Classification of Image Data

Mini project 3 – COMP 551 (Winter 2022)

ABSTRACT

In this assignment, we implemented multi-layer perceptron (MLP) and convolutional neural network (CNN) architectures. We then investigated their performance when classifying image data from the Fashion-MNIST dataset. In the case of MLP, we investigated the effect of changing the number of hidden layers, type of activation function, learning rate, dropout, and the impact of input preprocessing on the model's testing and training accuracies. In the best case, when we built a 2-layer MLP with a width of 128 units, added dropout as the regulariztion strategy, and fed the network with normalized images, we reached the test accuracy of about **89.09%.** Meanwhile, we implemented CNNs and investigated the effect of network architecture, kernel size, and dropout on the model's performance. Finally, we could attain a test accuracy of **91.14%,** when considering a CNN with 2 convolutional layers, 2 fully connected dense layers, and dropout regularization. CNNs work better than MLPs as these models care about image structure and understand spatial relations between pixels; however, when working with MLPs, the vectorized images are fed to the model, and the spatial relationship of nearby pixels is not taken into account. It is noteworthy to mention that CNNs need fewer parameters, which makes them converge faster than MLPs in terms of epochs. These advantages of CNN over MLP make it a better choice in this experiment.

INTRODUCTION

Image classification means assigning the label to an input image from a fixed set of categories (Kadam, S., 2020). One of the well-known benchmarks to test machine learning models' performance when performing image classification is the Fashion-MNIST dataset, which contains images of 10 categories of clothing. Here, we implemented MLP and CNN models to solve this classification problem.

Many researchers have tried to solve this question so far; for instance, Bhatnagar et al. (2017) introduced three different CNN architectures to classify images in Fashion-MNIST dataset. They were able to get a test accuracy of 92.54% by using a two-layer CNN paired with batch normalization and skip connections; furthermore, CNN-SVM and CNN-Softmax gave them a test accuracy of 90.72% and 91.86%, respectively (Kadam, S., 2020).

In our assignment, the model accuracy obtained via CNN implementation was 91.14%. We got this value by considering two convolutional and two dense layers; furthermore, dropout was added to this model for regularization purposes (128 filters – 3*3 kernel – batch size: 100 – early stopping). In our MLP Architecture, using two dense layers, each containing 128 units (batch size equal to 100 - 50 epochs - ReLU activation function - normalized data), we reached the accuracy of 89.09 on the test data.

DATASET AND PREPROCESSING

In this assignment, we used the fashion MNIST dataset. Xiao et al. (2017) presented the new Fashion-MNIST dataset, which is a fashion product images dataset. It comprises 28*28 grayscale images of 70,000 images, of which 60,000 are used for training and 10,000 are used for testing purposes. This dataset consists of 10 fashion item classes. (Kadam, S., 2020). Further investigation revealed that the data was equally distributed between the 10 valid labels. In the training set, there are 6,000 instances per label, and in the test set, there are 1,000 instances per label.

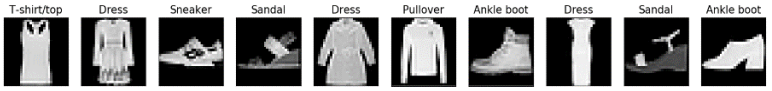


Figure 1 Fashion-MNIST dataset – 10 categories exist in this dataset

Some preprocessing steps have been applied to the data before model implementation:

I. **Data normalization:** each image was z-scored prior to being introduced to the model.
II. **Vectorization**: when working with MLP, we vectorized each image (28*28 → 784*1) and introduced this vector to the model, but when working with CNNs the 28*28 matrix itself was used.
III. **One-hot encoding**: we converted the label data into categorical formats using one-hot encoding (Greeshma, K. V., 2019).

