from google.colab import drive

drive.mount('/content/drive')

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import cv2

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import models, layers, Sequential

train\_path = '/content/drive/MyDrive/training'

img\_generator = keras.preprocessing.image.ImageDataGenerator()

seed = 10

images\_set = img\_generator.flow\_from\_directory(

    train\_path,

    shuffle=False,

    batch\_size=64,

    class\_mode='binary',

    target\_size=(256, 320)

)

num\_images = 7252 # gotten from the output of the cell above

num\_batches = num\_images // 64 + 1

# initialize an empty list to store the images

X = []

Y = []

# loop over the batches and extract the images

for i in range(num\_batches):

    batch = next(images\_set)

    batch\_images = batch[0] # this contains the images

    batch\_labels = batch[1] # this contains 0s and 1s

    for ind, lb in enumerate(batch\_labels):

        '''

        a label of 0 means the image belongs to ground truth image,

        and a label of 1 means that the image belongs to the ground truth mask

        '''

        if lb == 0:

            X.append(batch\_images[ind])

        else:

          Y.append(np.mean(batch\_images[ind], axis=2)) # Y shape is (m, 256, 320)

    if i % 10 == 0:

        print(f'Batch {i}')

# convert the lists to numpy arrays

X = np.array(X)

Y = np.array(Y)

from sklearn.utils import shuffle

X, Y = shuffle(X, Y, random\_state=100)

# Normalize and reshape the mask set (Y)

Y = (Y >= 100).astype('int').reshape(-1, 256, 320, 1)

Y.min(), Y.max()

# we get 2000 images for training and evaluation

X = np.array(X[:2000])

Y = np.array(Y[:2000])

# Split the datset into train and val sets

from sklearn.model\_selection import train\_test\_split

X\_train, X\_val, Y\_train, Y\_val = train\_test\_split(X, Y, test\_size=.1, random\_state=100)

print("Shape of X\_train:", X\_train.shape)

print("Shape of X\_val:", X\_val.shape)

print("Shape of Y\_train:", Y\_train.shape)

print("Shape of Y\_val:", Y\_val.shape)

**OUTPUT:**

Found 7252 images belonging to 4 classes.

Batch 0

Batch 10

Batch 20

Batch 30

Batch 40

Batch 50

Batch 60

Batch 70

Batch 80

Batch 90

Batch 100

Batch 110

Shape of X\_train: (1800, 256, 320, 3)

Shape of X\_val: (200, 256, 320, 3)

Shape of Y\_train: (1800, 256, 320, 1)

Shape of Y\_val: (200, 256, 320, 1)

**CODE:**

# define the model's architecture

from keras.layers import Input, Conv2D, MaxPooling2D, Dropout, UpSampling2D, Concatenate

from keras.models import Model

def unet(input\_size=(256,320,3)):

    inputs = Input(input\_size)

    rescale = keras.layers.Rescaling(1./255)(inputs)

    # Encoder

    conv1 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(rescale)

    conv1 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv1)

    pool1 = MaxPooling2D(pool\_size=(2, 2))(conv1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool1)

    conv2 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv2)

    pool2 = MaxPooling2D(pool\_size=(2, 2))(conv2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool2)

    conv3 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv3)

    pool3 = MaxPooling2D(pool\_size=(2, 2))(conv3)

    conv4 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool3)

    conv4 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv4)

    pool4 = MaxPooling2D(pool\_size=(2, 2))(conv4)

    # Decoder

    conv5 = Conv2D(1024, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(pool4)

    conv5 = Conv2D(1024, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv5)

    drop5 = Dropout(0.5)(conv5)

    up6 = Conv2D(512, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(UpSampling2D(size=(2,2))(drop5))

    merge6 = Concatenate(axis=3)([conv4, up6])

    conv6 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge6)

    conv6 = Conv2D(512, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv6)

    up7 = Conv2D(256, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(UpSampling2D(size=(2,2))(conv6))

    merge7 = Concatenate(axis=3)([conv3, up7])

    conv7 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge7)

    conv7 = Conv2D(256, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv7)

    up8 = Conv2D(128, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(UpSampling2D(size=(2,2))(conv7))

