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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
import os
print(os.listdir('/content/drive/MyDrive/flowers'))
```

```
['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

```
import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import style
import seaborn as sns
%matplotlib inline
style.use('fivethirtyeight')
sns.set(style='whitegrid',color_codes=True)
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
from keras.preprocessing.image import ImageDataGenerator
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
from keras.layers import Dropout, Flatten,Activation
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization
import tensorflow as tf
import random as rn
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='/content/drive/MyDrive/flowers/daisy'
FLOWER_SUNFLOWER_DIR='/content/drive/MyDrive/flowers/sunflower'
FLOWER_TULIP_DIR='/content/drive/MyDrive/flowers/tulip'
FLOWER_DANDI_DIR='/content/drive/MyDrive/flowers/dandelion'
FLOWER_ROSE_DIR='/content/drive/MyDrive/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))
```

```
100%|██████████| 764/764 [00:09<00:00, 78.18it/s] 764
```

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

```
100%|██████████| 733/733 [00:10<00:00, 68.86it/s] 1497
```

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

```
100%|██████████| 984/984 [00:14<00:00, 68.61it/s] 2481
```

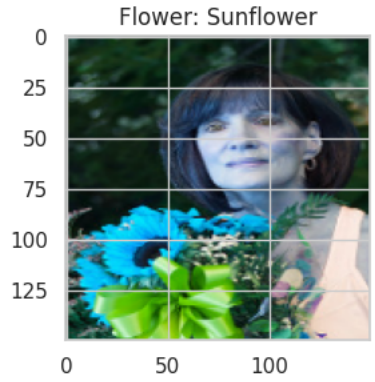
```
make_train_data('Dandelion',FLOWER_DANDI_DIR)
print(len(X))
```

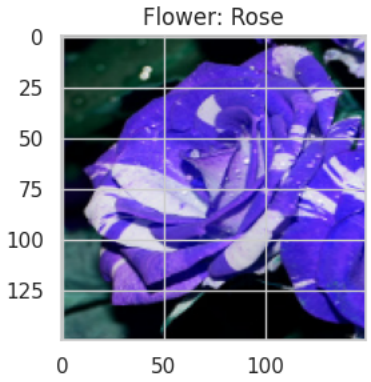
```
100%|██████████| 1052/1052 [00:15<00:00, 66.98it/s] 3533
```

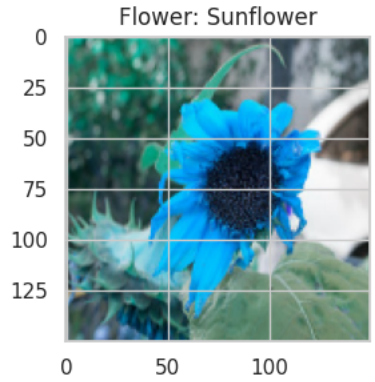
```
make_train_data('Rose',FLOWER_ROSE_DIR)
print(len(X))
```

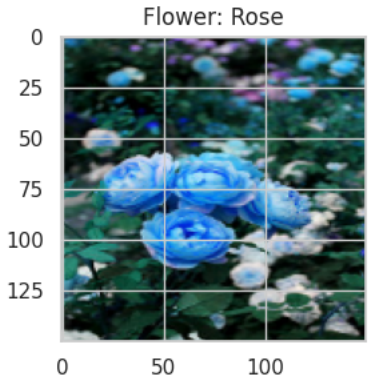
```
100%|██████████| 784/784 [00:12<00:00, 65.04it/s]
4317
```

```
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)
```

```
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"))
```

```
batch_size=128
epochs=10
from keras.callbacks import ReduceLROnPlateau
red_lr= ReduceLROnPlateau(monitor='val_acc',patience=3,verbose=1,factor=0.1)
```

```
datagen = ImageDataGenerator(
        featurewise_center=False,
        samplewise_center=False,
        featurewise_std_normalization=False,
        samplewise_std_normalization=False,
        zca_whitening=False,
        rotation_range=10,
        zoom_range = 0.1,
        width_shift_range=0.2,
        height_shift_range=0.2,
        horizontal_flip=True,
        vertical_flip=False)
datagen.fit(x_train)
```

