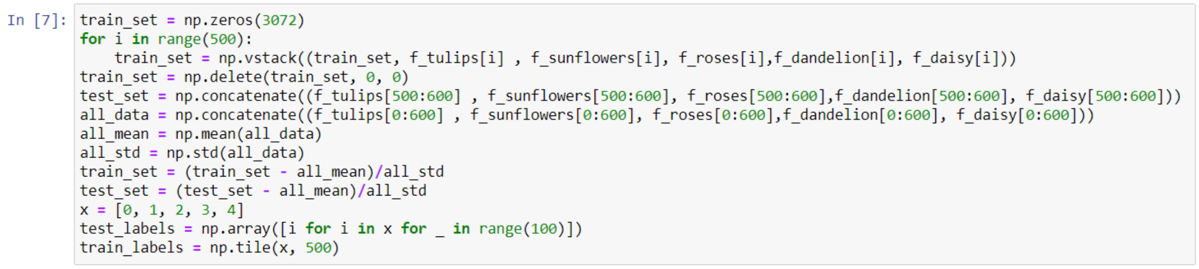
Assignment 3 Report

By Ahmed Aboutaleb

1. Source code of your implementation, including the random seed (to reproduce your results) [3%]

Notebook attached.

2. Explanation of any data-preprocessing you have used.



```
In [7]: train_set = np.zeros(3072)
        for i in range(500):
            train_set = np.vstack((train_set, f_tulips[i] , f_sunflowers[i], f_roses[i],f_dandelion[i], f_daisy[i]))
        train_set = np.delete(train_set, 0, 0)
        test_set = np.concatenate((f_tulips[500:600] , f_sunflowers[500:600], f_roses[500:600],f_dandelion[500:600], f_daisy[500:600]))
        all_data = np.concatenate((f_tulips[0:600] , f_sunflowers[0:600], f_roses[0:600],f_dandelion[0:600], f_daisy[0:600]))
        all_mean = np.mean(all_data)
        all_std = np.std(all_data)
        train_set = (train_set - all_mean)/all_std
        test_set = (test_set - all_mean)/all_std
        x = [0, 1, 2, 3, 4]
        test_labels = np.array([i for i in x for _ in range(100)])
        train_labels = np.tile(x, 500)
```

- Shuffled the training set each epoch in order to make sure that the model does not memorize a certain pattern.
- The mean and the standard deviation for all the features was calculated and was used to normalize the data using the equation:

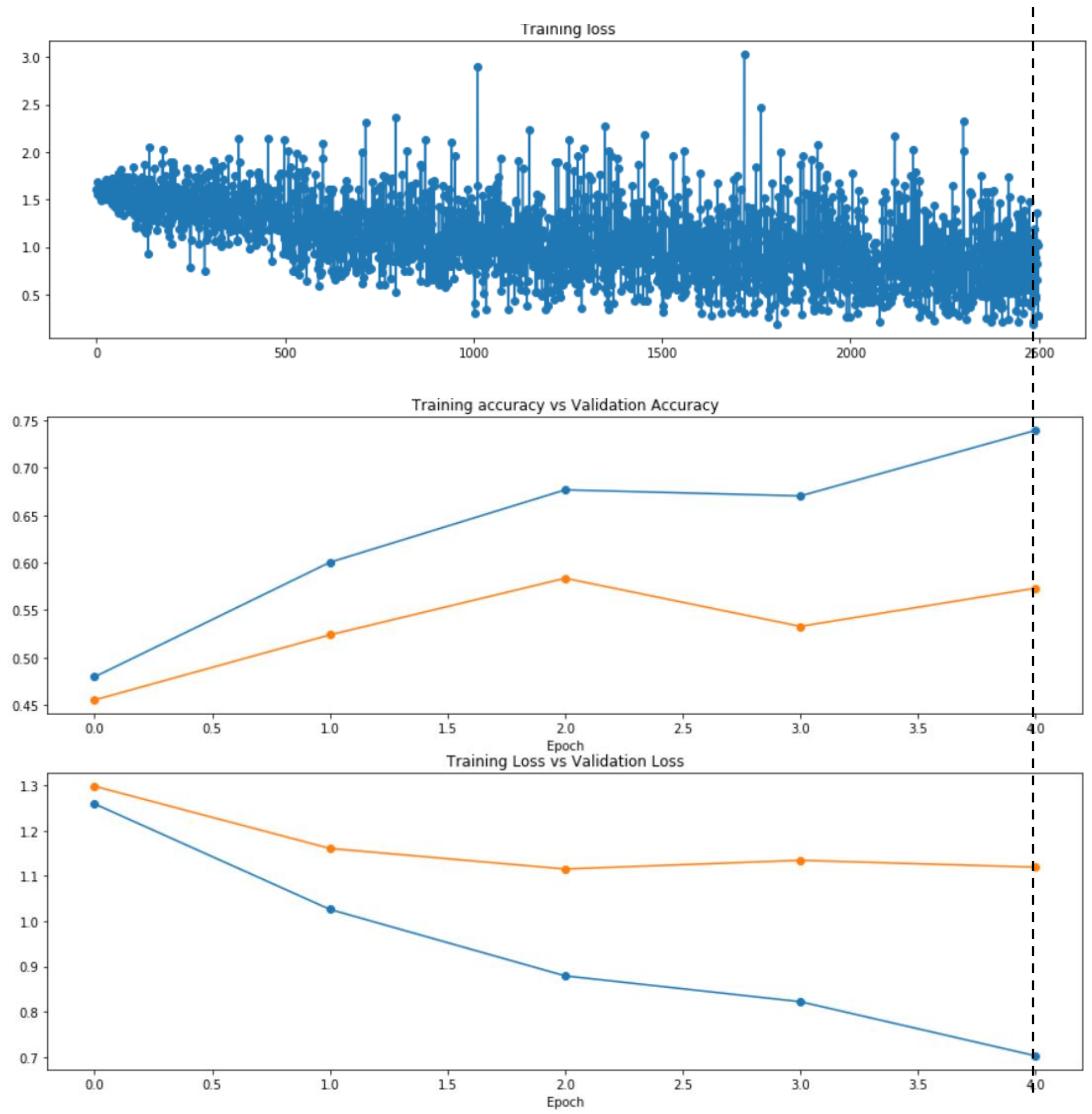
$$X = \frac{X - \mu}{\sigma}$$

3. Explanation of how did you choose your network architecture and how did you fine tune your hyper-parameters, e.g., learning rate, regularization, etc...(explanation associated with some numerical evidence is preferred) [1%]

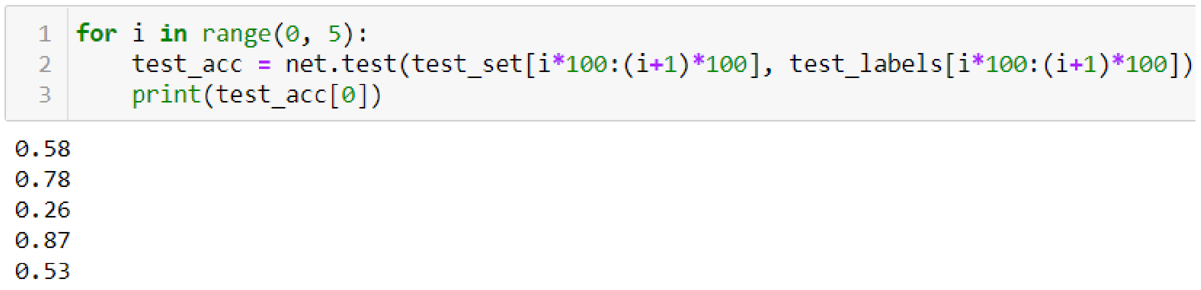
1. I used Convolutional, maxpool, and ReLU layers, followed by a fully connected classifier.
2. Created a function called conv, for convolutions, and that was used in both the forward and the backward paths. This function was used in both the backward and the forward paths.
3. In order to use the conv function, some dimesionality isssues had to be dealt with through transposing the matrices used in the backward propagation.
4. The most important thing to note is that I have worked around creating a separate “full convolution” function by making sure that I always pad to maintain size.
5. Adding more layers was not yielding any improvement in the accuracy, while it was just overcomplicating the model and requiring it to take more time to work.
6. Finally, the parameters like the learning rate and the reg loss where both doing the same effect, which in my opinion does not really affect the performance of the mode in this case. Decreasing any of them just made the model reach the same state but slower, which gives me the chance to stop the training at the right time. However, one point that is worth mentioning is that in this problem, given this dataset, the loss for the validation set did not increase even when the model overfits the training data, which shows that regularization is not giving any benefit here.
7. Different batch sizes where tried, but no big difference was observed.

4. A plot of your best training and validation losses (Y axis) versus number of epochs (X axis) showing when did you stop training and why. [1%]

```
net.train(100, 50, 0.001, train_set, train_labels)
```



5. Using the testing set, report your Correct Classification Rate of each of the 5 classes separately. (compare against K-NN classifier from assignment 1). [1%]

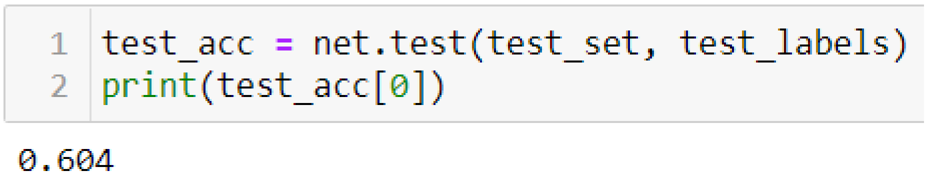


```
for i in range(0, 5):
    test_acc = net.test(test_set[i*100:(i+1)*100], test_labels[i*100:(i+1)*100])
    print(test_acc[0])
```

```
0.58
0.78
0.26
0.87
0.53
```

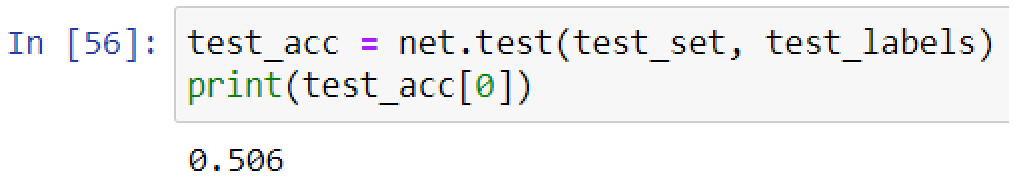
	KNN	FCNN	CNN
Tulips	8%	28%	58%
Sunflower	64%	77%	78%
Roses	21%	48%	26%
Dandelion	17%	40%	87%
Daisies	35%	60%	53%

6. Average Correct Classification Rate (ACCR) using the testing set. [4% competitive. Best student will receive 4% and others will get credit relative to how far their predictions are from the best one].¶



```
test_acc = net.test(test_set, test_labels)
print(test_acc[0])
```

```
0.604
```



```
In [56]: test_acc = net.test(test_set, test_labels)
         print(test_acc[0])
```

```
0.506
```

ACCR = 60.4%