    merge8 = Concatenate(axis=3)([conv2, up8])

    conv8 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge8)

    conv8 = Conv2D(128, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv8)

    up9 = Conv2D(64, 2, activation='relu', padding='same', kernel\_initializer='he\_normal')(UpSampling2D(size=(2,2))(conv8))

    merge9 = Concatenate(axis=3)([conv1, up9])

    conv9 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(merge9)

    conv9 = Conv2D(64, 3, activation='relu', padding='same', kernel\_initializer='he\_normal')(conv9)

    outputs = Conv2D(1, 1, activation='sigmoid')(conv9)

    model = Model(inputs=inputs, outputs=outputs)

    return model

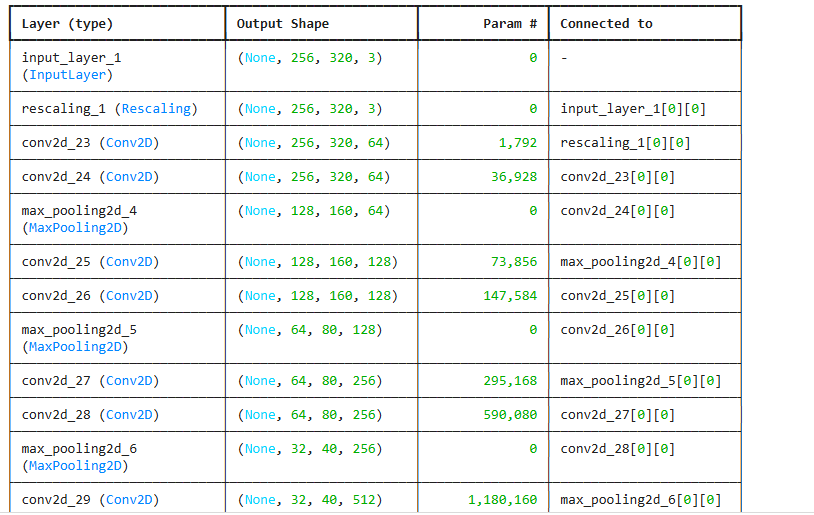
model = unet()

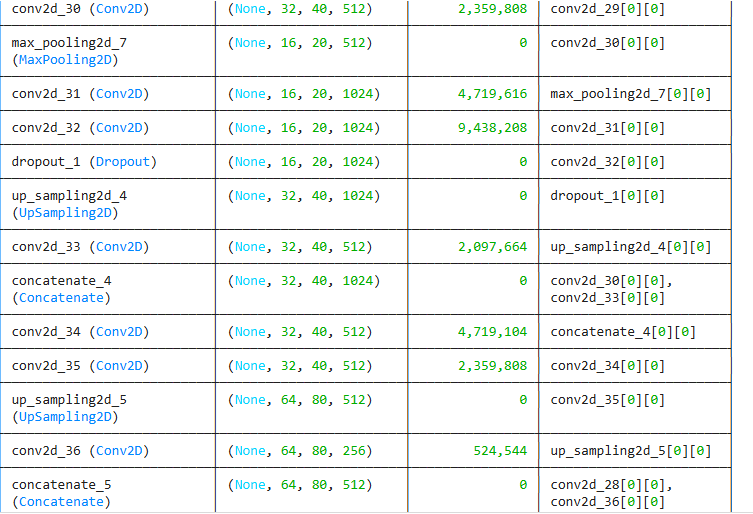
# Compile the model

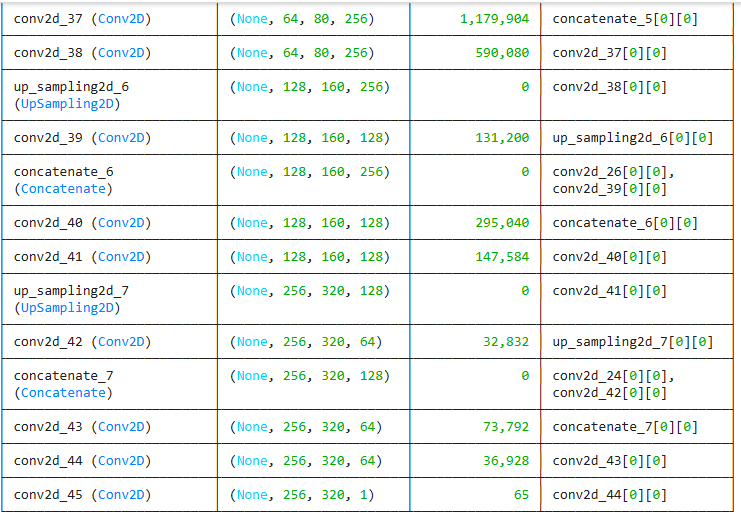
model.compile(optimizer='adam', loss=keras.losses.BinaryFocalCrossentropy(), metrics=['accuracy'])

