Mini-Project 2: Classification of Image Data with Multilayer Perceptrons and Convolutional Neural Networks

Yijie Zhang
McGill University
yj.zhang@mail.mcgill.ca

Tian Bai
McGill University
tian.bai3@mail.mcgill.ca

Yiping Liu
McGill University
yiping.liu@mail.mcgill.ca

Abstract

In this project, we implement a multi-layer perceptron (MLP) to classify image data on two datasets: FashionMNIST and CIFAR-10. The primary discovery of this study is that its performance can be affected by various factors like different initialization ways, different depths, and any other facts listed below. To achieve the best learning performance, we need to choose carefully between these features and carry out more experiments to verify actual performance. We also compared the performance of MLP with that of convolutional neural network (CNN), and pre-trained models (ResNet).

1 Introduction

The perceptron is a fundamental method of neural networks, taking an important place in the field of artificial intelligence. Developed in the late 1950s by Frank Rosenblatt, the perceptron marked a significant milestone in the quest for creating machines capable of learning from data(3). The MLP is a layered feedforward neural network that facilitates the unidirectional flow from the input layer, through the hidden layers, and to the output layer. Each connection between neurons is associated with its own weight. Neurons within the same layer sare a common activation function, typically a sigmoid for the hidden layers. Depending on specific applications, the output layer may implement a sigmoid or a linear function.

Various factors influence the performance of the MLP model. These include the selection of variables, the configuration of hidden layers and nodes, as well as the size of the training dataset. Additionally, training parameters like the learning rate, which governs the magnitude of weight adjustments, and momentum, which regulates weight change persistence, play a crucial role in shaping the model's performance. It's worth noting that an MLP with just one hidden layer may have limitations in capturing complex nonlinear relationships, potentially resulting in lower accuracies. Conversely, models with multiple hidden layers run the risk of overfitting the training data(4).

The primary object of this project is to a) implement the MLP on the target datasets and explore the influence of different factors such as the number of hidden layers, on the performance; b) Compare the performance among different models and the influence of different optimizers on CNN.

In the experiments, we found that a proper initialization way, depth, and activations can improve the performance of MLP. However, misuse of these can also cause the performance to get worse. To achieve the best performance, we need to carry out more experiments and select the best factor based on specific applications.

2 Datasets

2.1 Overview

The FashionMNIST dataset is a valuable dataset for machine learning and deep learning experiments. It consists of 10 categories of clothing and accessories including T-shirts, trousers, and so on. The 50000 images in the dataset are in grayscale, with a size of 28×28. It is similar to the MNIST but is more complex and practical.
CIFAR-10 is also a commonly used dataset for testing the performance of image classification algorithms. It comprises 60,000 32×32 pixel 3-channel color images. It could be divided into 10 categories such as airplane, automobile, and cat. It provides a relatively complex task without requiring computational resources needed for larger datasets.

2.2 Feature Analysis

To verify the characteristics of the datasets and prepare for the classification, we draw example images of these two datasets. As shown in Fig. 1, the FashionMNIST dataset is composed of 28×28 pixel grayscale images, which could be classified into 10 categories. Note that the sizes of the test set and train set are 60000 and 10000 respectively. Similarly, 10 example images of CIFAR-10 dataset is shown in Fig. 2. Note that the sizes of the test set and train set are 50000 and 10000 respectively. The results correspond with the original characteristics of these two datasets, and they are eligible for future experiments.

2.3 Preprocessing

To achieve a fair evaluation of our model performance, we split both datasets into training, validation, and testing subsets randomly. Specifically, 70% amount of the data samples are assigned to the training dataset, 20% assigned to the validation dataset and 10% assigned to the testing dataset. In this way, we can train our model with a majority of original data and choose the best hyperparameters in the validation dataset, then the model performance would be examined in the unseen testing dataset, to avoid information leakage that may lead to untrustful results.

2.4 Experimental Setting

Due to the power of the computational resources, we are unable to run too many epochs with a small learning rate until the model steadily reaches convergence, especially at CPU. Therefore, we set up the training epochs as 100 in all the experiments except Section 3.8 (50 epochs instead). By validation process, we found the best learning rate for 100 epochs is 0.01.

3 Results

3.1 Initialize model weights in different ways

In this part, we create several MLPs with a single hidden layer having 128 units on the FashionMNIST dataset and initialize the weights as (1) all zeros; (2) Uniform $[-1, 1]$; (3) Gaussian $N(0, 1)$, (4) Xavier and (5) Kaiming. Fig. 3, Fig. 4 and Table 1 show the comparison between the effects of different weight initialization on the training and validation curves and test accuracy. It is shown that the ways weights are initialized can have a huge impact on the training and performance of the model. Initializing with all zeros leads to all neurons producing the same output and training is impossible, since the gradients are never updated. In detail, during backpropagation, the update of every weight will be multiplied by the current weight, which is 0. The uniform distribution initialization and Gaussian distribution initialization lead to effective weight updates, but the performances are still poor, with very low test accuracy. As a commonly used initialization method targeted at keeping the variance stable during forward and backward passes within a relatively stable range, the Xavier (and the variant normalized Xavier) initialization initializes weights by randomly sampling from a uniform distribution and scaling them based on the number of input and output neurons. From the figure we could see that it keeps the signals in forward and backward passes within a relatively stable range, which improves training efficiency. Kaiming initialization addresses the initialization issue for the ReLU activation function by considering ReLU's characteristics. If ReLU is used as the activation function, using Kaiming initialization helps maintain the variance of the signals. For non-linear activation functions like ReLU, Kaiming initialization helps avoid the vanishing or exploding gradient

problem. In our case, normalized Xavier initialization gives the best results, but it does not indicate the common scenarios since the results are also affected by many factors such as the number of data samples and the model architecture. In general, if ReLU activation is used in an MLP model, one should use Kaiming initialization whose effectiveness is proven in theory. (2)

