EEL6814
Neurual Network and Deep Learning
Project 1

MLP and CNN on the Fashion-MNIST dataset

Authors

Chenxing Dong: generation, training and testing of the MLP model; writing paper sections of introduction, data, MLP and part of conclusion.

Hao Li: generation, training and testing of the CNN model; writing paper abstract, sections of CNN, comparison and part of conclusion.

MLP and CNN on the Fashion-MNIST dataset

Chenxing Dong and Hao Li

***Abstract*—Two commonly used artificial neural networks are widely used in image recognition and classification known as multilayer perceptron (MLP) and convolutional neural network (CNN). After the MNIST dataset has been practiced with numerous high performance results, a Fasion-MNIST dataset is developed for a new challenge to the uses of ANN. This paper focuses on building MLP and CNN network to classify the Fashion-MNIST data base in detail including the hyperparameter search and architecture selection. Based on our experiments, as applied with the best hyperparameter sets in each network, CNN outperforms MLP in terms of accuracy and loss score by 3% difference. However, MLP model spends 10 times less during the training.**

***Index Terms*—MLP, CNN, Fashion-MNIST**

I. Introduction

ARTIFICIAL neural network (ANN) is an artificial computational architecture that imitates the neural network (NN) in animals' or humans' brains. [1] An ANN is consist of a set of artificial neurons, connected forming a network. For each artificial neuron, it receives input information from several other neurons, processes the information and produces output information, and sends its output into several other neurons. By connecting these artificial neurons in different ways, different types of artificial neural networks can be formed. The idea of ANN was first proposed by [2]. The first generation of ANNs is known as the perceptron, introduced by [3] [4], which can be viewed as a single-layer ANN. The backpropagation algorithm [5] enabled the training of multi-layer networks and inspired the invention of the multi-layer perceptron (MLP; [6] [7]). The field of ANN cooled down after the proposal of the support vector machine (SVM; [8] [9]) but was reignited since the introduction of deep convolutional neural networks (CNNs; [10] [11]). In this paper, we apply two commonly used ANN architectures — MLP and CNN — to perform intelligent categorization of the Fashion-MNIST dataset [12].

II. Data

We use the Fashion-MNIST dataset [12] to train and test our MLP and CNN networks. It is inspired by the MNIST dataset [11], but aims to replace the MNIST dataset, because MNIST is too easy for modern ANNs, and is used by too many works, and is not able to represent modern computer vision problems.

Like MNIST, the images in the Fashion-MNIST dataset are grayscale images with 28×28 pixels. The images come from 10 categories that are all fashion products — T-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, and ankle boot, which are labeled 0 to 10. The specific labeling of the categories and example images are shown in Figure 1.

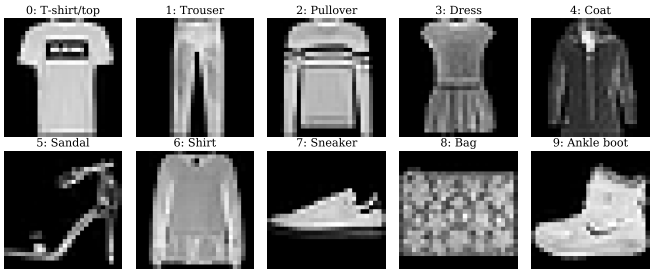

Fig. 1. Example images of the 10 categories of the Fashion-MNIST dataset.

The dataset consists of an already separated training dataset with 60,000 images and a testing dataset with 10,000 images. In our practice, we randomly sample 10% of the training dataset to be a validation set used in the hyperparameter search.

III. Multilayer perceptron

A. Fiducial model

To set up a fiducial model as a starting point for deeper hyperparameter search analysis, we look into several existing MLP applications on the Fashion-MNIST dataset [13] [14] [15] [16] [17]. The network architecture, activation function, output activation, loss function, learning algorithm, learning rate, number of epochs, batch size, and the quoted accuracy are listed in Table I. [13] has the best-quoted accuracy (88.78%), with a three-hidden-layer architecture (256-128-64), rectified linear unit (ReLU) activations in hidden layers, and log softmax activation in the output layer. An SGD learning algorithm with a learning rate of 0.01 is used in [13]. The network was trained with 20-batch data for 100 epochs. [14] achieved an accuracy of 87%, which is close to [13], with only two hidden layers (500-256). They used the same activation and loss function as [13], but used the Adam learning algorithm with a learning rate of 0.001, and didn't apply any activation in the output layer. Their model was trained with 20-batch data for only 20 epochs. [15] also used two hidden layers (128-64) and trained with the same learning algorithm and learning rate as [14] for 20 epochs. In addition, they added a dropout with a probability of 50% in each hidden layer and used the log softmax activation. They used the negative log-likelihood (NLL) loss function and trained with 32-batch data, achieving an accuracy of 85.234%. [16] used the same architecture as [15], but trained with a learning rate of 0.01 for 30 epochs on 64-batch data, and used a dropout probability of 20%, achieving an accuracy of 84.36%. [17] achieved an accuracy of 83%, though using three hidden layers (256-128-64). The activations and loss are the same as [16], and the learning

TABLE I
COMPARISON OF MLP REALIZATIONS ON FASHION-MNIST DATASET

Architecture[a]	Activation[b]	Output[c]	Loss	Optimizer	η[d]	N_{epoch}	Batch size	Accuracy (%)	Reference
256, 128, 64	ReLU	LogSoftmax	Cross Entropy	SGD	0.01	100	20	88.78	[13]
500, 256	ReLU	None	Cross Entropy	Adam	0.001	20	10	87	[14]
128, 64	ReLU+Dropout0.5	LogSoftmax	NLL	Adam	0.001	20	32	85.234	[15]
128, 64	ReLU+Dropout0.2	LogSoftmax	NLL	Adam	0.01	30	64	84.36	[16]
256, 128, 64	ReLU+Dropout0.2	LogSoftmax	NLL	SGD	0.01	40	64	83	[17]
500, 256	ReLU	None	Cross Entropy	Adam	10^{-4}	17	10	**89.25**	This work

[a] Number of nodes in each hidden layer, with 784-nodes input and 10-nodes output for the Fashion-MNIST dataset.
[b] The number after "Dropout" (0.5 or 0.2) represents the probability of the dropout.
[c] The activation function applied on the output layer.
[d] Learning rate or step size.

algorithm and learning rate are the same as [13]. They trained with 64-batch data for 40 epochs.

