Flowers Recognition

Introduction

We will be using the Flowers Recognition database.

This dataset contains 4242 images of flowers. The data collection is based on the data flicr, google images, yandex images.

The pictures are divided into five classes: chamomile, tulip, rose, sunflower, dandelion. For each class there are about 800 photos. Photos are not high resolution, about 320x240 pixels. Photos are not reduced to a single size, they have different proportions!

Objective

we can use this datastet to recognize plants from the photo usning **Convolutional Neural Networks**.

EDA

In [ ]:
```
class_names=['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

In [ ]:
```
# 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
from random import shuffle

#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.metrics import accuracy_score,confusion_matrix
from sklearn.preprocessing import LabelEncoder

#preprocess.
from tensorflow.keras.preprocessing.image import ImageDataGenerator

#dl libraraies
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.utils import to_categorical

# specifically for cnn
import tensorflow as tf
from tensorflow.keras.layers import Flatten,Activation
from tensorflow.keras.layers import Conv2D, MaxPooling2D
```

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

In [ ]:
```
def assign_label(img,flower_type):
    return flower_type
```

In [ ]:
```
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 [ ]:
```
# specifically for manipulating zipped images and getting numpy arrays of pixel valu
import cv2
from tqdm import tqdm
import os
```

In [ ]:
```
make_train_data('Daisy',FLOWER_DAISY_DIR)
print(len(X))
```

```
100%|██████████| 764/764 [00:21<00:00, 35.66it/s]
764
```

In [ ]:
```
make_train_data('Sunflower',FLOWER_SUNFLOWER_DIR)
print(len(X))
```

```
100%|██████████| 733/733 [00:13<00:00, 52.77it/s]
1497
```

In [ ]:
```
make_train_data('Tulip',FLOWER_TULIP_DIR)
print(len(X))
```

```
100%|██████████| 984/984 [00:16<00:00, 59.65it/s]
```

```
2481
```

In [ ]:
```
make_train_data('Dandelion',FLOWER_DANDI_DIR)
print(len(X))
```

```
100%|██████████| 1052/1052 [00:26<00:00, 39.69it/s]
3533
```

In [ ]:
```
make_train_data('Rose',FLOWER_ROSE_DIR)
print(len(X))
```

```
100%|██████████| 784/784 [00:04<00:00, 185.63it/s]
4317
```

In [ ]:
```
def display_random_image(class_names, images, labels):
    """
        Display a random image from the images array and its correspond label from t
    """

    index = np.random.randint(len(labels))
    plt.figure()
    plt.imshow(images[index])
    plt.xticks([])
    plt.yticks([])
    plt.grid(True)
    plt.title('Image #{} : '.format(index) + labels[index])
    plt.show()
```

In [ ]:
```
display_random_image(class_names, X, Z)
```

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

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

In [ ]:
```
x_train.shape
```

Out[ ]:
```
(3237, 150, 150, 3)
```

In [ ]:
```
np.random.seed(42)
```

Modelling using CNN.

In [ ]:
```
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation ='rel
model.add(MaxPooling2D(pool_size=(2,2)))


model.add(Conv2D(filters = 64, kernel_size = (3,3),padding = 'Same',activation ='rel
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 ='rel
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"))
```

In [ ]:
```
batch_size=128
epochs=50

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

In [ ]:
```
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 to
        height_shift_range=0.2,  # randomly shift images vertically (fraction of tot
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images


datagen.fit(x_train)
```

In [ ]:
```
model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['acc
```

In [ ]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 150, 150, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 37, 37, 96)        55392
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 18, 18, 96)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 18, 18, 96)        83040
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 9, 9, 96)          0
_________________________________________________________________
flatten (Flatten)            (None, 7776)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3981824
_________________________________________________________________
activation (Activation)      (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 5)                 2565
=================================================================
Total params: 4,143,749
Trainable params: 4,143,749
Non-trainable params: 0
_________________________________________________________________
```