3.2 Performance under different depth

In this section, we create three different models: (1) an MLP with no hidden layers; (2) an MLP with a single hidden layer having 128 units and ReLU activations, and (3) an MLP with 2 hidden layer each having 128 units with ReLU activations and test their performance on the FashionMNIST dataset.
Comparing the training and validation curves of the models in Fig. 5 and Fig. 6, we can see that as the number of hidden layers in the model increases, the training speed of the model increases so that the model with 2, 3, 4, and 5 hidden layers overfit quickly starting at approximately epoch 10. Also, as shown in Table 2, the test accuracy increases slightly (esp. comparing 2 layers and 4 layers) as the model becomes deeper, but there is no guarantee of high accuracy when the model is deep. Note that due to overfitting, our deeper models do not show a large performance advantage compared to shallow ones.
In summary, the performance of a model depends on its complexity and the accuracy of parameter tuning. A model that is too simple may not capture complex relationships, while a model that is too complex may overfit the training data. Therefore, choosing an appropriate model structure is a crucial decision that requires adjustments and optimization based on the specific problem and dataset.

3.3 Performance with different activations

Selecting the model with 2 hidden layers, we compare the performance of 5 activations (ReLU, logistic, tanh, leaky ReLU, and softmax) on the FashionMNIST dataset. From Fig. 7, Fig. 8 and Table 3, we can observe that the performance of logistic activation is significantly worse compared to others. This may be caused by its non-zero mean.
Different activation functions introduce different nonlinear properties, affecting the model's expressive power. Some activation functions may be more effective for specific types of data and problems. Meanwhile, they may also affect the way gradients propagate, thereby influencing the training speed and stability of the model.

3.4 Performance after adding regularization

Fig. 9, Fig. 10 and Table 4 show the results of independently adding L1 and L2 regularization. In general, L1 regularization encourages some of the less important features to have exactly zero as weights, which effectively performs feature selection. Hence some weights can be exactly zero and remove certain connections in the network resulting in a sparse model. It can be beneficial for tasks where feature selection is important and make the model more robust to irrelevant or noisy features. L2 regularization is also known as weight decay because it effectively reduces the magnitude of the weights. This can help prevent overfitting by discouraging the model from assigning too much importance to any single feature. This can help prevent overfitting by discouraging the model from assigning too much importance to any single feature. Besides, it tends to spread the importance of features more evenly across the network, resulting in a smoother decision boundary. Unlike L1 regularization, L2 regularization is less likely to completely remove features from consideration. It simply reduces the impact of less important features.
In our case, adding L1 regularization does not improve the performance while adding L2 regularization does slightly. Both methods slow the pace of convergence, helping prevent overfitting.

3.5 Performance with unnormalized images

Train the model with unnormalized images, the results are shown in Fig. 11, Fig. 12 and Table 5. When the input image is not normalized, the model is more sensitive to the absolute values of the pixel intensities, which can lead to issues where the model might overemphasize certain features based on their initial pixel values. Normalized images make the gradients more consistent and help avoid issues like vanishing or exploding gradients. We could see in the table that the test accuracy for normalized data is significantly better than that of unnormalized data. And it can be found that training with normalized images

has faster convergence since the unnormalized images require more careful hyperparameter tuning and a longer training time.

3.6 Comparison between CNN and MLP on FashionMNIST dataset

As shown in Fig. 13, Fig. 14 and Table 6, CNN slightly improves the accuracy compared to MLP on the FashionMNIST dataset. This could be caused by several reasons: (1) CNN is designed to recognize spatial patterns in data. It uses convolutional layers to scan the input data for local patterns, which makes them highly effective for tasks like image recognition when local features are crucial in the training. (2) CNN assumes nearby pixels to be highly correlated. By sharing weights through convolutional kernels, CNNs are able to reduce the number of parameters compared to a fully connected layer in an MLP. (3) CNNs are inherently translation invariant, which means they can recognize patterns regardless of their position in the image while MLPs do not have this property and they need to learn separate sets of weights for each possible position. However, potentially because that the input images are simple and only consist of 1 channel, the performance gap between MLP and CNN is not significant.

3.7 Comparison between CNN and MLP on CIFAR-10 dataset

As in Fig. 15, Fig. 16 and Table 7, CNN significantly outperforms MLP on this dataset. Here, the complexity of the input data allows distinguishing performance differences between the two models.

3.8 Performance of SGD optimizer and Adam optimizer on CNN

Fig. 17, Fig. 18, Fig. 19 and Fig. 20 denote the effect of parameters in Adam optimizer on the performance of CNN. Fig. 17 and Fig. 18 is obtained using the parameter $\beta = [\beta_1, \beta_2]$ where $\beta_2 = 0.9$ and β_1 ranges from 0 to 0.9. Similarly, Fig. 19 and Fig. 20 uses $\beta_2 = 0.9$ and β_1 ranges from 0 to 0.9. We did not perform a 2D grid search on β because the computation is too time-consuming on our machine. In theory, as the momentum increases, the optimizers make more consistent progress towards the optimal solution. And the training process is likely to be more stable with fewer oscillations and fluctuations. However, we observe that the effect of momentum on the test accuracy is rather unpredictable in our case. In detail, when $\beta_2 = 0.9$, the performance is better when β_1 is larger. However, $\beta_2 = 0.9$ shows an exactly opposite trend. This unpredictability could be due to the fact that the CNN cost function is in general non-convex, and momentum could help to escape from local minima. However, the occurrence of local minima varies and there is no general rule to determine which numerical value of β will be more helpful for escaping from local minima.