Since we are required to use one or two hidden layers in the MLP, we skipped the three-hidden-layer architecture with the best accuracy [13] and set up our fiducial model based on the architecture of [14]. We use two hidden layers with 500, 256 nodes in each layer respectively, while the input layer has $28 \times 28 = 784$ nodes and the output layer has 10 nodes representing the 10 categories. The fiducial model used the ReLU activation at each hidden layer but does not apply any activation in the output layer. It uses the cross-entropy loss, the Adam learning algorithm with a learning rate of 0.001, trained on 20-batch data. Starting from the fiducial model, we test different hyperparameters on the model and search for optimal hyperparameters.

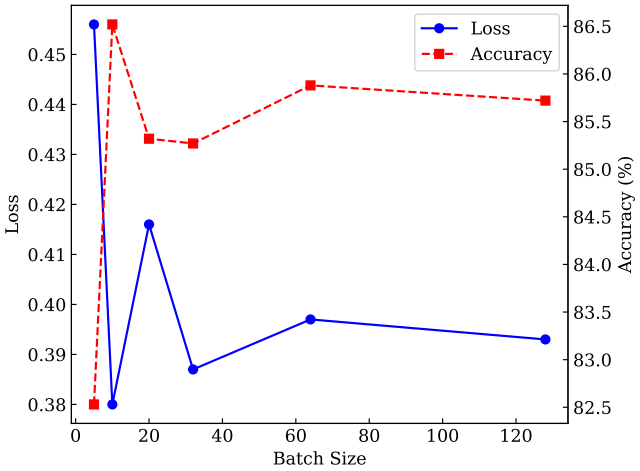

Fig. 2. MLP model performance for different batch sizes. Using batch size of 10 gives the best training performance.

B. Hyperparameters

To search for the optimal hyperparameters, we start from the fiducial model and test model performance for different batch sizes, network architectures, dropout probabilities, learning algorithms, learning rates, and stopping criteria sequentially. For each hyperparameter, we set all other parameters to the fiducial value and change only the hyperparameter being tested. If we find an optimal hyperparameter that is different from the fiducial value, we update the fiducial model with the newly found optimal hyperparameter. The hyperparameters search is done with the training dataset only. 10% of the training dataset is selected to form the validation set, and the rest is involved in the training. Because the search needs to train the model many times, they are trained for only one epoch. After training, the model is tested on the validation set. The loss and accuracy on the validation set are used to evaluate model performance and search for optimal hyperparameters.

1) batch size: The batch size is defined as the number of input data points participating in one back propagation iteration. The smaller the batch size is, the more backpropagation iterations are done within one epoch. We tested batch sizes of 5, 10, 20, 32, 64, and 128. The loss and accuracy for the model with each batch size are shown in Figure 2. The accuracy stays low (82.53%) at batch size 5, then raises at batch size 10 to 86.52%, but then drops down to 85% to 86% from batch size 20 and on. The optimal batch size lies around 10. We changed our fiducial model to have batch size 10.

TABLE II
MLP ARCHITECTURE SEARCH

Architecture[a]	Loss	Accuracy (%)
500, 256, 128, 64	0.432	84.90
256, 128, 64	0.449	84.03
500, 256	0.380	**86.52**
256, 128	0.393	85.95
128, 64	0.417	84.57
500	0.390	**86.47**
256	0.436	84.67
128	0.413	85.02
64	0.426	84.50

[a] Number of nodes in each hidden layer, with 784-nodes input and 10-nodes output for the Fashion-MNIST dataset.

2) Network Architecture: The network architecture is determined by the number of hidden layers in the network and the number of nodes in each hidden layer. We tried a four-hidden-layer network with 500, 256, 128, 64 nodes in the hidden layers sequentially, a three-hidden-layer network with 256, 128, 64 nodes in the hidden layers sequentially, three two-hidden-layer networks with (500, 256), (256, 128) and (128, 64) nodes in the hidden layers respectively, and four one-hidden-layer networks with 500, 256, 128 and 64 nodes in the hidden layers respectively. The loss and accuracy of the model

with each architecture are listed in Table II. Architectures with three and more hidden layers have accuracy less than 85%, which generally do not perform better than networks with two and less hidden layers, who on average have accuracy greater than 85%. Two architectures have accuracy greater than 86%, which are the two-hidden-layer (500, 256) network with the highest accuracy (86.52%) and the one-hidden-layer (500) network with an accuracy of 86.47%. This means a hidden layer with an intermediate number of nodes (500) between the input 784 nodes and the output 10 nodes is crucial for an optimal MLP architecture. We choose to use the two-hidden-layer (500, 256) network architecture, which is already in our fiducial setup.

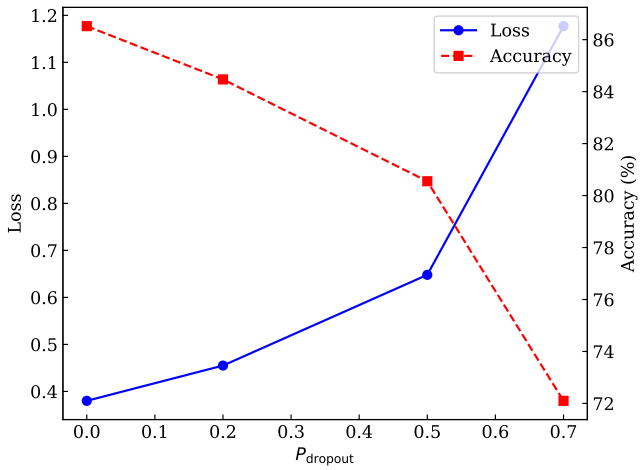

Fig. 3. MLP model performance for different dropout probabilities. $P_{\text{dropout}} = 0$ represents no dropouts, and has the best peformance.

