KMNIST Classification Problem using Multi-Layer Perceptron (MLPs) and Convolutional Neural Networks (CNNs)

This work is presented as a project for
EEL6814 – Deep Learning Course at the University of Florida

Group Members
Gijung Lee
Mayar Shahin

KMNIST classification assesment using Multilayer perceptron and Convolutional Neural Netwrks

Mayar Shahin
Physics department
University of Florida
mayar.shahin@ufl.edu

Gijung Lee
Electrical Engineering department
University of Florida
lee.gijung@ufl.edu

Abstract—This work is presented as a project for **EEL6814 – Deep Learning Course.** Over the last few decades, deep neural networks have proved very powerful in handling big data. In pattern recognition, particularly image problems, deep learning's performance surpasses classical machine learning in accuracy. The Multilayer Perceptron (MLP) and the Convolutional Neural Network (CNN) are two of the most popular deep learning architectures for classification. We present the two networks' performance to classify the Japanese KMNIST data set. In this report, we test the performance of different MLP and CNN architectures on the data set. We further tune hyperparameters and investigate how different parameters affect the classification efficiency.

I. Introduction

A. *The Multilayer Perceptron (MLP)*

A multilayer perceptron (MLP) is a type of of feedforward artificial neural network (ANN). The MLP consists of an input layer, an output layer and at least one hidden layers. Each layer has nodes which in general have nonlinear activations. MLPs are fully connected meaning that each node in one layer connects to every node in the following layer through a weight. The weights are learned by the MLP using an error back propagation algorithm [1].

There are a few hyperparameters involved in an MLP that affect performance. First, we need to know the role of the number of layers and the number of processing elements (PEs) or units in a layer. The PE creates a discriminant function thus making a decision boundary and allowing the network to represent different classes. Therefore, more layers help the MLP to classify easily with fewer numbers of PEs [2]. Second, an important hyperparameter is the learning rate. A small learning can get the learning stuck at a local minimum, while a higher learning rate helps the MLP learns optimal weights quickly but risks jumping around minima leading (instable learning). We need to select the proper learning rate to reach a global minimum.

B. *Convolutional Neural Networks (CNNs)*

The CNN is a neural network that has convolutional layers. In the convolutional layers, a ***filter*** is used to extract the features of the image. The filter is an $L \times L$ matrix that slides over the image sequentially from left to right and from top to bottom. The stride determines how many pixels will be moved when the filter moves. The stride affects the size of the output.

After this process, the filter has $L \times L$ weights and an output. The output is called a feature map. Each filter extracts features from the output in the previous layer. To get different features, we need multiple filters. As we have more layers, we can extract more features in the data. However, this process can lose edge pixel information. To solve this problem, ***padding*** is used by adding pixels that have a value of 0 or 1 to the sides of the input data [3]. Next step in a CNN is typically a pooling layer, which serves in down-sampling the feature maps by summarizing the sample with a patch. The job of convolutional layers is to extract features of the input. Afterwards we add one or more dense fully connected layers. The job of the dense layers is to do the actual classification/regression problem at hand.

II. Methods

A. *Data Preparation*

We download the publicly available data set **Kuzushiji-MNIST** (KMNIST) [1]. With advancement in deep learning and how well current classifiers work on the MNIST data set, more complicated data set is need to benchmark performance. KMNIST is used as a replacement for the MNIST dataset (28x28 grayscale, 70,000 images). It consists of 10 classes each representing 10 characters of Hiragana, a Japanese writing system, see Fig. 1. The data set is divided into 60000 training samples and 10000 testing samples. We set aside 10% of the training set for model validation as the learning is done.

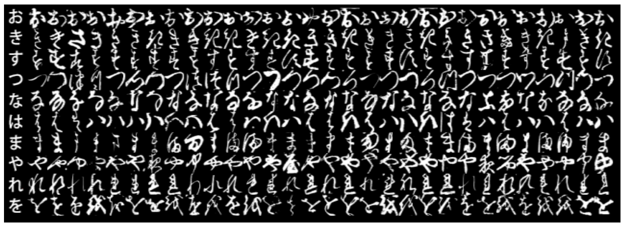


Figure 1: The 10 classes of KMNIST, with the first column showing each character's modern hiragana counterpart. [1]

B. *Multi-Layer Perceptrons (MLPs)*

To get optimal performance for classification, we used 2000 units in the first layer, 1000 units in the second layer, 500 units in the third layer, 250 units in the fourth layer. We included dropouts and batch normalization layers to avoid overfitting because this model was easily overfitted on the train set with many layers and units.

C. *Convolutional Neural Networks (CNNs)*

Working on this data set, we fix some hyperparameters before tuning the rest. Because we care about smaller details in these characters to get classification, we used convolutional filters of 3x3 size with stride is 1. We enabled padding to match input and output sizes. The pooling layer is of size 2x2, with stride 2 and no padding. We set the layer to Maxpooling because we care about detecting the brightest spots in the image. A CNN architecture can be complex with multiple tunable hyperparameters. Optimally, we want to try as many hyperparameters as feasible to reach optimal performance. A deep CNN can have up to hundreds of convolutional layers e.g. GoogLe net. Basic CNNs involve convolution-pooling pairs of layers as their building units followed by an MLP_like part (dense layers), illustrated in Figure 1. Many architectures involve a series of convolutions before pooling e.g. VGG16.