FEATURE EXTRACTION

METHODS

IMPLEMENTED MODELS

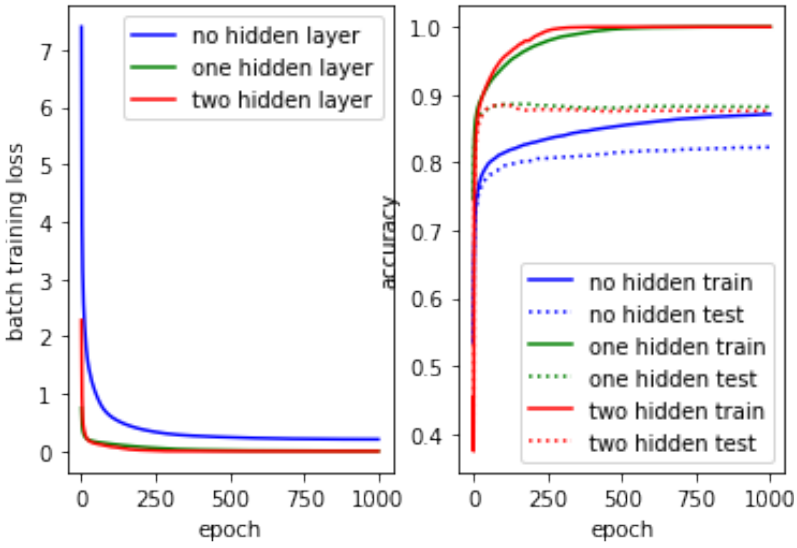
MULTI-LAYER PERCEPTRON (MLP)

A neural network is a layered structure of neurons where the knowledge of the network is contained within the connections, or weights, between the neurons. The very first layer of the network contains the input variables, and the final layer contains the output neurons. The layers in between these are called hidden layers. When a data record is evaluated using the input variables, the activations of the input layer neurons activate the neurons in the first hidden layer through the connections. This way, the activations travel through the network till the output layer is reached, where the final prediction is determined by predicting the label of the output node with the highest activation value. The activation of a neuron is determined by the linear combination of the activations in the previous layer multiplied by the corresponding set of weights.

I. Number of hidden layers

The performance results for three MLP network architectures (MLP with no hidden layer, MLP with one hidden layer, and MLP with two hidden layers) versus the number of epochs are presented in the following figure. One epoch is an iteration of the complete training set. In each epoch, the network adjusts the weights in the direction by considering the primary goal of reducing the loss function. As the iterative process of incremental adjustment goes on, the weights converge to the local optimal values (it might not be a global optimum). As a general rule of thumb, one should consider that usually, many epochs are required before training is completed (Mia, M., 2015). As expected, the test accuracy was lower than the training accuracy in all three models since the model

can perform better on the data it has previously seen but works slightly worse when unseen data is introduced to it.



epochs	Train 0 layer	Train 1 layer	Train 2 layers	Test 0 layer	Test 1 layer	Test 2 layers
10	74.87	**87.09**	84.82	73.63	**85.18**	83.47
50	79.73	91.36	**91.66**	77.87	87.65	**87.82**
100	81.37	93.92	**95.38**	78.74	**88.62**	88.39
150	82.42	95.56	**97.28**	79.56	**88.61**	87.93
200	83.1	96.84	**98.47**	80.16	**88.55**	87.45

As illustrated in the figure above, the model with no hidden layers does not work as well as the MLPs with one or two hidden layers, as it is too simple to follow the pattern within the data, and the linear activation of neurons cannot capture all the complexity of the introduced dataset. However, this model is less prone to overfitting (model with high bias).
As the number of layers increases, the models become more complex, introducing more parameters that your network needs to learn, and will be able to introduce non-linearity to the data. In this case, however, overfitting becomes more of a concern. Hence, the need for introducing a regularization mechanism becomes more significant. As illustrated by the table above, when it comes to comparing models with one and two hidden layers, one can see that the model with one hidden layer seems to work slightly better on the test data; this implies that the two-layer network might not be a need in this experiment to model data complexity; however, regularization methods would be able to make two-layers network free of this overfitted behavior. This section could be summarized by saying that if we build a very deep MLP, we might end up with a neural network that fails to generalize to newly introduced data (but it memorizes the training data).

II. Activation function

This section investigated the performance learning curves for the proposed MLP models with different activation functions in their units. Three activation function has been tested in this experiment: rectified linear unit (ReLU), leaky rectified linear unit (Leaky-ReLU), and hyperbolic tangent function (tanh).

