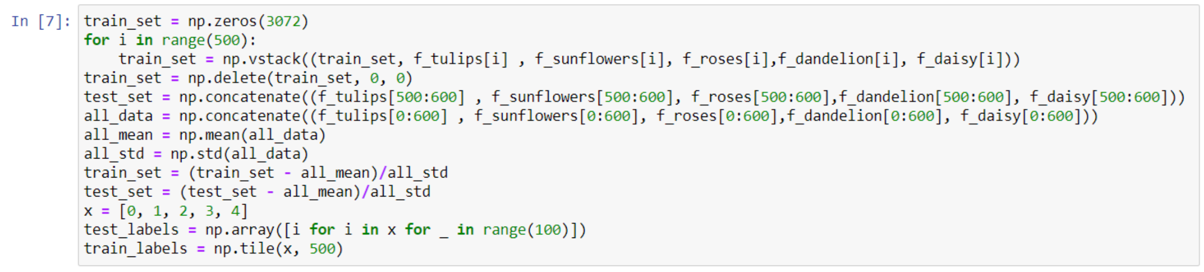
Assignment 2 Report

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)
```

- Made sure that the training dataset is very even to make sure that all the classes get an equal chance of manipulating the wight matrices. In order to achieve this, the training set was a repeated sequence [tulip, sunflowers, roses, dandelion, daisy, tulip, sunflowers, roses, dandelion, daisy……….].
- The mean and the standard deviation for all the features was calculated and was used to normalize the data using the equation:

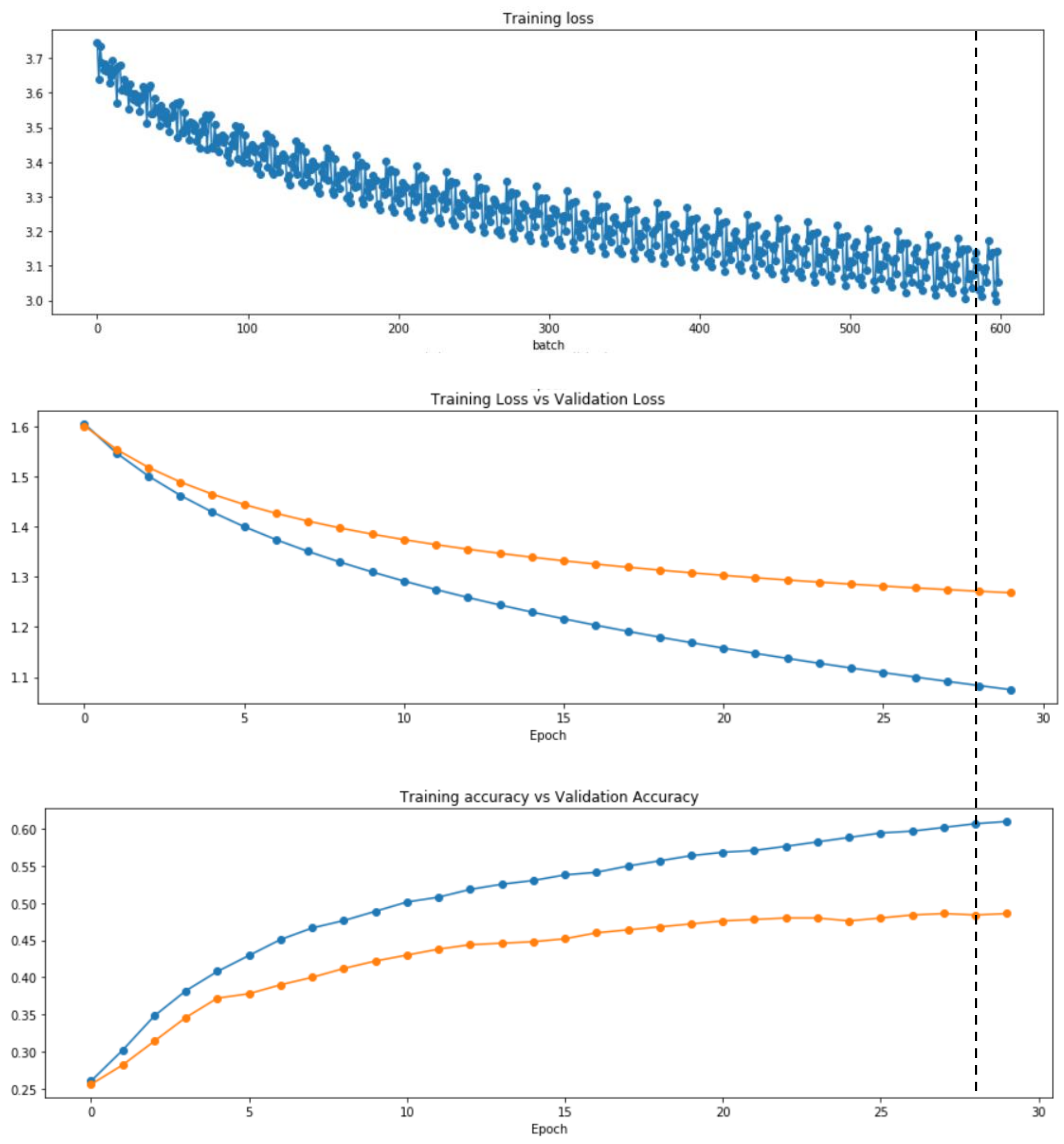
$$X = \frac{X - \mu}{\sigma}$$

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%]

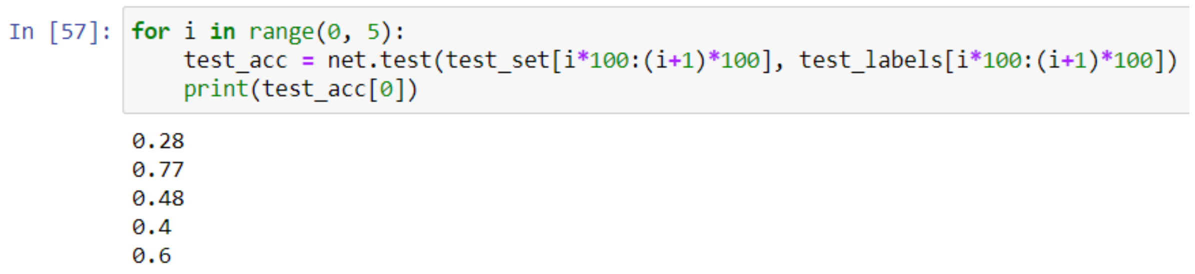
1. I tried first 0 hidden layers which obviously gave the lowest accuracy (30%) due to linearity.
2. Then for 1 hidden layer I tried several number of neurons (500, 1000, and 2000) and every time the number of neurons increased, the accuracy was slightly increased.