3) Dropout: Dropout is a process to randomly drop out some nodes during each training iteration [18]. We tested the model without dropout and with dropout probabilities of 0.2, 0.5, and 0.7. The loss and accuracy for the model with each dropout probability P_{dropout} are shown in Figure 3, where $P_{\text{dropout}} = 0$ represents the model without dropout. As the dropout probability increases, the loss increases and the accuracy decreases. The model without dropout has the best performance. This is probably because the fiducial two-hidden-layer MLP is concise enough to prevent overfitting, and any additional dropouts will lower down the training efficiency, thus perform worse than the model without dropout given the same number of training iterations. Therefore, we continue to use the model without dropout as our fiducial model.

4) Optimizer: We tried both the SGD and Adam learning algorithms in the training process. Adam outperforms SGD with an accuracy of 86.52%, which is far more than the accuracy with SGD of 76.17%. Studies have shown that Adam tends to perform well in the first few epochs of training, but SGD outperforms at later stages of training, and a strategy to use Adam at the beginning of training and later switch to SGD is recommended [19]. Since our hyperparameter search is trained only for one epoch, it is reasonable that Adam outperforms SGD. To reduce the complexity of our model, we keep using Adam during the whole training process and do not switch to SGD at later stages.

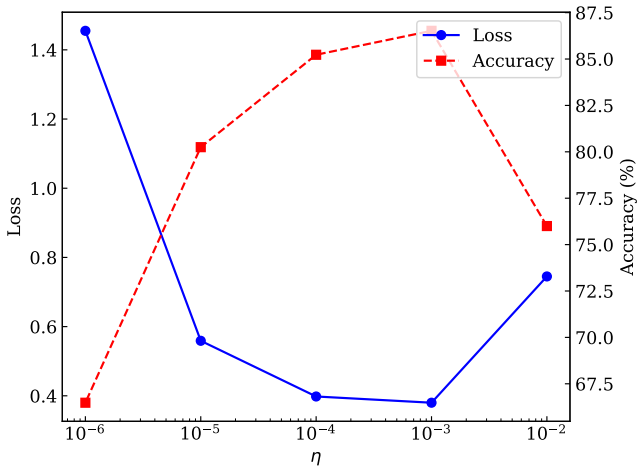

Fig. 4. MLP model performance for different learning rates. The learning rate of $\eta = 10^{-3}$ has the best performance.

5) Learning rate: Using the Adam learning algorithm, we tested learning rates of $10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}$ and 10^{-6}. The loss and accuracy of the model with each learning rate of η are shown in Figure 4. The accuracy increases as the learning rate increases, and peaks at $\eta = 10^{-3}$ with the highest accuracy of 86.52%, and then decreases at higher learning rates. At higher learning rates, the step size of the gradient descent is too coarse that it may jump to a higher loss level even though it is already close to the local minimum, so the loss raises. At lower learning rates, the step size is too small and the algorithm is not able to reach the local minimum given the small number of iterations in our hyperparameter search training. Note that lower learning rates can be complemented by increasing the number of iterations while still keeping the finer step sizes, which will potentially outperform the model currently with the best performance with the learning rate of 10^{-3}. Therefore, we test the stopping criteria together with learning rates of $10^{-3}, 10^{-4}, 10^{-5}$, and 10^{-6}, to find the optimal model with both learning rate and the number of epochs taken into account.

6) Stopping criteria: We used the best performance model found by searching in the hyperparameter space in previous subsections to do our analysis on the training stopping criteria. Instead of testing on the validation dataset, we used the full training set incorporating the validation set to training the model for up to 40 epochs, and test the model performance for each epoch using the testing dataset. The models with learning rates of $10^{-3}, 10^{-4}, 10^{-5}$, and 10^{-6} are tested for the stopping criteria. The testing loss and accuracy for each learning rate as a function of epochs are shown in Figure 5 and Figure 6, respectively. For the model with a learning rate of 10^{-3}, the testing loss keeps low in the beginning but decreases only for the first 5 epochs, and then gradually raises to a higher level; the accuracy acts the same way, reaching a high level of 87.45% at the 8th epoch and bounces around an average of 86.8%. For the model with a learning rate of 10^{-4}, the testing

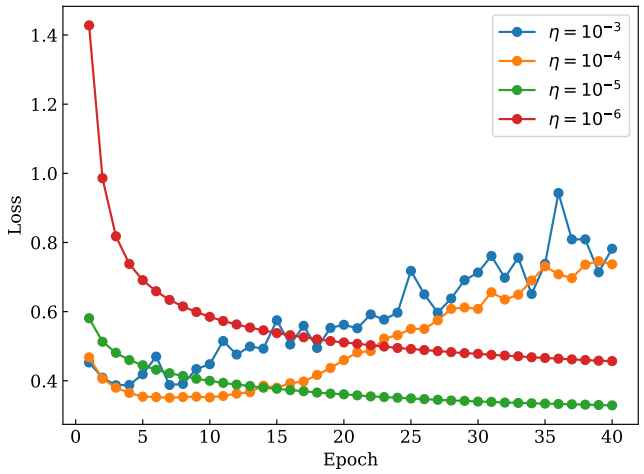

Fig. 5. MLP model testing loss as a function of epoch for different learning rates. The loss of the $\eta = 10^{-3}$ model decreases for the first few epochs but gradually raises up after 5 epochs. The loss of the $\eta = 10^{-4}$ model also raises up after 10 epochs. For the models with $\eta = 10^{-5}$ and $\eta = 10^{-6}$, the loss keeps decreasing for the first 40 epochs.

