Team ID	PNT2022TMID43024
Project Name	Real time communication powered by Ai for specially Abled

ASSIGNMENT-3

Problem Statement: - Build CNN Model for Classification of Flowers

1) Importing Various Modules.

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

# specifically for cnn
```

```python
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
```

```
Using TensorFlow backend.
```

2) Preparing the Data

2.1) Making the functions to get the training and validation set from the Images

In [3]:

```python
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'
```

In [4]:

```python
def assign_label(img,flower_type):
    return flower_type
```

In [5]:

```python
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))
```

In [6]:

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

```
100%|██████████| 769/769 [00:03<00:00, 215.70it/s]
769
```

In [7]:

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

```
100%|██████████| 734/734 [00:03<00:00, 206.81it/s]
1503
```

In [8]:

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

```
100%|██████████| 984/984 [00:04<00:00, 224.01it/s]
2487
```

In [9]:

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

```
  9%|▉         | 97/1055 [00:00<00:04, 235.89it/s]
---------------------------------------------------------------------------
-
error                                     Traceback (most recent call
last)
<ipython-input-9-95c78ead0c98> in <module>
----> 1 make_train_data('Dandelion',FLOWER_DANDI_DIR)
      2 print(len(X))

<ipython-input-5-001b1f747236> in make_train_data(flower_type, DIR)
      4         path = os.path.join(DIR,img)
      5         img = cv2.imread(path,cv2.IMREAD_COLOR)
----> 6         img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))
      7 
      8         X.append(np.array(img))

error: OpenCV(3.4.3) /io/opencv/modules/imgproc/src/resize.cpp:4044:
error: (-215:Assertion failed) !ssize.empty() in function 'resize'
```

In [10]:

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

```
100%|██████████| 784/784 [00:03<00:00, 235.31it/s]
3386
```

2.2) Visualizing some Random Images

In [11]:

```
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()
```

2.3) Label Encoding the Y array (i.e. Daisy->0, Rose->1 etc...) & then One Hot Encoding

In [12]:

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

2.4) Splitting into Training and Validation Sets

In [13]:

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

2.5) Setting the Random Seeds

In [14]:

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

In [ ]:

3) Modelling

3.1) Building the ConvNet Model

In [15]:

```
# # 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'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Conv2D(filters = 96, kernel_size = (3,3),padding = 'Same',activation
='relu'))
model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(5, activation = "softmax"))
```

3.2) Using a LR Annealer

In [16]:

```
batch_size=128
epochs=50

from keras.callbacks import ReduceLROnPlateau
red_lr= ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.1)
```

3.3) Data Augmentation to prevent Overfitting

In [17]:

```
datagen = ImageDataGenerator(
        featurewise_center=False,  # set input mean to 0 over the dataset
        samplewise_center=False,  # set each sample mean to 0
        featurewise_std_normalization=False,  # divide inputs by std of the dataset
        samplewise_std_normalization=False,  # divide each input by its std
        zca_whitening=False,  # apply ZCA whitening
        rotation_range=10,  # randomly rotate images in the range (degrees, 0 to 180)
        zoom_range = 0.1, # Randomly zoom image
        width_shift_range=0.2,  # randomly shift images horizontally (fraction of
total width)
        height_shift_range=0.2,  # randomly shift images vertically (fraction of
total height)
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(x_train)
```

3.4) Compiling the Keras Model & Summary

In [18]:

```
model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accu
racy'])
```

In [19]:

```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
```

```
conv2d_1 (Conv2D)            (None, 150, 150, 32)      2432
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 37, 37, 96)        55392
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 18, 18, 96)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 18, 18, 96)        83040
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 9, 9, 96)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 7776)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3981824
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 2565
=================================================================
Total params: 4,143,749
Trainable params: 4,143,749
Non-trainable params: 0
_________________________________________________________________
```

3.5) Fitting on the Training set and making predcitons on the Validation set

In [20]:

```
History = model.fit_generator(datagen.flow(x_train,y_train, batch_size=batch_size),
                              epochs = epochs, validation_data = (x_test,y_test),
                              verbose = 1, steps_per_epoch=x_train.shape[0] // batch_size)
# model.fit(x_train,y_train,epochs=epochs,batch_size=batch_size,validation_data = (x_test,y_test))
```

```
Epoch 1/50
19/19 [==============================] - 18s 947ms/step - loss: 1.3674 - acc: 0.3721 - val_loss: 1.0551 - val_acc: 0.5549
Epoch 2/50
19/19 [==============================] - 15s 768ms/step - loss: 1.1096 - acc: 0.5398 - val_loss: 1.0287 - val_acc: 0.5738
Epoch 3/50
19/19 [==============================] - 15s 770ms/step - loss: 1.0306 - acc: 0.5749 - val_loss: 0.8975 - val_acc: 0.6647
```