*D. **Activation function***

Throughout this work, we use the Rectified Linear Unit (ReLU) as the activation function for all convolutional and fully connected nodes. $f(x) = \max(0, x)$. ReLUs have a few significant advantages over other nonlinear activations: 1) They are computationally efficient since it just chooses between two numbers. 2) It avoids the problem of vanishing gradients encountered in deep neural networks trained with backpropagation. Tanh and sigmoid functions for instance are highly saturated away from their midpoints leading to vanishing grafients that slow down the learning. On the other hand the gradient of the ReLU is $f'(x) = 0$ for $x < 0$ and is $f'(x) = 1$ for $x > 0$. That means that adding layers in your network does not run into a numerical problem because multiplying the gradients will neither vanish nor explode. 3) In practice ReLU shows better performance [6,7].

*E. **Drop out***

To avoid overfitting, there are several methods. One of the methods is a dropout. The dropout makes a model not to train all the units at every iteration. The units will be selected randomly that are trained or not trained. We can decide how many units are trained or not trained by percentage. The unit cannot rely on some specific weights because the units are randomly selected every iteration. This acts as a regularization. This method can help the model to be efficient in the computational cost and not to be overfitted.

*F. **Batch Normalization***

The batch normalization standardize the current batch data by subtracting the mean of current layer's batch's activations and dividing by the standard deviation of the batch's activations. Batch normalization's two main advantages are: 1) Prevent oversaturation, a problem partially addressed by using ReLU. 2) According to the original paper [8], the distribution of the inputs to layers deep in the network may change after each mini-batch when the weights are updated. This can cause the learning algorithm to forever chase a moving target *"Internal Covariate Shift"*. Batch normalization is introduced to standardize the inputs by fixing a mean and variance for each layer which helps in decoupling the change in layers. This process is supposed to speed up the learning process because it allows for stability even when using higher learning rates. It also acts as a regularizer and decreases the need for dropout.

*G. **Early Stopping***

As the learning proceeds, the model can run into over-fitting the training set, thereby reducing accuracy on test data or data not previously seen by the model. To avoid this problem, we utilize early stopping. We take away part of the training set for validation and calculate the accuracy of classification in the validation set. As the training accuracy increases, the validation accuracy increases until it reaches a point where the model starts to over fit the training. If the model keeps decreasing the validation accuracy for a number of epochs (set by the patience parameter), we choose to stop learning and retrieve the most general model.

*H. **Optimizers***

We will use optimizers to update the weights of the network. In this project, we used two types of optimizers, the first one is called stochastic gradient descent (SGD) and the second one is called adaptive moment estimation (Adam). SGD is a basic form of gradient descent. SGD updates weights by subtracting the gradient multiplied learning rate from previous weights. SGD uses a few samples that are randomly selected for each iteration. Adam is mixture of the root mean sqaure prop (RMSProp) and Momentum. Adam updates the weights with squared gradient that is from calculated an exponential weighted moving average of the gradient. We tried both to see how those optimizers affect our MLP and CNN models.

*I. **Confusion Matrix***

For testing the final model's performance, we use a confusion matrix. The confusion matrix is a table presenting the number of classifications from the predicted class that match the actual class. Diagonal elements represent correct predictions, while off-diagonal represent confusions.

Actual / Predicted	Class 1	Class 2	Class 3
Class 1	🟩	🟥	🟥
Class 2	🟥	🟩	🟥
Class 3	🟥	🟥	🟩

Table 1: An example of a 3-class confusion matrix. Green represents correct predictions, while red represents confusions by the model.

III. RESULTS

*A. **Multi-Layer Perceptrons(MLPs)***

To find the best hyperparameters, we tried several experiments.

a) Effect of increasing the number of units in the hidden layers. First, we tried to find the effect of the number of units in the hidden layers. For this experiment, we used an MLP model with two hidden layers. Each hidden layer has different number of units. In the first set we used 10 units in the first layer and 5 units in the second layer. Second, we multiplied the number of units by 10. In the third set, we used 100 times more units than the first set. We trained each set for 10 epochs. The loss is getting much lower and the accuracy is getting much higher as the model has more units in each layer, Fig. 2.

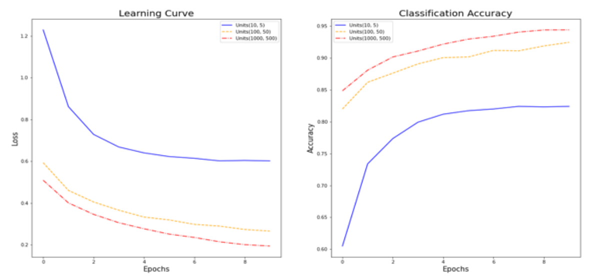


Figure 2: Learning curve and accuracy on the validation set when each layer has different number of the units.

Units	(10, 5)	(100, 50)	(1000, 500)
Test-Acc	66.72%	82.82%	87.86%

Table 2. Accuracy on the test set for different number of the units.

In table 3, the accuracy on the test set is getting much higher when the number of units is more. We notice that increasing the number of units helps to get better results. The first layer decides decision boundaries and the second layer captures convex regions. If we have more units in each layer, we can get more decision boundaries and convex regions, leading to better results.

Units	(1000, 500)	(2000, 500)	(1000, 1000)	(2000, 1000)
Test-Acc	87.37%	88.18%	87.19%	88.05%

Table 3. Accuracy on the test set by different number of the units.

