图3-3随机变换原方案

经过检查源码，发现是seed = np.random.randint()这部分代码由于版本更新，已经不再起到控制transform随机数的作用。

将源码修改为：

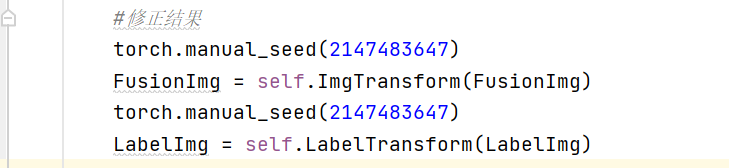


图3-4修正后随机数控制模块

修正后transform效果如图1-8：

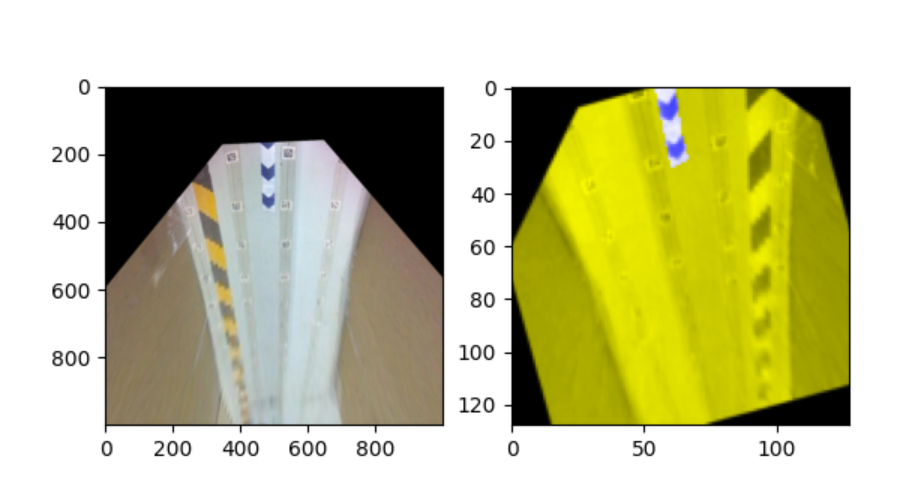


图3-5修正后transform图片

### 3.1.3封装dataset

为了便于调用数据集，定义PipeDatasetLoader()函数，函数实现以下功能：

（1）获取文件夹地址，生成训练集与测试集的原图像与标签PipeDataset（）类对应的对象，分别为TrainDataset和ValDataset

（2）将两个对象传入DataLoader（）完成整体的封装。

# 四.搭建模型

## 4.1.相关基础

搭建网络建立在继承nn.Module类的基础之上。

### 4.1.1.解读层作用

（1）nn.Conv2d()

**通过卷积层中卷积核运算，提取卷积核希望提取的特征。**

参数：

torch.nn.Conv2d(in\_channels, out\_channels, kernel\_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding\_mode='zeros')

kernel\_size：卷积核大小 stride：步长

padding：控制两侧填充 dilation：控制卷积核点与输入之间相乘的距离值，下图为dialation=1的情况。

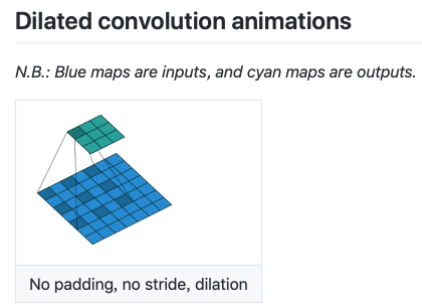


图4-1 dialation解释

Groups：控制输入与输出之间卷积层个数。 padding\_mode：控制填充的模式。

（2）nn.BatchNorm2d()

进行批量归一化，减小计算量加速训练。

torch.nn.BatchNorm2d(num\_features, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

参数：

num\_features：输入的通道数 eps：用于影响数值稳定性

momentum：用于调整var与mean的更新 affine：控制是否学习仿射

track\_running\_stats:控制追踪var与mean的数值

（3）nn.ReLU()

