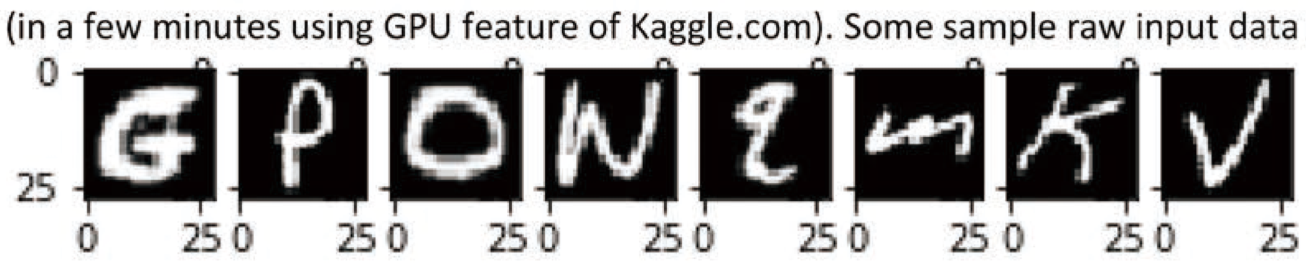
Offline Handwritten Letters Recognition Using CNN

Yingyin Xu   Taotao Qian   Emily Eunhee Kang   Chenhao Qian

Project Overview

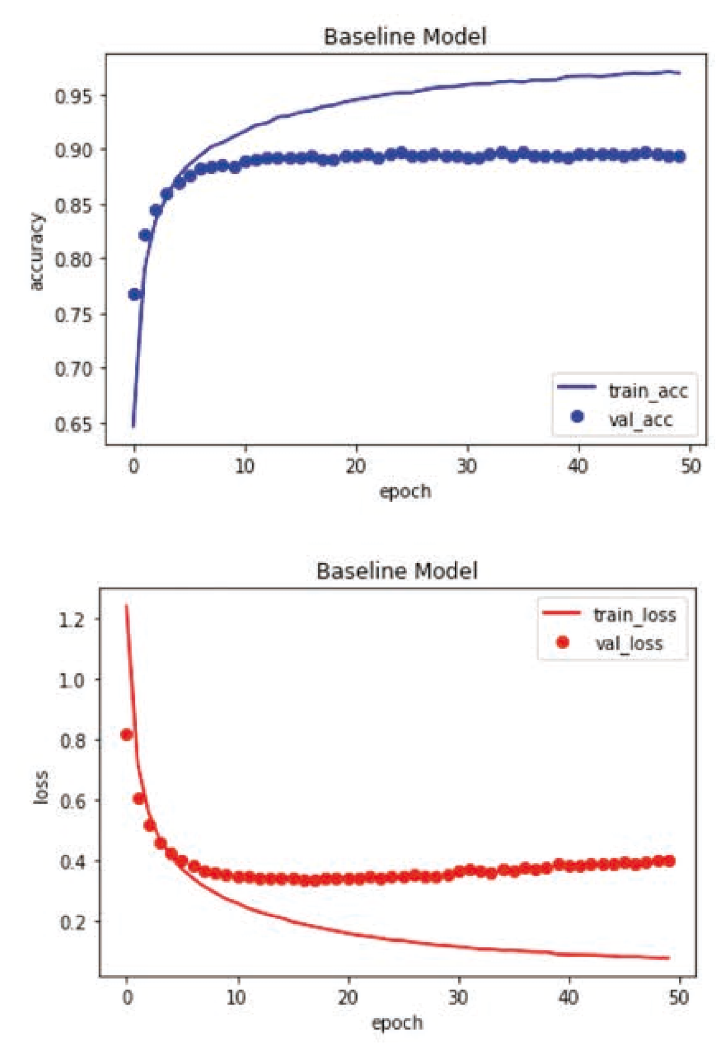
The main goal of this project is to construct a solid convolutional neural network model for recognizing handwritten characters in an offline setting. For the scope of this project, we restrict that the character being an English alphabetic letter in upper case. This restriction is set due to our limited resources, such as available dataset, computational power at hand, etc.

