# Flower Recognition Through CNN Keras

---

```
# Once mounted, you can access your Google Drive files and folders
# under the '/content/drive' directory.

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

```
Mounted at /content/drive
```

## Libraries

```
# imports various libraries that will be used for data visualization,
# manipulation, model selection, and deep learning.


# data visualisation and manipulation
import numpy as np
import matplotlib.pyplot as plt
# sets matplotlib to inline and displays graphs below the corressponding cell.
%matplotlib inline

#model selection
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder

#preprocess.
from keras.preprocessing.image import ImageDataGenerator
#Transfer Learning specific modules
from keras.applications.vgg16 import VGG16

#DeepLearning libraraies
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.utils import to_categorical

import tensorflow as tf
import random as rn
import cv2
from tqdm import tqdm
import os
```

## Data Preparation

## A. Making the functions to get the training and validation set from the Images

```
# Initializing empty lists for storing data.
X=[]
Z=[]

# Defining the size of the images (both width and height) to be resized to.
IMG_SIZE=150

# Defining the directories of different flower categories in the dataset.
# Change these directories to match the actual directory paths in your Google Drive.
FLOWER_DAISY_DIR='/content/drive/MyDrive/0_Assignment/Dataset/train/daisy'
FLOWER_SUNFLOWER_DIR='/content/drive/MyDrive/0_Assignment/Dataset/train/sunflower'
FLOWER_TULIP_DIR='/content/drive/MyDrive/0_Assignment/Dataset/train/tulip'
FLOWER_ROSE_DIR='/content/drive/MyDrive/0_Assignment/Dataset/train/rose'
```

```
# A function to assign a label to an image based on its flower type.
def assign_label(img,flower_type):
    return flower_type
```

```
# A function to create the training data by processing images from a specific directory.
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))
```

```
# Creating the training data for each flower category and printing the length of the data.
make_train_data('Daisy',FLOWER_DAISY_DIR)
print(len(X))

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

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

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

print("Total length of X (training data):", len(X))
print("Total length of Z (labels):", len(Z))
print("Unique labels (flower types):", set(Z))
```

