K-MNIST dataset classification using Multilayer Perceptron and Convolution Neural Network

Xiangsheng Chai (CNN part)
University of Florida
Gainesville, Florida, USA
xchai@ufl.edu

Shania Shakri (MLP part)
University of Florida
Gainesville, Florida, USA
shaniashakri@ufl.edu

***Abstract*—Machine Learning can be leveraged to answer some of the most difficult challenges required of a computer. Within Machine Learning lies Neural Networks, a state of the art method for visual signals used in handwriting recognition. In this paper, we implement and compare past and current machine learning models for handwritten symbol recognition of 70,000 Japanese letters on 10 class labels. Our findings include understanding the capacity and scalability of simple and complex machine learning methods. The results suggested a potential advantage for CNNs over MLPs for handling the KMNIST dataset. We have provided easy-to-understand evidence, like performance graphs and data, underscoring the rationale behind our choices.**

I. Introduction

Neural networks are a subset of machine learning that mimic the way biological neurons send signals to one another to leverage it for a high performance interpretation of a data set. They are comprised of a node layer containing the input, one or more hidden layers, and an output layer. Each node connects to one another and has an associated weight and threshold. If the output of a node is above the associated threshold value then the node is activated and it will send data to the next layer of the network. These networks rely on training data to learn and improve their accuracy over time as well. Once accurate to an acceptable degree, the networks can be used to reliably classify and cluster data at a high throughput. In this study, we want to see how we can leverage a neural network to create a model specifically intended to recognize mathematical symbols from an image [1].

Image classification refers to the task of identifying an image and categorizing it in one of the predefined classes. Image classification can leverage localization, which is when a bounding box is drawn around an object in an image in order to categorize it under a certain class. It can also include object detection which can categorizing multiple different objects in the image and localize them individually. There has been extensive research done in this field, referred to as Computer Vision, that can help guide the direction of future work to be done under the domain.

In this project we try to create models for identification of symbols in a multiclass image classification dataset. We conduct a series of experiments to find the best possible parameters for the 2 models to achieve acceptable results. Our CNN model achieves an accuracy of 98.56%, while the MLP model follows closely with 92.99%. These results are obtained from carefully chosen hyperparameters and the use of techniques like Batch Normalization and Dropout. Through our experiments We also find that CNNs outperform MLPs for this specific task. This is discussed in the Experiments section of this report with reasons behind our choice of hyperparameters.

II. Methods

A. Multi-layer Perceptrons

Introduced in the late 1970s, Multi-Layer Perceptrons (MLPs) are a class of artificial neural network structured around the foundational concept of the perception algorithm. They represent the simplest form of feedforward neural networks, a category that propagates data linearly from the input to the output layer [2].

1) Architecture: MLPs consist of multiple layers of nodes in a directed graph (Figure 1), with each layer fully connected to the next one. The most elementary form encompasses three layers: a) Input Layer: initiates the network, passing on input data without computation, b) Hidden Layers: These crucial layers compute inputs via weighted sums and non-linear activations, learning data complexities and c) Output Layer: Concludes the process, utilizing specific activations to perform tasks like classification or regression.

2) Learning: MLPs learn through backpropagation, optimizing weight adjustments in reverse from output to input, minimizing prediction errors using gradient descent. MLPs while versatile, falter with high-dimensional data, like images, and can overfit due to their dense, fully-connected nature. So, we compare the results of an MLP model to a CNN model for the mentioned dataset in this project.

B. Convolutional Neural Networks

Convolutions neural network (CNN) was first introduced by Yann LeCun [3] in 1980s. Compare to MLPs, CNNs use convolutional filters to capture the spacial information of sequence. Therefore, they are frequently used for processing and analyzing visual data, such as images and videos. They are particularly well-suited for image classification, hence CNNs could be a better tool to classify the KMINST dataset.