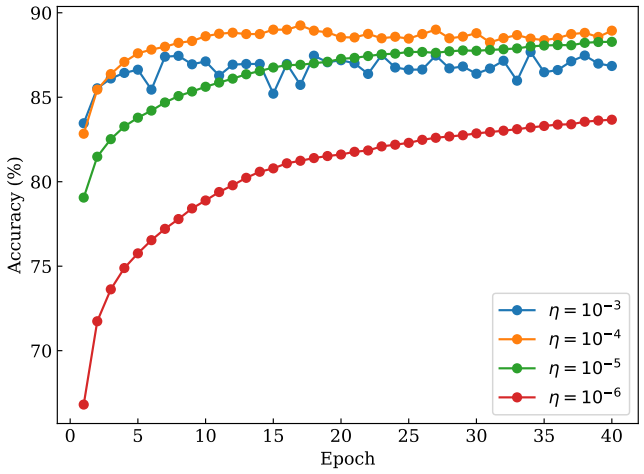

Fig. 6. MLP model testing accuracy as a function of epoch for different learning rates. The model with $\eta = 10^{-4}$ has the highest overall accuracy, achieving an accuracy of 89.25% at the 17th epoch, and an average accuracy of $88.7 \pm 0.2\%$ for iterations after the 10th epoch.

loss keeps the decreasing trend for longer than the $\eta = 10^{-3}$ model, but eventually raises after the 10th epoch; the accuracy reaches the highest level of 89.25% at the 17th epoch, with an average accuracy of 88.7% for the epochs after the 10th, which is the best performance among the four learning rates being tested. For the models with learning rates of 10^{-5} and 10^{-6}, the loss keeps decreasing and the accuracy keeps increasing within the first 40 epochs, which means these models have not reached the optimal solution within the given training iterations. They may eventually reach the optimal solution with higher precision if given more training iterations, but the learning efficiency will be much lower. Therefore, we decide to use the model with a learning rate of 10^{-4}, trained for ~ 17 epochs as our optimal model.

C. Results

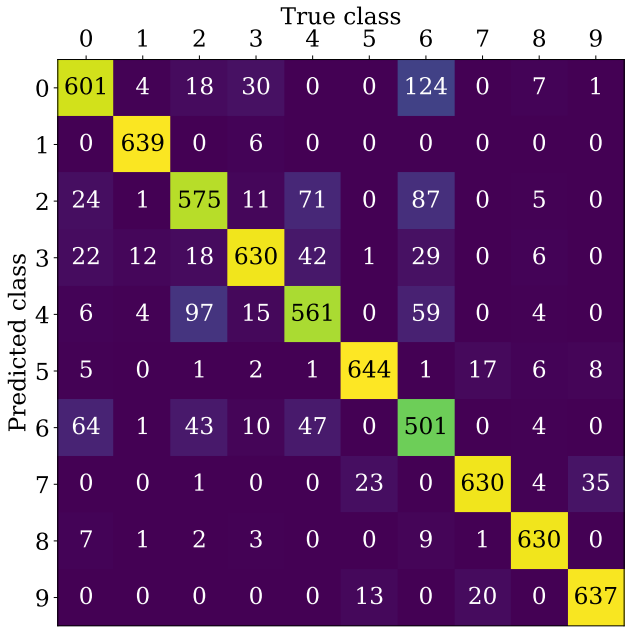

Fig. 7. The confusion matrix for our best performance MLP model with an accuracy of 89.25%.

After searching in the hyperparameter space, we decided to use MLP architecture as follows.

- An input layer with $28 \times 28 = 784$ nodes
- A fully connected layer with 500 nodes, activated by ReLU
- A fully connected layer with 256 nodes, activated by ReLU
- A fully connected output layer with 10 nodes, without activation

The cross-entropy loss function and the Adam learning algorithm are used, with a learning rate of 10^{-4}. The data is organized into batches with size 10 and is trained for 17 epochs when the best testing accuracy of 89.25% is achieved. For epochs between the 10th and the 40th, the average accuracy is 88.7% with a standard deviation of 0.2%. The architecture and the accuracy are listed in Table I. Compared to other MLP realizations on the Fashion-MNIST dataset, our best performance model outperforms the best of other work by 0.47 percentage points in accuracy.

The testing confusion matrix for the best performance model is shown in Figure 7. From the confusion matrix, the worst performance of the model is when the true class is 6 (shirt) but the model predicts class 0 (T-shirt/top). This is consistent with common sense, and even there is only one letter difference in their names. Other large ambiguities come from between pullover, coat, and shirt because they are all top clothes with long sleeves, and even for humans, they are hard to distinguish given only 28×28 pixel grayscale images. The best performance categories include sandals, trouser, and ankle boot. Trouser is the only bottom clothes in these 10 categories, with two distinguishing legs; sandal and ankle boot

are shoes that are pretty unique from clothes in shape, and each has unique features to distinguish them from other shoes. Therefore, the model performance results are reasonable and are consistent with common sense.

IV. Convolutional Neural Network

A. Base model

A convolutional neural network consists of many architectures. The common high-efficiency architectures are ResNet, Lenet-5, AlexNet, Xception, etc. This paper finds the optimal hyperparameters for a baseline model [20]. The first optimal parameter search is based on an architecture of 2 convolution layers with 3x3 kernel and activation function ReLU, 1 max-pooling layer with 2x2 kernel, 1 fully connected layer with 10 outputs and activation function softmax. The optimizer is set automatically as an Adam learning algorithm while using a sparse cross-entropy loss function. To prove the hyperparameters are optimal for the fashion MNIST data set, all parameters are tested with a variety of selections to seek the best performance.