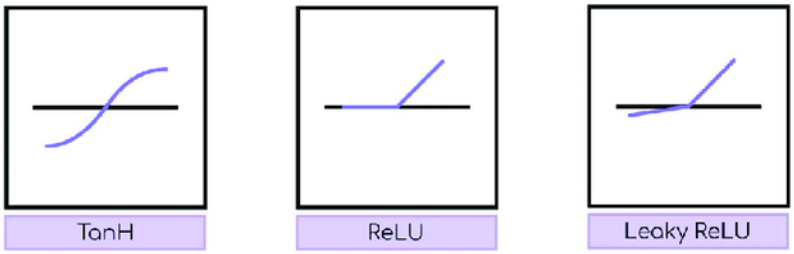


Figure 3 different activation functions under study (ReLU, Leaky ReLU, and tanh)

It has been shown that tanh can lead to the worst accuracies due to the "gradient vanishing" problem. ReLU and Leaky ReLU are better options for activation function, as they reduce the effect of vanishing gradients and make the backpropagation algorithm work appropriately with no numerical problem. As the ReLU activation function's derivative equals 0 when the unit is inactive, we initialized the weights before the learning procedure to avoid inactive units. As illustrated by the figure and table above, the Leaky-ReLU activation function seems to work slightly better as the number of epochs increases; the reason behind this observation could be the fact that Leaky-ReLU activation function has a non-zero gradient on its negative side, and this can help with the "Dying ReLU" problem which happens when ReLU face negative values and stops learning (Lu, L., 2019).

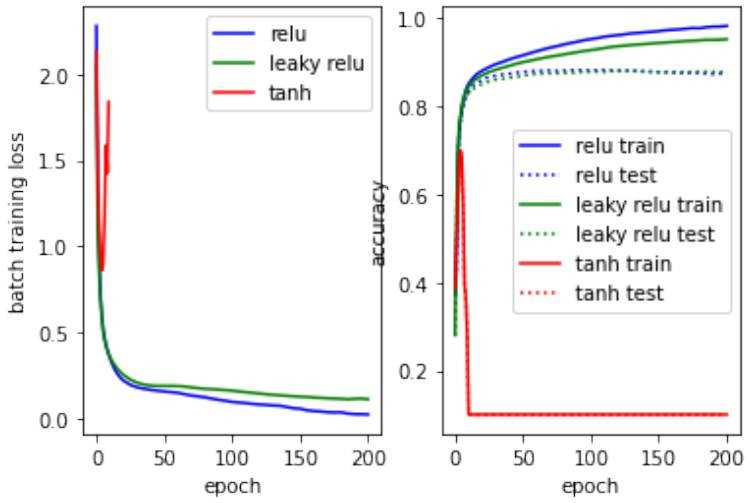


Figure 4 Effect of different activation functions on test and train performance
learning_rate=0.008, batch_size=100, num_hidden_layer=2, hidden_units=[128,128]

epochs	Train ReLU	Train L_ReLU	Train tanh	Test ReLU	Test L_ReLU	Test tanh
10	**85.08**	84.50	10	**83.51**	82.95	10
50	**91.62**	90.06	10	**87.83**	86.92	10
100	**95.32**	92.80	10	**88.17**	87.87	10
150	**97.13**	94.43	10	87.8	**87.86**	10
200	**98.31**	95.29	10	87.29	**87.88**	10

III. Dropout

MLPs with multiple hidden layers can be very powerful learning systems; however, in many cases, these systems suffer from overfitting in training, and the model may not generalize well. **Adding dropouts** is a way to solve this problem. This approach drops a random set of units and corresponding connections from the network during training and uses all the units at the test time. This method reduces the number of parameters to optimize in each training iteration and prevents units from too much co-adaptation. Each neural network can be seen as a set of small networks. Dropout selects a network from this set of parameters

at each training iteration for optimization. Since the weights of thinned networks are shared, a subset of parameters is updated at each training iteration (Salehinejad, H., 2019; Srivastava, N., 2014). Dropout has a hyperparameter p which we define as the probability of removing a unit from the network. In this experiment, we explored the effect of changing this hyperparameter on the model's accuracy.

