CSCE4604: Practical deep machine Learning

Assignment 3- Report

Submitted by: **Mohammed Abuelwafa**

Student ID: **900172603**

Date: 2th December 2020

Fall 2020

Data Preprocessing:

- The following was done while preprocessing the data for the training :

1. Subtract the mean image from the data set
```
X -= np.mean (X , axis = 0 )
```

2. Divide by 255.0 in order to normalize the pixel values to be between 0 and 1
```
X /= 255.0
```

3. Before calculating the loss, the largest score was subtracted from the scores array in order to avoid numerical explosion when calculating the NLL loss.

4. One hot encoding was used for labelling the labels of the testing data, validation data, training data since it was found more useful in calculating the accuracies and loss (like most of the current frameworks)

Choosing the network Architecure:

Implementation notes:

- Multiple learning rates were tried (0.001 , 0.002 , 0.005 , 0.001 , 0.008). However, 0.005 gave the best results.

- No regularization was used. However, in order to avoid overfitting dropout was implemented with a keep ratio = 85%

- SGD was used and the weights were updated after each epoch with minibatches was used for the best training. However, Adam was implemented as well, but did not give proper results (maybe had an implementation error from my side).

- The ReLU activation function was used in addition to keeping a close eye on the learning rate in addition to softmax function in the last classification layer.

Best training had a value of : learning rate = 5e-3

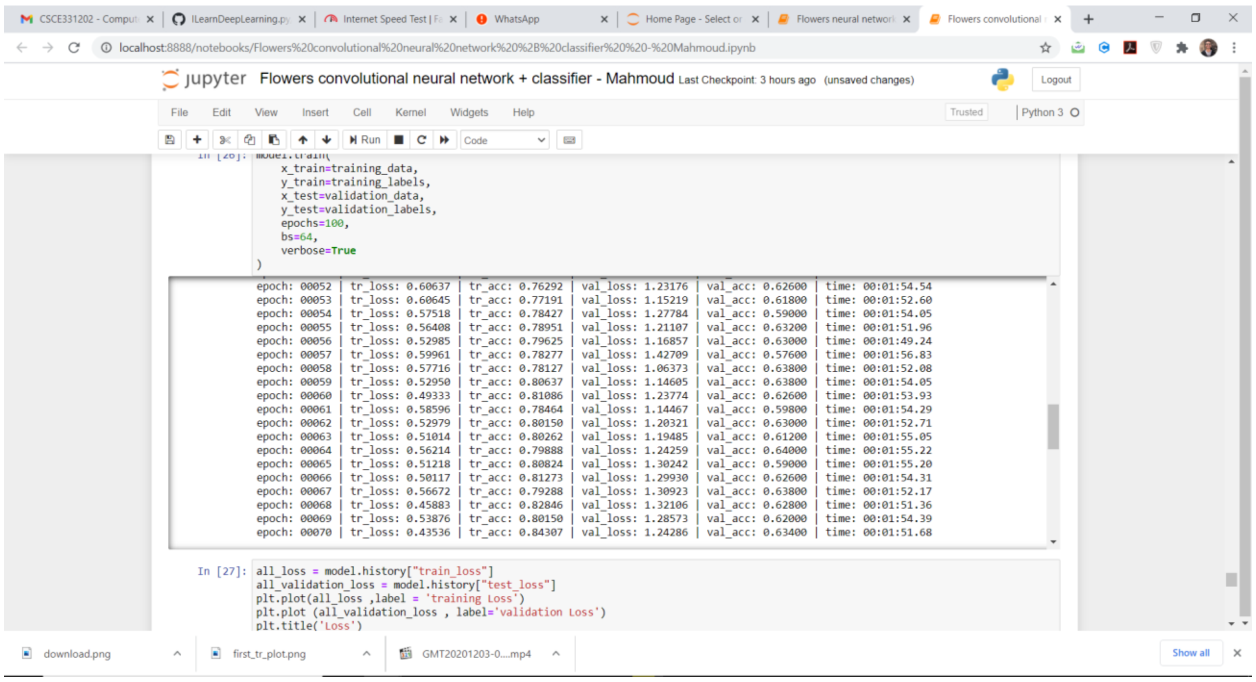
The used architecture:

1. Convolution layer with 32 kernels with kernel size = (3,3,3) , stride = 1 and padding to maintain the size. # input (N, 32, 32, 3) out (N, 32, 32, 32)

