基于深度学习的车道线检测

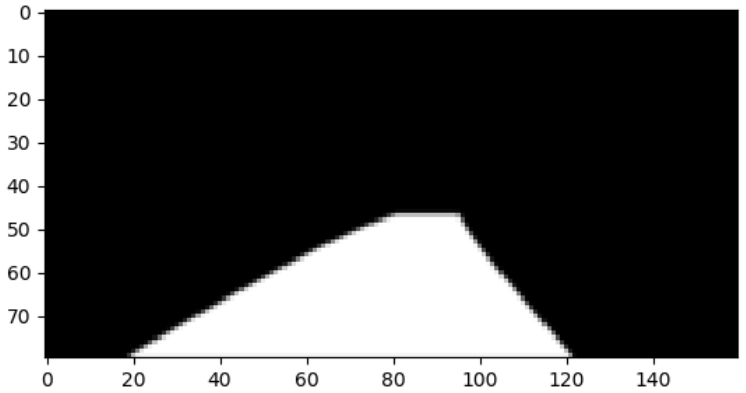
业务背景分析

- 车道检测属于图像分割的典型应用之一，使用计算机视觉深度学习模型在像素级来划分车道，这对于无人驾驶等领域有重要的作用
- 本案例基于深度学习算法进行车道检测，通过将视频信号进行分帧后交由模型进行检测车道位置信息，并通过图像合成将其检测结合与原视频帧合并可视化输出
- 本案例程序使用sklearn、pickle、cv2、PIL、IPython等库和Keras框架。

数据集说明

- 从12个视频（包括一天不同时间、不同天气、不同交通状况和弯曲道路）中选取21054个图像
- 17.4%是夜晚清晰场景，16.4%是早上雨天场景，66.2%是下午阴天场景
- 26.5%是直线或者近乎直线道路，30.2%是混合或者略弯的道路，43.3%是相当弯曲的道路
- 道路还包括不同的区域，比如在修路段和十字路口
- 滤去模糊和遮挡的图像，最终从中选取了14235个图像
- 在从10个中选取1个（视频相邻帧过于近似），1420个图像
- 加上一些其它处理，最终有1978个实际图像（ coeffs_labels.p -coeffs_train.p ）
- 旋转处理后，扩展为6382个图像 ,水平翻转后，最终12764个图像(full_CNN_labels.p-full_CNN_train.p)

数据集举例



- 原始图像缩小后尺寸
- **80*160*3**很小，确保训练效率
- **full_CNN_train.p**

- 标注结果，用于训练
- **full_CNN_labels.p**

Python数据存储：pickle模块的使用

- 在机器学习中，我们常常需要把训练好的模型存储起来，这样在进行决策时直接将模型读出，而不需要重新训练模型，这样就大大节约了时间。**Python**提供的**pickle**模块就很好地解决了这个问题，它可以序列化对象并保存到磁盘中，并在需要的时候读取出来，任何对象都可以执行序列化操作
- （1）**pickle.dump(obj, file, [,protocol])**

