Imperial College London

# ACSE-4: Machine Learning Mini-Project

**Team Sigmoid**
Ye Liu, Oliver Boom, Mingrui Zhang, Deborah Pelacani Cruz

May 24, 2019

## 1 Abstract

Adaptations of the well-established convolutional neural networks AlexNet and LeNet5 were created and implemented on a training routine for the Kuzushiji-MNIST dataset. The training was performed using a 'SupervisedLearning' wrapper class over 'PyTorch' and 'ScikitLearn' library modules. Attempts to maximise the prediction accuracy on a test set included: comparison between Adam and Stochastic Gradient Descent optimisation techniques; tuning of L2 regularisation weight decay; application of a set of in-built and external data augmentation approaches; K-folding validation; regularisation with dropout and batch normalisation, and different model capacities. The best performing model was found to be a modified version of the AlexNet network consisting of 4 convolutional layers with batch normalisation and in-built data augmentation. The resulting accuracy of this model was 98.87%, as it performed on an unseen set of images. Improvements to our models could have been achieved by performing further tests with external data augmentation and exploring the effects of other loss and activation functions, but these were restricted by our time-frame and computational power.

## 2 Introduction

Figure 1: Kuzushiji script

Extensive machine learning models have been applied to improve the recognition of hand-written numbers through neural network training on the MNIST data set. In a similar manner, the Kuzushiji MNIST dataset, here referred to as KMNIST, has been created in order to provide the same opportunities to the identification of Kuzushiji characters. The KMNIST dataset consists of 60,000 training images, each 28x28 pixels in a single grayscale channel, and distributed over 10 classes. The data is fully balanced between its 10 classes. It is considered a more challenging variant of the MNIST benchmark computer vision dataset. The purpose of this project is to create and implement a neural network to maximise the accuracy in their prediction.

### 2.1 Data

Adaptation of two well-known networks, LetNet5 and AlexNet, are used to perform a series of training and validation routines in order to understand how to better predict the KMNIST characters. Models with a high validation performance are then tested on an unseen dataset consisting of 10,000 images.

### 2.2 Code Structure and Submission

In our training approach, a 'SupervisedLearning' class was created as a wrapper to perform testing and validation using 'pytorch' and 'scikitlearn' as the main libraries. Given a set of input parameters, the class is able to split the full data into training and validation sets, normalise and apply transforms, and then train and validate a model. The input parameters and methods of this class allow for easy transfer learning and hyper-parameter tuning. It also provides an $early - stop$ and a $full - set$ training option. With this class, performing tests became a straight-forward task and ensured consistency as the group members ran different models in their respective machines. For full details on the structure of the code, visit the github page and its documentation. The wrapper class and other useful tool functions are part of the 'KMNIST_Learning.py' module.

The model performance on the test set was obtained through submissions on Kaggle. The accuracy of resulting models on the test set was only allowed to be calculated twice a day.

## 3 Data Pre-processing

An initial EDA of the dataset found the dataset to be of high quality and not requiring cleaning. No images were removed from the dataset giving a full 60,000 images for training and 10,000 images for testing.

The preprocessing of the data consists of two main steps: 1) standard score normalisation of the training, validation and test sets using the mean and standard deviation of the training set, and 2) a series of data augmentation transforms, which are optional to the user.

The following data augmentations are provided within the main class found in the 'KMNISTLearning.py' file. These standard 'torchvision.transforms' augmentations and are applied to the training set only:

- Random rotation by a maximum of 10 degrees.
- Random cropping
- Random Affine
- Color Jitter

Data augmentation was further experimented with through the 'albumentation' library. These augmentations were applied to the training set before learning in attempt to reduce generalisation error on our final model. Due to structure difficulties and time limitation, these augmentations were carried to the 'SupervisedLearning' class and is also embedded within the validation data set as it gets split.

