COMP 551 Assignment 2

Carl, Machaalani; Hugo, Morencé; Lily Jialun, Li

October 31, 2023

Abstract

In our study, we delved into the performance of multilayer perceptrons (MLP) on the FASHION MNIST and CIFAR-10 datasets, adjusting parameters like epochs, learning rates, weight initializations, and regularization strengths. The FASHION MNIST dataset offers 70,000 grayscale images of 28x28 pixels, showcasing a range of clothing items. Meanwhile, the CIFAR-10 dataset presents 60,000 colored images, 32x32 pixels each, spread over 10 unique classes such as airplanes and trucks. Along with the MLP, we implemented a convolutional neural network (CNN) to draw a performance comparison. Our findings underscore the role of hyperparameter tuning in enhancing model accuracy and give insights into how different architectures respond to image datasets, pointing to the importance of model selection in machine learning tasks.

1 Introduction

In this project, we aimed to develop a multi-layer perceptron to classify two bench mark dataset, FashionMNIST and CIFAR-10. We tested various parameters and their effects on model performance. We found that certain weight initialization methods designed for neural networks, such as kaiming and xavier initialization, outperformed other methods and provided better initial weights. We found that increasing the number of hidden layers from 0 to 1 contributes to an observable leap in testing accuracy, while adding more hidden layers improve the testing accuracy modestly. We found that increasing regularization strengths leads to divergence of model. In terms of activation function, ReLU and its variations outperformed Sigmoid. The CNN created for FashionMNIST dataset with PyTorch had a notably better accuracy than the MLP, averaging at 9% and 7% improvement in training and testing accuracy compared to the MLP. The CNN model performance for the CIFAR-10 dataset showed significant improvement for training accuracy but a decreased testing accuracy, indicating potential over-fitting. Despite this, the CNN still outperformed the MLP significantly. We observed that increasing momentum will lead to faster convergence of the model until the momentum is greater than 0.95. With Adam optimizer, the algorithms become more stable as learning rate decreases.

2 Dataset

The first dataset used for this project is the FashionMNIST dataset. It consists of 28x28 unicolored pixel images with 10 classes. These classes represent different types of clothes, however, in the pytorch loader, they are just referred to by numbers, going from 0 to 9. Even though the data is normally ranging from 0 to 255, we normalized it so that it fits the [0, 1] range. All the classes are well distributed inside the training and testing set. The training dataset contains 60k data with each class having 6k, while the testing dataset contains 10k data with each class having 1k. (Figure 4) The other dataset used for this project is the CIFAR-10 dataset. The CIFAR-10 dataset consists of 32x32 pixel images, each falling into one of 10 distinct classes, ranging from dogs, cats, and birds to vehicles like cars and planes. Before analysis, the data were scaled to fit the [0,1] range, and the inputs underwent reshaping for computational efficiency. An examination of class distribution revealed a balanced spread in the training set, with each class containing 6000 images; of these, 5000 were marked for training. Similarly, the test data also showcased a balanced distribution, with each class represented by 1000 images. (Figure 5)

3 Results

3.1 Task 1: Effects of weight-initialization methods on model performance

Choosing Best Parameters: Before proceeding to the tasks, we tested the model performance on different learning rates and batch sizes. The optimal learning rate is 0.005 and number of epoch is 50. Results shown in appendix, **Table 5 & 6**.

For task 1, we created several MLP models with one hidden layer of 128 units using different methods of weight initialization. All five models have a learning rate = 0.005 and epoch number = 50 as these are the optimal parameters we tested previously. When weights are initialized as all zeros, the average testing accuracy is the lowest. This is consistent with our understanding of model convergence and back-propagation. When all neurons in the network have the same output, they share the same gradient during back-propagation, meaning that they learn features in the same way. Interestingly, we observed on multiple runs that the average testing accuracy is consistently 10%. We looked closer into the code and observed that at some point, the output starts to be a NaN matrix. The model converges rapidly, giving the same predictions. This is an example of mode collapse due to bad learning rate or weight initialization. MLP models with kaiming and xavier initialization had equally higher accuracy among all tested methods. Xavier initialization scales the weights based on the number of input and output units, making it suitable for neural networks with Sigmoid or Tanh activation functions. Kaiming initialization is specifically designed for ReLU activation. It is able to adjust the weight variance considering the size of the previous layer, which prevents the vanishing gradient problems.

