2019 Machine Learning II
Kannada Digits Recognition

Junchi Tian, M.S.

Jingshu Song, M.S.

Yiming Dong, M.S.

The George Washington University

1. Introduction

For this final project, we worked on the classic MNIST competition we're all familiar with, and we used a recently released dataset of Kannada digits. Kannada is the official and administrative language of the state of Karnataka in India with nearly 60 million speakers worldwide. The language has roughly 45 million native speakers and is written using the Kannada script. Distinct glyphs are used to represent the numerals 0-9 in the language that appear distinct from the modern Hindu-Arabic numerals in vogue in much of the world today. The following picture shows the figure of Kannada.

Figure 1

One of reasons we choose this dataset because it is not as simple as Arabic dataset, also not hard to implement deep learning algorithm by us. Another important reason is that this model will be helpful to Karnataka people and scholars who interest in Kannada culture. This report includes the following parts: description of the data set, description of the deep learning network, experimental setup, result and summary.

2. Description of the dataset

Our dataset file contains four csv files which are train, test, sample submission and Dig MNIST. The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine, in the Kannada script. Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive. The training data set has 785 columns. The first column, called label, is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image. The test data set is the same as the training set, except that it does not contain the label column. The dig MNIST contains an additional labeled set of characters that can be used to validate or test model results before submitting to the leaderboard. It is an

additional real-world handwritten dataset (with 10k images) which can serve as an out-of-domain test dataset.

3. Description of the deep learning network and training algorithm

For this project, we used Keras as framework and CNN as deep network. We choose Keras because of its simplicity and readability. Also, considering the size of dataset, we don't need to worry about the problem of speed. We choose CNN because it shows high performance than simple MLP with two dimensional images. The following two paragraphs provide the background knowledge of keras and CNN.

Keras is an open-source neural-network library written in Python. Keras contains numerous implementations of commonly used neural-network building blocks such as layers, objectives, activation functions, optimizers, and a host of tools to make working with image and text data easier to simplify the coding necessary for writing Deep Neural Network code.

In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery. A simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. The figure1 shows the architecture of the CNN.

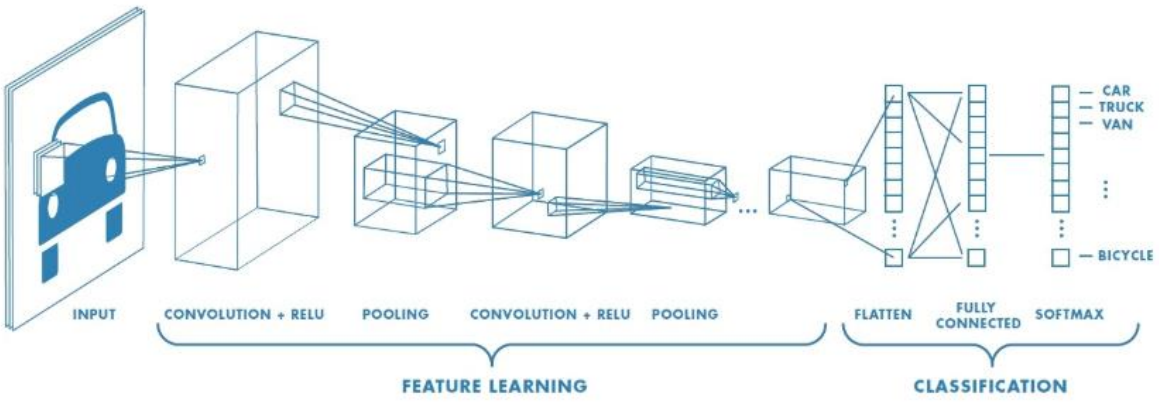

Figure 2

From the figure, we can notice that the input is the pixel matrix of the image and it will preprocess and transform to the convolution layer. Convolutional layer applies convolution operation on the input layer, passing the results to next layer. A convolution operation is basically computing a dot product between their weights and a small region they are connected

to in the input volume. After the convolution with Relu activation function, it will transform to the pooling layer which performs a down-sampling operation along the width in order to reduce dimensions. Then after some combination of previously defined architecture, flattening layer is used to flatten the input for fully connected layer. Next to these layers, the last layer is the output layer. The flatten layer will convert the 3-dimensions (height, width, depth) into a single long vector to feed it to the fully connected layer or Dense layer. It connects every neuron in one layer to every neuron in another layer. Fully Connected Layer and Output Layer Fully connected layers are the same hidden layers consisting of defined number of neurons connected with elements of another layer.