  函数的功能：将**obj**对象序列化存入已经打开的**file**中。
- （2）**pickle.load(file)**

  函数的功能：将**file**中的对象序列化读出。

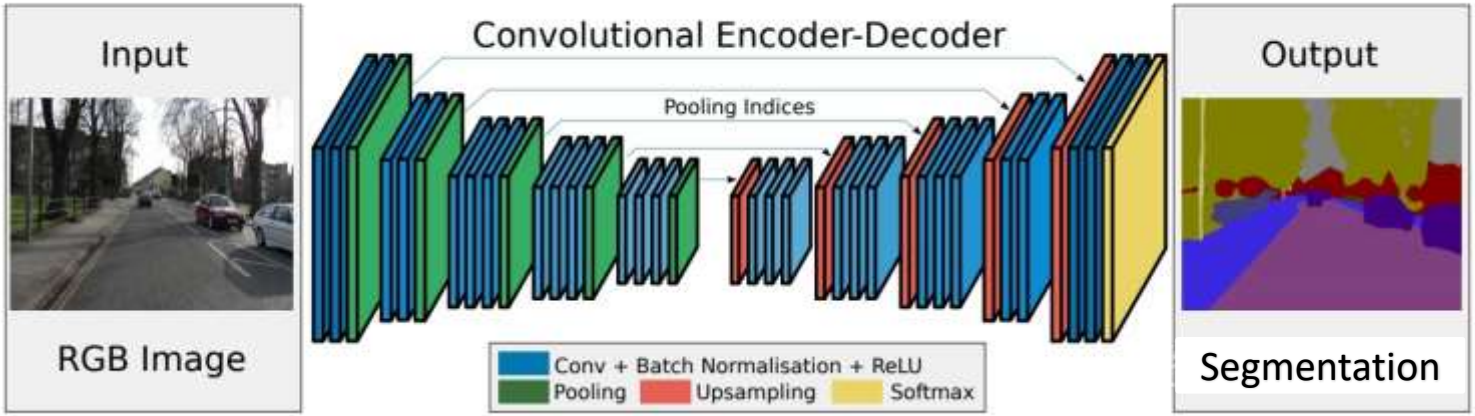
数据预处理

- 加载样本数据，并进行必要的归一化、样本顺序随机化等预处理后，把数据分割成训练集和测试集

```
# 加载数据
train_images = pickle.load(open("full_CNN_train.p", "rb" ))
# 加载标签
labels = pickle.load(open("full_CNN_labels.p", "rb" ))
# 对数据进行预处理
train_images = np.array(train_images)
labels = np.array(labels)
# 对标签进行归一化处理
labels = labels / 255
# 样本顺序随机化
train_images, labels = shuffle(train_images, labels)
# 划分训练集和测试集
X_train, X_val, y_train, y_val = train_test_split(train_images, labels, test_size=0.1)
print("loaded train samples:", len(train_images))
```

网络模型选择

- 车道检测采用了**SegNet**网络模型，**SegNet**网络是一种很有趣的图像分割技术，是一种**encoding-decoding**的结构，在使用时可直接调用标准的模型结构。

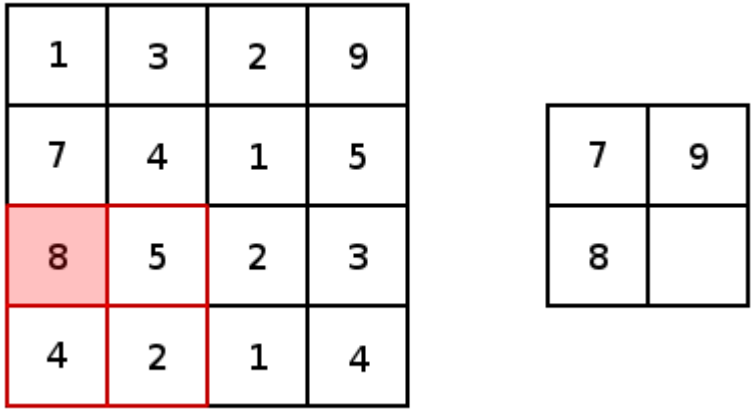


网络模型选择

- 图像分割的实现由一个卷积神经网络构成，该网络主要有两部分组成：encoder与decoder。encoder是一个沿用VGG16的网络模型，主要对物体信息进行解析。decoder将解析后的信息对应成最终的图像形式，即每个像素都用对应其物体信息的颜色（或者是label）来表示。
- encoder本身其实就是一连串的卷积网络。该网络主要由卷基层，池化层和BatchNormalization层组成。卷基层负责获取图像局域特征，池化层对图像进行下采样并且将尺度不变特征传送到下一层，而BN主要对训练图像的分布归一化，加速学习。
- Decoder对缩小后的特征图像进行上采样，然后对上采样后的图像进行卷积处理，目的是完善物体的几何形状，弥补Encoder当中池化层将物体缩小造成的细节损失。

