# Flower Recognition

**Bishwajit Ghorai**

Date

5/5/2020

**Course title**

Flower image Classification

## Contents

## Flower Recognition

### Dataset Content

This dataset was taken from [https://www.kaggle.com/alxmamaev/flowers-recognition](https://www.kaggle.com/alxmamaev/flowers-recognition).
This dataset contains 4363 images of flowers. The data was collected from data flicr, google images & yandex images. The pictures are divided into five category : Daisy, Dandelion, Rose, Sunflower, Tulip. There are more than 700 images for each category and each image has different proportions. We randomly print one photo of each category.

```
/content/drive/My Drive/flowers/daisy/1344985627_c3115e2d71_n.jpg
Name of Flower:  daisy ↓ at index:  502
```

```
/content/drive/My Drive/flowers/dandelion/4844697927_c70d644f40_n.jpg
Name of Flower:  dandelion ↓ at index:  827
```

```
/content/drive/My Drive/flowers/sunflower/3893436870_034b79d118_n.jpg
Name of Flower:  sunflower ↓ at index:  675
```

```
/content/drive/My Drive/flowers/rose/5336549532_49c711d49a_n.jpg
Name of Flower:  rose ↓ at index:  185
```

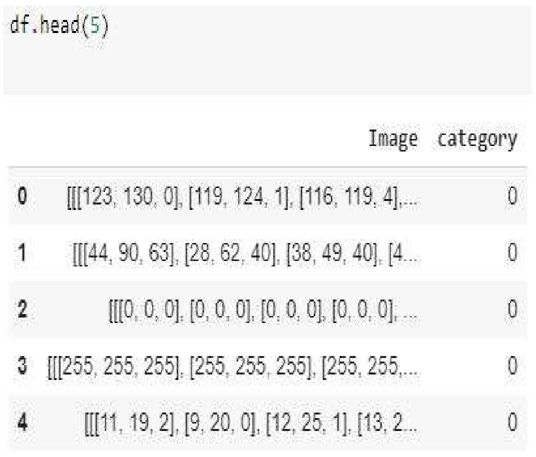
```
/content/drive/My Drive/flowers/tulip/5674127693_1ddbd81097.jpg
Name of Flower:  tulip ↓ at index:  816
```

As you see each image has a different proportion.

## Creating a data frame.

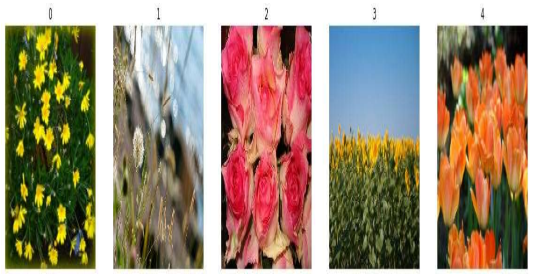
So, we have five folders and each folder contains a category of flowers. As each image has different proportion, it's better to resize each image into equal proportion for better classification. Each image is converted into it's respective matrix. Each category label is encoded into its numeric form, this done so that machine can read the code.

Daisy: '0'
Dandelion: '1'
Rose: '2'
Sunflower: '3'
Tulip: '4'