B. Hyperparameters

To obtain the optimal hyperparameters, many model testing is performed in terms of batch size, network architecture, learning rate, optimizer, loss function, stopping criteria, dropout [21]. During the testing, the hyperparameters are selected from previous testing results. Once the testing results proved the optimal value of a certain hyperparameter, the optimal value is inherited in the next hyperparameter testing. For example, the model keeps the optimal batch size from batch size testing for network architectures testing. Ideally, the performance increases as selection goes by. In reality, there is a cross-influence on different combinations of hyperparameters. Thus the performance is not guaranteed to be increasing as the hyperparameter selection proceeds. Note that this selection concentrates on only single optimal hyper-parameter instead of studying the effect of combinations of hyperparameters. During the selection progress, data is split into training and testing in a 6:1 ratio. Among the training set, 10% of it chooses to be a validation set while the rest remains in the training set. To reduce the test required for training, all testing perform training with one epoch. The training loss and accuracy are the standard performance criteria for evaluating the hyperparameter optimum unless exception stated.

1) Batch size: Six values of batch size are selected in ascending order of 5, 10, 20, 32, 64, and 128. Figure 8 exhibit the performance of the model in terms of loss and accuracy within the training set. It is observed that both standards possess a relationship close to linear growth and decay as the batch size increases. With a batch size of 5, the model maintains the highest accuracy at 87.29% while the accuracy drops to 83.34% at a batch size of 128. This represents that the optimal batch size at a lower value provides the best performance. The batch size is set to 5 as a result.

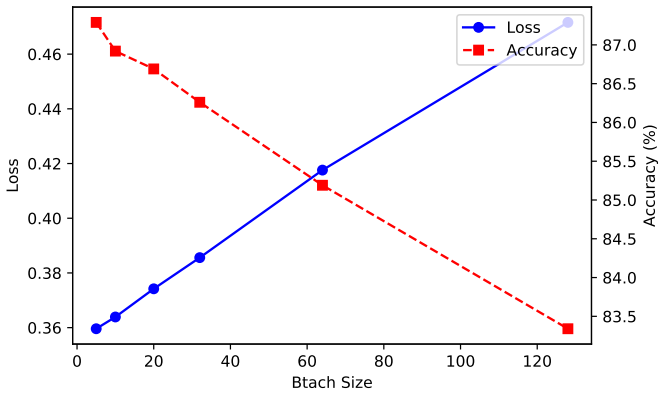

Fig. 8. CNN model performance in diverse batch size. Batch size of 5 performs better than others.

2) Network architecture: For the convolutional neural network, there are numerous choices for network architecture. In this work, the architecture is selected based on simplicity considering the number of layers. To achieve the goal, we tested a single convolution layer with a single pooling layer and a double convolution layer with a single layer. Due to the multiple parameters in each layer, we include the node number, order of convolution layer, kernel size, and pooling layer type in parameter selection.

TABLE III
CNN ARCHITECTURE SEARCH

Architecture	Loss	Accuracy (%)
Single layer	0.366	87.14
Double layer 21	0.362	86.99
Double layer 22	0.353	87.33
Double layer 12	0.352	87.39
Convolution kernel 5x5	0.384	86.29
Convolution kernel 3x3	0.352	87.39
Max pooling	0.352	87.39
Average pooling	0.365	87.02

TABLE IV
CNN NODE NUMBER SEARCH

Node number	Loss	Accuracy (%)
128, 256	0.620	**84.75**
64, 128	0.578	83.75
32, 64	0.500	82.29
16, 32	0.462	79.79
8, 16	0.429	78.20

Table III presents the performance according to layer settings. Note that the double layer with 21 refers to a double convolution layer structure that the layer with a higher number of nodes positioned before the layer with a lower number of nodes. The double-layer with 22 and 12 inherit similar concept. It is shown that the double convolution layer structure outperforms a single layer with a slight difference. When comparing the order of two layers, the double layer with 12 has the highest accuracy at 87.39% whereas 21 and 22

falling behind. In Figure 9, the loss and accuracy relative to the number of nodes are displayed.

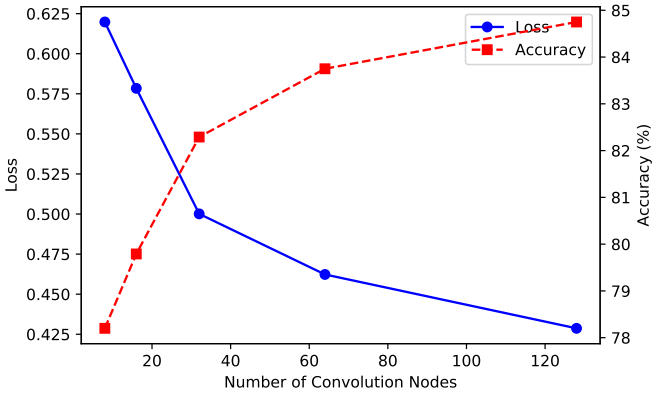

Fig. 9. CNN model performance for different numbers of convolution node. As the number increases, the performance increases accordingly.

According to the performance, a higher number of nodes provides a better accuracy score as well as minimizing the loss. Based on Table III, the squared 3x3 kernel in convolution layers improves the performance by 1% difference. For the pooling layer, we select max pooling method over average pooling due to its slightly better accuracy score. Note that this selection is based on one epoch of training, which could be rejected in a longer training process. In Table IV, the number of nodes in the convolution layer is presented with the corresponding performance score. The double-layer 12 structure is applied with a range of node numbers from 8 to 256. As the node number increases, the accuracy score increases until reaching a local maximum at 84.75%.

3) Optimizer: The optimizer selection is similar to the MLP setting. Both SGD and Adam are tested. SGD gives a loss of 0.748 and accuracy of 73.68%; while Adam gives a loss of 0.365 and accurracy of 87.02%. Adam provides a much better accuracy score with a difference of over 10%. Considering the general utilization of Adam, we keep it as the optimal optimizer for our network.

