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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
```

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

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

```
# 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_
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
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 ima
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%|██████████| 769/769 [08:49<00:00,  1.45it/s]769
```

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

```
100%|██████████| 734/734 [08:28<00:00,  1.44it/s]1503
```

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

```
100%|██████████| 984/984 [11:05<00:00,  1.48it/s]2487
```

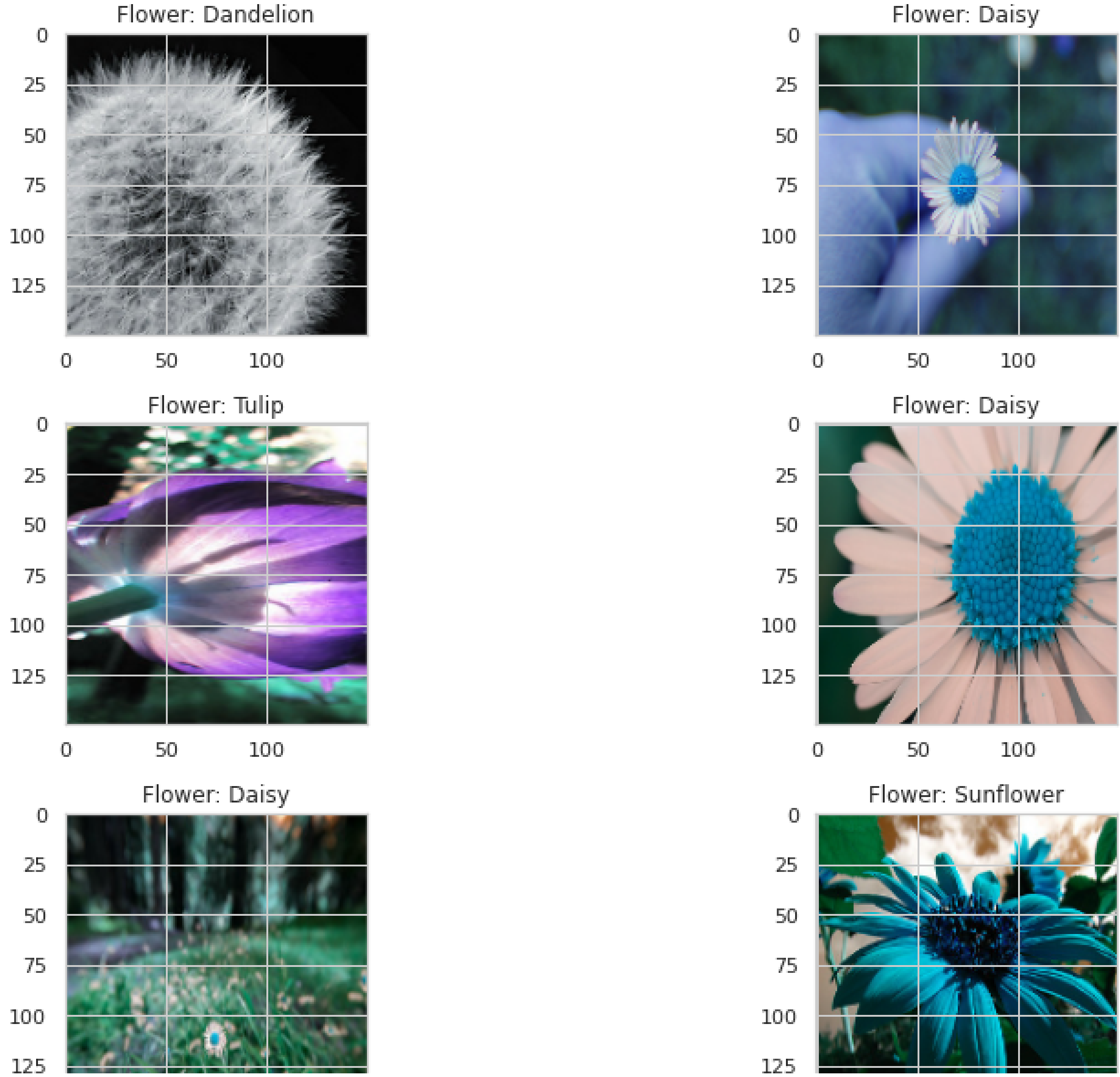
```
make_train_data('Dandelion',FLOWER_DANDI_DIR)
print(len(X))
```

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