model.summary()

**OUTPUT:**

****

****

****

Total params: 31,031,745 (118.38 MB)

Trainable params: 31,031,745 (118.38 MB)

Non-trainable params: 0 (0.00 B)

**CODE:**

import tensorflow as tf

from tensorflow import keras

# train the model

epochs = 5

batch\_size = 8

callbacks = [

    keras.callbacks.ModelCheckpoint("save\_at\_{epoch}.keras"),

]

model.fit(

    X\_train,

    Y\_train,

    epochs=epochs,

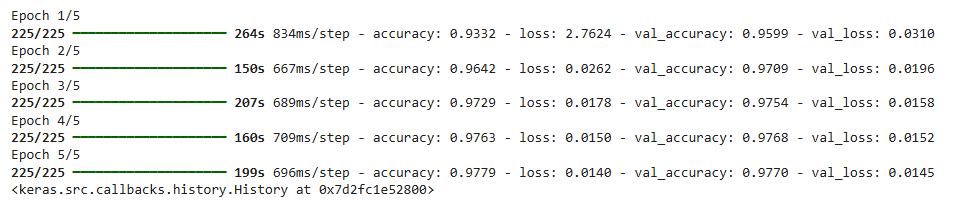
    callbacks=callbacks,

    validation\_data=(X\_val, Y\_val),

    batch\_size=batch\_size

)

**OUTPUT:**

****

**CODE:**

preds = model.predict(X\_val)

preds.max(), preds.min()



**CODE:**

!mkdir out

plt.figure(figsize=(10, 45))

s, e = 90, 98

index = 1

preds = (preds >= .5).astype('int')

for i, j, k in zip(X\_val[s:e], preds[s:e], Y\_val[s:e]):

    # write these images into file as well

    cv2.imwrite(f'./out/img-{index}.jpg', i)

    cv2.imwrite(f'./out/pred-{index}.jpg', j\*255.)

    cv2.imwrite(f'./out/ground-{index}.jpg', k\*255.)

    plt.subplot(10, 2, index)

    plt.imshow(i/255.)

    plt.title('Ground truth image')

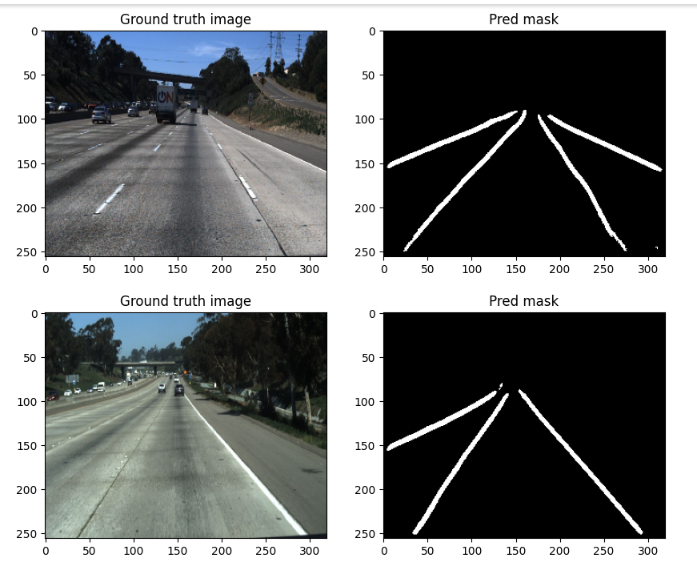
    plt.subplot(10, 2, index+1)