3.9 Performance of pre-trained model (ResNet-18)

We compared the performance of the pretrained ResNet-18 (5) model and our CNN model in the CIFAR-10 dataset under the same hyperparameters. We used the same split of training, validation, and testing datasets following a random split of 70%, 20%, and 10%, respectively. The training accuracy per step was utilized to compare them, as denoted in Fig. 21, we also recorded the test loss and accuracy for each model. For our CNN model, it finally achieved a loss of 1.564, and an accuracy of 0.631. For the pretrained ResNet model, the test loss is 1.856 and the accuracy is 0.803. It could be seen that the pretrained model converges much faster than our model and achieves better accuracy since it contains more parameters and has been pre-trained on a large dataset. This extensive pre-training process allows the ResNet model to capture a wide range of features from the data, which greatly aids in generalization when applied to our specific task. On the other hand, our CNN model, though trained from scratch, showed a decent performance but lagged behind in terms of accuracy. In summary, while our custom CNN model demonstrates the potential of neural networks in the given task, leveraging pre-trained models like ResNet offers significant advantages in terms of accuracy and training speed.

4 Discussion and Conclusion

4.1 Reflections, discussions, and conclusions

In this project, we implement the MLP on two datasets and explore the influence of different factors on the performance. Afterward, we further compare the MLP with CNN and ResNet. In this process, we find that increasing the depth and selecting a proper initialization way,

activation adding regularization and so on can improve the learning performance. However, compared with the MLP and ResNet, CNN achieves an even better performance. But we observed that factors like the optimizer also influence its results. In summary, various factors should be considered to achieve optimal performance, and it's a good practice to experiment and tune the hyperparameters based on the specific problem at hand.

4.2 Future investigation

There are still several potential areas for exploration in this project. Firstly, more factors that can influence the performance of MLP can be explored. Secondly, the interaction between different factors can be further studied. For example, a possible topic is the relationship between the loss under different depths and initializations.

5 Statement of Contributions

This project is completed as a team of 3, and the work of this project is rather evenly distributed among the group members. Tian Bai established the model's general structure and framework, while Yijie Zhang ran the experiments and made further adjustments and modifications to the model. Yiping Liu was responsible for the report writing part. Three team members reviewed and finalized the report together.

References

[1] Maulud, Dastan, and Adnan M. Abdulazeez. "A review on linear regression comprehensive in machine learning." Journal of Applied Science and Technology Trends 1.4 (2020): 140-147.

[2] He, K. M.; Zhang, X. Y.; Ren, S. Q.; Sun, J. Delving deep into rectifiers: Surpassing humanlevel performance on ImageNet classification. In: Proceedings of the IEEE International Conference on Computer Vision, 1026–1034, 2015.

[3] Kanal, Laveen N. "Perceptron." Encyclopedia of Computer Science. 2003. 1383-1385.

[4] Taud, Hind, and J. F. Mas. "Multilayer perceptron (MLP)." Geomatic approaches for modeling land change scenarios (2018): 451-455.

[5] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Appendix

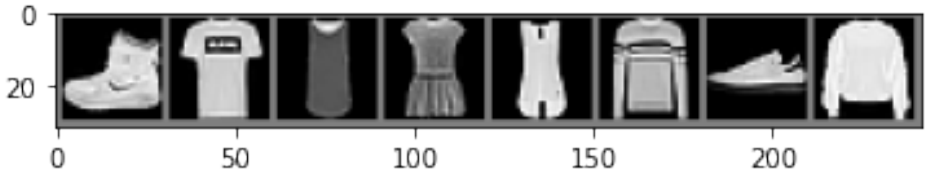

Figure 1: Example image in FashionMNIST dataset.

Figure 2: Example image in CIFAR-10 dataset.

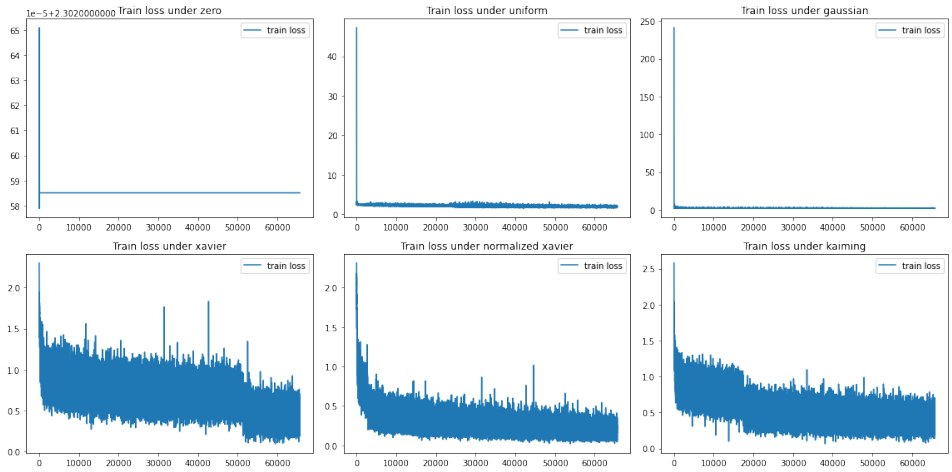

Figure 3: Train loss under various initializations.