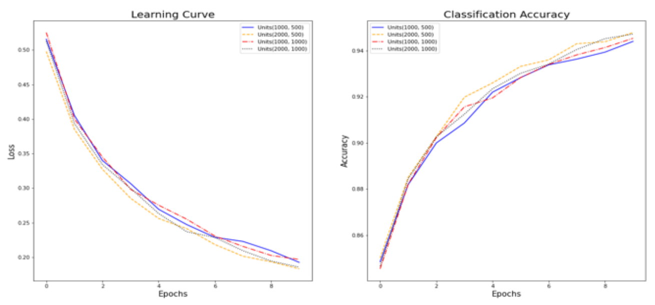


Figure 3. Learning curve and accuracy on the validation set when each layer has much larger number of the units.

b) Effect of increasing the number of hidden layers. As seen in Fig. 3, increasing the number of units in the first layer gets effectively better result than increasing the number of the units in the second layer. Second, we tried to find the effect of the number of the hidden layers. For this experiment, we used three types of MLP models, each model has 2 layers or 3 layers or 4 layers. As you can see the train accuracy graph in Fig. 4, the train accuracy is going higher as the number of layers is increased. However, we noticed that the model that has more layers can be overfitted. As you can check validation graph in Fig. 4, the accuracy of the model that has 4 layers doesn't keep going higher. Even the accuracy is lower than before in some epochs. This means that the model is overfitted. We may need more data to a get better result when we use many layers. The small amount of data is not enough to allow the MLP with many layers to learn features.

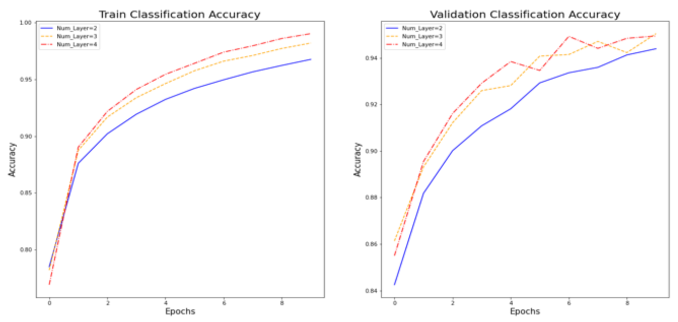


Figure 4. Accuracy on the train and validation sets.

Number of layers	Train	Valid	Test
2	96.76%	94.40%	87.93%
3	98.20%	95.05%	88.82%
4	99.03%	94.95%	88.97%

Table 4. Accuracy in different number of layers.

In table 4, you can check that the model that has 3 layers is the proper model to classify the dataset. It has almost similar test accuracy and highest validation accuracy. Third, I tried to find best learning rate for training the model.

Learning rate	Train	Valid	Test
0.01	96.8%	94.5%	87.2%
0.1	100.0%	96.4%	91.8%
0.5	99.7%	96.5%	91.3%

Table 5. Accuracy in different learning rate.

c) Effect of different learning rates. In table 5, we could check that the increased learning rate helps to get better results. However, it is not always right to use higher learning rate. When the learning rate is 0.1, the results are better than when the learning rate is 0.5. In Fif. 5, you can see that the loss is keep going lower when the learning rate is 0.1 however, the learning curve is not stable when the learning rate is 0.5. The loss goes to divergence. The 0.5 is too high for the learning late. We need to carefully choose the learning rate.

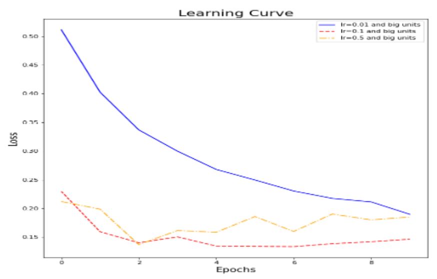


Figure 5. Learning curve in different learning rate.

d)* Effect *of dropout. To avoid overfitting and get better results, there are two ways. One way is dropout, and another way is batch normalization. As you can see table 6, we could get higher accuracy on the test set.

Dropout	Train	Test
0	100.00%	91.78%
1	99.54%	92.03%

Table 6. Accuracy of the models. 0 means that no dropout in the model, 1 means that one dropout used in the model.

The learning curve in Fig. 6 shows that, during training the model that has no dropout, the loss goes low faster than another model that has a dropout layer. When you see the accuracy graph in figure 5, the accuracy from model that has no dropout reaches 100 % however, the actual best accuracy on the test set reaches 92.03 % with dropout layer. You can check that the dropout helps to prevent overfitting by checking lower accuracy on the train set but higher accuracy on the test set.

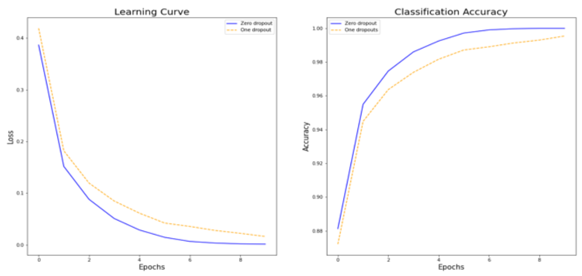


Figure 6. Dropout-Learning curve and Accuracy on the train set.

e)* Effect *of batch normalization. We had similar results when we used batch normalization. When you see table 7 and Fig. 7, the results show similar aspects. The best accuracy on the test set is 92.23 %.

Batch-Norm	Train	Test
0	100.00%	91.53%
1	99.54%	92.23%

Table 7. Accuracy of the models. 0 means that no batch normalization used in the model, 1 means that one batch normalization is used in the model.

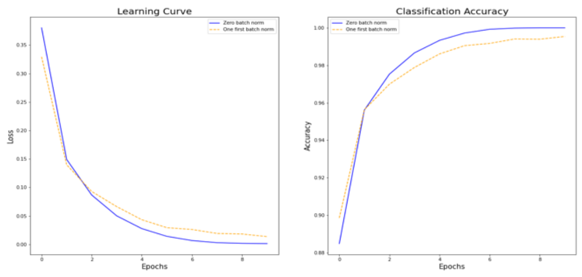


Figure 7. Batch normalization-Learning curve and Accuracy on the train set.

