Assignment 3 - Mythili K

Dataset

Download the dataset here

Importing libraries

In [1]:

```
import numpy as np
import pandas as pd
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D,MaxPooling2D,Flatten,Dense
```

Data augmentation

In [2]:

```
# Data augmentaion on training variable

train_datagen = ImageDataGenerator(rescale = 1./255 , zoom_range = 0.2 , horizontal_flip =
```

In [3]:

```
# Data augmentation on testing varibale

test_datagen = ImageDataGenerator(rescale = 1./255)
```

In [4]:

```
# Data augmentation on training data

xtrain = train_datagen.flow_from_directory('dataset/train/' ,
                                           target_size = (64,64) ,
                                           class_mode = 'categorical' ,
                                           batch_size=100)
```

```
Found 3019 images belonging to 5 classes.
```

In [5]:

```
# Data augmentation on testing data

xtest = test_datagen.flow_from_directory('dataset/test/',
                                         target_size=(64,64),
                                         class_mode='categorical',
                                         batch_size=100)
```

```
Found 438 images belonging to 5 classes.
```

Model Building

In [6]:

```
# Initializing the sequential model
model = Sequential()

# Convolutional layer
model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))
```

```
# Maxpooling layer
model.add(MaxPooling2D(pool_size=(2,2)))

# Flatten layer
model.add(Flatten())

# Hidden layer 1
model.add(Dense(64,activation='relu'))

# Hidden layer 2
model.add(Dense(32,activation='relu'))

# Output layer
model.add(Dense(5,activation='softmax'))  # output
```

Compiling the model

In [7]:

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

Training the model

In [8]:

```
out=model.fit(xtrain,
              epochs=50,
              validation_data = xtest,
              validation_steps = len(xtest))
```

```
Epoch 1/50
31/31 [==============================] - 73s 2s/step - loss: 1.5628 - accuracy: 0.3640 - v
al_loss: 1.2511 - val_accuracy: 0.4749
Epoch 2/50
31/31 [==============================] - 68s 2s/step - loss: 1.2220 - accuracy: 0.4929 - v
al_loss: 1.1798 - val_accuracy: 0.4772
Epoch 3/50
31/31 [==============================] - 66s 2s/step - loss: 1.1637 - accuracy: 0.5293 - v
al_loss: 1.0959 - val_accuracy: 0.5616
Epoch 4/50
31/31 [==============================] - 35s 1s/step - loss: 1.0881 - accuracy: 0.5671 - v
al_loss: 1.1107 - val_accuracy: 0.5320
Epoch 5/50
31/31 [==============================] - 23s 727ms/step - loss: 1.0322 - accuracy: 0.6055
- val_loss: 1.0971 - val_accuracy: 0.5639
Epoch 6/50
31/31 [==============================] - 30s 972ms/step - loss: 1.0166 - accuracy: 0.6022
- val_loss: 1.0813 - val_accuracy: 0.5639
Epoch 7/50
31/31 [==============================] - 26s 843ms/step - loss: 0.9718 - accuracy: 0.6293
- val_loss: 1.1142 - val_accuracy: 0.5525
Epoch 8/50
31/31 [==============================] - 53s 2s/step - loss: 0.9519 - accuracy: 0.6350 - v
al_loss: 1.1438 - val_accuracy: 0.5342
Epoch 9/50
31/31 [==============================] - 41s 1s/step - loss: 0.9531 - accuracy: 0.6313 - v
al_loss: 1.0313 - val_accuracy: 0.6142
Epoch 10/50
31/31 [==============================] - 27s 857ms/step - loss: 0.9133 - accuracy: 0.6535
- val_loss: 1.0819 - val_accuracy: 0.5868
Epoch 11/50
31/31 [==============================] - 25s 803ms/step - loss: 0.8945 - accuracy: 0.6555
- val_loss: 1.0212 - val_accuracy: 0.6301
```

```
Epoch 12/50
31/31 [==============================] - 21s 682ms/step - loss: 0.8574 - accuracy: 0.6698
- val_loss: 0.9979 - val_accuracy: 0.6164
Epoch 13/50
31/31 [==============================] - 26s 831ms/step - loss: 0.8378 - accuracy: 0.6870
- val_loss: 0.9670 - val_accuracy: 0.6324
Epoch 14/50
31/31 [==============================] - 27s 856ms/step - loss: 0.8524 - accuracy: 0.6767
- val_loss: 1.0705 - val_accuracy: 0.5890
Epoch 15/50
31/31 [==============================] - 31s 1s/step - loss: 0.8255 - accuracy: 0.6843 - v
al_loss: 0.9381 - val_accuracy: 0.6507
Epoch 16/50
31/31 [==============================] - 28s 901ms/step - loss: 0.7804 - accuracy: 0.7079
- val_loss: 1.0082 - val_accuracy: 0.6324
Epoch 17/50
31/31 [==============================] - 37s 1s/step - loss: 0.7846 - accuracy: 0.7098 - v
al_loss: 0.8904 - val_accuracy: 0.6621
Epoch 18/50
31/31 [==============================] - 59s 2s/step - loss: 0.7533 - accuracy: 0.7088 - v
al_loss: 0.9675 - val_accuracy: 0.6507
Epoch 19/50
31/31 [==============================] - 48s 2s/step - loss: 0.7581 - accuracy: 0.7128 - v
al_loss: 1.0419 - val_accuracy: 0.6119
Epoch 20/50
31/31 [==============================] - 60s 2s/step - loss: 0.7270 - accuracy: 0.7337 - v
al_loss: 0.9127 - val_accuracy: 0.6781
Epoch 21/50
31/31 [==============================] - 28s 893ms/step - loss: 0.7094 - accuracy: 0.7314
- val_loss: 0.9735 - val_accuracy: 0.6256
Epoch 22/50
31/31 [==============================] - 28s 899ms/step - loss: 0.6710 - accuracy: 0.7575
- val_loss: 1.0004 - val_accuracy: 0.6393
Epoch 23/50
31/31 [==============================] - 57s 2s/step - loss: 0.6749 - accuracy: 0.7559 - v
al_loss: 0.9598 - val_accuracy: 0.6553
Epoch 24/50
31/31 [==============================] - 65s 2s/step - loss: 0.6531 - accuracy: 0.7519 - v
al_loss: 0.9218 - val_accuracy: 0.6621
Epoch 25/50
31/31 [==============================] - 54s 2s/step - loss: 0.6728 - accuracy: 0.7473 - v
al_loss: 0.9782 - val_accuracy: 0.6484
Epoch 26/50
31/31 [==============================] - 58s 2s/step - loss: 0.6206 - accuracy: 0.7661 - v
al_loss: 0.9282 - val_accuracy: 0.6575
Epoch 27/50
31/31 [==============================] - 29s 919ms/step - loss: 0.6245 - accuracy: 0.7632
- val_loss: 1.1047 - val_accuracy: 0.6416
Epoch 28/50
31/31 [==============================] - 21s 687ms/step - loss: 0.6110 - accuracy: 0.7688
- val_loss: 0.9691 - val_accuracy: 0.6461
Epoch 29/50
31/31 [==============================] - 29s 935ms/step - loss: 0.5886 - accuracy: 0.7781
- val_loss: 0.9532 - val_accuracy: 0.6553
Epoch 30/50
31/31 [==============================] - 28s 891ms/step - loss: 0.5938 - accuracy: 0.7864
- val_loss: 0.9264 - val_accuracy: 0.6484
Epoch 31/50
31/31 [==============================] - 22s 713ms/step - loss: 0.5659 - accuracy: 0.7917
- val_loss: 1.0002 - val_accuracy: 0.6370
Epoch 32/50
31/31 [==============================] - 27s 865ms/step - loss: 0.5519 - accuracy: 0.7917
- val_loss: 0.9586 - val_accuracy: 0.6484
Epoch 33/50
31/31 [==============================] - 23s 727ms/step - loss: 0.5437 - accuracy: 0.7993
- val_loss: 1.2153 - val_accuracy: 0.6119
```