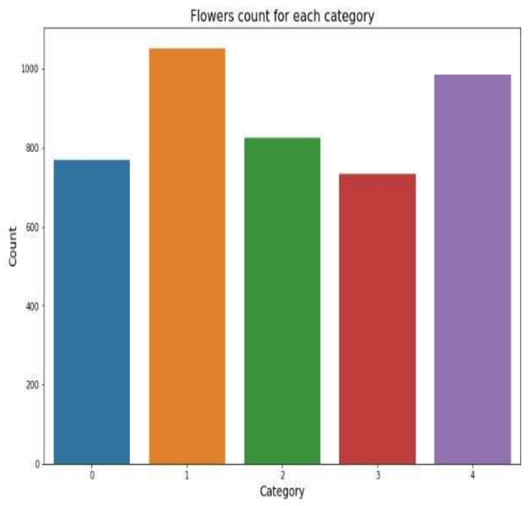
```
df.head(5)
```

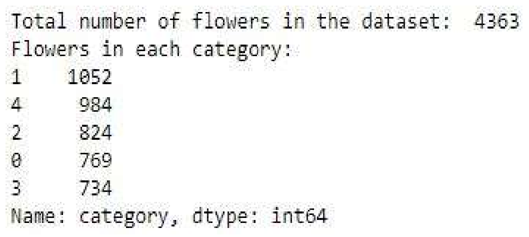
| | Image | category |
|---|---|---|
| 0 | [[[123, 130, 0], [119, 124, 1], [116, 119, 4],... | 0 |
| 1 | [[[44, 90, 63], [28, 62, 40], [38, 49, 40], [4... | 0 |
| 2 | [[[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0], ... | 0 |
| 3 | [[[255, 255, 255], [255, 255, 255], [255, 255,... | 0 |
| 4 | [[[11, 19, 2], [9, 20, 0], [12, 25, 1], [13, 2... | 0 |



## Data Visualization:

As, we see Sunflower (3) has fewer images compared to others and Dandelion (1) has the highest numbers of images. This imbalance in data can be rectified by using Data Augmentation.



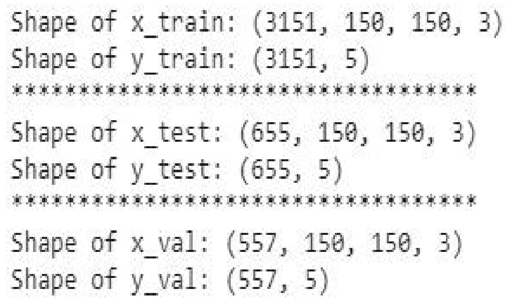


```
Total number of flowers in the dataset:  4363
Flowers in each category:
1    1052
4     984
2     824
0     769
3     734
Name: category, dtype: int64
```

## Data Preprocessing:

We split the data into train and test set and we further split the train set into train and validation set. Here, x components contains features of the data and y contains the Category/Labels.
As, we know there is a imbalance in the data we use data augmentation on the train set. We use np.utils.to_categorical on y to convert the array of labeled data to one-hot vector.

Shape of the data:



```
Shape of x_train: (3151, 150, 150, 3)
Shape of y_train: (3151, 5)
***********************************
Shape of x_test: (655, 150, 150, 3)
Shape of y_test: (655, 5)
***********************************
Shape of x_val: (557, 150, 150, 3)
Shape of y_val: (557, 5)
```

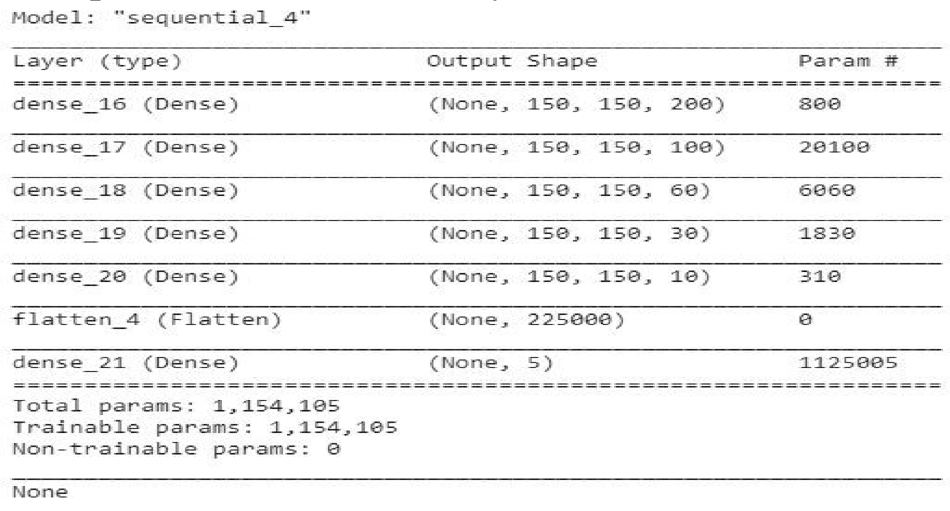
## Modelling:

We used two types of neural network in the dataset.