Examples of this data augmentation method can be found in the 'KMNIST _ TLDataAug.ipynb' notebook. The augmented function allows the user to choose the types of augmentation and their probabilities, as well as by how many images the training set must be expanded. Augmentations applied include a variety of random blurs, noise addition and image distortions with varied probabilities. A maximum of 0.4% increase in validation accuracy was observed using this method.

These augmented datasets were then fed into our pre-trained models using transfer learning on either the final or the final two layers.

## 4 Training Approach

### 4.1 Optimisation and Loss Functions

The main 'SupervisedLearning' class takes in as parameters an optimisation function and a loss function. As explained in the 'Validation Approach and Hyperparameter Selection' section below, a quick run of tests were made comparing the Adam and Stochastic Gradient Descent methods over the AlexNet model. The loss function used throughout the development of the project was the Cross Entropy loss function. Although other relevant loss functions could have been used and tested on, the time priority was given to more relevant hyperparameters and regularisations.

### 4.2 Early Stop

The early stop parameter within the main 'SuperviseLearning' class was implemented to prevent wasting time training a model that is not improving. The user may input a relative tolerance that the model must improve by on each epoch. If the relative tolerance threshold is not exceeded for a given number of epochs (labels the patience count), then the model saves the best weights and ends training early.

## 5 Validation Approach and Hyper-parameter Selection

Evaluation of our trained networks were performed on a validation set, which consisted of a 10% subset of the training data. Given the training data set is relatively large, a shuffled split is guaranteed create a well-balanced validation set. As the training accuracy and loss could not represent how the network would perform on an unseen data set, this cross-validation method became crucial to evaluate the network's final performance. There are two approaches available to the user in the 'KMNIST_Leanring.py' source code:

### 5.1 Hold-Out Validation

The original training data set is divided into two parts, 90 percent for training and 10 percent for validation using 'StratifiedShuffleSplit'. Then the validation accuracy and loss are calculated as the final generalisation results. This is implemented in the 'SupervisedLearning' class.

### 5.2 K-Fold

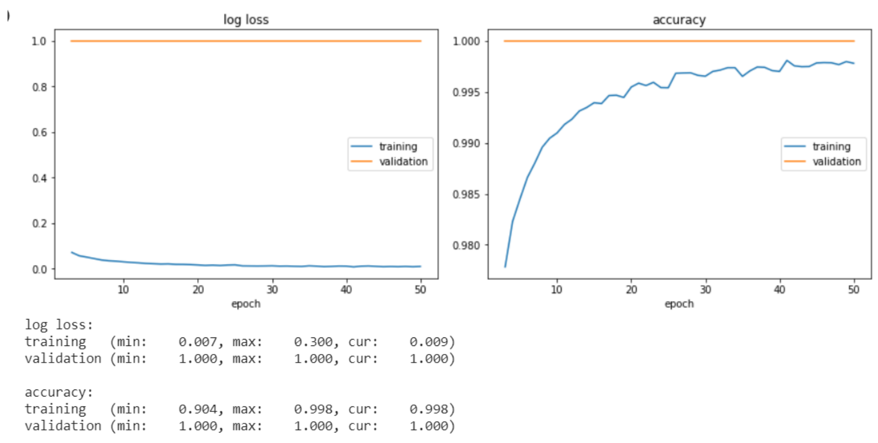
In order to better estimate the model generalization error, K-Fold validation is implemented as a child class of 'SupervisedLearning' called 'KFoldValidation'. The original training data set is randomly divided into a $k$ number of folds using the 'StratifiedKFold'. Then each fold is used as a validation data set in turn. Finally, the averaged validation loss and accuracy would be considered as the generalization result. This method is slower than the previous method, but does provide a more robust and more accurate estimation of the overall generalisation error.