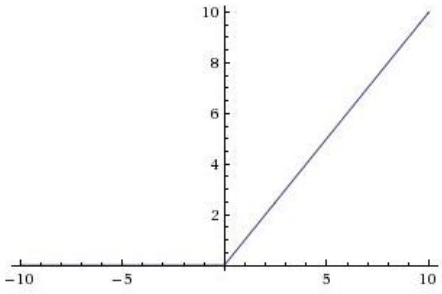

Figure 3

Relu is the most common activation function [A(x) = max (0, x)] and the Relu function is as shown above (figure 3). It gives an output x if x is positive and 0 otherwise. Compared with other function like Sigmoid, the reason why we choose Relu doesn't need to worry about the vanishing gradient and networks with Relu tend to show better convergence performance in practice. Besides that, Relu is less computationally expensive than tanh and sigmoid because it involves simpler mathematical operations.

The Softmax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. The output values are between the range [0,1] which is better for our model because we are able to avoid binary classification and accommodate as many classes or dimensions in our neural network model.

4. Experimental Setup

There are the following steps for the experimental step: data preprocessing, building and compiling the model, evaluating the model and visualizing the results.

4.1 Data Preprocessing

The data set used is Kannada MNIST dataset as we mentioned above. This dataset is a large database of handwritten digits (0 to 9). The first step is to load the dataset. Considering the file format of our dataset, it can be easily done through “pandas”. When we looked at the train data, the first column is the label, so it is selected as y train and the rest column is the x train. We reshaped the images with 28 * 28 *1 in put image. 28 *28 means the size of the image and 1 means the color of the image is white and black. After doing the necessary processing on the image, the label data, like y train and y test need to be converted into categorical formats for model.

After separating the train and test data, we worked on the image enhancement and we used Image Data Generator to increase the size of the train data in order to reduce the problem of overfitting. Figure 4 shows the steps we did for preprocess image. rotate the images’ angle (the value of the angle is 10), float the images with fraction of total width (20%), float images with fraction of total height (20%), random zoom image with 20%.

```
imagegen = ImageDataGenerator(
            rotation_range = 10,
            width_shift_range = 0.2,
            height_shift_range = 0.2,
            shear_range = 0.1,
            zoom_range = 0.2
             )
imagegen.fit(x_train)
```

Figure 4

4.2 Build and compile the model

After the data is processed, we need to build the architecture of the CNN model. Figure 5 shows the architecture and parameter of our model. The architecture followed here is 6 convolution layers followed by pooling layer, a fully connected layer and softmax layer respectively. For the convolutional layer, we can start with a convolutional layer with a kernel size (3,3) and a modest number of filters (64) followed by a max pooling layer. The way of padding is 'SAME', which means there is an additional layer that we can add to the border of an image in order to lose the information of border images, so we can keep the size of input and the output images the same and have a better accuracy. The proportion of the dropout is 20%, which means we randomly turn 20% neurons on and off to improve convergence. All layers used the Relu activation function [A(x) = max (0, x)]. It gives an output x if x is positive and 0 otherwise. Given that the problem is a multi-class classification task, we know that we will require an output layer with 10 nodes in order to predict the probability distribution of an image belonging to each of the 10 classes. This will also require the use of a softmax activation function.

```
model = Sequential()

#64 convolution filters used each of size 3x3
#image input size is 28 * 28 * 1
#the way of padding is "SAME"
#convolutional layer with relu activation function
#choose the best features via pooling
#randomly turn 20% neurons on and off to improve convergence
model.add(Conv2D(input_shape=(28, 28, 1),filters=64, kernel_size=(3, 3), padding='SAME', activation='relu'))
model.add(Conv2D(64, kernel_size=(3, 3), padding='SAME', activation='relu'))
model.add(BatchNormalization(momentum=0.5))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

#128 convolution filters used each of size 3x3
model.add(Conv2D(filters=128, kernel_size=(3, 3), padding='SAME', activation='relu'))
model.add(Conv2D(filters=128, kernel_size=(3, 3), padding='SAME', activation='relu'))
model.add(BatchNormalization(momentum=0.5))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

#256 convolution filters used each of size 3x3
model.add(Conv2D(filters=256, kernel_size=(3, 3), padding='SAME', activation='relu'))
model.add(Conv2D(filters=256, kernel_size=(3, 3), padding='SAME', activation='relu'))
model.add(BatchNormalization(momentum=0.5))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

#flatten since too many dimensions, we only want a classification output
model.add(Flatten())
#fully connected to get all relevant data
model.add(Dense(256, activation = "relu"))
model.add(BatchNormalization())
#output a softmax to squash the matrix into output probabilities
model.add(Dense(10, activation = "softmax"))
```