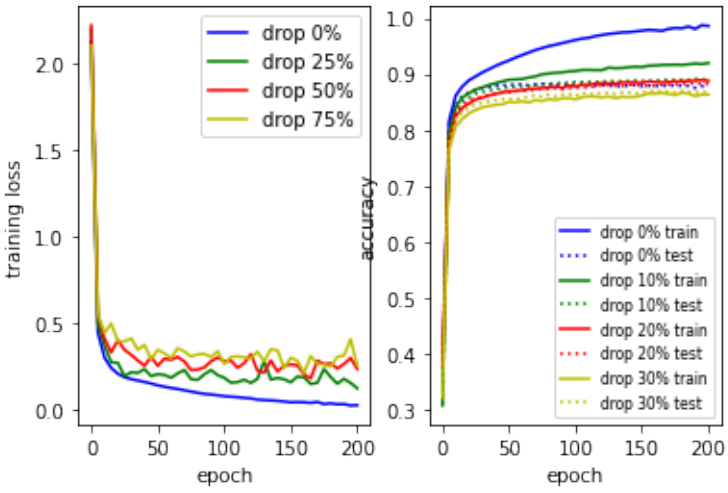


Figure 5 Effect of dropout on test and train performance
learning_rate=0.01, batch_size=100, num_hidden_layer=2, hidden_units=[128,128], activation='relu'

epochs	Train 0%	Train 10%	Train 20%	Train 30%	Test 0%	Test 10%	Test 20%	Test 30%
10	**86.16**	84.13	82.45	80.68	**84.51**	83.62	83	82
20	**88.88**	86.55	84.75	82.99	**86.78**	85.72	85.22	83.99
50	**92.42**	88.98	86.95	85.03	**88.07**	87.71	86.75	85.61
100	**96.09**	90.54	87.77	85.47	88.12	**88.58**	87.54	86.05
150	**97.74**	91.25	88.39	86.18	87.84	**88.79**	87.92	86.69
200	**98.59**	91.97	88.8	86.35	87.61	**89.09**	88.24	86.85

As illustrated by the figure above, one can observe that adding dropouts with the probability of 10% seems to make the model with two-layer hidden layers (each having 128 units) more appropriate, it can be seen that when we add no dropout (the blue curves), the test and train accuracy deviate so much and model shown a great amount of overfitting, however, by adding 10% dropout training and test accuracy curves become more similar, and the overfitting problem becomes less evident (e.g., 10% dropout means that each unit would be retained in the model when training, by the probability of 90%).

IV. Input Normalization

To obtain the previous results, we used the normalized input images (z-scored images). This section aims to study why this normalization is recommended. Here, we implemented MLP with two hidden layers, each with128 units with ReLU activation function; for training and testing procedures, we once used the unnormalized images and once used the normalized data.

We should note that it can be a better option to feed MLP with normalized data, as when working with normalized data, the weights that we found for our models would not be affected by feature scales. Standardization can make the training faster and more well-behaved by improving the numerical conditions and reducing the probability of being stuck in local optima.

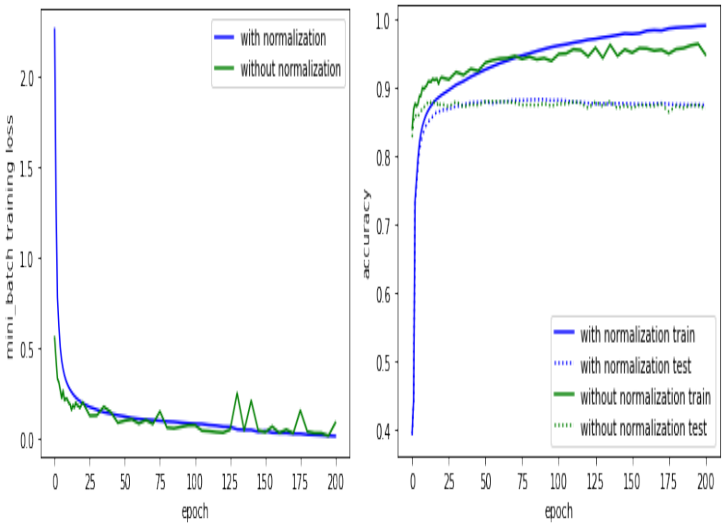


Figure 6 Effect of data normalization on test and train performance
learning_rate=0.01, batch_size=100, num_hidden_layer=2, hidden_units=[128,128], activation='relu'