The traditional CNNs usually contain the convolutional layers, pooling layers and fully connected layers. Convolutional layers apply a set of learnable filters to the input image,

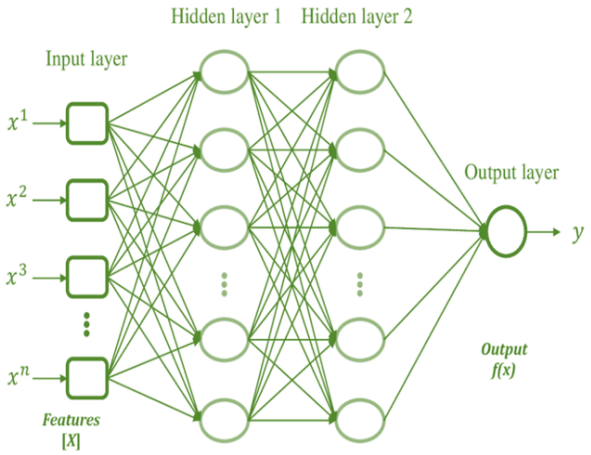

Fig. 1. MLP Model

enabling the network to recognize patterns and features at different scales. Pooling layers are used to downsample the feature maps generated by convolutional layers. Common pooling operations include max-pooling and average-pooling, which help reduce the spatial dimensions while retaining important information.

C. Inception CNNs

Since CNNs were created, many different architectures of CNNs have been proposed, such as AlexNet, VGGNet, ResNet, and so on. Inception CNNs, also known as GoogLeNet [4], is one of them. The key feature of the Inception architecture is its use of so-called "Inception modules." These modules are building blocks within the neural network that incorporate multiple parallel convolutional filters of different sizes. These parallel filters enable the network to capture features at various scales, from fine details to larger patterns, all within a single layer of the network.

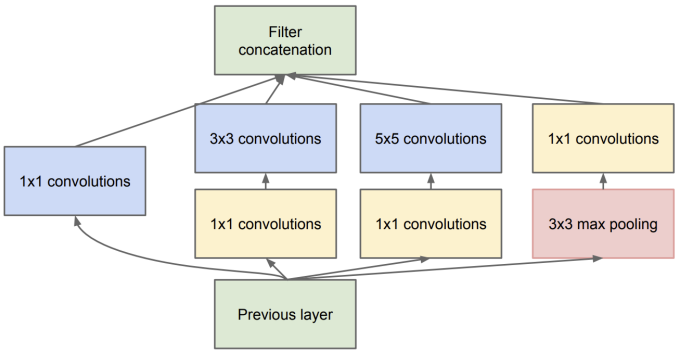

Fig. 2. Inception Module

Inception modules make efficient use of the parameters.The use of 1x1 convolutions within Inception modules reduces the dimensionality of the feature maps, which helps in reducing the computational load and the number of parameters. It is also helpful for regularization.

D. Ensemble learning

Ensembling is a popular strategy in machine learning and data science area. The idea behind ensembling is to take advantage of the diversity and strengths of different models to create a more robust and accurate predictive model. There are several common ensemble methods such as bagging and boosting. For instance, random forest using bagging to combines multiple decision trees to make better models. It efficiently solve the overfitting issue of decision tree. In this project, bagging with multiple CNNs can also provide better performance.

III. Implementation

A. Dataset

In this project, Kuzushiji-MNIST (KMNIST) dataset [5] was used to do the classification. Kuzushiji was widely used in Janpanese historical texts, literature, and official documents. However, as contemporary Japanese script evolved, Kuzushiji gradually became less common, making it difficult for modern readers to understand historical documents. Kuzushiji MNIST was created to address this issue. The dataset consists of characters written in Kuzushiji script that were extracted from historical documents.

The dataset includes a collection of characters in the Kuzushiji script, such as hiragana, katakana, and kanji characters. In KMNIST, there are total 10 labels. The labels 0 through 9 correspond to different classes of characters in the Kuzushiji script. Each label represents a specific character category or group.

KMNIST contains a total of 70,000 samples, 60,000 for training and 10,000 for test. Each sample is represented as a grayscale image. These images are typically 28x28 pixels in size, making them suitable for training and testing machine learning models.

B. Data Splitting

In this project, we have used the KMNIST dataset, which had 60,000 examples, and divided it to make the model's learning process better. We made two groups from these examples: 50,000 of them were used for training (teaching the model), and 10,000 were used for validation (testing the model). This step was very important for two main reasons: 1) It helped check how well the model was learning while it was being trained. This way, we could see if the model was learning correctly or just memorizing the training data (which is something we don't want, known as overfitting). 2) The validation group (the 10,000 examples) was very important because it helped us adjust the model's settings. This helped make sure the model can understand and interpret new, unseen data well.