```
 93%|█████████▎| 727/784 [08:13<00:36,  1.57it/s]
 93%|█████████▎| 728/784 [08:14<00:33,  1.66it/s]
 93%|█████████▎| 729/784 [08:15<00:39,  1.38it/s]
 93%|█████████▎| 730/784 [08:15<00:36,  1.47it/s]
 93%|█████████▎| 731/784 [08:16<00:33,  1.57it/s]
 93%|█████████▎| 732/784 [08:16<00:31,  1.64it/s]
 93%|█████████▎| 733/784 [08:17<00:30,  1.66it/s]

 94%|█████████▎| 734/784 [08:17<00:29,  1.67it/s]
 94%|█████████▍| 735/784 [08:18<00:30,  1.61it/s]
 94%|█████████▍| 736/784 [08:19<00:29,  1.64it/s]
 94%|█████████▍| 737/784 [08:19<00:27,  1.70it/s]
 94%|█████████▍| 738/784 [08:20<00:26,  1.73it/s]
 94%|█████████▍| 739/784 [08:20<00:25,  1.74it/s]
 94%|█████████▍| 740/784 [08:21<00:25,  1.71it/s]
 95%|█████████▍| 741/784 [08:22<00:25,  1.70it/s]
 95%|█████████▍| 742/784 [08:23<00:30,  1.39it/s]
 95%|█████████▍| 743/784 [08:23<00:27,  1.49it/s]
 95%|█████████▍| 744/784 [08:24<00:25,  1.57it/s]
 95%|█████████▌| 745/784 [08:24<00:24,  1.62it/s]
 95%|█████████▌| 746/784 [08:25<00:23,  1.62it/s]
 95%|█████████▌| 747/784 [08:25<00:22,  1.68it/s]
 95%|█████████▌| 748/784 [08:26<00:20,  1.75it/s]
 96%|█████████▌| 749/784 [08:26<00:19,  1.82it/s]
 96%|█████████▌| 750/784 [08:27<00:18,  1.79it/s]
 96%|█████████▌| 751/784 [08:27<00:17,  1.87it/s]
 96%|█████████▌| 752/784 [08:28<00:18,  1.76it/s]
 96%|█████████▌| 753/784 [08:29<00:18,  1.64it/s]
 96%|█████████▌| 754/784 [08:29<00:17,  1.69it/s]
 96%|█████████▋| 755/784 [08:30<00:16,  1.75it/s]
 96%|█████████▋| 756/784 [08:30<00:15,  1.79it/s]
 97%|█████████▋| 757/784 [08:31<00:15,  1.73it/s]
 97%|█████████▋| 758/784 [08:32<00:14,  1.81it/s]
 97%|█████████▋| 759/784 [08:32<00:14,  1.76it/s]
 97%|█████████▋| 760/784 [08:33<00:13,  1.71it/s]
 97%|█████████▋| 761/784 [08:33<00:13,  1.68it/s]
 97%|█████████▋| 762/784 [08:34<00:13,  1.67it/s]
 97%|█████████▋| 763/784 [08:35<00:12,  1.71it/s]
 97%|█████████▋| 764/784 [08:35<00:11,  1.71it/s]
 98%|█████████▊| 765/784 [08:36<00:10,  1.79it/s]
 98%|█████████▊| 766/784 [08:37<00:12,  1.42it/s]
 98%|█████████▊| 767/784 [08:37<00:11,  1.45it/s]
 98%|█████████▊| 768/784 [08:38<00:10,  1.52it/s]
 98%|█████████▊| 769/784 [08:39<00:11,  1.29it/s]
```