epochs	Train norm	Train no norm	Test norm	Test no norm
10	86.30	**90.41**	84.68	**87.82**
20	88.96	**91.56**	86.75	**87.66**
50	92.71	**93.72**	87.96	**88.03**
100	**96.22**	94.99	**88.24**	87.69
150	**97.91**	95.17	**87.68**	87.08
200	**99.13**	94.89	**87.5**	87.02

V. Learning rate (extra investigation)

Finally, we would like to see how the learning rate can affect the training and test accuracy curves. The higher the learning rate, the quicker the training curve converges. So, when the learning rate is higher, the training accuracy can get to its maximum in fewer epochs. Meanwhile, too large values can lead to learning sub-optimal weights and an unstable training process. Actually, very large learning rates would result in significant weight updates. Its manifestation would be the oscillation of performance curves over training epochs (as we see in the case of learning rates equal to 0.02 and 0.05 where the cures are no longer that smooth). Too small learning rates, on the other side, may result in a long training procedure and can get stuck (as we see in the case of LR equal to 0.0001).

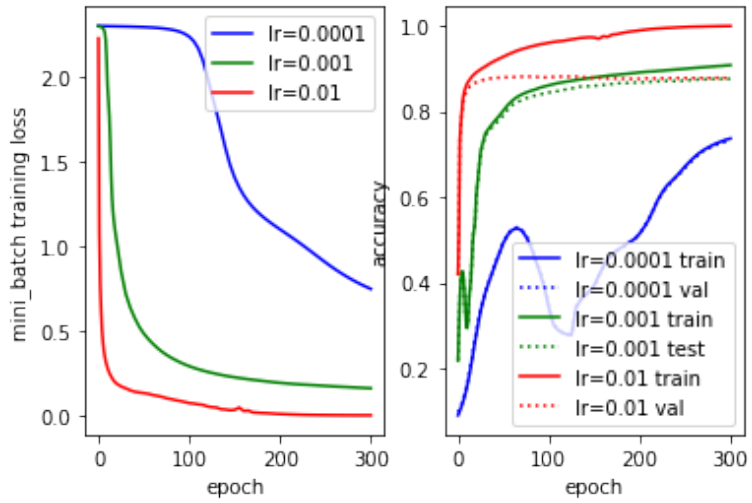


Figure 7 Effect of learning rate on test and train performance

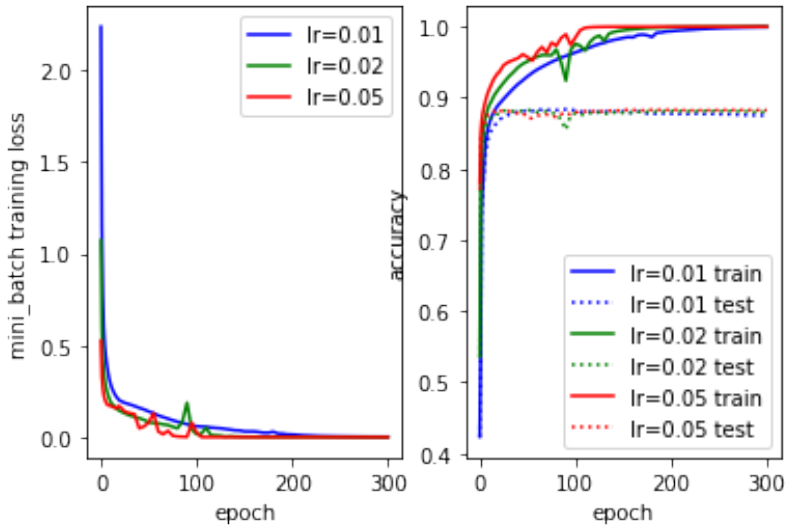


Figure 8 Effect of learning rate on test and train performance
batch_size=100, num_hidden_layer=2, hidden_units=[128,128], activation='relu'

