***PROJECT:02***

***Horse Vs Human Classification***

**STEP 1:** First, we have to import Drive, to access data from google drive.

**CODE:**

**from google.colab import drive**

**drive.mount('/content/drive')**

**STEP 2:** Import necessary library

**CODE:**

**from tensorflow.keras.models import Model**

**from tensorflow.keras.layers import Dense,Input,Flatten, Conv2D, MaxPooling2D**

**import tensorflow as tf**

**import os**

**STEP 3:**

* Store the folder

From the horse and human folder in drive, the horse folder storing in train\_horse\_directory and human folder storing in train\_human\_directory to separating the horse images and human images.

**CODE:**

#Directory with our training horse pictures

**train\_horses\_dir = os.path.join('/content/drive/MyDrive/Machine Learning/horse-or-human/horses')**

#Directory with our training human pictures

**train\_humans\_dir = os.path.join('/content/drive/MyDrive/Machine Learning/horse-or-human/humans')**

* Name of images present in horse and human folder

**CODE:**

**train\_horses\_name = os.listdir(train\_horses\_dir)**

**print(train\_horses\_name[:10])**

**train\_humans\_name = os.listdir(train\_humans\_dir)**

**print(train\_horses\_name[:10])**

* Total images in folder

**CODE:**

**print("total training horses images:", len(os.listdir(train\_horses\_dir)))**

**print("total training humans images:", len(os.listdir(train\_humans\_dir)))**

* Plot the images

**CODE:**

#Plotting the images

**import matplotlib.pyplot as plt**

**import matplotlib.image as mpimg**

# Parameters for our graph; we will output images in a 4\*4 configuration

**nrows = 4**

**ncols = 4**

#Index for iterating over images

**pic\_index = 0**

# setup matplotlib fig, and size it to fit 4\*4 pixs

**fig = plt.gcf()**

**fig.set\_size\_inches(ncols \* 4, nrows \* 4)**

**pic\_index += 8**

**next\_horses\_pix = [os.path.join(train\_horses\_dir, fname)**

**for fname in train\_horses\_name[pic\_index-8:pic\_index]]**

**next\_humans\_pix = [os.path.join(train\_humans\_dir, fname)**

**for fname in train\_humans\_name[pic\_index-8:pic\_index] ]**

**for i, img\_path in enumerate(next\_horses\_pix+next\_humans\_pix):**

# setup subplot; subplot indices start at 1

**sp = plt.subplot(nrows, ncols, i + 1)**

**sp.axis('off')** # (do not show axis or gridlines)

**img = mpimg.imread(img\_path)**

**plt.imshow(img)**

**plt.show()**

**STEP 4:** Creating neural network

We are using multiple layer (maxpooling and convolution layer)

And then we have the flatten the result. Here 32,64 represent the

number of filters which we are using in the model and kernel size present the size of the filter. We are using activation function relu.

we have to flatten the output.

We have to pass the input value to neural network which will flatten the output. Dense layer also connected for flatten output . we have to create the output layer and activation function is sigmoid. sigmoid use for binary classification and the value will be 0 or 1.

**CODE:**

# input shape is desired size of images 300\*300 with 3 channel RGB

**inp\_layer = Input(shape=(300,300,3))**

# first layer (Convolution)

**conv1 = Conv2D(32, kernel\_size=3, activation='relu')(inp\_layer)**

# Second layer (Convolution)

**conv2 = Conv2D(32, kernel\_size=3, activation='relu')(conv1)**

**pool1 = MaxPooling2D(pool\_size=(2,2))(conv2)**

# Third layer (Convolution)

**conv3 = Conv2D(64, kernel\_size=3, activation='relu')(pool1)**

# Fourth layer (Convolution)

**conv4 = Conv2D(64, kernel\_size=3, activation='relu')(conv3)**

**pool2 = MaxPooling2D(pool\_size=(2,2))(conv4)**

**flatten1 = Flatten()(pool2)**

# Fully connected layer

**dense1 = Dense(64, activation = 'relu')(flatten1)**

**op\_layer = Dense(1, activation='sigmoid')(dense1)**

**model\_clf = Model(inputs=inp\_layer, outputs=op\_layer )**

**STEP 5:** Model summary

**CODE:**

**model\_clf.summary()**

**STEP 6:** Compile the model

**CODE:**

**model\_clf.compile(optimizer='adam', loss='binary\_crossentropy',metrics = ['accuracy'])**

**STEP 6:** Data Normalization

We need to work on training and validation data and we have to create the image generator for training and validation data to rescale the images.we have to import the ImageDataGenerator and all image will rescale by 1/255

**CODE:**

**from tensorflow.keras.preprocessing.image import ImageDataGenerator**

**train\_data\_gen = ImageDataGenerator( rescale = 1.0/255. )**

**validation\_datagen = ImageDataGenerator( rescale = 1.0/255. )**

**STEP 7:** Setting the directory for training and validation path to check our data is successfully connected or not.

**CODE:**

* For the Training and validation path

**train\_path = '/content/drive/MyDrive/Machine Learning/horse-or-human/'**

**validation\_path ='/content/drive/MyDrive/Machine Learning/validation-horse-or-human/'**

**CODE: FOR TRAINING DATA**

# (Flow training images in batches of 128 using train\_data\_generator)

**train\_data\_generator = train\_data\_gen.flow\_from\_directory(**

**train\_path,**

**#** (This is the source directory for training images)

**target\_size = (300,300),**

**#** (Fix the size of the images to 300,300)

**batch\_size = 128,**

**#** (since we use binary\_crossentropy loss, we need binary labels)

**class\_mode = 'binary'**

**)**

**CODE: FOR VALIDATION DATA**

#(Flow Validation images in batches of 128 using validation\_data\_generator )

**validation\_data\_generator=validation\_datagen.flow\_from\_directory(**

**validation\_path,**

**target\_size = (300,300),**

**batch\_size = 20,**

**class\_mode = 'binary'**

**)**

**STEP 8:** We have to create the callback function

**CODE:**

**class myCallback(tf.keras.callbacks.Callback):**

**def on\_epoch\_end(self,epoch,logs={}):**

**if(logs.get('accuracy')>0.98):**

**print("Plese stop training our model is ready")**

**self.model.stop\_training=True**

**CODE:**

#early stopping

**callback = myCallback()**

**STEP 9:** Now we have to execute the model which is stored in model\_df. We are taking epoch is 30 providing validation set which stored in validation data generator and store the result in history. And also using callback function to reduce overfitting.

**CODE:**

**history = model\_clf.fit\_generator(**

**train\_data\_generator,**

**epochs = 30,**

**validation\_data = validation\_data\_generator,**

**callbacks = [callback]**

**)**

**STEP 10: Prediction for new images of horse and human**

**CODE:**

**import numpy as np**

**from google.colab import files**

**from keras.preprocessing import image**

**uploaded = files.upload()**

**for fn in uploaded.keys():**

#predicting images

**path = '/content/' + fn**

**img = image.load\_img(path, target\_size=(300,300))**

**x = image.img\_to\_array(img)**

**x = np.expand\_dims(x, axis=0)**

**images = np.vstack([x])**

**classes = model\_clf.predict(images, batch\_size=20)**

**print (classes[0])**

**if classes[0]>0.5:**

**print(fn + " is a human")**

**else:**

**print(fn + " is a horse")**