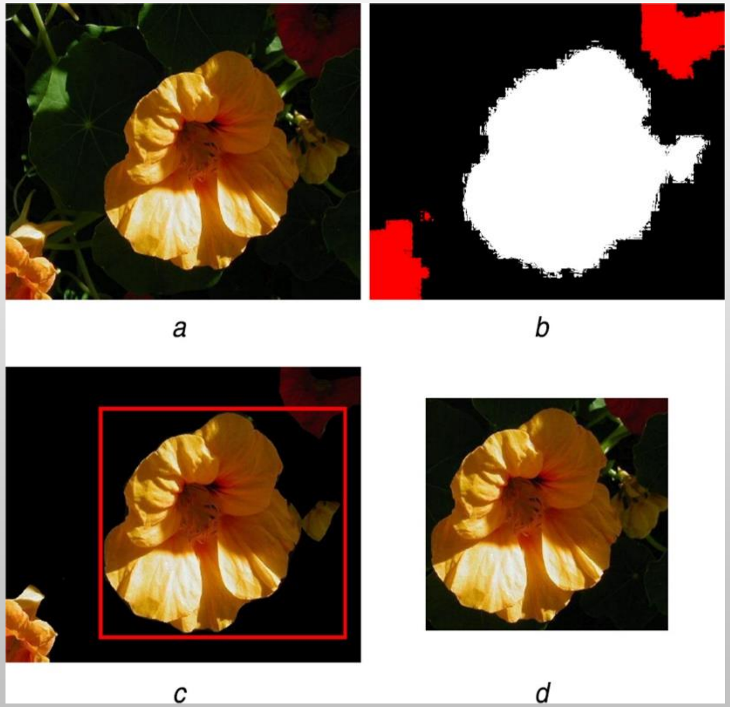
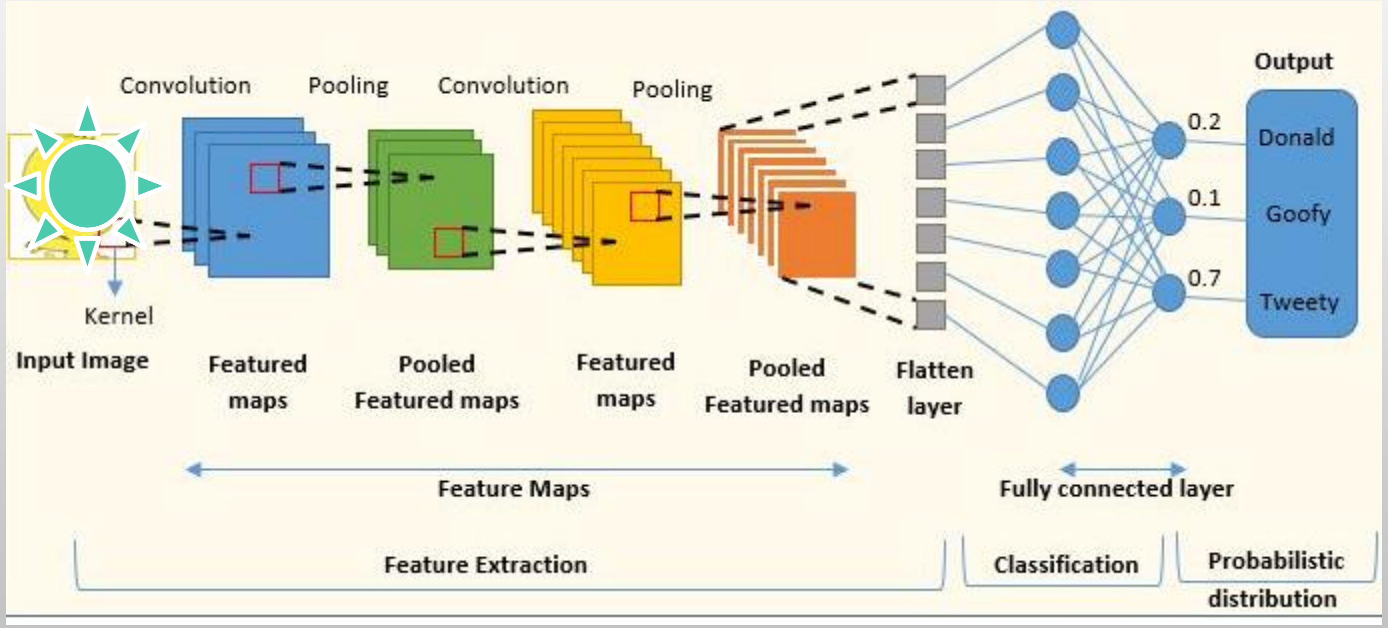
CNN MODEL FOR CLASSIFICATION OF FLOWERS

