```
import zipfile

with zipfile.ZipFile('/content/drive/MyDrive/dog-breed-identification.zip', 'r') as zip_ref:
    zip_ref.extractall('/content/')



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")


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


train_y = encoder.fit_transform(df_labels["breed"].values)


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_kernels
94668760/94668760 [==============================] - 3s 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 = 10
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 1/10
64/64 [==============================] - 998s 16s/step - loss: 0.7733 - accuracy: 0.7549 - val_loss: 0.7058 - val_accuracy: 0.7969
Epoch 2/10
64/64 [==============================] - 994s 16s/step - loss: 0.7339 - accuracy: 0.7763 - val_loss: 0.6868 - val_accuracy: 0.8037
Epoch 3/10
64/64 [==============================] - 975s 15s/step - loss: 0.7417 - accuracy: 0.7721 - val_loss: 0.6591 - val_accuracy: 0.8037
Epoch 4/10
```

```
64/64 [==============================] - 990s 16s/step - loss: 0.7379 - accuracy: 0.7733 - val_loss: 0.6699 - val_accuracy: 0.8057
Epoch 5/10
64/64 [==============================] - 996s 16s/step - loss: 0.7452 - accuracy: 0.7684 - val_loss: 0.6961 - val_accuracy: 0.7969
Epoch 6/10
64/64 [==============================] - 994s 16s/step - loss: 0.7175 - accuracy: 0.7856 - val_loss: 0.6932 - val_accuracy: 0.7998
Epoch 7/10
64/64 [==============================] - 997s 16s/step - loss: 0.6823 - accuracy: 0.7893 - val_loss: 0.6833 - val_accuracy: 0.8076
Epoch 8/10
64/64 [==============================] - 995s 16s/step - loss: 0.6850 - accuracy: 0.7942 - val_loss: 0.6920 - val_accuracy: 0.8076
Epoch 9/10
64/64 [==============================] - 999s 16s/step - loss: 0.6819 - accuracy: 0.7830 - val_loss: 0.7173 - val_accuracy: 0.8105
Epoch 10/10
64/64 [==============================] - 1009s 16s/step - loss: 0.6796 - accuracy: 0.7956 - val_loss: 0.6995 - val_accuracy: 0.8076
```

```
#load the model
model = load_model("model")

#get the image of the dog for prediction
pred_img_path = 'golden.jpeg'
#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
```