LR	10 Epochs train	50 epochs train	150 epochs train	200 epochs train	10 epochs test	50 epochs test	150 epochs test	200 epochs test
0.0001	15.87	49.33	39.35	51.45	16.42	49.57	39.22	51.48
0.001	29.40	80.18	87.88	89.07	29.53	79.07	85.94	86.74
0.01	86.42	92.03	97.2	98.88	84.83	87.9	87.74	87.57
0.02	88.99	95.11	99.48	99.91	86.83	88.25	87.9	88.12
0.05	91.22	95.69	99.98	99.98	87.8	87.11	88.26	88.29

CONVOLUTIONAL NEURAL NETWORKS (CNN)

A convolutional neural network (CNN) is one of the most well-known deep neural networks discovered by inspiring form mammals' visual system. CNN has multiple layers, namely the convolutional layer, pooling layer, and fully connected layer. The main objective of the convolutional layer is to obtain the features of an image by sliding a smaller matrix (kernel or filter) over the entire image and generating the feature maps. Each value from these filters is then passed through an activation function (ReLU, sigmoid, etc.). The next step in CNNs would be a pooling layer that replaces the network's output at certain locations by calculating a statistic measure (like calculating the maximum) of the nearby outputs and help reduce the output size by aggregating information together. This step helps with decreasing computation. Finally, the extracted features are flattened and input into a multi-layer perceptron architecture. Compared to MLPs, CNNs require fewer parameters to learn, which makes them require fewer data to get trained (Kadam, S., 2020; Çayir, A., 2018). We expect to see that CNN performs better than MLP when the input data is an image since the structure of the input itself and the relative position of pixels with regard to each other is considered when using CNNs.

Note that we first put 10,000 samples from the original training set aside as validation in the following experiment and trained the CNN models using the remaining 50,000 instances. We computed the weights using the train set, investigated the model's performance with those weights on the validation set, and then reported the best model's performance on the test set. The early stopping method was used to stop the training procedure when validation error started increasing to avoid overfitting.

It should be mentioned that in the following runs, batch size was always set to 100, the reason behind this choice was the report by Kadem, who investigated the rule of batch size on model performace of CNNs when the goal was fashion_MNIST image classication, they reported no significant role for batch size; however the accuracy on the test size was slioghly better in case of batch size equal to 100 (Kadam, S., 2019).

I. Effect of kernel size

First, we investigated the role of kernel size on CNN's accuracy. We used two convolutional layers and two fully connected (dense) layers in these architectures (CONV2D: 3*3 or 5*5 or 7*7 size, 32 filters; Activation (ReLU); POOL: 2*2; CONV2D: 3*3 or 5*5 or 7*7 size, 64 filters; Activation (ReLU); POOL: 2*2; Flattening; Dense layer with 128 Hidden Neurons; Activation (ReLU); Dense layer with 128 Hidden Neurons; Activation (ReLU); Dense layer with 10 Hidden Neurons; Activation (softmax)). The training and validation accuracy and loss curves are illustrated below. To report the test performance, we looked back into the stored weights of the models and considered the model which had led to the best validation accuracy (reported by the early stopping method); we used that model to report the unseen data accuracy/loss. The results suggest that using 3*3 kernel size can be the best choice. It is noteworthy to mention that we did not test for 1*1 kernel size as if we use such a kernel, the extracted features would be so local, and no information about the neighboring pixels would be mixed up to create a more informative feature.