Figure 5

After the architecture of the model is defined, the model needs to be compiled. The function of the optimizer is Nadam, which is recommended to leave the parameters of this optimizer at their

default values. Here, we are using categorical crossentropy loss function as it is a multi-class classification problem. Another important thing needed to be mentioned is the learning rate reduction. In order to get a better result, we worked on reducing the learning rate and figure 6 shows the parameters. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

- monitor: quantity of validation accuracy need to be monitored.
- patience: number of epochs (3) with no improvement after which learning rate will be reduced.
- verbose: verbose =1 means that messages will be updated.
- factor: factor by which the learning rate will be reduced.
- min_lr: lower bound on the learning rate is 0.00001.

```
learning_rate_reduction = ReduceLROnPlateau(monitor='val_acc',
                                            patience=3,
                                            verbose=1,
                                            factor=0.5,
                                            min_lr=0.00001)
```

Figure 6

4.3 Train and evaluate the model

After the model is defined and compiled, the model needs to be trained with training data to be able to recognize the handwritten digit, so we will fit the model with x train and y train. One epoch means one forward and one backward pass of all the training samples. Batch size implies number of training samples in one forward or backward pass. We set the batch size as 64 and the epoch as 100. The training output is shown in figure 2. We used the accuracy as the matrix to evaluate the model because we are implementing a multi classification algorithm. Then the

accuracy can represent how many percentage Kannada digits can be successfully recognized. After compiling model, we used two line charts to present the loss and accuracy.

5. Result

Test accuracy 98.54% implies the model is trained well for prediction.

The figure 7 and figure 8 show the performance of our best model in the train data set. Form the pictures, we can notice that our accuracy is more than 98% and the loss is lower than 0.1 finally. With more epochs, the performance is better and the difference between train and validation is smaller.