C. Preprocessing

- Normalization: Usually, normalization is an important step before training model in machine learning. For image dataset, there are two ways to make the normalization, rescaling the pixel values (minmax scaler) and subtracting

mean and divide by standard deviation (standard scaler). In this project, standard scaling was applied.

- Augmentation: Data augmentation can be applied to increase the diversity of training data. These techniques like random rotations, translations, and flips can help improve model generalization. For CNN, we utilized strong augmentation techniques include random affine, random rotation and so on. We applied random augment [6] in the data for CNN. The result is shown in Figure 3. For MLP, because it flattens the image into sequence and loses the spacial information, data augmentation doesn't work well as it in CNN. Hence, we only apply random affine for MLP.

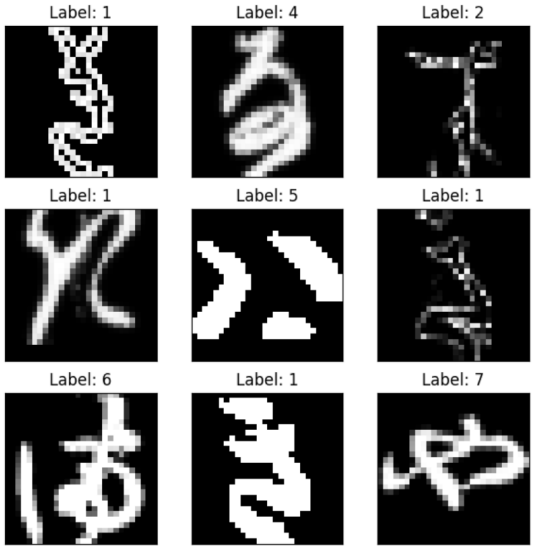

Fig. 3. Data Augmentation

D. Model Selection

1) MLP: One of the architecture tested in this project was a Multilayer Perceptron. This model was chosen due to its proficiency in learning from and making predictions based on the KMNIST dataset. The MLP was designed with three dense layers, accommodating the high dimensionality of the input data. The model comprised an input layer that flattens the 28x28 pixel images into a contiguous array of 784 elements, two hidden layers with 512 and 256 neurons, respectively, each followed by batch normalization, a ReLU activation function, and a dropout layer for regularization. An output layer designed with ten neurons corresponding to the ten classes of the KMNIST dataset, utilizing the softmax function to output a probability distribution over the ten classes. Each layer utilized the He initialization scheme, conducive to maintaining a consistent scale among neuron activations and accelerating the convergence of the stochastic gradient descent optimization.

2) CNN: In convolutional neural networks, the hidden layers consist of several concolutional layers followed by two fully connected layers. In order to extract the feature map and reduce the number of parameters, maxpool layers were introduced. Compared to average pooling, max pooling can extract more useful information in local map in image classification. In the same time, dropout layers and batch normalization were applied for each convolutional layer and fully connected layer in order to regularize the model.

Table I shows the structure of the CNN model for this project. It contains 3 convolutional layers, 2 maxpool layers and 2 fully connected layer. For first convolution layers we choose 5x5 kernel to capture more global features in the input and data, and 3x3 kernel was selected to effectively capture the local details in the feature map for second and third convolution layer. And same padding were used instead of no padding because we'd like to maintain the spatial size of feature maps after convolution. The third convolutional layer doesn't contain a maxpool layer. ReLU activation function was used for convolutional layers and fully connected layers, and Softmax for output layer. This is because we'd like to transfer more features to fully connected layer to improve the performance. Experiment proved that these strategies effectively improve the classification in validation dataset.

TABLE I
STRUCTURE OF CLASSIC CNN

Layer	Output Shape	Number of Parameters
Conv2d-1	$[-1, 64, 28, 28]$	1,664
BatchNorm2d-2	$[-1, 64, 28, 28]$	128
Dropout2d-3	$[-1, 64, 28, 28]$	0
MaxPool2d-4	$[-1, 64, 14, 14]$	0
Conv2d-5	$[-1, 128, 14, 14]$	73,856
BatchNorm2d-6	$[-1, 128, 14, 14]$	256
Dropout2d-7	$[-1, 128, 14, 14]$	0
MaxPool2d-8	$[-1, 128, 3, 3]$	0
Conv2d-9	$[-1, 200, 3, 3]$	230,600
BatchNorm2d-10	$[-1, 200, 3, 3]$	400
Dropout2d-11	$[-1, 200, 3, 3]$	0
Linear-12	$[-1, 256]$	461,056
BatchNorm1d-13	$[-1, 256]$	512
Linear-14	$[-1, 10]$	2,570