In [ ]:

```
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
```

```
Epoch 1/50
25/25 [==============================] - 62s 1s/step - loss: 1.4996 - accuracy: 0.33
77 - val_loss: 1.1777 - val_accuracy: 0.5176
Epoch 2/50
25/25 [==============================] - 20s 784ms/step - loss: 1.2336 - accuracy:
0.4693 - val_loss: 1.0705 - val_accuracy: 0.5880
Epoch 3/50
25/25 [==============================] - 19s 761ms/step - loss: 1.1225 - accuracy:
0.5429 - val_loss: 1.0079 - val_accuracy: 0.5824
Epoch 4/50
25/25 [==============================] - 19s 738ms/step - loss: 1.0591 - accuracy:
0.5764 - val_loss: 1.0275 - val_accuracy: 0.5769
Epoch 5/50
25/25 [==============================] - 22s 852ms/step - loss: 0.9921 - accuracy:
0.6092 - val_loss: 0.9213 - val_accuracy: 0.6389
Epoch 6/50
25/25 [==============================] - 19s 777ms/step - loss: 0.9332 - accuracy:
0.6375 - val_loss: 0.8440 - val_accuracy: 0.6639
Epoch 7/50
25/25 [==============================] - 20s 784ms/step - loss: 0.8963 - accuracy:
0.6497 - val_loss: 0.8979 - val_accuracy: 0.6509
Epoch 8/50
25/25 [==============================] - 20s 777ms/step - loss: 0.9035 - accuracy:
0.6571 - val_loss: 0.8089 - val_accuracy: 0.6861
Epoch 9/50
25/25 [==============================] - 19s 760ms/step - loss: 0.8623 - accuracy:
```