In this project, we used a simple neural network model as a baseline model to compare the performance of our models. Then, we build two versions of CNN model using two algorithm packages, one being Keras and the other is AlexNet. Both CNN models significantly outperform the baseline model, giving a classification accuracy over 95% (AlexNet is over 98%). The AlexNet algorithm offers the best performance. Yet the drawback of this model is its heavy resource consumption. On the other hand, Keras is much easier to configure and run a lot faster (in a few minutes using GPU feature of Kaggle.com). Some sample raw input data is shown below.

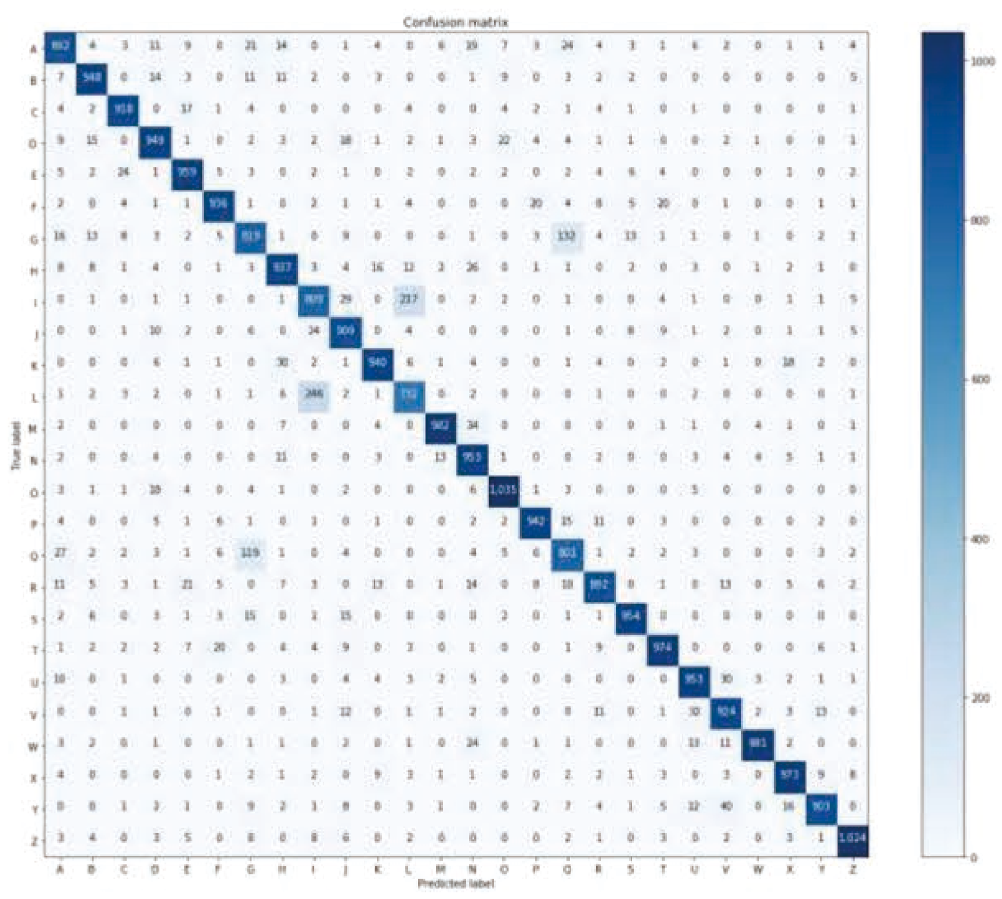


Baseline Model Evaluation

Baseline Model Performance in Training (50 epochs)



Baseline Model Performance on Leave-out Test Set (50 epochs)