作为激活函数，ReLU()使得一部分神经元输出为0，减轻过拟合，减少计算量。

ReLU(x)=max（0，x）

D. nn.MaxPool2d

减小图像大小、加速计算、使其检测出的特征更加健壮。

torch.nn.MaxPool2d（kernel\_size，stride = None，padding = 0，dilation = 1，return\_indices = False，ceil\_mode = False ）

参数：

return\_indices：如果设置为真，将会返回最大输出以及其索引

ceil\_mode：为True时，将使用ceil而不是floor来计算输出形状

（4）nn.ConvTranspose2d()

反卷积操作

torch.nn.ConvTranspose2d（in\_channels，out\_channels，kernel\_size，stride = 1，padding = 0，output\_padding = 0，groups = 1，bias = True，dilation = 1，padding\_mode ='zeros' ）

参数：

Padding：输入的每一条边补充0的层数，高宽都增加2\*padding

output\_padding：输出边补充0的层数，高宽都增加padding

## 4.2.封装网络模块

为了简化代码，将在网络中重复出现的结构进行封装。

设置\_block结构。

调用nn.Sequential()网络层按照次序排列：

nn.Conv2d()

nn.BatchNorm2d()

nn.ReLU()

nn.Conv2d()

nn.BatchNorm2d()

nn.ReLU()

\_block层可以视作一个特征抽取过程，即将原始不定长序列转换成一组固定长度的特征向量。

## 4.3.组网

**def** \_\_init\_\_（）

self.encoder1 = UNet.\_block()

self.pool1 = nn.MaxPool2d()

self.encoder2 = UNet.\_block()

self.pool2 = nn.MaxPool2d()

self.encoder3 = UNet.\_block()

self.pool3 = nn.MaxPool2d()

self.encoder4 = UNet.\_block()

self.pool4 = nn.MaxPool2d()

self.bottleneck = UNet.\_block()

self.upconv4 = nn.ConvTranspose2d()

self.decoder4 = UNet.\_block()

self.upconv3 = nn.ConvTranspose2d()

self.decoder3 = UNet.\_block()

self.upconv2 = nn.ConvTranspose2d()

self.decoder2 = UNet.\_block()

self.upconv1 = nn.ConvTranspose2d()

self.decoder1 = UNet.\_block()

self.conv = nn.Conv2d()

**def** forward(self, x):

enc1 = self.encoder1(x)

enc2 = self.encoder2(self.pool1(enc1))

enc3 = self.encoder3(self.pool2(enc2))

enc4 = self.encoder4(self.pool3(enc3))

bottleneck = self.bottleneck(self.pool4(enc4))

dec4 = self.upconv4(bottleneck)

dec4 = torch.cat((dec4, enc4), dim=1)

dec4 = self.decoder4(dec4)

dec3 = self.upconv3(dec4)

dec3 = torch.cat((dec3, enc3), dim=1)

dec3 = self.decoder3(dec3)

dec2 = self.upconv2(dec3)

dec2 = torch.cat((dec2, enc2), dim=1)

dec2 = self.decoder2(dec2)

dec1 = self.upconv1(dec2)

dec1 = torch.cat((dec1, enc1), dim=1)

dec1 = self.decoder1(dec1) # 2\*32\*256\*256

if self.WithActivateLast:

# return torch.sigmoid(self.conv(dec1)) # BS\*1\*256\*256

return self.ActivateFunLast(self.conv(dec1))

else:

return self.conv(dec1) # BS\*1\*256\*256

**def** \_\_init\_\_（）部分完成网络的定义；**def** forward(self, x)部分完成网络自上而下的搭建。

【执行\_03Unet.py】

在执行\_03Unet.py发现程序报错：TypeError: 'NoneType' object is not callable。原因在于定义Unet未设置激活层，添加激活层代码如下：