```
100%|██████████| 501/501 [00:06<00:00, 77.37it/s]
```

```
501
100%|██████████| 471/471 [00:05<00:00, 90.60it/s]
972
100%|██████████| 601/601 [00:08<00:00, 72.15it/s]
1573
100%|██████████| 497/497 [00:05<00:00, 90.06it/s] 2070
Total length of X (training data): 2070
Total length of Z (labels): 2070
Unique labels (flower types): {'Rose', 'Sunflower', 'Tulip', 'Daisy'}
```

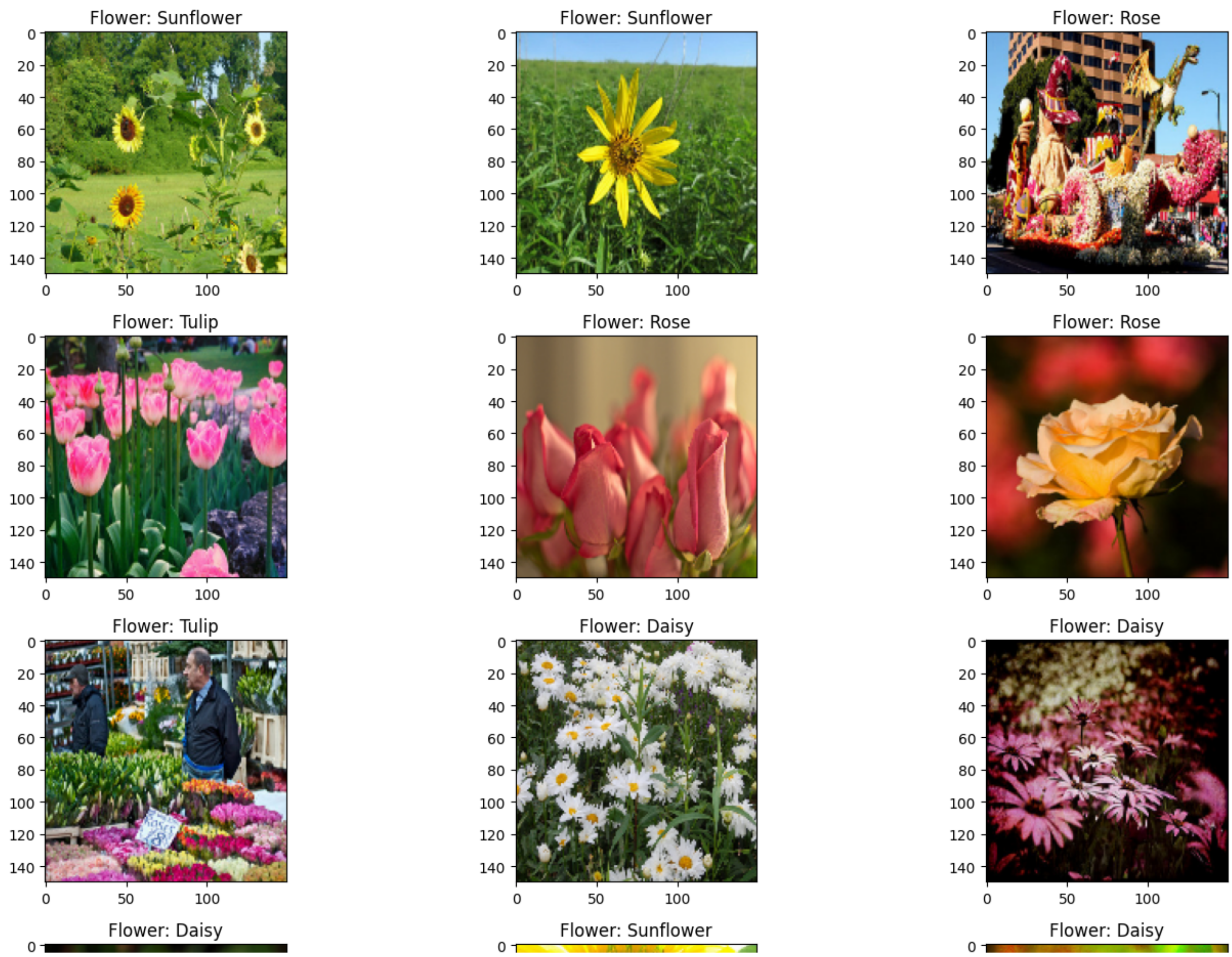
## B. Visualize some Random Images

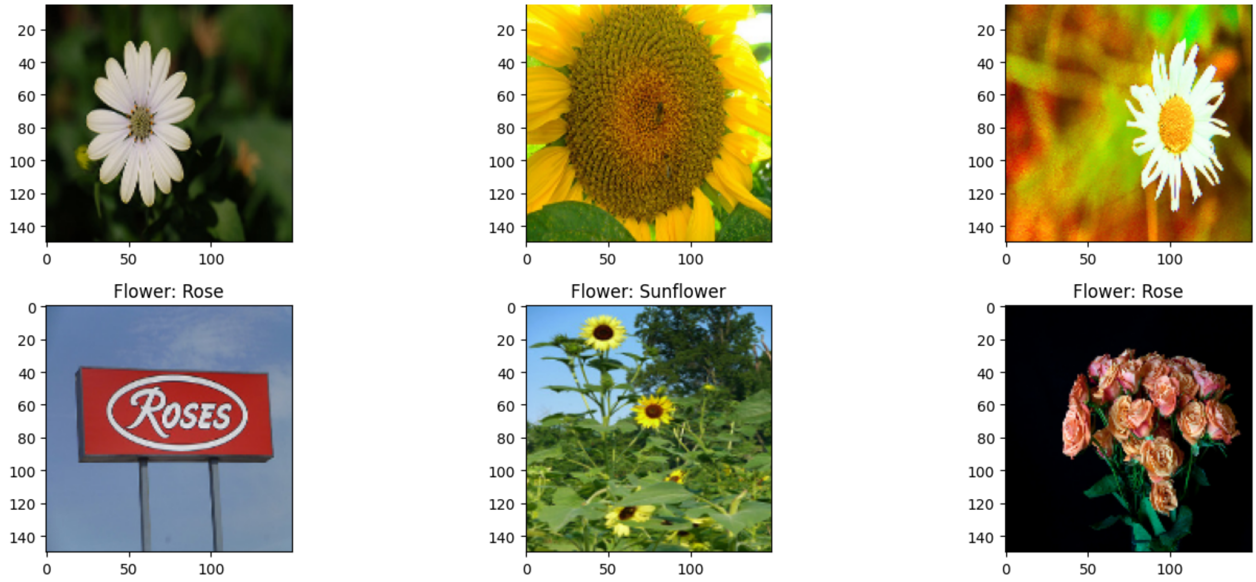
```python
# Creating subplots to display a grid of flower images with their corresponding labels.
fig,ax=plt.subplots(5,3)
fig.set_size_inches(15,15)
for i in range(5):
    for j in range (3):
        l=rn.randint(0,len(Z))