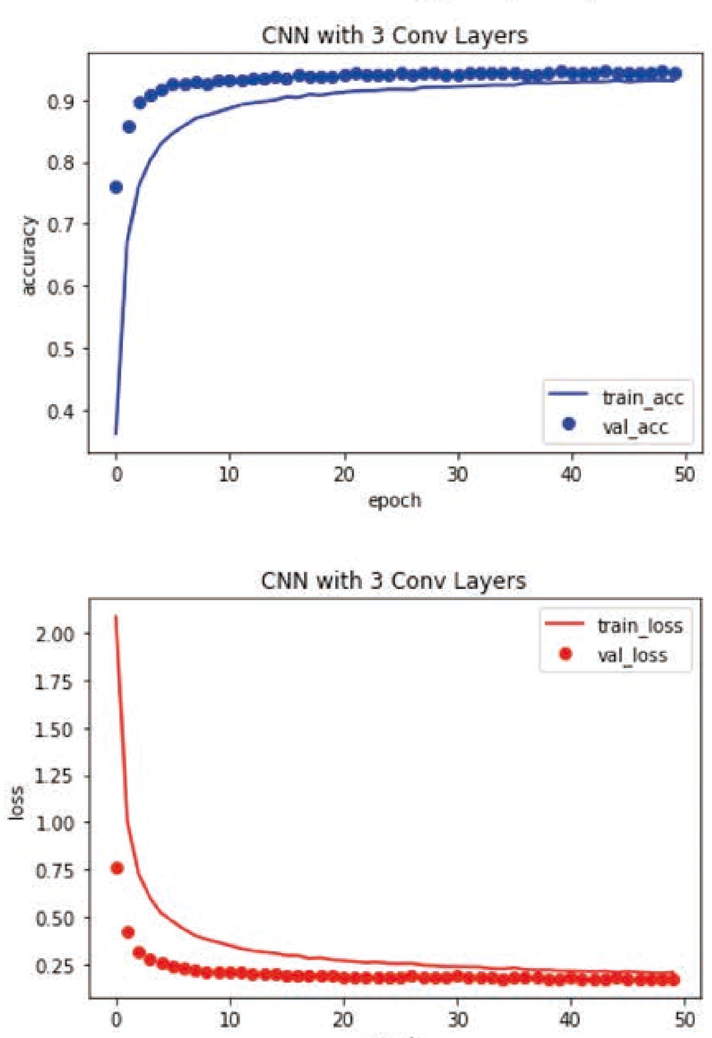


Overall Accuracy= 0.9023648648648649

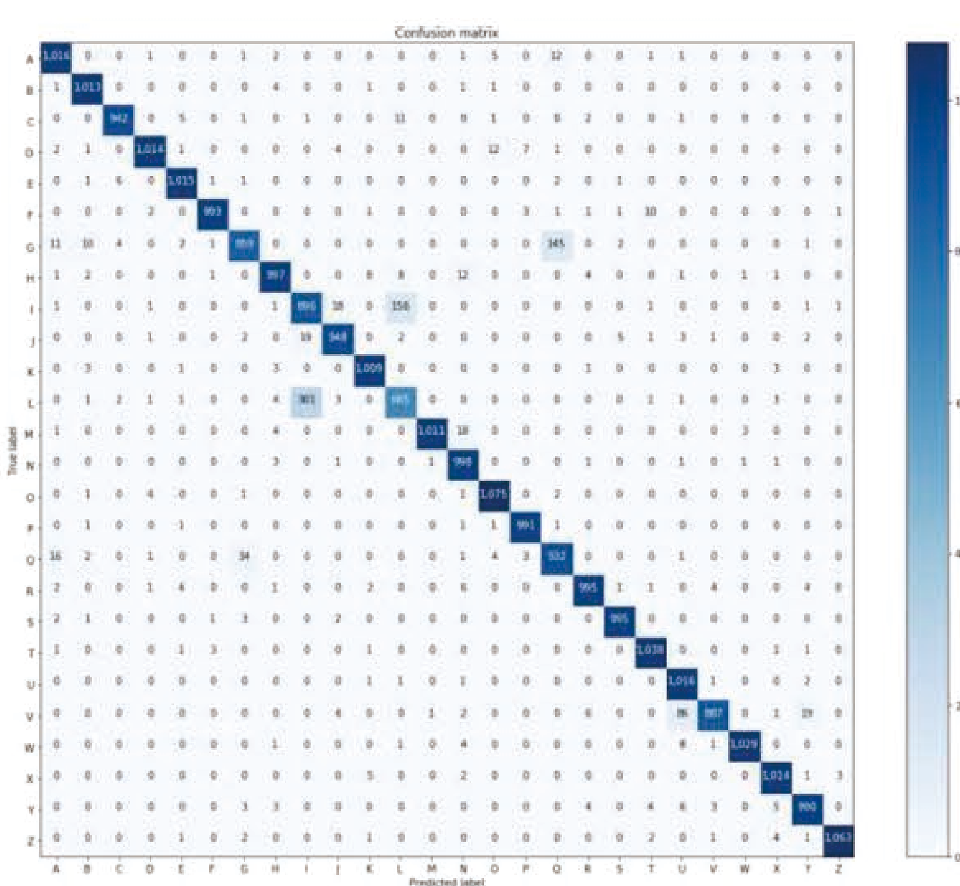
Misclassification= 0.09763513513513511

Keras CNN Model Evaluation

Model Performance in Training (50 epochs)



Model Performance on Leave-out Test Set (50 epochs)