ASSIGNMENT - 3





Contents

INTRODUCE

APPROACH

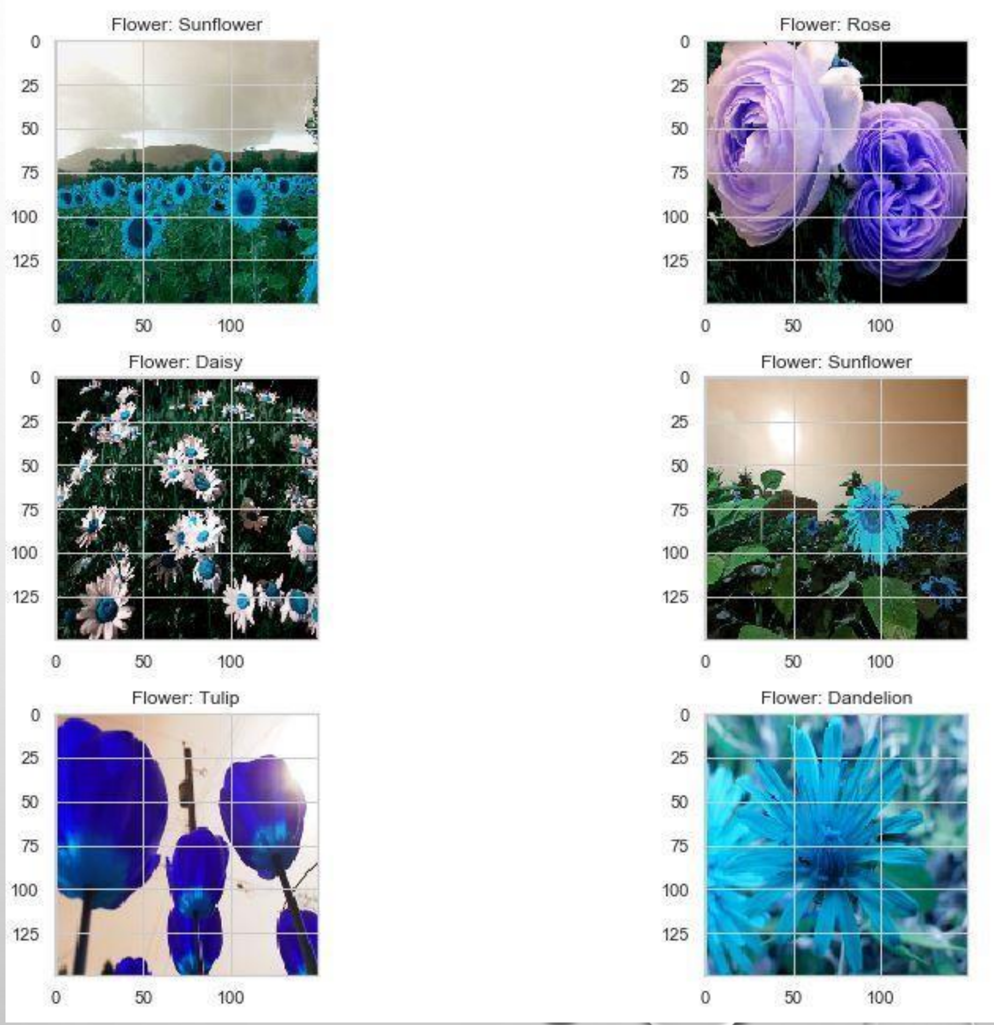
- THIS PROJECT IS ABOUT RECOGENITING THE TYPE OF FLOWERS.
- THIS PROJECT TRAINED CONVOLUTION NEURAL NETWORK WRITTEN IN KERAS TO PREDICT THE TYPE OF FLOWER ON THE VALIDATION SET.
- ALSO USED IMAGEDATAGENERATION TO AUGMENT THE TRAINING SET AND AVOID OVERFITTING PROBLEM AND A LR ANNEALER TO SCHEDULE THE LEARNING RATE.

DATASET

- THIS DATASET CONTAINS 4242 IMAGE OF FLOWERS. THE DATA COLLECTION IS BASED ON SCRAPED DATA FROM FLICKER, GOOGLE IMAGE, AND YANDEX IMAGE.
- THE PICTURE ARE DIVIDED INTO FIVE CLASSES: CHAMOMILE, TULIP, ROSE, SUNFLOWER, DANDELION.
- FOR EACH CLASS THERE ARE ABOUT 800 PHOTOS. PHOTOS ARE NOT HIGH RESOLUTION, ABOUT 30*240 PIXELS. PHOTOS ARE NOT REDUCE TO A SINGLE SIZE, THEY HAVE DIFFERENT PROPORTIONS.