### 5.3 Hyper-parameter Selection

Within the wrapper class, the tuning of many parameters are a straightforward task. However, due to the strict time-frame, only three were selected based on what we as a group decided to be the most relevant for the task. A summary of our findings are described below:

- **Optimiser**

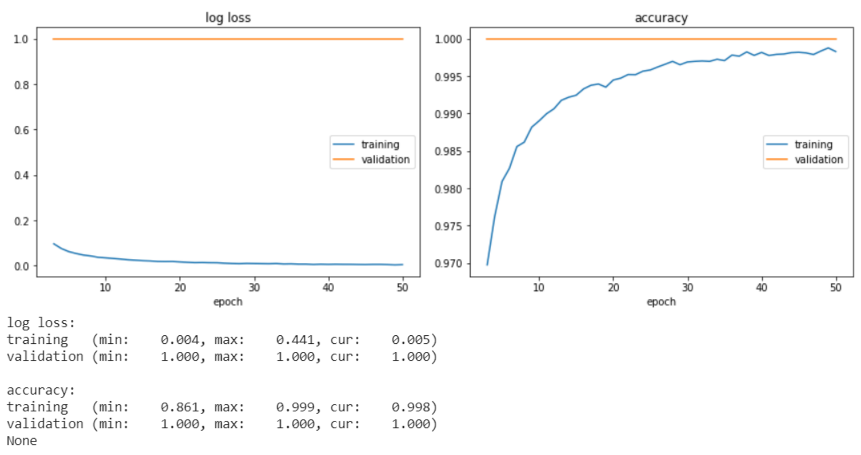
  The Adam optimiser gives better accuracy and runs faster than SGD in our DeepAlexNet network. We also tested the SGD optimiser using AlexNet,it could also optimise the model well as shown in the pictures below.Both of them get the validation accuracy of 0.998 after 50 epochs.



```
log loss:
training   (min:    0.007, max:    0.300, cur:    0.009)
validation (min:    1.000, max:    1.000, cur:    1.000)

accuracy:
training   (min:    0.904, max:    0.998, cur:    0.998)
validation (min:    1.000, max:    1.000, cur:    1.000)
```

Figure 2: DeepAlexNet with Adam optimiser



```
log loss:
training   (min:    0.004, max:    0.441, cur:    0.005)
validation (min:    1.000, max:    1.000, cur:    1.000)

accuracy:
training   (min:    0.861, max:    0.999, cur:    0.998)
validation (min:    1.000, max:    1.000, cur:    1.000)
None
```

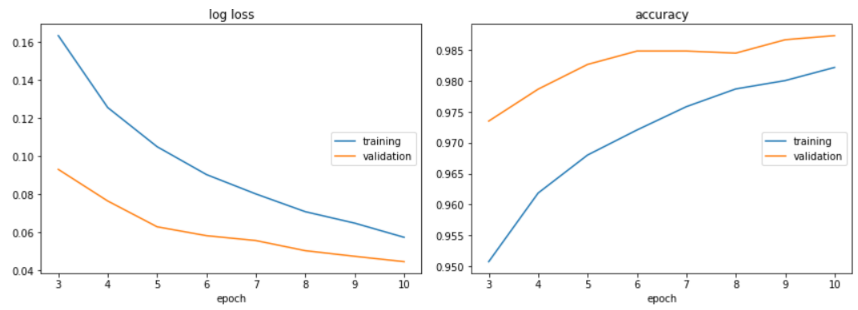
Figure 3: DeepAlexNet with SGD optimiser

- **Learning Rate**

  The learning rate used in the AlexNet with the Adam optimiser was $1 \times 10^{-4}$, as it was found that Adam results were oscillatory and unstable with larger learning rates. With SGD optimiser, it was found that a learning rate of 0.01 provided a sufficiently stable result.