```
Epoch 4/50
19/19 [==============================] - 15s 779ms/step - loss: 0.9500 -
acc: 0.6150 - val_loss: 1.0258 - val_acc: 0.5986
Epoch 5/50
19/19 [==============================] - 15s 771ms/step - loss: 0.9408 -
acc: 0.6352 - val_loss: 0.8534 - val_acc: 0.6659
Epoch 6/50
19/19 [==============================] - 15s 773ms/step - loss: 0.8849 -
acc: 0.6450 - val_loss: 0.8217 - val_acc: 0.6824
Epoch 7/50
19/19 [==============================] - 14s 757ms/step - loss: 0.8909 -
acc: 0.6493 - val_loss: 0.8131 - val_acc: 0.6860
Epoch 8/50
19/19 [==============================] - 15s 765ms/step - loss: 0.8273 -
acc: 0.6697 - val_loss: 0.7303 - val_acc: 0.7226
Epoch 9/50
19/19 [==============================] - 14s 758ms/step - loss: 0.8043 -
acc: 0.6879 - val_loss: 0.7364 - val_acc: 0.7131
Epoch 10/50
19/19 [==============================] - 14s 762ms/step - loss: 0.8012 -
acc: 0.6792 - val_loss: 0.6771 - val_acc: 0.7521
Epoch 11/50
19/19 [==============================] - 14s 761ms/step - loss: 0.8138 -
acc: 0.6789 - val_loss: 0.7210 - val_acc: 0.7107
Epoch 12/50
19/19 [==============================] - 15s 765ms/step - loss: 0.7827 -
acc: 0.6870 - val_loss: 0.7838 - val_acc: 0.6824
Epoch 13/50
19/19 [==============================] - 14s 761ms/step - loss: 0.8019 -
acc: 0.6843 - val_loss: 0.7249 - val_acc: 0.7226
Epoch 14/50
19/19 [==============================] - 15s 766ms/step - loss: 0.7274 -
acc: 0.7137 - val_loss: 0.6254 - val_acc: 0.7615
Epoch 15/50
19/19 [==============================] - 15s 764ms/step - loss: 0.6792 -
acc: 0.7381 - val_loss: 0.6146 - val_acc: 0.7686
Epoch 16/50
19/19 [==============================] - 14s 758ms/step - loss: 0.6489 -
acc: 0.7488 - val_loss: 0.6053 - val_acc: 0.7733
Epoch 17/50
19/19 [==============================] - 15s 765ms/step - loss: 0.6621 -
acc: 0.7463 - val_loss: 0.6521 - val_acc: 0.7509
Epoch 18/50
19/19 [==============================] - 15s 787ms/step - loss: 0.6664 -
acc: 0.7360 - val_loss: 0.6240 - val_acc: 0.7698
Epoch 19/50
19/19 [==============================] - 14s 757ms/step - loss: 0.6392 -
acc: 0.7450 - val_loss: 0.6863 - val_acc: 0.7308
```

```
Epoch 20/50
19/19 [==============================] - 15s 767ms/step - loss: 0.6411 -
acc: 0.7498 - val_loss: 0.6494 - val_acc: 0.7532
Epoch 21/50
19/19 [==============================] - 14s 759ms/step - loss: 0.6144 -
acc: 0.7669 - val_loss: 0.5729 - val_acc: 0.7887
Epoch 22/50
19/19 [==============================] - 14s 759ms/step - loss: 0.6019 -
acc: 0.7606 - val_loss: 0.6476 - val_acc: 0.7651
Epoch 23/50
19/19 [==============================] - 14s 756ms/step - loss: 0.5982 -
acc: 0.7624 - val_loss: 0.6235 - val_acc: 0.7651
Epoch 24/50
19/19 [==============================] - 14s 759ms/step - loss: 0.5960 -
acc: 0.7631 - val_loss: 0.6999 - val_acc: 0.7190
Epoch 25/50
19/19 [==============================] - 14s 757ms/step - loss: 0.6052 -
acc: 0.7646 - val_loss: 0.6245 - val_acc: 0.7780
Epoch 26/50
19/19 [==============================] - 14s 763ms/step - loss: 0.5379 -
acc: 0.7835 - val_loss: 0.5983 - val_acc: 0.7721
Epoch 27/50
19/19 [==============================] - 14s 751ms/step - loss: 0.5609 -
acc: 0.7821 - val_loss: 0.6182 - val_acc: 0.7615
Epoch 28/50
19/19 [==============================] - 14s 756ms/step - loss: 0.5226 -
acc: 0.7994 - val_loss: 0.5834 - val_acc: 0.7922
Epoch 29/50
19/19 [==============================] - 14s 751ms/step - loss: 0.5393 -
acc: 0.7847 - val_loss: 0.6063 - val_acc: 0.7674
Epoch 30/50
19/19 [==============================] - 14s 742ms/step - loss: 0.5416 -
acc: 0.7979 - val_loss: 0.5908 - val_acc: 0.7816
Epoch 31/50
19/19 [==============================] - 14s 739ms/step - loss: 0.5047 -
acc: 0.8101 - val_loss: 0.5826 - val_acc: 0.7851
Epoch 32/50
19/19 [==============================] - 14s 741ms/step - loss: 0.4784 -
acc: 0.8119 - val_loss: 0.6112 - val_acc: 0.7828
Epoch 33/50
19/19 [==============================] - 14s 745ms/step - loss: 0.4866 -
acc: 0.8068 - val_loss: 0.6614 - val_acc: 0.7509
Epoch 34/50
19/19 [==============================] - 14s 751ms/step - loss: 0.5058 -
acc: 0.8065 - val_loss: 0.5518 - val_acc: 0.7898
Epoch 35/50
19/19 [==============================] - 14s 741ms/step - loss: 0.4497 -
acc: 0.8321 - val_loss: 0.5333 - val_acc: 0.8158
```

