from google.colab import drive

drive.mount('/content/drive')

%cd '/content/drive/MyDrive/‘

!unzip ML.zip

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.image as npimg

import os

import keras

from keras.models import Sequential

from keras.optimizers import Adam

from keras.layers import Convolution2D, MaxPooling2D, Dropout, Flatten, Dense

import cv2

import pandas as pd

import random

import ntpath

from sklearn.utils import shuffle

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

datadir = 'ML'

columns = ['image\_id', 'angle', 'speed']

data = pd.read\_csv(os.path.join(datadir, 'training\_norm.csv'), names = columns, skiprows=[0])

pd.set\_option('display.max\_colwidth', -1)

data.head()

data.dtypes

#Updating datatypes for angle and speed

data["angle"] = pd.to\_numeric(data["angle"])

data["speed"] = pd.to\_numeric(data["speed"])

data['image\_id']= data['image\_id'].map(str)

data.head()

def addjpg(name):

return name + '.png'

data['image\_id'] = data['image\_id'].apply(addjpg)

data.head()

data.dtypes

def load\_img\_to\_list(datadir, df):

image\_path = []

angle = []

speed = []

for i in range(len(data)):

indexed\_data = data.iloc[i]

img = indexed\_data[0]

path = os.path.join(datadir, img.strip())

if(os.path.isfile(path)):

image\_path.append(path)

angle.append(indexed\_data[1])

speed.append(indexed\_data[2])

image\_paths = np.asarray(image\_path)

angle = np.asarray(angle)

speed = np.asarray(speed)

return image\_paths, angle, speed

image\_paths, angle, speed = load\_img\_to\_list('/content/drive/My Drive/ML/training\_data/', data)

angle.shape

speed.shape

image\_paths.shape

X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(image\_paths, angle, test\_size=0.2, random\_state=0)

print("Training Samples: {}\nValid Samples: {}".format(len(X\_train), len(X\_valid)))

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

axes[0].hist(Y\_train, bins=25, width=0.05, color='blue')

axes[0].set\_title('Training set')

axes[1].hist(Y\_valid, bins=25, width=0.05, color='red')

axes[1].set\_title('Validation set’)

def img\_preprocess(img):

img = npimg.imread(img)

img = img[60:135, :, :]

img = cv2.cvtColor(img, cv2.COLOR\_RGB2YUV)

img = cv2.GaussianBlur(img, (3, 3), 0)

img = cv2.resize(img, (100, 100))

img = img / 255

return img

!pwd

X\_train = np.array(list(map(img\_preprocess, X\_train)))

X\_valid = np.array(list(map(img\_preprocess, X\_valid)))

print(len(X\_train))

!pwd

%cd '/content/drive/My Drive/ML'

!pwd

path = "myModels"

exists = os.path.exists(path)

if not exists:

os.makedirs(path)

print("Path created!!”)

from tensorflow.keras.applications import ResNet50

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dropout, Flatten, Dense

from tensorflow.keras.optimizers import Adam

# MAKING A RESNET BASED MODEL

def myModel():

resnet = ResNet50(weights='imagenet', include\_top=False, input\_shape=(100, 100, 3))

for layer in resnet.layers[:-4]:

layer.trainable = False

for layer in resnet.layers:

print(layer, layer.trainable)

model = Sequential()

model.add(resnet)

model.add(Dropout(0.5))

model.add(Flatten())

model.add(Dense(100, activation='elu'))

model.add(Dropout(0.5))

model.add(Dense(50, activation='elu'))

model.add(Dropout(0.5))

model.add(Dense(10, activation='elu'))

model.add(Dropout(0.5))

model.add(Dense(1))

optimizer = Adam(learning\_rate=1e-3)

model.compile(loss='mse', optimizer=optimizer, metrics=['accuracy'])

return model

%cd 'myModels'

model = myModel()

print(model.summary)