- Dense Neural Network (Unaugmented dataset)
- Convolutional Neural Network (CNN) (Augmented Dataset )

### Dense Neural Network:

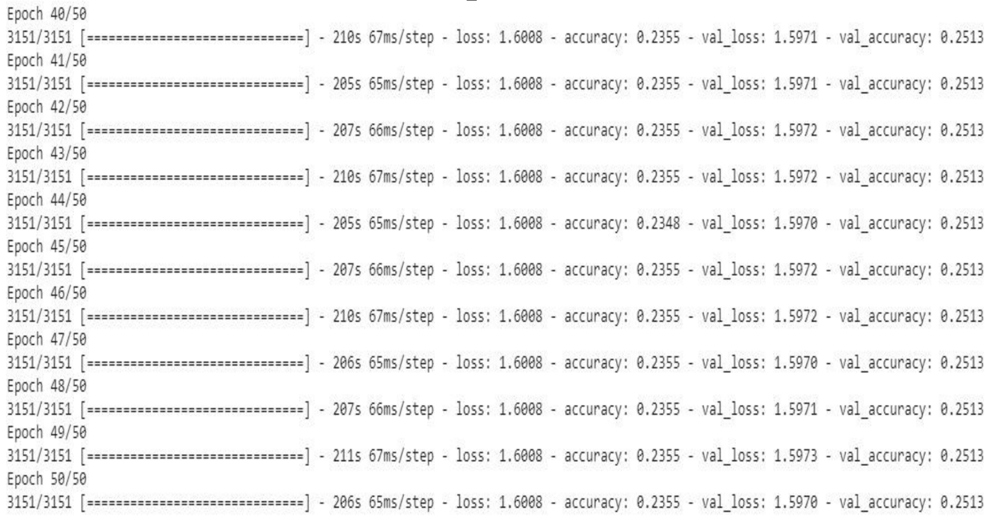
We used a Dense Neural Network consisting of 5 layers.
There are 5 relu activation and output activation softmax.
If we print the model summary:



```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_16 (Dense)             (None, 150, 150, 200)     800
_________________________________________________________________
dense_17 (Dense)             (None, 150, 150, 100)     20100
_________________________________________________________________
dense_18 (Dense)             (None, 150, 150, 60)      6060
_________________________________________________________________
dense_19 (Dense)             (None, 150, 150, 30)      1830
_________________________________________________________________
dense_20 (Dense)             (None, 150, 150, 10)      310
_________________________________________________________________
flatten_4 (Flatten)          (None, 225000)            0
_________________________________________________________________
dense_21 (Dense)             (None, 5)                 1125005
=================================================================
Total params: 1,154,105
Trainable params: 1,154,105
Non-trainable params: 0
_________________________________________________________________
None
```

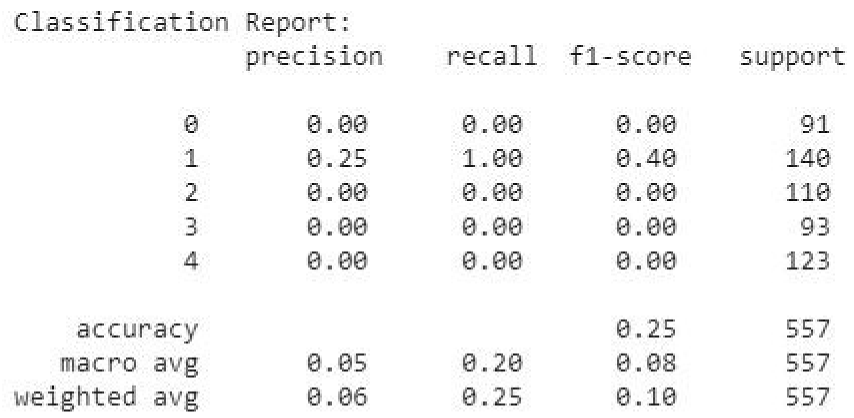
## Compile and train the model:

The batch size is of 32 and epochs is 50.



```
Epoch 40/50
3151/3151 [==============================] - 210s 67ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5971 - val_accuracy: 0.2513
Epoch 41/50
3151/3151 [==============================] - 205s 65ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5971 - val_accuracy: 0.2513
Epoch 42/50
3151/3151 [==============================] - 207s 66ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5972 - val_accuracy: 0.2513
Epoch 43/50
3151/3151 [==============================] - 210s 67ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5972 - val_accuracy: 0.2513
Epoch 44/50
3151/3151 [==============================] - 205s 65ms/step - loss: 1.6008 - accuracy: 0.2348 - val_loss: 1.5970 - val_accuracy: 0.2513
Epoch 45/50
3151/3151 [==============================] - 207s 66ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5972 - val_accuracy: 0.2513
Epoch 46/50
3151/3151 [==============================] - 210s 67ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5972 - val_accuracy: 0.2513
Epoch 47/50
3151/3151 [==============================] - 206s 65ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5970 - val_accuracy: 0.2513
Epoch 48/50
3151/3151 [==============================] - 207s 66ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5971 - val_accuracy: 0.2513
Epoch 49/50
3151/3151 [==============================] - 211s 67ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5973 - val_accuracy: 0.2513
Epoch 50/50
3151/3151 [==============================] - 206s 65ms/step - loss: 1.6008 - accuracy: 0.2355 - val_loss: 1.5970 - val_accuracy: 0.2513
```

Last 10 epochs are shown here, but you can see that loss is quite large and accuracy is quite low.

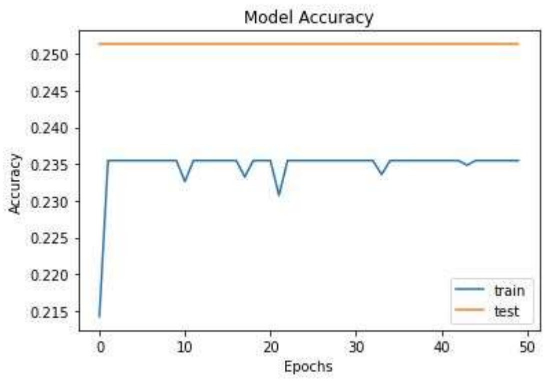


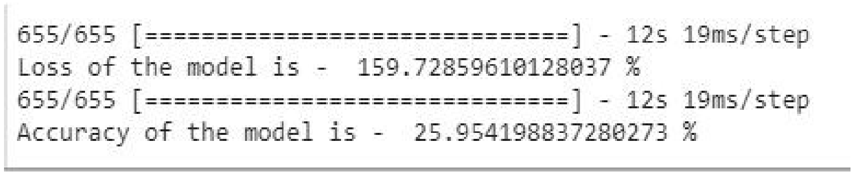
```
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        91
           1       0.25      1.00      0.40       140
           2       0.00      0.00      0.00       110
           3       0.00      0.00      0.00        93
           4       0.00      0.00      0.00       123

    accuracy                           0.25       557
   macro avg       0.05      0.20      0.08       557
weighted avg       0.06      0.25      0.10       557
```

Well predictions are quite unbalanced.



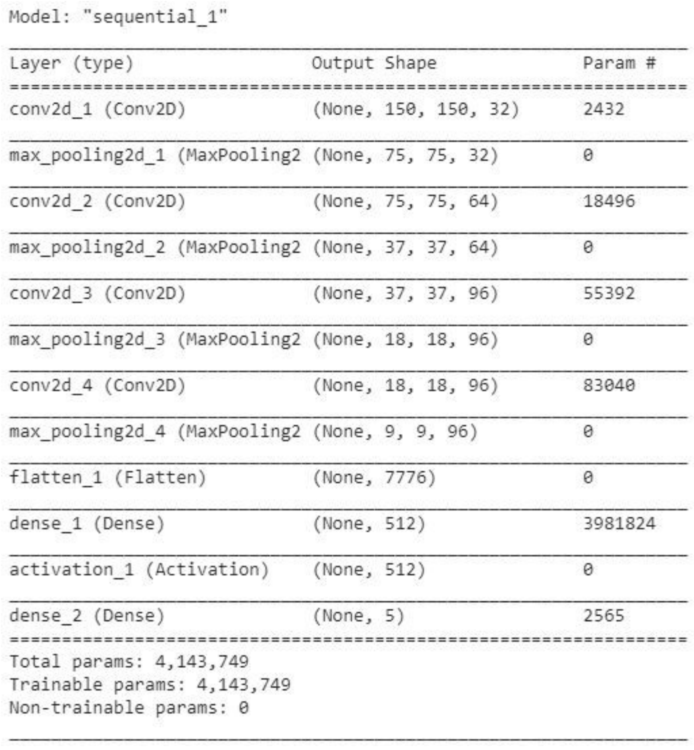


```
655/655 [==============================] - 12s 19ms/step
Loss of the model is -  159.72859610128037 %
655/655 [==============================] - 12s 19ms/step
Accuracy of the model is -  25.954198837280273 %
```

