| i | Label | Hypothesis 1 |  |  |  | Hypothesis 2 |  |  |  |
|---|---|---|---|---|---|---|---|---|---|
|  |  | $D_0$ | $f_1 \equiv$ $[x >$__ 2] | $f_2 \equiv$ $[y >$__5] | $h_1 \equiv$ [ ______f1] | $D_1$ | $f_1 \equiv$ $[x >$__10] | $f_2 \equiv$ $[y >$__11] | $h_2 \equiv$ [ ______f2] |
| (1) | (2) | (3) | (4) | (5) | (6) | (7) | (8) | (9) | (10) |
| 1 | $-$ | 0.1 | - | + | - | 0.0625 | - | - | - |
| 2 | $-$ | 0.1 | - | - | - | 0.0625 | - | - | - |
| 3 | $+$ | 0.1 | + | + | + | 0.0625 | - | - | - |
| 4 | $-$ | 0.1 | - | - | - | 0.0625 | - | - | - |
| 5 | $-$ | 0.1 | - | + | - | 0.0625 | - | + | + |
| 6 | $-$ | 0.1 | + | + | + | 0.25 | - | - | - |
| 7 | $+$ | 0.1 | + | + | + | 0.0625 | + | - | - |
| 8 | $-$ | 0.1 | - | - | - | 0.0625 | - | - | - |
| 9 | $+$ | 0.1 | - | + | - | 0.25 | - | + | + |
| 10 | $+$ | 0.1 | + | + | + | 0.0625 | - | - | - |

Table 1: Table for Boosting results

To get $\alpha_0$, we use the weighted sum of mistakes in cases where h = $x_1$ and h = $x_2$. When h = $x_1$ $\epsilon$ = 0.2 and h = $x_2$ $\epsilon$ = 0.3. Thus putting $\epsilon$ = 0.2 in the formula (for log base 2):

$$\alpha_0 = 0.5log\frac{(1-\epsilon)}{\epsilon}$$

we get $\alpha_0$ = 1.
Now we can compute the new distribution using $\alpha_0$:
Plugging it in the formula, we get $D_1$(i) = $\frac{1}{20Z_0}$ if h($x_i$ ) $\neq y_i$ and $D_1$(i) = $\frac{1}{5Z_0}$ if h($x_i$) = $y_i$. Now we count the number of cases in both scenarios to calculate $Z_0$

$$0.4Z_0 + 0.4Z_0 = 1$$
$$Z_0 = 0.8$$

Then $D_1$(i) = $\frac{1}{16}$ if if h($x_i$ ) $\neq y_i$
Then $D_1$(i) = $\frac{1}{4}$ if if h($x_i$ ) = $y_i$
$\epsilon_{f1}$ = 0.25 + 2 (1/16) = 0.375
$\epsilon_{f2}$ = 0 + 4 (1/16) = 0.25
So using $\epsilon_{f2}$ to calculate $\alpha_1$:

$$\alpha_1 = 0.5log\frac{(1-0.25)}{0.25}$$
$$\alpha_1 = 0.8$$

Then $H_{final}$ = sign( 1 $\times$ [x > 2] + 0.8 $\times$ [y > 11])

Question 2 - Part A: 1. Very simply w = (-1,-1) and bias = 0 separates the data.

2. w = (-0.5, -0.5) and bias = 0

3. We find two closest points to each other from different classes. These are point 1 and point 6. So the separating hyperplane will be perpendicular to the line connecting points 1,6 and it passes through the midline of points 1 and 6. The slope of this hyperplane will be -1 and since we know it passes through the midpoint of points 1 and 6, we can calculate that the bias is 0.

Question 2 - Part B:

1. I = [1,6]

2. $\alpha$ = [0.25,0.25]

