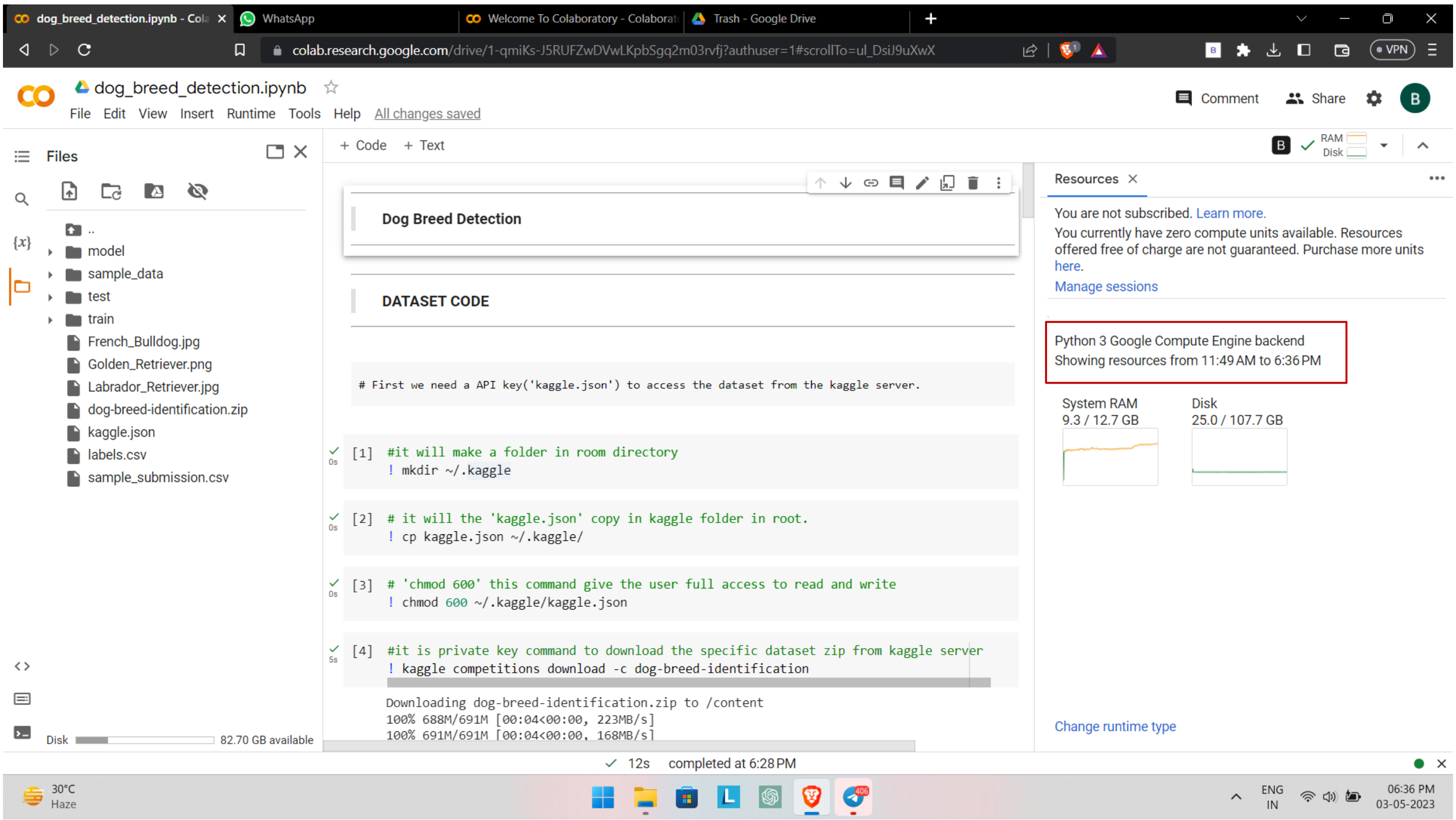
# Execution Report


dog_breed_detection.ipynb - Cola
WhatsApp
Welcome To Colaboratory - Colaborat
Trash - Google Drive
colab.research.google.com/drive/1-qmiKs-J5RUFZwDVwLKpbSgq2m03rvfj?authuser=1#scrollTo=ul_DsiJ9uXwX
VPN
dog_breed_detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
+ Code + Text
RAM
Disk
Resources

Files:

- ..
- model
- sample_data
- test
- train
- French_Bulldog.jpg
- Golden_Retriever.png
- Labrador_Retriever.jpg
- dog-breed-identification.zip
- kaggle.json
- labels.csv
- sample_submission.csv

Disk 82.70 GB available

**Dog Breed Detection**

**DATASET CODE**

```
# First we need a API key('kaggle.json') to access the dataset from the kaggle server.
```

```
[1]  #it will make a folder in room directory
     ! mkdir ~/.kaggle
```

```
[2]  # it will the 'kaggle.json' copy in kaggle folder in root.
     ! cp kaggle.json ~/.kaggle/
```

```
[3]  # 'chmod 600' this command give the user full access to read and write
     ! chmod 600 ~/.kaggle/kaggle.json
```

```
[4]  #it is private key command to download the specific dataset zip from kaggle server
     ! kaggle competitions download -c dog-breed-identification

     Downloading dog-breed-identification.zip to /content
     100% 688M/691M [00:04<00:00, 223MB/s]
     100% 691M/691M [00:04<00:00, 168MB/s]
```

You are not subscribed. Learn more.

You currently have zero compute units available. Resources offered free of charge are not guaranteed. Purchase more units here.

Manage sessions

Python 3 Google Compute Engine backend
Showing resources from 11:49 AM to 6:36 PM