After testing it on test data set, the results are not that impressive. So, we augmented the train data on CNN.

## Convolutional Neural Network:

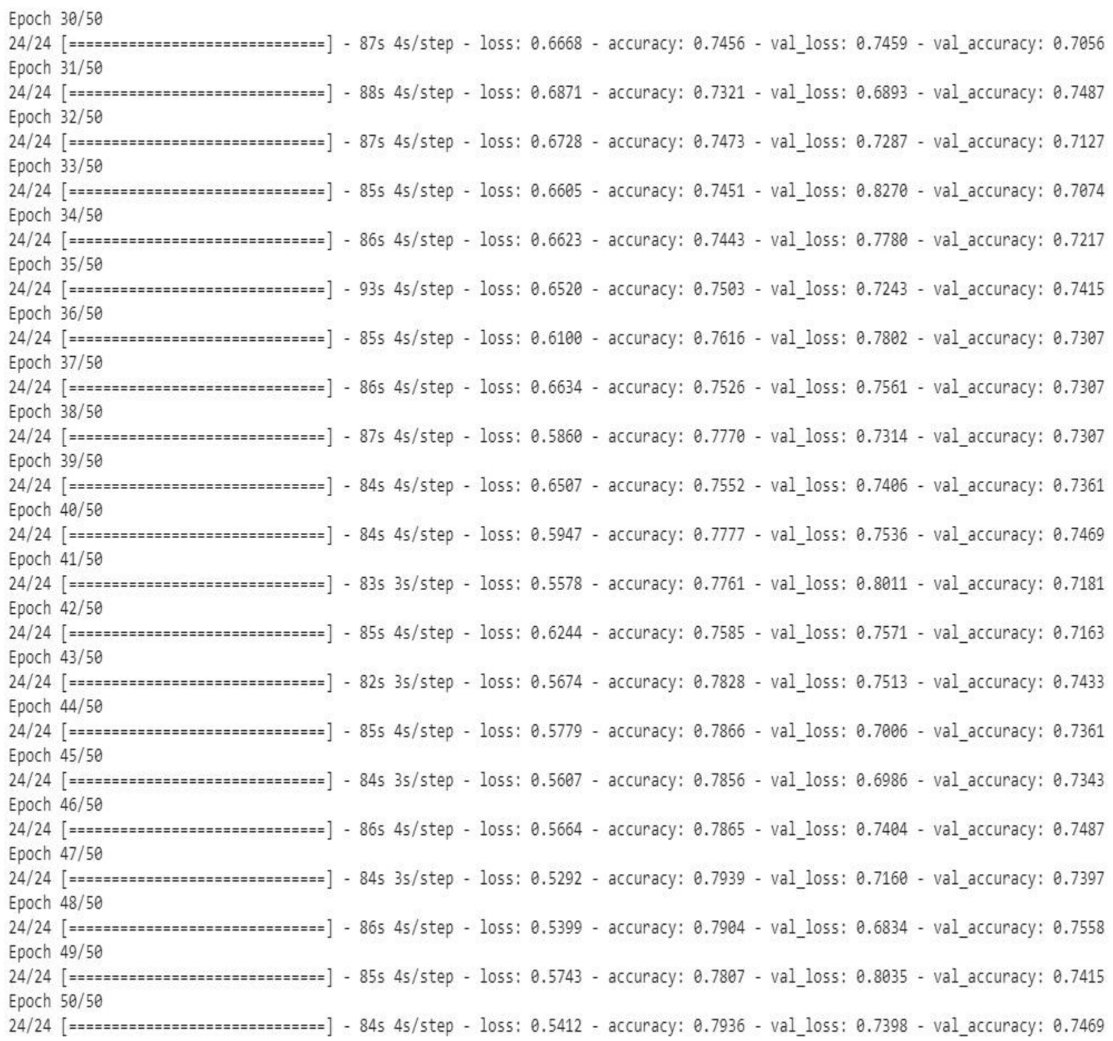
CNN takes tensors of shape (image height, image width, color channels), ignoring the batch size. Color channels refers to (R,G,B).
We configured our CNN to process inputs of shape (150,150,3). This can be done by passing the argument input_shape to our first layer.
The width and height dimensions tend to shrink as you go deeper in the network. The number of output channels for each Conv2D layer is controlled by the first argument. To complete our model we feed our last Output tensor from the convolutional base into two Dense layers to perform classification.
Here, the architecture of my model i.e. the model summary.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 150, 150, 32)      2432      
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 75, 75, 32)        0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 75, 75, 64)        18496     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 37, 37, 64)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 37, 37, 96)        55392     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 18, 18, 96)        0         
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 18, 18, 96)        83040     
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 9, 9, 96)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 7776)              0         
_________________________________________________________________
dense_1 (Dense)              (None, 512)               3981824   
_________________________________________________________________
activation_1 (Activation)    (None, 512)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 2565      
=================================================================
Total params: 4,143,749
Trainable params: 4,143,749
Non-trainable params: 0
_________________________________________________________________
```

## Compile and train the model

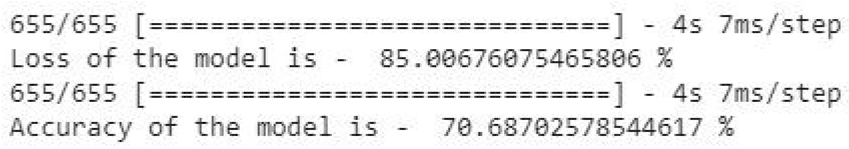
We are using batch size of 128 and epochs is 50.