***f)* Conclusion.** With these results, we found the optimal hyperparameters. The final model uses 4 layers, each layer had 2000, 1000, 500, and 250 units. We also included dropout and batch normalization. With these hyperparameters, we could reach an accuracy of 92.3% on the test set. Moreover, as you can see in Fig. 8, we could avoid overfitting even while using large number of the units and layers by using dropouts and batch normalizations. As the model is not overfitted, weI could train the model with more epochs (15). With these choices, we could get an accuracy 92.94 % on the test set.

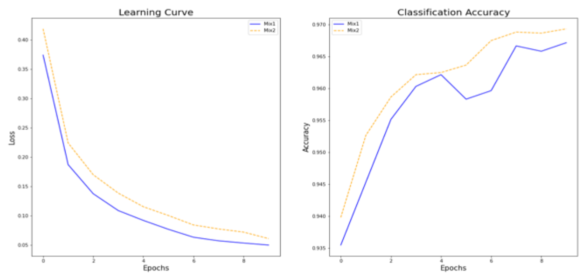
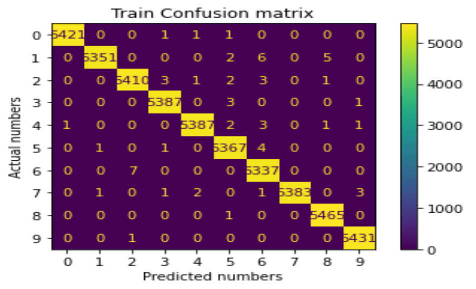


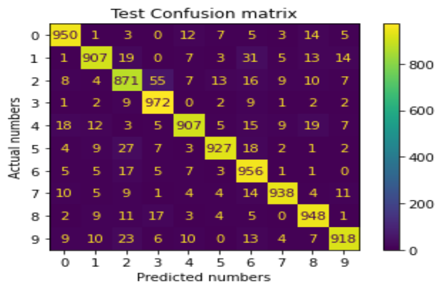
Figure 8. Learning curve on the train set and Accuracy on the valid set. Mix 1: no dropout and batch norm (blue line), Mix 2: dropouts and batch norms included (yellow dashed line)



	0	1	2	3	4	5	6	7	8	9
0	5421	0	0	1	1	1	0	0	0	0
1	0	5351	0	0	0	2	6	0	5	0
2	0	0	5410	3	1	2	3	0	1	0
3	0	0	0	5387	0	3	0	0	0	1
4	1	0	0	0	5387	2	3	0	1	1
5	0	1	0	1	0	5367	4	0	0	0
6	0	0	7	0	0	0	5337	0	0	0
7	0	1	0	1	2	0	1	5383	0	3
8	0	0	0	0	0	1	0	0	5465	0
9	0	0	1	0	0	0	0	0	0	5431

Figure 9. Confusion matrix on the train set.

As you can see figure 8, the model classifies the letters well on the train set. However, in figure 9, the model has difficulties to classify letters on test set especially for class 1 and class 6. Those letters are difficult to distinguish by human also as you see figure 10.



	0	1	2	3	4	5	6	7	8	9
0	950	1	3	0	12	7	5	3	14	5
1	1	907	19	0	7	3	31	5	13	14
2	8	4	871	55	7	13	16	9	10	7
3	1	2	9	972	0	2	9	1	2	2
4	18	12	3	5	907	5	15	9	19	7
5	4	9	27	7	3	927	18	2	1	2
6	5	5	17	5	7	3	956	1	1	0
7	10	5	9	1	4	4	14	938	4	11
8	2	9	11	17	3	4	5	0	948	1
9	9	10	23	6	10	0	13	4	7	918

Figure 10. Confusion matrix on the test set.

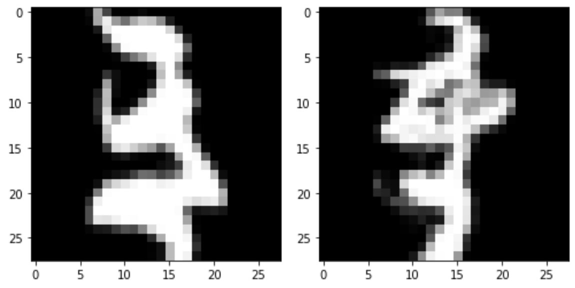


Figure 11. Letters in class 1 and class 6.

B. *Convolutional Neural Networks (CNNs)*

For the following set of experiments, the dense layers following the convolutional parts of the CNN are set to two dense layers. The first dense layer has 128 units, the second has 64, both with ReLU activations. The final layer consists of 10 units with a SoftMax activation function for classification. The output of the network is interpreted as a probability for the image to belong to each of the 10 classes.

a) Effect of increasing the depth of the CNN. In this project, we choose to start by changing the number of convolutional-pooling layer pairs. We test the performance on 3 different CNNs that have 1,2 and 3 layers. Adding a second layer performs significantly better on both validation and test sets, see Fig. 12. Although we do not see a big improvement to validation accuracy by adding a third layer, the test accuracy goes up by 1%, Table 8.