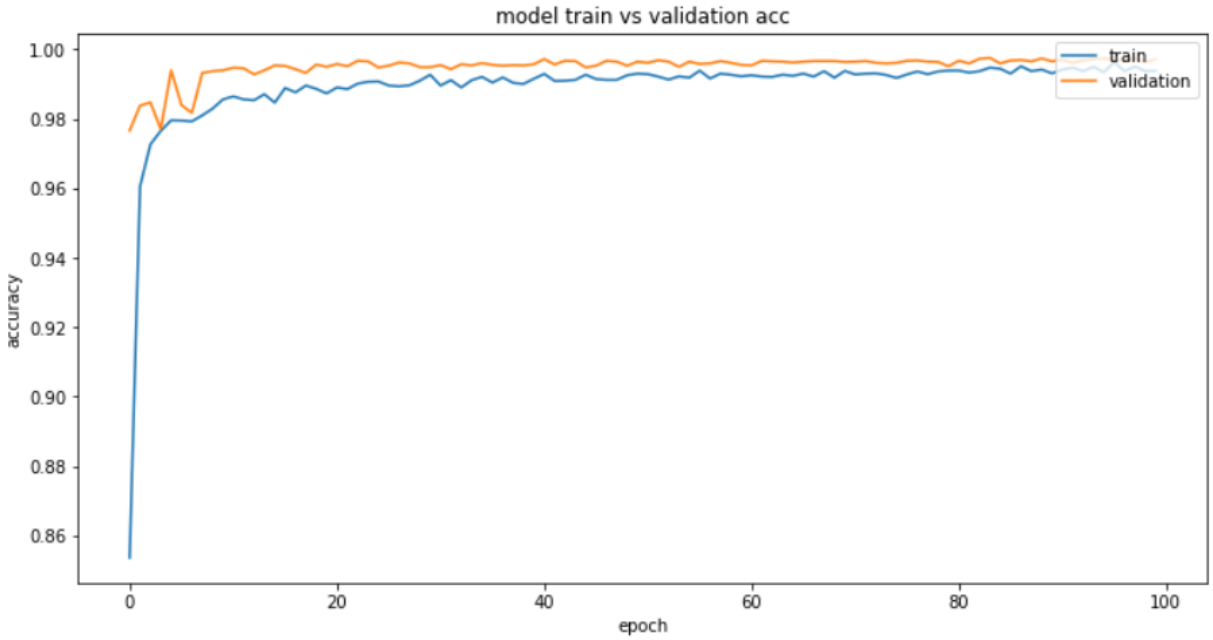

Figure 7

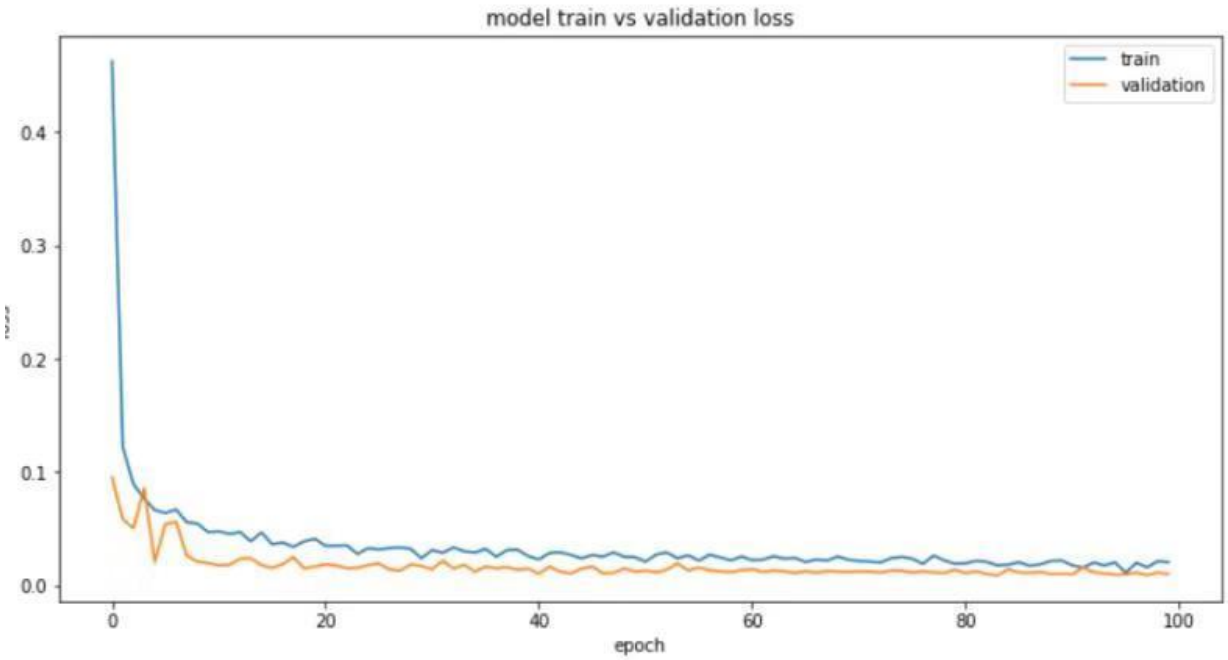

Figure 8

The figure 9 presents the final result of recognition. The diagonal line means the correct number we predict. The other number except 0 means cases the model didn't predict right. The number 0 means that the predicted number is the same as the actual one. For example, the number 13 means that there are 13 images of 1 was predicted as 0. In conclusion, there are highly similarity between 1 and 0, 6 and 9.

	0	1	2	3	4	5	6	7	8	9
0	1162	13	1	0	0	0	0	0	1	0
1	0	1218	0	0	0	0	0	0	0	0
2	1	0	1223	0	0	0	0	0	0	0
3	0	1	1	1177	0	0	0	5	0	0
4	0	0	0	1	1218	2	0	0	0	0
5	0	0	0	2	0	1186	0	0	0	0
6	0	0	0	0	0	0	1156	3	0	10
7	0	1	0	2	0	0	4	1210	0	2
8	0	0	0	0	0	0	0	0	1186	0
9	0	0	0	0	0	0	3	0	0	1211

Figure 9

The figure 10 shows the part results that the model did the wrong prediction. Take the first picture as the example. The actual number should be 4, but we predicted as 9.