2. Relu layer # input (N, 32, 32, 32) out (N, 32, 32, 32)
3. Convolution layer with 32 kernels with kernel size = (3,3,32) , stride = 1 and padding to maintain the size. # input (N, 32, 32, 32) out (N, 32, 32, 32)
4. Relu layer # input (N, 32, 32, 32) out (N, 32, 32, 32)
5. Max pool layer with pooling size = (2,2) and stride = 2 # input (N, 32, 32, 32) out (N, 16, 16, 32)
6. Dropout layer (keep_prob = 85%) # input (N, 16, 16, 32) out (N, 16, 16, 32)
7. Convolution layer with kernel size = (3,3,32) , stride = 1 and padding to maintain the size. # input (N, 16, 16, 32) out (N, 16, 16, 32)
8. Relu layer # input (N, 16, 16, 32) out (N, 16, 16, 32)
9. Max pool layer with pooling size = (2,2) and stride = 2 # input (N, 16, 16, 32) out (N, 8, 8, 32)
10. Convolution layer with kernel size = (3,3,32) , stride = 1 and padding to maintain the size. # input (N, 8, 8, 32) out (N, 8, 8, 32)
11. Relu layer . # input (N, 8, 8, 32) out (N, 8, 8, 32)
12. Max pool layer with pooling size = (2,2) and stride = 2 . # input (N, 8, 8, 32) out (N, 4, 4, 32)
13. Dropout layer (keep_prob = 85%) # input (N, 4, 4, 32) out (N, 4, 4, 32)
14. Convolution layer with kernel size = (3,3,32) , stride = 1 and padding to maintain the size. # input (N, 4, 4, 32) out (N, 4, 4, 32)
15. Relu layer # input (N, 4, 4, 32) out (N, 4, 4, 32)
16. Max pool layer with pooling size = (2,2) and stride = 2 # input (N, 4, 4, 32) out (N, 2, 2, 32)
17. Flatten layer # input (N, 2, 2, 32) out (N, 2* 2* 32)
18. Fully Connected layer # input (N, 2* 2* 32) , out (N , 5)
19. Softmax layer # input (N,5) out (N,5)

Best Training:

Best validation accuracy attained : 67%

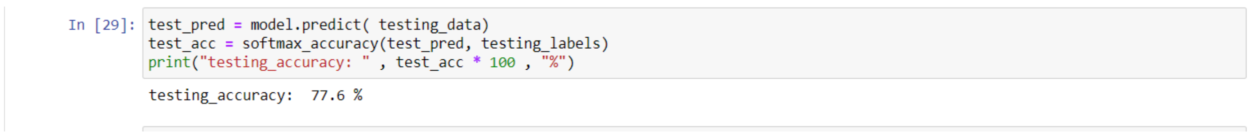

localhost:8888/notebooks/Flowers%20convolutional%20neural%20network%20%2B%20classifier%20%20-%20Mahmoud.ipynb
jupyter Flowers convolutional neural network + classifier - Mahmoud Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
download.png
first_tr_plot.png
GMT20201203-0....mp4
Show all

```
In [26]: model.train(
             x_train=training_data,
             y_train=training_labels,
             x_test=validation_data,
             y_test=validation_labels,
             epochs=100,
             bs=64,
             verbose=True
         )
```

```
epoch: 00052 | tr_loss: 0.60637 | tr_acc: 0.76292 | val_loss: 1.23176 | val_acc: 0.62600 | time: 00:01:54.54
epoch: 00053 | tr_loss: 0.60645 | tr_acc: 0.77191 | val_loss: 1.15219 | val_acc: 0.61800 | time: 00:01:52.60
epoch: 00054 | tr_loss: 0.57518 | tr_acc: 0.78427 | val_loss: 1.27784 | val_acc: 0.59000 | time: 00:01:54.05
epoch: 00055 | tr_loss: 0.56408 | tr_acc: 0.78951 | val_loss: 1.21107 | val_acc: 0.63200 | time: 00:01:51.96
epoch: 00056 | tr_loss: 0.52985 | tr_acc: 0.79625 | val_loss: 1.16857 | val_acc: 0.63000 | time: 00:01:49.24
epoch: 00057 | tr_loss: 0.59961 | tr_acc: 0.78277 | val_loss: 1.42709 | val_acc: 0.57600 | time: 00:01:56.83
epoch: 00058 | tr_loss: 0.57716 | tr_acc: 0.78127 | val_loss: 1.06373 | val_acc: 0.63800 | time: 00:01:52.08
epoch: 00059 | tr_loss: 0.52950 | tr_acc: 0.80637 | val_loss: 1.14605 | val_acc: 0.63800 | time: 00:01:54.05
epoch: 00060 | tr_loss: 0.49333 | tr_acc: 0.81086 | val_loss: 1.23774 | val_acc: 0.62600 | time: 00:01:53.93
epoch: 00061 | tr_loss: 0.58596 | tr_acc: 0.78464 | val_loss: 1.14467 | val_acc: 0.59800 | time: 00:01:54.29
epoch: 00062 | tr_loss: 0.52979 | tr_acc: 0.80150 | val_loss: 1.20321 | val_acc: 0.63000 | time: 00:01:52.71
epoch: 00063 | tr_loss: 0.51014 | tr_acc: 0.80262 | val_loss: 1.19485 | val_acc: 0.61200 | time: 00:01:55.05
epoch: 00064 | tr_loss: 0.56214 | tr_acc: 0.79888 | val_loss: 1.24259 | val_acc: 0.64000 | time: 00:01:55.22
epoch: 00065 | tr_loss: 0.51218 | tr_acc: 0.80824 | val_loss: 1.30242 | val_acc: 0.59000 | time: 00:01:55.20
epoch: 00066 | tr_loss: 0.50117 | tr_acc: 0.81273 | val_loss: 1.29930 | val_acc: 0.62600 | time: 00:01:54.31
epoch: 00067 | tr_loss: 0.56672 | tr_acc: 0.79288 | val_loss: 1.30923 | val_acc: 0.63800 | time: 00:01:52.17
epoch: 00068 | tr_loss: 0.45883 | tr_acc: 0.82846 | val_loss: 1.32106 | val_acc: 0.62800 | time: 00:01:51.36
epoch: 00069 | tr_loss: 0.53876 | tr_acc: 0.80150 | val_loss: 1.28573 | val_acc: 0.62000 | time: 00:01:54.39
epoch: 00070 | tr_loss: 0.43536 | tr_acc: 0.84307 | val_loss: 1.24286 | val_acc: 0.63400 | time: 00:01:51.68
```

```
In [27]: all_loss = model.history["train_loss"]
         all_validation_loss = model.history["test_loss"]
         plt.plot(all_loss ,label = 'training Loss')
         plt.plot (all_validation_loss , label='validation Loss')
         plt.title('Loss')
```

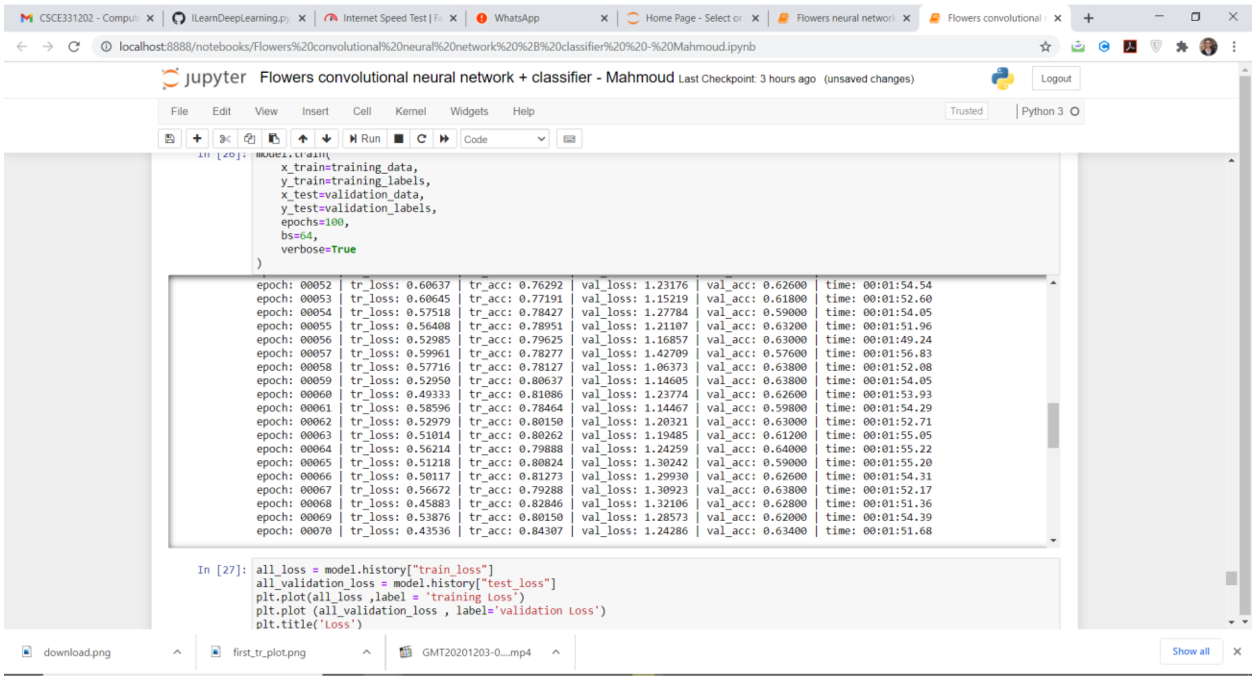
Testing accuracy:



```
In [29]: test_pred = model.predict( testing_data)
         test_acc = softmax_accuracy(test_pred, testing_labels)
         print("testing_accuracy: " , test_acc * 100 , "%")

         testing_accuracy:  77.6 %
```