PREPARING THE DATA

Dataset



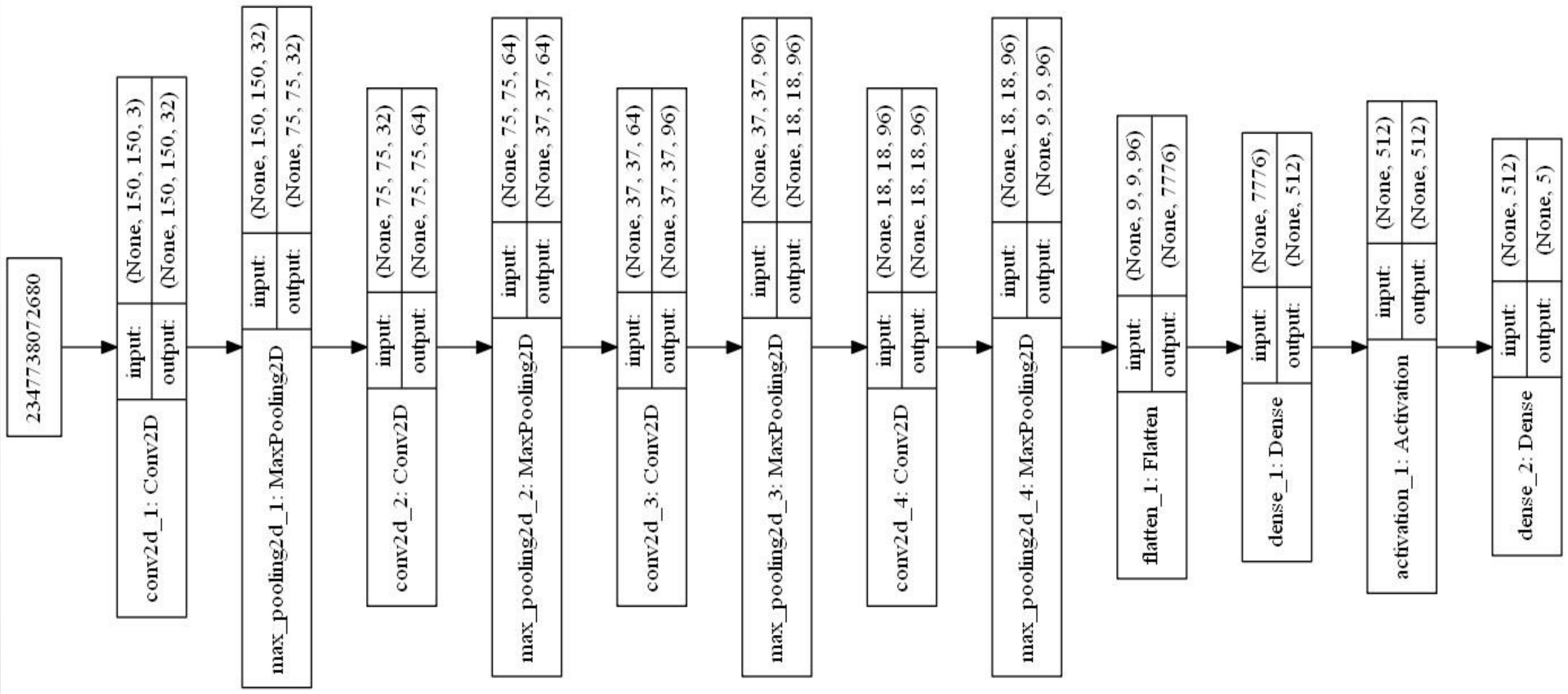
PREPARING THE DATA

RESIZE

- **SMALLER AND CONSTANT SIZE OF ALL IMAGE ARE REQUIRED FOR** CNN TO DO IMAGE CLASSIFICATION, BECAUSE THE MODEL REQUIRES A CONSTANT INPUT DIMENSIONALITY AND LOW RESOLUTION WILL SPEED UP THE MODEL TRAINING.
- IN THE PROJECT, REASONABLE RESOLUTION OF 150*150 PIXELS IS APPLIED IS APPLIED TO EACH IMAGE.

Model design

Build CNN architecture



Using plot_model to visualize the model

Model design

Build CNN architecture

- 4x(conv2d+max-pooling)+flatten+dense+activation+dense

Conv2D layer

- This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.

MODEL DESIGN

Conv2D layer

```
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'Same',activation ='relu', input_shape = (1
```

- When using this layer in a model, provide the keyword argument input_shape (tuple of integers, does not include the batch axis), e.g. input_shape=(128,128,3) for 128x128 RGB picture in data_format="channels_last".
- The ordering of the dimensions in the inputs. "channels_last" corresponds to inputs with shape(batch,height,width,channels).

MODEL DESIGN

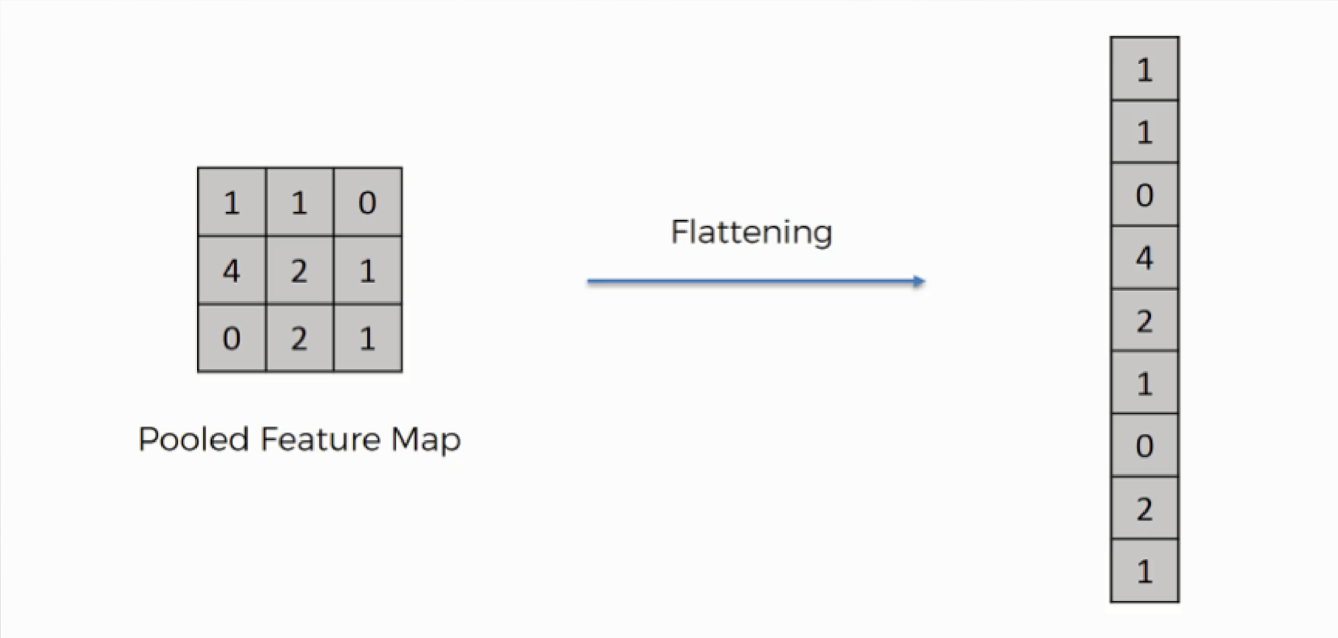
Pooling layer

- In pooling layers, features are extracted and compressed into a small map, which simplifies the neural network computation complexity, leading to the decrease of the volume of parameters and computation.
- In this project, we use max pooling.