Table II shows the structure of Inception CNN. The difference of Inception CNN and Classic CNN is each convolutional layer was followed by an inception module except the third one. Inception modules is helpful to make the neural network deeper and enhance feature extraction at the same time. It is also a good strategy to make the model more robust to avoid overfitting.

E. Regularization

Besides data augmentation, we also applied several strategies in building networks and training to regularize the parameters.

- *Dropout*: By randomly setting some neurons to zero, dropout introduces a form of noise into the network. This noise forces the network to be less deterministic, as it can't predict which neurons will be active in the next iteration. The network must learn to be robust to this noise, which leads to better generalization. In this project, we test different probability of an element to be zeroed.

TABLE II
STRUCTURE OF INCEPTION CNN

Layer	Output Shape	Number of Parameters
Conv2d-1	$[-1, 64, 28, 28]$	1,664
BatchNorm2d-2	$[-1, 64, 28, 28]$	128
Dropout2d-3	$[-1, 64, 28, 28]$	0
MaxPool2d-4	$[-1, 64, 14, 14]$	0
Inception-5	$[-1, 88, 14, 14]$	22992
BatchNorm2d-6	$[-1, 88, 28, 28]$	176
Dropout2d-7	$[-1, 88, 28, 28]$	0
Conv2d-8	$[-1, 128, 14, 14]$	101,504
BatchNorm2d-9	$[-1, 128, 14, 14]$	256
Dropout2d-10	$[-1, 128, 14, 14]$	0
MaxPool2d-11	$[-1, 128, 7, 7]$	0
Inception-12	$[-1, 88, 7, 7]$	27600
BatchNorm2d-13	$[-1, 88, 7, 7]$	176
Dropout2d-14	$[-1, 88, 7, 7]$	0
Conv2d-15	$[-1, 200, 7, 7]$	158,600
BatchNorm2d-16	$[-1, 200, 7, 7]$	400
Dropout2d-17	$[-1, 200, 7, 7]$	0
Linear-18	$[-1, 256]$	2,509,056
BatchNorm1d-19	$[-1, 256]$	512
Linear-20	$[-1, 10]$	2,570

- *Batch Normalization*: Batch Normalization normalizes the activations of each layer by subtracting the mean and dividing by the standard deviation of the activations within a mini-batch. This normalization step helps mitigate the problem of internal covariate shift, which is the change in the distribution of activations as the network learns.
- *Early Stopping*: By stopping the training process before the model becomes overly complex and starts to fit the noise in the training data, early stopping can avoid overfitting. Usually, we use the loss in validation set as our early stop criterion.

F. Training

1) MLP: The selection of the hyperparameters for this model was done in such a way that we attain maximum accuracy since beyond a certain dropout rate the model might become too regularized, potentially leading to underfitting and the validation accuracy might start decreasing. While experimenting with different dropout values, values above 0.2 (0.4, 0.6, 0.8) seemed to hinder the performance of the model and caused much fluctuations in validation accuracy. For MLP, applying Batch Normalisation after each fully connected layer helped in maintaining the stability of activations during training, which can lead to faster convergence and better generalization. As for the early stopping a patience of 5 epochs for early stopping was a suitable choice as it strikes a balance between preventing overfitting and allowing for fluctuations in validation loss during training. We tried values more than 5 (like 8 or 10) and it made the model more resilient to minor fluctuations in validation loss and presented a missed opportunity for early convergence. While a lower value of 2 or 3 resulted in premature stopping, preventing the model from fully converging. For the number of layers, we chose 2 hidden layers instead of a single one to enhance the model's capacity to recognize intricate patterns. Training deeper models can be more computationally efficient than relying on a single, densely packed layer. From experimentation we saw an additional layer allows the model to achieve similar capacity with exponentially fewer neurons, striking a balance between computational efficiency and predictive performance.

