# Identifying characters in KMNIST dataset using neural networks

## Declan Stockdale

# Overview

The number of potential architectures a neural network can take is essentially limitless added to the fact that there are additional performance heuristics and hyperparameter tuning, it can be daunting where to begin. This report implements multiple deep learning architectures starting from a simple naïve model of just a few hidden layers and culminates in a convolutional neural network.

The MNIST (Modified National Institute of Standards and Technology database) is a commonly used database for benchmarking testing performance of models. An alternative dataset that is gaining attention is the Kuzushiji-MNIST (KMNIST) dataset. The original MNIST dataset consists of thousands of handwritten numbers 0 through to 9. The KMNIST dataset alternatively consists of thousands of examples of one of 10 handwritten Kuzushiji (outdated Japanese alphabet) characters with the aim to extract written history from the potentially hundreds of millions of books written using the outdated alphabet which haven’t yet been translated [1].

# Methods and results

## Initial processing

The data has been downloaded from the Centre for Open Data in the Humanities (CODH) website [2]. All analysis has been performed using the python language and implementing using various packages most notably the Keras package which is user-friendly version of the more well-known TensorFlow package written by Google.

The combined dataset is comprised of 70,000 characters and was supplied as a train set of 60,000 images and a test set of the remaining 10,000 images. All images were 28x28 pixels and appeared to be centered. The images were converted into grayscale for easier processing. If this isn’t performed the model may add importance to the colour of the line or background rather than the shape of the lines.

Each Kuzushiji character is encoded as a number with the modern hiragana equivalent shown in table 1 below with a visual example from the training set below in figure 1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Label | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Actual | お | き | す | つ | な | は | ま | や | れ | ん |

Table 1. KMNIST label to hiragana equivalent

## A picture containing text Description automatically generated

## A picture containing text, clipart Description automatically generated

Figure 1. Display of first occurrence of each character with equivalent label in KMIST dataset

## Use of callbacks

#### reduceLR – Reduce learning rate

The learning rate is generally set at some value (0.001) but when approaching an optimum value may not be sufficiently small enough which results in a plateau of the accuracy of the model. This callback allows for the dynamic reduction of the learning rate once a plateau is reached in the aim of improving the accuracy of the model.

#### Early stopping

This is very useful as the computational power and time needed for training can be large. Early stopping stops the model once a given metric stops improving after a given number of epochs. The metric in this case is the validation accuracy which will be maximized. If the model plateaus and we continued the training process past this point, it would continue to learn the training data and would struggle to generalize to the testing set. It would also be a waste of power and potentially money if using a paid service.

#### Modelcheckpoint

This callback saves the best model for a given metric which is validation accuracy in our case. If during training a higher validation accuracy is achieved the previous best model is overwritten with the new best model. This may also be useful is there is a limit on how long training can occur or the potential of hardware, power failure or time limitations during training.

## Experiments

For each experiment the batch size was predefined as 128 and the number of epochs was set to 500. The output layer was a softmax activation function with an output of 10 classes, one for each unique Kuzushiji character in the dataset. The loss function used was the categorical cross entropy as this is a multiclass problem. A validation split of 20% was of the training data was used to obtain the validation metrics.

### Experiment 1. Naïve model with two hidden layers

A very simple model was first implemented using only 2 hidden dense layers each of 512 neurons using the relu activation function for both hidden layers. The network architecture has been displayed in **Figure 2** using the Keras plot plot\_model function.

The first two layers involve first loading the images as 28x28 images where the trailing 1 indicates they are greyscale. The images are then flattened to a single array of size 784 (28x28 = 784).

The layers titled dense and dense\_1 are the dense layers comprised of 512 densely connected neuron where each of the neurons is connected to all neurons in the previous layer. The final layer, dense\_2, is the output layer and is also a densely connected layer, however it only has 10 outputs, one for each possible character. The number of parameters can be seen in figure 2.

### Diagram Description automatically generated

Figure 2. Architecture of naïve model with two hidden layers

The final accuracy of the model on the testing set was 89.84%. From the confusion matrix in figure 3, it can be seen that the wrong predictions occur across almost all classes with no obvious pattern.