MODEL DESIGN

Flatten layer

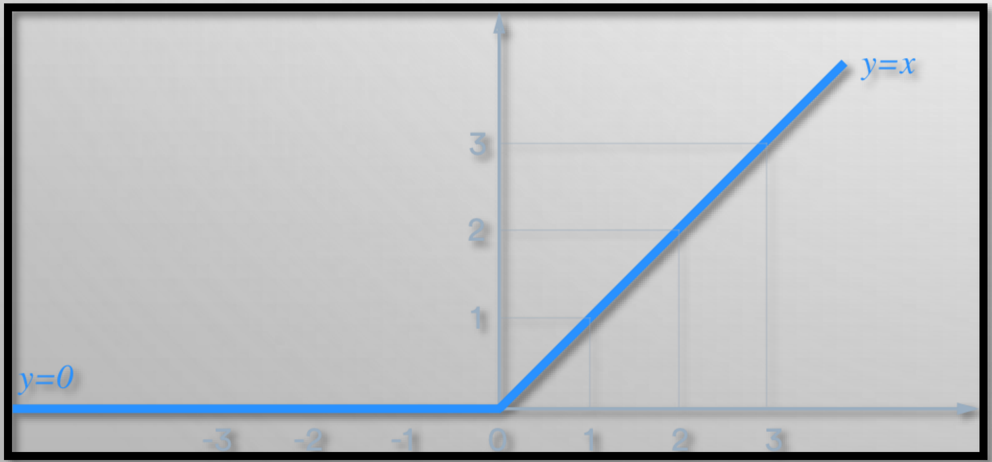
- A flatten layer collapses the spatial dimensions of the input into the channel dimension.



MODEL DESIGN

Activation layer(ReLU)

- Rectified Linear Unit (ReLU) is a piecewise linear function implemented in this model. The ReLU activation function is given by : different proportions.

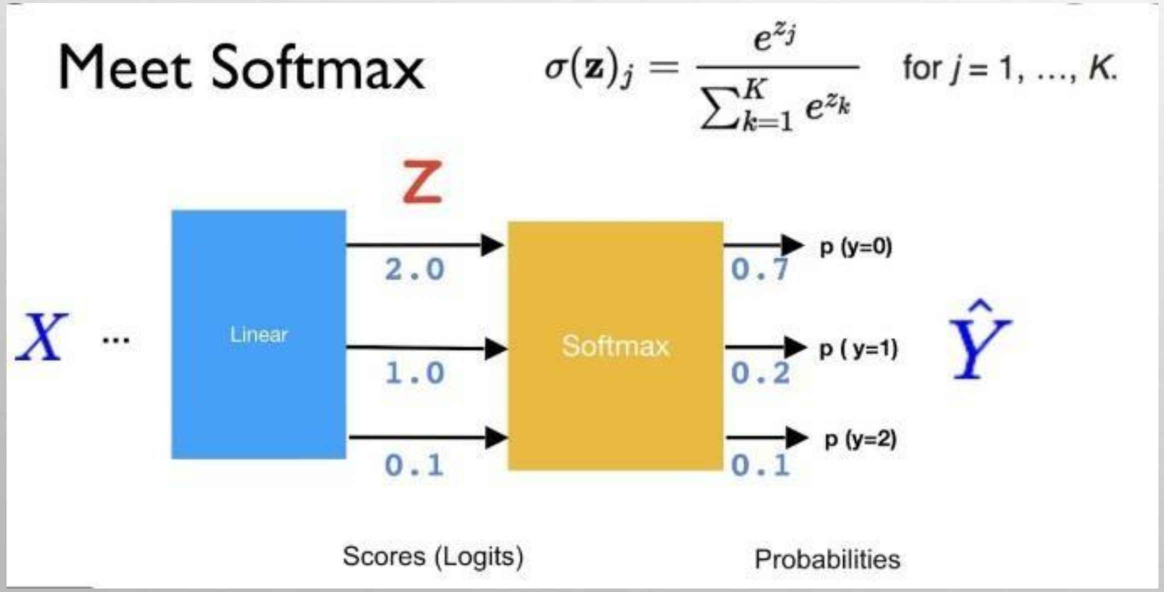


$$ReLU = \begin{cases} x, & \text{if } n > 0 \\ 0, & \text{if } x \leq 0 \end{cases}$$

MODEL DESIGN

Softmax Function

- ✓ The softmax fuction is used in neural network when we want to build a multi-class classifier which solves the problem of assigning an instance to one class when the number of possible clsses is larger than two.



Softmax Function

- The softmax fuction is used in neural networks when we want to build a muti-class classifier which solves the problem of assigning an instance to one class when the number of possible classes is larger than two.

MODEL DESIGN

Learning rate annealing(ReduceLROnPlateau)

- Reduce learning rate when a metric has stopped improving.
- Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback mointors a quantity and if no improvement is seen for a ‘patience’ number of epochs, the learning rate is reduced.

MODEL DESIGN

Data Augmentation(ImageDataGenerator class)

The Keras ImageDataGenerator class actually works by:

- Accepting a batch of image used for training.
- Taking this batch and applying a series of random transformations to each image in the batch (including random rotation, resizing, shearing, ect.)
- Replacing the original batch with the new, randomly transformed batch.
- Training the CNN on this randomly transformed batch(i.e., the original data itself is not used for training)

