```
# Importing the necessary packages

from zipfile import ZipFile

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

import tensorflow

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

from tqdm import tqdm # for tracking progress of an iterative loop
import cv2  # for loading of images

from sklearn.metrics import accuracy_score
```

Dog Breed Classification

In this project we will use traditional CNN, CNN with data augmentation and finally transfer Learning by VGG16 model with weights pre-trained on Imagenet to solve the dog breed classification problem

Load Dataset Files

```
# Mounting the google drive

from google.colab import drive
drive.mount('/content/gdrive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.a

Enter your authorization code:
··········
Mounted at /content/gdrive
```

Now, upload the given dataset file shared with you in your google drive and give its path for the below given `project_path` variable. For example, a path is given below according to the file path in our google drive. You need to change this to match the path of yours.

```
project_path = "/content/gdrive/My Drive/Colab Notebooks/DogBreed_Classification/"
```

Run the below code to extract all the images in the train.zip files given in the dataset. We are going to use these images as train and validation sets and their labels in further steps.

```
with ZipFile(project_path + 'train.zip', 'r') as z:
  z.extractall()
```

Repeat the same step for test.zip

```
with ZipFile(project_path + 'test.zip', 'r') as z:
  z.extractall()
```

Repeat the same step for sample_submission.csv.zip

```
with ZipFile(project_path + 'sample_submission.csv.zip', 'r') as z:
  z.extractall()
```

Repeat the same step for labels.csv.zip

```
with ZipFile(project_path + 'labels.csv.zip', 'r') as z:
```

```
with ZipFile(project_path + "labels.csv.zip", 'r') as z:
  z.extractall()
```

After this process, we will have 4 files - Train folder, test folder and labels.csv and sample_submission.csv as part of your google drive

Read labels.csv file using pandas

```
labels_df = pd.read_csv("labels.csv")
```

```
labels_df.head()
```

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever

```
labels_df.shape
```

```
(10222, 2)
```

Print the count of each category of Dogs given in the dataset

```
# Number of categories
len(labels_df['breed'].unique())
```

```
120
```

```
print("The number of unique categories in the dataset are: ", labels_df['breed'].nunique())
```

```
print("The count of each category of dogs in the dataset are:\n", labels_df['breed'].value_counts())
```

```
The count of each category of dogs in the dataset are:
 scottish_deerhound      126
maltese_dog             117
afghan_hound            116
entlebucher             115
bernese_mountain_dog    114
                       ...
brabancon_griffon        67
komondor                 67
golden_retriever         67
briard                   66
eskimo_dog               66
Name: breed, Length: 120, dtype: int64
```

Get one-hot encodings of labels

```
# As the data is in the form of strings, first, the labels are converted into numbers

Label_Encoder = LabelEncoder()
labels_df['breed'] = Label_Encoder.fit_transform(labels_df['breed'])
```

```
labels_df.head()
```

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	19
1	001513dfcb2ffafc82cccf4d8bbaba97	37
2	001cdf01b096e06d78e9e5112d419397	85
3	00214f311d5d2247d5dfe4fe24b2303d	15
4	0021f9ceb3235effd7fcde7f7538ed62	49

```
# Second, the labels with integer values are converted into one hot encoders
```

```
one_hot_labels = tensorflow.keras.utils.to_categorical(labels_df['breed'])
```

```
one_hot_labels.shape
```

```
(10222, 120)
```

Preparing training dataset

1. Write a code which reads each and every id from labels.csv file and loads the corresponding image (in RGB - 128, 128, 3) from the train folder.
2. Create 2 variables
   a. x_train - Should have all the images of the dogs from train folder
   b. y_train - Corresponding label of the dog
   Note: The id of the dog images and its corresponding labels are available in labels.csv file
   Hint: Watch the video shared on "Preparing the training dataset" if you face issue on creating the training dataset

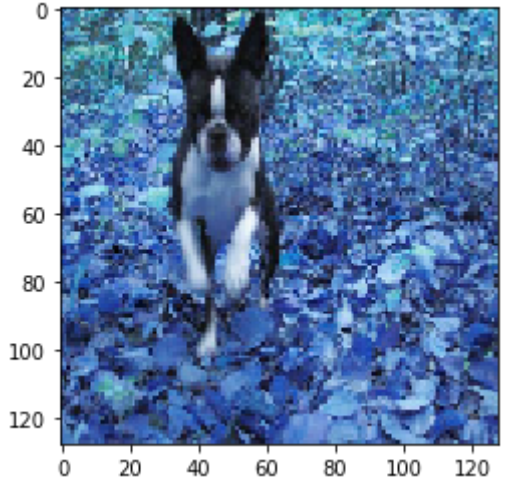
```
X_train = []
Y_train = []

for f, img in tqdm(labels_df.values): # f for format .jpg
  train_img = cv2.imread('./train/{}.jpg'.format(f), 1)
  label = one_hot_labels[img]
  train_img_resize = cv2.resize(train_img, (128, 128))
  X_train.append(train_img_resize)
  Y_train.append(label)
```

```
100%|██████████| 10222/10222 [00:29<00:00, 349.08it/s]
```

```
plt.imshow(X_train[0])
```

```
<matplotlib.image.AxesImage at 0x7f4974484f60>
```




```
Y_train[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0.], dtype=float32)
```

```
print("Length of X_train:", len(X_train), "\nLength of Y_train:", len(Y_train))
```

```
Length of X_train: 10222
Length of Y_train: 10222
```