3. After adding another layer, the accuracy was increased further until it reached 50%+
4. 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.
5. 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.
6. I kept decreasing the learning rate to make sure that the model does not overfit, and I increased the number of epochs.
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%]

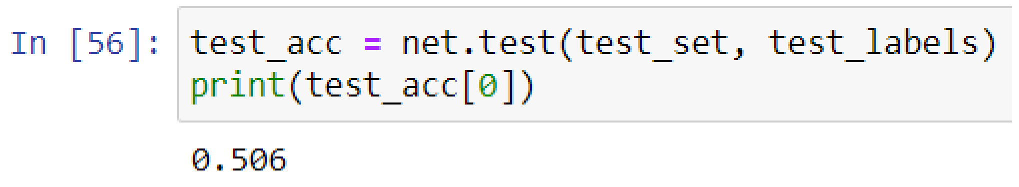


```
In [57]: for i in range(0, 5):
             test_acc = net.test(test_set[i*100:(i+1)*100], test_labels[i*100:(i+1)*100])
             print(test_acc[0])

         0.28
         0.77
         0.48
         0.4
         0.6
```

	KNN	FCNN
Tulips	8%	28%
Sunflower	64%	77%
Roses	21%	48%
Dandelion	17%	40%
Daisies	35%	60%

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].¶



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

         0.506
```

ACCR = 50.6%