        # Displaying the image at the selected index on the current subplot.
        ax[i, j].imshow(cv2.cvtColor(X[l], cv2.COLOR_RGB2BGR))

        # Setting the title of the subplot to the corresponding flower label.
        ax[i, j].set_title('Flower: ' + Z[l])

plt.tight_layout();
```




Flower: Rose
Flower: Sunflower
Flower: Rose
ROSES

```
print("Number of Pixles in each image :",150*150*3)
```

```
Number of Pixles in each image : 67500
```

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

```
# Label encoding and one-hot encoding the labels.
le=LabelEncoder()
Y=le.fit_transform(Z)
Y=to_categorical(Y,4)
print(Y.shape)

# Printing information about the X data.
print(type(X))
print(len(X))
print(X[1].shape)


X_NEW=np.array(X) # Converting X to a NumPy array for further processing.

# Image Standardization: Scaling the pixel values between 0 and 1.
X_NEW=X_NEW/255
```

```
(2070, 4)
<class 'list'>
2070
(150, 150, 3)
```

## D. Splitting into Training and Test Sets

```
# Splitting the data into training and testing sets using train_test_split.
X_train,X_test,y_train,y_test=train_test_split(X_NEW,Y,test_size=0.25,random_state=42)

# Printing the shapes of the training and testing sets.
print("Shape of X_train:", np.shape(X_train))
print("Shape of y_train:", np.shape(y_train))
print("Shape of X_test:", np.shape(X_test))
print("Shape of y_test:", np.shape(y_test))
```

```
Shape of X_train: (1552, 150, 150, 3)
Shape of y_train: (1552, 4)
Shape of X_test: (518, 150, 150, 3)
Shape of y_test: (518, 4)
```

## E. Setting random seeds for reproducibility.

```
np.random.seed(42) # Setting the random seed for NumPy.
```

```
rn.seed(42) # Setting the random seed for Python's random module.
tf.random.set_seed(42) # Setting the random seed for TensorFlow.
```

## 3. Modelling (Creating an ImageDataGenerator for data augmentation.)

```
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 widt
        height_shift_range=0.2,  # randomly shift images vertically (fraction of total heigh
        horizontal_flip=True,  # randomly flip images
        vertical_flip=False)  # randomly flip images
```

```
# Loading the VGG16 pre-trained model and its weights.
base_model = VGG16(include_top=False, weights=None, input_shape=(150, 150, 3), pooling='avg

weights_path = '/content/drive/MyDrive/0_Assignment/Dataset/vgg16_weights_tf_dim_ordering_t

base_model.load_weights(weights_path)

# Printing the summary of the VGG16 base model.
base_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 150, 150, 3)]     0

 block1_conv1 (Conv2D)       (None, 150, 150, 64)      1792

 block1_conv2 (Conv2D)       (None, 150, 150, 64)      36928

 block1_pool (MaxPooling2D)  (None, 75, 75, 64)        0

 block2_conv1 (Conv2D)       (None, 75, 75, 128)       73856

 block2_conv2 (Conv2D)       (None, 75, 75, 128)       147584

 block2_pool (MaxPooling2D)  (None, 37, 37, 128)       0

 block3_conv1 (Conv2D)       (None, 37, 37, 256)       295168

 block3_conv2 (Conv2D)       (None, 37, 37, 256)       590080

 block3_conv3 (Conv2D)       (None, 37, 37, 256)       590080

 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0
```

```
 block3_pool (MaxPooling2D)  (None, 18, 18, 256)       0

 block4_conv1 (Conv2D)       (None, 18, 18, 512)       1180160

 block4_conv2 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_conv3 (Conv2D)       (None, 18, 18, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 9, 9, 512)         0