Stopping the training point :



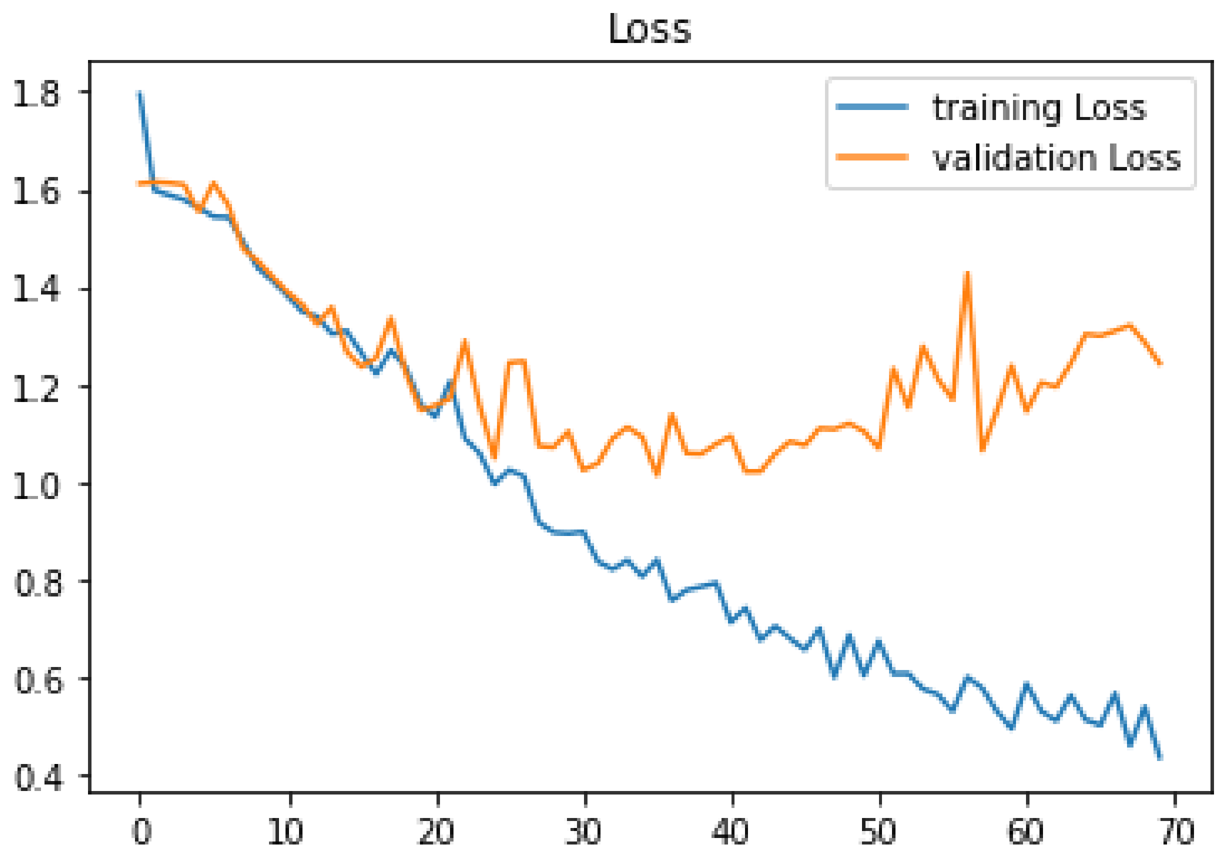
```
In [26]: model.train(
             x_train=training_data,
             y_train=training_labels,
             x_test=validation_data,
             y_test=validation_labels,
             epochs=100,
             bs=64,
             verbose=True
         )
```

```
epoch: 00052 | tr_loss: 0.60637 | tr_acc: 0.76292 | val_loss: 1.23176 | val_acc: 0.62600 | time: 00:01:54.54
epoch: 00053 | tr_loss: 0.60645 | tr_acc: 0.77191 | val_loss: 1.15219 | val_acc: 0.61800 | time: 00:01:52.60
epoch: 00054 | tr_loss: 0.57518 | tr_acc: 0.78427 | val_loss: 1.27784 | val_acc: 0.59000 | time: 00:01:54.05
epoch: 00055 | tr_loss: 0.56408 | tr_acc: 0.78951 | val_loss: 1.21107 | val_acc: 0.63200 | time: 00:01:51.96
epoch: 00056 | tr_loss: 0.52985 | tr_acc: 0.79625 | val_loss: 1.16857 | val_acc: 0.63000 | time: 00:01:49.24
epoch: 00057 | tr_loss: 0.59961 | tr_acc: 0.78277 | val_loss: 1.42709 | val_acc: 0.57600 | time: 00:01:56.83
epoch: 00058 | tr_loss: 0.57716 | tr_acc: 0.78127 | val_loss: 1.06373 | val_acc: 0.63800 | time: 00:01:52.08
epoch: 00059 | tr_loss: 0.52950 | tr_acc: 0.80637 | val_loss: 1.14605 | val_acc: 0.63800 | time: 00:01:54.05
epoch: 00060 | tr_loss: 0.49333 | tr_acc: 0.81086 | val_loss: 1.23774 | val_acc: 0.62600 | time: 00:01:53.93
epoch: 00061 | tr_loss: 0.58596 | tr_acc: 0.78464 | val_loss: 1.14467 | val_acc: 0.59800 | time: 00:01:54.29
