ELL409 Assignment 2

GitHub Repo:

Amogh Agrawal
2018EE10441

Shauryasikt Jena
2018EE10500

***Abstract*—This is a report documenting our team playing around with Support Vector Machines and Neural Networks on varied datasets ranging from a small binary classification problem to practical datasets like MNIST, SVHN and Tiny Imagenet. We further implemented various Convolutional Neural Network architectures using Pytorch apart from the Fully Connected Neural Network made using numpy.**

***Index Terms*—SVM, Neural Network, CNN, MNIST, SVHN, Tiny ImageNet**

I. Support Vector Machines

Support Vector Machines have been popular algorithms in the past for classification problems (before neural networks were made feasible by increasing data sizes). They still show important properties about Machine Learning and thus we study their use on a small binary classification dataset: The Health Dataset.

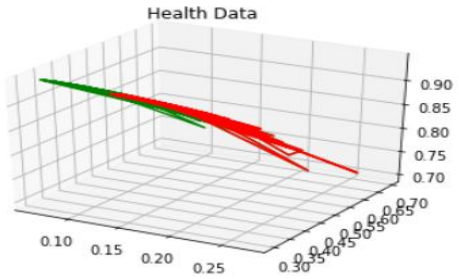

Fig. 1. 3D Dataset Plot

A. Convex Optimisation

We were allowed to use Scikit-learn for implementing for SVM, or libSVM for convex optimization purposes to ascertain the support vectors. However, Scikit-learn does not emphasize the maths behind its working and libSVM documentation is not convenient for comprehension, we used CVXOPT library for our calculations. Going further, we tested the SVM with various kernels.

1) Linear Kernel: Linear Kernel is the most primitive kernel to be used and as expected its performance is not noteworthy. At a testing accuracy of merely 60%, we can conclude that the given data is not linearly separable and hence this kernel is not suited.

```
Train Accuracy: 0.6666666666666666
Test Accuracy: 0.6
```

Fig. 2. SVM with Linear Kernel

2) Polynomial Kernel: It is more advanced than the linear kernel and hence as a rule should show better accuracy on given dataset. Upon observation, we can conclude that the kernel with degree=2 is not appreciably suited whereas kernels with degrees 3 and 4 showed considerable accuracy.

```
Degree 2 Train Accuracy: 0.4984126984126984
Degree 2 Test Accuracy: 0.6
Degree 3 Train Accuracy: 0.7507936507936508
Degree 3 Test Accuracy: 0.7571428571428571
Degree 4 Train Accuracy: 0.7666666666666667
Degree 4 Test Accuracy: 0.7714285714285715
```

Fig. 3. SVM with Polynomial Kernels with varied degrees

3) Gaussian Kernel: This is the most advanced kernel that we tried, and best suited to this dataset as evidenced in Assignment 1. The accuracy recorded for this kernel was the highest among all above. For the most optimal C and γ values, we did a grid search from 0.5 to 1.5 and 1e-3 to 1e-2 respectively.

```
Most optimal C value: 0.5
Most optimal gamma value: 0.001
Train Accuracy: 0.7714285714285715
Test Accuracy: 0.7714285714285715
```

Fig. 4. SVM with Gaussian Kernel with best parameters after grid search

B. The Best Kernel

We notice that the polynomial kernel with degree 4 and the Gaussian kernel are in close contention. However, the Gaussian Kernel has better training accuracy, almost as good as testing. The test accuracy being higher for the degree 4 polynomial kernel is not desired generally, so we move on ahead with the Gaussian Kernel with C=0.5 and $\gamma = 0.001$. The dataset was reduced to 2D by PCA for ease of plotting.

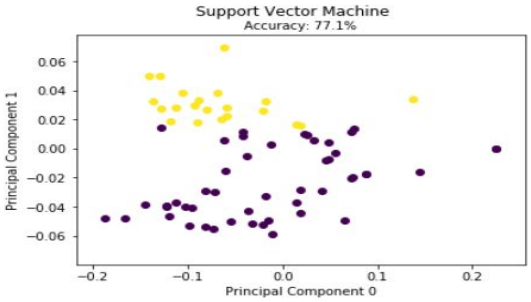

Fig. 5. SVM scatter-plot with Gaussian Kernel

C. Decision Boundary

After running the SVM on the best performing kernel, we plot the decision boundary as shown (without support vectors).