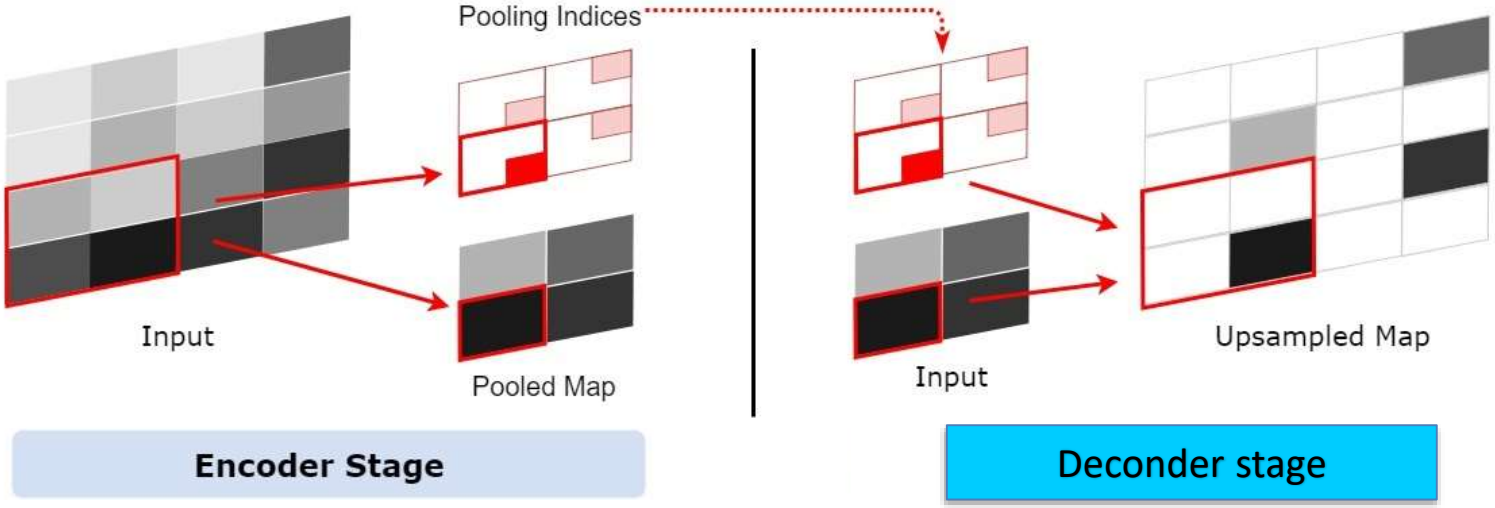
网络模型选择

- **2x2 Max Pooling**



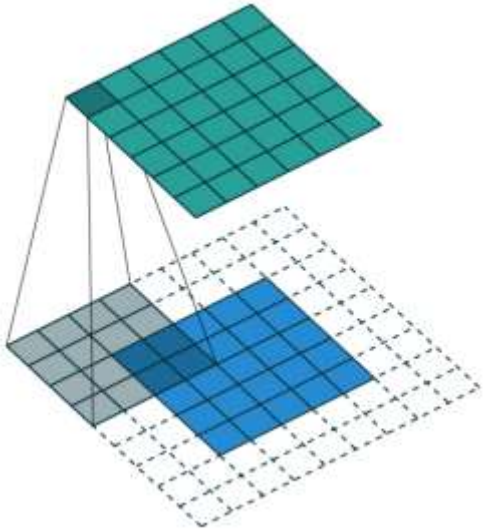
网络模型选择

➢ **Pooling Indices**

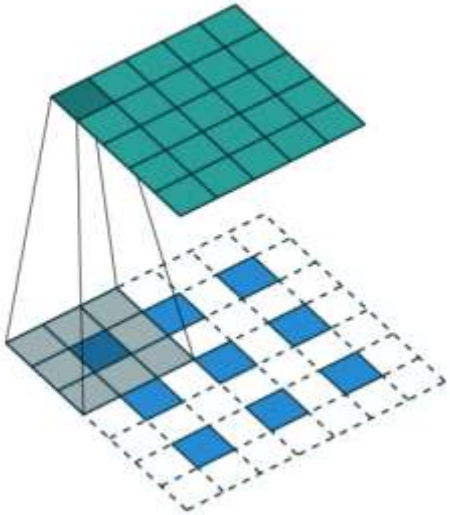


网络模型选择

- **Encoder阶段的卷积命名为“卷积”，Decoder阶段的卷积命名为“反卷积“（或”转置卷积“）**



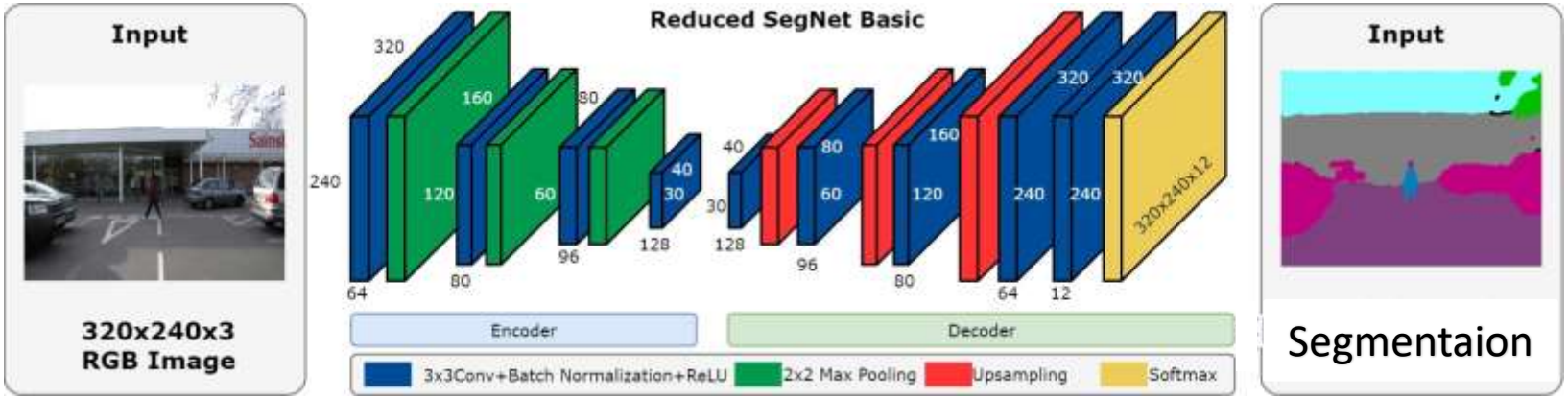
卷积操作：蓝色是输入 青色是输出



反卷积操作：蓝色是输入 青色是输出

网络模型选择

➢ **SegNet Basic**



网络模型选择

- **SegNet Basic**

网络模型选择（1）

- 网络输入层为80×160 ×3（对应R、G、B值）的车辆道路行驶图像，标签为80 ×16 ×1，只用G通道重新绘制车道，网络结构的代码如下

```
def create_model(input_shape, pool_size):
    model = Sequential()
    # 对输入层进行归一化处理
    model.add(BatchNormalization(input_shape=input_shape))
    # 卷积层 1，名为 Conv1
    model.add(Conv2D(8, (3, 3), padding='valid', strides=(1,1), activation = 'relu',
name = 'Conv1'))
    model.add(Conv2D(16, (3, 3), padding='valid', strides=(1,1), activation = 'relu',
name = 'Conv2'))
    # 最大池化层
    model.add(MaxPooling2D(pool_size=pool_size))
    model.add(Conv2D(16, (3, 3), padding='valid', strides=(1,1), activation = 'relu',
name = 'Conv3'))
    model.add(Dropout(0.2))
    model.add(Conv2D(32, (3, 3), padding='valid', strides=(1,1), activation = 'relu',
name = 'Conv4'))
    model.add(Dropout(0.2))
    model.add(Conv2D(32, (3, 3), padding='valid', strides=(1,1), activation = 'relu',
name = 'Conv5'))
    model.add(Dropout(0.2))
    model.add(MaxPooling2D(pool_size=pool_size))
    model.add(Conv2D(64, (3, 3), padding='valid', strides=(1,1), activation = 'relu',