epoch: 00062 | tr_loss: 0.52979 | tr_acc: 0.80150 | val_loss: 1.20321 | val_acc: 0.63000 | time: 00:01:52.71
epoch: 00063 | tr_loss: 0.51014 | tr_acc: 0.80262 | val_loss: 1.19485 | val_acc: 0.61200 | time: 00:01:55.05
epoch: 00064 | tr_loss: 0.56214 | tr_acc: 0.79888 | val_loss: 1.24259 | val_acc: 0.64000 | time: 00:01:55.22
epoch: 00065 | tr_loss: 0.51218 | tr_acc: 0.80824 | val_loss: 1.30242 | val_acc: 0.59000 | time: 00:01:55.20
epoch: 00066 | tr_loss: 0.50117 | tr_acc: 0.81273 | val_loss: 1.29930 | val_acc: 0.62600 | time: 00:01:54.31
epoch: 00067 | tr_loss: 0.56672 | tr_acc: 0.79288 | val_loss: 1.30923 | val_acc: 0.63800 | time: 00:01:52.17
epoch: 00068 | tr_loss: 0.45883 | tr_acc: 0.82846 | val_loss: 1.32106 | val_acc: 0.62800 | time: 00:01:51.36
epoch: 00069 | tr_loss: 0.53876 | tr_acc: 0.80150 | val_loss: 1.28573 | val_acc: 0.62000 | time: 00:01:54.39
epoch: 00070 | tr_loss: 0.43536 | tr_acc: 0.84307 | val_loss: 1.24286 | val_acc: 0.63400 | time: 00:01:51.68
```

```
In [27]: all_loss = model.history["train_loss"]
         all_validation_loss = model.history["test_loss"]
         plt.plot(all_loss ,label = 'training Loss')
         plt.plot (all_validation_loss , label='validation Loss')
         plt.title('Loss')
```

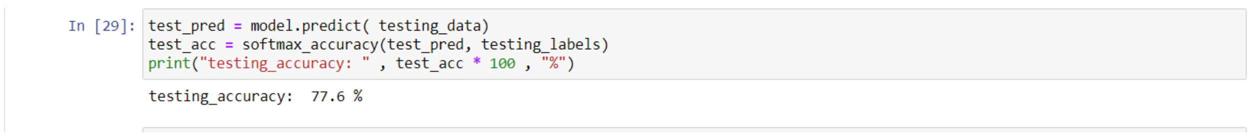
- The training stopped here due to multiple signs of overfitting (increasing validation loss and decreasing training loss) in addition to limited time factors.