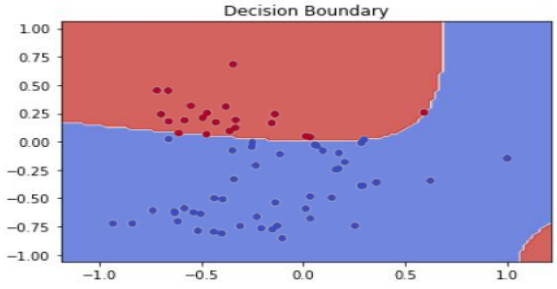

Fig. 6. Decision Boundary with Support Vectors

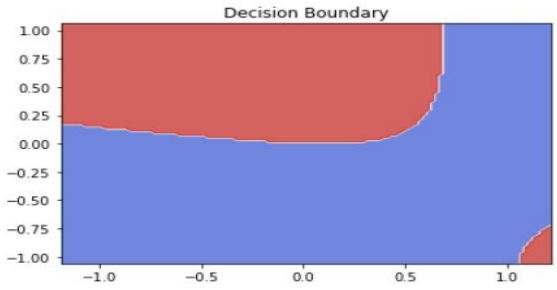

Fig. 7. Decision Boundary without Support Vectors

D. Self-implemented SMO

We iteratively calculated the support vectors by convex optimization here. On comparing times, our implementation is about 2 times slower than the automated optimization by CVXOPT.

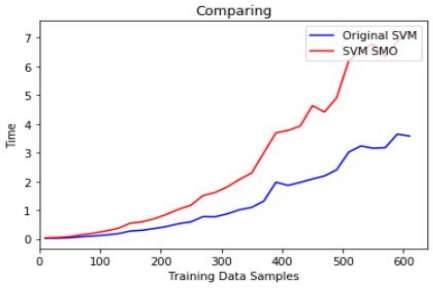

Fig. 8. Original CVXOPT SVM vs self-written SMO SVM

II. Neural Networks

Neural Networks have become ubiquitous in their use for solving complicated Machine Learning problems involving large amounts of data due to their universal modelling properties. We explore the Fully Connected Neural Network as well as convolutional architectures in this section by training on the Tiny ImageNet dataset and the MNIST dataset.

A. Fully Connected Neural Network

In this section, we trained a shallow neural network (one hidden layer) on the MNIST data set. Hyper-parameter tuning (by trial and error mostly) got us to use 30 hidden layers and a learning rate of 0.5. We used a cross entropy loss function for better learning and generalization and further looked at the effect of regularization by implementing L1, L2, dropout, early stopping and batch-normalization. We run all algorithms for 50 epochs for a batch size of 10.