```
 98%|█████████▊| 770/784 [08:39<00:09,  1.43it/s]
 98%|█████████▊| 771/784 [08:41<00:10,  1.22it/s]
 98%|█████████▊| 772/784 [08:41<00:09,  1.32it/s]
 99%|█████████▊| 773/784 [08:42<00:07,  1.40it/s]
 99%|█████████▊| 774/784 [08:42<00:06,  1.44it/s]
 99%|█████████▉| 775/784 [08:43<00:05,  1.58it/s]
 99%|█████████▉| 776/784 [08:44<00:05,  1.58it/s]
 99%|█████████▉| 777/784 [08:44<00:04,  1.64it/s]
 99%|█████████▉| 778/784 [08:45<00:03,  1.72it/s]
 99%|█████████▉| 779/784 [08:45<00:02,  1.75it/s]
 99%|█████████▉| 780/784 [08:46<00:02,  1.73it/s]
100%|█████████▉| 781/784 [08:46<00:01,  1.74it/s]
100%|█████████▉| 782/784 [08:47<00:01,  1.76it/s]
100%|█████████▉| 783/784 [08:48<00:00,  1.69it/s]
100%|██████████| 784/784 [08:48<00:00,  1.48it/s]3658
```

```
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)
import tensorflow
tensorflow.random.set_seed(42)
```

```
# # modelling starts using a CNN.

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

```
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=50

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


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 widtl
        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)


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


model.summary()
```

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

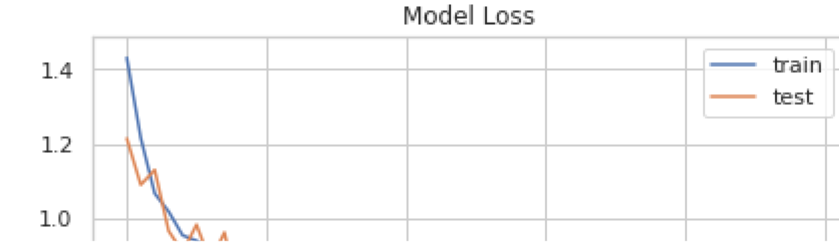
```
max_pooling2d (MaxPooling2D) (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 75, 75, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 37, 37, 64)        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
_________________________________________________________________
```

```
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_t
```

```
21/21 [==============================] - 129s 6s/step - loss: 0.5460 - accuracy: 0.7
Epoch 23/50
21/21 [==============================] - 129s 6s/step - loss: 0.5440 - accuracy: 0.7
Epoch 24/50
21/21 [==============================] - 128s 6s/step - loss: 0.5709 - accuracy: 0.7
Epoch 25/50
21/21 [==============================] - 129s 6s/step - loss: 0.5479 - accuracy: 0.7
Epoch 26/50
21/21 [==============================] - 132s 6s/step - loss: 0.5245 - accuracy: 0.8
Epoch 27/50
21/21 [==============================] - 129s 6s/step - loss: 0.5482 - accuracy: 0.7
Epoch 28/50
21/21 [==============================] - 129s 6s/step - loss: 0.5069 - accuracy: 0.8
Epoch 29/50
21/21 [==============================] - 130s 6s/step - loss: 0.5447 - accuracy: 0.7
Epoch 30/50
21/21 [==============================] - 130s 6s/step - loss: 0.5092 - accuracy: 0.8
Epoch 31/50
21/21 [==============================] - 129s 6s/step - loss: 0.4579 - accuracy: 0.8
Epoch 32/50
21/21 [==============================] - 129s 6s/step - loss: 0.4840 - accuracy: 0.8
Epoch 33/50
21/21 [==============================] - 129s 6s/step - loss: 0.4437 - accuracy: 0.8
Epoch 34/50
```

```
21/21 [==============================] - 130s 6s/step - loss: 0.4532 - accuracy: 0.8
Epoch 35/50
21/21 [==============================] - 132s 6s/step - loss: 0.4273 - accuracy: 0.8
Epoch 36/50
21/21 [==============================] - 129s 6s/step - loss: 0.4585 - accuracy: 0.8
Epoch 37/50
21/21 [==============================] - 130s 6s/step - loss: 0.4208 - accuracy: 0.8
Epoch 38/50
21/21 [==============================] - 130s 6s/step - loss: 0.4030 - accuracy: 0.8
Epoch 39/50
21/21 [==============================] - 130s 6s/step - loss: 0.4133 - accuracy: 0.8
Epoch 40/50
21/21 [==============================] - 128s 6s/step - loss: 0.3784 - accuracy: 0.8
Epoch 41/50
21/21 [==============================] - 129s 6s/step - loss: 0.3825 - accuracy: 0.8
Epoch 42/50
21/21 [==============================] - 129s 6s/step - loss: 0.4041 - accuracy: 0.8
Epoch 43/50
21/21 [==============================] - 129s 6s/step - loss: 0.3639 - accuracy: 0.8
Epoch 44/50
21/21 [==============================] - 130s 6s/step - loss: 0.3359 - accuracy: 0.8
Epoch 45/50
21/21 [==============================] - 129s 6s/step - loss: 0.3223 - accuracy: 0.8
Epoch 46/50
21/21 [==============================] - 130s 6s/step - loss: 0.3407 - accuracy: 0.8
Epoch 47/50
21/21 [==============================] - 130s 6s/step - loss: 0.2788 - accuracy: 0.8
Epoch 48/50
21/21 [==============================] - 129s 6s/step - loss: 0.2834 - accuracy: 0.9
Epoch 49/50
21/21 [==============================] - 130s 6s/step - loss: 0.3351 - accuracy: 0.8
Epoch 50/50
21/21 [==============================] - 130s 6s/step - loss: 0.3724 - accuracy: 0.8
```

```
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()
```