```
Epoch 30/50
24/24 [==============================] - 87s 4s/step - loss: 0.6668 - accuracy: 0.7456 - val_loss: 0.7459 - val_accuracy: 0.7056
Epoch 31/50
24/24 [==============================] - 88s 4s/step - loss: 0.6871 - accuracy: 0.7321 - val_loss: 0.6893 - val_accuracy: 0.7487
Epoch 32/50
24/24 [==============================] - 87s 4s/step - loss: 0.6728 - accuracy: 0.7473 - val_loss: 0.7287 - val_accuracy: 0.7127
Epoch 33/50
24/24 [==============================] - 85s 4s/step - loss: 0.6605 - accuracy: 0.7451 - val_loss: 0.8270 - val_accuracy: 0.7074
Epoch 34/50
24/24 [==============================] - 86s 4s/step - loss: 0.6623 - accuracy: 0.7443 - val_loss: 0.7780 - val_accuracy: 0.7217
Epoch 35/50
24/24 [==============================] - 93s 4s/step - loss: 0.6520 - accuracy: 0.7503 - val_loss: 0.7243 - val_accuracy: 0.7415
Epoch 36/50
24/24 [==============================] - 85s 4s/step - loss: 0.6100 - accuracy: 0.7616 - val_loss: 0.7802 - val_accuracy: 0.7307
Epoch 37/50
24/24 [==============================] - 86s 4s/step - loss: 0.6634 - accuracy: 0.7526 - val_loss: 0.7561 - val_accuracy: 0.7307
Epoch 38/50
24/24 [==============================] - 87s 4s/step - loss: 0.5860 - accuracy: 0.7770 - val_loss: 0.7314 - val_accuracy: 0.7307
Epoch 39/50
24/24 [==============================] - 84s 4s/step - loss: 0.6507 - accuracy: 0.7552 - val_loss: 0.7406 - val_accuracy: 0.7361
Epoch 40/50
24/24 [==============================] - 84s 4s/step - loss: 0.5947 - accuracy: 0.7777 - val_loss: 0.7536 - val_accuracy: 0.7469
Epoch 41/50
24/24 [==============================] - 83s 3s/step - loss: 0.5578 - accuracy: 0.7761 - val_loss: 0.8011 - val_accuracy: 0.7181
Epoch 42/50
24/24 [==============================] - 85s 4s/step - loss: 0.6244 - accuracy: 0.7585 - val_loss: 0.7571 - val_accuracy: 0.7163
Epoch 43/50
24/24 [==============================] - 82s 3s/step - loss: 0.5674 - accuracy: 0.7828 - val_loss: 0.7513 - val_accuracy: 0.7433
Epoch 44/50
24/24 [==============================] - 85s 4s/step - loss: 0.5779 - accuracy: 0.7866 - val_loss: 0.7006 - val_accuracy: 0.7361
Epoch 45/50
24/24 [==============================] - 84s 3s/step - loss: 0.5607 - accuracy: 0.7856 - val_loss: 0.6986 - val_accuracy: 0.7343
Epoch 46/50
24/24 [==============================] - 86s 4s/step - loss: 0.5664 - accuracy: 0.7865 - val_loss: 0.7404 - val_accuracy: 0.7487
Epoch 47/50
24/24 [==============================] - 84s 3s/step - loss: 0.5292 - accuracy: 0.7939 - val_loss: 0.7160 - val_accuracy: 0.7397
Epoch 48/50
24/24 [==============================] - 86s 4s/step - loss: 0.5399 - accuracy: 0.7904 - val_loss: 0.6834 - val_accuracy: 0.7558
Epoch 49/50
24/24 [==============================] - 85s 4s/step - loss: 0.5743 - accuracy: 0.7807 - val_loss: 0.8035 - val_accuracy: 0.7415
Epoch 50/50
24/24 [==============================] - 84s 4s/step - loss: 0.5412 - accuracy: 0.7936 - val_loss: 0.7398 - val_accuracy: 0.7469
```

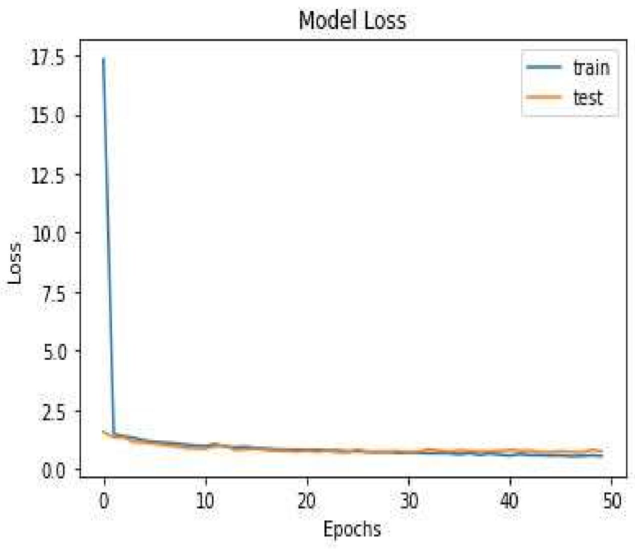
We have shown only last 10 epochs.

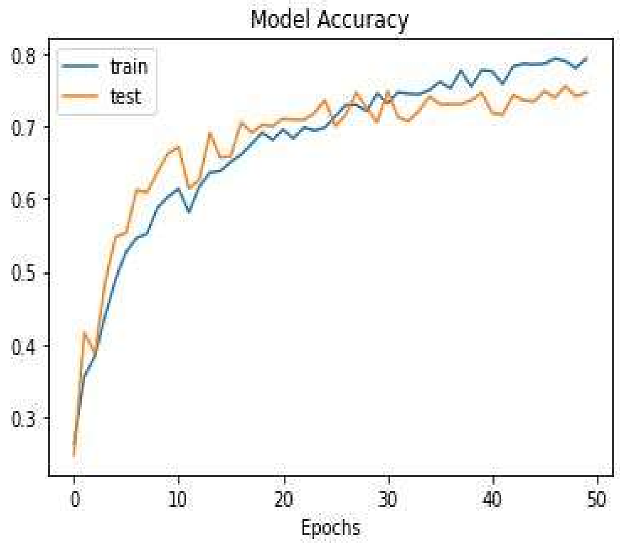
## Evaluate the model