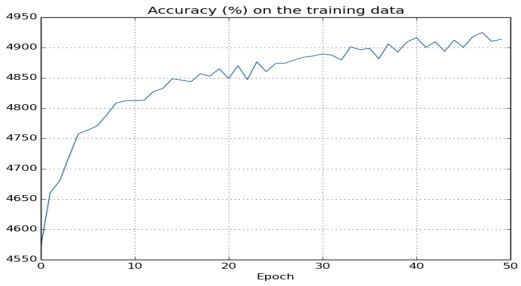

Fig. 9. Unregularized Neural Network Training Accuracy

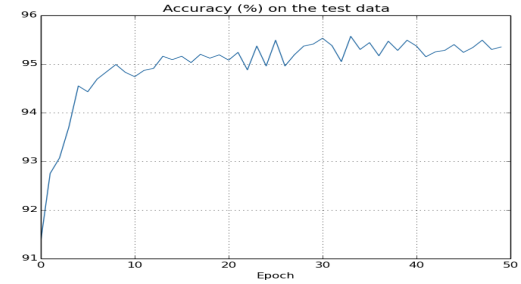

Fig. 10. Unregularized Neural Network Test Accuracy

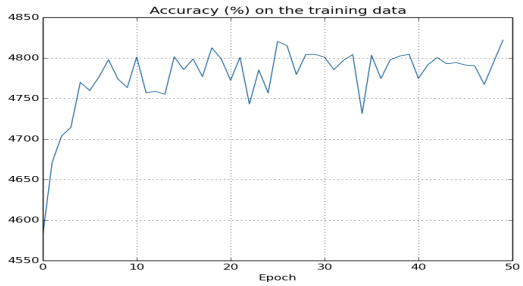

Fig. 11. L1 Regularized Neural Network Training Accuracy

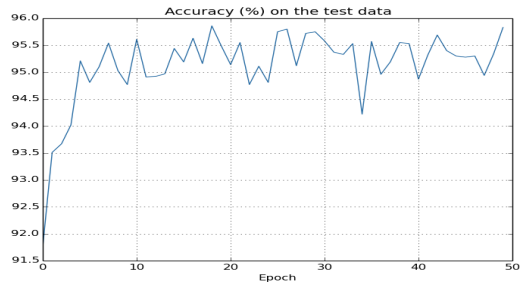

Fig. 12. L1 Regulairzed Neural Network Test Accuracy

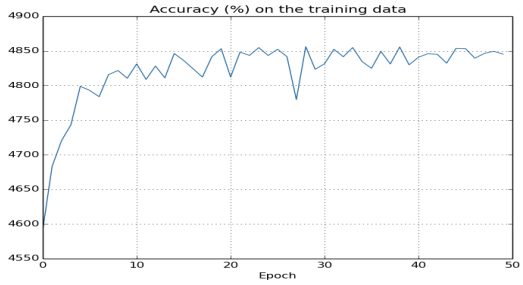

Fig. 13. L2 Regulairzed Neural Network Training Accuracy

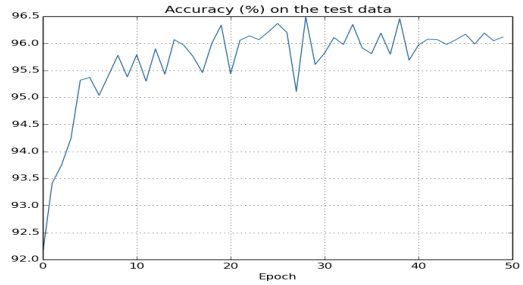

Fig. 14. L2 Regulairzed Neural Network Test Accuracy

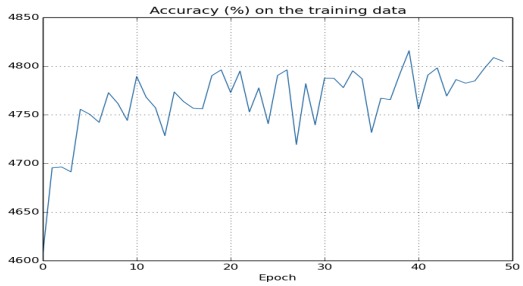

Fig. 15. Batch Normalization Neural Network Training Accuracy

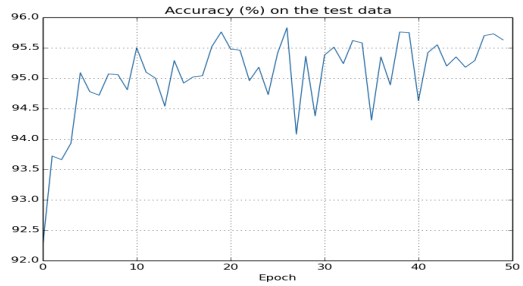

Fig. 16. Batch Normalization Neural Network Test Accuracy

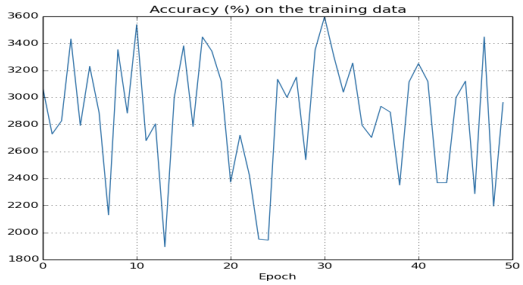

Fig. 17. Dropout Neural Network Training Accuracy

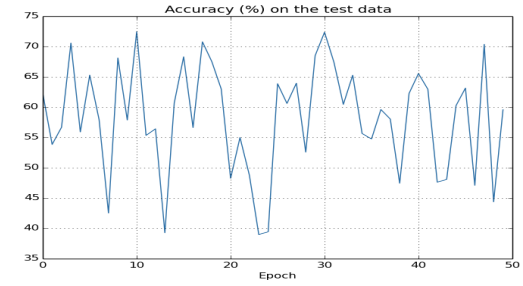

Fig. 18. Dropout Neural Network Test Accuracy

B. Convolutional Neural Networks

Convolutional Neural Network architectures allow more complex functions to be modelled easily using parameter sharing, receptive field and convolution operation. It tries

to model biological processes to give better learning. We implemented LeNet-5, AlexNet and VGG-16 using the PyTorch Neural Network and saw their performance on the Tiny ImageNet Dataset using the Adam optimizer. We see that VGG-16 performs the best out of these 3 while AlexNet was giving good generalization but slow convergence. VGG-16 was able to remember the dataset well with almost 80% training accuracy and decent generalization with 30% test accuracy. Both AlexNet and LeNet weren't able to cross the 20% mark in the accuracy charts in the time they were run. LeNet was not well equipped for the dataset which was quite difficult and even a human wouldn't get as good an accuracy as VGG-16.