System RAM
9.3 / 12.7 GB

Disk
25.0 / 107.7 GB

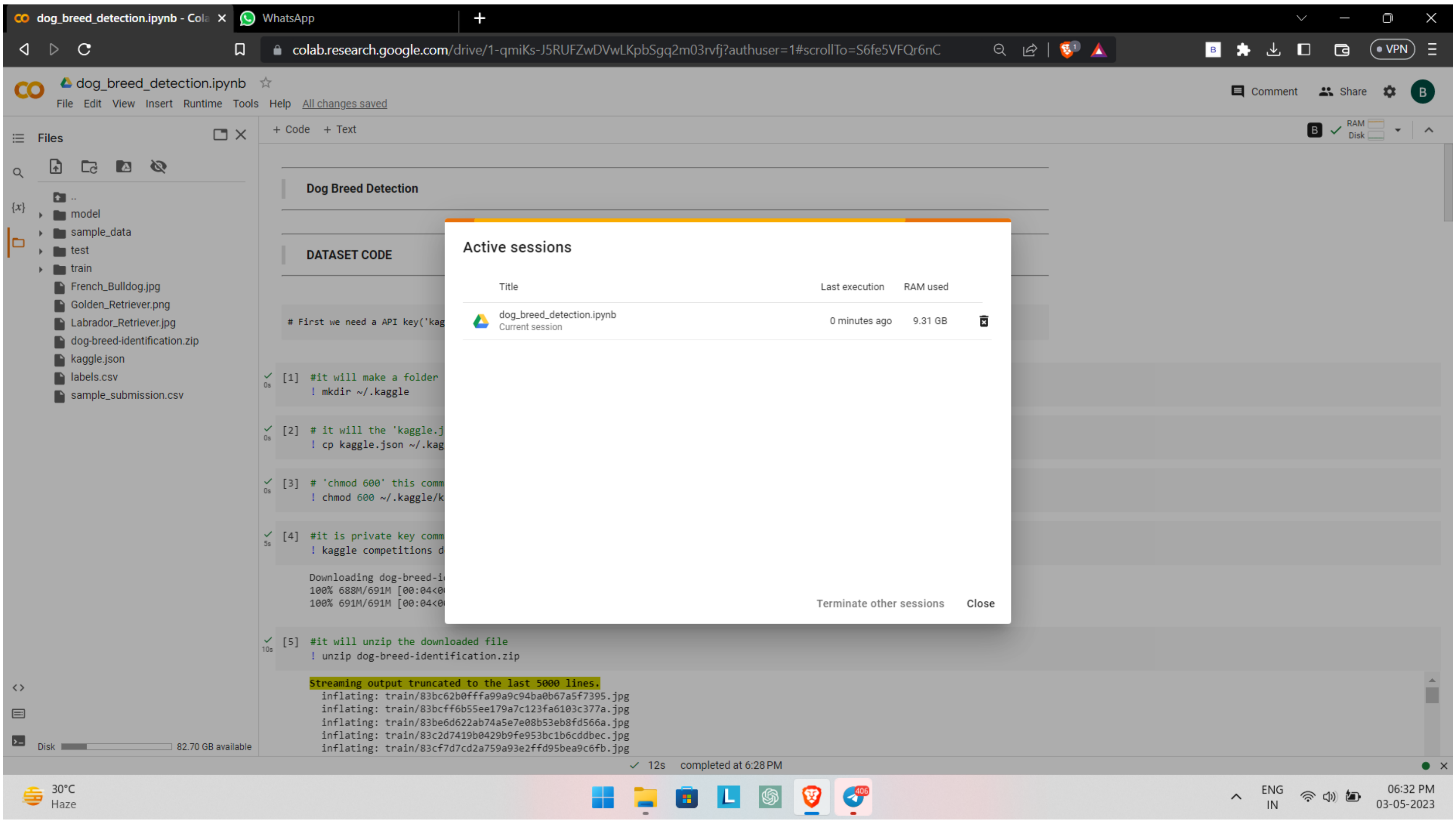
Change runtime type

✓ 12s completed at 6:28 PM

30°C Haze

ENG IN

06:36 PM
03-05-2023


dog_breed_detection.ipynb - Cola
WhatsApp
colab.research.google.com/drive/1-qmiKs-J5RUFZwDVwLKpbSgq2m03rvfj?authuser=1#scrollTo=S6fe5VFQr6nC
VPN
dog_breed_detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment
Share
Files
+ Code + Text
RAM
Disk
Dog Breed Detection
DATASET CODE
Terminate other sessions
Close

**Active sessions**

| Title | Last execution | RAM used |
|---|---|---|
| dog_breed_detection.ipynb<br>Current session | 0 minutes ago | 9.31 GB |

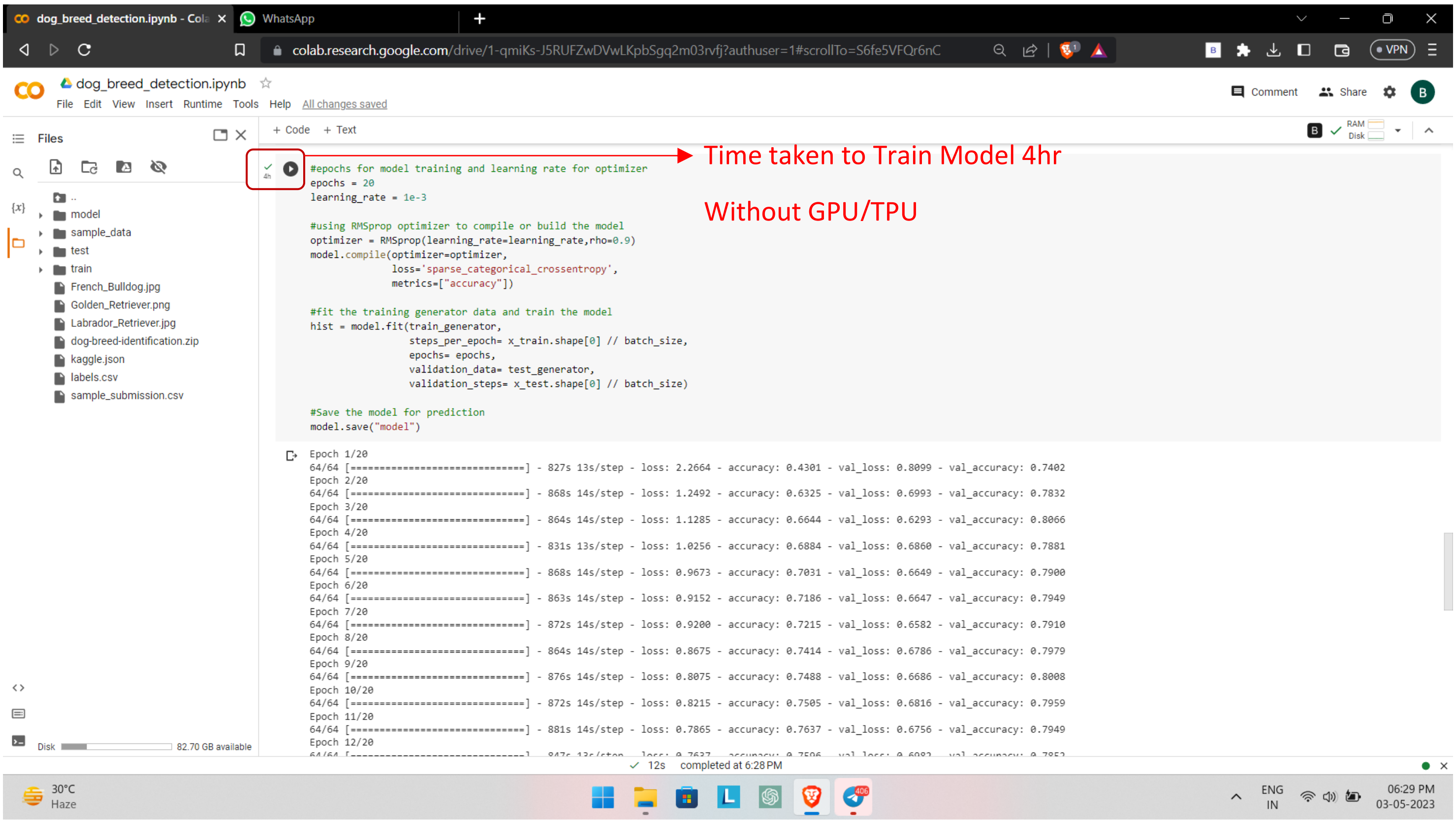
```
[5]  #it will unzip the downloaded file
     ! unzip dog-breed-identification.zip

     Streaming output truncated to the last 5000 lines.
       inflating: train/83bc62b0fffa99a9c94ba0b67a5f7395.jpg
       inflating: train/83bcff6b55ee179a7c123fa6103c377a.jpg
       inflating: train/83be6d622ab74a5e7e08b53eb8fd566a.jpg
       inflating: train/83c2d7419b0429b9fe953bc1b6cddbec.jpg
       inflating: train/83cf7d7cd2a759a93e2ffd95bea9c6fb.jpg
```