- **Weight Decay**

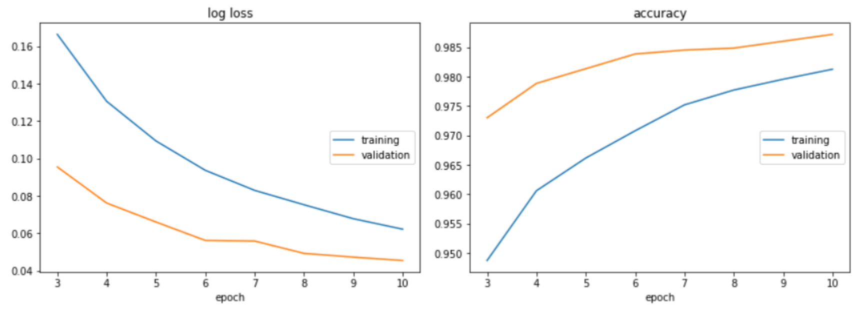
  As for the weight decay, tests were performed for the following 4 values [ 0, 1e-3, 1e-4, 1e-5 ] with the original AlexNet using SGD, each for 10 epochs.The graphs below show the different accuracy with increasing weight decay.



```
log loss:
training   (min:    0.057, max:    0.791, cur:    0.057)
validation (min:    0.045, max:    0.238, cur:    0.045)

accuracy:
training   (min:    0.749, max:    0.982, cur:    0.982)
validation (min:    0.928, max:    0.987, cur:    0.987)
```

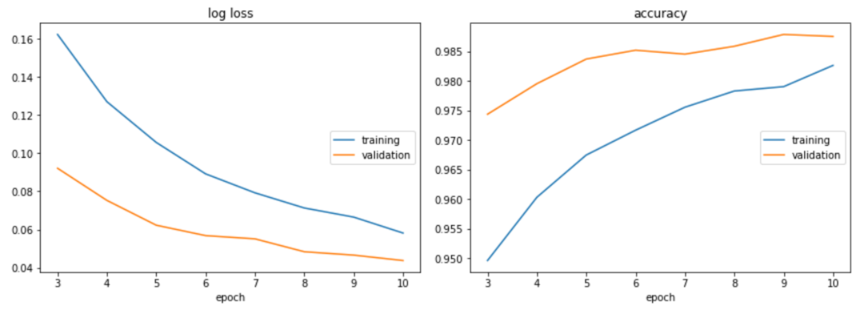
Figure 4: AlexNet with weight decay of 0



```
log loss:
training   (min:    0.062, max:    0.797, cur:    0.062)
validation (min:    0.045, max:    0.260, cur:    0.045)

accuracy:
training   (min:    0.746, max:    0.981, cur:    0.981)
validation (min:    0.920, max:    0.987, cur:    0.987)
None
```

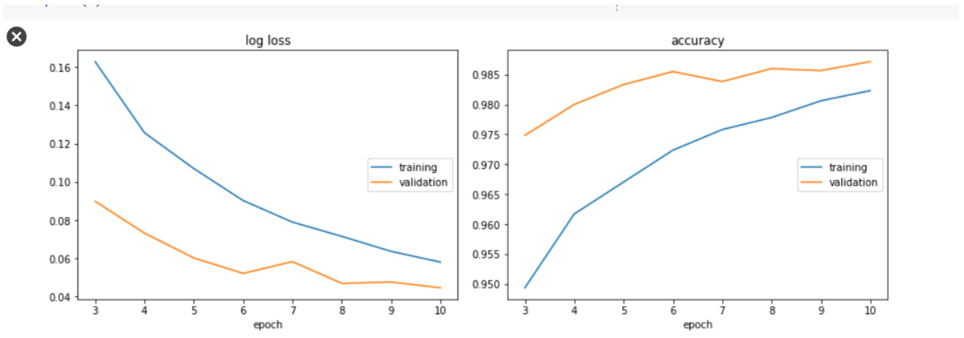
Figure 5: AlexNet with weight decay of $1 \times 10^{-3}$



```
log loss:
training   (min:    0.058, max:    0.773, cur:    0.058)
validation (min:    0.044, max:    0.242, cur:    0.044)

accuracy:
training   (min:    0.754, max:    0.983, cur:    0.983)
validation (min:    0.928, max:    0.988, cur:    0.988)
None
```

