An Evaluation of Current Network Based Models on Kuzushiji-MNIST Dataset

Zachary Wilkerson, *Student Member, IEEE*

***Abstract*—Supervised machine learning has become the go-to method for performing image classification. Using class labels associated with the images, such supervised models are able to learn the relationship between image and class quite efficiently. There are many different algorithms available to perform such classification, such as random forests, Bayesian networks, support vector machines, neural networks, among others. In this paper, we evaluate the performance of two neural network models, a multilayer perceptron and a convolutional neural network, on the task of image classification using the Kuzushiji-MNIST dataset. We find that the convolutional neural network achieves higher performance when compared to the multilayer perceptron.**

***Index Terms*—deep learning, supervised learning, image classification, convolutional neural networks.**

I. Introduction

ONE application of machine learning is image classification, where the goal is to link images to their associated class or category. To solve such image classification problems, supervised machine learning algorithms such as random forests [1], Bayesian networks [2], and support vector machines [3] have been used. With the more recent interest in deep learning, algorithms like multilayer perceptrons (MLPs) and convolutional neural networks (CNNs) have become the dominant algorithms for use in the image classification field due to their superior performance compared to traditional methods [4].

The primary goal of this paper is to compare the performance of a MLP and a CNN on the image classification of handwritten Japanese characters. Based on past research comparing MLP performance to CNN performance on a similar image classification task, we expect the CNN to provide better classification performance compared to the MLP [5].

II. Description

A. Dataset

A popular benchmark dataset used in image classification is the MNIST dataset [6], which consists of 70,000 images in 28x28 grayscale format which correspond to 10 categories of handwritten digits (0-9). Using benchmark datasets such as MNIST allow machine learning researchers to collectively examine the challenges associated with that specific task – classifying digits. The task itself doesn't matter all too much, and so some researchers believe that additional datasets should be introduced which provide tasks that are more socially and culturally relevant. In turn, the Kuzushiji-MNIST dataset [7], a relative to the MNIST dataset, consisting of 70,000 images in 28x28 grayscale format corresponding to 10 categories of Japanese handwritten characters, was introduced to expose the world to classical Japanese literature. In this paper, we make use of the Kuzushiji-MNIST dataset for all evaluations. We maintain the designated train/test split provided by the authors – 60,000 images for training and 10,000 images for testing. A few example images from the Kuzushiji-MNIST dataset are shown in Figure 1.

Fig. 1: Examples from Kuzushiji-MNIST dataset.

B. Implementation Details

To implement both our MLP and CNN models in Python, we make use of the Keras API [8]. We design our models based on previous models that have proven to perform well on the original MNIST dataset [9] [10]. We anticipate that the structure of such models will perform similarly on the Kuzushiji-MNIST dataset as the applications are very similar – categorization of 10 characters. The hyperparameters of each model will be tuned to adjust for the new application.

Our MLP model starts with a dense layer with a ReLU activation function, followed by a dropout layer, then a second dense layer with a ReLU activation function, followed by a second dropout layer, then a third dense layer of 10 units with a softmax activation function to output class probabilities.

Our CNN model starts with two convolutional layers, both with ReLU activation functions, followed by a max pooling layer, then a dropout layer, then a flattening layer, then a dense layer with a ReLU activation function, then a second dropout layer, then a second dense layer of 10 units with a softmax activation function to output class probabilities.

Note that throughout both networks, we made the choice to use ReLUs with the intent to reduce the training time of our models and included dropout layers to reduce overfitting [11]. To maintain some similarity between networks, all models

use the categorical cross-entropy loss function, use the Adam optimizer, and record accuracy per epoch.

C. Hyperparameter Tuning

The hyperparameters for both our models were selected with the help of the Keras Tuner [12]. Specifically, the Hyperband algorithm [13] was used. This algorithm initializes a large number of models and slowly eliminates low performing models using small epoch training runs. This algorithm was chosen to train with, as opposed to a traditional random search, because of its adaptive resource allocation and integrated early-stopping to decrease time to convergence. We set our Hyperband algorithm to maximize validation accuracy, using a maximum of 50 epochs with a maximum of 100 trials (number of hyperparameter combinations) and 3 executions (number of models to be built) per trial, along with early-stopping enabled with a minimum change equal to 0.001 to indicate improvement and a patience of 10 epochs where the best weights are restored. These values were selected to do as extensive a search as possible with the limited time and computation resources available. To maintain reproducibility when using our Hyperband tuner, we set the seed to 55. We perform our search for the best hyperparameters of both models using our Hyperband tuner, taking 10% of our training data set to be used as a validation set.