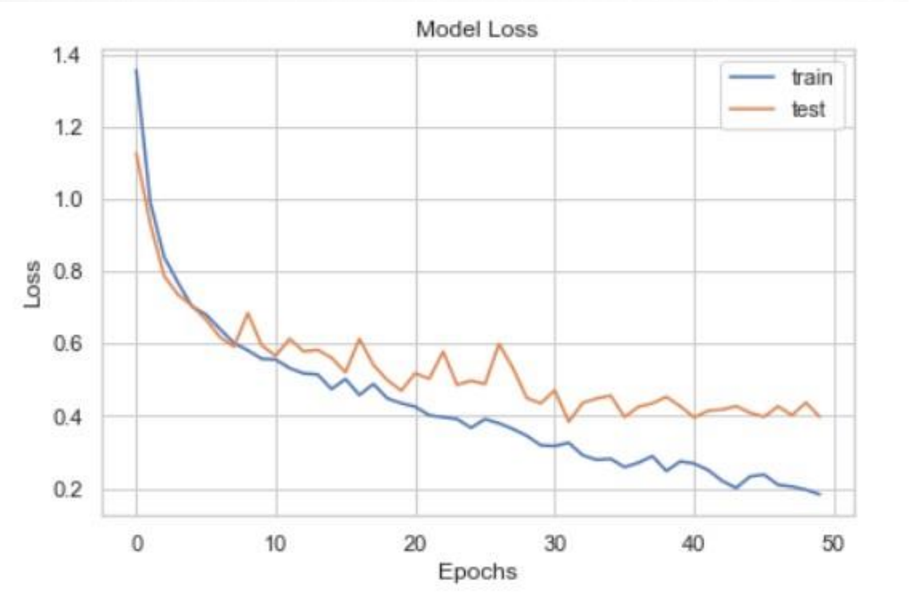
MODEL DESIGN

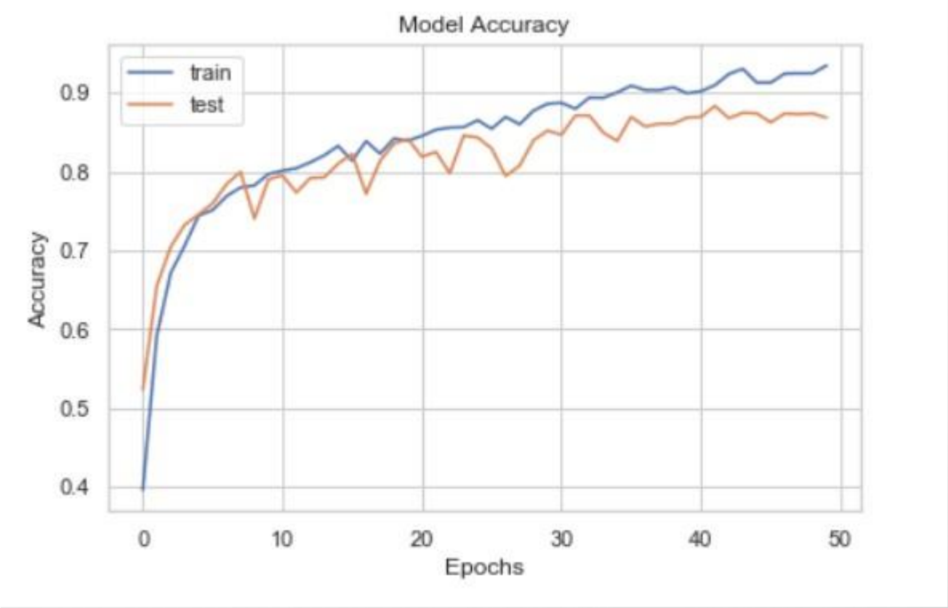
Model architecture-4

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

Result





Finally the accuracy on the validation set using the self-laid ConvNet is over 85%.