4) Loss function: There are many loss functions such as mean squared error, but it works the best for a single class. As for multiple class classification, cross-entropy is a common choice. We tested the model with cross-entropy and sparse cross-entropy function. The main difference is the input data format where cross-entropy requires one-hot encoded category labels and sparse cross-entropy uses a category index. The cross-entropy gives a loss of 0.359 and accuracy of 87.24%, while the sparse cross-entropy gives a loss of 0.351 and accuracy of 87.48%. The sparse cross-entropy outperforms the cross-entropy since it needs a single dimension for labels instead of ten. This has greatly reduced the computation and time for calculating the loss and thus creates an advantage in multiple classes classification.

5) Learning rate: Figure 10 contains the loss and accuracy score under five selections of the learning rate. As shown in Figure 10, the performance of the model is enhanced with a smaller choice of learning rate. Nevertheless, this increase reaches a peak at $\eta = 10^{-3}$ with the accuracy maintained as 87.14% then drops to 74.29%. Considering the Adam algorithm, this performance curve is reasonable since gradient descent requires the right amount to find the local minimum.

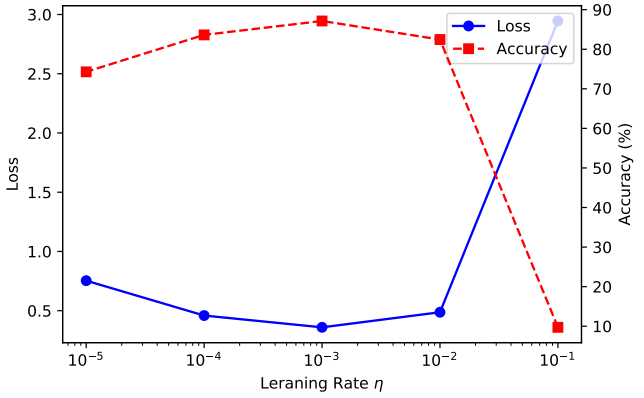

Fig. 10. CNN model performance with different learning rates. At $\eta = 10^{-3}$, the model outperforms the rest learning rates.

6) Stopping criteria: Stopping criteria is defined as monitoring the change of validation set accuracy. The goal is to maximize the validation accuracy with a choice of patience. Patience here means that the learning process will be terminated if the goal is not met after a certain number of epochs. The unit of patience is in epoch number. A sequence of patience values is tested with 20 epochs of training.

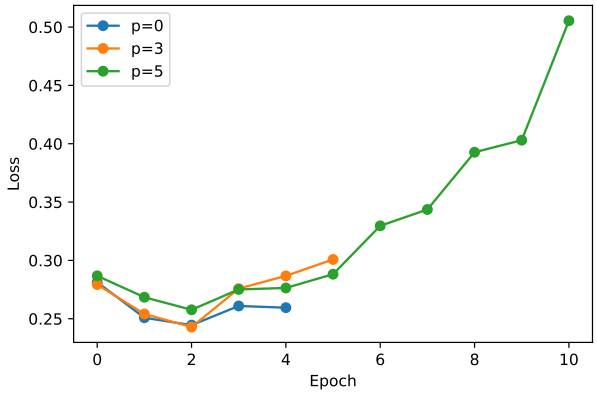

Fig. 11. CNN model loss with three patience values. At p=3, the model outperforms the rest.

In Figures 11 and 12, the loss and accuracy score is labeled with the patience of 0, 3, and 5. The accuracy increases significantly in the first three epochs and then drops a bit. The patience of 5 provides a longer run, however, stops the training at the 10th epoch with no other higher accuracy score. Its loss increases contrary to the accuracy as epoch number increases. Considering all three performances, the patience of 3 is the optimal selection since it provides the highest accuracy score while maintaining the loss at a relatively lower level.

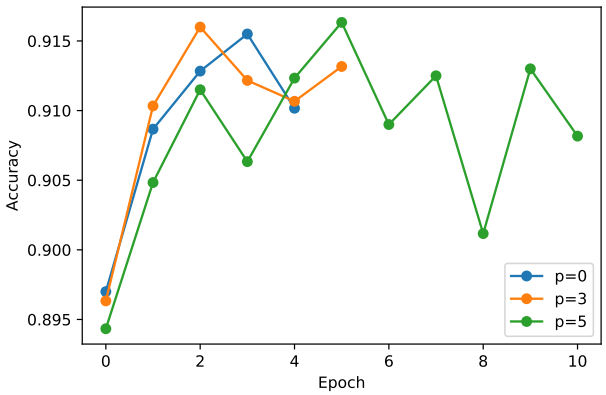

Fig. 12. CNN model accuracy with three patience values. At p=3, the model outperforms the rest.

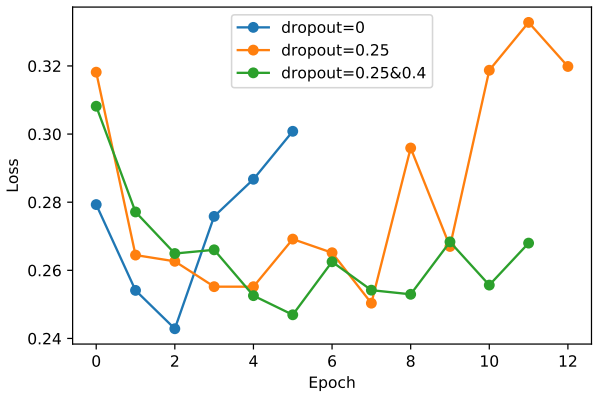

Fig. 14. CNN model loss with dropout testing. At $P_{\text{dropout}} = 0.25$, the model maintains the loss at a lower level.

7) Dropout: During the process of testing stopping criteria, one problem appears to prevent performance from improving as the number of epoch increases. As shown in Figures 11 and 12, the accuracy stops increasing after a few epochs and the loss starts to increase rapidly. After examining, it is determined that our model is overfitted. To avoid this issue, we add dropout in the model to reduce the complexity of the layers [22]. Figures 13 & 14 exhibits the performance with no dropout, one dropout of 0.25, and two dropouts of 0.4 & 0.25. It appears that one dropout with 0.25 has the best performance.