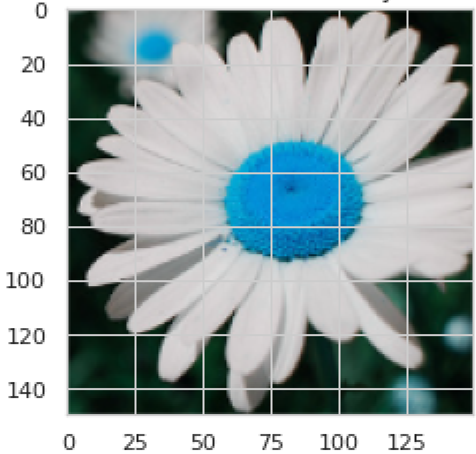



```
plt.plot(History.history['accuracy'])
plt.plot(History.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```

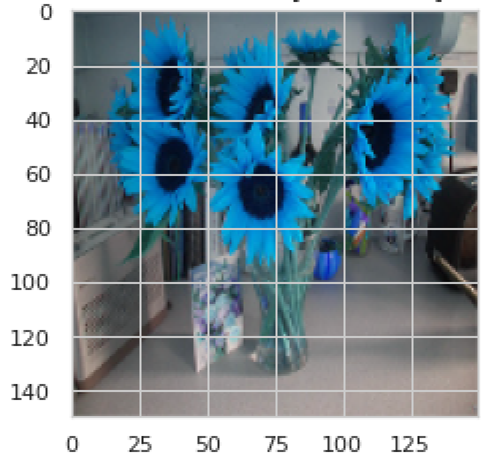

```
# getting predictions on val set.
pred=model.predict(x_test)
pred_digits=np.argmax(pred,axis=1)