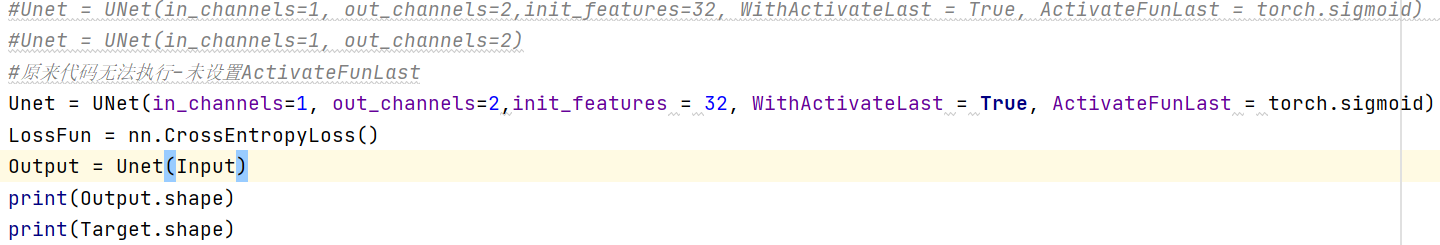


图4-2修改代码

完成修改后程序正常运行，完成随机样本的测试。

## 4.4.设计loss函数

为了检测识别得出的车道线与标签的车道线误差大小，引入dice系数作为loss函数。

dice loss 来自 dice coefficient，是一种用于评估两个样本的相似性的度量函数，取值范围在0到1之间，取值越大表示越相似。dice coefficient定义如下:

Dice=

其中是X和Y之间的交集，和分表表示X和Y的元素的个数，分子乘2为了保证分母重复计算后取值范围在[0,1]之间。

因此dice loss可以写为:

对于二分类问题，一般预测值分为以下几种:

* TP: true positive，真阳性，预测是阳性，预测对了，实际也是正例。
* TN: true negative，真阴性，预测是阴性，预测对了，实际也是负例。
* FP: false positive，假阳性，预测是阳性，预测错了，实际是负例。
* FN: false negative，假阴性，预测是阴性，预测错了，实际是正例。

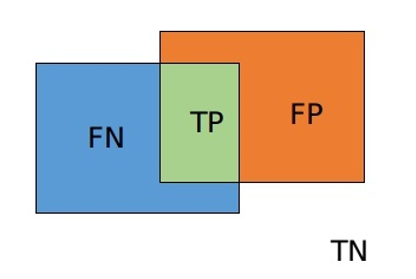


图4-3预测值

dice coefficient可以写成如下形式:

dice=

# 五.配置模型准备训练

## 5.1. \_03Training.py代码解读

（1）配置基础参数

（1）导入数据集

PipeDatasetLoader(FolderPath, BatchSize)

（2）生成网络

Unet=UNet(in\_channels=3, out\_channels=1, init\_features=4, WithActivateLast=**True**, ActivateFunLast = torch.sigmoid).to(**'cpu'**)

网络最后一层加入sigmoid函数作为激活层。添加sigmoid函数作为激活层的原因在于后续的loss计算的输入对应格式为概率分布形式。

1. 选用loss函数：

Criterion=nn.BCELoss()

BCELoss()的计算公式：

Y为实际target，X为模型预测

是二分类任务时的交叉熵计算函数。其分类限定为二分类，y 必须是{0,1}。此函数的input 应该为概率分布的形式。因此（2）生成网络的时候，需要在最后一层补充sigmod（）激活层。

1. 配置优化器

Optimizer = torch.optim.Adam(Unet.parameters(), lr=Lr)

选用 Adam优化方法。Adam 是一种自适应学习率的优化方法，利用梯度的一阶矩估计和二阶矩估计动态地调整学习率。

（5）设置训练记录

logging.basicConfig（）

在txt文件记录训练的epoch，loss等数值。

（6）设置学习率torch.optim.lr\_scheduler.StepLR(Optimizer,step\_size=LrDecayPerEpoch, gamma=LrDecay)

等间隔调整学习率，调整倍数为 gamma 倍，调整间隔为 step\_size。间隔单位是 step。

可以考虑换用**lr\_scheduler.ReduceLROnPlateau（）**在某个指标保持不变的时候更换学习率。

（7）循环训练

设置循环伦次：700

进入训练模式：Unet.train()