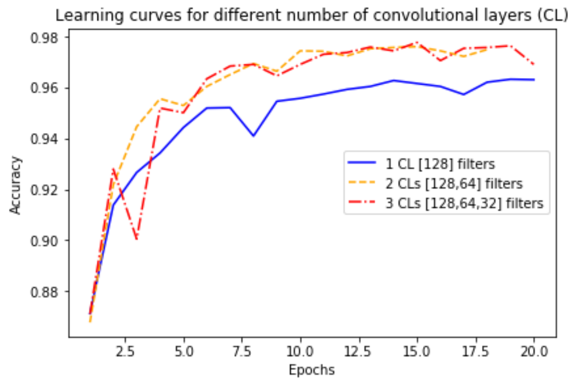


Figure 12: Comparison of CL performance of test set classification on three different architectures

	Training Accuracy	Validation Accuracy	Test Accuracy
(128)	99.36%	96.33%	91.59%
(128,64)	99.43%	97.62%	93.66%
(128,64,32)	99.22%	97.78%	94.14%

Table 8: Performance comparison between different number of layers.

b) **Effect of increasing the number of filters each layer of CNN.** In this experiment we consecituvely halve the number of filters per layer and monitor the effect on the accuracy of the classification. The valiation accuracy does not seem to improve see Fig. 13. However, the test accuracy significantly decreases as we decrease number of filters, see Table 9.

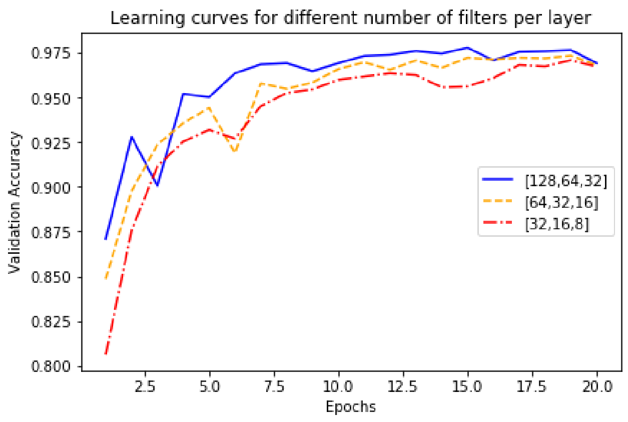


Figure 13: Comparison of CNN performance using different number of filters per convolutional layer. Legend: [# of filters in layer 1, # of filters in layer 2, # of filters in layer 3]

	Validation Accuracy	Test Accuracy
(128,64,32)	97.78%	94.14%
(64,32,16)	97.33%	92.17%
(32,16,8)	97.13%	91.79%

Table 9: Performance comparison when using less filters per layer.

c) **Effect of turning batch normalizaton on.** We try turning on batch normalization for two of our trained networks and investigate the effect. As explained in the MLP section, we see better results. Accuracy results are presented in table 10.

Batch Normalization	Training Accuracy	Validation Accuracy	Test Accuracy
On	98.86%	97.33%	92.17%
Off	99.70%	97.72%	93.14%

Table 10: Performance comparison between different number of layers.

d) **Effect of using dropout. We try different dropout percentages and placements.** First we add dropout to dense layers only. Then, we add dropout after convolutional layer too and compare performance. As seen in Fig. 14, adding dropout does not improve the accuracy significantly. It slightly improves the test accuracy, Table 10. Plotting the training accuracy shows how dropout makes sure the model does not overfit the training data. Dropout of 25% in both convolution and dense layers drastically decreases the training fit after the same number of epochs. We are suspecting that the dropout increase in test accuracy was insignificant because we had already applied batch normalization which acts as a regularizer on its own and decreases the need for dropout.

Dropout	Training Accuracy	Validation Accuracy	Test Accuracy
(0,0)	99.70%	97.72%	93.14%

(0.25,0.25)	98.37%	98.08%	93.40%
(0,0.25)	99.35%	97.65%	93.67%

Table 10: Performance comparison for different dropout structures. Dropout (c,d) denotes a dropout of value (c) after each convolutional layers and a dropout of value (d) after each dense layer.

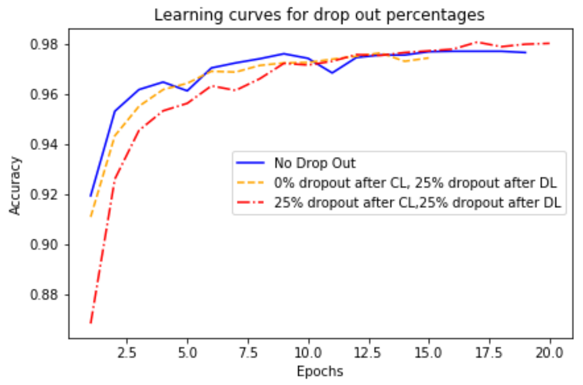


Figure 14: Comparison of CNN performance with different drop out setups. The yellow curve represents adding no dropout layers after convolutions and adding a 25% drop out after every dense layer before the final layer. The red curve adds a drop out after each CL and DL.

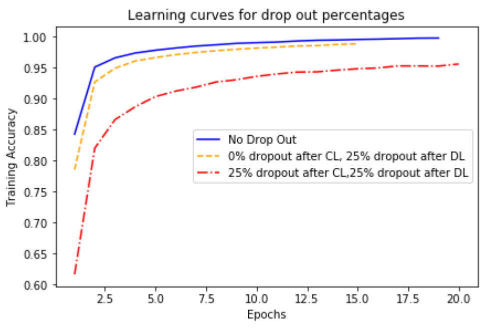


Figure 15: Comparison of CNN training curves for different dropouts.

***e)* Effect of using different optimizers and changing corresponding learning rates.** For further tuning the model, We try out different optimizer algorithms. We fix the CNN architecture and only change the algorithm. Accuracy of predictions are presented in table 11 and Fig. 16. We compare SGD and Adam optimizers with learning rates (0.01,0.05,0.1) and (0.001, 0.01) respectively. SGD with the lowest learnng rate performs significantly worse and plateaus at a lower accuracy. As we increase learning rate, we see improvement in classification accuracy. Using Adam optimizer yields simmilar results but has the advantage of being significantly faster.