Best Training Loss vs epochs Graph:



Best ACCR:

An Accuracy of : 77.6% was obtained. (in comparison to 27.8% in the KNN Classifier and 55.4 with the fully connected)



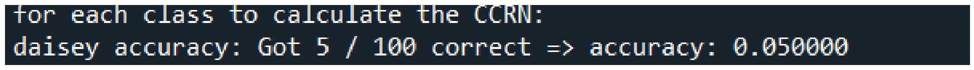
```
In [29]: test_pred = model.predict( testing_data)
         test_acc = softmax_accuracy(test_pred, testing_labels)
         print("testing_accuracy: " , test_acc * 100 , "%")

         testing_accuracy:  77.6 %
```

Correct classification rate (CCRn) comparison:

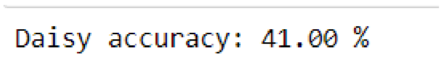
For Daisy:

KNN : 5%



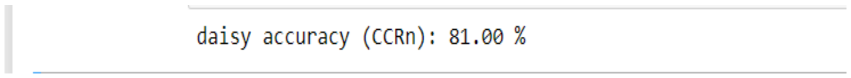
```
for each class to calculate the CCRN:
daisey accuracy: Got 5 / 100 correct => accuracy: 0.050000
```

Neural network: 41%



```
Daisy accuracy: 41.00 %
```

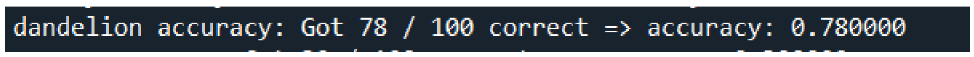
CNN : 81%



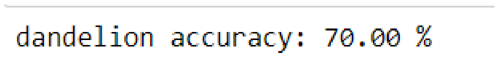
```
daisy accuracy (CCRn): 81.00 %
```

For Dandelion:

KNN : 78%



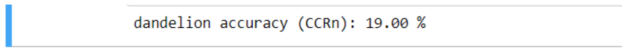
```
dandelion accuracy: Got 78 / 100 correct => accuracy: 0.780000
```



```
dandelion accuracy: 70.00 %
```

Neural network: 70%

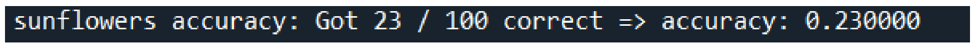
CNN : 19%



```
dandelion accuracy (CCRn): 19.00 %
```

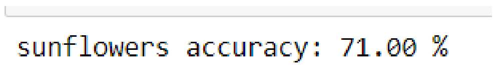
For Sunflowers:

KNN : 23%

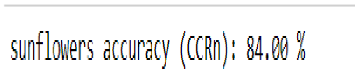


```
sunflowers accuracy: Got 23 / 100 correct => accuracy: 0.230000
```

Neural network: 71 %



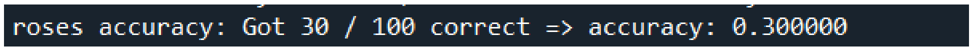
```
sunflowers accuracy: 71.00 %
```



```
sunflowers accuracy (CCRn): 84.00 %
```

CNN : 84%

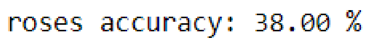
For Roses:



```
roses accuracy: Got 30 / 100 correct => accuracy: 0.300000
```

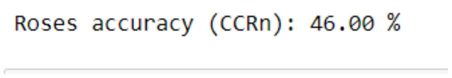
KNN : 30%

Neural network: 38%



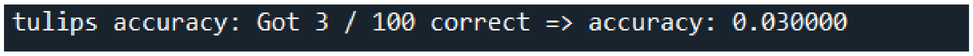
```
roses accuracy: 38.00 %
```

CNN : 46%



```
Roses accuracy (CCRn): 46.00 %
```

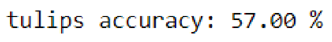
For Tulips:



```
tulips accuracy: Got 3 / 100 correct => accuracy: 0.030000
```

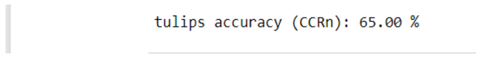
KNN : 3%

Neural network: 57%



```
tulips accuracy: 57.00 %
```

CNN:



```
tulips accuracy (CCRn): 65.00 %
```

The CNN generally performed better than the KNN classifier and the fully connected neural network in both the ACCR and the CCRn.

➔ The submitted code output is not for the best training. Multiple attempts were performed. However, the best training output was lost.