Refresh

Data used: flowers-image

Project Description

Main objective of this project was to learn and practice image processing in python. This project also includes ANN(Artificial Neural Network)Deep Learning model for predicting the type of flower from the input image.

Note: Generally we prefer using CNN(Convolutional neural Network) but we have used ANN for exploration and practice purpose.

We will be importing data from "https://upscfever.com/datasets/flowers-new.zip" for training and validation of the ANN model.

We will be utilising the following pyhton libraries: numpy, pandas, matplotlib, os, cv2, sklearn, tensorflow, keras, for data preprocessing and model training.

Preprocessing for image data includes resizing, conversion to numpy array, label encoding and one hot encoding(for target variable) and standardising for input variables. We have used functions like flatten() from keras, which returns a 1D array for the given image array (3D), which can be further used to train ANN model.

Note: Accuracy for model is low because main focus was on learning image processing.

Step_1 : Importing and unzipping data

```
!wget https://upscfever.com/datasets/flowers-new.zip
```

```
--2023-04-13 08:27:04--  https://upscfever.com/datasets/flowers-new.zip
Resolving upscfever.com (upscfever.com)... 104.21.90.10, 172.67.193.2, 2606:4700:3033::ac43:c102, ...
Connecting to upscfever.com (upscfever.com)|104.21.90.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: 'flowers-new.zip'

flowers-new.zip         [               <=>  ]   5.74M  51.4KB/s    in 40s

2023-04-13 08:27:46 (147 KB/s) - 'flowers-new.zip' saved [6021364]
```

```
!unzip flowers-new.zip
```

Step_2 : Importing libraries

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2           #cv2 is used for reading images
```

```
X = []
y = []
```

Step_2.1 : Reading jpg files

```
'''
reading jpg files to X, y
'''
def readingFiles(folderpath,foldername):

  paths=os.path.join(folderpath, foldername)

  for img in os.listdir(paths):#os.listdir - get names of files inside a folder given the path of folder
   #read first file name
```

```
   filepath = os.path.join(paths, img)
   #read image
   img_read = cv2.imread(filepath, cv2.IMREAD_COLOR)
   #resize image
   img_resized = cv2.resize(img_read, (256,256))
   #covert to numpy
   img_np = np.array(img_resized)
   #add to X
   X.append(img_np)
   #add foldername as label to y
   y.append(foldername)
```

```
folderpath= '/content/flowers'

flowername = os.listdir('/content/flowers')

for i in flowername:
 readingFiles(folderpath, i)
```

```
len(X)
```

```
117
```

```
len(y)
```

```
117
```

```
#converting X and y to numpy arrays

X = np.array(X)
y = np.array(y)
```

Step_3 : One Hot Encoding

Step_3.1 : LableEncoding and ohe of y

```
from sklearn.preprocessing import LabelEncoder #as y is having string values(name of types of flowers); in order to convert them into label ..we use this
from tensorflow.keras.utils import to_categorical #labels are then converted into classes using this code that is to_categorical()
```

```
#labeling flower names inside flower folder

enc = LabelEncoder()
y_le = enc.fit_transform(y)
```

```
y_le
```

```
array([4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
       4, 4, 4, 4, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0])
```

```
#ohe
y_ohe = to_categorical(y_le, num_classes=5)
```

Step_3.2 : Standardisation of X

```
X_scaled = X/255
```

Step_4 : Splittting data

```
from sklearn.model_selection import train_test_split
Xtrain, Xtest, ytrain, ytest = train_test_split(X_scaled, y_ohe, test_size=0.2, random_state=34, shuffle=True, stratify=y_ohe)

Xtrain, Xval, ytrain, yval = train_test_split(Xtrain, ytrain, test_size=0.2, random_state=34, shuffle=True, stratify=ytrain)
```

```
Xtrain.shape
```

```
(74, 256, 256, 3)
```

```
Xval.shape
```

```
(19, 256, 256, 3)
```

```
Xtest.shape
```

```
(24, 256, 256, 3)
```

Step_5 : ANN

```
'''
Note : Flatten is used to convert the image data into 1D
'''
from keras.models import Sequential
from keras.layers import Dense, Flatten

flowerANN = Sequential()

flowerANN.add(Flatten())
```

Step_5.1 : Adding layers

```
#256*256*3 - input dimensions(Xscaled shape)
flowerANN.add(Dense(units=1024, activation='relu'))
#hidden layer
flowerANN.add(Dense(units=350, activation='relu'))
#final layer - classification problem with 5 classes
flowerANN.add(Dense(units=5, activation='softmax'))
```