Figure 6: AlexNet with weight decay of $1 \times 10^{-4}$



```
log loss:
training   (min:    0.058, max:    0.796, cur:    0.058)
validation (min:    0.045, max:    0.245, cur:    0.045)

accuracy:
training   (min:    0.747, max:    0.982, cur:    0.982)
validation (min:    0.926, max:    0.987, cur:    0.987)
None
```

Figure 7: AlexNet with weight decay of 1x10-5

From the above graphs (Figures 4-7) it is possible to infer that the AlexNet network with the weight decay of 0 and 1e-4 performed better than the other two. As the best performing weight decay, $1 \times 10^{-4}$ was selected for further runs. For Adam optimiser, the weight decay 0 was the best choice.

## 6 Neural Network Architectures

Four neural network architectures were implemented as modified versions of the LeNet5 and AlexNet networks. The choices made in these modifications are justified by training time and increase in validation accuracy. The class to generate each model can be found in the KMNIST_SDeepAlex_old_aug.ipynb‘ notebook.

- **mLeNet5**: The capacity of the model was improved by increasing the number of output neurons in the second fully connected layer from 120 to 1150. The dropout regularization technique was also applied during training in order to prevent overfitting and improve the overall generalisation error.
- **mAlexNet**: A traditional AlexNet architecture with the first 3 layers removed. All fully-conneced layers are kept with the last layer outputting 10 classes instead. Model regularisation with batch normalisation was also implemented along with the dropout technique during training.
- **DeepAlexNet**: An improved version of **mAlexNet**, with 4 overall convolution layers. Figure 8.a shows a summary of the full architecture, including hidden layers.
- **SuperDeepAlexNet**: further improved version of mAlexNet with 5 convolution layers. This net has about $4 \times 10^5$ neurons and $1.8 \times 10^7$ parameters (Figure 8.b)

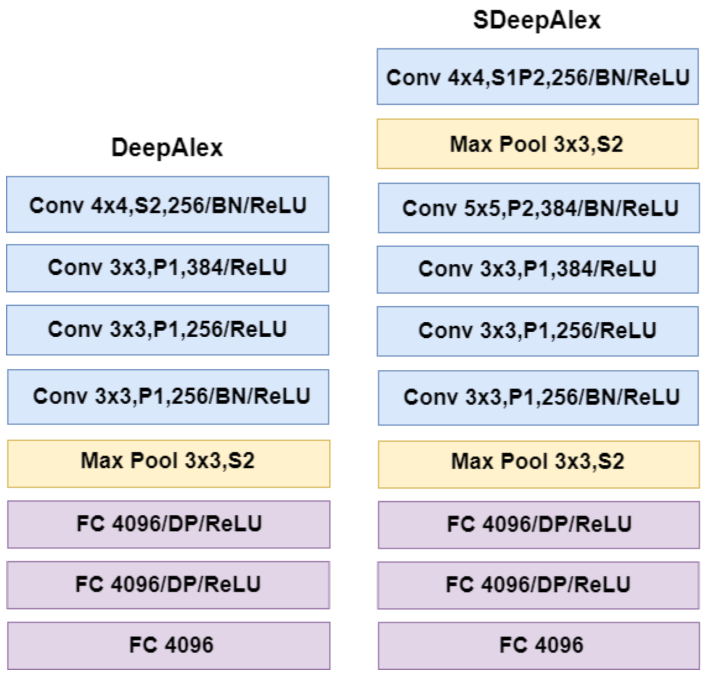
### 6.1 Ensemble Network

In addition to the individual architectures, a majority voting ensemble approach was also employed. Included in the ensemble was any model that achieved a validation accuracy of greater than 98%. The probability vector from the individual models are averaged and the most probably of this vector is selected. This strategy is most effective when a diverse selection of models is used, and the flaws of the individual models can be negated. Model averaging and stacking are powerful tools given a diverse set of models.