Looking at the training curves (see appendix,**Figure 6**), the gaussian curve starts with a high loss value and the loss decreases sharply in the first couple of epochs and continues to decrease. For the uniform curve, it behaves in a similar fashion as the gaussian curve, but starts in a lower loss, indicating a better weight in the beginning. For kaiming and xavier initialization, both curves start at a loss value less than 1, further proving our conclusion that these types of weight initialization are optimal for neural networks.

Final Accuracy \ Weight Initialization Methods	**Zeros**	**Uniform**	**Gaussian**	**Xavier**	**Kaiming**
Average Training Accuracy	10.00%	79.33%	77.53%	84.64%	84.91%
Average Testing Accuracy	10.00%	77.49%	76.38%	83.30%	83.59%

Table 1: Final Accuracy of our MLP trained on 30 epochs using different weight initializations

3.2 Task 2: Effects of neural network depth on model performance

In task 2, we investigated the effect of number of hidden layers and activation function for each hidden layer on model performance. Results are shown in table below. When using ReLU activation functions in the hidden layers, we introduced non-linearity that enables the network to learn complex representations of the input data. Without this, the network behaves linearly, which is less expressive. The increase from 82.46% to 84.40% could be attributed to the model's increased ability to capture non-linear patterns in the data. Similarly, adding more layers (increasing depth) to the network allows it to capture more hierarchical features. The deeper layers might capture more complex, high-level features. In the future, it would be interesting to test on additional number of hidden layers to see if this trend persists.

Number of hidden layers	**zero**	**one**	**two**
Testing Accuracy	82.46%	84.40%	84.94%

Table 2: Final Testing Accuracy of MLP trained on 50 epochs using an MLP with no hidden layers, an MLP with a single hidden layer having 128 units and ReLU activations, and an MLP with 2 hidden layers each having 128 units with ReLU activations

3.3 Task 3: Effect of Different Activation Functions

In this experiment, we evaluated three activation functions—ReLu, Leaky ReLu, and Sigmoid—in MLPs with two hidden layers on the Fashion MNIST dataset. All models shared the same architecture: two hidden layers with 128 neurons, a learning rate of 0.01, and 100 epochs, differing only in their activation functions. The results, as shown in the attached figure, revealed that ReLu achieved the highest accuracy at approximately 0.8799. Leaky ReLu followed closely with 0.8722, while Sigmoid trailed at 0.8253. This suggests that ReLu functions slightly outperformed the others, with Sigmoid's lower performance possibly linked to the vanishing gradient issue in deeper architectures.

Activation Function	Accuracy
ReLu	0.8799
Leaky ReLu	0.8722
Sigmoid	0.8253

Figure 1: Test accuracies across different activation functions

3.4 Task 4: Effect of Different Regularization Strengths

3.4.1 Fashion Dataset

In this study, we set the stage with an MLP structure incorporating 2 hidden layers, each having 128 units and ReLU activations. This design was chosen for consistency with prior evaluations. Varied levels of regularization strength, denoted by the hyperparameter λ, were tested, specifically: $1e^{-07}$, $1e^{-06}$, $1e^{-05}$, 0.0001, 0.001, and 0.01.

Our results, as depicted in Figure 2, unveil intriguing findings. For L1 regularization, the peak accuracy achieved was 0.8485 at $\lambda = 0.001$. Meanwhile, L2 regularization recorded a maximum accuracy of 0.8445 at a noticeably smaller λ value of $1e^{-06}$. As λ increased to 0.01, a pronounced dip in accuracy was observed for L1 regularization, suggesting sensitivity to higher regularization values.