MODEL DESIGN

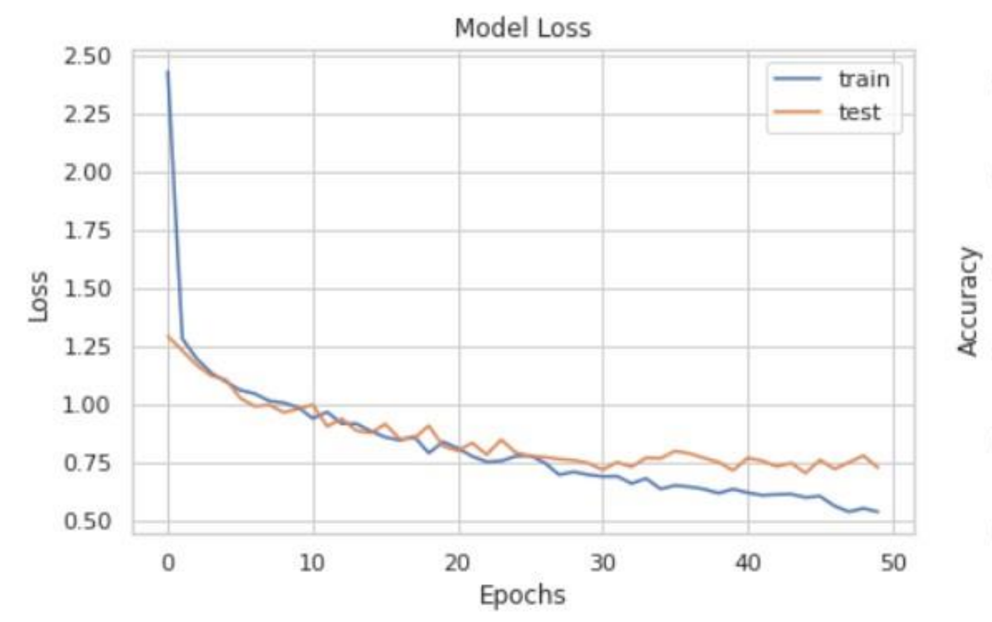
Model architecture-2

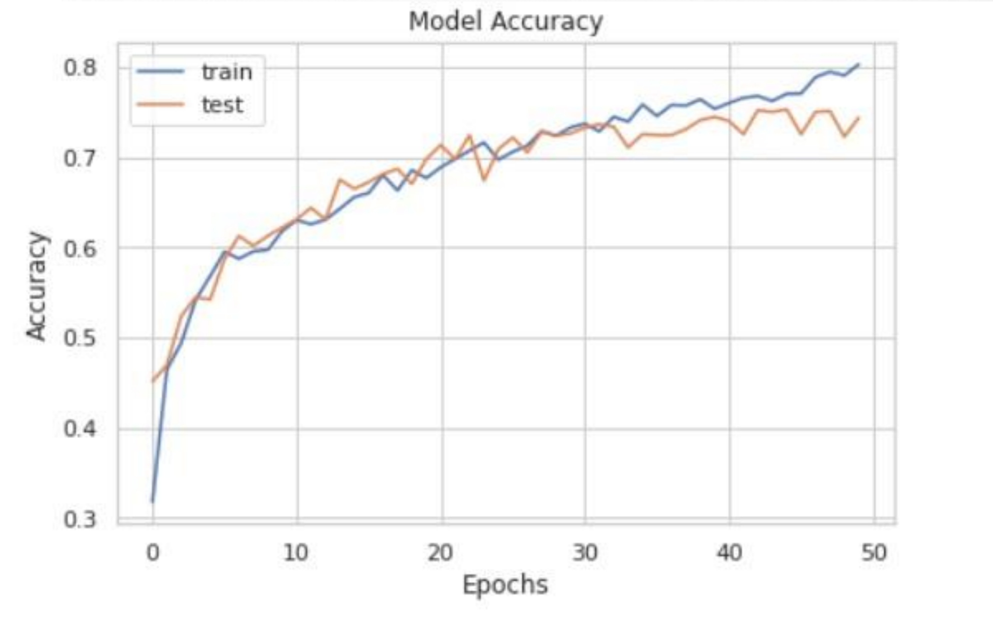
Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 150, 150, 32)	2432
max_pooling2d_1 (MaxPooling2	(None, 75, 75, 32)	0
conv2d_2 (Conv2D)	(None, 75, 75, 64)	18496
max_pooling2d_2 (MaxPooling2	(None, 37, 37, 64)	0
flatten_1 (Flatten)	(None, 87616)	0
dense_1 (Dense)	(None, 512)	44859904
activation_1 (Activation)	(None, 512)	0
dense_2 (Dense)	(None, 5)	2565

Total params: 44,883,397
Trainable params: 44,883,397
Non-trainable params: 0

MODEL DESIGN

Result-2





Finally the accuracy on the validation set using the self-laid ConvNet is close to 75%.

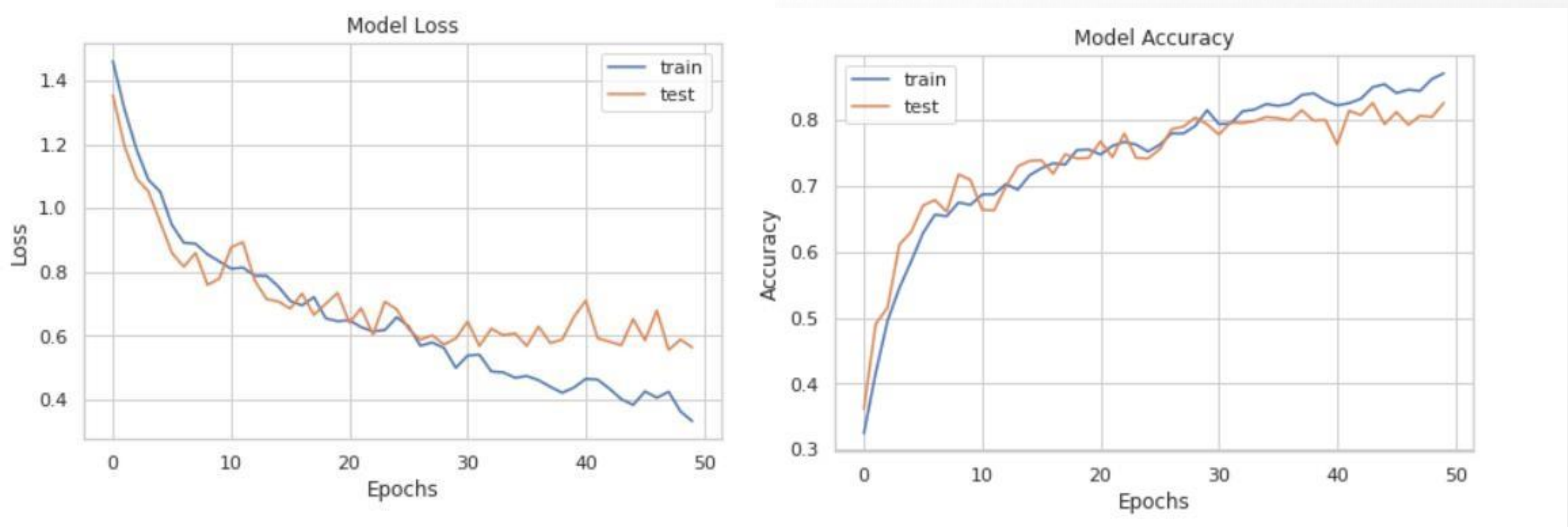
MODEL DESIGN

Model architecture-6

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 150, 150, 32)	2432
max_pooling2d_5 (MaxPooling2	(None, 75, 75, 32)	0
conv2d_6 (Conv2D)	(None, 75, 75, 64)	18496
max_pooling2d_6 (MaxPooling2	(None, 37, 37, 64)	0
conv2d_7 (Conv2D)	(None, 37, 37, 96)	55392
max_pooling2d_7 (MaxPooling2	(None, 18, 18, 96)	0
conv2d_8 (Conv2D)	(None, 18, 18, 96)	83040
max_pooling2d_8 (MaxPooling2	(None, 9, 9, 96)	0
conv2d_9 (Conv2D)	(None, 9, 9, 96)	83040
max_pooling2d_9 (MaxPooling2	(None, 4, 4, 96)	0
conv2d_10 (Conv2D)	(None, 4, 4, 96)	83040
max_pooling2d_10 (MaxPooling	(None, 2, 2, 96)	0
flatten_2 (Flatten)	(None, 384)	0
dense_3 (Dense)	(None, 512)	197120
activation_2 (Activation)	(None, 512)	0
dense_4 (Dense)	(None, 5)	2565

Total params: 525,125
Trainable params: 525,125
Non-trainable params: 0

Result-6



Finally the accuracy on the validation set using the self-laid ConvNet is around 80%.