1) MLP Model: We begin with the first dense layer of our network. We use a ReLU activation function and learn the optimal number of units for this layer. We then learn the dropout rate of the first dropout layer before reaching the second dense layer which also has a ReLU activation function and for which we also learn the optimal number of units. Finally, we learn the optimal dropout rate of the final dropout layer before we reach our final dense layer with a softmax activation function. We also learn the optional learning rate of our Adam optimizer using a log sampling. All possible hyperparameter combinations that we tested are shown in Table I.

Layer/Optimizer	Hyperparameter Values
First Dense Layer	Units: [32:32:512]
First Dropout Layer	Rate: [0:0.05:0.5]
Second Dense Layer	Units: [32:32:512]
Second Dropout Layer	Rate: [0:0.05:0.5]
Adam Optimizer	Learning Rate: $[10^{-4} : 10^{-1}]$ (using log sampling)

TABLE I: List of hyperparameter options available during tuning of MLP model.

2) CNN Model: We begin with the first convolutional layer of our network. We learn the optimal number of filters for this layer while using a filter size of (3,3) with (1,1) strides and no padding and a ReLU activation function. The second convolutional layer of our network is the same. Our max pooling layer pools with a pool size of (2,2). We learn the dropout rate of the following dropout layer. After flattening, we learn the optimal number of units for the following dense layer. Finally, we learn the optimal dropout rate of the final dropout layer before we reach our final dense layer with a softmax activation function. We also learn the optimal learning rate of our Adam optimizer using a log sampling. All possible hyperparameter combinations that we tested are shown in Table II.

Layer/Optimizer	Hyperparameter Values
First Convolutional Layer	Filters: [16:16:256]
Second Convolutional Layer	Filters: [16:16:256]
First Dropout Layer	Rate: [0:0.1:0.5]
First Dense Layer	Units: [32:32:512]
Second Dropout Layer	Rate: [0:0.1:0.5]
Adam Optimizer	Learning Rate: $[10^{-4} : 10^{-1}]$ (using log sampling)

TABLE II: List of hyperparameter options available during tuning of CNN model.

III. Evaluation

A. MLP Model

After 100 tuning trials that took a little over 2 hours to complete, we obtained our best performing MLP hyperparameters based on the highest validation accuracy. We present these hyperparameters in Table III.

Layer/Optimizer	Hyperparameter Values
First Dense Layer	Units: 448
First Dropout Layer	Rate: 0.45
Second Dense Layer	Units: 448
Second Dropout Layer	Rate: 0.05
Adam Optimizer	Learning Rate: 0.0004

TABLE III: Best performing hyperparameters obtained during tuning of MLP model.

We fit our best performing MLP model using the training data, where 10% was used as a validation set, for 150 epochs with a batch size of 128. We also make use of early-stopping with a minimum change equal to 0.001 to indicate improvement and a patience of 30 epochs where the best weights are restored. We find that training for 21 epochs is the best option to reduce overfitting for our model. This is emphasized by the results of Figure 2, where we see our training accuracy continue to rise while our validation accuracy maintains around the same value. We also note that our training loss flattens out while our validation loss continues to increase, indicating training has gone on for too long. We evaluate our trained model on the test data and obtain 92.5% accuracy. The confusion matrix for this model is shown in Figure 3.

B. CNN Model

After 100 tuning trials that took a little over 40 hours to complete, we obtained our best performing CNN hyperpa-

rameters based on the highest validation accuracy. We present these hyperparameters in Table IV.