# 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


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
```

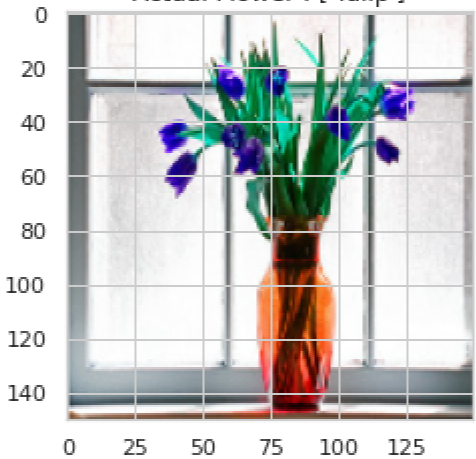
Predicted Flower : ['Daisy']
Actual Flower : ['Daisy']



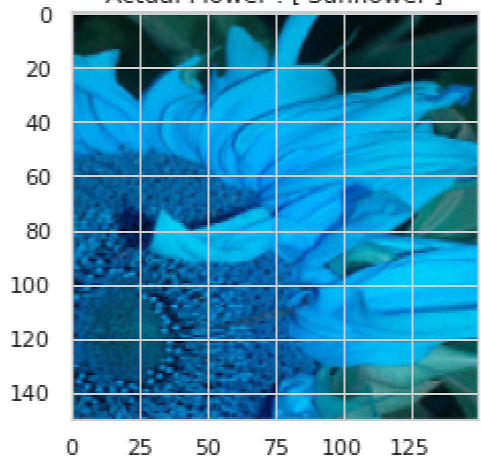
Predicted Flower : ['Sunflower']
Actual Flower : ['Sunflower']



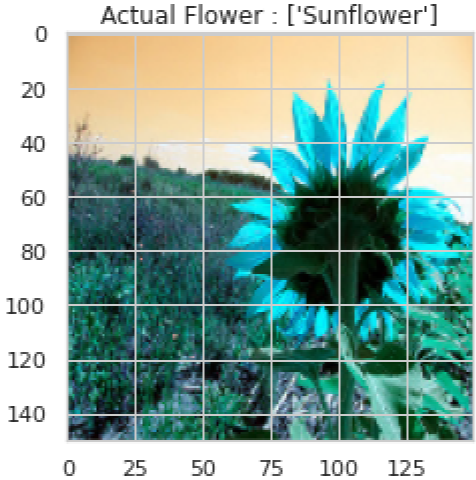
Predicted Flower : ['Tulip']
Actual Flower : ['Tulip']



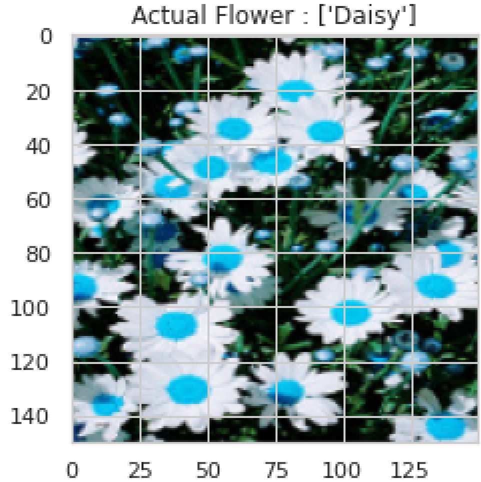
Predicted Flower : ['Sunflower']
Actual Flower : ['Sunflower']



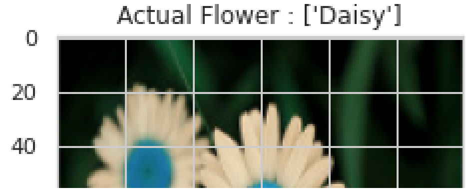
Predicted Flower : ['Sunflower']
Actual Flower : ['Sunflower']



Predicted Flower : ['Daisy']
Actual Flower : ['Daisy']



Predicted Flower : ['Daisy']
Actual Flower : ['Daisy']



Predicted Flower : ['Daisy']
Actual Flower : ['Daisy']