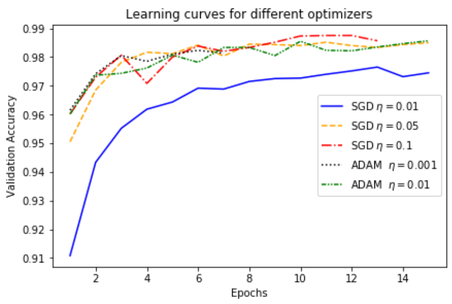


Figure 16: Comparison of CNN performance with different optimizers. Curves not extending to the full numbers of epochs were early stopped because validation accuracy was not improving. CNN architecture: [64,32,16] CL, [128,64] DL .with [0,25%] drop out and batch normalization on.

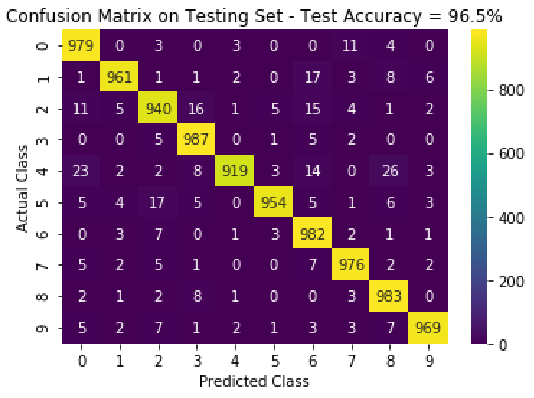
	Training Accuracy	**Validation Accuracy**	**Test Accuracy**
SGD η = 0.01	99.35%	97.65%	93.67%
SGD η = 0.05	99.79%	98.52%	95.59%
SGD η = 0.1	99.83%	98.75%	95.94%
Adam η = 0.001	99.15%	98.23%	94.58%
Adam η = 0.01	99.77%	98.57%	95.87%

Table 11: Performance of different optimizers for CNNs of the same architecture.

***f)* Conclusion.** After testing out the different hyperparameter, we further experimented with CNN architecture and got better performance with an architecture of 3 convolutional layers with 32, 32 ands 64 filters respectively. We used batch normalization and dropout of 25% after each dense layer only . We used ReLU activation and Adam optimizer with learning rate =0.001. The model reaches an accuracy of 96.5% on test data (not previously seen by it). Fig. 17 and Fig. 18 present the confusoion matrix.

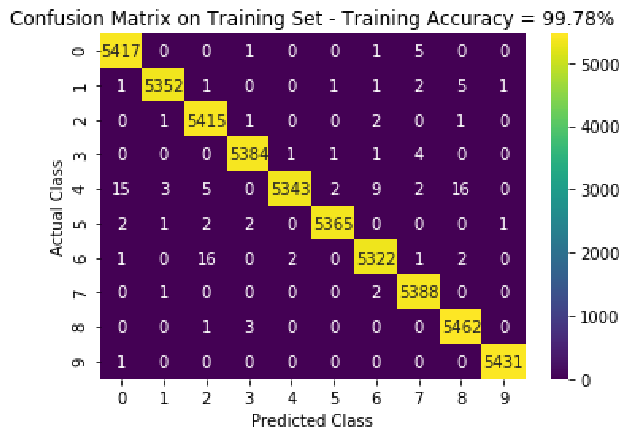
Training Accuracy	**Validation Accuracy**	**Test Accuracy**
99.78%	98.78%	96.50%

Table 12: Final model performance



	0	1	2	3	4	5	6	7	8	9
0	979	0	3	0	3	0	0	11	4	0
1	1	961	1	1	2	0	17	3	8	6
2	11	5	940	16	1	5	15	4	1	2
3	0	0	5	987	0	1	5	2	0	0
4	23	2	2	8	919	3	14	0	26	3
5	5	4	17	5	0	954	5	1	6	3
6	0	3	7	0	1	3	982	2	1	1
7	5	2	5	1	0	0	7	976	2	2
8	2	1	2	8	1	0	0	3	983	0
9	5	2	7	1	2	1	3	3	7	969

Figure 17: Confusion Matrix as a measure for performance of the CNN on the test set. Number of samples in the test set are 10,000.



	0	1	2	3	4	5	6	7	8	9
0	5417	0	0	1	0	0	1	5	0	0
1	1	5352	1	0	0	1	1	2	5	1
2	0	1	5415	1	0	0	2	0	1	0
3	0	0	0	5384	1	1	1	4	0	0
4	15	3	5	0	5343	2	9	2	16	0
5	2	1	2	2	0	5365	0	0	0	1
6	1	0	16	0	2	0	5322	1	2	0
7	0	1	0	0	0	0	2	5388	0	0
8	0	0	1	3	0	0	0	0	5462	0
9	1	0	0	0	0	0	0	0	0	5431

Figure 18: Confusion Matrix as a measure for performance of the CNN on the test set. Number of samples in the test set are 54,000.

IV. Discussion

Although MLPs are capable of Image classification, they are not ideal. Since MLPs take a vector as input, flattening of the image is required which results in losing local spatial information. On the other hand, CNNs take matrices or tensors as input so they learn the relation between neighboring pixels. MLPs are sensitive to translation of the image components, while a CNN scans the whole image each time it applies a filter. Therefore CNNs are now the go-to method for prediction problems with images as input.