### Chart, line chart Description automatically generatedCalendar Description automatically generated

Figure 3. Left Accuracy of the naïve model over the number of epochs. Right Confusion matrix of predicted values

### 

### Experiment 2. Adding additional layers

One obvious parameter we can tune is the number of hidden layers of the architecture. A deeper network might learn the important features in the images or it may overfit to the training data. The number of hidden layers investigated ranged from 2 up to 10. The results of the highest validation accuracy before early stopping occurred can be seen in table 2 below

|  |  |
| --- | --- |
| Hidden layers | Accuracy % |
| 2 | 89.84 |
| 4 | 88.20 |
| 6 | 86.60 |
| 8 | 78.55 |
| 10 | 71.44 |

Table 2. Number of hidden layers and final accuracy of model

It can be clearly seen that increasing the number of hidden layers results in a reduced accuracy. This is indicative of the model’s overfitting on the training data and not being able to generalize back to the validation set. It’s possible that increasing training time may eventually help the model’s performance but that’s not a guarantee and it’s preferred to use a simpler model when available. In this case the simpler model is also the best performing model.

### Experiment 3. Adding dropout

Dropout is a one of many regularization methods that we can implement to reduce overfitting. It works by ignore some percentage of neurons in the hope that other neurons contribute to the prediction. An oversimplified explanation would be that it limits the reliance on a single neuron for the prediction and asks more neurons for their opinion before making a prediction. This improves the ability of the network to generalize and reduces overfitting on the training data.

For this experiment, the previous model architecture of 2 hidden layers is reused with subsequent dropout of 20% applied after each hidden layer.

Chart, line chart

Description automatically generated Calendar

Description automatically generated

Figure 4. Left Accuracy of the naïve model with dropout over the number of epochs. Right Confusion matrix of predicted values

The accuracy of this model was 89.15% which is slightly lower than the model without dropout. This may indicate that the model is efficient, and neurons randomly removed during dropout are necessary.

### Experiment 4. Convolutional Neural Network (CNN)

So far, the architectures used have been simplistic, but we can instead implement a CNN architecture which is widely used in deep learning on images. As the problem is very similar to the MNIST problem, it will be worth trying a CNN architecture that achieved a high accuracy and see how it compares to the KMIST database with the addition of the added callbacks. The one adapted achieved an accuracy over 99% in Kaggle competitions [**Kaggle ref]**. Its architecture can be seen in figure 5.

The largest difference compared to the previous models is the use of kernels and down sampling techniques. The input is not initially flattened but input as a 28x28 image. A two 3x3 convolution kernels are applied which reduce the output to 26x26 and 24x24 respectively, they also add nonlinearity to the neural network. The next step is a Max Pool further reduces the input. Another round of kernels and max pools are applied before finally flattening output and connecting to dense layers similar to the previous models. The printout of the model can be found in the appendix

Results from repeated CNNs can be different and its generally good practice to repeat the same experiment multiple times and use the average metric. Due to the time needed to complete multiple runs, only a single run was performed.

The accuracy of the CNN model was 96.11% which is a significant jump compared to the non-CNN models implemented earlier. The confusion matrix in figure 6 also shows a significant drop in incorrectly identified classes

Diagram

Description automatically generated with medium confidence

Figure 5. Architecture of CNN used.

Line chart

Description automatically generated with low confidenceCalendar

Description automatically generated

Figure 6. Left Accuracy of the CNN model over the number of epochs. Right Confusion matrix of predicted values

### Experiment 5 Generate synthetic data

In cases where the number of examples in the training set are limited, we have a few options. The absolute best option would be to collect more training examples but in this case that’s not an option. Instead, we can create more training examples by randomly modifying existing training examples.

The Keras package ImageDataGenerator allows for multiple options modify images. The options chosen for this dataset are a random rotation of 15 degrees, a slight zoom, shifting the width and height of the image and distorting the image. All of these characteristics may be observed in the testing set due to the nature of handwritten characters with different slants, size of characters or may occur due to off centering of training examples in the data collection and cleaning process.