3. Objective Function Value = 0.25

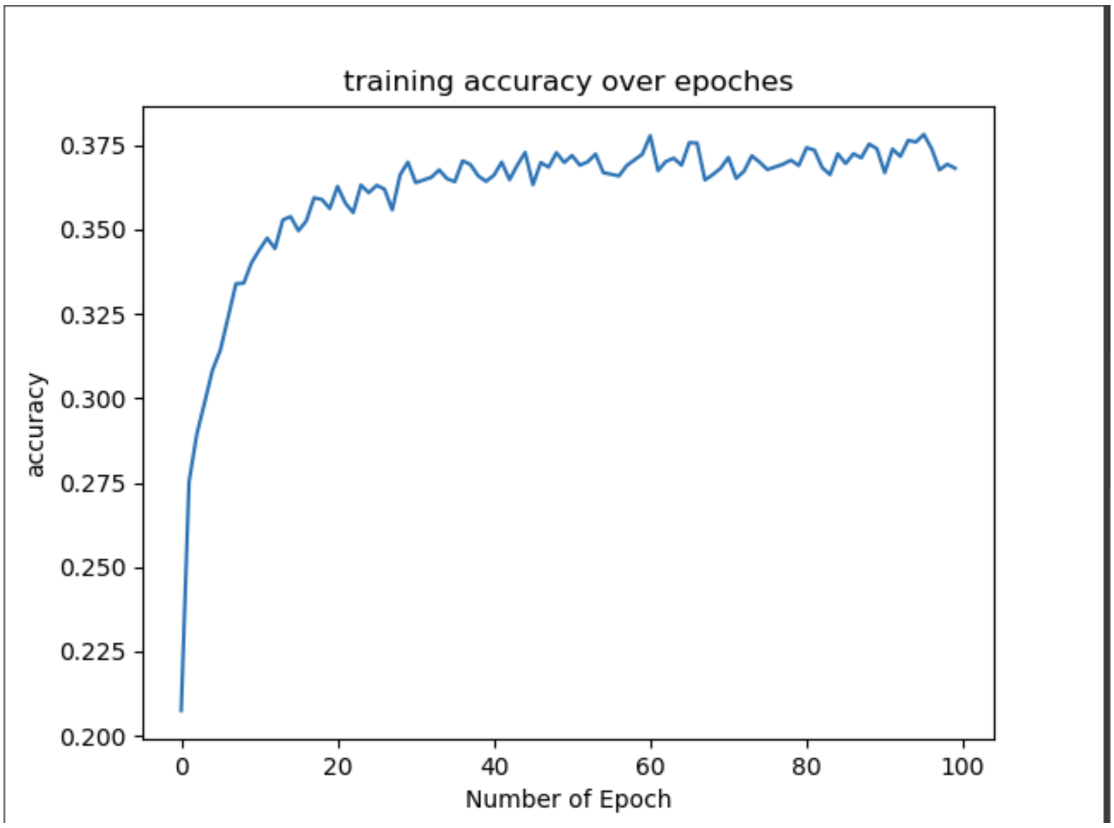
Question 2 - Part C:

In a SVM we search for two things: a hyperplane with the largest minimum margin, and a hyperplane that correctly separates as many instances as possible. TC is a regularization parameter which determines the trade off between these two. A large C will end up in a narrow margin, because if we find a hyperplane that does a better job of getting all the training points classified correctly, the optimization will choose that smaller margin hyperplane. With this in mind, we can say that when C equals to infinity we have a hard margin that perfectly separates the data. On the other hand a C value less than 1 will cause the optimizer to look for a larger-margin separating hyperplane, even if that hyperplane mis-classifies more points.

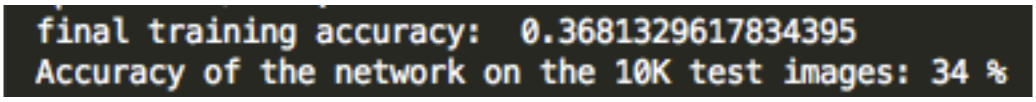
**ALI KOZLU**
**Pennkey: akozlu**
**HW4 Report**