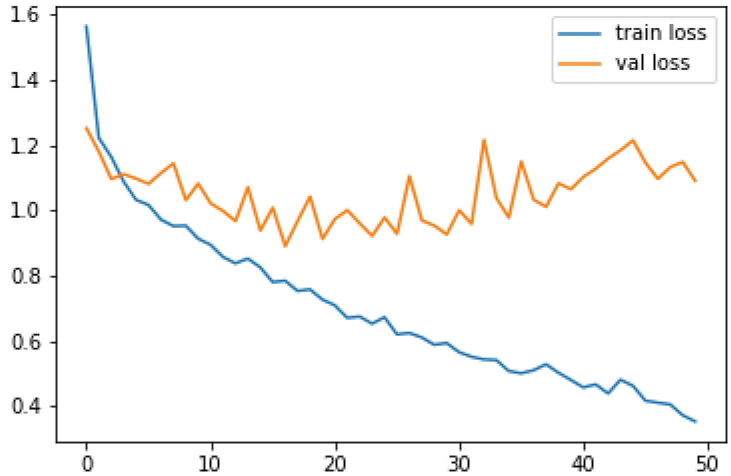
```
Epoch 34/50
31/31 [==============================] - 23s 740ms/step - loss: 0.5418 - accuracy: 0.7996
- val_loss: 1.0377 - val_accuracy: 0.6530
Epoch 35/50
31/31 [==============================] - 25s 822ms/step - loss: 0.5082 - accuracy: 0.8168
- val_loss: 0.9776 - val_accuracy: 0.6530
Epoch 36/50
31/31 [==============================] - 27s 854ms/step - loss: 0.5012 - accuracy: 0.8175
- val_loss: 1.1493 - val_accuracy: 0.6142
Epoch 37/50
31/31 [==============================] - 28s 906ms/step - loss: 0.5106 - accuracy: 0.8198
- val_loss: 1.0325 - val_accuracy: 0.6484
Epoch 38/50
31/31 [==============================] - 32s 1s/step - loss: 0.5288 - accuracy: 0.8046 - v
al_loss: 1.0109 - val_accuracy: 0.6667
Epoch 39/50
31/31 [==============================] - 26s 854ms/step - loss: 0.5031 - accuracy: 0.8185
- val_loss: 1.0826 - val_accuracy: 0.6370
Epoch 40/50
31/31 [==============================] - 25s 797ms/step - loss: 0.4808 - accuracy: 0.8331
- val_loss: 1.0646 - val_accuracy: 0.6553
Epoch 41/50
31/31 [==============================] - 28s 911ms/step - loss: 0.4581 - accuracy: 0.8248
- val_loss: 1.1024 - val_accuracy: 0.6461
Epoch 42/50
31/31 [==============================] - 25s 804ms/step - loss: 0.4672 - accuracy: 0.8258
- val_loss: 1.1276 - val_accuracy: 0.6461
Epoch 43/50
31/31 [==============================] - 27s 872ms/step - loss: 0.4397 - accuracy: 0.8433
- val_loss: 1.1588 - val_accuracy: 0.6530
Epoch 44/50
31/31 [==============================] - 24s 771ms/step - loss: 0.4815 - accuracy: 0.8205
- val_loss: 1.1843 - val_accuracy: 0.6484
Epoch 45/50
31/31 [==============================] - 25s 792ms/step - loss: 0.4629 - accuracy: 0.8304
- val_loss: 1.2142 - val_accuracy: 0.6370
Epoch 46/50
31/31 [==============================] - 26s 840ms/step - loss: 0.4174 - accuracy: 0.8476
- val_loss: 1.1455 - val_accuracy: 0.6461
Epoch 47/50
31/31 [==============================] - 24s 763ms/step - loss: 0.4114 - accuracy: 0.8546
- val_loss: 1.0965 - val_accuracy: 0.6621
Epoch 48/50
31/31 [==============================] - 31s 1000ms/step - loss: 0.4060 - accuracy: 0.8543
- val_loss: 1.1322 - val_accuracy: 0.6461
Epoch 49/50
31/31 [==============================] - 26s 843ms/step - loss: 0.3729 - accuracy: 0.8662
- val_loss: 1.1479 - val_accuracy: 0.6667
Epoch 50/50
31/31 [==============================] - 25s 813ms/step - loss: 0.3540 - accuracy: 0.8768
- val_loss: 1.0904 - val_accuracy: 0.6689
```

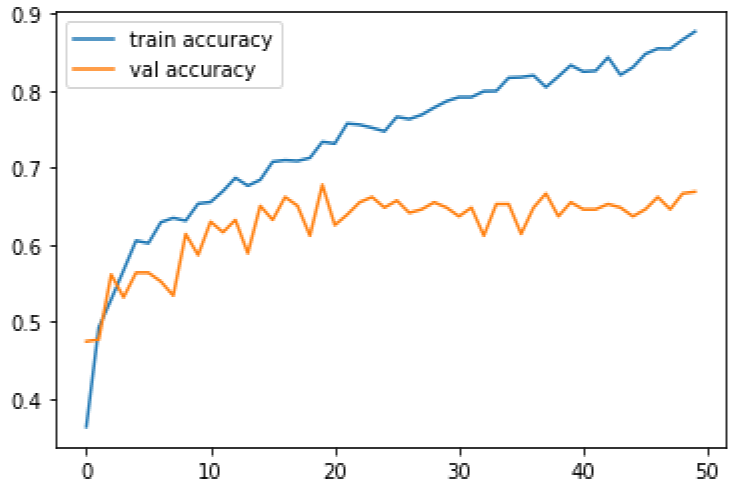
Plotting loss and accuracy

In [9]:

```
#plot the loss
plt.plot(out.history['loss'], label='train loss')
plt.plot(out.history['val_loss'], label='val loss')
plt.legend()
plt.show()

# plot the accuracy
plt.plot(out.history['accuracy'], label='train accuracy')
plt.plot(out.history['val_accuracy'], label='val accuracy')
```

```
plt.legend()
plt.show()
```