name = 'Conv6'))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (3, 3), padding='valid', strides=(1,1), activation = 'relu',
name = 'Conv7'))
    model.add(Dropout(0.2))
    model.add(MaxPooling2D(pool_size=pool_size))
    # 上采样层 1
    model.add(UpSampling2D(size=pool_size))
```

网络模型选择（2）

```
        # 反卷积层 1
        model.add(Conv2DTranspose(64, (3, 3), padding='valid', strides=(1,1), activation
= 'relu', name = 'Deconv1'))
        model.add(Dropout(0.2))
        model.add(Conv2DTranspose(64, (3, 3), padding='valid', strides=(1,1), activation
= 'relu', name = 'Deconv2'))
        model.add(Dropout(0.2))
        # 上采样层 2
        model.add(UpSampling2D(size=pool_size))
        model.add(Conv2DTranspose(32, (3, 3), padding='valid', strides=(1,1), activation
= 'relu', name = 'Deconv3'))
        model.add(Dropout(0.2))
        model.add(Conv2DTranspose(32, (3, 3), padding='valid', strides=(1,1), activation
= 'relu', name = 'Deconv4'))
        model.add(Dropout(0.2))
        model.add(Conv2DTranspose(16, (3, 3), padding='valid', strides=(1,1), activation
= 'relu', name = 'Deconv5'))
        model.add(Dropout(0.2))
        model.add(UpSampling2D(size=pool_size))
        model.add(Conv2DTranspose(16, (3, 3), padding='valid', strides=(1,1), activation
= 'relu', name = 'Deconv6'))
        # 输出层
        model.add(Conv2DTranspose(1, (3, 3), padding='valid', strides=(1,1), activation =
'relu', name = 'Final'))
        return model
```