**Experiment 1:**

*Training Accuracy over epochs:*



*Final Training and Test Accuracy:*

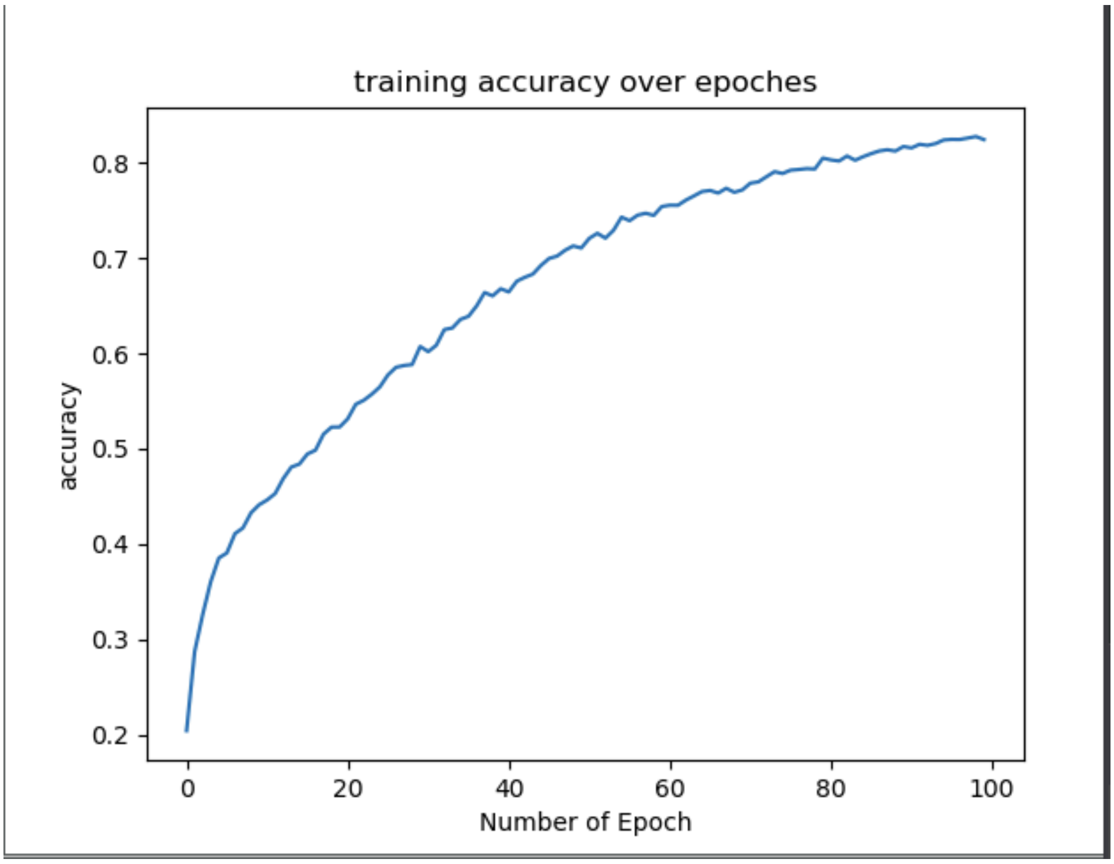


```
final training accuracy:  0.3681329617834395
Accuracy of the network on the 10K test images: 34 %
```

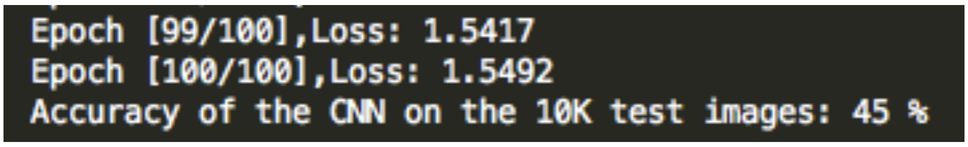
For the feed forward neural network, we receive a loss drop from ~2.25 to ~2,which allows us to do better than random guessing. Since random guessing would return %10 accuracy, the fact that training accuracy and test accuracy are around %34 shows us that the network is actually learning something.

**Experiment 2: CNN**

*Training Accuracy over epochs:*



Test Accuracy:

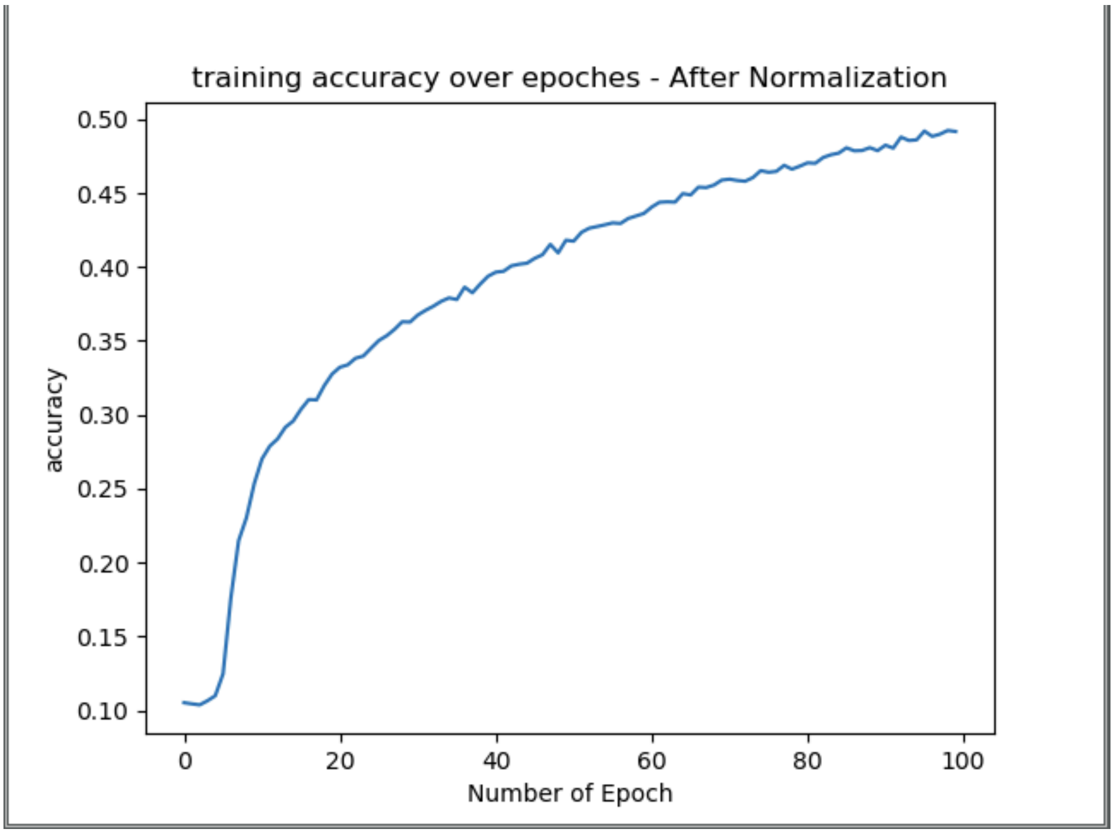


```
Epoch [99/100],Loss: 1.5417
Epoch [100/100],Loss: 1.5492
Accuracy of the CNN on the 10K test images: 45 %
```

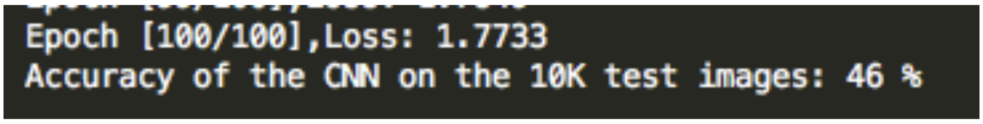
My final training accuracy for CNN was around %80-81, but the test accuracy was %45. The improved training and test accuracy show that feed forward neural network is not as sophisticated as the convolutional neural network. So CNN was able to find more expressive models as the loss went down to nearly ~1.5. However the fact that training accuracy was very high compared to test accuracy showed us that our model was over fitting.