In [10]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 flatten (Flatten)           (None, 30752)             0

 dense (Dense)               (None, 64)                1968192

 dense_1 (Dense)             (None, 32)                2080

 dense_2 (Dense)             (None, 5)                 165

=================================================================
Total params: 1,971,333
Trainable params: 1,971,333
Non-trainable params: 0
_________________________________________________________________
```

Save the model

In [11]:

```
model.save('flower.h5')
```

Testing the model

In [12]:

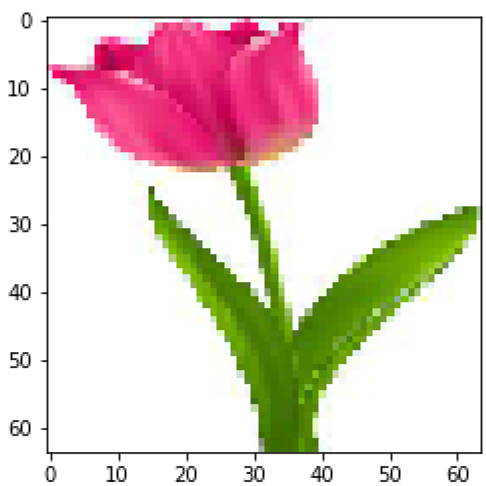
```
# Creating list
flow = ['daisy','dandelion','rose','sunflower','tulip']
def tester(img):
    img = image.load_img(img,target_size=(64,64))
    # Converting images into array
    x = image.img_to_array(img)
    # Expanding the dimensions
    x = np.expand_dims(x,axis=0)
    # Predicting the higher probability index
    pred = np.argmax(model.predict(x))
    return print("Predicted class : ",flow[pred])

# Showing image
def show(img):
    img = image.load_img(img,target_size=(64,64))
    plt.imshow(img)
```

Testing using flower images

In [13]:

```
tester('flower1.jpg')
show('flower1.jpg')
```

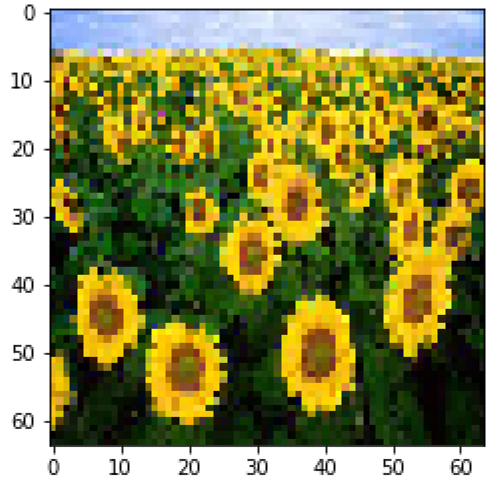
```
Predicted class :  tulip
```



In [14]:

```
tester('flower2.jpg')
show('flower2.jpg')
```

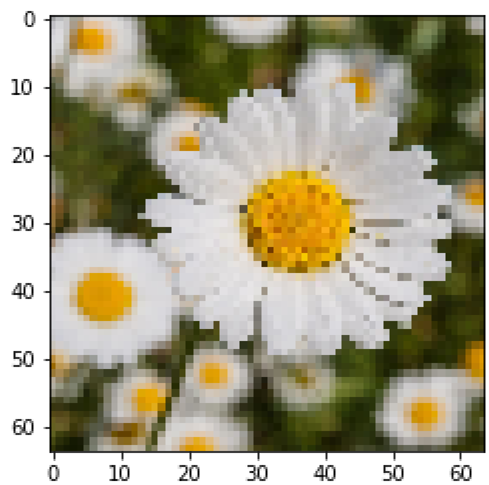
```
Predicted class :  sunflower
```



In [15]:

```
tester('flower5.jpg')
show('flower5.jpg')
```

```
Predicted class :  daisy
```



Inference

- The dataset comprises of five different classes of flowers with 4317 images
- The dataset is divided as 70% for training and 30% for testing and validation
- Model was built using Convolutional Neural network
- Accuracy : 85%
- Validation accuracy : 69%
- Testing accuracy : 90%