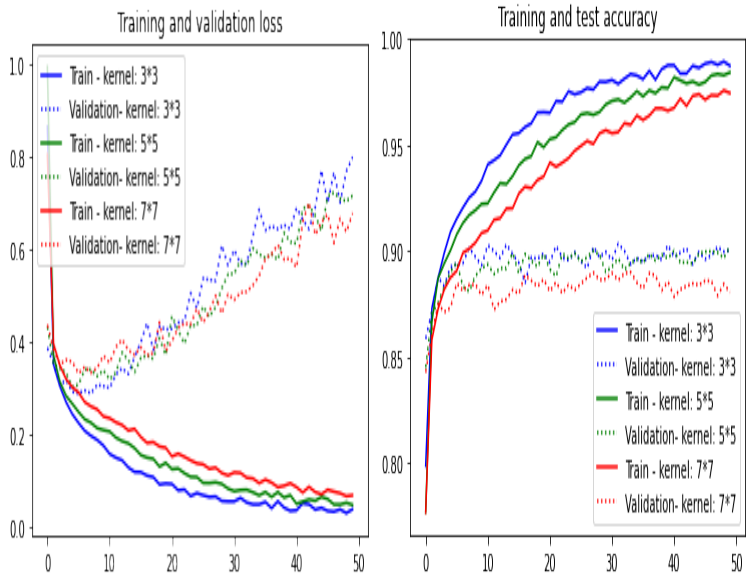


Figure 9 Effect of kernel size on validation and train performance

Kernel size	3*3	5*5	7*7
Test accuracy	**89.23**	88.72	87.70

II. Effect of number of filters of convolutional layers

The other hyperparameter that we investigated was the effect of the number of filters of the convolutional filter on model accuracy. We tried 32 – 64, 64 – 64, and 128 – 128.

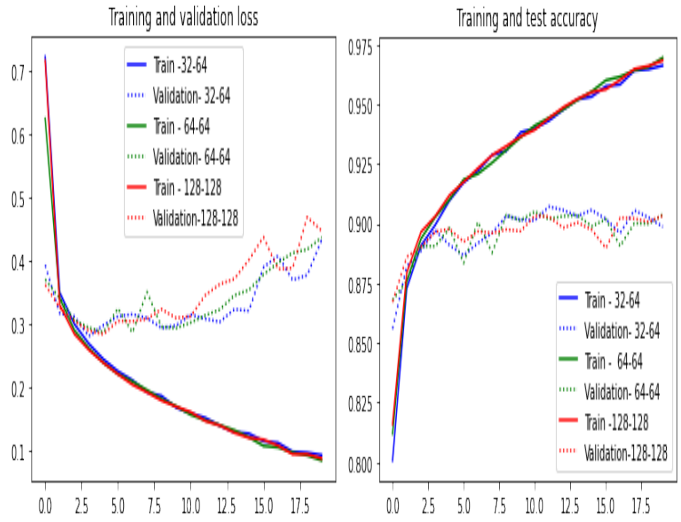


Figure 10 Effect of number of filters of convolutional layers on validation and train performance

Filter	32-64	64-64	128-128
Test accuracy	88.76	88.34	**89.58**

III. Dropout

This section investigated the effect of adding the dropout regularization method to the CNN implementation. It is clear that when we add dropout to the CNN models, the generalization of the model becomes more evident; these models have validation accuracy curves that are located above the validation accuracy curve obtained when no dropout was added (the dotted blue curve); hence, one can say that adding dropout is helping the model generalization with no doubt.

Among the values we test, we can see that the more dropout we do while training, the more generalized the model becomes (the validation and curve sets become more similar).