✓ 12s completed at 6:28 PM

30°C Haze

ENG IN

06:32 PM
03-05-2023

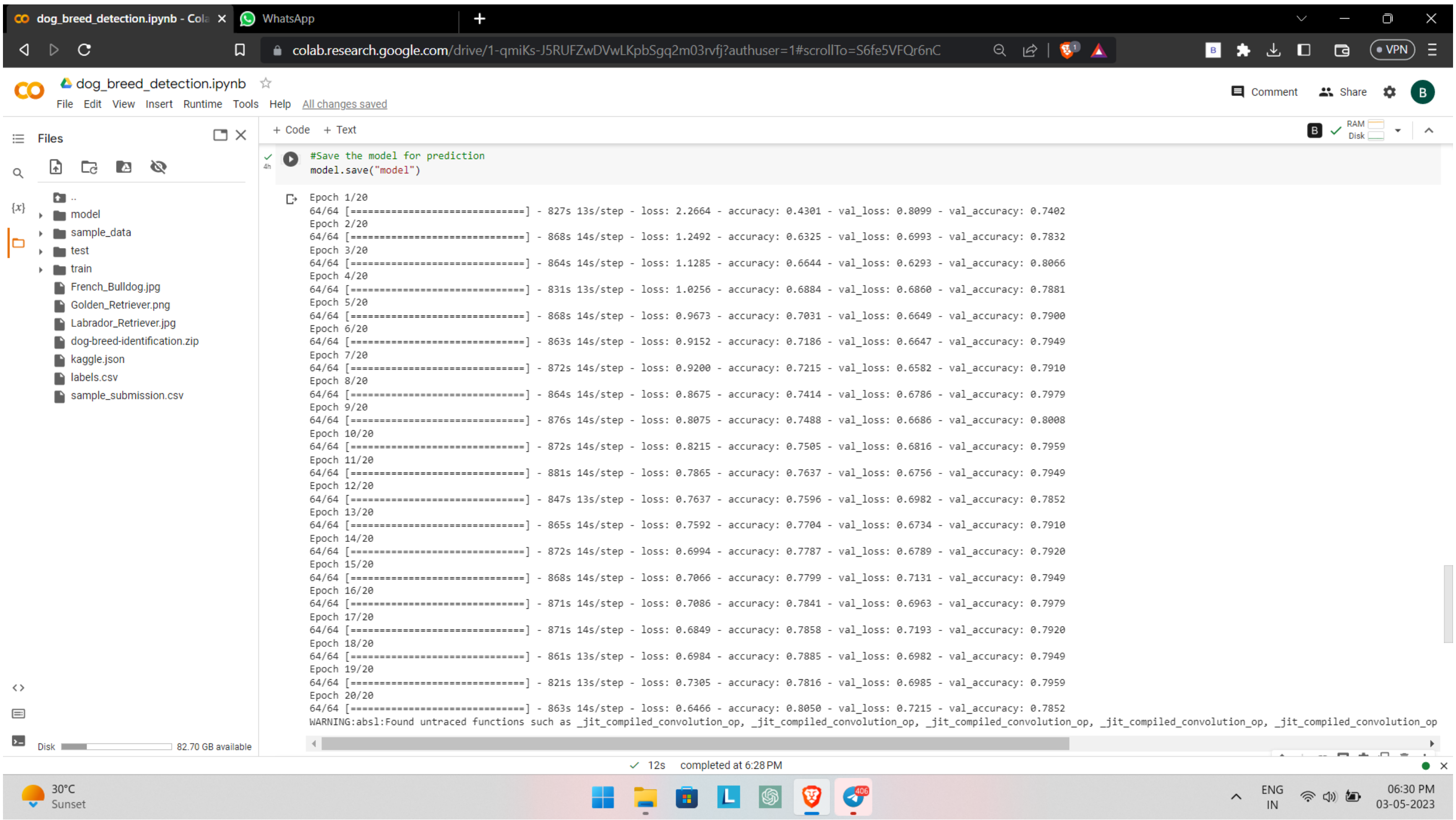


Time taken to Train Model 4hr

Without GPU/TPU

```
#epochs for model training and learning rate for optimizer
epochs = 20
learning_rate = 1e-3

#using RMSprop optimizer to compile or build the model
optimizer = RMSprop(learning_rate=learning_rate,rho=0.9)
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])

#fit the training generator data and train the model
hist = model.fit(train_generator,
                 steps_per_epoch= x_train.shape[0] // batch_size,
                 epochs= epochs,
                 validation_data= test_generator,
                 validation_steps= x_test.shape[0] // batch_size)

#Save the model for prediction
model.save("model")
```

```
Epoch 1/20
64/64 [==============================] - 827s 13s/step - loss: 2.2664 - accuracy: 0.4301 - val_loss: 0.8099 - val_accuracy: 0.7402
Epoch 2/20
64/64 [==============================] - 868s 14s/step - loss: 1.2492 - accuracy: 0.6325 - val_loss: 0.6993 - val_accuracy: 0.7832
Epoch 3/20
64/64 [==============================] - 864s 14s/step - loss: 1.1285 - accuracy: 0.6644 - val_loss: 0.6293 - val_accuracy: 0.8066
Epoch 4/20
64/64 [==============================] - 831s 13s/step - loss: 1.0256 - accuracy: 0.6884 - val_loss: 0.6860 - val_accuracy: 0.7881
Epoch 5/20
64/64 [==============================] - 868s 14s/step - loss: 0.9673 - accuracy: 0.7031 - val_loss: 0.6649 - val_accuracy: 0.7900
Epoch 6/20
64/64 [==============================] - 863s 14s/step - loss: 0.9152 - accuracy: 0.7186 - val_loss: 0.6647 - val_accuracy: 0.7949
Epoch 7/20
64/64 [==============================] - 872s 14s/step - loss: 0.9200 - accuracy: 0.7215 - val_loss: 0.6582 - val_accuracy: 0.7910
Epoch 8/20
64/64 [==============================] - 864s 14s/step - loss: 0.8675 - accuracy: 0.7414 - val_loss: 0.6786 - val_accuracy: 0.7979
Epoch 9/20
64/64 [==============================] - 876s 14s/step - loss: 0.8075 - accuracy: 0.7488 - val_loss: 0.6686 - val_accuracy: 0.8008
Epoch 10/20
64/64 [==============================] - 872s 14s/step - loss: 0.8215 - accuracy: 0.7505 - val_loss: 0.6816 - val_accuracy: 0.7959
Epoch 11/20
64/64 [==============================] - 881s 14s/step - loss: 0.7865 - accuracy: 0.7637 - val_loss: 0.6756 - val_accuracy: 0.7949
Epoch 12/20
64/64 [==============================] - 847s 13s/step - loss: 0.7637 - accuracy: 0.7596 - val_loss: 0.6982 - val_accuracy: 0.7852
```