Normalize the training data and convert into 4 dimensions so that it can be used as an input to conv layers in the model

```
X_train_arr = np.array(X_train) / 255
Y_train_arr = np.array(Y_train)
```

```
X_train_arr = X_train_arr.reshape(X_train_arr.shape[0], 128, 128, 3)
```

Split the training and validation data from `x_train_data` and `y_train_data` obtained from above step

```
X_train_data, X_val_data, Y_train_data, Y_val_data = train_test_split(X_train_arr, Y_train_arr, test_size = 0.3, random_state = 47)
```

Loading the test data

Read the id column from the samples_submission.csv and store it in test_img

```
test_img = pd.read_csv("sample_submission.csv")["id"]
test_img.head()
```

```
0    000621fb3cbb32d8935728e48679680e
1    00102ee9d8eb90812350685311fe5890
2    0012a730dfa437f5f3613fb75efcd4ce
3    001510bc8570bbeee98c8d80c8a95ec1
4    001a5f3114548acdefa3d4da05474c2e
Name: id, dtype: object
```

Run the below code to load the test image files in x_test_feature

```
img_rows = 128
img_cols = 128
```

```
x_test_feature = []
for f in tqdm(test_img.values): # f for format ,jpg
    img = cv2.imread('./test/{}.jpg'.format(f), 1)
    img_resize = cv2.resize(img, (img_rows, img_cols))
    x_test_feature.append(img_resize)
```

```
100%|██████████| 10357/10357 [00:28<00:00, 363.77it/s]
```

Normalize the test data and convert it into 4 dimensions

```
x_test_feature = np.array(x_test_feature) / 255
```

```
x_test_feature = x_test_feature.reshape(x_test_feature.shape[0], 128, 128, 3)
```

```
x_test_feature.shape
```

```
(10357, 128, 128, 3)
```

Build a basic conv neural network with 2 conv layers (kernel sizes - 5 and 3) add layers as mentioned below for classification.

1. Add a Dense layer with 256 neurons with `relu` activation
2. Add a Dense layer with 120 neurons as final layer (as there are 120 classes in the given dataset) with `softmax` activation for classifiaction.

```
# Clear out tensorflow memory
tensorflow.keras.backend.clear_session()

# Define Model
model = tensorflow.keras.models.Sequential()

# 1st Conv Layer
model.add(tensorflow.keras.layers.Conv2D(32, (5,5), activation='relu', input_shape=(128, 128, 3)))

# 2nd Conv Layer
model.add(tensorflow.keras.layers.Conv2D(64, (3,3), activation='relu'))
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

```
# Flattening the data
model.add(tensorflow.keras.layers.Flatten())

# 1st dense layer
model.add(tensorflow.keras.layers.Dense(256, activation='relu'))

# Output layer
model.add(tensorflow.keras.layers.Dense(120, activation = 'softmax'))
```

Use batch_size = 128 and epochs = 10 and execute the model

```
# Loss and Optimizer
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

# Training the model
basic_model = model.fit(X_train_data, Y_train_data,
                        batch_size=128,
                        epochs=10,
                        validation_data=(X_val_data, Y_val_data))
```

```
Train on 7155 samples, validate on 3067 samples
Epoch 1/10
7155/7155 [==============================] - 17s 2ms/sample - loss: 6.8813 - acc: 0.0317 - val_loss: 4.0981 - val_acc: 0.0401
Epoch 2/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 3.9017 - acc: 0.0794 - val_loss: 4.0464 - val_acc: 0.0522
Epoch 3/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 3.2286 - acc: 0.2194 - val_loss: 4.4605 - val_acc: 0.0414
Epoch 4/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 1.7169 - acc: 0.5737 - val_loss: 5.6908 - val_acc: 0.0378
Epoch 5/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 0.4296 - acc: 0.9023 - val_loss: 8.0055 - val_acc: 0.0375
Epoch 6/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 0.0804 - acc: 0.9846 - val_loss: 9.5319 - val_acc: 0.0326
Epoch 7/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 0.0347 - acc: 0.9961 - val_loss: 9.0635 - val_acc: 0.0329
Epoch 8/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 0.0174 - acc: 0.9983 - val_loss: 10.7931 - val_acc: 0.0349
Epoch 9/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 0.0082 - acc: 0.9986 - val_loss: 11.0687 - val_acc: 0.0381
Epoch 10/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 0.0054 - acc: 0.9996 - val_loss: 11.4136 - val_acc: 0.0395
```

The model accuracy is very poor !!!!

Use Data Augmentation in the above model to see if the accuracy improves