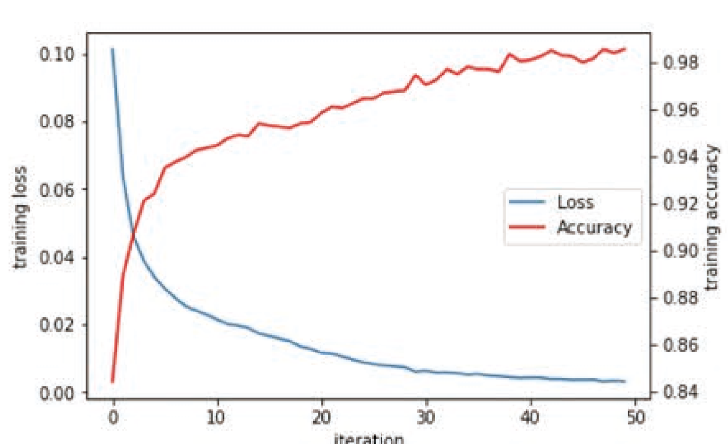


Overall Accuracy= 0.9542417417417417

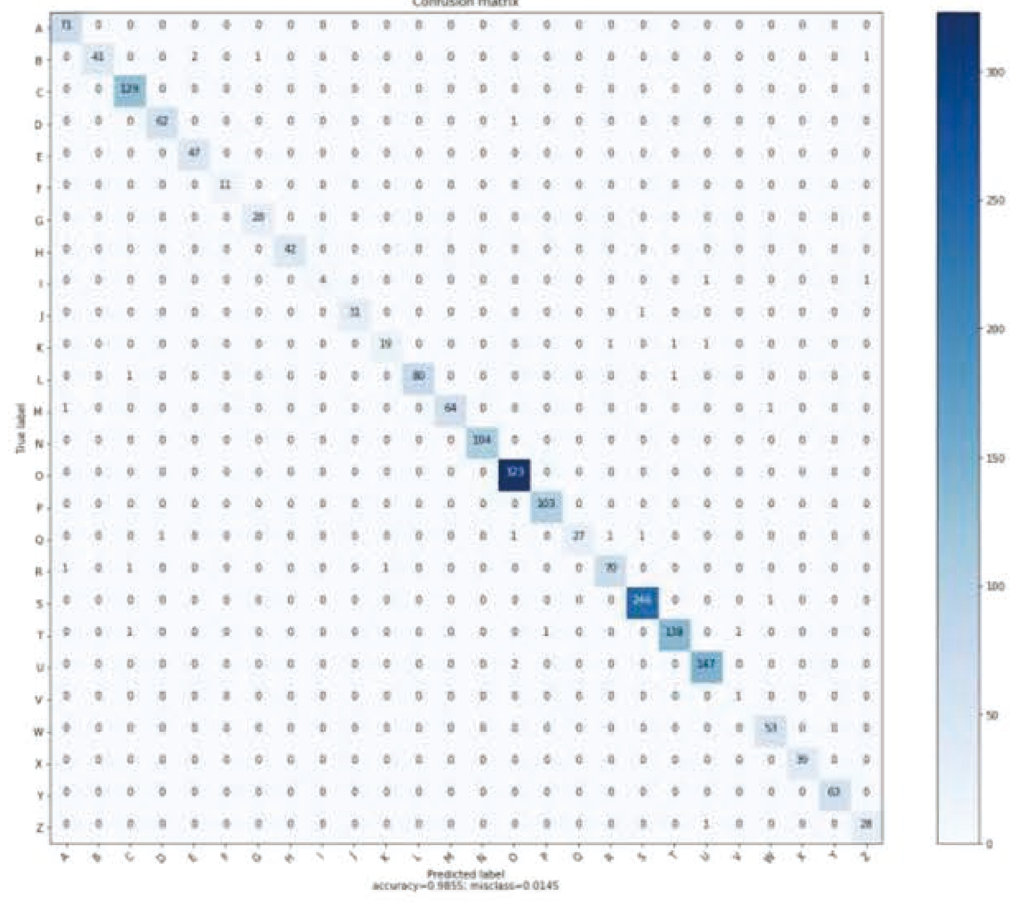
Misclassification= 0.0457582582582583

AlexNet CNN Model Evaluation

Model Performance in Training (50 epochs)



Model Performance on Leave-out Test Set (50 epochs)



Overall Accuracy= 0.9855

Misclassification= 0.0145

From the comparison of model accuracy and misclassification rate, we can see that both CNN models significantly outperform our baseline neural network. Although AlexNet gives the best performance, it consumes the greatest amount of resources and converges the slowest with the resources at our hand. Hence we have to test it on a relatively small test set to obtain results.

CNN Model Construction

Each CNN model consists of two parts: convolution and neural network.

In our Keras model construction, we implemented 3 hidden layers as convolution steps. Each of such step consists of two convolutional layers with ReLU activation, one pooling layer and one dropout layer. The model is then connected to a fully connected layer and finally using a softmax output. The final classification is given by simply finding the category with highest probability. The model summary is shown on the right with a structure graph below.

The AlexNet is constructed similarly but with 5 hidden layers before connecting to the fully connected one.