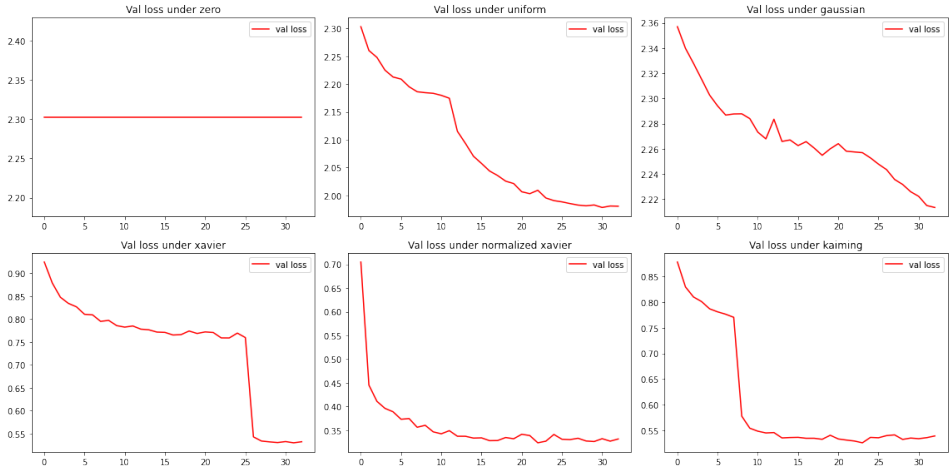

Figure 4: Validation loss under various initializations.

zero	uniform	Gaussian	xavier	normalized xavier	kaiming
0.10250	0.2705	0.18067	0.81500	0.88367	0.80137

Table 1: Test accuracy under various initializations.

0 layer	1 layer	2 layer	3 layer	4 layer	5 layer
0.85117	0.80317	0.78967	0.78267	0.88100	0.78283

Table 2: Test accuracy under various number of hidden layers.

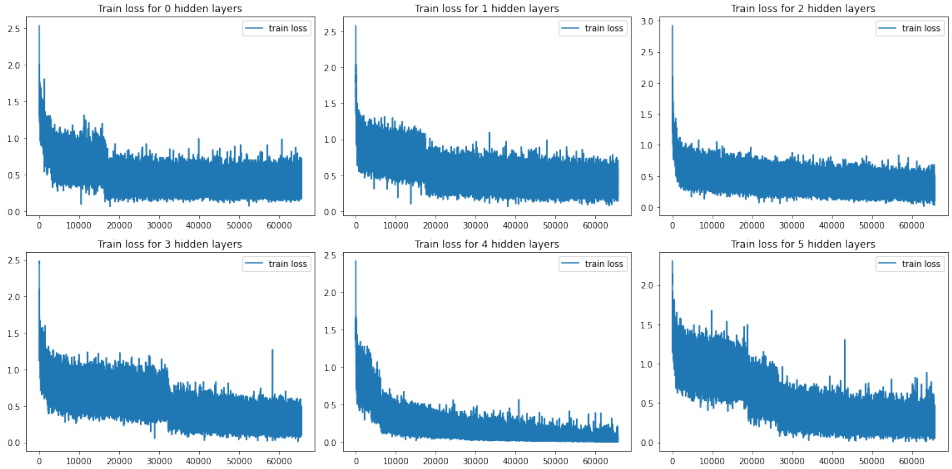

Figure 5: Train loss for MLP models with various depth.

ReLU	logistic	tanh	leaky ReLU	softplus
0.78967	0.58700	0.87600	0.87817	0.88250

Table 3: Test accuracy for MLP models with different activation functions.

No regularization	L1 regularization	L2 regularization
0.78967	0.77250	0.83683

Table 4: Test accuracy for MLP models with different regularizations.

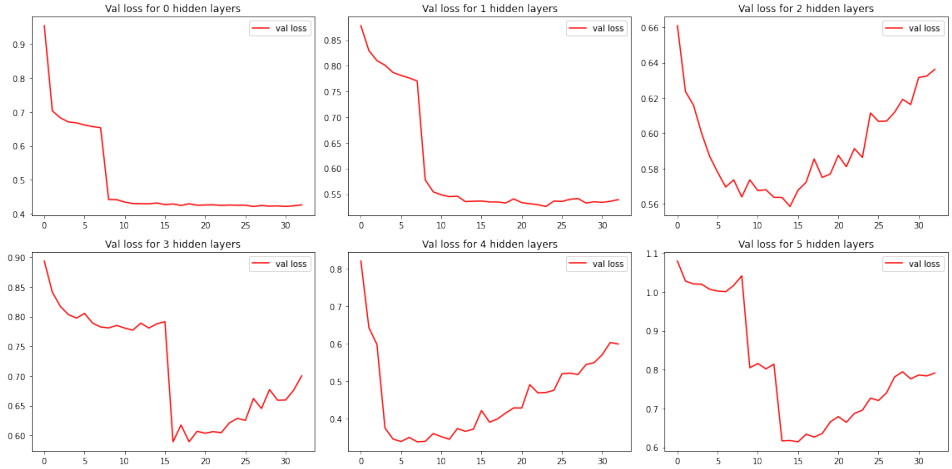

Figure 6: Train loss for MLP models with various depth.

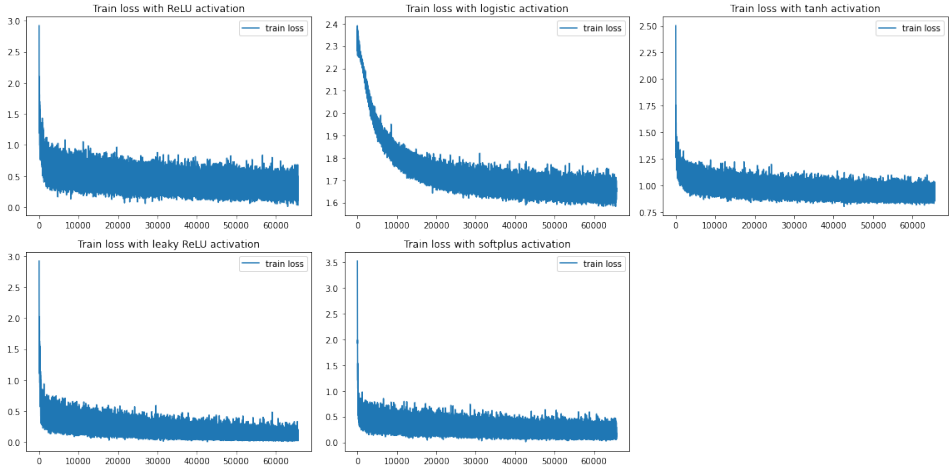

Figure 7: Train loss for MLP models with different activation functions.