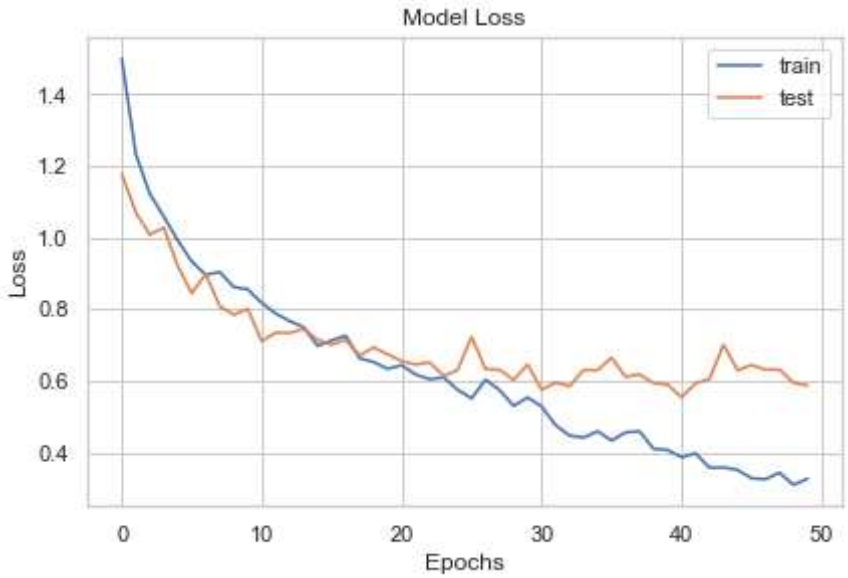
```
0.6748 - val_loss: 0.7841 - val_accuracy: 0.6954
Epoch 10/50
25/25 [==============================] - 21s 817ms/step - loss: 0.8556 - accuracy:
0.6652 - val_loss: 0.8014 - val_accuracy: 0.6722
Epoch 11/50
25/25 [==============================] - 20s 804ms/step - loss: 0.8181 - accuracy:
0.6850 - val_loss: 0.7105 - val_accuracy: 0.7222
Epoch 12/50
25/25 [==============================] - 19s 732ms/step - loss: 0.7881 - accuracy:
0.7002 - val_loss: 0.7345 - val_accuracy: 0.7120
Epoch 13/50
25/25 [==============================] - 18s 723ms/step - loss: 0.7669 - accuracy:
0.7134 - val_loss: 0.7337 - val_accuracy: 0.7185
Epoch 14/50
25/25 [==============================] - 20s 789ms/step - loss: 0.7495 - accuracy:
0.7186 - val_loss: 0.7451 - val_accuracy: 0.7157
Epoch 15/50
25/25 [==============================] - 19s 752ms/step - loss: 0.6975 - accuracy:
0.7414 - val_loss: 0.7143 - val_accuracy: 0.7204
Epoch 16/50
25/25 [==============================] - 19s 740ms/step - loss: 0.7130 - accuracy:
0.7343 - val_loss: 0.7011 - val_accuracy: 0.7167
Epoch 17/50
25/25 [==============================] - 19s 733ms/step - loss: 0.7262 - accuracy:
0.7289 - val_loss: 0.7147 - val_accuracy: 0.7278
Epoch 18/50
25/25 [==============================] - 21s 831ms/step - loss: 0.6629 - accuracy:
0.7444 - val_loss: 0.6708 - val_accuracy: 0.7407
Epoch 19/50
25/25 [==============================] - 21s 808ms/step - loss: 0.6527 - accuracy:
0.7491 - val_loss: 0.6936 - val_accuracy: 0.7370
Epoch 20/50
25/25 [==============================] - 21s 817ms/step - loss: 0.6338 - accuracy:
0.7655 - val_loss: 0.6740 - val_accuracy: 0.7389
Epoch 21/50
25/25 [==============================] - 35s 1s/step - loss: 0.6439 - accuracy: 0.76
23 - val_loss: 0.6551 - val_accuracy: 0.7481
Epoch 22/50
25/25 [==============================] - 21s 811ms/step - loss: 0.6180 - accuracy:
0.7642 - val_loss: 0.6456 - val_accuracy: 0.7620
Epoch 23/50
25/25 [==============================] - 18s 723ms/step - loss: 0.6047 - accuracy:
0.7694 - val_loss: 0.6521 - val_accuracy: 0.7454
Epoch 24/50
25/25 [==============================] - 19s 735ms/step - loss: 0.6096 - accuracy:
0.7629 - val_loss: 0.6138 - val_accuracy: 0.7667
Epoch 25/50
25/25 [==============================] - 21s 809ms/step - loss: 0.5752 - accuracy:
0.7765 - val_loss: 0.6300 - val_accuracy: 0.7528
Epoch 26/50
25/25 [==============================] - 21s 842ms/step - loss: 0.5512 - accuracy:
0.7974 - val_loss: 0.7228 - val_accuracy: 0.7231
Epoch 27/50
25/25 [==============================] - 21s 847ms/step - loss: 0.6035 - accuracy:
0.7626 - val_loss: 0.6327 - val_accuracy: 0.7583
Epoch 28/50
25/25 [==============================] - 21s 817ms/step - loss: 0.5740 - accuracy:
0.7806 - val_loss: 0.6308 - val_accuracy: 0.7491
Epoch 29/50
25/25 [==============================] - 21s 819ms/step - loss: 0.5303 - accuracy:
0.7999 - val_loss: 0.6022 - val_accuracy: 0.7630
Epoch 30/50
25/25 [==============================] - 21s 830ms/step - loss: 0.5537 - accuracy:
0.7858 - val_loss: 0.6458 - val_accuracy: 0.7546
```

```
Epoch 31/50
25/25 [==============================] - 21s 840ms/step - loss: 0.5285 - accuracy:
0.8077 - val_loss: 0.5754 - val_accuracy: 0.7944
Epoch 32/50
25/25 [==============================] - 20s 790ms/step - loss: 0.4765 - accuracy:
0.8327 - val_loss: 0.5954 - val_accuracy: 0.7889
Epoch 33/50
25/25 [==============================] - 21s 835ms/step - loss: 0.4475 - accuracy:
0.8340 - val_loss: 0.5851 - val_accuracy: 0.7861
Epoch 34/50
25/25 [==============================] - 21s 815ms/step - loss: 0.4416 - accuracy:
0.8382 - val_loss: 0.6304 - val_accuracy: 0.7722
Epoch 35/50
25/25 [==============================] - 21s 820ms/step - loss: 0.4592 - accuracy:
0.8241 - val_loss: 0.6300 - val_accuracy: 0.7787
Epoch 36/50
25/25 [==============================] - 21s 839ms/step - loss: 0.4333 - accuracy:
0.8434 - val_loss: 0.6643 - val_accuracy: 0.7630
Epoch 37/50
25/25 [==============================] - 21s 825ms/step - loss: 0.4562 - accuracy:
0.8260 - val_loss: 0.6115 - val_accuracy: 0.7806
Epoch 38/50
25/25 [==============================] - 20s 778ms/step - loss: 0.4592 - accuracy:
0.8308 - val_loss: 0.6183 - val_accuracy: 0.7731
Epoch 39/50
25/25 [==============================] - 20s 798ms/step - loss: 0.4106 - accuracy:
0.8479 - val_loss: 0.5936 - val_accuracy: 0.7870
Epoch 40/50
25/25 [==============================] - 20s 772ms/step - loss: 0.4074 - accuracy:
0.8479 - val_loss: 0.5896 - val_accuracy: 0.7926
Epoch 41/50
25/25 [==============================] - 20s 790ms/step - loss: 0.3868 - accuracy:
0.8530 - val_loss: 0.5540 - val_accuracy: 0.7981
Epoch 42/50
25/25 [==============================] - 19s 735ms/step - loss: 0.3977 - accuracy:
0.8488 - val_loss: 0.5932 - val_accuracy: 0.7898
Epoch 43/50
25/25 [==============================] - 19s 731ms/step - loss: 0.3577 - accuracy:
0.8614 - val_loss: 0.6054 - val_accuracy: 0.7833
Epoch 44/50
25/25 [==============================] - 19s 754ms/step - loss: 0.3586 - accuracy:
0.8678 - val_loss: 0.7009 - val_accuracy: 0.7648
Epoch 45/50
25/25 [==============================] - 20s 789ms/step - loss: 0.3516 - accuracy:
0.8688 - val_loss: 0.6295 - val_accuracy: 0.7639
Epoch 46/50
25/25 [==============================] - 19s 733ms/step - loss: 0.3280 - accuracy:
0.8758 - val_loss: 0.6442 - val_accuracy: 0.7806
Epoch 47/50
25/25 [==============================] - 20s 809ms/step - loss: 0.3258 - accuracy:
0.8797 - val_loss: 0.6315 - val_accuracy: 0.7815
Epoch 48/50
25/25 [==============================] - 21s 809ms/step - loss: 0.3437 - accuracy:
0.8668 - val_loss: 0.6316 - val_accuracy: 0.7880
Epoch 49/50
25/25 [==============================] - 19s 743ms/step - loss: 0.3098 - accuracy:
0.8810 - val_loss: 0.5948 - val_accuracy: 0.8000
Epoch 50/50
25/25 [==============================] - 21s 813ms/step - loss: 0.3268 - accuracy:
0.8716 - val_loss: 0.5869 - val_accuracy: 0.8046
```