λ (Lambdas)	L1 Accuracy	L2 Accuracy
1e-07	0.8439	0.8299
1e-06	0.8407	0.8445
1e-05	0.8446	0.8412
0.0001	0.8414	0.8426
0.001	0.8485	0.8419
0.01	0.781	0.8219

Figure 2: Test Accuracy with L1 and L2 Regularizations on the Fashion Dataset

λ (Lambdas)	L1 Accuracy	L2 Accuracy
1e-07	0.4627	0.4562
1e-06	0.4671	0.4744
1e-05	0.4492	0.4517
0.0001	0.4666	0.4616
0.001	0.4369	0.4612
0.01	0.3128	0.4427

Figure 3: Test Accuracy with L1 and L2 Regularizations on the Cifar Dataset

3.4.2 CIFAR-10 Dataset

Building on our exploration with the Fashion dataset, we further assessed the L1 and L2 regularizations' impact on an MLP model using the CIFAR-10 dataset. The input layer was adjusted to accommodate the CIFAR-10 dataset's dimensionality. Observing figure 3, for L1 regularization, peak accuracy was attained at 0.4666 for $\lambda = 0.0001$. On the contrary, L2 regularization's top performance was 0.4744 with λ set at $1e^{-06}$. Interestingly, L1 regularization displayed a significant decline to 0.3128 when λ increased to 0.01, indicating a susceptibility to higher regularization strengths in the CIFAR-10 dataset. This once again underscores the importance of tuning the regularization hyperparameter for optimal performance across different datasets.

3.5 Task 5: Effect of Input Normalization on Model Accuracy

Input normalization often impacts the performance of deep learning models. We tested this by training an MLP with both normalized and unnormalized images from the Fashion and CIFAR datasets. For

the Fashion dataset, the unnormalized model showed an accuracy of 0.864, slightly surpassing the normalized model's 0.8379. In contrast, for the CIFAR dataset, the normalized model outperformed with an accuracy of 0.4844 compared to 0.4109 from the unnormalized version. Notably, training the unnormalized models required a lower learning rate of 0.0001, compared to 0.01 for the normalized ones, due to training instability at higher rates. This highlights the importance of careful parameter tuning based on input preprocessing choices.

3.6 Task 6 : CNN on FashionMNIST

For the next 3 subtasks, we implemented a CNN. The CNN is defined in two part. The first part is made with two convolutional layers, each with activation functions ReLu and a Max Pooling with a kernel size of 2. The second part is made with two fully connected layers, also with activation functions Relu (as asked in the instructions), and one hidden layer of size 128. We also apply a reshaping step between these two parts such that the 3D form of our input (color; 2D image) is flattened to 1D.

For the FashionMNIST dataset, the size of our channels went from 1 (data is unicolor), to 16 and to 32, with kernel sizes of 5 and paddings of 2 for both layers. This made our input size for the connected layers equal to $32 * 7 * 7 = 1568$. This choice of parameters is based on experiments and inspiration from many online examples. Since the FasionMNIST is a simple dataset, going to 32 channels instead of 64 makes the model not overly complicated for what it is. We got our best results for 20 epochs using a learning rate of 0.01 and a momentum of 0.9 (on a mini-batch gradient descent with a batch size of 100). The final training accuracy was 96.08% and our final testing accuracy was 91.24%, which are both pretty high. The model doesn't seem to underfit or overfit and it doesn't seem to oscillate much in the Training loss, maybe implying that our model has fairly low bias and low variance. This result is better than the one we get from the MLP which often turns around 87% training accuracy and 84% testing accuracy. (Figure 8)