重置loss，在训练当中传入图片，计算预测输出与标注的loss：

TrainLoss = 0

InputImg = InputImg.float().to('cpu')

Label = Label.float().to('cpu')

Weight = Label \* (WeightCoefficient-1) + 1

Criterion.weight = Weight #loss下的参数

Optimizer.zero\_grad() #设置梯度为none

with torch.set\_grad\_enabled(True): #是否有梯度参数

OutputImg = Unet(InputImg)

BatchLoss = Criterion(OutputImg, Label)

BatchLoss.backward() #向前传递

Optimizer.step() #参数优化

TrainLoss += BatchLoss.item() #累加loss数值

AveTrainLoss = TrainLoss / TrainDataset.\_\_len\_\_() \* BatchSize

每50轮次进行一次测试

进入测试模式：Unet.eval()

传入测试集原图，标签图片，计算loss，

ValLoss = 0   
for Iter, (InputImg, Label, SampleName) in enumerate(ValDataLoader):  
 InputImg = InputImg.float().to('cpu')  
 Label = Label.float().to('cpu') #传入图片标签  
 Weight = Label \* (WeightCoefficient - 1) + 1  
 Criterion.weight = Weight  
 with torch.set\_grad\_enabled(False): # 等同于torch.no\_grad()  
 OutputImg = Unet(InputImg)  
 BatchLoss = Criterion(OutputImg, Label   
 ValLoss += BatchLoss.item()  
AveValLoss = ValLoss / ValDataset.\_\_len\_\_() #获得平均的loss

