CMPSC 448 Final Report

Jonathan Yudhistira, John Durrell, Eddie Ubri, Jacob Snyder

## Task, Dataset, and Preprocessing

In this project, we were asked to find and choose a classification task and implement two deep learning systems for that task. The task that we chose was letter and digit classification where the model would be fed an image of a handwritten or typed letter or digit and output exactly what symbol the image contained. We chose to implement a Recurrent Neural Network and a Convolutional Neural Network for this task. The dataset we used for this task is the EMNIST balanced dataset which is a set of handwritten letters and digits with 47 classes and 131,600 samples. We chose this dataset because of the balancing scheme it uses where each digit/letter has the same number of samples. The dataset mixes the samples for some lowercase with their uppercase counterparts as they are largely indistinguishable (such as “X” and “x” or “O” and “o”). For preprocessing, the RNN reshapes the training and test sets from the dimensions (124800, 28, 28) to dimensions (124800, 784, 1) and converts the training and test labels to one-hot encoding. The only preprocessing the CNN implementation does is transforming the EMNIST dataset into a Pytorch tensor. Any other preprocessing such as normalizing the colors to black and white is handled outside of this implementation, instead having already been done by EMNIST.

## Implementation and Architectures

The RNN model uses 6 layers from Keras, first an LSTM layer with an output space with dimensionality 64, a batch normalization layer to normalize the hidden layers, another LSTM layer with output space dimensionality 32, another batch normalization layer, a ReLU activation function with output space dimensionality 64, and a SoftMax layer with output space dimensionality 47, which is the number of classes in the dataset. The model uses the Adam optimization algorithm from Keras, which is a stochastic gradient descent algorithm based on estimations of first and second-order moments. The model’s loss function is provided by Keras’ categorical crossentropy function, and the model records the accuracy metric of itself.

The CNN model uses 7 layers; the first layer is a convolutional layer which has a ReLU activation function with a kernel size of 5 that maps 1 input channel to 50 output channels. It is followed by a max pooling layer with a stride of 2. These two layers are then repeated with the convolutional layer mapping 50 input channels to 240 output channels with a kernel size of 5. The next three layers are fully connected layers, each using a ReLU activation function, that together map the number of features to 47. For the loss function, the CNN model uses the crossentropy function from PyTorch. The optimization function used was a stochastic gradient descent function also from PyTorch.

## Training Details

By default, the EMNIST dataset employs an ~86%/~14% train test split for the data, having 112,800 samples for the training data and 18,800 samples for the testing data. This split is used by both the RNN and CNN models. When training the RNN model, using 20 epochs was chosen along with a batch size of 64. This choice for the batch size made the learning algorithm mini-batch gradient descent. We also chose to use a learning rate of 0.001 for the RNN.

For the CNN, we also chose to use 20 epochs but were able to choose a lower batch size of 32 making the learning algorithm for the CNN also mini-batch gradient descent. Unlike the RNN, we chose to use a variable learning rate that would change depending on the current epoch. During the first 10 epochs, the learning rate is 0.1, then 0.01 for the next 5 epochs, and finally 0.001 for the last 5 epochs.

## Results, Observations, and Conclusions

After training the two models, the Convolutional Neural Network was found to perform slightly better than the Recurrent Neural Network in terms of accuracy and much better in terms of speed. The CNN peaked at 89% accuracy and took about 40 seconds per epoch in training. However, the RNN peaked at 88% accuracy and took about 80 seconds per epoch.

Both models reached high accuracy rather quickly, but the Convolutional Neural Network still performed better earlier than the Recurrent Neural Network. The CNN reached 85% accuracy after one epoch while the RNN had lower accuracy scores for the first few epochs, not reaching 82% accuracy until its tenth epoch. However, both models did have their accuracies plateau before finishing training.

Considering the results of the accuracy and speed of both models, we believe that using a Convolutional Neural Network would be a better choice for this classification task than using a Recurrent Neural Network. This is even further supported by the results from earlier epochs for each model since the CNN was able to reach a higher accuracy with less training than the RNN, so with proper optimization it would need fewer training epochs to reach its maximum accuracy than the RNN would.

## Challenges and Obstacles

For both models, the biggest challenges revolved around the batch size and performance of the models. For the CNN, the training process was maxing out the memory available in the GPU training instance, around 1.3GB. We found that having smaller batch sizes allowed us to increase the size of the model and thus increase its performance. For the RNN, training the model was extremely slow with smaller batch sizes to the point of not being able to feasibly tune the parameters of the model easily. We solved this by increasing the batch size to 128 while tuning and testing the model and switching back to 64 for the batch size when getting the results of the model.

Our code is publicly available at https://github.com/jdurrell/CMPSC448-Letter-Classification.