Levenberg-Marquardt

We wanted to check how different training algorithms affect the results. The Levenberg-Marquardt is a combination of gradient descent and gaussian newton methods. The Levenberg-Marquardt acts as gradient descent when it is far from the local minimum just ignoring the curvature of the loss function. After that, it acts as a gaussian newton as it gets close to the local minimum. After switching to gaussian newton, it can focus on the curvature of the loss function. With information about the curvature of the loss function, the model can update better parameters. [10]

For this experiment we used simple MLP model that has two hidden layers. The first layer has 100 units, and the second layer has 50 units. We assumed that we could get better results with the Levenberg-Marquardt however, the model with the SGD got a better result. We got an accuracy of 78.6 % by using the Levenberg-Marquardt optimizer and an accuracy of 88.6 % by using the SGD optimizer. We assume that initial weights were not good for the Levenberg-Marquardt. If we try to use updated weights from other training, then the model may get better results with the Levenberg-Marquardt because the training can start better positions to find the minimum point.

RBF network vs MLP

We also tried RBF (Radial Basis Function) Network. As the data set is images, the MLP will have better results than the RBF. Through the hidden layers, the MLP can get redundant features that make the model train well [11]. However, the RBF has only one hidden layer. With this hidden layer, the RBF cannot sufficiently learn features in high dimensional data set. We could get an accuracy of 72.84 %. If we are using a simpler and smaller dataset that has low features (low dimensional), we would better use this RBF network and can get the results faster than the MLP because RBF has only one hidden layer.

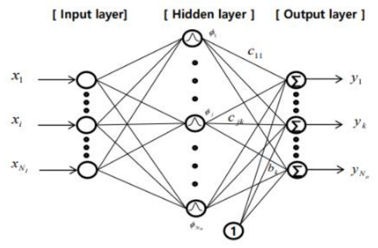


Figure 19. RBF architecture schematic. [9]

V. Conclusion

KMNIST is a complex data set which needs a more complex architectures than the ones we mentioned to substantially increase accuracy. However, we are limited by the available computational resources. For future work, we can try data augmentation which increases the size of the available training set and also train the model on different orientations and shifts of the character.

References

[1] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning*.

[2] J. Principe, N. Euliano and W. Lefebvre, Neural and adaptive systems. New York: Wiley, 2000.

[3] "Understanding of Convolutional Neural Network (CNN) — Deep Learning", Medium, 2021. [Online]. Available: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. [Accessed: 24- Oct- 2021].

[4] "GitHub - rois-codh/kmnist: Repository for Kuzushiji-MNIST, Kuzushiji-49, and Kuzushiji-Kanji", GitHub, 2021. [Online]. Available: https://github.com/rois-codh/kmnist. [Accessed: 22- Oct- 2021].

[5] A. Krizhevsky, I. Sutskever and G. Hinton, "ImageNet classification with deep convolutional neural networks", Communications of the ACM, vol. 60, no. 6, pp. 84-90, 2017. Available: 10.1145/3065386.

[6] A. Agarap, "Deep Learning using Rectified Linear Units (ReLU)", 2019.

[7] D. Liu, "A Practical Guide to ReLU", Medium, 2021. [Online]. Available: https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7. [Accessed: 27- Oct- 2021].

[8] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", 2015.

[9] "지능제어 (7) - Radial Basis Function Network(RBFN) - 방사형 구조의 신경망", 네이버 블로그 | ▒Uri-Sarang's Lab▒, 2021. [Online]. Available: https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=9409290274&logNo=221553102800. [Accessed: 27- Oct- 2021].

[10] "Intro to optimization in deep learning: Momentum, RMSProp and Adam", Paperspace Blog, 2021. [Online]. Available: https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/. [Accessed: 27- Oct- 2021].

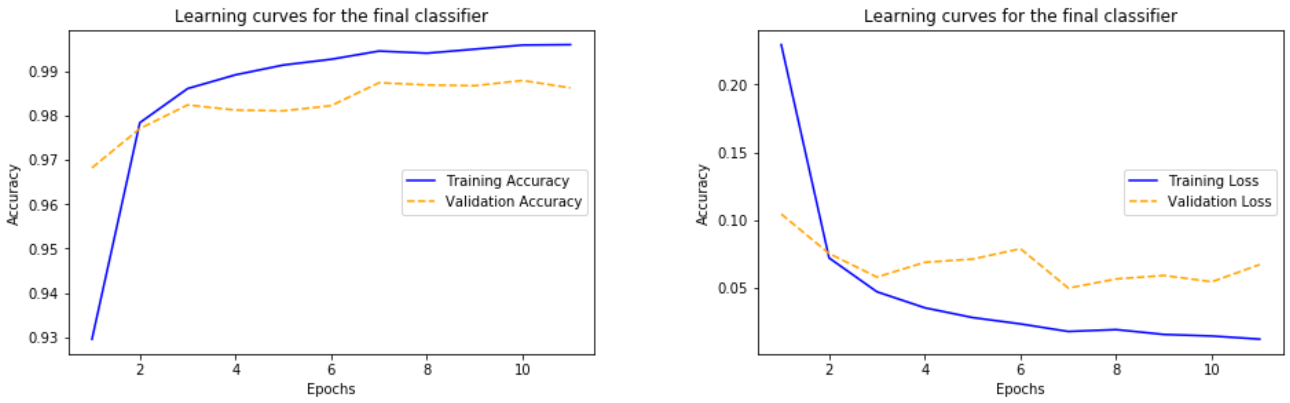
[11] "Radial Basis Function Network", HackerEarth Blog, 2021. [Online]. Available: https://www.hackerearth.com/blog/developers/radial-basis-function-network/. [Accessed: 27- Oct- 2021].