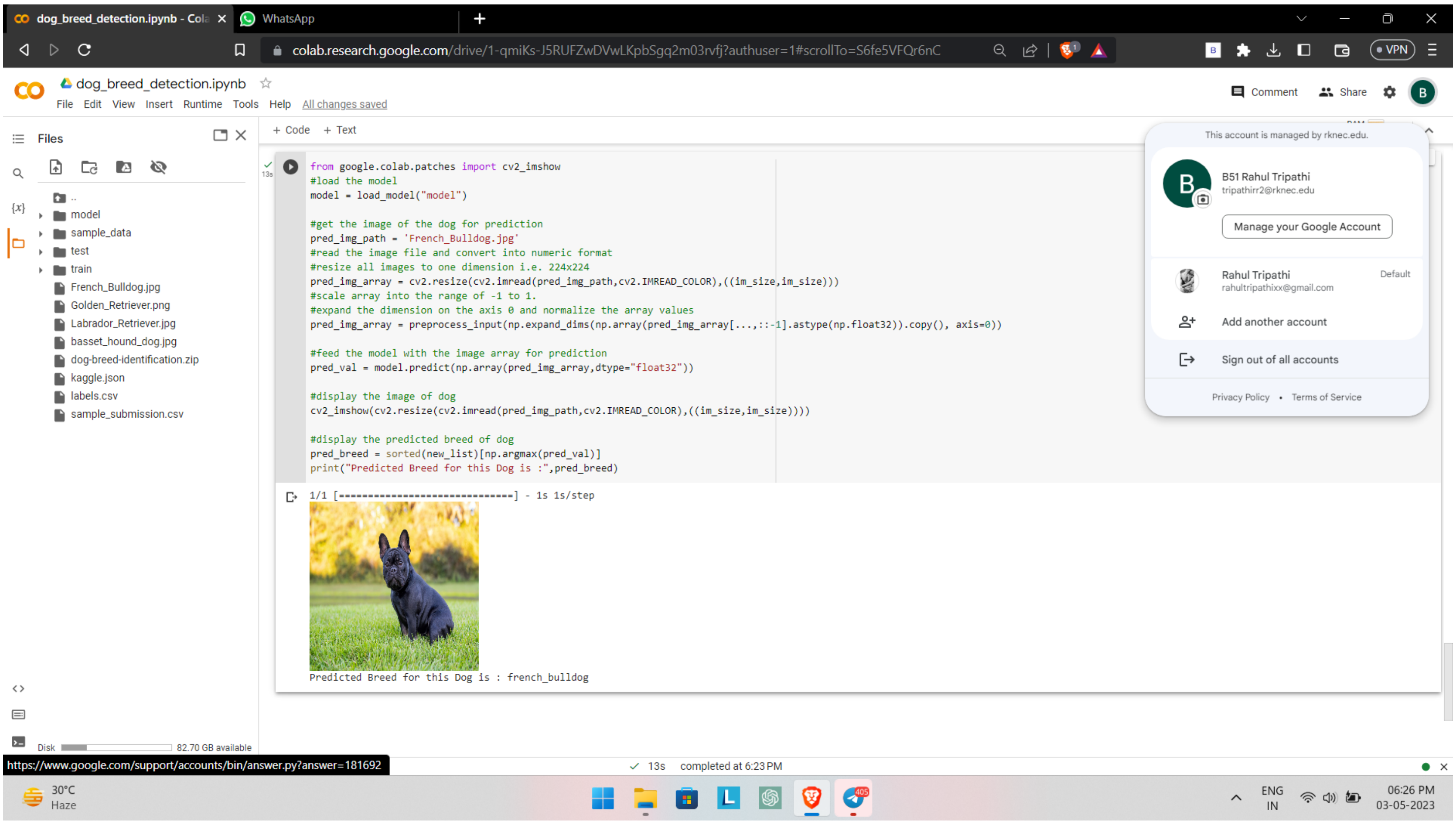


```
#Save the model for prediction
model.save("model")
```

```
Epoch 1/20
64/64 [==============================] - 827s 13s/step - loss: 2.2664 - accuracy: 0.4301 - val_loss: 0.8099 - val_accuracy: 0.7402
Epoch 2/20
64/64 [==============================] - 868s 14s/step - loss: 1.2492 - accuracy: 0.6325 - val_loss: 0.6993 - val_accuracy: 0.7832
Epoch 3/20
64/64 [==============================] - 864s 14s/step - loss: 1.1285 - accuracy: 0.6644 - val_loss: 0.6293 - val_accuracy: 0.8066
Epoch 4/20
64/64 [==============================] - 831s 13s/step - loss: 1.0256 - accuracy: 0.6884 - val_loss: 0.6860 - val_accuracy: 0.7881
Epoch 5/20
64/64 [==============================] - 868s 14s/step - loss: 0.9673 - accuracy: 0.7031 - val_loss: 0.6649 - val_accuracy: 0.7900
Epoch 6/20
64/64 [==============================] - 863s 14s/step - loss: 0.9152 - accuracy: 0.7186 - val_loss: 0.6647 - val_accuracy: 0.7949
Epoch 7/20
64/64 [==============================] - 872s 14s/step - loss: 0.9200 - accuracy: 0.7215 - val_loss: 0.6582 - val_accuracy: 0.7910
Epoch 8/20
64/64 [==============================] - 864s 14s/step - loss: 0.8675 - accuracy: 0.7414 - val_loss: 0.6786 - val_accuracy: 0.7979
Epoch 9/20
64/64 [==============================] - 876s 14s/step - loss: 0.8075 - accuracy: 0.7488 - val_loss: 0.6686 - val_accuracy: 0.8008
Epoch 10/20
64/64 [==============================] - 872s 14s/step - loss: 0.8215 - accuracy: 0.7505 - val_loss: 0.6816 - val_accuracy: 0.7959
Epoch 11/20
64/64 [==============================] - 881s 14s/step - loss: 0.7865 - accuracy: 0.7637 - val_loss: 0.6756 - val_accuracy: 0.7949
Epoch 12/20
64/64 [==============================] - 847s 13s/step - loss: 0.7637 - accuracy: 0.7596 - val_loss: 0.6982 - val_accuracy: 0.7852
Epoch 13/20
64/64 [==============================] - 865s 14s/step - loss: 0.7592 - accuracy: 0.7704 - val_loss: 0.6734 - val_accuracy: 0.7910
Epoch 14/20
64/64 [==============================] - 872s 14s/step - loss: 0.6994 - accuracy: 0.7787 - val_loss: 0.6789 - val_accuracy: 0.7920
Epoch 15/20
64/64 [==============================] - 868s 14s/step - loss: 0.7066 - accuracy: 0.7799 - val_loss: 0.7131 - val_accuracy: 0.7949
Epoch 16/20
64/64 [==============================] - 871s 14s/step - loss: 0.7086 - accuracy: 0.7841 - val_loss: 0.6963 - val_accuracy: 0.7979
Epoch 17/20
64/64 [==============================] - 871s 14s/step - loss: 0.6849 - accuracy: 0.7858 - val_loss: 0.7193 - val_accuracy: 0.7920
Epoch 18/20
64/64 [==============================] - 861s 13s/step - loss: 0.6984 - accuracy: 0.7885 - val_loss: 0.6982 - val_accuracy: 0.7949
Epoch 19/20
64/64 [==============================] - 821s 13s/step - loss: 0.7305 - accuracy: 0.7816 - val_loss: 0.6985 - val_accuracy: 0.7959
Epoch 20/20
64/64 [==============================] - 863s 14s/step - loss: 0.6466 - accuracy: 0.8050 - val_loss: 0.7215 - val_accuracy: 0.7852
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolution_op
```