#Creating checkpoints

checkpoint\_path = "/content/drive/My Drive/ML/training\_data/myModels/cp\_angle.ckpt"

checkpoint\_dir = os.path.dirname(checkpoint\_path)

cp\_callback = keras.callbacks.ModelCheckpoint(filepath=checkpoint\_path, save\_weights\_only=True, verbose=1)

history = model.fit(X\_train, Y\_train, epochs=32, validation\_data=(X\_valid, Y\_valid), batch\_size=64, verbose=1, shuffle=1, callbacks=[cp\_callback])

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.legend(['training', 'validation'])

plt.title('Loss')

plt.xlabel('Epoch')

model.save\_weights('/content/drive/My Drive/ML/training\_data/myModels/cp\_angle.ckpt’)

X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(image\_paths, speed, test\_size=0.2, random\_state=0)

print("Training Samples: {}\nValid Samples: {}".format(len(X\_train), len(X\_valid)))

fig, axes = plt.subplots(1, 2, figsize=(12, 4))

axes[0].hist(Y\_train, bins=25, width=0.05, color='blue')

axes[0].set\_title('Training set')

axes[1].hist(Y\_valid, bins=25, width=0.05, color='red')

axes[1].set\_title('Validation set’)

X\_train = np.array(list(map(img\_preprocess, X\_train)))

X\_valid = np.array(list(map(img\_preprocess, X\_valid)))

model = myModel()

print(model.summary)

#Creating checkpoints

checkpoint\_path = "/content/drive/My Drive/ML/training\_data/myModels/cp\_speed.ckpt"

checkpoint\_dir = os.path.dirname(checkpoint\_path)

cp\_callback = keras.callbacks.ModelCheckpoint(filepath=checkpoint\_path, save\_weights\_only=True, verbose=1)

history = model.fit(X\_train, Y\_train, epochs=28, validation\_data=(X\_valid, Y\_valid), batch\_size=128, verbose=1, shuffle=1, callbacks=[cp\_callback])

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.legend(['training', 'validation'])

plt.title('Loss')

plt.xlabel('Epoch')

model.save\_weights('/content/drive/My Drive/ML/training\_data/myModels/cp\_speed.ckpt’)

speed\_model = myModel()

speed\_model.load\_weights('/content/drive/My Drive/ML/training\_data/myModels/cp\_speed.ckpt')

# Re-evaluate the model

loss, acc = speed\_model.evaluate(X\_valid, Y\_valid, verbose=1)

print("Restored model, accuracy: {:5.2f}%".format(100 \* acc))

angle\_model = myModel()

angle\_model.load\_weights('/content/drive/My Drive/ML/training\_data/myModels/cp\_angle.ckpt')

# Re-evaluate the model

loss, acc = angle\_model.evaluate(X\_valid, Y\_valid, verbose=1)

print("Restored model, accuracy: {:5.2f}%".format(100 \* acc))

%cd '/content/drive/My Drive/ML/test\_data/'

folder\_dir = r'/content/drive/My Drive/ML/test\_data/'

image\_id = []

angle = []

speed = []

for img in os.listdir(folder\_dir):

if img.endswith(".png"):

image\_id.append(img)

image = img\_preprocess(img)

image = np.expand\_dims(image, axis=0)

ang = angle\_model.predict(image)

angle.append(ang)

speed1 = speed\_model.predict(image)

speed.append(speed1)

%cd '/content/drive/My Drive/ML/'

import pandas as pd

dict = {'image\_id':image\_id, 'angle':angle, 'speed':speed}

df = pd.DataFrame(dict)

df.to\_csv('answers1.csv',index = False)

import pandas as pd

filename = "answers1.csv"

column\_index = 0

# Read the CSV file into a Pandas DataFrame

df = pd.read\_csv("answers1.csv")

# Use the str.replace() method to remove the ".png" extension from the values in the specified column

df.iloc[:, 0] = df.iloc[:, 0].str.replace(".png", "")

# Write the modified DataFrame back to the original CSV file

df.to\_csv('answers1.csv', index=False)

import pandas as pd

#Load the CSV file into a pandas DataFrame

df = pd.read\_csv('answers1.csv')

#Sort the DataFrame by a specific column

df\_sorted = df.sort\_values(by='image\_id', ascending=True)

#Save the sorted DataFrame back to a CSV file

df\_sorted.to\_csv('output.csv', index=False)