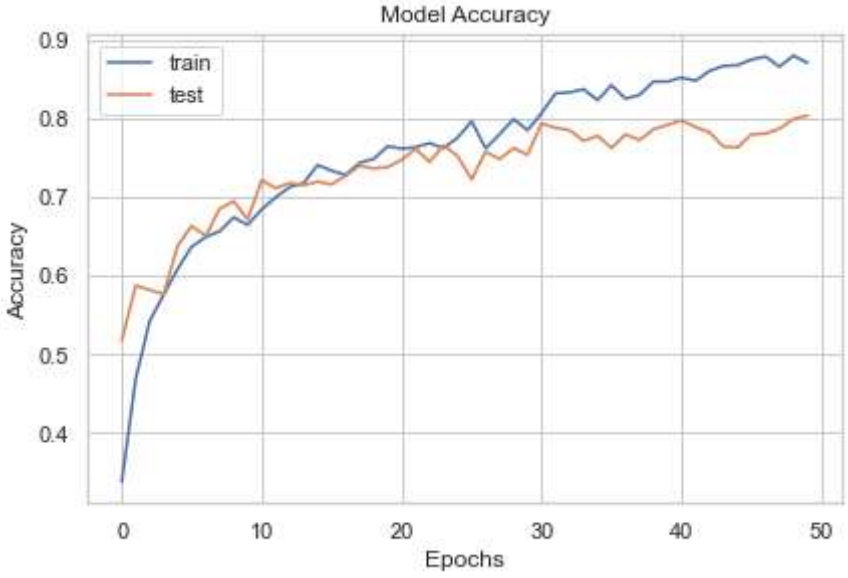
In [ ]:

```
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 [ ]:
```
plt.plot(History.history['accuracy'])
plt.plot(History.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```