**Number of Epoch Taken 20 Time to get best possible Output**

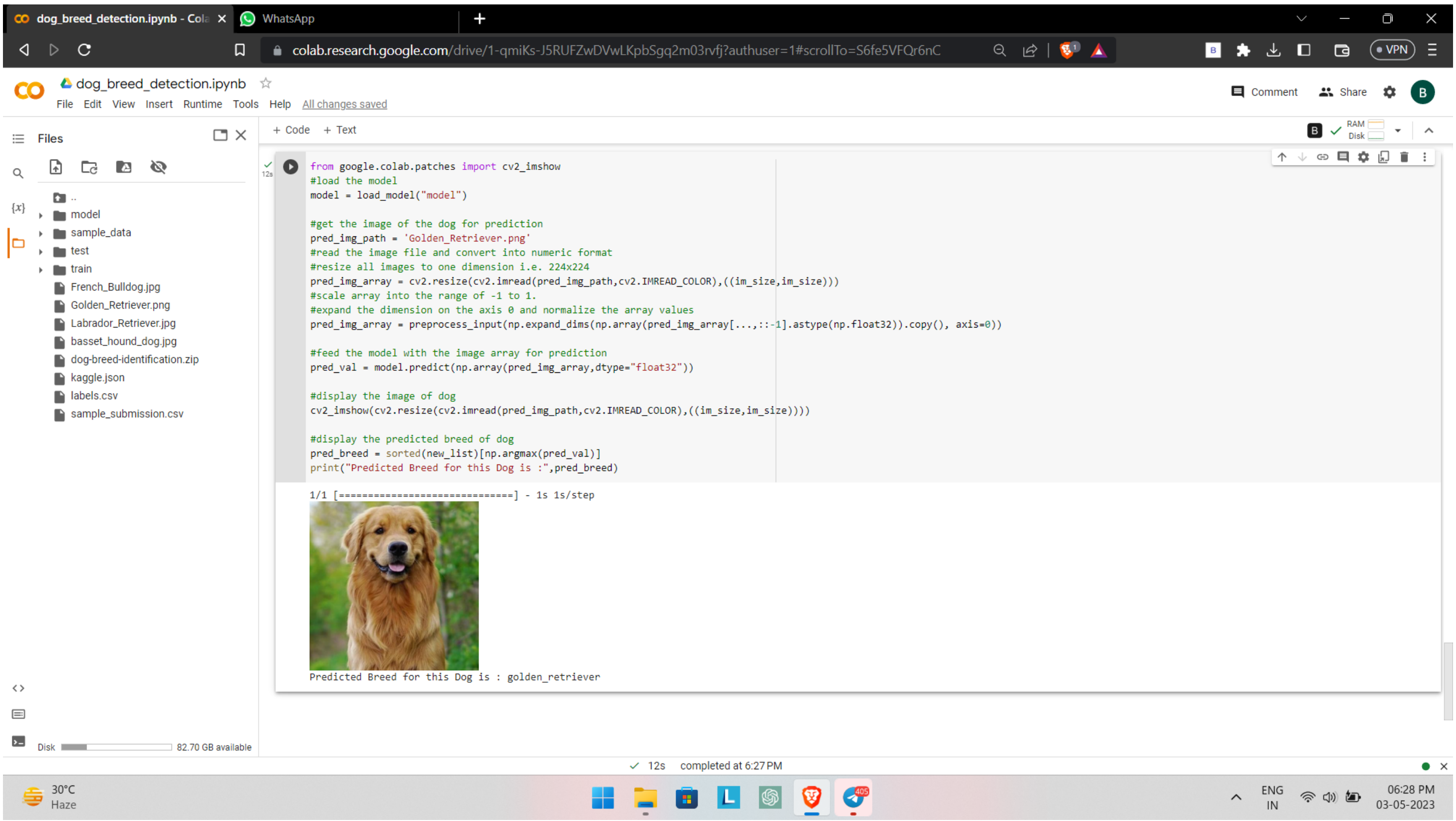
# Output with Three Different Breed Images of Dog



```
from google.colab.patches import cv2_imshow
#load the model
model = load_model("model")

#get the image of the dog for prediction
pred_img_path = 'French_Bulldog.jpg'
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array = cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size)))
#scale array into the range of -1 to 1.
#expand the dimension on the axis 0 and normalize the array values
pred_img_array = preprocess_input(np.expand_dims(np.array(pred_img_array[...,::-1].astype(np.float32)).copy(), axis=0))

#feed the model with the image array for prediction
pred_val = model.predict(np.array(pred_img_array,dtype="float32"))

#display the image of dog
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size))))

#display the predicted breed of dog
pred_breed = sorted(new_list)[np.argmax(pred_val)]
print("Predicted Breed for this Dog is :",pred_breed)
```