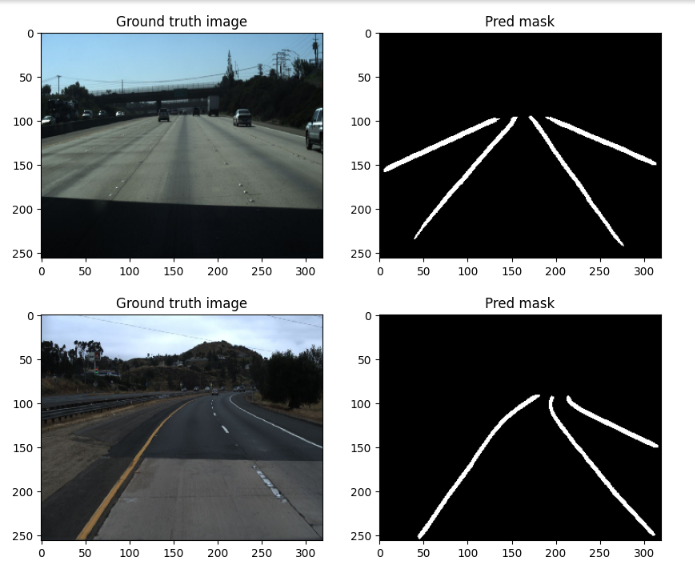
    plt.imshow(j, cmap='gray')

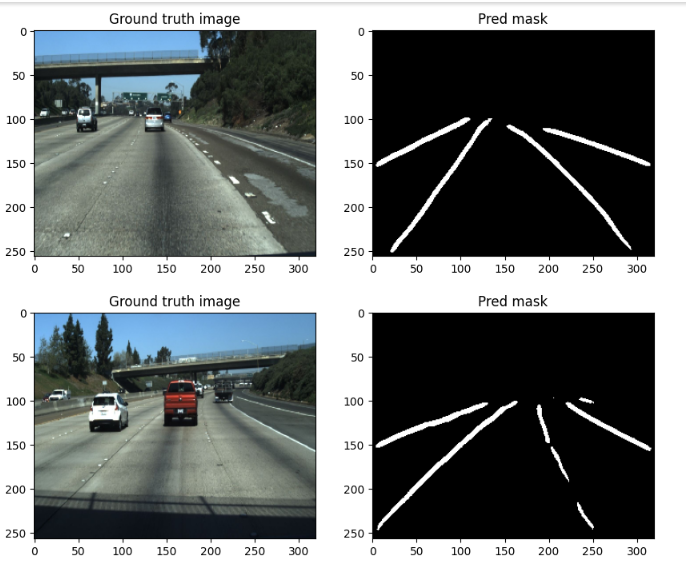
    plt.title('Pred mask')

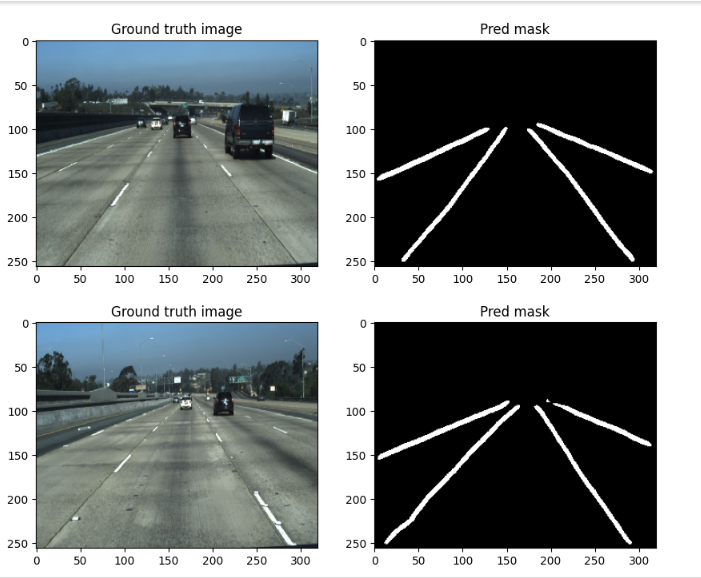
    index += 2

**OUTPUT:**

****

****

****

****