## 7 Results &Discussion

### 7.1 The Highest Scoring Models

Models with highest validation accuracies were selected to perform a trainnig over the full data-set prior to Kaggle submission. Starting from a simpler LeNet5, gains in validation accuracy were first found in the use of the deep and



(a) DeepAlexNet (b) SDeepAlexNet

Figure 8: Neural Network Architecture showing all hidden layers and their number of output neurons. BN and DP refer to the implementation of batch normalisation and dropout, respectively, in the hidden layers. ReLU was the chosen activation function for all these layers

more sophisticated AlexNet instead. From longer training runs it was found that a model with greater capacity could still have been beneficial as no over-fitting was occurring. Modifications to AlexNet were made to increase capacity, but still no over-fitting was found. Due to the limited training time available it was necessary to prioritise either increasing model capacity or increasing training data through data augmentations. As over-fitting was not observed in the validation set the decision to focus on capacity was made and more epochs were ran. Augmented datasets were created and through using transfer learning used as further training data on the DeepAlexNet and SDeepAlexNet trained networks due to time constrains. Applying transfer learning with augmented data on these two models provided us with unexpected lower accuracy results, although this technique was poorly tested. Their full running settings are as follows:

DeepAlexNet

- Batch Normalisation
- Dropout
- In-Class augmented data
- Transfer Learning with external augmented data (training data expanded with 20,000 additional images)
- Optimiser: SGD
- Loss Function: Cross-Entropy
- Learning Rate: $1 \times 10^{-2}$
- Total number of epochs: 50
- L2 Weight Decay: $1 \times 10^{-4}$

- Overall Test Accuracy: 97.90%

DeepAlexNet

- Batch Normalisation
- Dropout
- In-Class augmented data
- Optimiser: Adam
- Loss Function: Cross-Entropy
- Learning Rate: $1 \times 10^{-4}$
- Total number of epochs: 50
- L2 Weight Decay: 0
- Overall Test Accuracy: 98.77%

SDeepAlexNet

- Batch Normalisation
- Dropout
- In-Class augmented data
- Optimiser: SGD
- Loss Function: Cross-Entropy
- Learning Rate: $1 \times 10^{-2}$
- Total number of epochs: 50
- L2 Weight Decay: $1 \times 10^{-4}$
- Overall Test Accuracy: 98.57%

## 7.2 Ensemble

It was suspected that the lack of diversity between the included models (with all models being a variation on either LeNet5 or AlexNet) would mean that an ensemble approach would not have it's usual synergistic effect. This suspicion was confirmed when comparing the individual models validation scores and to that of the ensemble. The voting ensemble achieved a greater accuracy than the average of the models it was composed of, performing similarly with the best few results, but it did not achieve new accuracy highs.

A late submission was also made to see if the lack of performance on the validation set matched poorer performance on the test set, with the ensembled network achieving a score of 0.97833, which is good but not as accurate as the best individual networks.

## 8 Conclusion, Limitations and Further Work

The final best performing model was on the DeepAlexNet network using Adam optimisation on 50 epochs and applying in-built data augmentation. The network has 4 convolution layers providing enough capacity for the features, which could overcome the underfitting problem. On the other hand, to suppress the potential over-fitting problem, dropout and data augmentation are applied.

The biggest limitation for this work was the short time frame and lack of computer power so that more tests could be performed when training the data set. Given these technical tools, improvement to our model accuracy could have been achieved by producing more diversified models and ensembling them, as well as performing a full training with a multiplied and augmented training set as opposed as only performing transfer learning. It could be useful to explore the effects of different activation and loss functions during training. Exploring other well-known networks besides LeNet5 and AlexNet would also improve the results obtained during this project.