**Experiment 3: Normalized CNN**

*Training Accuracy over epochs:*



*Test Accuracy:*



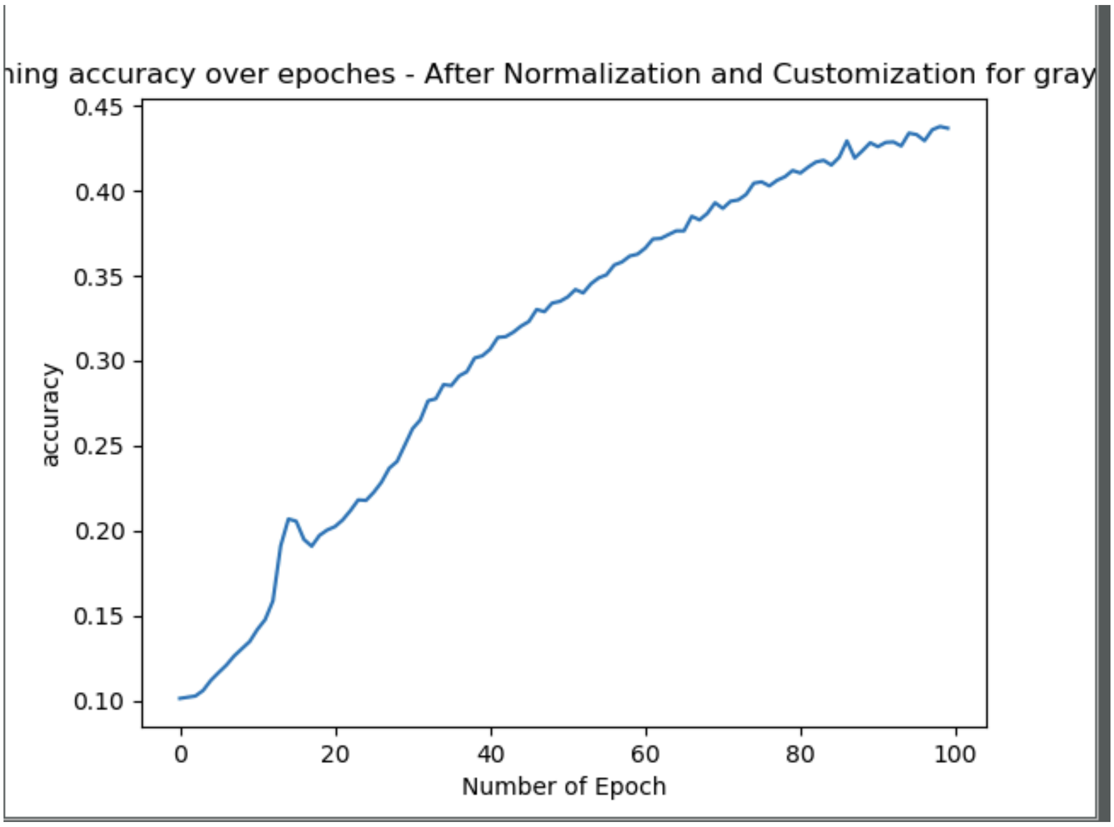
```
Epoch [100/100],Loss: 1.7733
Accuracy of the CNN on the 10K test images: 46 %
```

We observe that our test accuracy stayed the same whereas training accuracy dropped to ~%45-50. This can be considered an improvement since our model is not over fitting anymore. Thus getting all of our data on the same scale increased our ability to learn. This could be because standardized features values implicitly weights all features equally in their representation or it reduces the chances of getting stuck in a local optima.

**Additional Experiment 3: CNN with Gray scale and Normalized inputs**

To increase improvement, I experimented with normalized gray scale images. The (32,32,3) colored image was changed to a 2d 32x32 matrix before it was normalized.

The only thing I changed in CNN architecture was to change the number of inputs from 3 to 1. The rest stayed the same. My hypothesis was that processing gray scale images would reduce first layer complexity, thereby decreasing error rate.



The results showed that training accuracy dropped to %43 whereas test accuracy dropped to %42. So processing gray scale images actually did not improve our model. A possible explanation is that the epoch size should be increased to reach the same testing accuracy because the loss did not reach local minima after 100 epochs. I designed another custom CNN architecture to specifically train grayscale images. It is called Custom_CNN in the script. I experimented with different activation functions such hyperbolic tangent sigmoid and tried increasing number of convolutional layers. In the end I came up with a structure with 3 hidden layers and 2 fully connected layers.

**Experiment 4: Hyper parameter Tuning**

My tuning focused on two approaches. First approach was to tune batch size, learning rate and kernel size of our original CNN model. Second approach was to design a new CNN architecture, which I did under Custom_CNN.
The following parameter changes on our original CNN gave %58 training accuracy and %52 test accuracy, which is a %12 improvement on our Experiment 3 test accuracy and %14 improvement on our Experiment 3 training accuracy.
Batch Size: 128
Kernel: 3x3

Learning Rate: 0.03
Epochs: 200

On the other hand, other approach was to come up with a new CNN architecture to train gray scale and normalized images. I experimented with different number of convolutional layers and different input, output scenarios for fully connected layers. I found some papers online that were explaining how to train a soft max classifier in cases where the last fully connected layer is applied the soft max function and the output layer uses this "trained" soft max classifier to make better predictions.[1] I got stuck when I couldn't figure out how to apply a soft max function to 10 dimensions (our labels). So the custom CNN includes an additional convolutional layer and only includes one fully connected layer which takes 16 feature maps of size 1x1 as input (after multiple pooling and kernel operations). This architecture reached the around similar accuracy with Experiment 4 after 500 epochs.

---

[1]

Zeiler, M.D. andFergus, R. (2013) "Visualizing and Understanding Convolutional Networks", Cornell University Library, arXiv:1311.2901v3

▪