3.7 Task 7 : CNN on CIFAR-10

For the CIFAR-10 dataset, the size of our channels went from 3 (data has 3 color RGB), to 32 and to 64, with kernel sizes of 4 and paddings of 1 for both layers. This made our input size for the connected layers equal to $64*7*7 = 3136$. This choice of parameters is also based on experiments and inspiration from some online examples. However, since the CIFAR-10 dataset is way more complex than the FashionMNIST, going to 64 channels was useful to incorporate as much information as we could from our inputs. We got our best results for 30 epochs using a learning rate of 0.005 and a momentum of 0.9 (on a mini-batch gradient descent with a batch size of 100). The final training accuracy was 99.92% and our final testing accuracy was 71.12%. Although the training accuracy is almost perfect, the testing accuracy is passable. This may imply that our model is overfitting. Even though our current best result with the CNN overfits the data, it can still be noted that it still performed way better than the MLP. The MLP appears to struggle to find patterns on this dataset and the best results turn around 52% for the training accuracy and 49% for the testing accuracy. (Figure 9)

3.8 Task 8 : CNN with multiple Momentum and Adam Optimizer

We tested lots of different momentum for the CIFAR-10 dataset, all using a learning rate of 0.005 and 15 epochs. The momentums that we tested were respectively equal to $[0; 0.25; 0.5; 0.75; 0.9; 0.95]$. We can observe that as we increased the momentum, both the training and testing accuracy increased, except at the 0.95 mark where only the training accuracy increased. This may imply that as we increase the momentum, our algorithm is converging faster. However, if we increase it too much, our model might also overfit since we're getting considerably better training accuracy but not considerably better testing accuracy. Let's also note that as we increased our momentum, the algorithm became more stable. There was less variation in the training loss and it seems like the algorithm is converging around a certain loss faster. (Figure 10)

We also tested lots of different learning rate for the Adam optimizer with the CIFAR-19 dataset. The learning rates that we tested were respectively equal to $[0.01; 0.005; 0.001; 0.0005; 0.0001]$. We can observe that as we decreased the learning rate, both the training and testing accuracy also increased, or until the 0.0001 learning rate at least. However, let's also note that the algorithm seems to become more stable as we decrease the learning rate, until the 0.0001 learning rate again. The worst results from the 0.0001 learning rate could have been because the learning rate became too small and so not much progress happened at each step. However, since it seems to be less stable than before at this learning rate, it's also possible that

Final Accuracy \ Momentum	0	0.25	0.5	0.75	0.9	0.95
Training Accuracy	40.66%	48.07%	56.52%	68.44%	86.27%	95.94%
Testing Accuracy	40.88%	47.11%	55.68%	64.85%	70.54%	69.67%

Table 3: Final Accuracy of our model trained on 15 epochs using different Momentum

our model got stuck in a suboptimal region where there is a lot of variation happening. Compared to the SGD however, no learning rate here seems to make our model overfit. (Figure 11)

Final Accuracy \ Learning Rate	0.01	0.005	0.001	0.0005	0.0001
Training Accuracy	47.31%	62.63%	70.26%	79.90%	55.25%
Testing Accuracy	42.55%	50.13%	56.49%	69.71%	54.17%

Table 4: Final Accuracy of our model trained on 15 epochs using different Learning Rate for Adam

4 Discussion and Conclusion

In this project, we implemented and tested our MLP from scratch using two different datasets, the simpler one FashionMNIST, and the complex one CIFAR-10. We learned the impact of initializing different weights with different distributions. While starting with zeros weight will always lead to 1 class for all of our output, we find that initializing our weight with a distribution like Xavier or Kaiming may have a positive impact on the learning process. We also learned that applying more layers to our MLP might increase the accuracy of our model as it might process more information efficiently, however, adding too many layers might overcomplicate our model. Moreover, we noticed that different activation layers might have some very different effects on our model. For instance, using Relu gave us pretty good and stable results while using Sigmoid gave us results that converged too fast.7