 block5_conv1 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv2 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_conv3 (Conv2D)       (None, 9, 9, 512)         2359808

 block5_pool (MaxPooling2D)  (None, 4, 4, 512)         0

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

```
# Defining the complete model architecture.
model = Sequential() # Creating a Sequential model.

# Adding the base model (VGG16) as the first layer.
model.add(base_model)

# Adding additional layers to the model.
model.add(Dense(512, activation='relu')) # Dense layer with 512 units and ReLU activation.
model.add(Dense(64, activation='relu')) # Dense layer with 64 units and ReLU activation.
model.add(Dense(4, activation='softmax')) # Dense layer with 4 units and softmax activation.

# Printing the summary of the complete model.
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 512)               14714688

 dense (Dense)               (None, 512)               262656

 dense_1 (Dense)             (None, 64)                32832

 dense_2 (Dense)             (None, 4)                 260

=================================================================
Total params: 15,010,436
Trainable params: 15,010,436
Non-trainable params: 0
_________________________________________________________________
```

```
# Setting the VGG model to be untrainable.
# base_model.trainable = False

# Compiling the model.
model.compile(optimizer=Adam(learning_rate=1e-4), loss='categorical_crossentropy', metrics=


batchSize = 256
ep = 23

# Training the model.
history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batchSize),
epochs=ep, use_multiprocessing=True,
verbose=1, steps_per_epoch=X_train.shape[0] // batchSize)
```