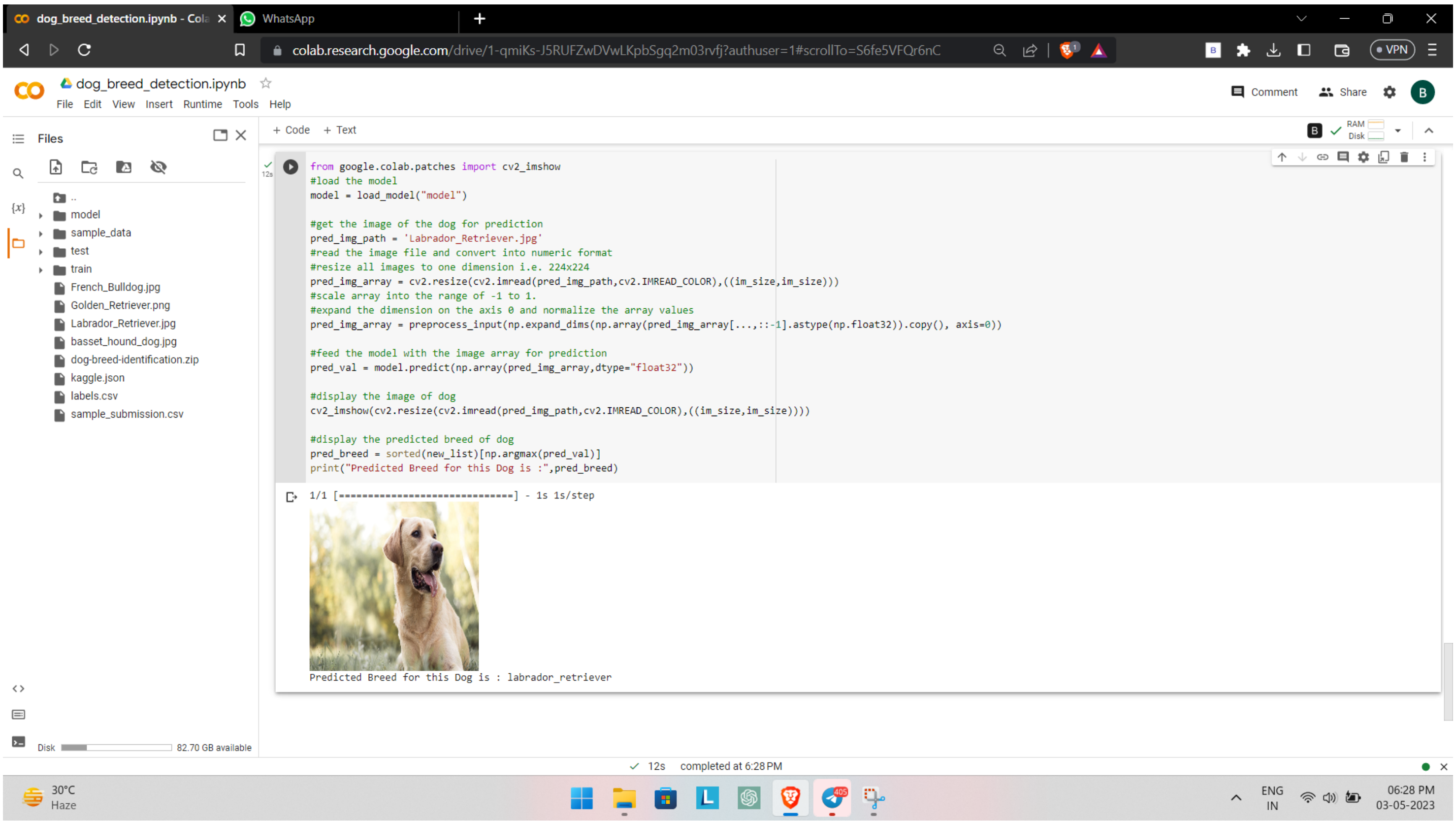
```
1/1 [==============================] - 1s 1s/step
Predicted Breed for this Dog is : french_bulldog
```

dog_breed_detection.ipynb - Cola
WhatsApp
colab.research.google.com/drive/1-qmiKs-J5RUFZwDVwLKpbSgq2m03rvfj?authuser=1#scrollTo=S6fe5VFQr6nC
dog_breed_detection.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
Files
+ Code + Text
model
sample_data
test
train
French_Bulldog.jpg
Golden_Retriever.png
Labrador_Retriever.jpg
basset_hound_dog.jpg
dog-breed-identification.zip
kaggle.json
labels.csv
sample_submission.csv
This account is managed by rknec.edu.
B51 Rahul Tripathi
tripathirr2@rknec.edu
Manage your Google Account
Rahul Tripathi
rahultripathixx@gmail.com
Default
Add another account
Sign out of all accounts
Privacy Policy • Terms of Service
Disk 82.70 GB available
https://www.google.com/support/accounts/bin/answer.py?answer=181692
13s completed at 6:23 PM
30°C
Haze
ENG
IN
06:26 PM
03-05-2023



```
from google.colab.patches import cv2_imshow
#load the model
model = load_model("model")

#get the image of the dog for prediction
pred_img_path = 'Golden_Retriever.png'
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array = cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size)))
#scale array into the range of -1 to 1.
#expand the dimension on the axis 0 and normalize the array values
pred_img_array = preprocess_input(np.expand_dims(np.array(pred_img_array[...,::-1].astype(np.float32)).copy(), axis=0))

#feed the model with the image array for prediction
pred_val = model.predict(np.array(pred_img_array,dtype="float32"))

#display the image of dog
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size))))

#display the predicted breed of dog
pred_breed = sorted(new_list)[np.argmax(pred_val)]
print("Predicted Breed for this Dog is :",pred_breed)
```

```
1/1 [==============================] - 1s 1s/step
Predicted Breed for this Dog is : golden_retriever
```



```
from google.colab.patches import cv2_imshow
#load the model
model = load_model("model")

#get the image of the dog for prediction
pred_img_path = 'Labrador_Retriever.jpg'
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array = cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size)))
#scale array into the range of -1 to 1.
#expand the dimension on the axis 0 and normalize the array values
pred_img_array = preprocess_input(np.expand_dims(np.array(pred_img_array[...,::-1].astype(np.float32)).copy(), axis=0))

#feed the model with the image array for prediction
pred_val = model.predict(np.array(pred_img_array,dtype="float32"))

#display the image of dog
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size))))

#display the predicted breed of dog
pred_breed = sorted(new_list)[np.argmax(pred_val)]
print("Predicted Breed for this Dog is :",pred_breed)
```

```
1/1 [==============================] - 1s 1s/step
Predicted Breed for this Dog is : labrador_retriever
```