```
train_datagen = tensorflow.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=90,
    width_shift_range=0.2,
    height_shift_range=0.2,
    rescale=1./255,
    horizontal_flip=True,
    zoom_range=0.2,
    shear_range=0.2)
```

```
val_datagen = tensorflow.keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255)
```

- Using the above objects, create the image generators with variable names `train_generator` and `val_generator`

  You need to use train_datagen.flow() and val_datagen.flow()

```
train_generator = train_datagen.flow(X_train_data, y=Y_train_data)
val_generator = train_datagen.flow(X_train_data, y=Y_train_data)
```

- Fit the model using fit_generator() using `train_generator` and `val_generator` from the above step with 10 epochs

```
model.fit_generator(train_generator, epochs=10, steps_per_epoch=200,
                    validation_data=val_generator, validation_steps=200)
```

```
Epoch 1/10
199/200 [============================>.] - ETA: 0s - loss: 4.3239 - acc: 0.0280Epoch 1/10
200/200 [==============================] - 22s 108ms/step - loss: 4.1192 - acc: 0.0292
200/200 [==============================] - 43s 217ms/step - loss: 4.3229 - acc: 0.0283 - val_loss: 4.1192 - val_acc: 0.0292
Epoch 2/10
199/200 [============================>.] - ETA: 0s - loss: 4.1322 - acc: 0.0293Epoch 1/10
200/200 [==============================] - 22s 108ms/step - loss: 4.1151 - acc: 0.0292
200/200 [==============================] - 42s 212ms/step - loss: 4.1321 - acc: 0.0291 - val_loss: 4.1151 - val_acc: 0.0292
Epoch 3/10
199/200 [============================>.] - ETA: 0s - loss: 4.1333 - acc: 0.0305Epoch 1/10
200/200 [==============================] - 21s 106ms/step - loss: 4.0983 - acc: 0.0350
200/200 [==============================] - 42s 210ms/step - loss: 4.1332 - acc: 0.0308 - val_loss: 4.0983 - val_acc: 0.0350
Epoch 4/10
199/200 [============================>.] - ETA: 0s - loss: 4.1145 - acc: 0.0382Epoch 1/10
200/200 [==============================] - 21s 105ms/step - loss: 4.0977 - acc: 0.0403
200/200 [==============================] - 41s 207ms/step - loss: 4.1147 - acc: 0.0380 - val_loss: 4.0977 - val_acc: 0.0403
Epoch 5/10
199/200 [============================>.] - ETA: 0s - loss: 4.1097 - acc: 0.0345Epoch 1/10
200/200 [==============================] - 21s 106ms/step - loss: 4.1143 - acc: 0.0389
200/200 [==============================] - 41s 207ms/step - loss: 4.1096 - acc: 0.0347 - val_loss: 4.1143 - val_acc: 0.0389
Epoch 6/10
199/200 [============================>.] - ETA: 0s - loss: 4.0638 - acc: 0.0448Epoch 1/10
200/200 [==============================] - 21s 103ms/step - loss: 4.0430 - acc: 0.0420
200/200 [==============================] - 41s 204ms/step - loss: 4.0644 - acc: 0.0448 - val_loss: 4.0430 - val_acc: 0.0420
Epoch 7/10
199/200 [============================>.] - ETA: 0s - loss: 4.0609 - acc: 0.0382Epoch 1/10
200/200 [==============================] - 21s 104ms/step - loss: 4.0231 - acc: 0.0419
200/200 [==============================] - 41s 205ms/step - loss: 4.0602 - acc: 0.0383 - val_loss: 4.0231 - val_acc: 0.0419
Epoch 8/10
199/200 [============================>.] - ETA: 0s - loss: 4.0376 - acc: 0.0404Epoch 1/10
200/200 [==============================] - 21s 103ms/step - loss: 4.0160 - acc: 0.0397
200/200 [==============================] - 40s 202ms/step - loss: 4.0380 - acc: 0.0405 - val_loss: 4.0160 - val_acc: 0.0397
Epoch 9/10
199/200 [============================>.] - ETA: 0s - loss: 4.0292 - acc: 0.0442Epoch 1/10
200/200 [==============================] - 21s 104ms/step - loss: 4.0095 - acc: 0.0448
200/200 [==============================] - 41s 203ms/step - loss: 4.0293 - acc: 0.0445 - val_loss: 4.0095 - val_acc: 0.0448
Epoch 10/10
199/200 [============================>.] - ETA: 0s - loss: 4.0329 - acc: 0.0400Epoch 1/10
200/200 [==============================] - 20s 100ms/step - loss: 4.0271 - acc: 0.0481
200/200 [==============================] - 40s 198ms/step - loss: 4.0329 - acc: 0.0401 - val_loss: 4.0271 - val_acc: 0.0481
<tensorflow.python.keras.callbacks.History at 0x7f4960677908>
```

Model accuracy is still poor!!!

Lets use Transfer Learning

Download the vgg wieght file from here :
https://github.com/MinerKasch/applied_deep_learning/blob/master/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5

Use the below code to load VGG16 weights trained on ImageNet