```
Epoch 36/50
19/19 [==============================] - 14s 742ms/step - loss: 0.4184 -
acc: 0.8355 - val_loss: 0.5814 - val_acc: 0.7769
Epoch 37/50
19/19 [==============================] - 14s 737ms/step - loss: 0.3989 -
acc: 0.8450 - val_loss: 0.6347 - val_acc: 0.7887
Epoch 38/50
19/19 [==============================] - 14s 744ms/step - loss: 0.4255 -
acc: 0.8381 - val_loss: 0.6923 - val_acc: 0.7568
Epoch 39/50
19/19 [==============================] - 14s 762ms/step - loss: 0.4132 -
acc: 0.8389 - val_loss: 0.6282 - val_acc: 0.7887
Epoch 40/50
19/19 [==============================] - 14s 756ms/step - loss: 0.4231 -
acc: 0.8346 - val_loss: 0.5511 - val_acc: 0.8076
Epoch 41/50
19/19 [==============================] - 14s 747ms/step - loss: 0.4408 -
acc: 0.8277 - val_loss: 0.5862 - val_acc: 0.7887
Epoch 42/50
19/19 [==============================] - 14s 738ms/step - loss: 0.3850 -
acc: 0.8518 - val_loss: 0.6169 - val_acc: 0.7863
Epoch 43/50
19/19 [==============================] - 14s 736ms/step - loss: 0.3757 -
acc: 0.8571 - val_loss: 0.5783 - val_acc: 0.7839
Epoch 44/50
19/19 [==============================] - 14s 730ms/step - loss: 0.3467 -
acc: 0.8686 - val_loss: 0.6838 - val_acc: 0.7745
Epoch 45/50
19/19 [==============================] - 14s 735ms/step - loss: 0.3528 -
acc: 0.8588 - val_loss: 0.5599 - val_acc: 0.8052
Epoch 46/50
19/19 [==============================] - 14s 745ms/step - loss: 0.3390 -
acc: 0.8665 - val_loss: 0.7213 - val_acc: 0.7816
Epoch 47/50
19/19 [==============================] - 14s 737ms/step - loss: 0.3561 -
acc: 0.8725 - val_loss: 0.5799 - val_acc: 0.8017
Epoch 48/50
19/19 [==============================] - 14s 747ms/step - loss: 0.3701 -
acc: 0.8616 - val_loss: 0.6822 - val_acc: 0.7804
Epoch 49/50
19/19 [==============================] - 14s 740ms/step - loss: 0.3353 -
acc: 0.8736 - val_loss: 0.5710 - val_acc: 0.8135
Epoch 50/50
19/19 [==============================] - 14s 745ms/step - loss: 0.3410 -
acc: 0.8676 - val_loss: 0.6586 - val_acc: 0.7898
```

```
In [ ]:
```

4) Evaluating the Model Performance

In [21]:
```
plt.plot(History.history['loss'])
plt.plot(History.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```

In [22]:
```
plt.plot(History.history['acc'])
plt.plot(History.history['val_acc'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```

5) Visualizing Predictons on the Validation Set

In [23]:
```
# getting predictions on val set.
pred=model.predict(x_test)
pred_digits=np.argmax(pred,axis=1)
```

In [24]:
```
# now storing some properly as well as misclassified indexes'.
i=0
prop_class=[]
mis_class=[]

for i in range(len(y_test)):
    if(np.argmax(y_test[i])==pred_digits[i]):
        prop_class.append(i)
    if(len(prop_class)==8):
        break

i=0
for i in range(len(y_test)):
    if(not np.argmax(y_test[i])==pred_digits[i]):
        mis_class.append(i)
    if(len(mis_class)==8):
        break
```

CORRECTLY CLASSIFIED FLOWER IMAGES

In [25]:

```
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[prop_class[count]])
        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]]]))))
        plt.tight_layout()
        count+=1
```

MISCLASSIFIED IMAGES OF FLOWERS

In [26]:

```
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
```