TABLE III
STRUCTURE OF MLP MODELS

Layer	Output Shape	Number of Parameters
Flatten	784	-
fc1	512	Linear, BatchNorm1d, ReLU, Dropout(0.2)
fc2	256	Linear, BatchNorm1d, ReLU, Dropout(0.2)
fc3	10	Linear

2) CNN: In CNN, the main hyperparameters we need to decide are number of epochs, batch size and learning rate. Number of epochs depends on batch size and learning rate, and we also use early stop strategy. Hence, it is not necessary to find an accurate value of number of epochs. We set it 80 as default. For batch size, I chose three different sizes to evaluate the performance 32, 64, and 128. Batch size 32 needs more epochs to be converged, and there is no significant difference between 64 and 128. As a result, to get better converge speed and relax the memory, 100 was chosen as the training batch size. The optimizer we used is Adam with default parameters. We tried to change the momentum, but it came to no difference. Learning rate is the most influential parts in the training. Although we used Adam as our optimizer, too large learning rate also affect the convergence to the global minim. After hyperparameters tuning, 0.0005 was selected as the final step size. Eventually, we used total 60,000 training samples to train the model in order to get the best performance. We also make a comparison between Classic CNN and Inception CNN, and used emsembling to get the best performance. Details will be shown in the Experiment section.

IV. EXPERIMENT

A. MLP

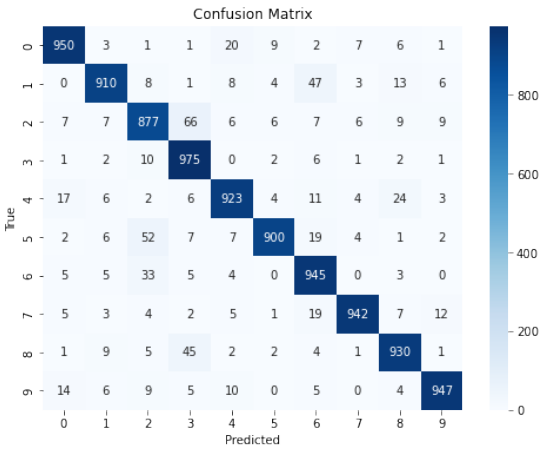

Fig. 4. Confusion Matrix: MLP

The results we achieved from the KMNIST dataset classification using a Multi-Layer Perceptron (MLP) model show acceptable generalization capabilities, with a test accuracy of 92.99% with accurate predictions highlighted by the predominant diagonal values in the confusion matrix (Figure 4). However, we see that the model struggles with class '2' notably, due to similarities with class '4'. As the training progresses, the model keeps learning, which is a good thing. However, after around 20 epochs, the model's performance on new, unseen data doesn't improve much anymore.

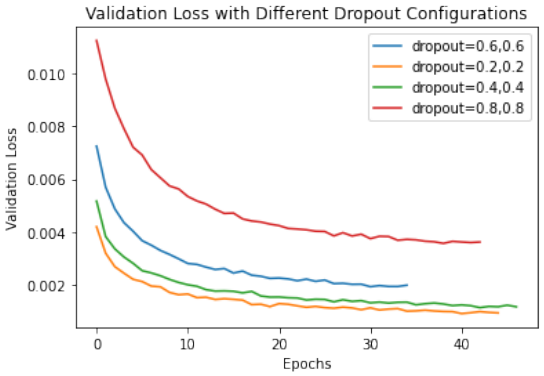

Fig. 5. Effects of different Dropouts in MLP

The figure (Figure 5) shows why we chose 0.2 to be our dropout for the MLP model. Values higher than 0.2 caused high learning initially but led to slower convergence whereas 0.2 gave a more stable and consistent convergence.

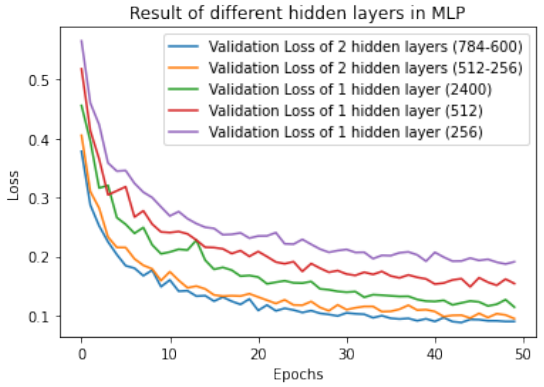

Fig. 6. Effects of different different Hidden layers and units in MLP

To verify our choice of 2 layer architecture and number of hidden units for out model, we ran a couple iterations with different hidden layers and hidden units respectively as shown in Figure 6. The figure goes on to show that 2 layer architecture outperforms a single layer network for a dataset as large as the KMNIST. However, it is also noteworthy to mention that a dece t convergence can be attained from using single layer with many more units (eg., single layer 2400 units) but a 2 layer structure still stands out as IT might be able to break down features more intricately in the first layer and then combine those in the subsequent layer for more abstract representations.