MODEL DESIGN

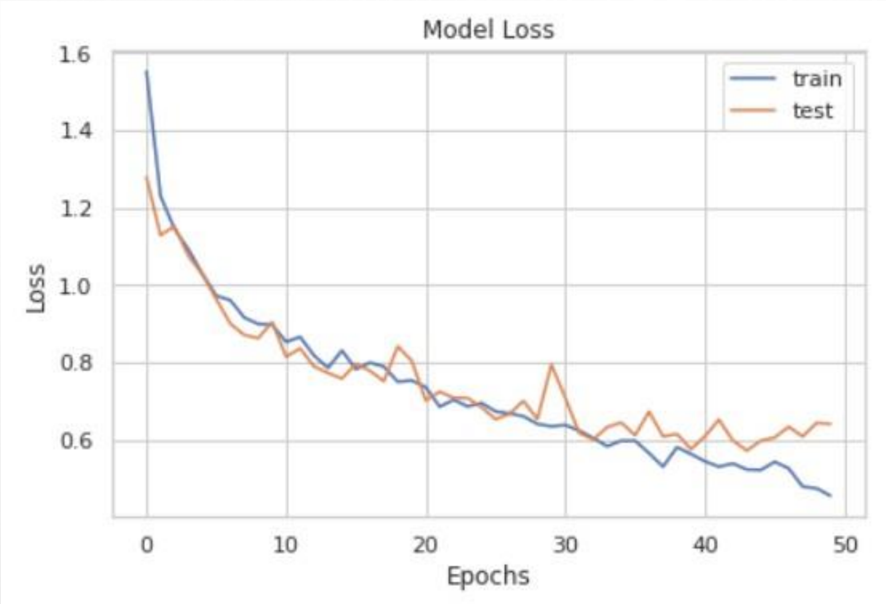
Why

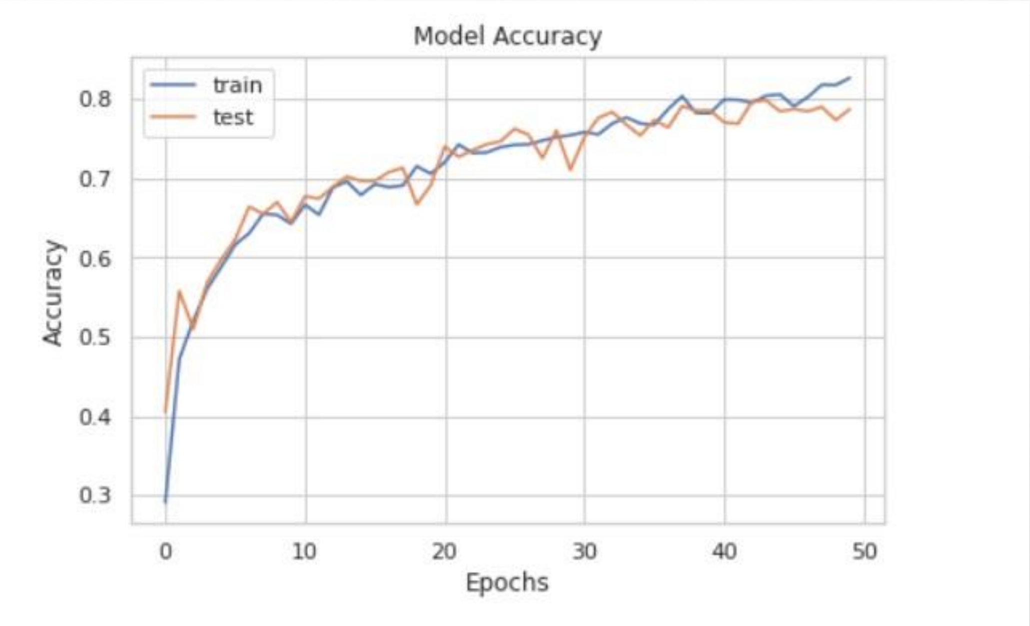
When there are too many hidden layers, the accuracy may decline.

This is because the more hidden layers the Gradient in the Back Propagation algorithm goes through, the smaller it will be and gradually approaches zero.

This phenomenon is called Vanishing Gradient Problem.