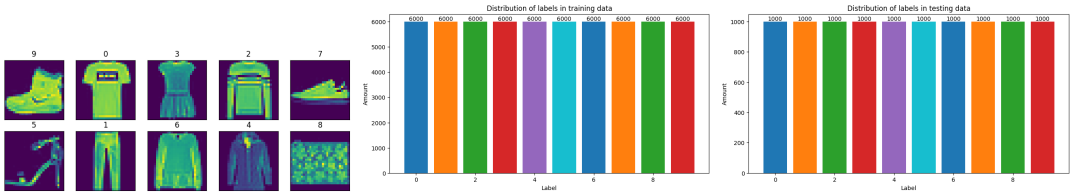
In this project, we also implemented and tested a CNN algorithm using pytorch. This CNN outperformed our MLP in all of our tasks. However, let's note that since CNN is a more complex algorithm than MLP, it was also easier to make an overcomplex model that overfit our data, just like with the CIFAR-10. Using Pytorch, we also observed the impact of different momentum with SGD and the effect of using Adam optimizer instead.

One possible direction for future investigation could be to investigate the effect of regularization on the accuracy of our MLP. We found out that as we increased the number of epochs in some of our tests, our algorithm gave better training accuracy but not testing accuracy. This might mean that our model overfit at some point so seeing the effect of different regularization parameters might be interesting. Another thing that could be worth trying could be to work with even more convolution layers and see how it would recognize the patterns of our current data, or on a different dataset with even more features. However, if we're working with an even bigger dataset, it might be great to also learn some dimension reduction algorithms like PCA.

5 Statement of Contributions

- Carl oversaw the acquisition, preprocessing, and analysis of the CIFAR-10 dataset. Additionally, he played a key role in implementing the MLP and conducted experiments 3 through 5.
- Hugo oversaw the analysis of the FashionMNIST dataset. Additionally, he played a key role in implementing the CNN and conducted experiments 6 through 8, as well as the discussion and conclusion part of this report.
- Lily (Jialun) played a key role in finding out the optimal learning parameters for FashionMNIST dataset, implementing the MLP for experiments 1 and 2, as well as the introduction part of this report.

6 Appendix



(a) (b) (c)

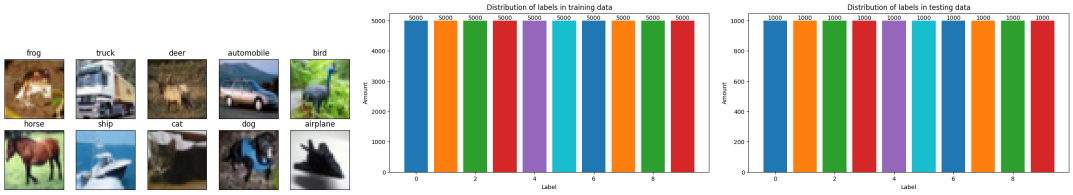
Figure 4: (a)Examples of FashionMNIST dataset classes. (b)Class distribution of training data. (c)Class distribution of testing data.

Final Accuracy \ Learning Rate	**0.0005**	**0.001**	**0.005**	**0.01**	**0.05**
Average Training Accuracy	76.92%	83.02%	87.48%	87.69%	84.20%
Average Testing Accuracy	76.06%	81.70%	85.506%	84.97%	81.24%

Table 5: Final Accuracy of MLP trained on various number of epochs using learning rate = 0.005

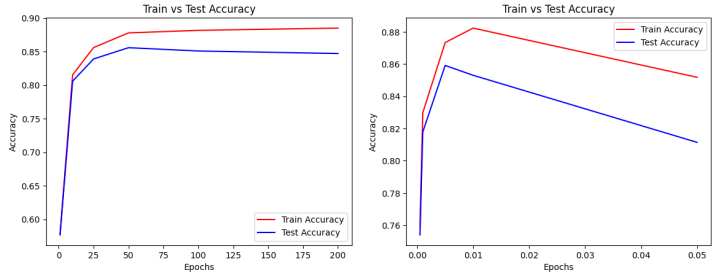
Final Accuracy \ Epoch Number	**10**	**25**	**50**	**100**	**200**
Average Training Accuracy	81.54%	85.61%	87.80%	88.18%	88.51%
Average Testing Accuracy	80.60%	83.91%	85.59%	85.10%	84.72%