We further compared the SGD, SGD with momentum and Adam optimizer by running LeNet-5 on the MNIST dataset. We see that Adam optimizer converges quicker and gives way better generalisation. This is because Adam is able to optimize the learning rate based on the current state of the network and thus lead to faster learning. Momentum improves performance of SGD as expected. We also saw that the MNIST dataset gave much better convergence and generalization mainly because of it being an easy dataset to understand.

When we run LeNet-5 on Tiny Imagenet, we run for 120 epochs with a batch size of 32. While using LexNet on Tiny Imagenet, we have a batch size of 64 for the same 120 epochs. Finally for VGG-16 on Tiny ImageNet and comparing optimizers on LeNet-5 for the MNIST dataset, we reduce number of epochs to 60 with the same batch size of 32.

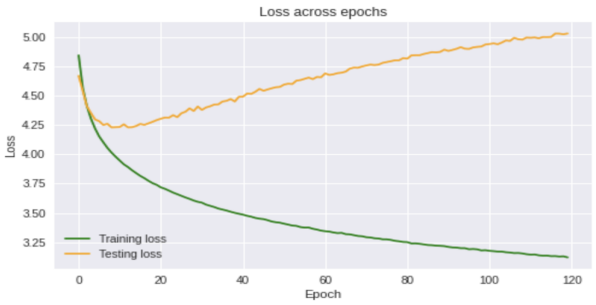

Fig. 19. LeNet-5 on Tiny ImageNet

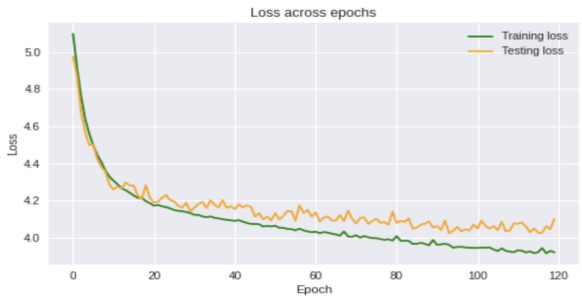

Fig. 20. AlexNet on Tiny ImageNet

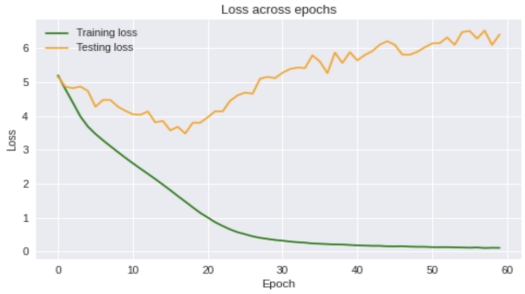

Fig. 21. VGG-16 on Tiny ImageNet

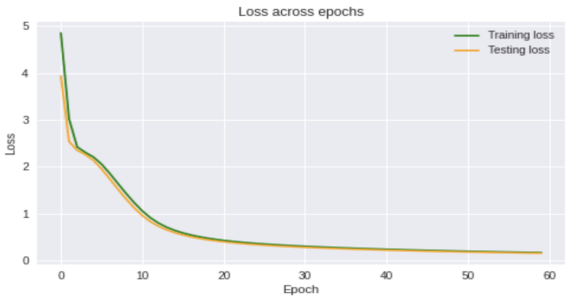

Fig. 22. AlexNet on Tiny ImageNet

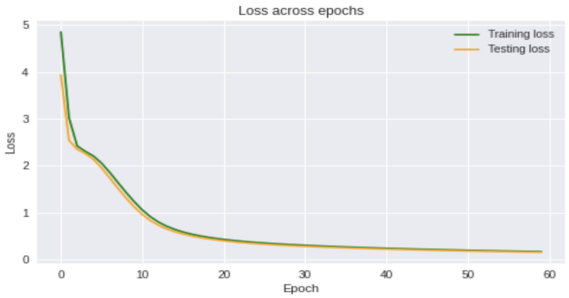

Fig. 23. AlexNet on Tiny ImageNet

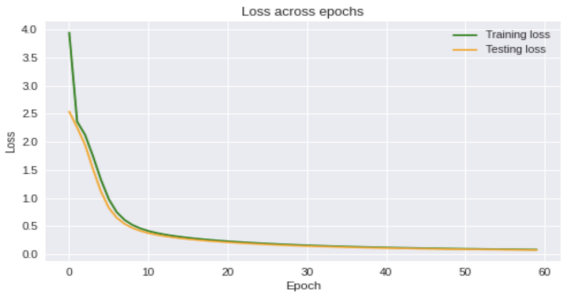

Fig. 24. AlexNet on Tiny ImageNet

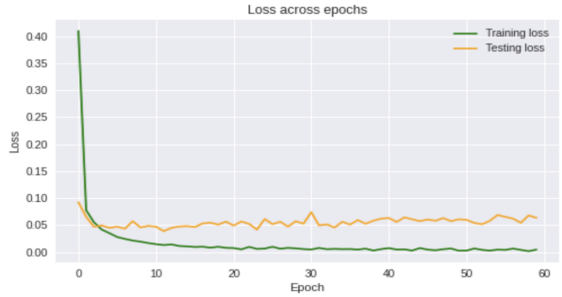

Fig. 25. AlexNet on Tiny ImageNet

III. Learning++

In this section, we used existing networks we created to solve interesting problems.