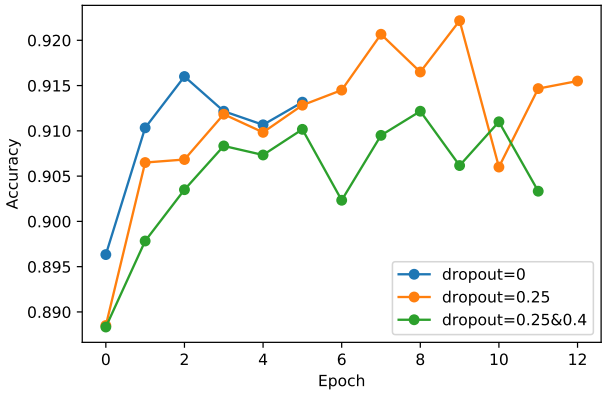

Fig. 13. CNN model accuracy with dropout testing. At $P_{\text{dropout}} = 0.25$, the model maintains the accuracy at a higher level.

C. Results

With all hyperparameter testing results, the CNN architecture is decided as follows.

- An input layer with $28 \times 28 = 784$ nodes
- A 3x3 kernel convolution layer with 256 nodes, activated by ReLU
- A 3x3 kernel convolution layer with 128 nodes, activated by ReLU
- A 2x2 kernel pooling layer with max pooling method
- A fully connected output layer with 10 nodes, activated by softmax

Before data is classified in the output layer, dropout is activated with 0.25. The Adam learning algorithm is utilized with a learning rate of 10^{-3} as well as the sparse cross-entropy for loss function. To prevent meaningless training, the model is set to stop if the validation accuracy does not improve after 3 more epochs with a batch size of 5. From current observation, the model gives better performance when the number of epoch increases. Nevertheless, it has to be restricted due to the time cost for authors' hardware support. According to Figure 13, the best performance appear around epoch 9 and thus the optimal number of the epoch. Figure 15 has the confusion matrix for this best performance model, achieving an accuracy of 91.39%.

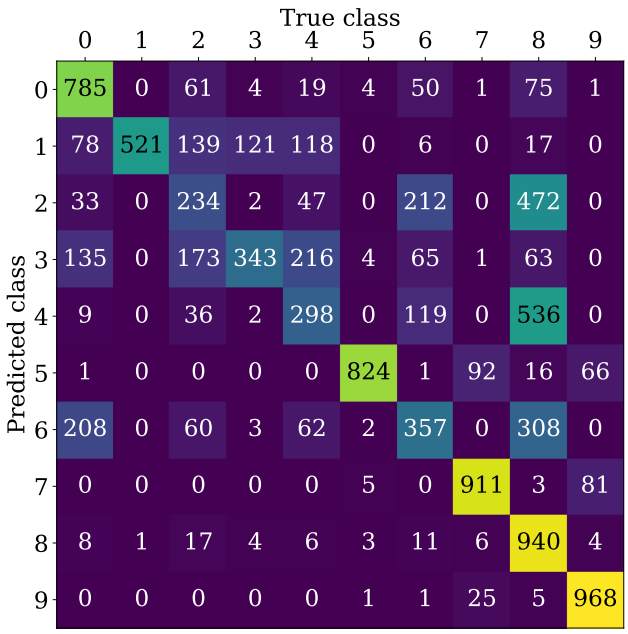

Fig. 15. CNN model best performance with achieved accuracy at 91.39%.

V. Comparison

Both MLP and CNN models can classify the 10 classes with the fashion MNIST data set. To compare the results, we listed the required time for each epoch, accuracy from each best model, and the architecture setting for each model. Due to the constraint of hardware, it is decided that epoch number is set to 9, which neither influences the result nor spending too much time during training according to Figure 6 and 13.

A. Time cost

As shown in Figure 16, it is observed that the CNN model spends more time on each epoch. The difference is over 10 times. Considering the convolution process, each image is nested for multiplications of all pixels with the 3x3 kernel. This process alone takes more than 10 times of multiplication than the MLP model. Since both models run without GPU, all multiplications have to be completed one at a time. With a good CPU, this still costs longer to process simply due to the larger amount of computation required. Here we found that CNN is much slower than MLP when involving large data set training.

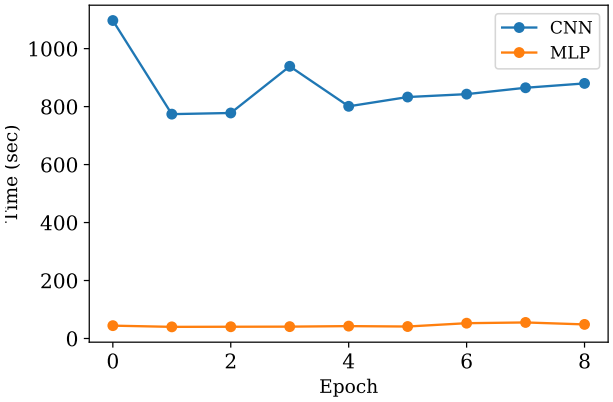

Fig. 16. Time cost in MLP and CNN model within 9 epochs. CNN spend 10 times more than MLP in each epoch.

B. Performance

To better examine the performance, both models were tested with their best hyperparameters with an epoch number of 9. In Figures 17 and 18, we can observe that the accuracy of the CNN model outperforms the MLP model from the beginning of the training as well as the loss. Both models increase the performance as the epoch number increases. One thing to note is the fluency of CNN is worse than MLP. The CNN model varies the performance at each epoch although it is increasing but not as smoothly as the MLP model. It is possibly caused by existing overfitting that is not avoided by using dropout. However, this does not influence the result that CNN indeed achieves better performance over MLP for around 3% difference.

