Multi-label Classification of Image Data

Zebang Yu , Junjie Yang , Maoyu Wang

***Abstract*—The deep neural network is very useful in Multi-label classification of Image Data. In this project, we employ the convolutional neural network to explore its capability of identifying the digits on a given image using a modified version of the MNIST dataset.**

I. Introduction

In this report, we implemented the deep neural network model with TensorFlow and Keras, and trained it with images of digits. We studies the performance of CNN model on this Multi-label classification task and tunes hyper-parameters to achieve best accuracy. The performance of our model is promising.

II. Model

We employ the convolutional neural network as our model, which take input shape (64,64,1) and produce five outputs. The model involves six convolution layers and three pooling layers. we divide them into three parts. For each part, it contains two convolution layers and one pooling layer. For every convolutional layer, the size of the kernel is 3×3, and we exert padding on them. And we choose an increasing filter size [64,128,256] for each of two convolution layers. The pooling strategy we choose is the max function. The max-pooling will slightly promote the transformation invariance and decrease the number of variables to avoid overfitting. And we avoid using large pool size since it will diminished some important features. Moreover, we apply dropout to each part to reduce the problem of overfitting. Finally, we add 5 hidden layers to get multi-output.

We plot a graph of validation and train accuracy of our best performing model as a function of training epochs. Within five epochs, the train and validation accuracy has approached its maximum very closely. The model's final accuracy on Kaggle is 99.404%.

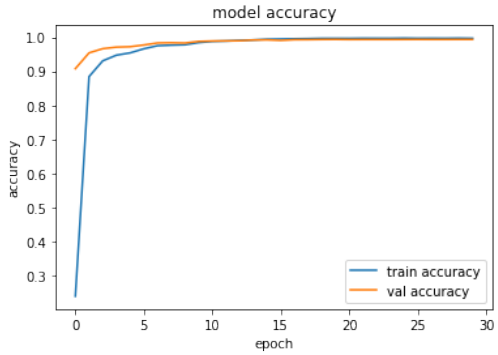

Fig. 1: Train and validation accuracy

This model involves a significant number of hyper-parameters. The model architecture, optimizer, and their parameters are under our consideration. The details of the model architecture are given in the table in the appendix. The optimizer we pick is the RMSprop algorithm. It maintains a moving average of the square of gradients, and divide the gradient by the root of this average. And we find it performs effectively in the test. we also make a comparison with Adam. However, the validation accuracy will be slightly lower than that of RMSprop. We think it might be because Adam does not converge to an optimal solution in this task. Thus we still choose RMSprop as our optimizer.

The rate of decay is set to 0.8 with a default learning rate of 0.001, which we find gives us the optimal rate of convergence. If the rate of decay is small, convergence will be too early. And we find that the model gives us the best performance when the batch size is 64. With smaller batch sizes, the training time increases, and the performance declines.

	batch size=16	batch size=32	batch size=64	batch size=128
decay rate=0.8	0.9829	0.9909	0.9943	0.9931
decay rate=0.2	0.9556	0.9735	0.9872	0.9904

Fig. 2: choice of hyper-parameters

III. Data processing

In the beginning, we refer to VGG16 to process our data. We transform the image shape from (64,64,1) to (64,64,3). However, after we train the model, there is no difference between BGR data and raw data. We speculate the BGR

data might be very useful to other tasks such as vehicle classification that needs color value. For hand-writing digits classification, which is related to digits' shape, the color value seems trivial. We also try other ways to deal with data. For example, we divide data by 255 to put them in the interval [0,1]. But the result is not so desirable. And we try to transform all pixels with value to 1. However, it does not perform effectively since this transformation will make some digits like 8 be recognized as 1. Therefore, we decided to use data without process.

IV. Conclusion

At first, we tried to use a pre-existed model called VGG16, but the performance is not desirable. Thus, we chose to implement the model from scratch by ourselves instead of using pre-existed models. On our first attempt, we used the One-hot encoding API direct from Keras. After applying the technique, we are getting a $m \times 5 \times 11$ list for the train labels, where m is the number of instances we have. Then, we tried to fit the model using the train dataset and encoded the train label. However, the result was quite lower than the baseline. As a result, we implement our encoding function, which turns the train label of $m \times 5$ to $5 \times m \times 11$. Thus, we increase our test accuracy to 97.3%.

After dealing with the label, we tried to improve the performance by modifying the layers for the model. In the beginning, the model we implemented had an over-fitting issue, where we achieve 100% for train accuracy and 97.3% for validation accuracy. To solve that, we add Dropout throughout the model, which will help us solve the overfitting problem. We tried to set the probability to 50%, this solve the overfitting problem, and also we got a test accuracy of 98.6%. We also tried other probabilities. As a result, we achieve the highest accuracy when we set the probabilities to 25%, with the accuracy be 99.3%.

V. Discussion

However, this model can be modified to achieve higher accuracy in the future. In this model, even though we take some pre-existed models and papers as a reference, for example, VGG and the article *Deep Residual Learning for Image Recognition*[1], we chose the parameters based on the experiments we performed. We didn't in-depth understand their model structures due to the time-limit. We believe those research results could give us a better understanding of CNN and a guide to build our network. Moreover, a different approach to train the model can be made. We trained our model without using Data Augmentation. We observe the hand-writing images and we find that some digits have rotated from 5-10 degrees. if we implement data augmentation, we speculate it will bring better generalization performance than a simple dropout.

Furthermore, another technique will increase accuracy. In this project, we only trained a single CNN model. In the future, several models can be trained, and the prediction can be made by considering the predictions from all the models to achieve a better result.

Statement of Contributions

Zebang Yu: Report writing. Plotting.

Junjie Yang: Data processing. Model Implementing. Report Writing.

Maoyu Wang: Tuning parameters. Report writing.

Reference

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," arXiv [cs.CV], 2015.

APPENDIX A
MODEL ARCHITECTURE

Operation Layer		Number of Filters	Size of Each Filter	Stride Value	Padding Value	Size of Output Image
Input image		-	-	-	-	$64 \times 64 \times 1$
Convolution Layer (two times)	Convolution ReLU	64	3×3	1×1	1×1	$64 \times 64 \times 64$
Pooling Layer	Max pooling	1	2×2	2×2	0	$32 \times 32 \times 64$
Convolution Layer (two times)	Convolution ReLU	128	3×3	1×1	1×1	$32 \times 32 \times 128$
Pooling Layer	Max pooling	1	2×2	2×2	0	$16 \times 16 \times 128$
Convolution Layer	Convolution ReLU	256	3×3	1×1	1×1	$16 \times 16 \times 256$
Pooling Layer	Max pooling	1	2×2	2×2	0	$8 \times 8 \times 128$
Inner Product Layer	Fully connected	-	-	-	-	11