A. Unbalanced 3-class classification

We used the MNIST dataset to create an unbalanced training set with 3 classes. We split them into test and training set and ran them on the previously implemented FCN with both Cross Entropy and Mean Squared Error loss functions. We further tried L2 regularization to get better generalization (although the network worked pretty well even without it).

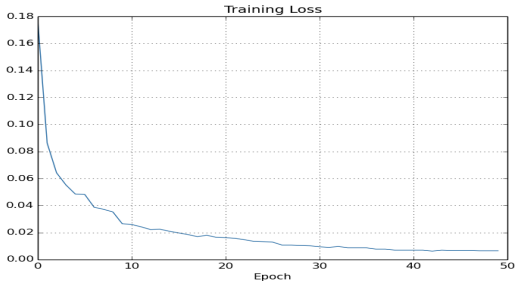

Fig. 26. Unregularized MSE Neural Network Training Loss

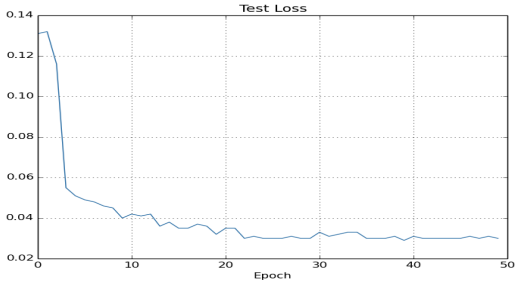

Fig. 27. Unregularized MSE Neural Network Test Loss

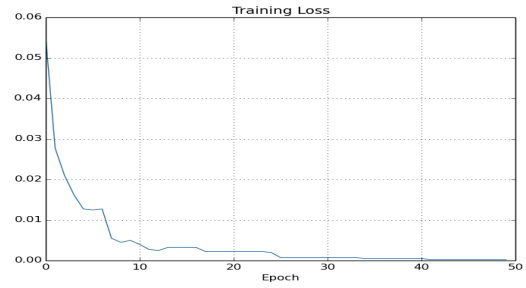

Fig. 28. Unregularized Cross Entropy Neural Network Training Loss

Fig. 29. Unregularized Cross Entropy Neural Network Test Loss

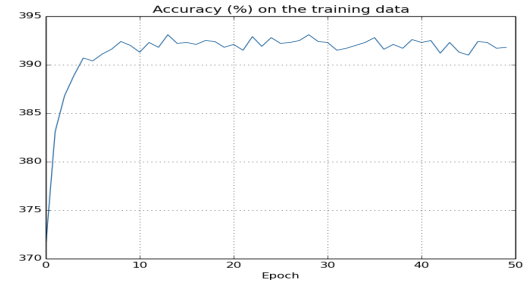

Fig. 30. L1 Regularized MSE Neural Network Training Loss

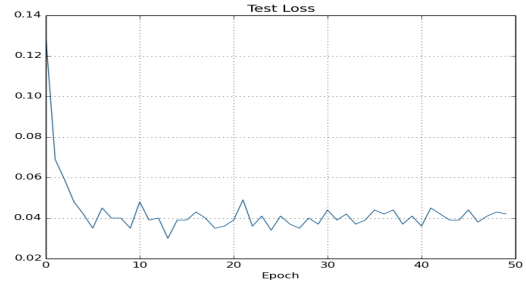

Fig. 31. L2 Regularized MSE Neural Network Test Loss

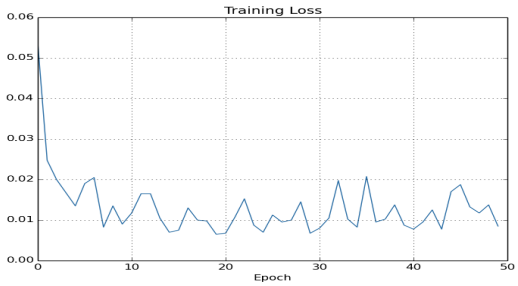

Fig. 32. L2 Regularized Cross Entropy Neural Network Training Loss

Fig. 33. L2 Regularization Cross Entropy Neural Network Test Loss

B. K-means Clustering

We implemented the K-means clustering on the SVHM dataset with 10000 samples due to availability of limited time and computational power. The actual cluster plots for K=3,5 & 10 show that the data is not cleanly distributed and highly irregular. This is a possible explanation for the extremely low accuracy of K-means Clustering.