Layer/Optimizer	Hyperparameter Values
First Convolutional Layer	Filters: 32
Second Convolutional Layer	Filters: 112
First Dropout Layer	Rate: 0.45
First Dense Layer	Units: 288
Second Dropout Layer	Rate: 0.40
Adam Optimizer	Learning Rate: 0.0013

TABLE IV: Best performing hyperparameters obtained during tuning of CNN model.

We fit our best performing CNN model using the training data, where 10% was used as a validation set, for 150 epochs with a batch size of 128. We also make use of early-stopping with a minimum change equal to 0.001 to indicate improvement and a patience of 30 epochs where the best weights are restored. We find that training for 5 epochs is the best option to reduce overfitting for our model. This is emphasized by the results of Figure 4, where we see our training accuracy continue to rise while our validation accuracy maintains around the same value. We also note that our training loss flattens out while our validation loss continues to increase, indicating training has gone on for too long. We evaluate our trained model on the test data and obtain 95.3% accuracy. The confusion matrix for this model is shown in Figure 5.

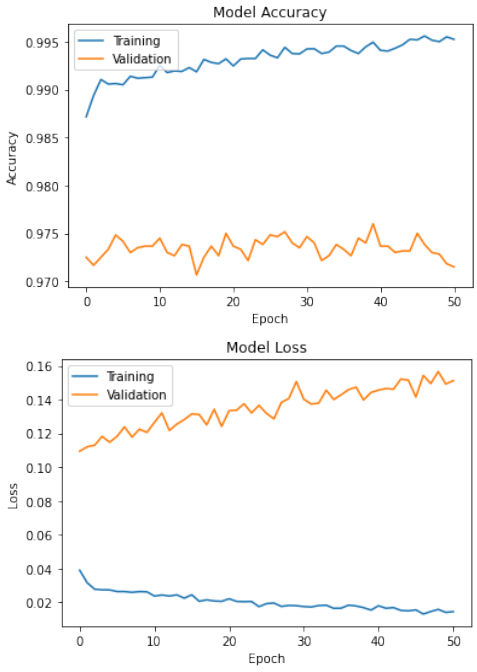

Fig. 2: Plots of accuracy (top) and loss (bottom) versus epoch for chosen MLP model.

C. General Remarks

We make note of the number of trainable parameters of both networks. Our MLP model has 557,322 trainable parameters,

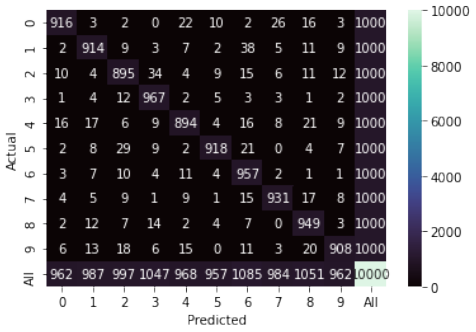

Fig. 3: Confusion matrix for chosen MLP model.

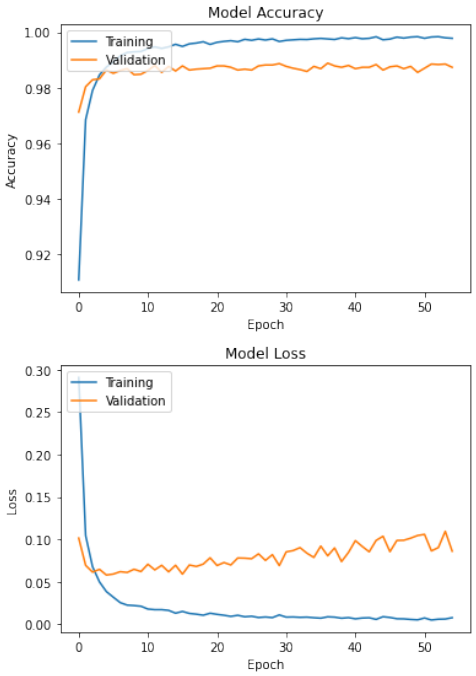

Fig. 4: Plots of accuracy (top) and loss (bottom) versus epoch for chosen CNN model.

4,680,730 trainable parameters. This difference is expected, as CNNs simply require we learn the values of our small dimension filters whereas MLPs require we learn the weights between all nodes in our densely connected network.