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

In [ ]:
```
test_digits = np.argmax(y_test,axis=1)
```

In [ ]:
```
print("Accuracy : {}".format(accuracy_score(test_digits, pred_digits)))
```

```
Accuracy : 0.8046296296296296
```

In [ ]:
```
pred_labels = le.inverse_transform(pred_digits)
```

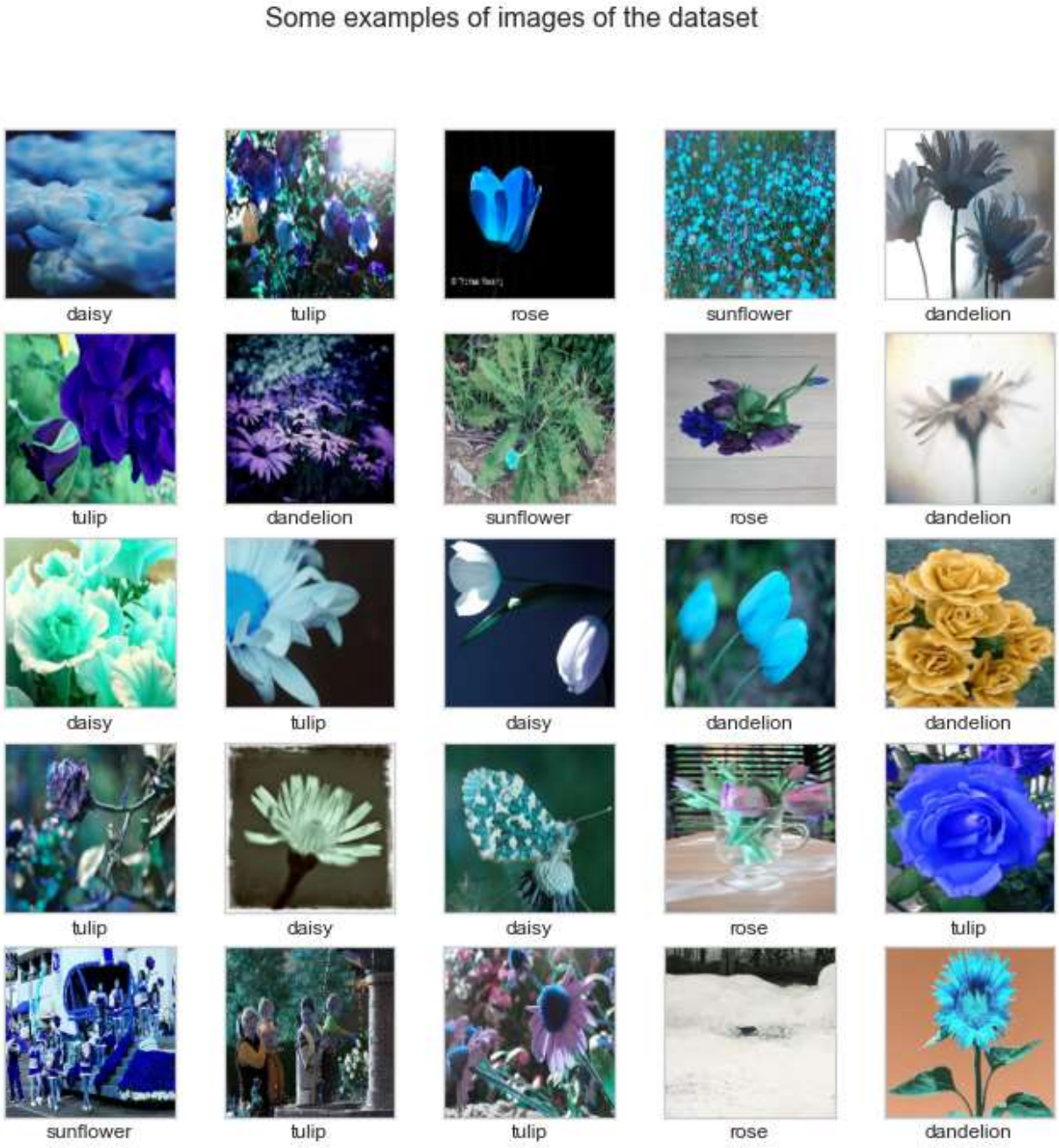
In [ ]:
```
display_random_image(class_names, x_test, pred_labels)
```