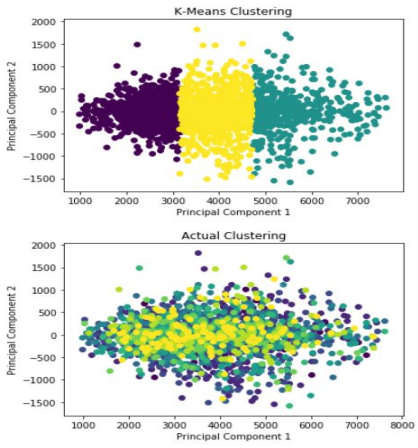

Fig. 34. K-means clustering with K=3

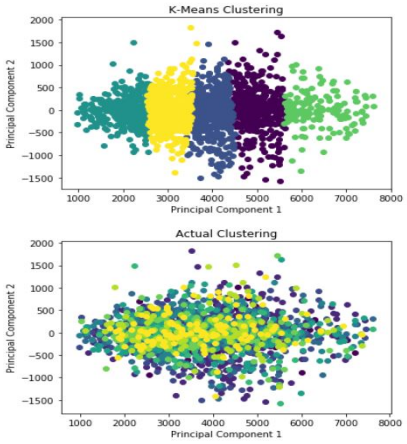

Fig. 35. K-means clustering with K=5

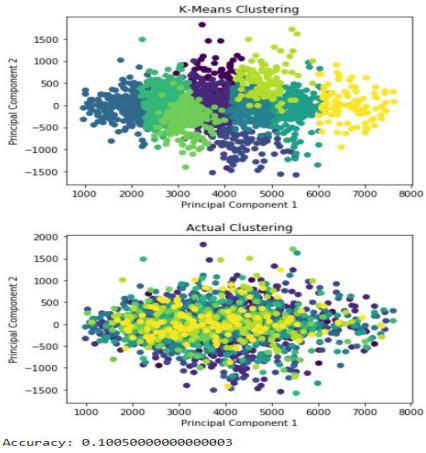

Fig. 36. K-means clustering with K=10

For higher values of K=15,20 we used the elbow method by using WCSS. The following plot was attained.

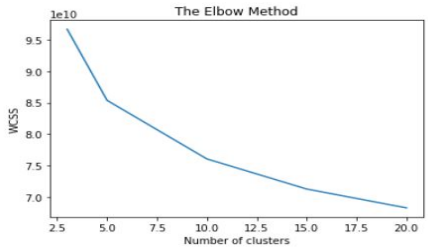

Fig. 37. K-means clustering with K=3,5,10,15,20 via Elbow Method by WCSS

Moreover, even on training a LeNet-5 model on the complete SVHN dataset, the accuracy still remained a dismal 19%. This just goes on to show that the SVHN is a challenging dataset with not many algorithms or architectures available for its absolute best classification.

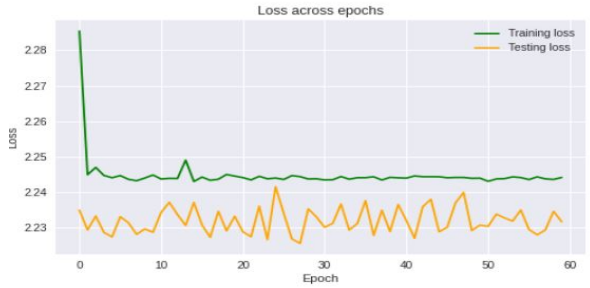

Fig. 38. Losses on training LeNet-5 on SVHN Dataset

```
Epoch: 9   Train loss: 2.245    Test loss: 2.229    Train accuracy: 18.921  Test accuracy: 19.587
Epoch: 19  Train loss: 2.244    Test loss: 2.233    Train accuracy: 18.887  Test accuracy: 19.445
Epoch: 29  Train loss: 2.244    Test loss: 2.233    Train accuracy: 18.921  Test accuracy: 19.587
Epoch: 39  Train loss: 2.244    Test loss: 2.236    Train accuracy: 18.921  Test accuracy: 19.587
Epoch: 49  Train loss: 2.244    Test loss: 2.231    Train accuracy: 18.921  Test accuracy: 19.587
Epoch: 59  Train loss: 2.244    Test loss: 2.232    Train accuracy: 18.921  Test accuracy: 19.587
```

Fig. 39. Statistics on training LeNet-5 on SVHN Dataset

C. Cross Training

We used the LeNet-5 architecture to implement cross training on SVHN and MNIST. We first trained on SVHN and tested on MNIST and the repeated the reverse. We compressed (or expanded) both sets to 1 channel 32 x 32 images. We noted that the generalization was much better when the test set was SVHN which makes sense because it is in some sense more complicated, and its learnings can be applied to MNIST more easily than the reverse. We used a batchsize of 64 and ran the network for 60 epochs. Using Adam optimizer and cross entropy loss gives us good results which wouldn't be the same had we used MSE loss function, which can also be the solution to the problem posed.