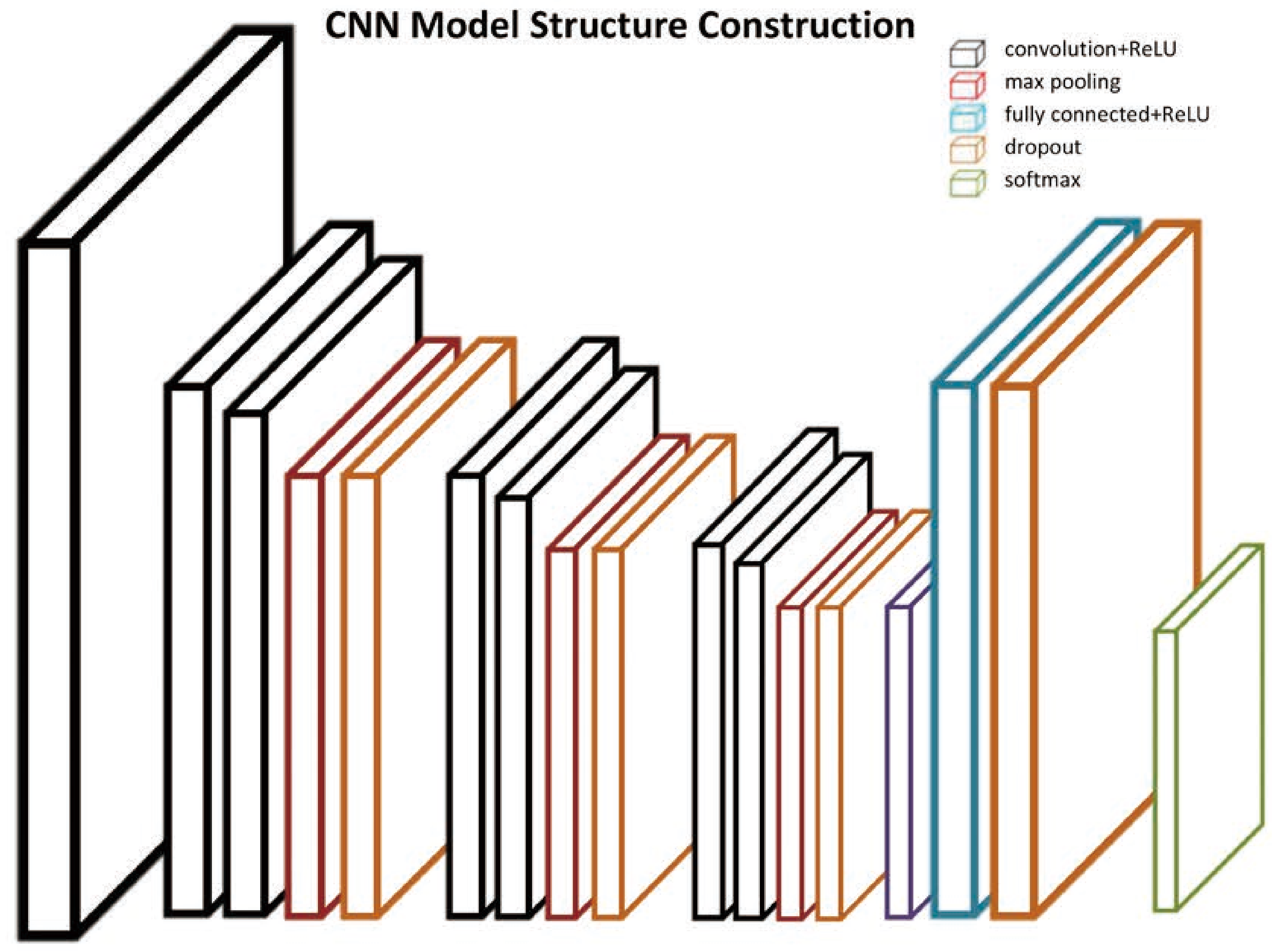
For each algorithm, we use cross-validation to obtain the best model. We also record model weights in the iteration with highest accuracy as parameters for the final model.

CNN Model Construction Summary

Layer (type)	Output Shape	Param #
conv2d_7 (Conv2D)	(None, 28, 28, 64)	640
conv2d_8 (Conv2D)	(None, 26, 26, 64)	36928
max_pooling2d_4 (MaxPooling2	(None, 13, 13, 64)	0
dropout_7 (Dropout)	(None, 13, 13, 64)	0
conv2d_9 (Conv2D)	(None, 13, 13, 64)	36928
conv2d_10 (Conv2D)	(None, 11, 11, 64)	36928
max_pooling2d_5 (MaxPooling2	(None, 5, 5, 64)	0
dropout_8 (Dropout)	(None, 5, 5, 64)	0
conv2d_11 (Conv2D)	(None, 5, 5, 64)	36928
conv2d_12 (Conv2D)	(None, 3, 3, 64)	36928
max_pooling2d_6 (MaxPooling2	(None, 1, 1, 64)	0
dropout_9 (Dropout)	(None, 1, 1, 64)	0
flatten_4 (Flatten)	(None, 64)	0
dense_7 (Dense)	(None, 512)	33280
dropout_10 (Dropout)	(None, 512)	0
dense_8 (Dense)	(None, 26)	13338

Total params: 231,898
Trainable params: 231,898
Non-trainable params: 0

CNN Model Structure Construction



Insight on CNN Model's Feature Extraction

Now that we have the complete model with a relatively acceptable performance, we would like to have some insight on how the convolutional layers are actually extracting features within a raw image input, and how the features change among layers.

We use a single image input, which is given below as demonstration and show the result after each convolutional and pooling layer.

As we can see, the earliest layers show recognizable traits of the original image but later layers more and more on abstract and local properties.

These insights are based on Keras CNN model for simplicity.