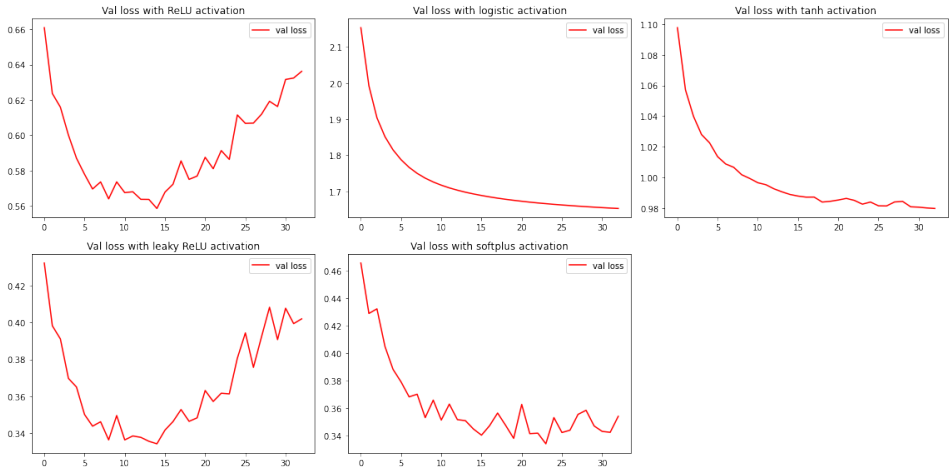

Figure 8: Validation loss for MLP models with different activation functions.

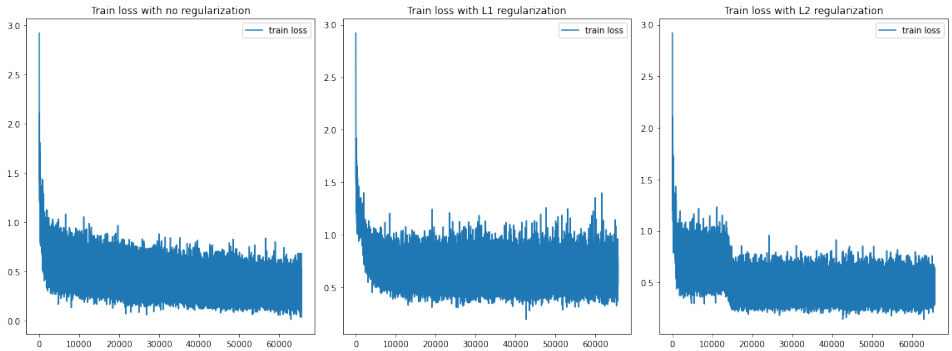

Figure 9: Train loss for MLP models with different regularizations.

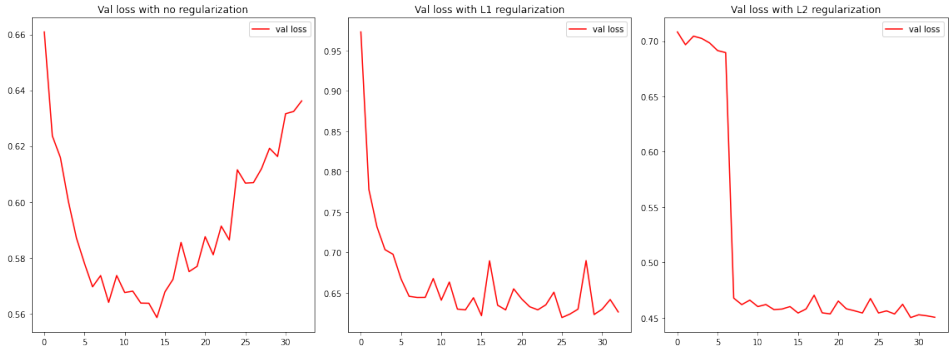

Figure 10: Validation loss for MLP models with different regularizations.

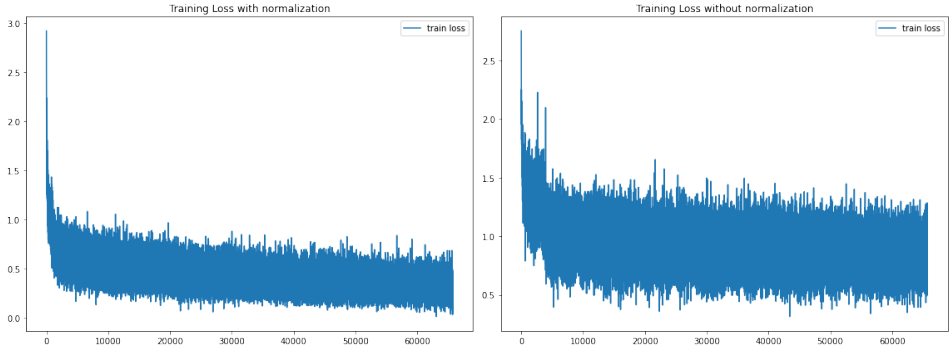

Figure 11: Train loss for normalized/unnormalized data.