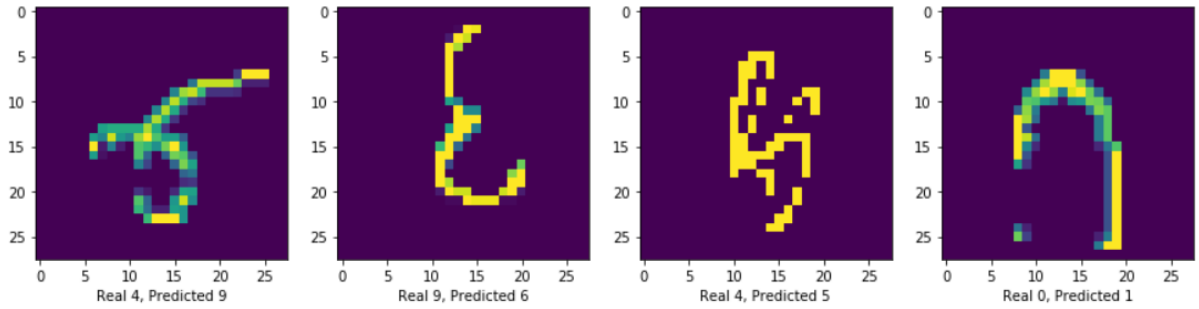

Figure 10

Then, we test our model in the test dataset in the Kaggle competition public board, we got the rank 318/1040 (around top 30%) and the 98.54% accuracy (shown on Figure 11)

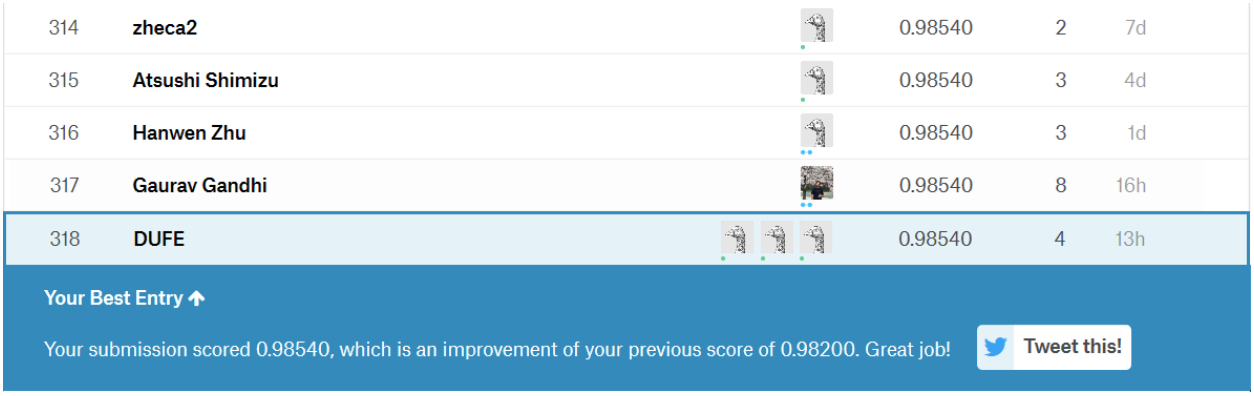

Figure 11

6. Summary and Conclusion

In summary, we used Kannada handwritten digits dataset to recognize Kannada numeric from 0 to 9, so this is the multi-classification problem and we used Keras as framework and CNN as deep network. We load the data and use image data generator to preprocess image, after that, we build 6 convolution layers with Relu activation function followed by pooling layer, a fully connected layer and softmax layer respectively, and we compile the model. Finally, we visualized the result by line charts and shows the score of test loss and test accuracy.

There are three things we learned from this project. The first one is preprocess image by Image data generator, which can rotate, float and reshape images to increase the size of the image and reduce the problem of overfitting. The second thing we learned is learning rate reduce. Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This

callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced. The last thing we learned is how to deal with the problem of overfitting.

In terms of where can be improved, we could do a more complicated model and select more appropriate parameters to improve accuracy. Besides that, for some special situation, I think we should deal with them differently.

Finally, we got the rank 318/1040 in the Kaggle competition. Form this project, we have a better understanding of deep learning theory and python code. We will apply what learned from this class into the future work.

7. Reference

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.

Prabhu, Vinay Uday. "Kannada-MNIST: A new handwritten digits dataset for the Kannada language." arXiv preprint

Vinay Uday's github: https://github.com/vinayprabhu/Kannada_MNIST