设计车道检测模型

- 定义网络的一些超参数，通过训练构建并生成车道检测模型，然后将生成的模型存储，代码如下

```
batch_size = 128
# 训练回合数
epochs = 10
# 池化核大小
pool_size = (2, 2)
# 输入大小
input_shape = X_train.shape[1:]
# 构建模型
model = create_model(input_shape, pool_size)
# 构建数据生成器，实现数据增强
datagen = ImageDataGenerator(channel_shift_range=0.2)
datagen.fit(X_train)
model.compile(optimizer='Adam', loss='mean_squared_error')
# 可视化模型概要
model.summary()
```

设计车道检测模型

- 代码运行后结果如下

Layer (type)	Output Shape	Param #
batch_normalization_1 (Batch	(None, 80, 160, 3)	12
Conv1 (Conv2D)	(None, 78, 158, 8)	224
Conv2 (Conv2D)	(None, 76, 156, 16)	1168
max_pooling2d_1 (MaxPooling2	(None, 38, 78, 16)	0
Conv3 (Conv2D)	(None, 36, 76, 16)	2320
dropout_1 (Dropout)	(None, 36, 76, 16)	0
Conv4 (Conv2D)	(None, 34, 74, 32)	4640
dropout_2 (Dropout)	(None, 34, 74, 32)	0
Conv5 (Conv2D)	(None, 32, 72, 32)	9248
dropout_3 (Dropout)	(None, 32, 72, 32)	0
max_pooling2d_2 (MaxPooling2	(None, 16, 36, 32)	0
Conv6 (Conv2D)	(None, 14, 34, 64)	18496
dropout_4 (Dropout)	(None, 14, 34, 64)	0
Conv7 (Conv2D)	(None, 12, 32, 64)	36928
dropout_5 (Dropout)	(None, 12, 32, 64)	0
max_pooling2d_3 (MaxPooling2	(None, 6, 16, 64)	0
up_sampling2d_1 (UpSampling2	(None, 12, 32, 64)	0
Deconv1 (Conv2DTranspose)	(None, 14, 34, 64)	36928
dropout_6 (Dropout)	(None, 14, 34, 64)	0
Deconv2 (Conv2DTranspose)	(None, 16, 36, 64)	36928
dropout_7 (Dropout)	(None, 16, 36, 64)	0
up_sampling2d_2 (UpSampling2	(None, 32, 72, 64)	0
Deconv3 (Conv2DTranspose)	(None, 34, 74, 32)	18464
dropout_8 (Dropout)	(None, 34, 74, 32)	0
Deconv4 (Conv2DTranspose)	(None, 36, 76, 32)	9248
dropout_9 (Dropout)	(None, 36, 76, 32)	0
Deconv5 (Conv2DTranspose)	(None, 38, 78, 16)	4624
dropout_10 (Dropout)	(None, 38, 78, 16)	0
up_sampling2d_3 (UpSampling2	(None, 76, 156, 16)	0
Deconv6 (Conv2DTranspose)	(None, 78, 158, 16)	2320
Final (Conv2DTranspose)	(None, 80, 160, 1)	145

Total params: 181,693
Trainable params: 181,687
Non-trainable params: 6

训练模型

- 采用梯度下降法，对构建的模型进行训练，训练后的模型保存在文件full_CNN_model_tiny.h5中。为了改善车道检测模型的效果，还需要对模型的训练进行优化。可以思考下如何对车道检测模型进行优化？

```
    model.fit_generator(datagen.flow(X_train, y_train, batch_size=batch_size), steps_
per_epoch=len(X_train)/batch_size,epochs=epochs,   verbose=1,   validation_data=(X_val,
y_val))
    # 保存模型
    model.trainable = False
    model.compile(optimizer='Adam', loss='mean_squared_error')
    model.save('full_CNN_model_tiny.h5')
```

车道检测模型测试

- 定义工具道路线类、图像实现方法等类，然后加载训练好的车道检测模型。读取input目录下的demo.mp4，逐帧读取并输入模型进行检测，然后计算检测结果的均值，并将检测结果绘制输出，代码如下

```
# 道路线类
class Lanes():
    def __init__(self):
        self.recent_fit = []

        self.avg_fit = []
#图像显示方法
def arrayShow(imageArray):
    ret, jpg = cv2.imencode('.jpg', imageArray)
    return Image(jpg)
#加载检测模型
model = load_model('full_CNN_model.h5')
#读取视频文件
vs = cv2.VideoCapture("input/demo1.mp4")
frameIndex = 0
lanes = Lanes()
```