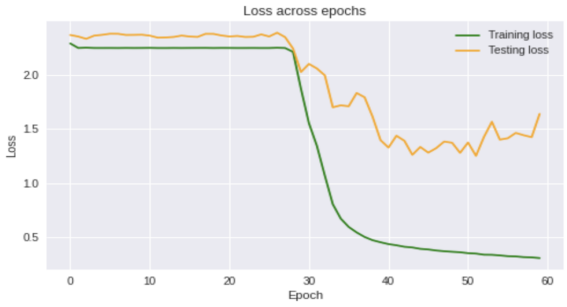

Fig. 40. LeNet-5 Trained on SVHN Tested in MNIST with Cross Entropy Loss

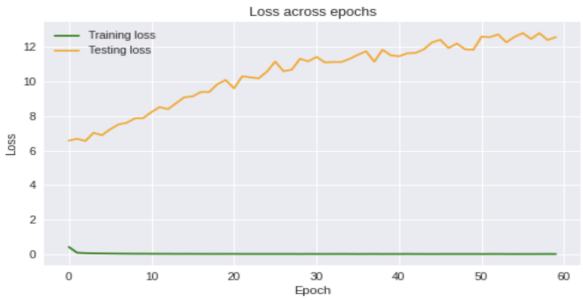

Fig. 41. LeNet-5 Trained on MNIST Tested in SVHN with Cross Entropy Loss

We can conclude from this part and the previous one that the training on MNIST is tremendously more fruitful than on SVHN.

D. Missing Classes

In this section, we look at the performance of the FCN when the training set consists of only a part of all the classes. (5 in this case). We see that without any modification the training accuracy is about 50%. This is because the network has never seen the other 5 classes and never classifies any input as the missing classes. This means that our primitive algorithm has worse than random performance for these novel classes. We ran this algorithm for 50 epochs for a batch size of 10.

Fig. 42. Missing Classes Training Loss

Fig. 43. Missing Classes Test Loss

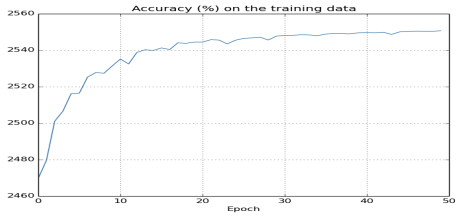

Fig. 44. Missing Classes Training Accuracy

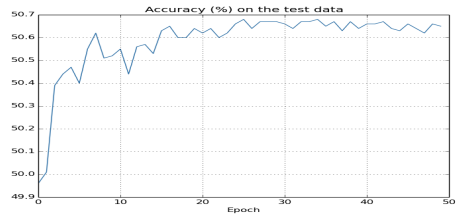

Fig. 45. Missing Classes Test Accuracy

In our research for this assignment, we came across an algorithm called *Zero Shot Learning*, that is designed specifically to tackle such issues. However, that is not a novel method. Due to time constraints and limited computational power we shall briefly discuss our idea to improve accuracy on the novel classes.

After training on the 5 visible classes, we notice which class has argmax values closest to '1' yet their their difference between argmax and argmin values are lower than a threshold for the actual class '1' objects. This has the highest chances of being 7. Similarly, 8 can be mapped to 3, 9 to 4. This way the accuracy on some novel classes, hence overall, can be increased.

E. 2D PCA and t-SNE on SVHN

We ran PCA and t-SNE on the SVHN dataset using standard Scilearn functions. We tried coding them from scratch but they were taking too long on the large SVHN dataset. The t-SNE was not able to distinguish the classes well between the different classes (although it works well on MNIST). This further shows how difficicult the SVHN dataset is compared to MNIST and gives further reason for better performance when training on SVHn than when training on MNIST.