```
# Instantiate the model with the pre-trained weights (no top)

base_model= tensorflow.keras.applications.VGG16(weights=('/content/gdrive/My Drive/Colab Notebooks/vgg16_weights_tf_dim_ordering_tf_k
                 include_top=False, pooling='avg', input_shape=(128, 128, 3))
```

Print the summary of the base_model

```
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 128, 128, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 128, 128, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 128, 128, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 64, 64, 64)        0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 64, 64, 128)       73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 64, 64, 128)       147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 32, 32, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 32, 32, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 32, 32, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 16, 16, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 16, 16, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 16, 16, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 8, 8, 512)         0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 8, 8, 512)         2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 4, 4, 512)         0
```

```
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)              0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

Add the following classification layers to the imported VGG Model

1. Flatten Layer
2. Dense layer with 1024 neurons with activation as Relu
3. Dense layer with 256 neurons with activation as Relu
4. Dense layer with 120 neurons with activation as Softmax

```
vgg16_model = base_model.output

# Flatten layer
vgg16_model = tensorflow.keras.layers.Flatten()(vgg16_model)
#vgg16_model = tensorflow.keras.layers.Dropout(0.4)(vgg16_model)

# Dense layer
vgg16_model = tensorflow.keras.layers.Dense(1024, activation="relu")(vgg16_model)
#vgg16_model = tensorflow.keras.layers.Dropout(0.4)(vgg16_model)

# Dense layer
vgg16_model = tensorflow.keras.layers.Dense(256, activation="relu")(vgg16_model)
#vgg16_model = tensorflow.keras.layers.Dropout(0.2)(vgg16_model)

# Output layer
vgg16_model = tensorflow.keras.layers.Dense(120, activation="softmax")(vgg16_model)
```

Make all the layers in the base_model (VGG16) to be non-trainable

```
# Freezing layers in the model which don't have 'dense' in their name

for layer in base_model.layers:
  layer.trainable = False
```

Fit and compile the model with batch_size = 128 and epochs = 10 and execute the model

Try to get training and validation accuracy to be more than 90%

```
# Final model
final_model = tensorflow.keras.models.Model(base_model.input, vgg16_model)
```

```
# Loss and Optimizer
final_model.compile(loss = "categorical_crossentropy", optimizer= 'adam', metrics=["accuracy"])

# Training the model
final_model = final_model.fit(X_train_data, Y_train_data,
                        batch_size=128,
                        epochs=10,
                        validation_data=(X_val_data, Y_val_data))
```

```
Train on 7155 samples, validate on 3067 samples
Epoch 1/10
7155/7155 [==============================] - 12s 2ms/sample - loss: 4.1399 - acc: 0.0477 - val_loss: 3.9298 - val_acc: 0.0587
Epoch 2/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 3.7734 - acc: 0.0929 - val_loss: 3.6938 - val_acc: 0.0962
Epoch 3/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 3.4932 - acc: 0.1377 - val_loss: 3.4686 - val_acc: 0.1366
Epoch 4/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 3.2681 - acc: 0.1716 - val_loss: 3.3532 - val_acc: 0.1477
Epoch 5/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 3.1003 - acc: 0.2049 - val_loss: 3.2757 - val_acc: 0.1617
Epoch 6/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 2.9938 - acc: 0.2254 - val_loss: 3.2011 - val_acc: 0.1718
Epoch 7/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 2.8725 - acc: 0.2454 - val_loss: 3.1794 - val_acc: 0.1787
Epoch 8/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 2.7741 - acc: 0.2692 - val_loss: 3.0957 - val_acc: 0.1898
Epoch 9/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 2.6822 - acc: 0.2969 - val_loss: 3.0820 - val_acc: 0.1969
Epoch 10/10
7155/7155 [==============================] - 10s 1ms/sample - loss: 2.5891 - acc: 0.3103 - val_loss: 3.0901 - val_acc: 0.2028
```