Error analysis

In [ ]:
```
def display_examples(class_names, images, labels):
    """
        Display 25 images from the images array with its corresponding labels
    """

    fig = plt.figure(figsize=(10,10))
    fig.suptitle("Some examples of images of the dataset", fontsize=16)
    for i in range(25):
        plt.subplot(5,5,i+1)
        plt.xticks([])
        plt.yticks([])
        plt.grid(False)
        plt.imshow(images[i], cmap=plt.cm.binary)
        plt.xlabel(class_names[labels[i]])
    plt.show()
```

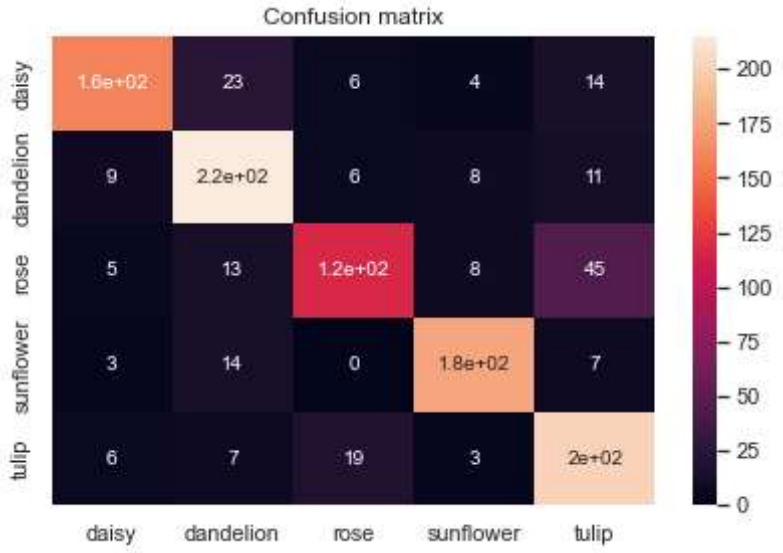
In [ ]:
```
def print_mislabeled_images(class_names, test_images, test_labels, pred_labels):
    """
        Print 25 examples of mislabeled images by the classifier, e.g when test_labe
    """
    BOO = (test_labels == pred_labels)
    mislabeled_indices = np.where(BOO == 0)
    mislabeled_images = test_images[mislabeled_indices]
    mislabeled_labels = pred_labels[mislabeled_indices]

    title = "Some examples of mislabeled images by the classifier:"
    display_examples(class_names,  mislabeled_images, mislabeled_labels)
```

In [ ]:
```
print_mislabeled_images(class_names, x_test, test_digits, pred_digits)
```



In [ ]:

```
CM = confusion_matrix(test_digits, pred_digits)
ax = plt.axes()
sns.heatmap(CM, annot=True,
            annot_kws={"size": 10},
            xticklabels=class_names,
            yticklabels=class_names, ax = ax)
ax.set_title('Confusion matrix')
plt.show()
```



Conclusion: The classifier has trouble with 1 kind of images.

It has trouble with `rose and tulip` . it can detects dandelion very accurately!

Possible flaws and next steps

We have overfitting problem and we can face it by :

1. using more data
2. regularization(L2,dropout,Data_augmentation)
3. find better neural network Architecture

`or` we can stop at 20 epochs as shown in the model complexity curve