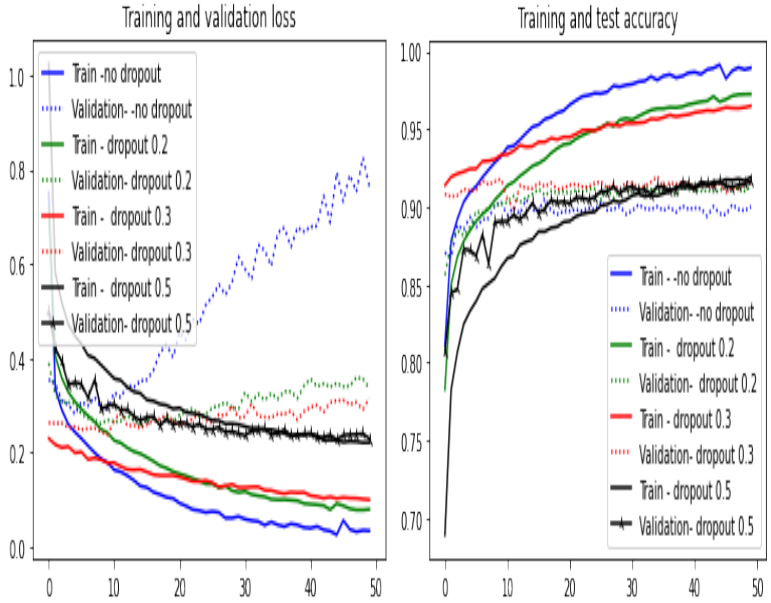


Figure 11 Effect of adding dropout on validation and train performance

dropout	0%	20%	30%	50%
Test accuracy	89.14	90.85	**91.14**	90.85

SUMMARY OF THE RESULTS

The MLP model that gave us the best test accuracy:

- Epoch: 200
- **Structure:**
  2 hidden layers
  Each having 128 units
  Dropout (0.1 probability of removing a unit during training procedure)
  ReLU activation function
- Learning rate: 0.01
- Batch size: 100
- Loss: cross entropy

The CNN model that gave us the best test accuracy:

- Epoch: identified by early stopping
- **Structure:**
  CONV2D: 3*3 size, 128 filters
  Activation (ReLU)
  POOL: 2*2
  CONV2D: 3*3 size, 128 filters
  Activation (ReLU)
  POOL: 2*2
  Flattening.
  Dropout (0.3)
  Dense layer with 128 Hidden Neurons.
  Activation (ReLU)
  Dropout (0.3)
  Dense layer with 128 Hidden Neurons.
  Activation (ReLU)
  Dense layer with 10 Hidden Neurons.
  Activation (softmax))
- Batch size:100
- Loss = cross entropy
- Optimizer: Adam

DISCUSSION

Suffice it to say that when the goal is image classification, CNN models can work better than MLPs as they consider the spatial relationship of nearby pixels. We investigated the need for regularization methods, such as dropout, when working with models with many parameters such as MLPs and CNNs and saw how adding dropout could avoid model overfitting and generalization. When working with MLPs, we saw how adding layers to MLPs can make them more complex and how reducing them can introduce less expressive models. Depending on the dataset type, we might require different number of layers. We saw how the units' activation function could affect the model and noticed that activation functions such as tanh and sigmoid might make the MLP model fail as they make the model face the vanishing gradient problem. We also investigated the importance of input normalization and noticed how it could make the performance curves well-behaved. Finally, we studied the effect of learning rate on accuracy curves and saw how very small or very large values for learning rate adversely affect models. At last, we noticed how CNN architecture and parameters of convolutional layers such as kernel size can improve or degrade models' performance. Hence, all these parameters need to be tuned for a newly introduced classification problem to get the maximum possible efficiency.

STATEMENT OF CONTRIBUTIONS

- Asa Borzabadi Farahani: CNN implementation + report
- Gurucharan Marthi Krishna Kumar: MLP implementation
- Yunhan Li: MLP implementation

REFRENCES

1. Kadam, S. S., Adamuthe, A. C., & Patil, A. B. (2020). CNN Model for Image Classification on MNIST and Fashion-MNIST Dataset. *Journal of Scientific Research*, *64*(2), 11.
2. Salehinejad, H., & Valaee, S. (2019, May). Ising-dropout: A regularization method for training and compression of deep neural networks. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 3602-3606). IEEE.
3. Çayir, A., Yenidoğan, I., & Dağ, H. (2018, September). Feature extraction based on deep learning for some traditional machine learning methods. In *2018 3rd International Conference on Computer Science and Engineering (UBMK)* (pp. 494-497). IEEE.
4. *Greeshma, K. V., & Sreekumar, K. (2019). Hyperparameter optimization and regularization on fashion-MNIST classification.* International Journal of Recent Technology and Engineering (IJRTE)*,* 8*(2), 3713-3719.*
5. Mia, M. M. A., Biswas, S. K., Urmi, M. C., & Siddique, A. (2015). An algorithm for training multilayer perceptron (MLP) for Image reconstruction using neural network without overfitting. *International Journal of Scientific & Technology Research*, *4*(02), 271-275.
6. Lu, L., Shin, Y., Su, Y., & Karniadakis, G. E. (2019). Dying relu and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*.
7. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, *15*(1), 1929-1958.