调用torch.save()保存模型

使用LrScheduler.step()更新参数

当训练次数达一定次数进入测试模式

# 六.模型训练与检测

## 6.1.训练网络与调参

（1）在\_03Training路径下运行\_01TrainMain.py实现神经网络训练；

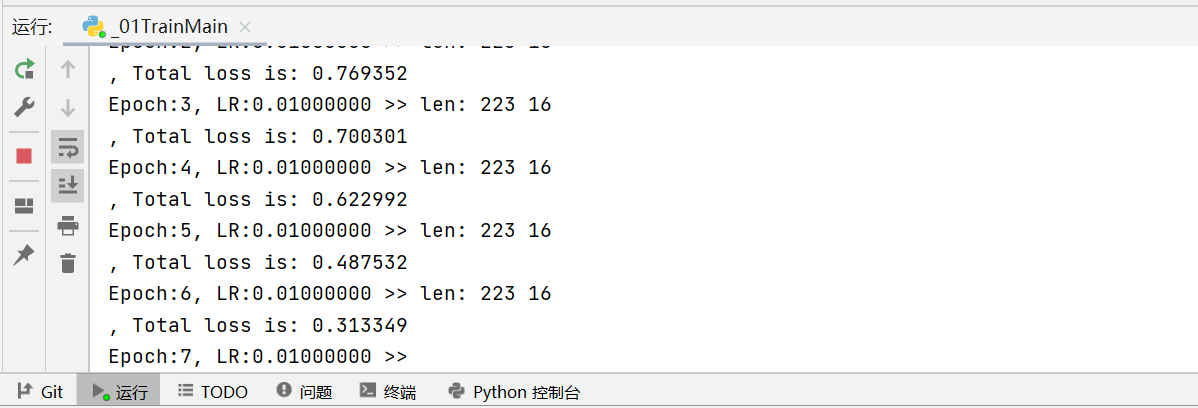


图6-1训练过程展示

（2）训练过程中对超参数进行调整

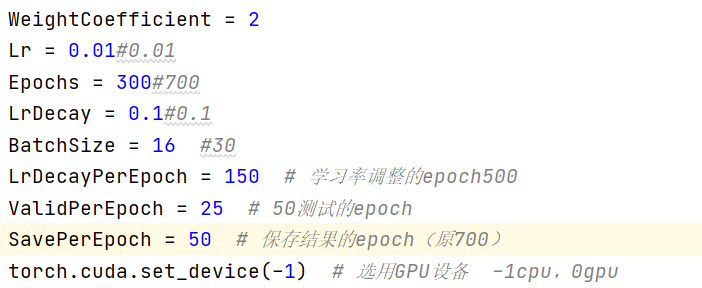


图6-2超参数修改

观察到训练到120轮次左右时，训练集loss震荡，推测是学习率偏大，调整LrDecayPerEpoch = 150，提早进行学习率的调整；

为了频繁地观察验证集的loss，修改ValidPerEpoch = 25；

为了检验训练过程中获得模型在泛化能力等方面的性能，修改为每50轮次保存一个模型用于检测：SavePerEpoch = 50

（3）查看loss变化曲线

运行\_10ShowTrainingProcess.py获得训练集与测试集loss曲线变化

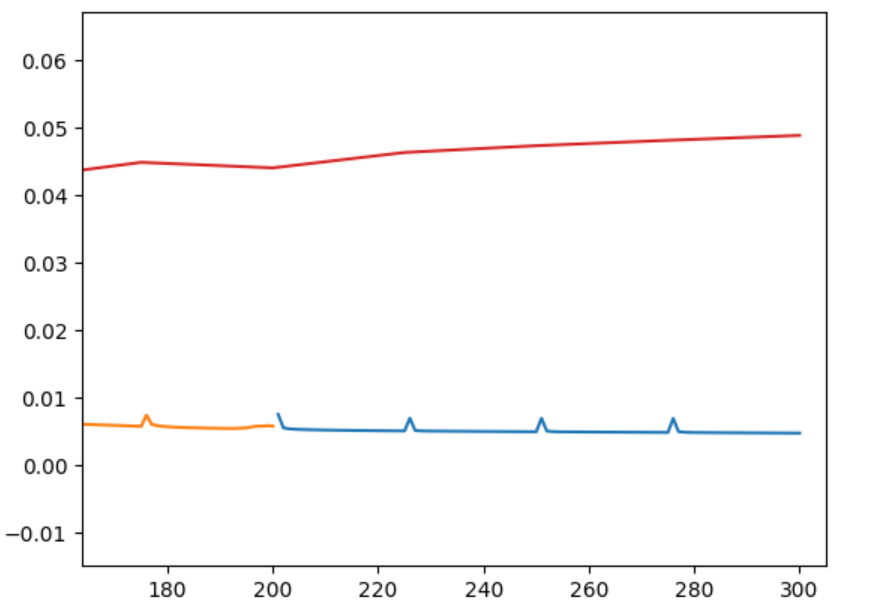
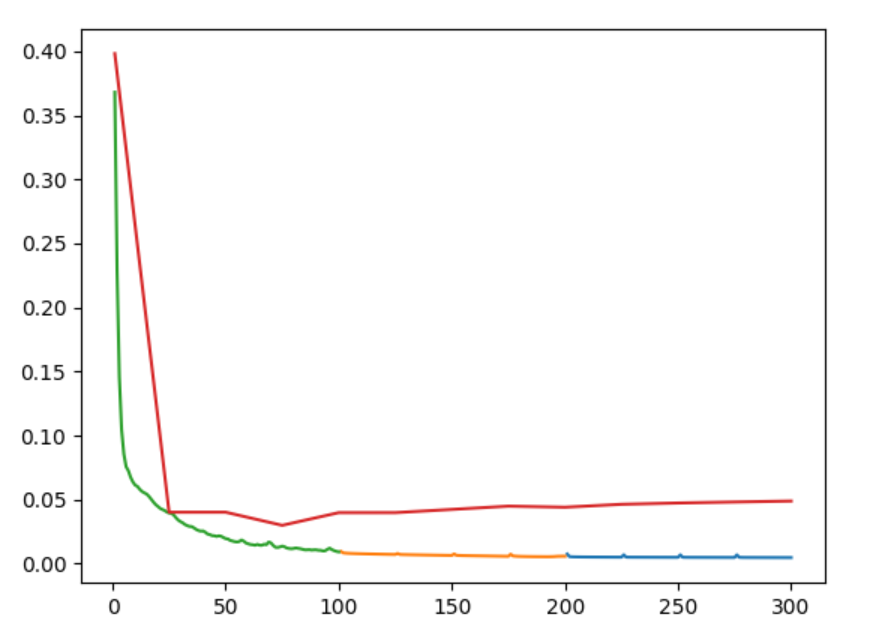