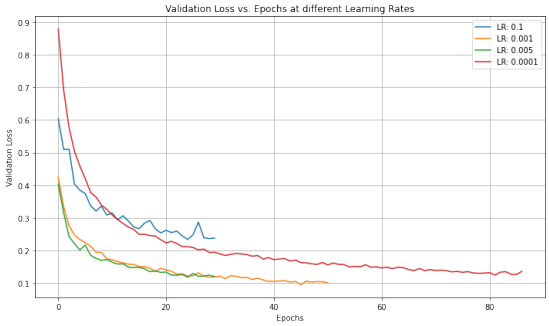

Fig. 7. Effects of different different learning rates in MLP

To justify our choice of learning rate for MLP model of the KMNIST dataset we ran experiments for different learning rate values (0.1, 0.001, 0.005, 0.0001) as seen in Figure 7. A learning rate of 0.1 exhibits high variance and risks oscillating around the minima or even overshooting it, whereas a rate of 0.0001 progresses sluggishly, possibly getting stuck in the lower local minima and hence taking longer time to converge. The 0.001 rate offers a steady descent in the loss landscape without significant fluctuations, ensuring a smoother journey towards the desired minima. While learning rate 0.005 performs as good as 0.001, it attains a lower validation loss more quickly and stands the chance of overfitting.

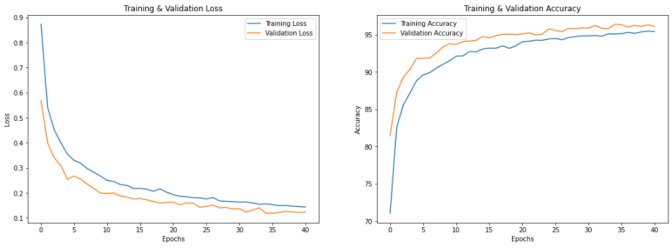

Fig. 8. Model performance: MLP

TABLE IV
ACCURACY AND LOSS METRICS

Metric	Accuracy	Loss
Validation	96.35%	0.1175
Train	94.88%	0.1634
Test	92.99%	0.2360

The curves in Figure 8 show that the training loss continues to decrease while the validation loss plateaus, which entails potential overfitting. To make sure that the training doesn't proceed indefinitely but halts when validation loss stops improving we applied early stopping to ensure optimal performance on unseen data.

B. CNN

Firstly, we made experiment to verify the effect of data augmentation. Both inception CNN models with and without

data augmentation were trained to compare the performance in learning curve. For training data without augmentation, the loss of training is close to zero but the loss of validation decreases instead of increases. It illustrates that the model is overfitting after 30 epochs. And network with data augmentation also performs better in validation accuracy the network without data augmentation. The results verify that data augmentation improves the generalization and robustness of neural network. It makes neural network robust to variability and enhance feature learning.

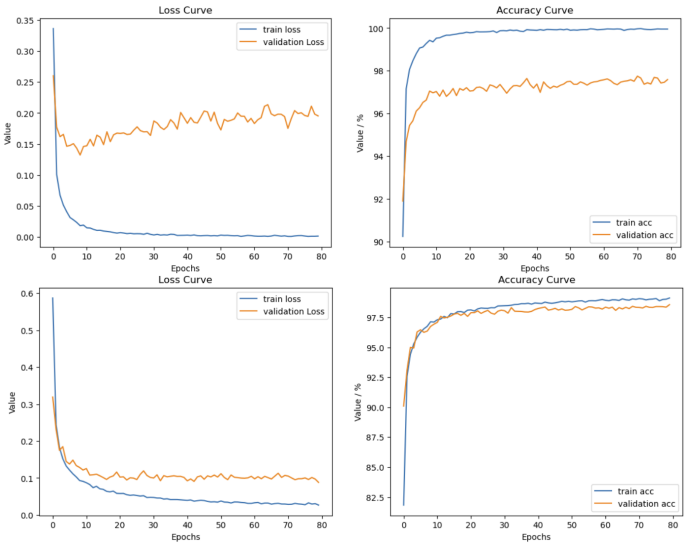

Fig. 9. Effect of data augmentation (Top is without data augmentation)