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

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizer
```

```
model.summary()
```

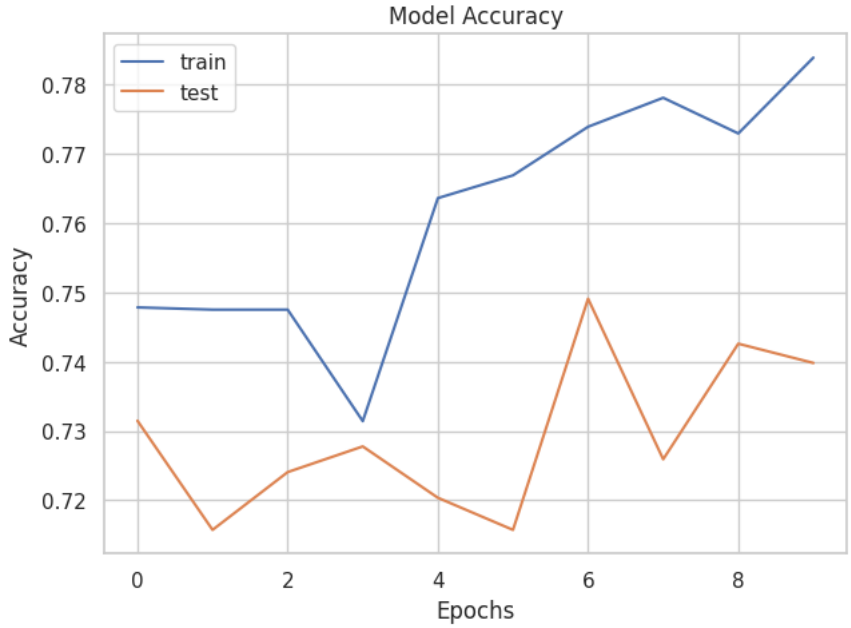
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 150, 150, 32)      2432

 max_pooling2d (MaxPooling2  (None, 75, 75, 32)        0
 D)

 conv2d_1 (Conv2D)           (None, 75, 75, 64)        18496

 max_pooling2d_1 (MaxPoolin  (None, 37, 37, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 37, 37, 96)        55392

 max_pooling2d_2 (MaxPoolin  (None, 18, 18, 96)        0
 g2D)

 conv2d_3 (Conv2D)           (None, 18, 18, 96)        83040

 max_pooling2d_3 (MaxPoolin  (None, 9, 9, 96)          0
 g2D)

 flatten (Flatten)           (None, 7776)              0

 dense (Dense)               (None, 512)               3981824

 activation (Activation)     (None, 512)               0

 dense_1 (Dense)             (None, 5)                 2565

=================================================================
Total params: 4143749 (15.81 MB)
Trainable params: 4143749 (15.81 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
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)
```

```
Epoch 1/10
25/25 [==============================] - 241s 9s/step - loss: 0.6815 - accuracy: 0.7478 - val_loss: 0.7172 - val_accuracy: 0.7315
Epoch 2/10
25/25 [==============================] - 220s 9s/step - loss: 0.6630 - accuracy: 0.7475 - val_loss: 0.7383 - val_accuracy: 0.7157
Epoch 3/10
25/25 [==============================] - 220s 9s/step - loss: 0.6625 - accuracy: 0.7475 - val_loss: 0.7092 - val_accuracy: 0.7241
Epoch 4/10
25/25 [==============================] - 221s 9s/step - loss: 0.6869 - accuracy: 0.7314 - val_loss: 0.6898 - val_accuracy: 0.7278
Epoch 5/10
25/25 [==============================] - 222s 9s/step - loss: 0.6480 - accuracy: 0.7636 - val_loss: 0.7236 - val_accuracy: 0.7204
Epoch 6/10
25/25 [==============================] - 223s 9s/step - loss: 0.6191 - accuracy: 0.7669 - val_loss: 0.7595 - val_accuracy: 0.7157
Epoch 7/10
25/25 [==============================] - 219s 9s/step - loss: 0.5967 - accuracy: 0.7739 - val_loss: 0.6431 - val_accuracy: 0.7491
Epoch 8/10
25/25 [==============================] - 219s 9s/step - loss: 0.5893 - accuracy: 0.7781 - val_loss: 0.6565 - val_accuracy: 0.7259
Epoch 9/10
25/25 [==============================] - 219s 9s/step - loss: 0.5977 - accuracy: 0.7729 - val_loss: 0.6593 - val_accuracy: 0.7426
Epoch 10/10
25/25 [==============================] - 220s 9s/step - loss: 0.5602 - accuracy: 0.7839 - val_loss: 0.6630 - val_accuracy: 0.7398
```

```
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()
```


```
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()
```