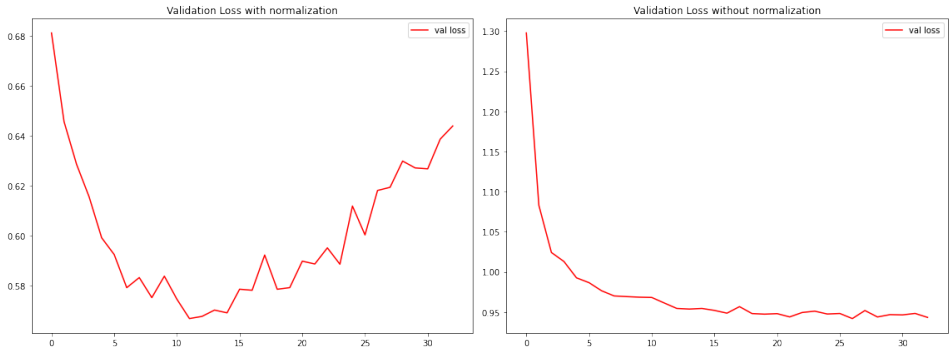

Figure 12: Validation loss for normalized/unnormalized data.

With normalization	No normalization
0.78967	0.65667

Table 5: Test accuracy for normalized/unnormalized data.

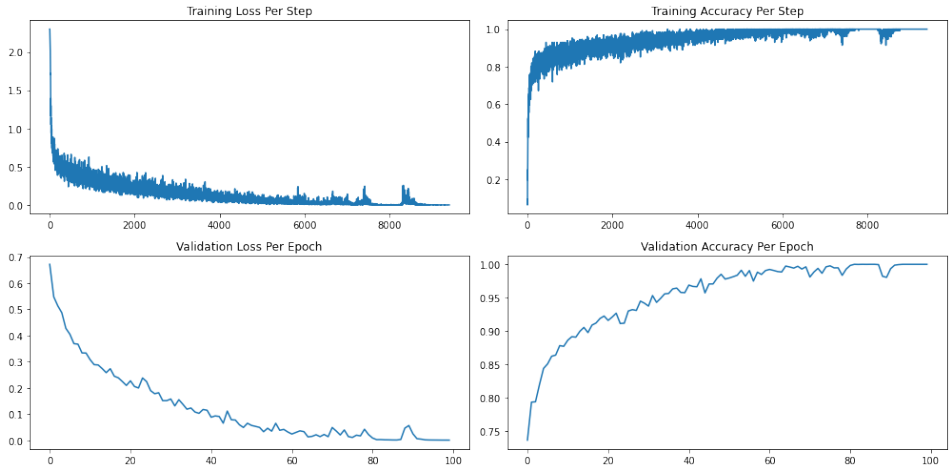

Figure 13: Performance of CNN model for FashionMNIST dataset.

	test loss	test accuracy
MLP	0.86700	0.78967
CNN	0.94261	0.88950

Table 6: Performance comparison for MLP vs. CNN on FashionMNIST dataset.

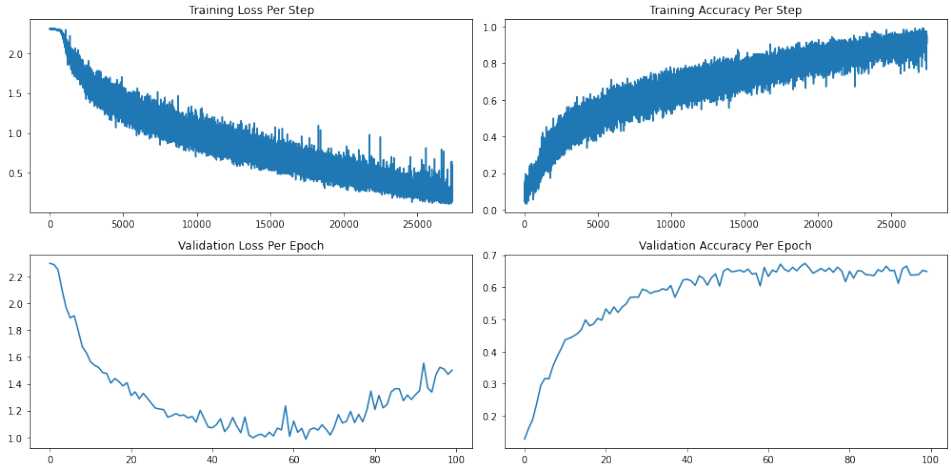

Figure 14: Performance of CNN model for CIFAR-10 dataset.

	test loss	test accuracy
MLP	4.57345	0.47280
CNN	1.49087	0.65220

Table 7: Performance comparison for MLP vs. CNN on CIFAR-10 dataset.

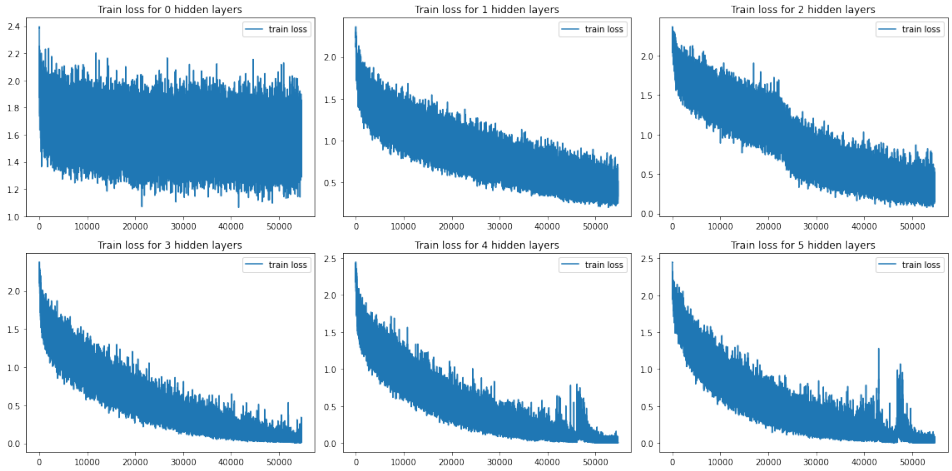

Figure 15: Train loss of MLP model for CIFAR-10 dataset.