```
                            #step_5.2 : compile

flowerANN.compile(loss='categorical_crossentropy', metrics='accuracy', optimizer='adam')
```

step_6 : Saving best model

```
from keras.callbacks import ModelCheckpoint

mc = ModelCheckpoint(filepath='bestmodel.h5', monitor='val_accuracy', mode='max', verbose=1, save_best_only=True)
```

step_7 : Fit

```
history = flowerANN.fit(Xtrain, ytrain, epochs=50, validation_data=(Xval, yval), callbacks=[mc])
```

```
     Epoch 1/50
     3/3 [==============================] - ETA: 0s - loss: 150.5106 - accuracy: 0.2162
```

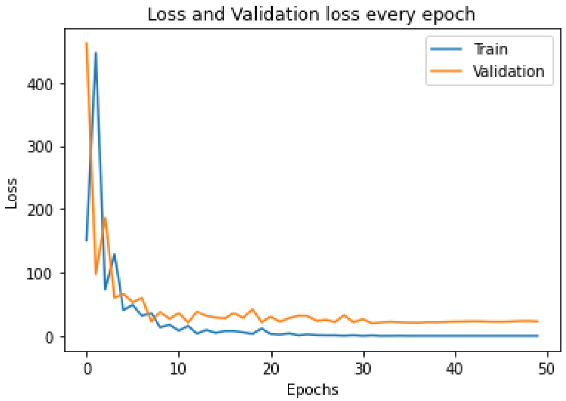
```
Epoch 1: val_accuracy improved from -inf to 0.21053, saving model to bestmodel.h5
3/3 [==============================] - 18s 6s/step - loss: 150.5106 - accuracy: 0.2162 - val_loss: 462.3159 - val_accuracy: 0.2105
Epoch 2/50
3/3 [==============================] - ETA: 0s - loss: 446.5936 - accuracy: 0.2027
Epoch 2: val_accuracy improved from 0.21053 to 0.26316, saving model to bestmodel.h5
3/3 [==============================] - 10s 5s/step - loss: 446.5936 - accuracy: 0.2027 - val_loss: 97.9030 - val_accuracy: 0.2632
Epoch 3/50
2/3 [===================>..........] - ETA: 0s - loss: 78.1556 - accuracy: 0.3281
Epoch 3: val_accuracy did not improve from 0.26316
3/3 [==============================] - 0s 70ms/step - loss: 73.1130 - accuracy: 0.3514 - val_loss: 186.1529 - val_accuracy: 0.2105
Epoch 4/50
2/3 [===================>..........] - ETA: 0s - loss: 137.4956 - accuracy: 0.2656
Epoch 4: val_accuracy did not improve from 0.26316
3/3 [==============================] - 0s 71ms/step - loss: 128.8701 - accuracy: 0.2568 - val_loss: 60.1853 - val_accuracy: 0.2632
Epoch 5/50
2/3 [===================>..........] - ETA: 0s - loss: 40.0468 - accuracy: 0.5000
Epoch 5: val_accuracy did not improve from 0.26316
3/3 [==============================] - 0s 65ms/step - loss: 40.7674 - accuracy: 0.4865 - val_loss: 66.0495 - val_accuracy: 0.2105
Epoch 6/50
2/3 [===================>..........] - ETA: 0s - loss: 54.2809 - accuracy: 0.4375
Epoch 6: val_accuracy improved from 0.26316 to 0.31579, saving model to bestmodel.h5
3/3 [==============================] - 11s 5s/step - loss: 49.1924 - accuracy: 0.4595 - val_loss: 53.3754 - val_accuracy: 0.3158
Epoch 7/50
3/3 [==============================] - ETA: 0s - loss: 31.3019 - accuracy: 0.4189
Epoch 7: val_accuracy did not improve from 0.31579
3/3 [==============================] - 0s 74ms/step - loss: 31.3019 - accuracy: 0.4189 - val_loss: 59.7075 - val_accuracy: 0.1579
Epoch 8/50
3/3 [==============================] - ETA: 0s - loss: 35.9949 - accuracy: 0.4459
Epoch 8: val_accuracy did not improve from 0.31579
3/3 [==============================] - 0s 74ms/step - loss: 35.9949 - accuracy: 0.4459 - val_loss: 22.5369 - val_accuracy: 0.3158
Epoch 9/50
2/3 [===================>..........] - ETA: 0s - loss: 12.1356 - accuracy: 0.6094
Epoch 9: val_accuracy did not improve from 0.31579
3/3 [==============================] - 0s 66ms/step - loss: 13.2798 - accuracy: 0.5541 - val_loss: 37.6412 - val_accuracy: 0.3158
Epoch 10/50
2/3 [===================>..........] - ETA: 0s - loss: 17.0514 - accuracy: 0.5469
Epoch 10: val_accuracy improved from 0.31579 to 0.36842, saving model to bestmodel.h5
3/3 [==============================] - 11s 5s/step - loss: 17.4615 - accuracy: 0.5000 - val_loss: 26.4732 - val_accuracy: 0.3684
Epoch 11/50
3/3 [==============================] - ETA: 0s - loss: 8.1444 - accuracy: 0.7297
Epoch 11: val_accuracy did not improve from 0.36842
3/3 [==============================] - 0s 76ms/step - loss: 8.1444 - accuracy: 0.7297 - val_loss: 35.9053 - val_accuracy: 0.3158
Epoch 12/50
2/3 [===================>..........] - ETA: 0s - loss: 16.1198 - accuracy: 0.5781
Epoch 12: val_accuracy did not improve from 0.36842
3/3 [==============================] - 0s 67ms/step - loss: 15.4932 - accuracy: 0.5541 - val_loss: 20.9533 - val_accuracy: 0.2632
Epoch 13/50
2/3 [===================>..........] - ETA: 0s - loss: 3.6370 - accuracy: 0.7969
Epoch 13: val_accuracy did not improve from 0.36842
3/3 [==============================] - 0s 66ms/step - loss: 3.4924 - accuracy: 0.8108 - val_loss: 37.9496 - val_accuracy: 0.2632
Epoch 14/50
2/3 [===================>..........] - ETA: 0s - loss: 9.4690 - accuracy: 0.7188
Epoch 14: val_accuracy did not improve from 0.36842
3/3 [==============================] - 0s 73ms/step - loss: 9.2354 - accuracy: 0.7297 - val_loss: 31.3726 - val_accuracy: 0.3684
Epoch 15/50
2/3 [===================>..........] - ETA: 0s - loss: 5.1810 - accuracy: 0.7500
```