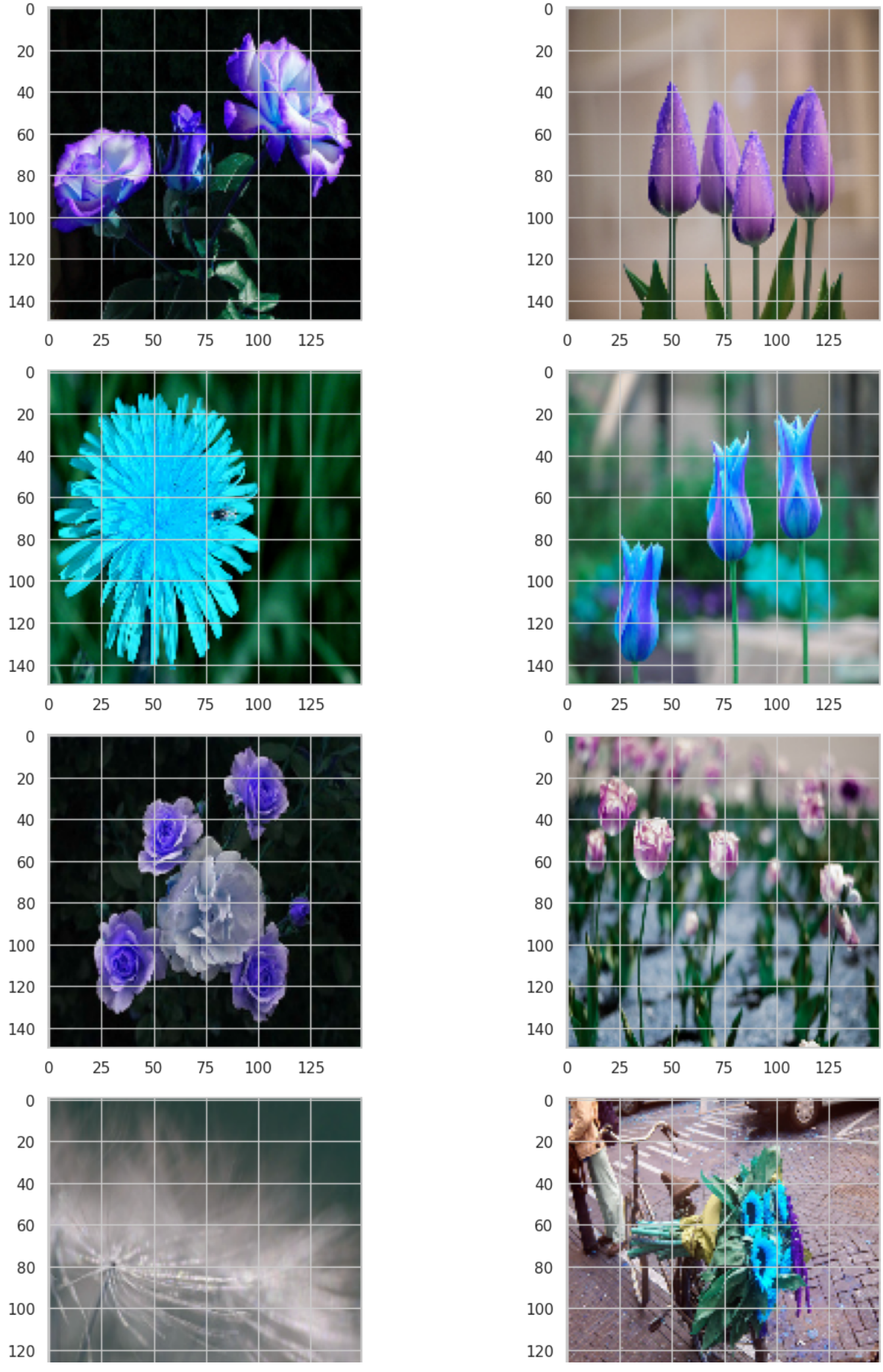



```
pred=model.predict(x_test)
pred_digits=np.argmax(pred,axis=1)
```

```
     34/34 [==============================] - 18s 522ms/step
```

```
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
```

```
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]])
        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]])
```

```
plt.tight_layout()
count+=1
```