ASSIGNMENT 3

Assignment Date	27-09-2022
Student Name	B.Sivabharathi
Student Roll.No	960519104079
Maximum Marks	2 Marks

```
import os
print(os.listdir('../input/flowers/flowers'))
# Ignore  the warnings
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')

# data visualisation and manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns

#configure
# sets matplotlib to inline and displays graphs below the corressponding cell.
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)

#model selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.metrics import
accuracy_score,precision_score,recall_score,confusion_matrix,roc_curve,roc_auc_score
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder

#preprocess.
from keras.preprocessing.image import ImageDataGenerator

#dl libraraies
from keras import backend as K
from keras.models import Sequential
from keras.layers import Dense
from tensorflow.keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
```

```
#from keras.optimizers import Adam,SGD,Adagrad,Adadelta,RMSprop
from tensorflow.keras.utils import to_categorical

# specifically for cnn
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization

import tensorflow as tf
import random as rn

# specifically for manipulating zipped images and getting numpy arrays of pixel values of images.
import cv2
import numpy as np
from tqdm import tqdm
import os
from random import shuffle
from zipfile import ZipFile
from PIL import Image

X=[]
Z=[]
IMG_SIZE=150
FLOWER_DAISY_DIR='../input/flowers/flowers/daisy'
FLOWER_SUNFLOWER_DIR='../input/flowers/flowers/sunflower'
FLOWER_TULIP_DIR='../input/flowers/flowers/tulip'
FLOWER_DANDI_DIR='../input/flowers/flowers/dandelion'
FLOWER_ROSE_DIR='../input/flowers/flowers/rose'

def assign_label(img,flower_type):
    return flower_type

def make_train_data(flower_type,DIR):
    for img in tqdm(os.listdir(DIR)):
        label=assign_label(img,flower_type)
        path = os.path.join(DIR,img)
        img = cv2.imread(path,cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))

        X.append(np.array(img))
        Z.append(str(label))

make_train_data('Daisy',FLOWER_DAISY_DIR)
print(len(X))

make_train_data('Sunflower',FLOWER_SUNFLOWER_DIR)
print(len(X))

```

```
make_train_data('Tulip',FLOWER_TULIP_DIR)
print(len(X))

make_train_data('Dandelion',FLOWER_DANDI_DIR)
print(len(X))

make_train_data('Rose',FLOWER_ROSE_DIR)
print(len(X))

fig,ax=plt.subplots(5,2)
fig.set_size_inches(15,15)
for i in range(5):
    for j in range (2):
        l=rn.randint(0,len(Z))
        ax[i,j].imshow(X[l])
        ax[i,j].set_title('Flower: '+Z[l])

            plt.tight_layout()

            le=LabelEncoder()
            Y=le.fit_transform(Z)
            Y=to_categorical(Y,5)
            X=np.array(X)
            X=X/255

            x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=42)

            np.random.seed(42)
            rn.seed(42)
            tf.set_random_seed(42)np.random.seed(42)
            rn.seed(42)
            tf.set_random_seed(42)

            # # modelling starts using a CNN.

            model = Sequential()
            model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation
    ='relu', input_shape = (150,150,3)))
            model.add(MaxPooling2D(pool_size=(2,2)))


            model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation
    ='relu'))
            model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))


            model.add(Conv2D(filters =96, kernel_size = (3,3),padding = 'Same',activation
    ='relu'))
```

```
128.        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
129.
130.        model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation
    ='relu'))
131.        model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))
132.
133.        model.add(Flatten())
134.        model.add(Dense(512))
135.        model.add(Activation('relu'))
136.        model.add(Dense(5, activation = "softmax"))
137.
138.        batch_size=128
139.        epochs=50
140.
141.        from keras.callbacks import ReduceLROnPlateau
142.        red_lr= ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.1)
143.
144.        datagen = ImageDataGenerator(
145.            featurewise_center=False,  # set input mean to 0 over the dataset
146.            samplewise_center=False,  # set each sample mean to 0
147.            featurewise_std_normalization=False,  # divide inputs by std of the dataset
148.            samplewise_std_normalization=False,  # divide each input by its std
149.            zca_whitening=False,  # apply ZCA whitening
150.            rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
151.            zoom_range = 0.1, # Randomly zoom image
152.            width_shift_range=0.2,  # randomly shift images horizontally (fraction of total
    width)
153.            height_shift_range=0.2,  # randomly shift images vertically (fraction of total
    height)
154.            horizontal_flip=True,  # randomly flip images
155.            vertical_flip=False)  # randomly flip images
156.
157.
158.        datagen.fit(x_train)
159.
160.        model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['
    accuracy'])
161.        model.summary()
162.
163.        History = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
164.                                epochs = epochs, validation_data = (x_test,y_test),
165.                                verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)
166.        # model.fit(x_train,y_train,epochs=epochs,batch_size=batch_size,validation_data =
    (x_test,y_test))
167.
168.        plt.plot(History.history['loss'])
169.        plt.plot(History.history['val_loss'])
170.        plt.title('Model Loss')
```

```
171.        plt.ylabel('Loss')
172.        plt.xlabel('Epochs')
173.        plt.legend(['train', 'test'])
174.        plt.show()
175.
176.        plt.plot(History.history['acc'])
177.        plt.plot(History.history['val_acc'])
178.        plt.title('Model Accuracy')
179.        plt.ylabel('Accuracy')
180.        plt.xlabel('Epochs')
181.        plt.legend(['train', 'test'])
182.        plt.show()
183.
184.        # getting predictions on val set.
185.        pred=model.predict(x_test)
186.        pred_digits=np.argmax(pred,axis=1)
187.
188.        # now storing some properly as well as misclassified indexes'.
189.        i=0
190.        prop_class=[]
191.        mis_class=[]
192.
193.        for i in range(len(y_test)):
194.            if(np.argmax(y_test[i])==pred_digits[i]):
195.                prop_class.append(i)
196.            if(len(prop_class)==8):
197.                break
198.
199.        i=0
200.        for i in range(len(y_test)):
201.            if(not np.argmax(y_test[i])==pred_digits[i]):
202.                mis_class.append(i)
203.            if(len(mis_class)==8):
204.                break
205.
206.        warnings.filterwarnings('always')
207.        warnings.filterwarnings('ignore')
208.
209.        count=0
210.        fig,ax=plt.subplots(4,2)
211.        fig.set_size_inches(15,15)
212.        for i in range (4):
213.            for j in range (2):
214.                ax[i,j].imshow(x_test[prop_class[count]])
215.                ax[i,j].set_title("Predicted Flower :
    "+str(le.inverse_transform([pred_digits[prop_class[count]]]))+"\n"+"Actual Flower :
    "+str(le.inverse_transform(np.argmax([y_test[prop_class[count]]]))))
216.                plt.tight_layout()
```

```
                count+=1

        warnings.filterwarnings('always')
        warnings.filterwarnings('ignore')

        count=0
        fig,ax=plt.subplots(4,2)
        fig.set_size_inches(15,15)
        for i in range (4):
            for j in range (2):
                ax[i,j].imshow(x_test[mis_class[count]])
                ax[i,j].set_title("Predicted Flower :
    "+str(le.inverse_transform([pred_digits[mis_class[count]]]))+"\n"+"Actual Flower :
    "+str(le.inverse_transform(np.argmax([y_test[mis_class[count]]]))))
                plt.tight_layout()
                count+=1
```