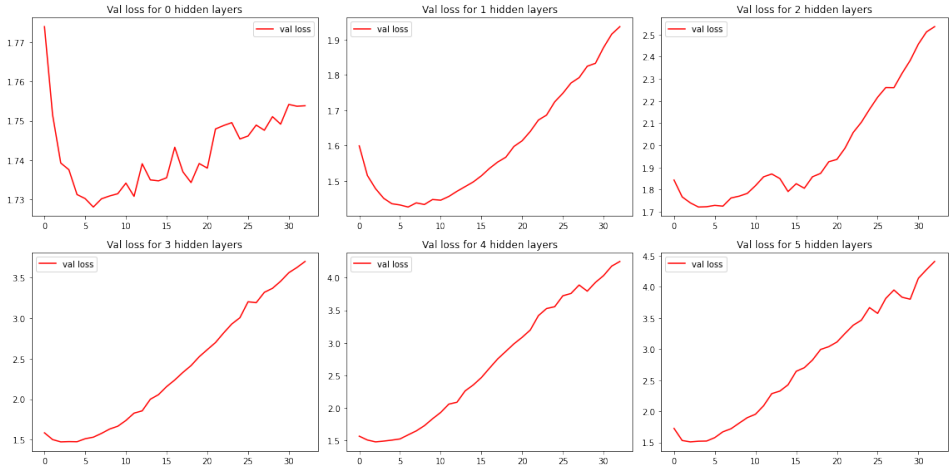

Figure 16: Validation loss of MLP model for CIFAR-10 dataset.

	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
0.9	0.59	0.5742	0.5512	0.502	0.5992	0.5876	0.5614	0.562	0.4528	0.421
0.99	0.6356	0.6582	0.6582	0.6518	0.6414	0.6498	0.6622	0.6538	0.6674	0.6626

Table 8: Performance comparison for various β. Each column represent different β_1 and each row represent different β_2.

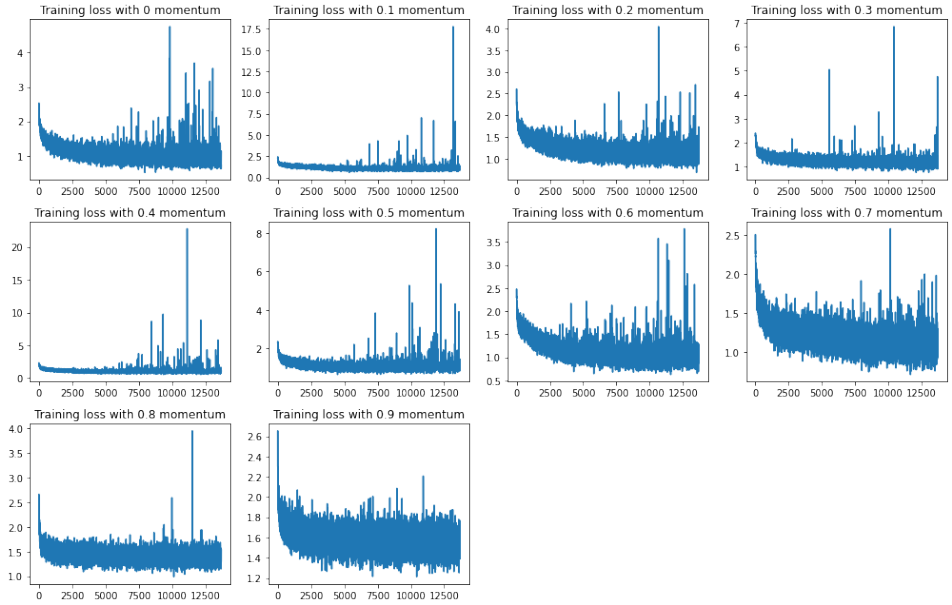

Figure 17: Train performance of model with various momentum (β_1), under $\beta_2 = 0.9$ with Adam optimizer.

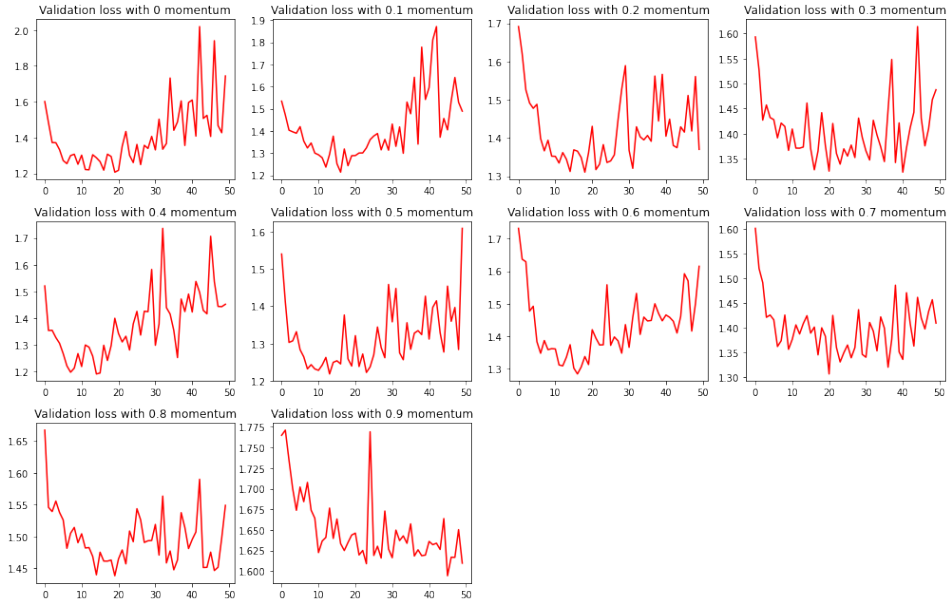

Figure 18: Validation performance of model with various momentum (β_1), under $\beta_2 = 0.9$ with Adam optimizer.