After data augmentation, we compared the effect of different learning rates. Because of the long time in training network, we only tested 3 learning rates, 0.001. 0.0005 and 0.0001. We use inception CNN to test the learning rate. The learning curves of different learning rates are shown in figure 10.

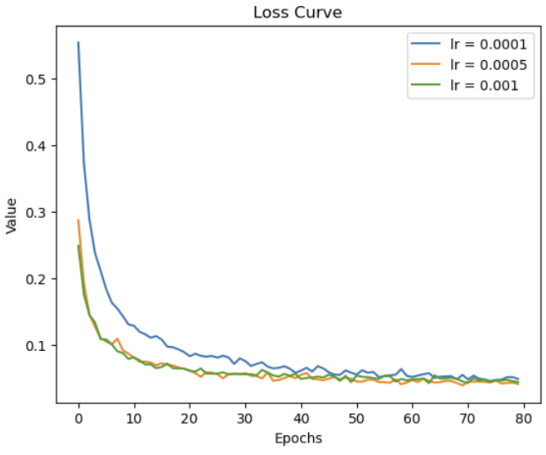

Fig. 10. Effects of different learning rates in CNN

As we can see, 0.001 is quite large that the learning curve is unstable and can't converge well to the global minima. 0.0001 is well, but 80 epochs is not sufficient for it to converge. Consider the trade-off between converge speed and stability, we selected 0.0005 as the final learning rate.

Next, we make a comparison between classic CNN model and inception CNN. The performances are shown in figure 9 and table V.

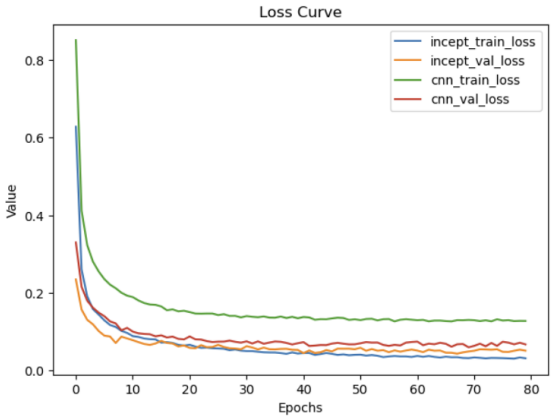

Fig. 11. Comparison of two models in learning curve

TABLE V
ACCURACY IN TWO CNNS

Methods	Accuracy	
	Training set	Test set
Classic	96.23%	97.77%
Inception	99.11%	98.56%

From the result, inception CNN has the better accuracy and lower loss in training set. It means inception CNN has more complicated structure and generalizes the data well. It improves 0.8% in accuracy in test set.

The performance of inception CNN indicates that our model has strong ability on classification of KMINST data set, with a relatively high accuracy 98.56%. The confusion matrix shows the precise prediction on each class.

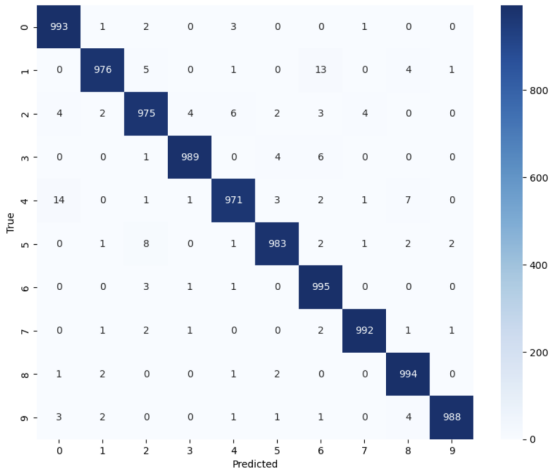

Fig. 12. Confusion Matrix: Inception CNN

To achieve the best accuracy of our networks, bagging strategy in ensemble learning was applied to make prediction. Classic CNN and two kinds of inception CNNs were utilized, and it achieved the best accuracy 98.63% of this project.