```
655/655 [==============================] - 4s 7ms/step
Loss of the model is -  85.00676075465806 %
655/655 [==============================] - 4s 7ms/step
Accuracy of the model is -  70.68702578544617 %
```

If we see, the accuracy went down when we tested it on test set.





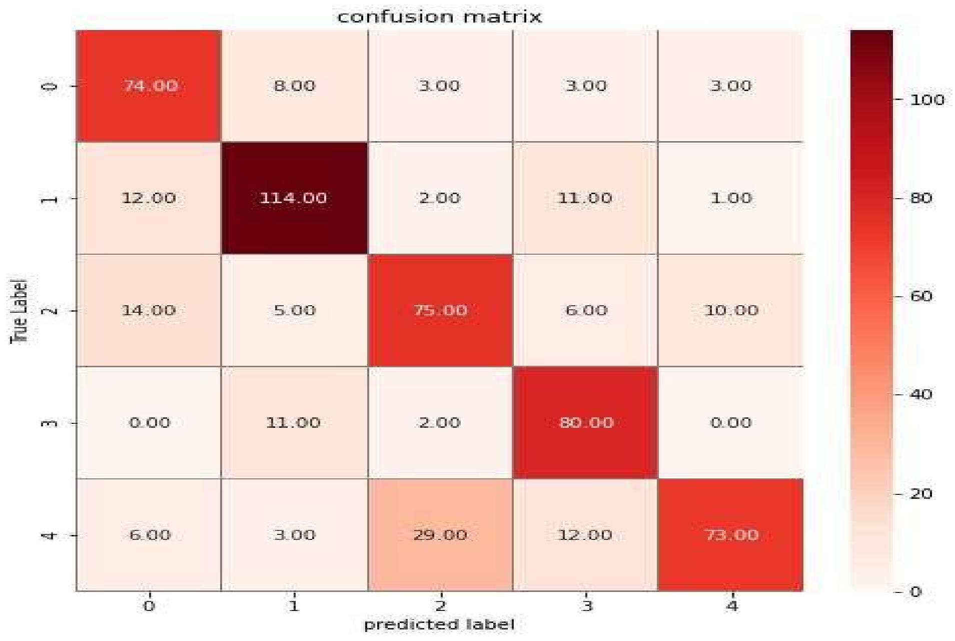
If we  look at the graph, model doesn't look too bad.