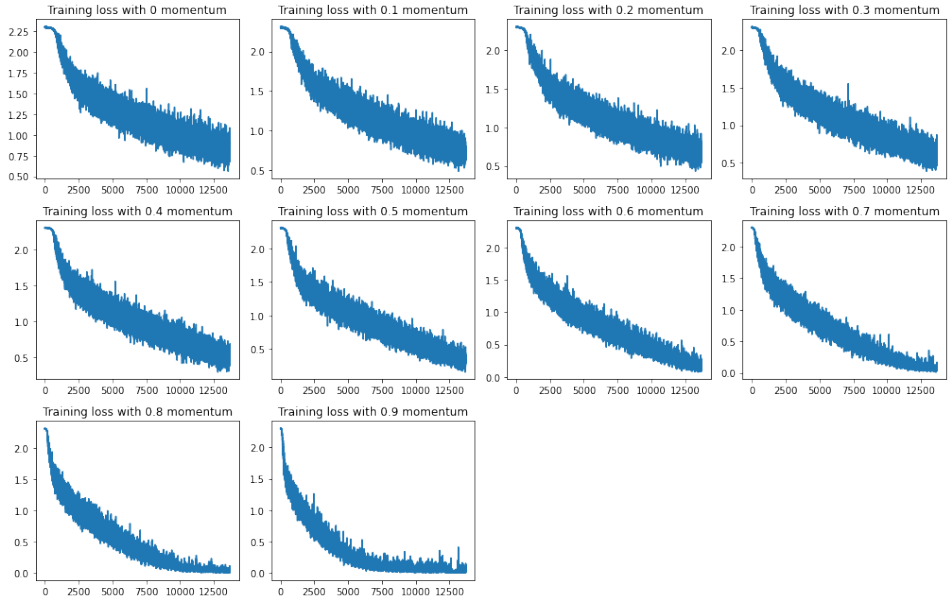

Figure 19: Train performance of model with various momentum (β_1) with SGD optimizer.

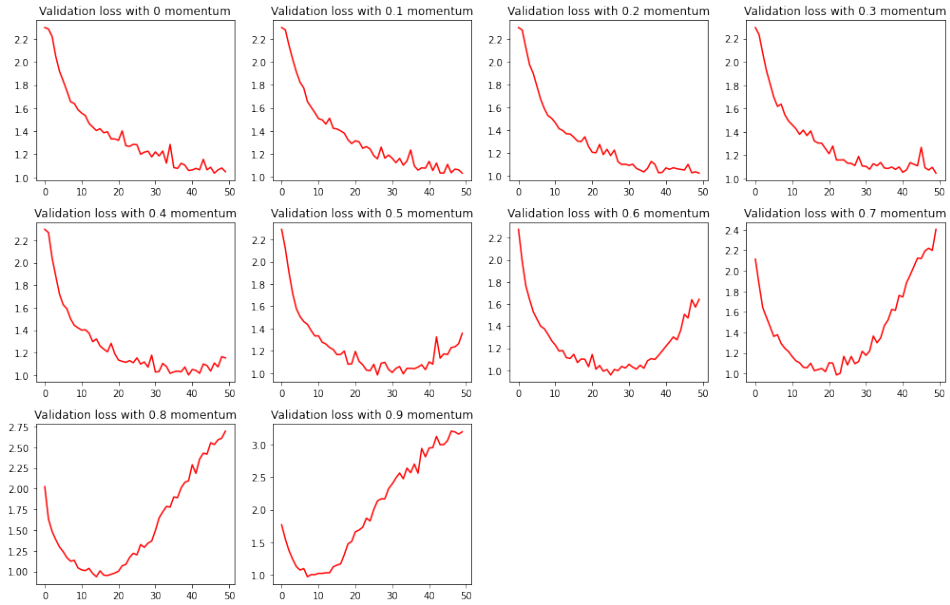

Figure 20: Validation performance of model with various momentum (β_1) with SGD optimizer.

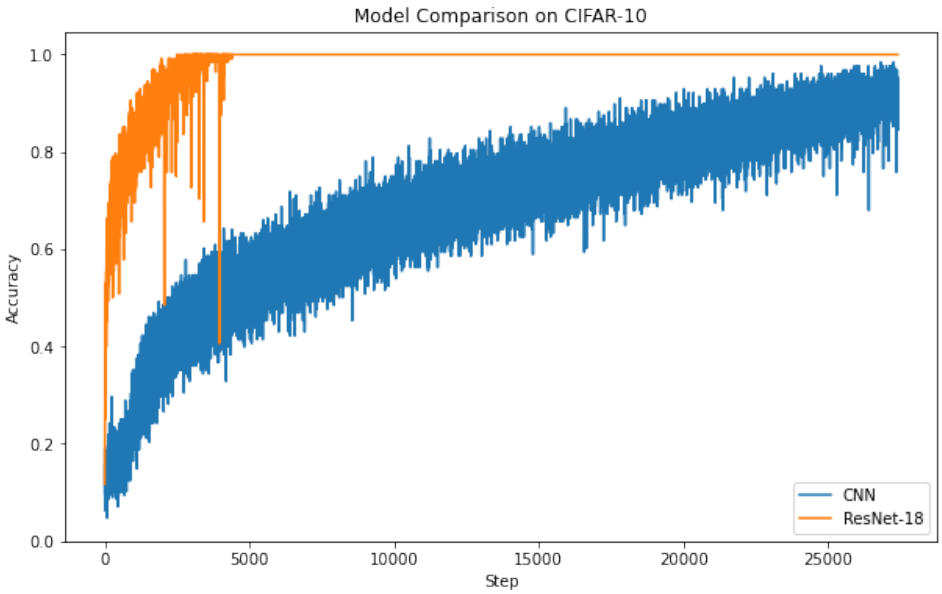

Figure 21: Performance comparison between CNN and ResNet.