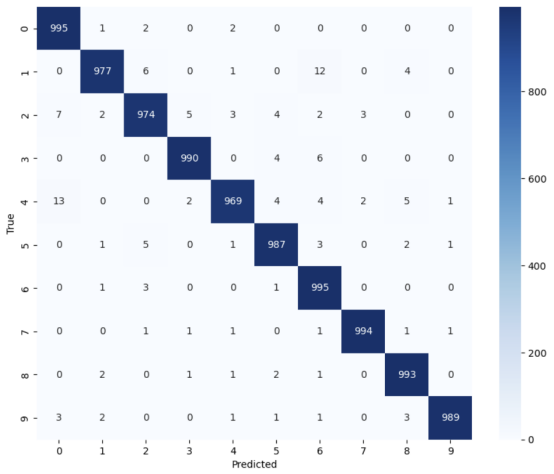

Fig. 13. Confusion Matrix: Ensemble CNN

C. Comparison

CNNs and MLPs are both types of useful neural networks in deep learning. Both of them introduce non-linearity and allow the model to learn complex relationships in the data. However, the difference between them is significant. Based on this project, we summarized the differences from the following aspects:

- *Structure*: CNN has more complicated structure than MLP. CNN is typically deeper networks with many layers like convolutional layers and pooling layers, as they aim to learn hierarchical features. MLP can be simpler and have fewer layers, depending on the specific application. Usually, CNN is available and powerful in more condition.
- *Capability*: Convolutional layer in CNN provides it the ability to automatically learn and extract features from images. It is effective in capturing spatial hierarchies of features in images. Compares to CNN, MLP need to flatten the image before training, and it will lose spatial information. MLP does not share weights among their neurons, and each connection has its own weight. This can lead to a large number of parameters.
- *Training*: Because CNN has complicated structure, it needs more memory and takes more time in training step, which is not economic for real task in industry and company. It's also difficult to make hyperparameters tuning in CNN.

From the performance of KMINST classification, CNN with inception modules provides 98.56% accuracy compared to MLP provides 92.99% accuracy in test set. It brings a more than 5% improvement. The results shown in confusion matrix indicates that CNN effectively solve some classification puzzles in MLP. As a conclusion, CNN performs better than MLP in image classification.

V. Conclusion

In this project, we explored the application of CNNs and MLPs for multiclass image classification using the KMNIST dataset. We investigated various aspects of model design, architecture, and training strategies to achieve the best performance.

Our experiments yielded exceptional results, with the CNN model achieving an outstanding accuracy of 98.56%, closely followed by the MLP model at 92.99%. These impressive outcomes were the product of careful hyperparameter selection and the strategic application of techniques such as Batch Normalization, Dropout, and data augmentation. Additionally, our findings also show that while Batch Normalization alone may not effectively counter overfitting, combining it with Dropout accelerates training and improves performance.

Furthermore, we experimented with the Inception architecture within the CNN, leveraging the power of Inception modules to efficiently capture features at various scales, ultimately improving classification performance. Ensemble learning was another strategy we employed to create a more robust and accurate predictive model. Combining multiple CNNs using bagging techniques allowed us to achieve best overall performance in 98.63% accuracy.

Overall, our study illustrates the significance of carefully designing and training neural networks for image classification tasks. The selection of appropriate model architecture, regularization techniques, and hyperparameters significantly influences the model's performance. The comparison between CNNs and MLPs shows different advantages and disadvantages in image classification.

References

[1] Richard Szeliski. 2010. Computer Vision: Algorithms and Applications (1st. ed.). Springer-Verlag, Berlin, Heidelberg.

[2] David E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group (Eds.). 1986. Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations. MIT Press, Cambridge, MA, USA.

[3] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.

[4] C. Szegedy et al., "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Boston, MA, USA, 2015, pp. 1-9, doi: 10.1109/CVPR.2015.7298594.

[5] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," arXiv preprint arXiv:1812.01718, 2018.

[6] E. D. Cubuk, B. Zoph, J. Shlens and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Seattle, WA, USA, 2020, pp. 3008-3017, doi: 10.1109/CVPRW50498.2020.00359.