Result-4x32

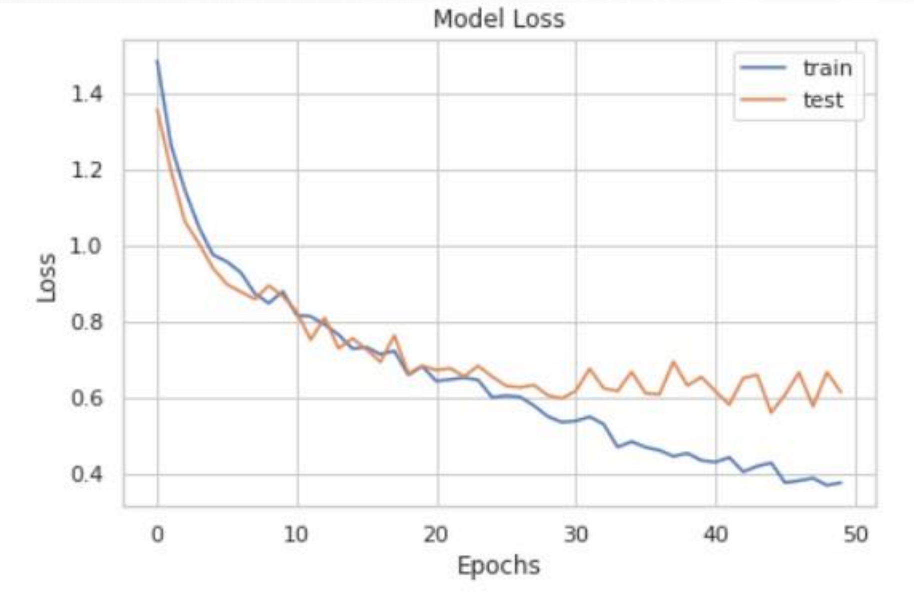


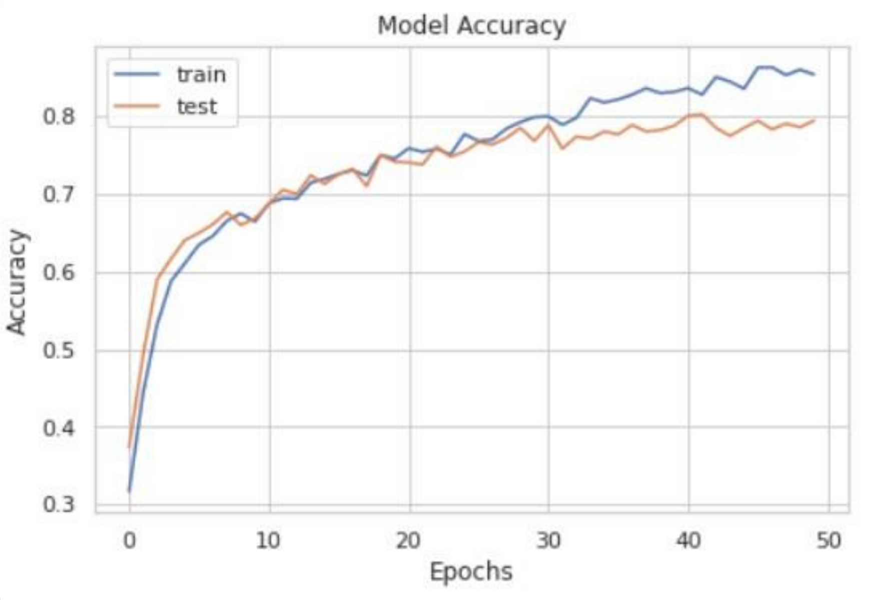


Finally the accuracy on the validation set using the self-laid ConvNet is around 80%.

MODEL DESIGN

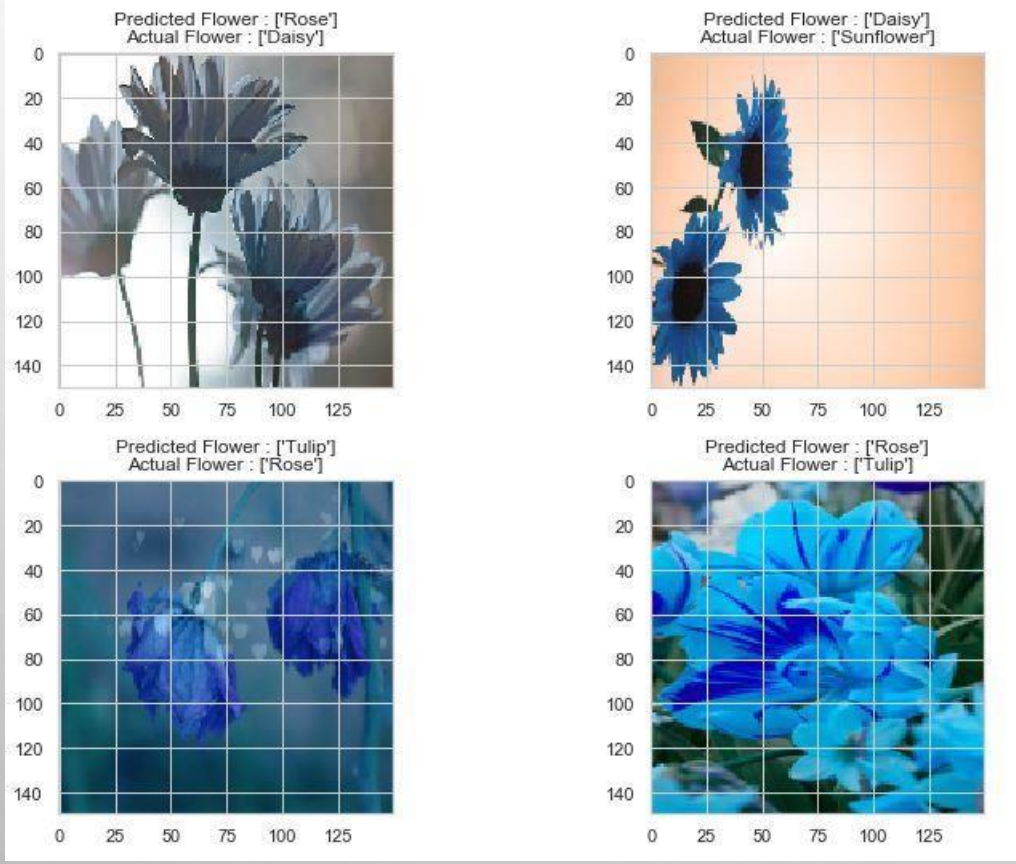
Result-4x64





Finally the accuracy on the validation set using the self-laid ConvNet is around 80%.

Misclassified images of flowers



- The reason why the model misclassified could be because the flowers are not front facing, too big, too small ad so on.