图6-3 Loss曲线

## 6.2．模型效果检测

训练结束后，在\_03Training\Output下得到x.pt,此为神经网络模型，可用于车道线检测。由于我们在\_03Training部分保存下多个模型，我们可以依据对多个模型进行测试，依据结果调整超参数，重新开展新一轮训练。

### 6.2.1.查看网络在具体图片的效果

为了更加直观的观测到训练后神经网络对于车道线的分割效果，我们对\_20TestMain.py的代码进行了一定的修改。具体内容如图6-3：



图6-4\_20TestMain.py部分修改

运行代码获得输出的图片，图片效果如图6-4

从左往右依次为Label[0]，ori\_result，ResultImg，OutputImg

Label[0]为输入网络的标记图片

ori\_result为输入图片的灰度化后原图

ResultImg为神经网络标注结果与原图合成出来的图片

OutputImg为网络的直接输出

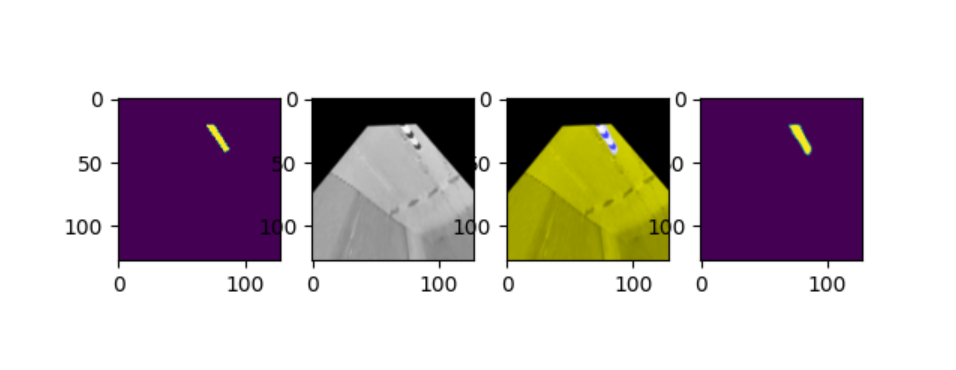
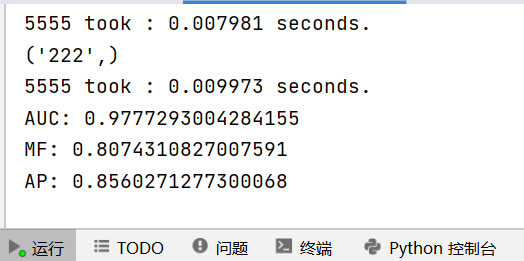


图6-5 网络效果

### 6.2.2.查看网络整体效果

运行\_20TestMain.py获得如下参数与曲线。



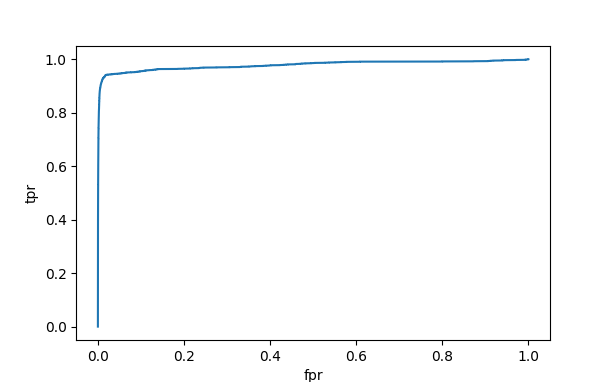
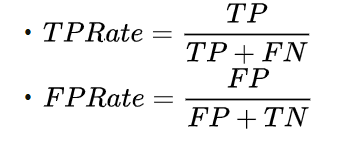