Table 6: Final Accuracy of MLP trained on various number of learning rates using epoch number = 50



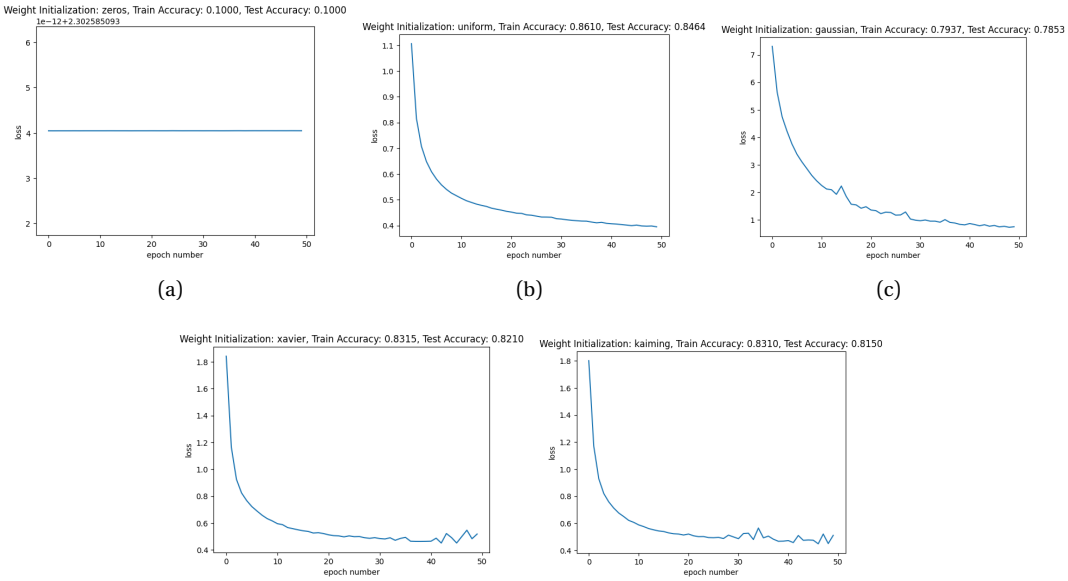
(a) (b) (c)

Figure 5: (a)Examples of CIFAR-10 dataset classes. (b)Class distribution of training data. (c)Class distribution of testing data.



(a) (b)

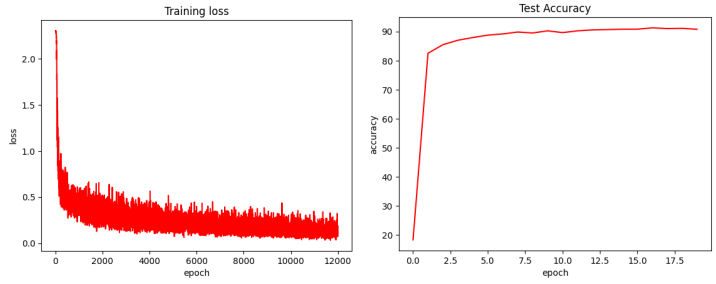
Figure 6: Training Loss vs. Total number of epochs for (a) Zeros (b) Uniform (c) Gaussian (d) Xavier (e) Kaiming weight initialization methods



(a) (b) (c)

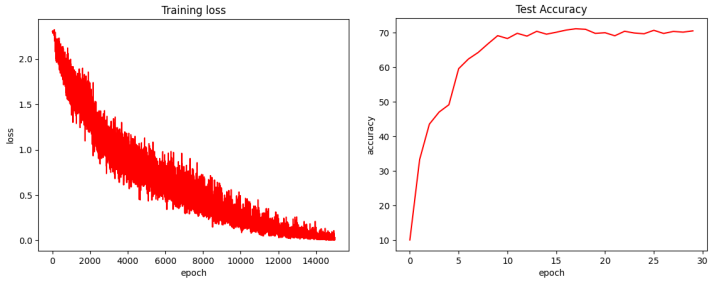
(d) (e)