It is also important to note the drastic difference in tuning time required for both networks. Our MLP model took approximately 2 hours to complete tuning, whereas our CNN model took approximately 40 hours to complete tuning. Therefore, the CNN model took 20 times longer to tune. We attribute this drastic difference to the fact that the CNN model, while much lower in parameters, requires significantly more multiplication operations due to the nature of the convolution layers. A 2.6 GHz 6-Core Intel Core i7 processor was used for the entirety of this work, which may have also impacted the tuning time.

The CNN model clearly outperforms the MLP model, given the higher test accuracy with superior performance in classifying all classes (see Figures 3 and 5). This result is not surprising, as the convolution process associated with a CNN model is able to use adjacent pixel information to extract

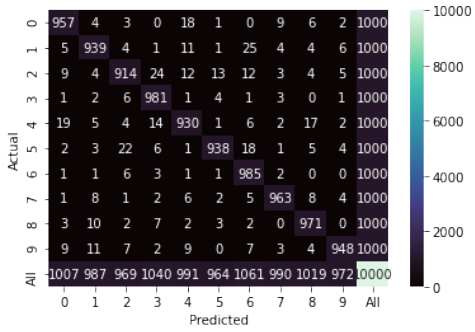

Fig. 5: Confusion matrix for chosen CNN model.

features to pass through the model while a MLP has no such ability.

D. Improvements

The models we found can be considered the best models that exist for their given architecture – the tuning process gave us the ideal hyperparameters needed to give the best performing model. However, there may be other architectures that provide better classification results, such as adding/removing dense layers in the MLP and convolutional layers in the CNN. In future work, it would be beneficial to investigate these changes and determine their effects on performance outcomes.

We may also want to consider introducing additional data into the training process of our models, perhaps through data augmentation. Previous work has already shown the effectiveness of image augmentation for image classification [14].

IV. Conclusion

In this work, we tuned, trained, and evaluated two neural network models on the task of image classification using the Kuzushiji-MNIST dataset via Keras in Python. We showed that the convolutional neural network gave superior performance compared to the multilayer perceptron. We also discussed the differences between both models in terms of tuning time and number of trainable parameters, as well as mentioned some improvements that could be made in an attempt to increase both models performance.

References

[1] A. Bosch, A. Zisserman, and X. Munoz, "Image classification using random forests and ferns," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8.

[2] K. Jayech and M. A. Mahjoub, "Clustering and bayesian network for image of faces classification," 2012.

[3] M. Chandra and S. Bedi, "Survey on svm and their application in image classification," *International Journal of Information Technology*, vol. 13, 01 2018.

[4] "Advances in computer vision," *Advances in Intelligent Systems and Computing*, 2020. [Online]. Available: http://dx.doi.org/10.1007/978-3-030-17795-9

[5] S. Ben Driss, M. Soua, R. Kachouri, and M. Akil, "A comparison study between MLP and Convolutional Neural Network models for character recognition," in *SPIE Conference on Real-Time Image and Video Processing*, ser. Real-Time Image and Video Processing 2017, vol. 10223, Anaheim, CA, United States, Apr. 2017. [Online]. Available: https://hal-upec-upem.archives-ouvertes.fr/hal-01525504

[6] 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.

[7] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *CoRR*, vol. abs/1812.01718, 2018. [Online]. Available: http://arxiv.org/abs/1812.01718

[8] Keras, "Keras documentation: About keras." [Online]. Available: https://keras.io/about/

[9] R. Atienza, *Advanced Deep Learning with TensorFlow 2 and Keras: Apply DL, GANs, VAEs, deep RL, unsupervised learning, object detection and segmentation, and more, 2nd Edition*. Packt Publishing, 2020. [Online]. Available: https://books.google.com/books?id=68rTDwAAQBAJ

[10] Keras, "Keras documentation: Simple mnist convnet." [Online]. Available: https://keras.io/examples/vision/mnist_convnet/

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., vol. 25. Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

[12] Keras, "Keras documentation: Kerastuner api." [Online]. Available: https://keras.io/api/keras_tuner/

[13] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," 2018.

[14] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," 2017.