图6-6参数， ROC曲线

1. **AUC曲线**

**原理:**

纵轴的Tpr：指True Positive Rate（真阳率）、横轴的fpr指False Positive（伪阳率）。



AUC即以tpr为纵轴，fpr为横轴的ROC曲线下的面积，可以反映模型的泛化能力。

**算法实现：**

阅读\_21CalEvaluationIndicator.py代码

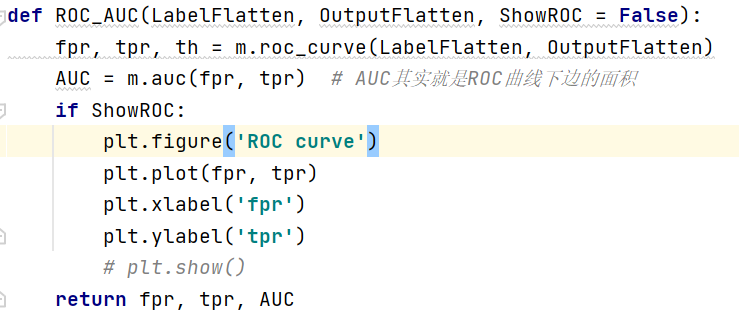


图6-7\_21CalEvaluationIndicator.py代码

由代码可知实现方案为调用sklearn.metrics.roc\_curve()绘制ROC函数，再调用sklearn.metrics.auc()计算求出AUC。

**(2)pr曲线、 MF(Maximum F-measure)、AP(average\_precision)**

**原理：**

PR曲线实则是以precision（精准率）和recall（召回率）这两个为变量而做出的曲线，其中recall为横坐标，precision为纵坐标。设定一系列阈值，计算每个阈值对应的recall和precision，即可计算出PR曲线各个点。

Precision与recall计算方式如下：

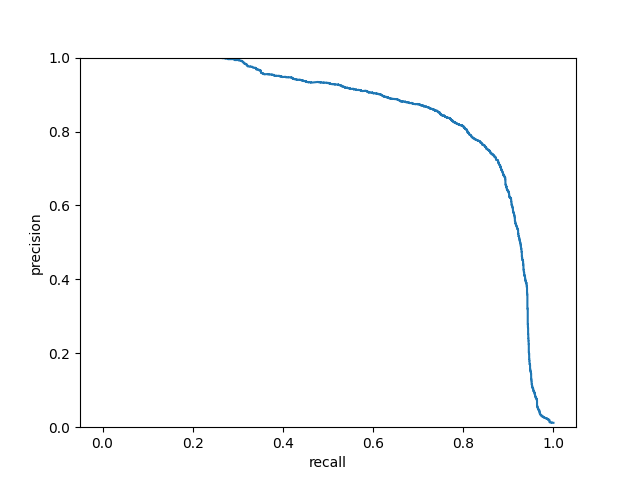


图6-8Precision-recall曲线

AP为PR曲线下边的面积，反应模型分割的平均精度，用于反应模型的识别准确率。

F-measure：精确率和召回率的调和平均数。由于分别用precision和recall两个指标不直观，将其他们合并为一个，这就是F-measure。F-measure计算式如下：

Maximum F-measure：输出最大的F-measure。

**算法实现：**

调用 m.precision\_recall\_curve()绘制Pr曲线

调用m.average\_precision\_score()计算pr曲线下面积获得AP

通过F1ScoreS = 2 \* (precision \* recall) / ((precision + recall) + sys.float\_info.min)计算所有的F-measure，再获取最大数值MF = F1ScoreS[np.argmax(F1ScoreS)]。

### 6.2.3.运行时间

运行\_21CalEvaluationIndicator.py获得如下结果

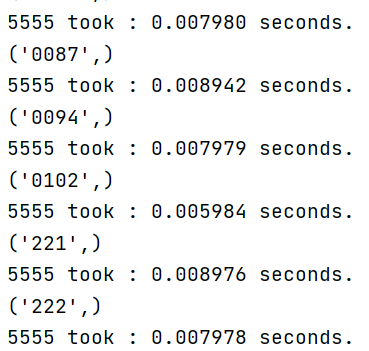


图6-9运行模型耗时

处理单张图片的时间在0.01s之内，处理速度较快。