## Complete Code Execution is Below

## Dog Breed Detection

## DATASET CODE

```
 # First we need a API key('kaggle.json') to access the dataset from the kaggle server.
```

```
#it will make a folder in room directory
! mkdir ~/.kaggle
```

```
# it will the 'kaggle.json' copy in kaggle folder in root.
! cp kaggle.json ~/.kaggle/
```

```
# 'chmod 600' this command give the user full access to read and write
! chmod 600 ~/.kaggle/kaggle.json
```

```
#it is private key command to download the specific dataset zip from kaggle server
! kaggle competitions download -c dog-breed-identification
```

```
     Downloading dog-breed-identification.zip to /content
     100% 688M/691M [00:04<00:00, 223MB/s]
     100% 691M/691M [00:04<00:00, 168MB/s]
```

```
#it will unzip the downloaded file
! unzip dog-breed-identification.zip
```

```
     Streaming output truncated to the last 5000 lines.
       inflating: train/83bc62b0fffa99a9c94ba0b67a5f7395.jpg
       inflating: train/83bcff6b55ee179a7c123fa6103c377a.jpg
       inflating: train/83be6d622ab74a5e7e08b53eb8fd566a.jpg
       inflating: train/83c2d7419b0429b9fe953bc1b6cddbec.jpg
       inflating: train/83cf7d7cd2a759a93e2ffd95bea9c6fb.jpg
       inflating: train/83d405858f0931722ef21e8ac0adee4d.jpg
       inflating: train/83d4125a4c3c7dc5956563276cb1cd74.jpg
       inflating: train/83f0bb565b2186dbcc6a9d009cb26ff2.jpg
       inflating: train/83fad0718581a696132c96c166472627.jpg
       inflating: train/83fbbcc9a612e3f712b1ba199da61f20.jpg
       inflating: train/8403d8936430c2f05ab7d74d23c2c0cb.jpg
       inflating: train/8406d837b2d7fac1c3cd621abb4c4f9e.jpg
       inflating: train/840b67d26e5e43f8eb6430f62d4ba1ac.jpg
       inflating: train/840db91ba4600148f3dcb06ec419b421.jpg
       inflating: train/840dbad5a691c22611d85b2488bf4cbb.jpg
       inflating: train/8410ced9ebc1759a7ebce5c42bfb5222.jpg
       inflating: train/841463629c4833816e216cbb041c2778.jpg
       inflating: train/8429dcca4ae91c4e0345e4ba48b0d69f.jpg
       inflating: train/842e3c6e44fda4102fe83d07dac72b3e.jpg
       inflating: train/8431a6ce7c70e5e36698e821eedf24b5.jpg
       inflating: train/8434b6c3cee87e28395197d6fc7d3489.jpg
       inflating: train/8436be99589db6a99cfac1b894421ea6.jpg
       inflating: train/843cbc1fc239d24534859bd272c3bc16.jpg
       inflating: train/843d766d92a7b6d6a85a81e56a99c51f.jpg
       inflating: train/84421c01900b34e3e1ba42f2424fbd33.jpg
       inflating: train/844dde39a9e8987e510e8d46ec4da714.jpg
       inflating: train/8452a26d7243a299ea782a7ba4036f1f.jpg
       inflating: train/8454b5e6546f04871561de8f10d868c7.jpg
       inflating: train/84564a69c0d0fa36e0810188943683a1.jpg
       inflating: train/84605f5fc5ad89a66b9b277e1223e962.jpg
       inflating: train/8463aa43d88bee057082434ccc806bb0.jpg
       inflating: train/8467fbd75a8fe64da70df5410b6c4f09.jpg
       inflating: train/846d6384787fff8dc17d488e6b86c209.jpg
       inflating: train/8470a6fdf4db9b088494aaa9384ba9d0.jpg
       inflating: train/84728e78632c0910a69d33f82e62638c.jpg
       inflating: train/8477ac111ca6a9f11c2edfa43a933cad.jpg
       inflating: train/8480ad94841309fc4ce874c4b1afc90c.jpg
       inflating: train/848133f97b3e97b1b0fab0402e572d98.jpg
       inflating: train/8485bc3f3fd64b90be74d7f020c61f54.jpg
       inflating: train/8486e8159f169e8c3d4697e5c859760f.jpg
       inflating: train/848f7a0b665b118e4a3b85029b1794e0.jpg
       inflating: train/8490222d4744064aa7a8621a1c274965.jpg
       inflating: train/8494afd34e3a2e81bec37e4dfdc67f8d.jpg
       inflating: train/84aaf49fb53d423d4aed05ab79559b0c.jpg
       inflating: train/84ab21940432e5b42cfacc58cd84c861.jpg
       inflating: train/84accc2dc9f5bb3ebee89fe1bf23639c.jpg
       inflating: train/84adb2cc13b65cf25418cde969b9bb0e.jpg
       inflating: train/84b612a8e43c6debbc9951cb24ec9ba0.jpg
       inflating: train/84b62d2def32fc85092cabe2c722c135.jpg
```

```
      inflating: train/84bcd47e09b0ef3f0b6e3f47f232a77c.jpg
      inflating: train/84be9b9f59aa586f1b188781b2c47a3e.jpg
      inflating: train/84c6bdd4bb818edd4c088f27312d028f.jpg
      inflating: train/84d2dd9eff021b6095a4b1e2ba3c1c0c.jpg
      inflating: train/84de398dd5408d91b133e2e95628120a.jpg
      inflating: train/84dfe42ce71204b367c2b4000eb6ba5c.jpg
      inflating: train/84e567b15311f0c891858f56f0175867.jpg
      inflating: train/84f5f076b0b951d68f88c8b795b7135e.jpg
```

## MAIN CODE

```
# load all required libraries for Dog's Breed Identification Project
import cv2
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import load_model, Model
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout, BatchNormalization
from tensorflow.keras.applications.resnet_v2 import ResNet50V2, preprocess_input
```

```
#read the csv file
df_labels = pd.read_csv("labels.csv")
#store training and testing images folder location
train_file = 'train/'
test_file = 'test/'
```

```
#check the total number of unique breed in our dataset file
print("Total number of unique Dog Breeds :",len(df_labels.breed.unique()))
```

```
    Total number of unique Dog Breeds : 120
```

```
#specify number
num_breeds = 60
im_size = 224
batch_size = 64
encoder = LabelEncoder()
```

```
#get only 60 unique breeds record
breed_dict = list(df_labels['breed'].value_counts().keys())
new_list = sorted(breed_dict,reverse=True)[:num_breeds*2+1:2]
#change the dataset to have only those 60 unique breed records
df_labels = df_labels.query('breed in @new_list')
```

```
#create new column which will contain image name with the image extension
df_labels['img_file'] = df_labels['id'].apply(lambda x: x + ".jpg")
```

```
#create a numpy array of the shape
#(number of dataset records, image size , image size, 3 for rgb channel ayer)
#this will be input for model
train_x = np.zeros((len(df_labels), im_size, im_size, 3), dtype='float32')

#iterate over img_file column of our dataset
for i, img_id in enumerate(df_labels['img_file']):
  #read the image file and convert into numeric format
  #resize all images to one dimension i.e. 224x224
  #we will get array with the shape of
  # (224,224,3) where 3 is the RGB channels layers
  img = cv2.resize(cv2.imread(train_file+img_id,cv2.IMREAD_COLOR),((im_size,im_size)))
  #scale array into the range of -1 to 1.
  #preprocess the array and expand its dimension on the axis 0
  img_array = preprocess_input(np.expand_dims(np.array(img[...,::-1].astype(np.float32)).copy(), axis=0))
  #update the train_x variable with new element
  train_x[i] = img_array
```

```
#This will be the target for the model.
#convert breed names into numerical format
train_y = encoder.fit_transform(df_labels["breed"].values)
```

```
#split the dataset in the ratio of 80:20.
#80% for training and 20% for testing purpose
x_train, x_test, y_train, y_test = train_test_split(train_x,train_y,test_size=0.2,random_state=42)
```

```
#Image augmentation using ImageDataGenerator class
train_datagen = ImageDataGenerator(rotation_range=45,
                                   width_shift_range=0.2,
                                   height_shift_range=0.2,
                                   shear_range=0.2,
                                   zoom_range=0.25,
                                   horizontal_flip=True,
                                   fill_mode='nearest')

#generate images for training sets
train_generator = train_datagen.flow(x_train,
                                     y_train,
                                     batch_size=batch_size)

#same process for Testing sets also by declaring the instance
test_datagen = ImageDataGenerator()

test_generator = test_datagen.flow(x_test,
                                     y_test,
                                     batch_size=batch_size)


#building the model using ResNet50V2 with input shape of our image array
#weights for our network will be from of imagenet dataset
#we will not include the first Dense layer
resnet = ResNet50V2(input_shape = [im_size,im_size,3], weights='imagenet', include_top=False)
#freeze all trainable layers and train only top layers
for layer in resnet.layers:
    layer.trainable = False

#add global average pooling layer and Batch Normalization layer
x = resnet.output
x = BatchNormalization()(x)
x = GlobalAveragePooling2D()(x)
x = Dropout(0.5)(x)
#add fully connected layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.5)(x)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50v2_weights_tf_dim_ordering_tf_ker
94668760/94668760 [==============================] - 1s 0us/step
```