车道检测模型测试

```
#循环读取图像帧
while True:
    #读取视频帧
    (grabbed, frame_source) = vs.read()
    if not grabbed:
        break
    #将视频帧 resize 设置为模型的输入大小
    frame_show = cv2.resize(frame_source, (576, 288))
    frame = cv2.resize(frame_show, (160, 80))
    rgb_small_frame = frame[None,:,:,:]
    #模型推理检测
    prediction = model.predict(rgb_small_frame)[0] * 255
    #存储预测结果到列表
    lanes.recent_fit.append(prediction)
    #只使用最近的数据
    if len(lanes.recent_fit) > 5:
        lanes.recent_fit = lanes.recent_fit[1:]
    #计算平均预测结果
    lanes.avg_fit = np.mean(np.array([i for i in lanes.recent_fit]), axis = 0)
    #绿化 R 和 B 通道
    blanks = np.zeros_like(lanes.avg_fit).astype(np.uint8)
    lane_drawn = np.dstack((blanks, lanes.avg_fit, blanks))
    #恢复成原始视频大小
    lane_image = cv2.resize(lane_drawn, (576, 288))
    #合并原始图像和检测结果
    img = cv2.addWeighted(frame_show, 0.3, lane_image, 0.7, 0,dtype = cv2.CV_32F)
    #清空绘图空间
    clear_output(wait=True)
    #显示处理结果
    display(arrayShow(img))
    #按键盘中的 Q 键退出检测
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
#释放资源
print("[INFO] cleaning up...")
vs.release()
cv2.destroyAllWindows()
```

车道检测模型测试

- 车道检测结果如图。

车道检测模型测试

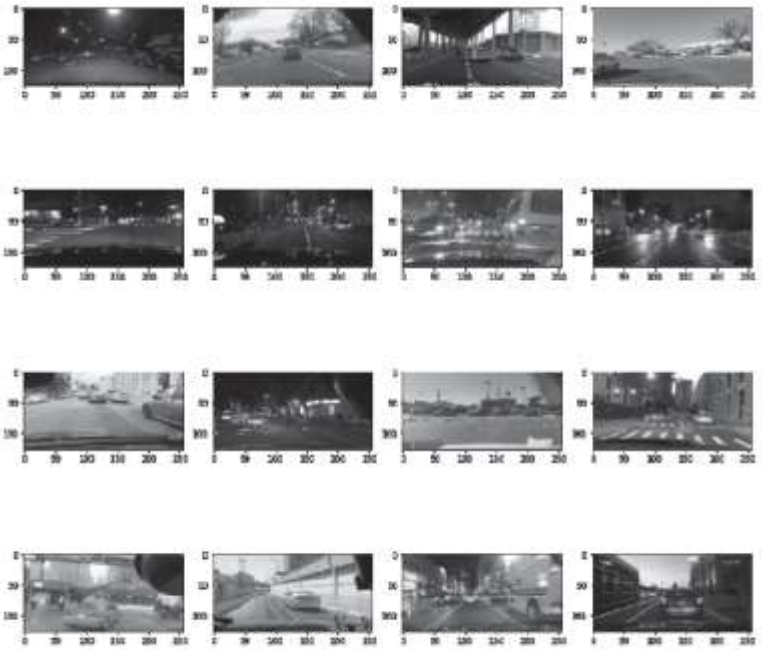
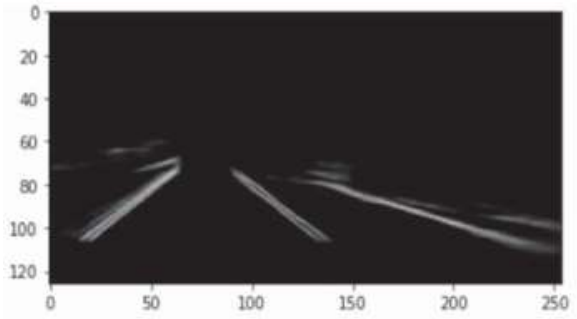