```
Classification Report:
                   precision    recall  f1-score   support

           0          0.70      0.81      0.75        91
           1          0.81      0.81      0.81       140
           2          0.68      0.68      0.68       110
           3          0.71      0.86      0.78        93
           4          0.84      0.59      0.70       123

    accuracy                              0.75       557
   macro avg          0.75      0.75      0.74       557
weighted avg          0.76      0.75      0.74       557
```

It's looks like the prediction are done on quite balanced.



| True Label \ predicted label | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 74.00 | 8.00 | 3.00 | 3.00 | 3.00 |
| 1 | 12.00 | 114.00 | 2.00 | 11.00 | 1.00 |
| 2 | 14.00 | 5.00 | 75.00 | 6.00 | 10.00 |
| 3 | 0.00 | 11.00 | 2.00 | 80.00 | 0.00 |
| 4 | 6.00 | 3.00 | 29.00 | 12.00 | 73.00 |

We see from confusion matrix that tulip is difficult to detect and sunflower is the easiest to detect.

## References:

- https://www.tensorflow.org/tutorials/images/cnn
- https://towardsdatascience.com/image-classification-python-keras-tutorial-kaggle-challenge-45a6332a58b8
- https://www.kaggle.com/rajmehra03/flower-recognition-cnn-keras
- https://www.kaggle.com/alxmamaev/flowers-recognition