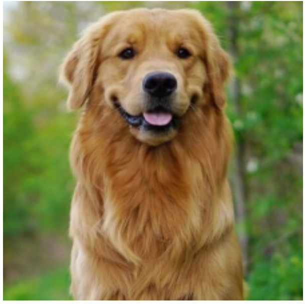
```
#add output layer having the shape equal to number of breeds
predictions = Dense(num_breeds, activation='softmax')(x)

#create model class with inputs and outputs
model = Model(inputs=resnet.input, outputs=predictions)
#model.summary()


#epochs for model training and learning rate for optimizer
epochs = 20
learning_rate = 1e-3

#using RMSprop optimizer to compile or build the model
optimizer = RMSprop(learning_rate=learning_rate,rho=0.9)
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=["accuracy"])

#fit the training generator data and train the model
hist = model.fit(train_generator,
                 steps_per_epoch= x_train.shape[0] // batch_size,
                 epochs= epochs,
                 validation_data= test_generator,
                 validation_steps= x_test.shape[0] // batch_size)

#Save the model for prediction
model.save("model")
```

```
Epoch 1/20
64/64 [==============================] - 827s 13s/step - loss: 2.2664 - accuracy: 0.4301 - val_loss: 0.8099 - val_accuracy: 0.7402
Epoch 2/20
64/64 [==============================] - 868s 14s/step - loss: 1.2492 - accuracy: 0.6325 - val_loss: 0.6993 - val_accuracy: 0.7832
Epoch 3/20
64/64 [==============================] - 864s 14s/step - loss: 1.1285 - accuracy: 0.6644 - val_loss: 0.6293 - val_accuracy: 0.8066
Epoch 4/20
64/64 [==============================] - 831s 13s/step - loss: 1.0256 - accuracy: 0.6884 - val_loss: 0.6860 - val_accuracy: 0.7881
Epoch 5/20
64/64 [==============================] - 868s 14s/step - loss: 0.9673 - accuracy: 0.7031 - val_loss: 0.6649 - val_accuracy: 0.7900
Epoch 6/20
64/64 [==============================] - 863s 14s/step - loss: 0.9152 - accuracy: 0.7186 - val_loss: 0.6647 - val_accuracy: 0.7949
```

```
Epoch 7/20
64/64 [==============================] - 872s 14s/step - loss: 0.9200 - accuracy: 0.7215 - val_loss: 0.6582 - val_accuracy: 0.7910
Epoch 8/20
64/64 [==============================] - 864s 14s/step - loss: 0.8675 - accuracy: 0.7414 - val_loss: 0.6786 - val_accuracy: 0.7979
Epoch 9/20
64/64 [==============================] - 876s 14s/step - loss: 0.8075 - accuracy: 0.7488 - val_loss: 0.6686 - val_accuracy: 0.8008
Epoch 10/20
64/64 [==============================] - 872s 14s/step - loss: 0.8215 - accuracy: 0.7505 - val_loss: 0.6816 - val_accuracy: 0.7959
Epoch 11/20
64/64 [==============================] - 881s 14s/step - loss: 0.7865 - accuracy: 0.7637 - val_loss: 0.6756 - val_accuracy: 0.7949
Epoch 12/20
64/64 [==============================] - 847s 13s/step - loss: 0.7637 - accuracy: 0.7596 - val_loss: 0.6982 - val_accuracy: 0.7852
Epoch 13/20
64/64 [==============================] - 865s 14s/step - loss: 0.7592 - accuracy: 0.7704 - val_loss: 0.6734 - val_accuracy: 0.7910
Epoch 14/20
64/64 [==============================] - 872s 14s/step - loss: 0.6994 - accuracy: 0.7787 - val_loss: 0.6789 - val_accuracy: 0.7920
Epoch 15/20
64/64 [==============================] - 868s 14s/step - loss: 0.7066 - accuracy: 0.7799 - val_loss: 0.7131 - val_accuracy: 0.7949
Epoch 16/20
64/64 [==============================] - 871s 14s/step - loss: 0.7086 - accuracy: 0.7841 - val_loss: 0.6963 - val_accuracy: 0.7979
Epoch 17/20
64/64 [==============================] - 871s 14s/step - loss: 0.6849 - accuracy: 0.7858 - val_loss: 0.7193 - val_accuracy: 0.7920
Epoch 18/20
64/64 [==============================] - 861s 13s/step - loss: 0.6984 - accuracy: 0.7885 - val_loss: 0.6982 - val_accuracy: 0.7949
Epoch 19/20
64/64 [==============================] - 821s 13s/step - loss: 0.7305 - accuracy: 0.7816 - val_loss: 0.6985 - val_accuracy: 0.7959
Epoch 20/20
64/64 [==============================] - 863s 14s/step - loss: 0.6466 - accuracy: 0.8050 - val_loss: 0.7215 - val_accuracy: 0.7852
WARNING:absl:Found untraced functions such as _jit_compiled_convolution_op, _jit_compiled_convolution_op, _jit_compiled_convolutio
```

```
from google.colab.patches import cv2_imshow
#load the model
model = load_model("model")

#get the image of the dog for prediction
pred_img_path = 'Golden_Retriever.png'
#read the image file and convert into numeric format
#resize all images to one dimension i.e. 224x224
pred_img_array = cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size)))
#scale array into the range of -1 to 1.
#expand the dimension on the axis 0 and normalize the array values
pred_img_array = preprocess_input(np.expand_dims(np.array(pred_img_array[...,::-1].astype(np.float32)).copy(), axis=0))

#feed the model with the image array for prediction
pred_val = model.predict(np.array(pred_img_array,dtype="float32"))

#display the image of dog
cv2_imshow(cv2.resize(cv2.imread(pred_img_path,cv2.IMREAD_COLOR),((im_size,im_size))))

#display the predicted breed of dog
pred_breed = sorted(new_list)[np.argmax(pred_val)]
print("Predicted Breed for this Dog is :",pred_breed)
```

```
1/1 [==============================] - 1s 1s/step
```



```
Predicted Breed for this Dog is : golden_retriever
```

✓ 13s completed at 5:15 PM ● ×