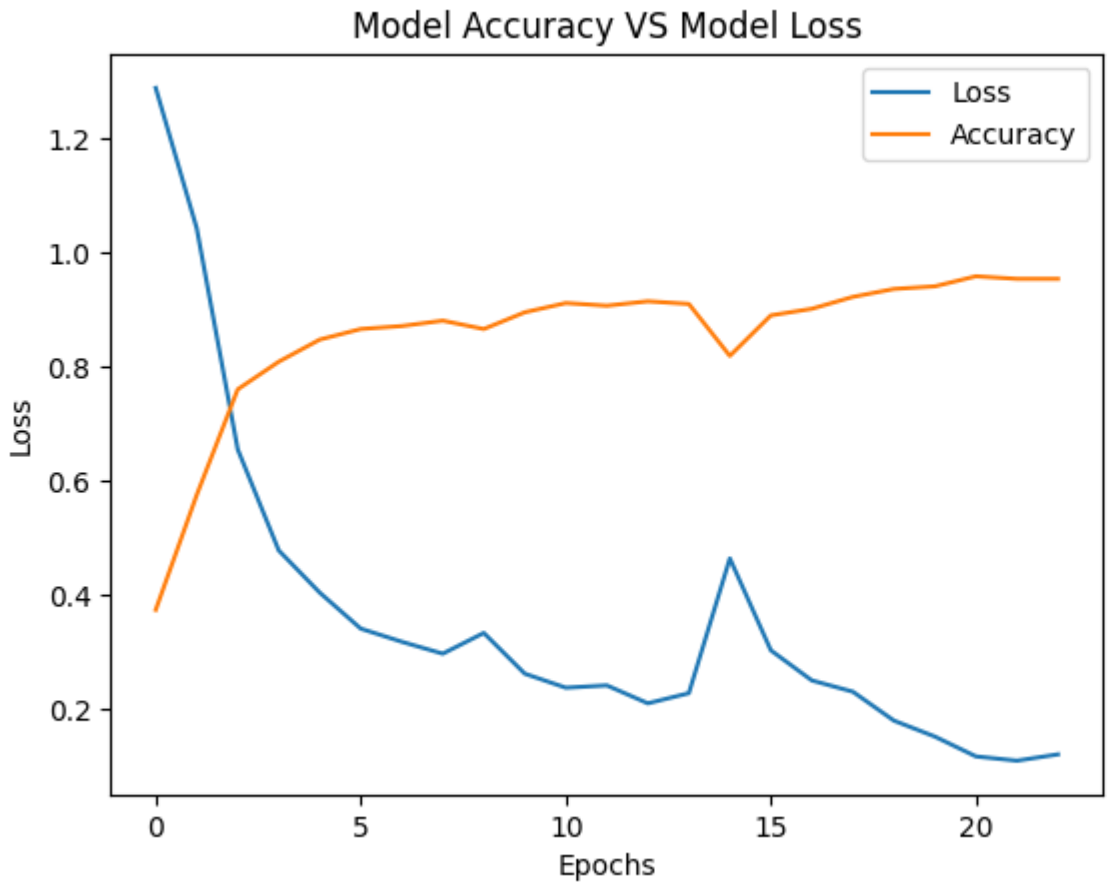
```
/usr/local/lib/python3.10/dist-packages/keras/optimizers/legacy/adam.py:117: UserW
  super().__init__(name, **kwargs)
<ipython-input-15-8326ef6f0125>:12: UserWarning: `Model.fit_generator` is deprecat
  history = model.fit_generator(datagen.flow(X_train, y_train, batch_size=batchSiz
Epoch 1/23
6/6 [==============================] - 65s 2s/step - loss: 1.2891 - accuracy: 0.37
Epoch 2/23
6/6 [==============================] - 16s 3s/step - loss: 1.0432 - accuracy: 0.57
Epoch 3/23
6/6 [==============================] - 22s 3s/step - loss: 0.6556 - accuracy: 0.76
Epoch 4/23
6/6 [==============================] - 26s 4s/step - loss: 0.4784 - accuracy: 0.80
Epoch 5/23
6/6 [==============================] - 20s 4s/step - loss: 0.4044 - accuracy: 0.84
Epoch 6/23
6/6 [==============================] - 27s 4s/step - loss: 0.3413 - accuracy: 0.86
Epoch 7/23
6/6 [==============================] - 29s 4s/step - loss: 0.3183 - accuracy: 0.87
Epoch 8/23
6/6 [==============================] - 23s 3s/step - loss: 0.2974 - accuracy: 0.88
Epoch 9/23
6/6 [==============================] - 25s 5s/step - loss: 0.3337 - accuracy: 0.86
Epoch 10/23
6/6 [==============================] - 19s 4s/step - loss: 0.2623 - accuracy: 0.89
Epoch 11/23
6/6 [==============================] - 23s 3s/step - loss: 0.2377 - accuracy: 0.91
Epoch 12/23
6/6 [==============================] - 21s 3s/step - loss: 0.2418 - accuracy: 0.90
Epoch 13/23
6/6 [==============================] - 22s 3s/step - loss: 0.2103 - accuracy: 0.91
Epoch 14/23
6/6 [==============================] - 24s 3s/step - loss: 0.2281 - accuracy: 0.91
Epoch 15/23
6/6 [==============================] - 24s 5s/step - loss: 0.4644 - accuracy: 0.81
Epoch 16/23
6/6 [==============================] - 23s 3s/step - loss: 0.3032 - accuracy: 0.89
Epoch 17/23
6/6 [==============================] - 24s 4s/step - loss: 0.2505 - accuracy: 0.90
Epoch 18/23
6/6 [==============================] - 22s 4s/step - loss: 0.2307 - accuracy: 0.92
Epoch 19/23
6/6 [==============================] - 20s 3s/step - loss: 0.1800 - accuracy: 0.93
```

```
Epoch 20/23
6/6 [==============================] - 22s 3s/step - loss: 0.1520 - accuracy: 0.94
Epoch 21/23
6/6 [==============================] - 28s 4s/step - loss: 0.1173 - accuracy: 0.95
Epoch 22/23
6/6 [==============================] - 19s 4s/step - loss: 0.1098 - accuracy: 0.95
Epoch 23/23
6/6 [==============================] - 23s 3s/step - loss: 0.1209 - accuracy: 0.95
```

```
model.evaluate(X_test,y_test)
```

```
17/17 [==============================] - 4s 110ms/step - loss: 0.3056 - accuracy:
[0.3056107461452484, 0.915057897567749]
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['accuracy'])
plt.title('Model Accuracy VS Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['Loss', 'Accuracy'])
plt.show()
```




```
model.save_weights('/content/drive/MyDrive/0_Assignment/Dataset/our_trained_model_weights/my

```