图 11.3 训练图像



U-Net车道检测（1）

```
#导入 Python 相关组件
#!/usr/bin/env python
# coding: utf-8
import cv2
import pandas
import numpy as np
import keras
import glob
import os
import json
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from keras.models import *
from keras.layers import *
from keras.optimizers import *
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, Callback
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array,
load_img
from keras import backend as keras
from skimage.draw import line
from tqdm import tqdm_notebook as tqdm
from keras_tqdm import TQDMNotebookCallback
```

```
#定义数据存放位置
LABEL_PATH = '../data/bdd100k/labels/100k/'
LABEL_VAL_PATH = LABEL_PATH + 'val'
LABEL_TRAIN_PATH = LABEL_PATH + 'train'
DATA_PATH_TRAIN = '../data/bdd100k-1/images/100k/train/'
DATA_PATH_VAL = '../data/bdd100k-1/images/100k/val/'
VAL_LOAD = 500
TRAIN_LOAD = 5000
#加载标签数据
#从 BDD100K 数据集中加载车道直线标记信息
def load_label(path, to_load):
    count = 0
    onlyfiles = glob.glob(path+"/*.json")
    formatted_data = []
    for ff in onlyfiles:
        if count > to_load:
```

U-Net车道检测（2）

```
            break
        with open(ff) as json_file:
            data = json.load(json_file)
            image_name = data["name"]
            lanes = []
            for entry in data['frames'][0]['objects']:
                cat = entry['category']
                if 'lane' not in cat:
                    continue
                if len(entry['poly2d'])>2:
                    continue
                lanes.append(entry['poly2d'])
        formatted_data.append([image_name+".jpg", lanes])
        count += 1
    print("Loaded " + str(len(formatted_data)) + " entries")
    return formatted_data
val_labels = load_label(LABEL_VAL_PATH, VAL_LOAD)
train_labels = load_label(LABEL_TRAIN_PATH, TRAIN_LOAD)
#输出结果
Loaded 501 entries
Loaded 5001 entries

#对图像进行压缩
DOWNSCALE = 2
def label_to_image(label):
    lines = label[1]
    image = np.zeros([int(720 / DOWNSCALE),int(1280 / DOWNSCALE),3])
    for cur in lines:
        y1 = int(cur[0][0] / DOWNSCALE)
        x1 = int(cur[0][1] / DOWNSCALE)
        y2 = int(cur[1][0] / DOWNSCALE)
        x2 = int(cur[1][1] / DOWNSCALE)
        rr, cc = line(x1,y1,x2,y2)
        rr = np.clip(rr, 0, int(720 / DOWNSCALE) - 2)
        cc = np.clip(cc, 0, int(1280 / DOWNSCALE) -2)
        image[rr    ,cc, :] = 1.0
        image[rr    ,cc - 1, :] = 1.0
        image[rr    ,cc + 1, :] = 1.0
        image[rr - 1 ,cc , :] = 1.0
        image[rr + 1 ,cc , :] = 1.0
        image[rr - 1 ,cc - 1, :] = 1.0
        image[rr + 1 ,cc + 1, :] = 1.0
    image = cv2.resize(image, (254,126))
    return image
#生成训练集标签
y_train = []
for label in train_labels:
    y_train.append(label_to_image(label))

#构建验证集标签
y_val = []
```

U-Net车道检测（3）

```
for label in val_labels:
    y_val.append(label_to_image(label))
#加载图片集，并加载单个图像
def load_image(path):
    img = load_img(path)
    x = img_to_array(img)
    x = cv2.resize(x, (256,128))
    x = x.reshape((1,) + x.shape)
    x = x / 255
return x

#加载图片数据集
def load_images(dir, labels):
    images = []
    for label in labels:
        image = label[0]
        path = dir + image
        images.append(load_image(path))
    return images
#训练集图片和验证集图片加载
x_train = load_images(DATA_PATH_TRAIN, train_labels)
x_val = load_images(DATA_PATH_VAL, val_labels)
#对图像进行格式转化
x_train = np.array(x_train)
x_val = np.array(x_val)
x_train = x_train.reshape(len(x_train),128,256,3)
x_val = x_val.reshape(len(x_val),128,256,3)
```

```
#保存图片中间结果
np.save('x_train.npy', x_train)
np.save('x_val.npy', x_val)
#保存标签中间结果
np.save('y_train.npy', y_train)
np.save('y_val.npy', y_val)

#加载数据（节省预处理工作）
x_train = np.load('x_train.npy')
x_val = np.load('x_val.npy')
y_train = np.load('y_train.npy')
y_val = np.load('y_val.npy')

#可视化数据
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
#定义输出图表格式，即 4 行 4 列显示图片
nrows = 4
ncols = 4
#图片索引编号
```