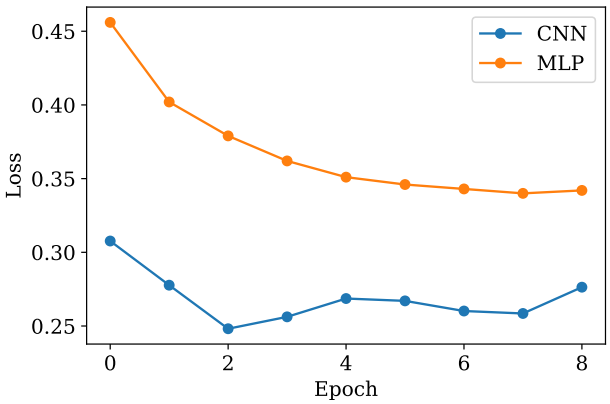

Fig. 17. Loss of MLP and CNN model within 9 epochs. CNN model possesses a lower loss score than MLP.

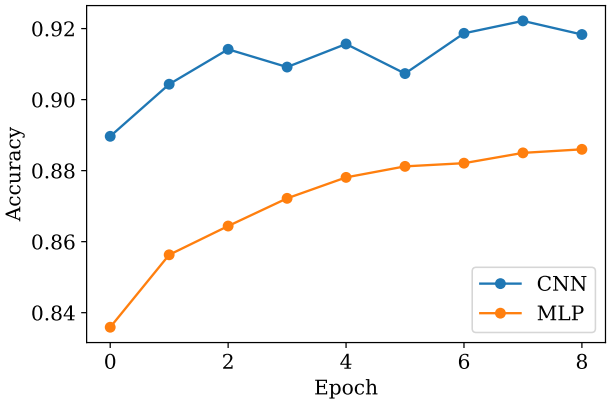

Fig. 18. Accuracy of MLP and CNN model within 9 epochs. CNN model possesses a higher accuracy score than MLP.

C. Architecture

CNN model consists of five layers including an input layer, two convolution layers, one pooling layer, and one output layer. The total number of node in all layers are 1178. MLP model has four layers containing one input layer, two fully connected layers, and one output layer with 1550 nodes. From instinct, more nodes should present better results. However, the CNN model utilizes convolution layer to extract features and then perform classification. This leads to a systematic sorting of the data compared to MLP, which computes for all connected nodes. The convolution process increases the efficiency while decreases the number of weights significantly.

VI. Conclusion

We applied MLP and CNN networks on the Fashion-MNIST dataset. Using the validation set, we searched for the optimal hyperparameters in both networks. Our MLP achieved an accuracy of 89.25% while our CNN achieved an accuracy of 91.39%, both comparable to and slightly better than literature performance. From the comparison of MLP and CNN results, we demonstrated differences between the two networks in terms of time cost, accuracy score and architecture. We

leverage CNN model more for the Fashion-MNIST dataset classification because of its high performance and architecture benefits when processing image input. MLP on the other side outperforms CNN on time cost since the less computation required while maintaining a relatively good performance. For the future work, the CNN model has a great potential for a better performance if increasing the number of epoch and conduct image augmentation on the input source.

REFERENCES

[1] K. Suzuki, *Artificial neural networks: architectures and applications*. InTech, 2013.

[2] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biolophysics*, vol. 5, pp. 115–133, 1943.

[3] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[4] ——, *Principles of neurodynamics: Perceptions and the theory of brain mechanisms*. Spartan, 1962.

[5] P. J. Werbos, "Applications of advances in nonlinear sensitivity analysis," in *System modeling and optimization*. Springer, 1982, pp. 762–770.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel distributed processing: Explorations in the microstructure of cognition*, J. L. M. David E. Rumelhart, Ed. MIT Press, 1986, vol. 1.

[7] ——, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[8] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, ser. COLT '92. New York, NY, USA: Association for Computing Machinery, 1992, pp. 144–152. [Online]. Available: https://doi.org/10.1145/130385.130401

[9] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[10] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms," *arXiv e-prints*, p. arXiv:1708.07747, Aug. 2017.

[13] A.-R. El-Shikh, "Fashion-mnist-mlp-classification-pytorch," Dec 2018. [Online]. Available: https://github.com/AbdelrahmanElShikh/Fashion-Mnist-MLP-Classification-Pytorch

[14] S. Gupta, "classification-of-mnist-and-fashion-mnist-dataset-with-mlp-and-cnn-as-classifiers-using-pytorch," Jan 2019. [Online]. Available: https://github.com/shivi1394/classification-of-MNIST-and-fashion-MNIST-dataset-with-MLP-and-CNN-as-classifiers-using-Pytorch

[15] CyrilBP, "Fashion-mnist," May 2020. [Online]. Available: https://github.com/CyrilBP/Fashion-MNIST

[16] SJ-Ray, "Fmnist_mlp_pth_classifier," May 2019. [Online]. Available: https://github.com/SJ-Ray/FMNIST_MLP_PTH_Classifier

[17] A. Bat, "Fashion mnist dataset training using pytorch," Feb 2020. [Online]. Available: https://medium.com/@aaysbt/fashion-mnist-data-training-using-pytorch-7f6ad71e96f4

[18] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv e-prints*, p. arXiv:1207.0580, Jul. 2012.

[19] N. Shirish Keskar and R. Socher, "Improving Generalization Performance by Switching from Adam to SGD," *arXiv e-prints*, p. arXiv:1712.07628, Dec. 2017.

[20] M. Bhobe, "dl-tensorflow-keras," Apr 2019. [Online]. Available: https://github.com/mjbhobe/dl-tensorflow-keras

[21] G. Preda, "Cnn with tensorflow|keras for fashion mnist," 2018. [Online]. Available: https://www.kaggle.com/gpreda/cnn-with-tensorflow-keras-for-fashion-mnist

[22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, p. 1929–1958, Jan. 2014.