Figure 7: Training Loss vs. Total number of epochs for (a) Zeros (b) Uniform (c) Gaussian (d) Xavier (e) Kaiming weight initialization methods



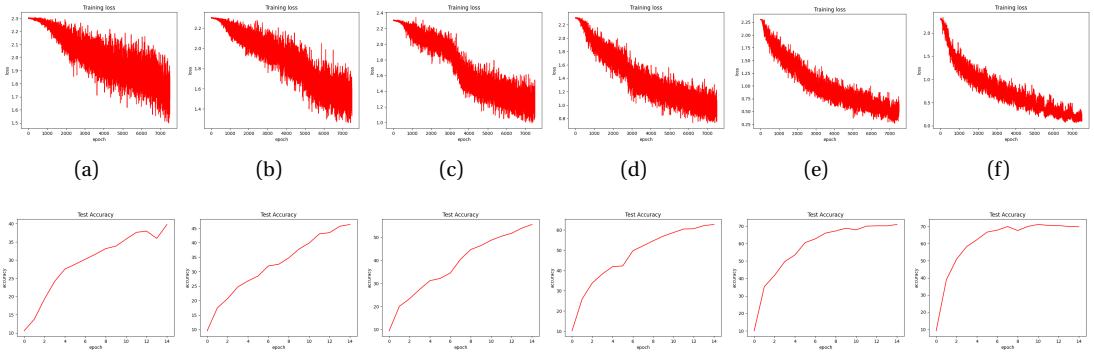
(a) (b)

Figure 8: (a) Training Loss vs. Total number of epochs (b) Average Testing Accuracy vs. Epochs



(a) (b)

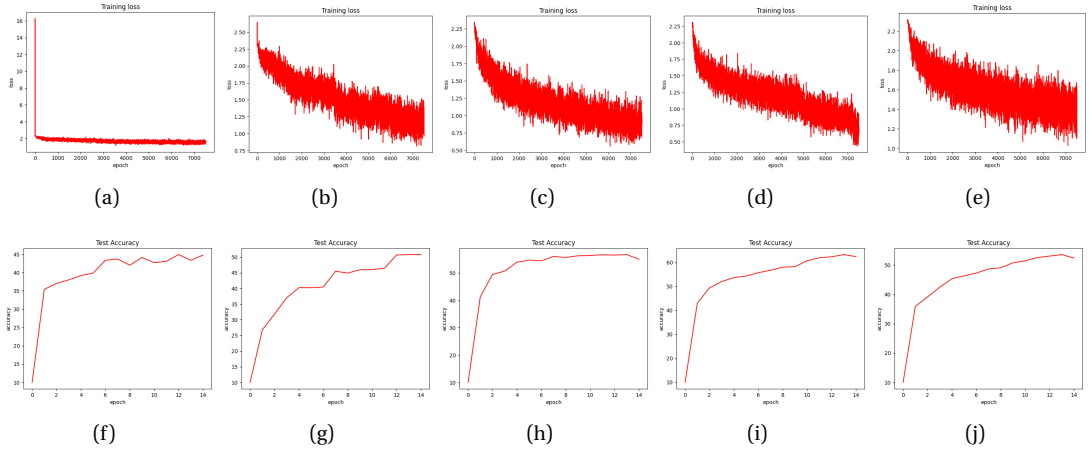
Figure 9: (a) Training Loss vs. Total number of epochs (b) Average Testing Accuracy vs. Epochs



(a) (b) (c) (d) (e) (f)

(g) (h) (i) (j) (k) (l)

Figure 10: Effect of Momentum on Training Loss and Testing Accuracy vs Epochs. From left to right, using momentum equals to 0; 0.25; 0.5; 0.75; 0.9 and 0.95



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Figure 11: Effect of different Learning Rate with Adam optimizer on Training Loss and Testing Accuracy vs Epochs. From left to right, using learning rate equals to 0.01; 0.005; 0.001; 0.0005; 0.0001