***PROJECT:01***

***CAT VS DOG CLASSIFICATION USING CNN MODEL***

**STEP:1**

Run command to kaggle

**COMMAND:**

!mkdir -p ~/.kaggle

!cp kaggle.json ~/.kaggle/

**STEP:2**

In Kaggle profile, we have to create new API token which will download the Kaggle.json file and we have to upload in google colab.

Copy API command from dog vs cat dataset and paste this command in colab notebook with!

**COMMAND:**

!kaggle datasets download -d salader/dogs-vs-cats

This will download data in colab notebook.

* Unzip the data file with

**COMMAND:**

**import zipfile**

**zip\_ref = zipfile.ZipFile('/content/dogs-vs-cats.zip', 'r')**

**zip\_ref.extractall('/content')**

**zip\_ref.close()**

**STEP 3:**

Import library

**CODE:**

**import tensorflow as tf**

**from tensorflow import keras**

**from keras import Sequential**

**from keras.layers import Dense,Conv2D,MaxPooling2D,Flatten,BatchNormalization,Dropout**

**STEP 4:**

In train folder we have dog and cat images and we have to give this image to model so we required generators which will divide images in to batches. And it will load the batches once at a time in RAM and remove it after use and load another batch.

Generators is very useful to process large amount of data.

We are using generators which is provided by keras. The name of Generators is image\_dataset\_from\_directory function.

We are making two generator one for training data and other for testing data.

**CODE:**

**train\_ds = keras.utils.image\_dataset\_from\_directory(**

**directory = '/content/train',**

**labels='inferred',**

**label\_mode = 'int',**

**batch\_size=32,**

**image\_size=(256,256)**

**)**

**validation\_ds = keras.utils.image\_dataset\_from\_directory(**

**directory = '/content/test',**

**labels='inferred',**

**label\_mode = 'int',**

**batch\_size=32,**

**image\_size=(256,256)**

**)**

**OUTPUT:**

Found 20000 files belonging to 2 classes.

Found 5000 files belonging to 2 classes.

**STEP 5:**

Normalization:

Images stored in numpy array and the range it from 0 to 256 and we have to convert it into 0 to 1 .so basically, we have to normalize the numpy array to find better result.

**CODE:**

**def process(image,label):**

**image = tf.cast(image/255. ,tf.float32)**

**return image,label**

**train\_ds = train\_ds.map(process)**

**validation\_ds = validation\_ds.map(process)**

**STEP:6**

Create the CNN model: There would be three convolution layers.In

1st layer there would be 32 filters.

2nd layer there would be 64 filters.

3rd layer there would be 128 filters.

**CODE:**

**model = Sequential()**

**model.add(Conv2D(32,kernel\_size=(3,3),padding='valid',activation='relu',input\_shape=(256,256,3)))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))**

**model.add(Conv2D(64,kernel\_size=(3,3),padding='valid',activation='relu'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))**

**model.add(Conv2D(128,kernel\_size=(3,3),padding='valid',activation='relu'))**

**model.add(BatchNormalization())**

**model.add(MaxPooling2D(pool\_size=(2,2),strides=2,padding='valid'))**

**model.add(Flatten())**

**model.add(Dense(128,activation='relu'))**

**model.add(Dropout(0.1))**

**model.add(Dense(64,activation='relu'))**

**model.add(Dropout(0.1))**

**model.add(Dense(1,activation='sigmoid'))**

* Further step is Model Summary:

**CODE:**

**model.summary()**

Now we have to compile the model with adam optimizer, loss function is binary crossentrophy for binary classification problem, metrics is accuracy.

**CODE:**

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

**STEP 7:**

Now we have to execute the model which is stored in train\_ds ,epoch is 10 and providing validation set which I stored in validation\_ds.and storing the result in History to plot the graph further.

**CODE:**

**History=model.fit(train\_ds,epochs=10,validation\_data=validation\_ds)**

**STEP 8:**

without reducing overfitting plot the graph between training accuracy and validation accuracy. from the graph when epoch is increasing training accuracy is also increasing but the validation accuracy is fluctuating between 75 to 80. The gap between the training accuracy and validation accuracy is saying that the model is overfitted.

We also plot the graph between validation loss and training loss. When the epoch I increasing the training loss is decreasing and validation loss is increasing.

After adding batch normalization and dropout in convolution layer in our CNN Model. And it will improve the model accuracy.it also improve the training of our model.

**CODE FOR TRAINING AND VALIDATION ACCURACY:**

**import matplotlib.pyplot as plt**

**plt.plot(history.history['accuracy'],color='red',label='train')**

**plt.plot(history.history['val\_accuracy'],color='blue',label='validation')**

**plt.legend()**

**plt.show()**

**CODE FOR TRAINING AND VALIDATION LOSS:**

**plt.plot(history.history['loss'],color='red',label='train')**

**plt.plot(history.history['val\_loss'],color='blue',label='validation')**

**plt.legend()**

**plt.show()**

**STEP 9:**

**STEP 9:**

Prediction on new images.

* Import the required library

**CODE:**

**import cv2**

**import matplotlib.pyplot as plt**

* Show/Read image

**CODE:**

**test\_img = plt.imread('/content/dog.10.jpg')**

* Check the shape of image

**CODE:**

**test\_img.shape**

* Resize the image

**COD:**

**test\_img = cv2.resize(test\_img,(256,256))**

* Reshape the image

We pass the data to our model in form of batches and we have only one image so we converted in to 4d tensor (1,256,256,3) so the meaning of that is there is only one image and dimension is (256,256,3) and it will stored in test input.

**COD:**

**test\_input = test\_img.reshape((1,256,256,3))**

* Predict the model

From this we will find array 1or 0 for test\_input.1 or 0 will assign according to images.

**CODE:**

**model.predict(test\_input)**