U-Net车道检测（4）

```
pic_index = 0
In [38]:
#构建显示图
fig = plt.gcf()
fig.set_size_inches(ncols * 4, nrows * 4)
for i in range(nrows * ncols):
    #设置子图显示
    sp = plt.subplot(nrows, ncols, i + 1)
    #读取图片文件
    y_i = y_train[i]
    x_train_i = x_train[i]
    x_train_i = cv2.resize(x_train_i, (254,126))
    x_train_i = np.array([x_train_i])
    x_train_i = x_train_i.reshape(len(x_train_i),126,254,3)
    combined = x_train_i[0] + y_i
    plt.imshow((combined * 255).astype(np.uint8))
plt.show()
#模型训练
#定义评价指标（重合度）
def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    coef = (2. * intersection + K.epsilon()) / (K.sum(y_true_f) + K.sum(y_pred_f) +
K.epsilon())
    return coef
```

```
#定义网络结构
def unet(pretrained_weights = None,input_size = (128,256,3)):
    inputs = Input(input_size)
    conv1 = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(inputs)
    conv1 = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv1)
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pool1)
    conv2 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv2)
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pool2)
    conv3 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv3)
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pool3)
    conv4 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv4)
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)
    conv5 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pool4)
    conv5 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
```

U-Net车道检测（5）

```
'he_normal')(conv5)
        drop5 = Dropout(0.5)(conv5)
        up6 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(drop5))
        merge6 = concatenate([drop4,up6], axis = 3)
        conv6 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge6)
        conv6 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv6)
        up7 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv6))
        merge7 = concatenate([conv3,up7], axis = 3)
        conv7 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge7)
        conv7 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv7)
        up8 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv7))
        merge8 = concatenate([conv2,up8], axis = 3)
        conv8 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge8)
        conv8 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv8)
        up9 = Conv2D(32, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv8))
        merge9 = concatenate([conv1,up9], axis = 3)
        conv9 = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge9)
        conv9 = Conv2D(32, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv9)
        conv9 = Conv2D(3, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv9)
        conv10 = Conv2D(3, 3, activation = 'sigmoid')(conv9)
        model = Model(input = inputs, output = conv10)
        model.compile(optimizer = Adam(lr = 5e-4), loss = 'binary_crossentropy', metrics =
[dice_coef])
        print(model.summary())
        if(pretrained_weights):
            print("Loading Weights from " + pretrained_weights)
            model.load_weights(pretrained_weights)
    return model

# 调用U-net模型
model = unet()
#保存网络结构
model_json = model.to_json()
with open("model-small.json", "w") as json_file:
    json_file.write(model_json)

# 构建一条测试记录
def predict_model():
    test_image =
load_image('../data/bdd100k-1/images/100k/test/fd5bae34-fbf76acf.jpg')
    test = np.array([test_image])
```

U-Net车道检测（6）

```
    test = test.reshape(len(test),128,256,3)
    lanes = model.predict(test, verbose=1)
    y = lanes
    x = cv2.resize(test[0], (254,126))
    x = np.array([x])
    x = x.reshape(len(x),126,254,3)
    combined = x[0] + (y*2)
    return combined[0]

#构建预测方法
class Predict(Callback):
    def on_train_begin(self, logs={}):
        self.losses = []
#每个 epoch 结束时执行一次预测
    def on_epoch_end(self, epoch, logs={}):
        lanes = predict_model()
        plt.imshow(lanes)
        print(lanes.shape)
        plt.savefig('result-' + str(epoch)+'.png')
        return
```

```
    #定义模型保存位置
    model_checkpoint = ModelCheckpoint('unet_small.hdf5', monitor='loss',verbose=1,
save_best_only=True)
    #定义模型预测方法
    predict_cb = Predict()
    import pandas.util.testing as tm

    #模型训练
    model.fit(np.array(x1),np.array(y1),validation_data=(np.array(x2),np.array(y2)) ,
batch_size=2, epochs=5, verbose=1, shuffle=True, callbacks=[model_checkpoint, predict_cb])

    #模型测试
    test_image = load_image('../data/bdd100k-1/images/100k/test/fd5bae34-d63db3d7.jpg')
    test = np.array([test_image])
    test = test.reshape(len(test),128,256,3)
    lanes = model.predict(test)
    print(lanes.shape)
    #原始图像
    plt.imshow(test[0])
    #输出的预测结果
    plt.imshow(lanes[0])
```


THANK
YOU