```
#ANN architecture
flowerANN.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 flatten (Flatten)           (None, 196608)            0         
                                                                 
 dense (Dense)               (None, 1024)              201327616 
                                                                 
 dense_1 (Dense)             (None, 350)               358750    
                                                                 
 dense_2 (Dense)             (None, 5)                 1755      
                                                                 
=================================================================
Total params: 201,688,121
Trainable params: 201,688,121
Non-trainable params: 0
_________________________________________________________________
```

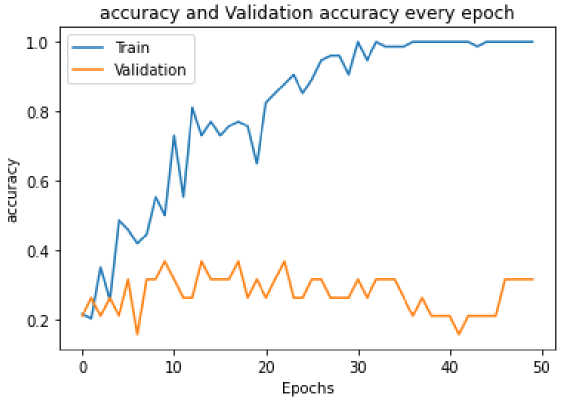
```
import matplotlib.pyplot as plt
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.title('Loss and Validation loss every epoch')
plt.show()
```

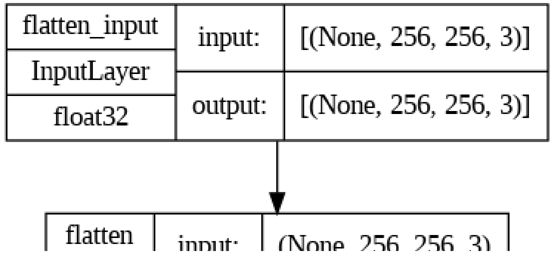



```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.xlabel('Epochs')
```

```
plt.ylabel('accuracy')
plt.legend(['Train', 'Validation'])
plt.title('accuracy and Validation accuracy every epoch')
plt.show()
```




```
from tensorflow.keras.utils import plot_model
plot_model(flowerANN, show_shapes=True, show_dtype=True, show_layer_activations=True, show_layer_names=True)
```




```
ypred =flowerANN.predict(Xtest)
```

```
1/1 [==============================] - 0s 87ms/step
```

```
import numpy as np
ypredclasses = np.argmax(ypred, axis=-1)
```

```
ypredclasses
```

```
array([1, 1, 2, 1, 0, 1, 1, 3, 1, 2, 4, 2, 0, 3, 1, 1, 3, 1, 2, 3, 1, 2,
       3, 2])
```