SUPPLEMENTARY MATERIAL

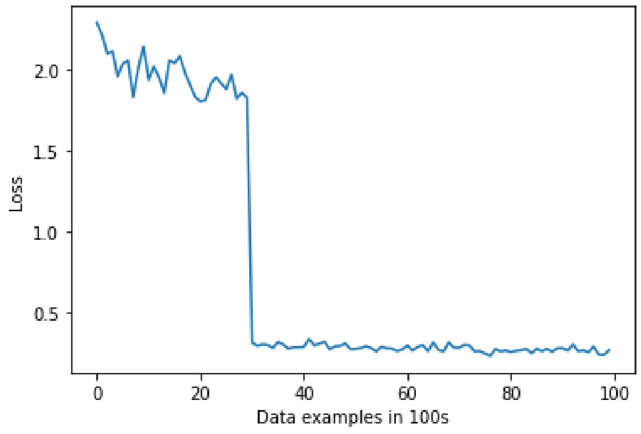
ID	Convolutional Layers	Dropout	Batch Norm	Activation	Optimizer	Train Acc	Val Acc	Test Acc
CNN1	[128]	0	FALSE	ReLU	sgd	99.36%	96.33%	91.59%
CNN2	[128,64]	0	FALSE	ReLU	sgd	99.43%	97.62%	93.66%
CNN3	[128,64,32]	0	FALSE	ReLU	sgd	99.22%	97.78%	94.14%
CNN4	[64,32,16]	0	FALSE	ReLU	sgd	98.86%	97.33%	92.17%
CNN5	[32,16,8]	0	FALSE	ReLU	sgd	98.26%	97.13%	91.79%
CNN6	[64,32,16]	0	TRUE	ReLU	sgd	99.70%	97.72%	93.14%
CNN7	[32,16,8]	0	TRUE	ReLU	sgd	98.57%	96.90%	92.34%
CNN8	[64,32,16]	[0.25,0.25]	TRUE	ReLU	sgd	98.37%	98.08%	93.40%
CNN9	[64,32,16]	[0,0.25]	TRUE	ReLU	sgd	99.35%	97.65%	93.67%
CNN10	[64,32,16]	[0,0.5]	TRUE	ReLU	sgd	99.28%	97.92%	93.47%
CNN11	[64,32,16]	[0.5,0.5]	TRUE	ReLU	sgd	95.88%	95.68%	86.64%
CNN12	[64,32,16]	[0,0.25]	TRUE	ReLU	Sgd = 0.05	99.79%	98.52%	95.59%
CNN13	[64,32,16]	[0,0.25]	TRUE	ReLU	Sgd = 0.1	99.83%	98.75%	95.94%
CNN14	[64,32,16]	[0,0.25]	TRUE	ReLU	Adam = 0.001	99.15%	98.23%	94.58%
CNN15	[32,64,64]	[0,0.25]	TRUE	ReLU	Adam = 0.001	99.76%	98.80%	96.12%
CNN16	[64,32,16]	[0,0.25]	TRUE	ReLU	Adam = 0.01	99.77%	98.57%	95.87%
CNN17	[64,32,16]	[0,0.25]	TRUE	ReLU	Adam = 0.05	99.55%	98.62%	95.84%
CNN18	[64,32,16]	[0,0.25]	TRUE	ReLU	Adam = 0.1	99.50%	98.43%	94.64%
CNN19	[32,64,64]	[0,0.25]	TRUE	LReLU	Adam = 0.001	98.38%	97.88%	93.42%
CNN20	[32,64,64]	[0,0.25]	TRUE	LReLU	SGD = 0.1	99.82%	98.73%	95.77%
CNN21	[32,32,64]	[0,0.25]	TRUE	ReLU	Adam = 0.001	99.78%	98.78%	96.50%
CNN22	[16,32,64]	[0,0.25]	TRUE	ReLU	Adam = 0.001	99.78%	98.88%	96.16%
CNN23	[32,32,64]	[0,0.25]	TRUE	ReLU	Adam = 0.01	98.82%	97.88%	94.56%
CNN24	[32,32,64]	[0,0.25]	TRUE	ReLU	SGD 0.1	99.88%	98.97%	96.16%
CNN25	[32,32,64]	[0,0.25]	TRUE	ReLU	SGD 0.05	99.97%	98.98%	96.26%
CNN26	[16,32,32,64]	[0,0.25]	TRUE	ReLU	SGD 0.1	99.73%	98.93%	96.01%
CNN27	[16,32,32,64]	[0,0.25]	TRUE	ReLU	ADAM 0.005	99.60%	98.85%	95.91%
CNN28	[32,32,32,64]	[0,0.25]	TRUE	ReLU	ADAM 0.01	98.23%	98.86%	94.39%

Supp. Table 1: All CNN models trained and tested

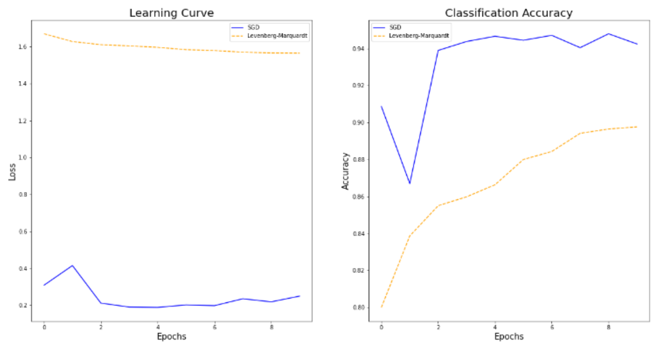
Supplementary Figures



Supp Fig 1Final CNN model performance



Supp Fig 2 RBF performance



Supp Fig 3: Performance of MLP using levenberg-marquardt optimizer