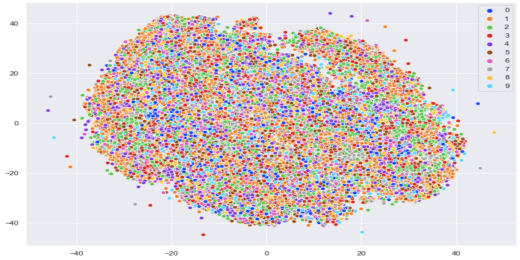

Fig. 46. Missing Classes Training Loss

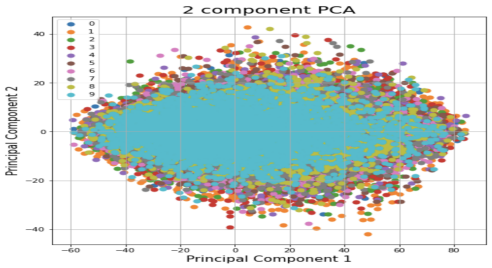

Fig. 47. Missing Classes Training Loss

APPENDIX

Here we put all those images which were unfortunate enough not to make it to the main paper.

F. Neural Network

We put data about losses of the neural network under different regularizers, we can see similar graphs for accuracy over time.

Fig. 48. Unregularized Neural Network Training Loss

Fig. 49. Unregularized Neural Network Test Loss

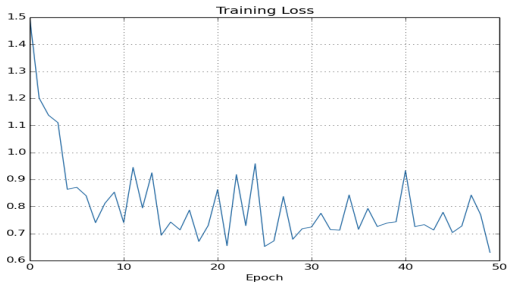

Fig. 50. L1 Regularized Neural Network Training Loss

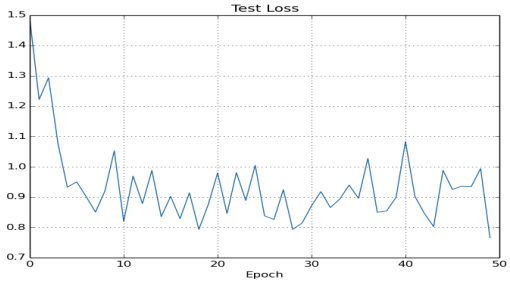

Fig. 51. L1 Regulairzed Neural Network Test Loss

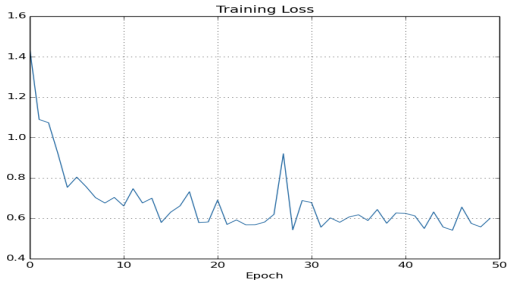

Fig. 52. L2 Regulairzed Neural Network Training Loss

Fig. 53. L2 Regulairzed Neural Network Test Loss

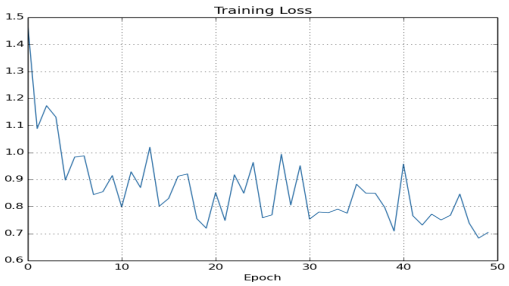

Fig. 54. Batch Normalization Neural Network Training Loss

Fig. 55. Batch Normalization Neural Network Test Loss

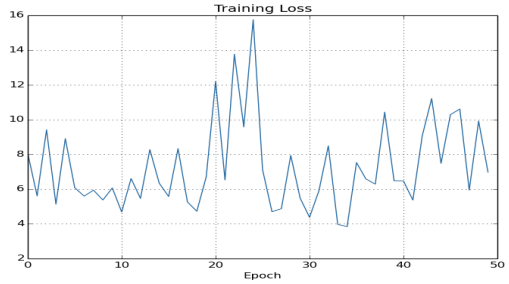

Fig. 56. Dropout Neural Network Training Loss

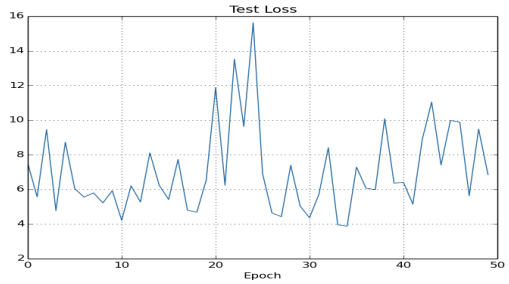

Fig. 57. Dropout Neural Network Test Loss