```
y_actual = enc.inverse_transform(ypredclasses)#inverse_transform helps to scale back the data to the original representation
```

```
y_actual
```

```
array(['dandelion', 'dandelion', 'rose', 'dandelion', 'daisy',
       'dandelion', 'dandelion', 'sunflower', 'dandelion', 'rose',
       'tulip', 'rose', 'daisy', 'sunflower', 'dandelion', 'dandelion',
       'sunflower', 'dandelion', 'rose', 'sunflower', 'dandelion', 'rose',
       'sunflower', 'rose'], dtype='<U9')
```

```
from keras.models import load_model
bestmodel = load_model('/content/bestmodel.h5')
```

Step_8: Evaluate test set on final model

```
flowerANN.evaluate(Xtest, ytest)
```

```
1/1 [==============================] - 0s 31ms/step - loss: 14.0639 - accuracy: 0.4583
[14.063925743103027, 0.4583333432674408]
```

Step_9: Evaluate on bestmodel saved on checkpoint

```
bestmodel.evaluate(Xtest, ytest)
```

```
1/1 [==============================] - 0s 127ms/step - loss: 14.7816 - accuracy: 0.5000
[14.781609535217285, 0.5]
```

Step_10: Calculation of avoidable bayes and variance

```
y_true  =        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1, 3, 1, 1, 4, 4, 4,
y_pred =       [3, 3, 3, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1,1,1,1,1,1,1,1,1,1,1,1,1,1,3,3,1,1,3,1,1,1,1,1,1,1,1,1,1,0,2,2,4,4,3,0,4,2,0,2,2,0,2,2,2,4,3,2,4,4,2,2,0,0,4,3, 3, 4, 3, 3, 3, 0, 3
y_self =       [0, 0, 0, 0, 0, 0, 1, 0, 3, 3, 0, 0, 0, 1,1,1,1,1,1,1,1,1,1,1,4,1,1,1,3,1,2,2,1,1,1,1,1,1,1,1,2,2,2,2,2,2,4,2,4,2,2,2,4,2,4,4,2,2,4,2,4,2,2,2,2,4,3, 3, 3, 3, 3, 3, 3, 3

#y_true: what actually the image is(0, 1, 2, 3, 4 represents category of flowers which has been label encoded)
#y_pred: after model training, how model predicted the image
#y_self: how we see as what flower it is
```

```
## calculation of Baye's accuracy

from sklearn.metrics import accuracy_score
accuracy_score(y_true,y_self)
```

```
0.6153846153846154
```

```
## training set acc -

bestmodel.evaluate(Xtrain, ytrain)
```

```
3/3 [==============================] - 0s 14ms/step - loss: 7.1420 - accuracy: 0.7568
[7.141990661621094, 0.7567567825317383]
```

```
## val set acc

bestmodel.evaluate(Xval, yval)
```

```
1/1 [==============================] - 0s 29ms/step - loss: 26.4732 - accuracy: 0.3684
[26.473234176635742, 0.3684210479259491]
```

```
## avoidable_bias = baye's accuracy - train_accuracy

bayes_accuracy  = 61.5
train_accuracy   =75.6
avoidable_bias =(61.5-75.6)
print(avoidable_bias)
```

```
-14.099999999999994
```

```
##    variance = Training_accuracy - validation_accuracy

Training_accuracy = 75.6
validation_accuracy = 36.84
variance = (75.6 - 36.84)
print(variance)
```

```
38.75999999999999
```

Predict for 1 image

```
#read first file name
filepath = '/content/flowers/tulip/11746080_963537acdc.jpg'#given here the path of image which we want to predict
#read image
img_read = cv2.imread(filepath, cv2.IMREAD_COLOR)
#resize image
img_resized = cv2.resize(img_read, (256, 256))
#covert to numpy
img_np = np.array(img_resized)
#add dimension
img_np_d = np.expand_dims(img_np, axis=0)
#shape
img_np_d.shape
#scale
img_np_d_scaled = img_np_d/255.0
```

```
bestmodel.predict(img_np_d_scaled)
```

```
1/1 [==============================] - 0s 55ms/step
array([[1.4507330e-06, 2.6123430e-06, 2.0094492e-22, 9.1677685e-23,
        9.9999595e-01]], dtype=float32)
```

```
import numpy as np
ypredclasses = np.argmax(bestmodel.predict(img_np_d_scaled), axis=-1)
```

```
1/1 [==============================] - 0s 15ms/step
```

```
ypredclasses
```

```
array([4])
```

```
y_actual = enc.inverse_transform(ypredclasses) #to check the class of above given array
```

```
y_actual
```

```
array(['tulip'], dtype='<U9')
```