Early stopping occurred significantly later compared to other models due to the data generation figure 7. It alters the input images as they are fed into the neural network so there is more to learn and is why the accuracy improves over a longer period compared to the model without feeding in the modified images. The confusion matrix in figure 7 also shows a similar result to the original CNN architecture which is expected. It appears that the majority of incorrectly labelled characters occur in the characters labelled 2,4 and 5. More data or more data augmentation may improve the model to improve predictions for these classes.

Chart, line chart

Description automatically generated Calendar

Description automatically generated

Figure 7. Left Accuracy of the CNN model over the number of epochs. Right Confusion matrix of predicted values

## Findings

Table 3 below shows the accuracy and the complexity of the trainable parameters of each model. It can clearly be seen the increasing the number of layers in the naïve model resulted in a reduction of performance possibly due to added complexity. If early stopping was removed, we may see improvements, but the training time would be prohibitive.

It can also be seen the number of trainable parameters in the CNN model is comparable to the naïve model with 2 hidden layers. This is due to the usage of kernels and max pooling which substantially reduces the input between layers.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Accuracy % | Trainable parameters | Epochs before stopping |
| Naïve 2 hidden layers | 89.84 | 669,706 | 18 |
| Naïve 4 hidden layers | 88.20 | 1,195,018 | 25 |
| Naïve 6 hidden layers | 86.60 | 1,720,330 | 19 |
| Naïve 8 hidden layers | 78.55 | 2,245,642 | 16 |
| Naïve 10 hidden layers | 71.44 | 2,770,954 | 16 |
| Naïve 2 with dropout 20% | 89.15 | 669,706 | 29 |
| CNN | 97.46 | 691,786 | 19 |
| CNN + data augmentation | 98.48 | 691,786 | 49 |

Table 3. Training metrics of all models reported

The results from table 4 below show the CNN models greatly outperform the naïve models in every metric and its quite clear in the number of correctly classified images how superior they performed. The difference between the CNN with and without data augmentation is minimal with only a slight increase in the number of correctly classified images which is closer than the accuracy suggested in table 3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Correct | Incorrect | precision | Recall | F1 score |
| Naïve 2 hidden layers | 9012 | 988 | 0.90 | 0.90 | 0.90 |
| Naïve 2 with dropout 20% | 8884 | 1116 | 0.89 | 0.89 | 0.89 |
| CNN | 9845 | 155 | 0.98 | 0.98 | 0.98 |
| CNN + data augmentation | 9849 | 151 | 0.98 | 0.98 | 0.98 |

Table 4. Performance metric of all models taken from macro average from sklearn.metrics package

The metric reported for all models so far has been accuracy, but it may not be the best metric. Precision is way to measure true positives and recall measures false negatives. As its important to be correct in all predictions we should use the F1 score which combines precision and recall. The AUCROC may also be a metric to explore and compare various models.

If applying this architecture to actual prediction we could improve performance by integrating into natural language processing grammar check which could pick up if predicted labels make grammatical sense.

### Conclusion

Various architectures of deep learning neural nets have been developed and tested for their performance on the KMNIST dataset. CNNs outperformed all other models achieving an accuracy of 97.46% which increased to 98.48 with data augmentation. Of the other models, it was found that using a simple model of 2 hidden layers outperformed similar architectures with additional layers. It was also found that dropout marginally reduced the accuracy of the model.

It appears that the incorrectly labelled samples occur in only a handful of classes. More data collection or a more sophisticated data augmentation methodology could be developed to further increase the predictive power of future models.

## References

1. <https://www.kaggle.com/code/aakashnain/kmnist-mnist-replacement/notebook>
2. http://codh.rois.ac.jp/kmnist/dataset/kmnist/
3. https://www.kaggle.com/code/elcaiseri/mnist-simple-cnn-keras-accuracy-0-99-top-1/notebook

## Appendix

Table

Description automatically generated

Appendix figure 1. Architecture of CNN used in experiment 4.