Multi-label Classification of Image Data

Mike Gao Talise Wang Zhangyuan Nie

Introduction

Convolutional Neural Networks (CNNs) have been widely applied in image classification tasks.[1] Compared to other classification methods such as SVM, CNN performs well in extracting the features of an image, hence gives you more optimal results when classifying the image dataset. In this project, we choose a modified version of the MNIST dataset of 56000 images of size 64x64 where each of them contains 1-5 digits as the object that needs to be classified and used two types of CNN architecture as the base of our implementation, namely, SimpleNet[2] and VGG-16[3]. A CNN model is built by different types of layers, such as the convolutional layers, batch normalization layers, pooling layers, dropout layers and gaussian-noise layers. Convolutional layers convolve the input and pass its result to the next layer. Batch normalization layers improve the speed of learning, Pooling layers reduce the dimensions of the data by combining the outputs of neuron clusters at one layer into a single neuron in the next layer. Both dropout layers and gaussian-noise layers mitigate overfitting. To get a better understanding of how good this model prediction is along with the epoch numbers, we plotted out the loss and accuracy performance result of both the training dataset and test dataset along with the increasing numbers of epochs after the training of the models has been completed. Based on these plots, we were able to see how the accuracy changed with each epoch. At last, we used ensembles from our best performing model (AnotherNet) to get higher accuracy.

Results

> Victorious warriors win first and then go to war, while defeated warriors go to war first and then seek to win
> — *Sun Tzu, The Art of War*

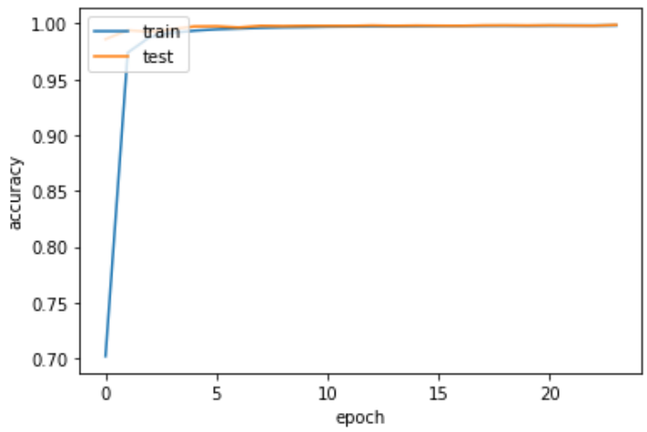


Fig. 1 - Mean Accuracy of the best model over all 5 digits

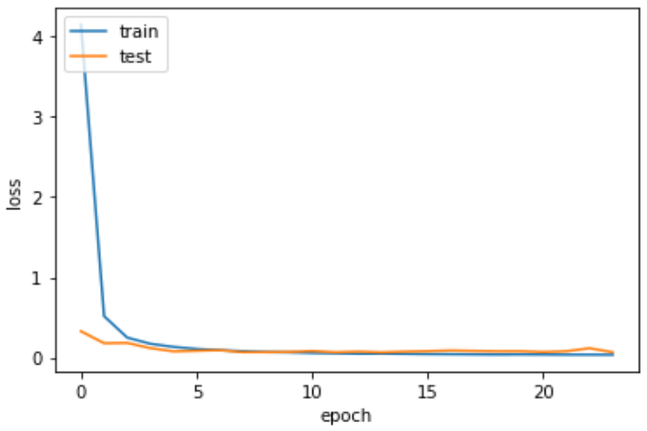


Fig. 2 - Cross-Entropy Loss of the best model

The overall trend of accuracy is maintained across different digits. The training accuracy and the validation accuracy increase consistently as the number of epochs increases for our best model. The cross-entropy loss decreases as the number of epochs increases.

Dataset Analysis & Data Augmentation

> In the midst of chaos, there is also opportunity
> — *Sun Tzu, The Art of War*

First, we noticed that the source image contains sharp edges. In order to obtain better results, we used linear interpolation in *cv2.resize* to enlarge the image and then applied denoising with a 5x5 Gaussian Kernel to reduce the sharp edges [Fig A1 & A2]. We have also used Keras' ImageGenerator to increase the diversity of our training set by applying random transformations to the images [Fig A3] such as shift, zoom and rotation. We believe this step could also help to prevent overfitting.

Network Design (SimpleNet and AnotherNet)

> If you know the enemy and know yourself, you need not fear the result of a hundred battles
> — *Sun Tzu, The Art of War*

Before starting to build the neural network, we have decided to do an extensive research and survey to narrow down the generic neural network architecture to use. Ultimately, we have decided to use SimpleNet[2] and VGG-16[3] as a starting point of our model. We chose them for a few reasons, namely, there is extensive research published on both models, both have been adapted and used for the original MNIST dataset before (and achieved decent results on the original MNIST dataset), and both are not so deep that it becomes untrainable. We also do not want an overly complicated model that's inefficient to train. To prevent overfitting and improve the model's ability to generalize, we added batch normalization between each convolution block and used a dropout layer with a rate of 0.2. We have used the *GaussianNoise* layers to further emphasize the features extracted through the *Conv2D* layers, and through experimentation we believe the *GaussianNoise* layers actually improved our performance significantly on the test set, allowing us to reach the 99% accuracy with a single model. We implemented early stopping when the model has converged to a stable value and removed the redundant kernels by using the Layer-Wise Polishment[4]. We have decided to choose 100 epochs as our limit, however, our model always converges at around 25 (AnotherNet) and 35 (SimpleNet). With regards to batch size, we chose the largest batch size for our GPU memory, which in our case is 128.

Ensemble

> To know your enemy, you must become your enemy
> — *Sun Tzu, The Art of War*

We implemented the ensemble technique to seek higher accuracy for classification. The benefit through the use of ensembles is demonstrated in many Kaggle competitions and academic competitions. Our highest performance ensemble is using 5 model weights for AnotherNet, a model we designed based on VGG-16. Each individual model can reach around 99.3% to 99.4% accuracy on the public portion of the test set, and after using ensemble, we can easily reach 99.6 - 99.7% accuracy.

Conclusion

> There is no instance of a nation benefitting from a prolonged warfare
> — *Sun Tzu, The Art of War*

We believe the major difficulty in this project lies in optimizing the convolutional neural network architecture. It is very hard to compare between many models, or use grid search to iterate over a variety of different hyperparameters because each training takes a significant amount of time. Nevertheless, we have obtained the near perfect accuracy of 99.714%. We believe it's pretty difficult to get a better result since there exists some ambiguous images which even humans can not figure out which digits are shown, plus, there is always the possibility of a labeling error. For future optimizing direction, we can focus on the relationship between the dropout rate and the accuracy. if the dropout rate is too small, it will cause the overfitting, but if the rate of dropout is too big we will lose the data, which will also decrease the accuracy. We can draw a plot on the data accuracy along with the increasing dropout rate starting from 0 to choose the best rate for the dropout layers.

Contributions

Mike Gao implemented the base model inspired by VGG, as well as the SimpleNet model initially published by Hansanpour et al. Talise Wang optimized the base model and transformed the VGG model to the new model, AnotherNet. Zhangyuan Nie generated the graphs and added many useful utility functions. We all contributed to the report equally.

Appendix A - Dataset Analysis

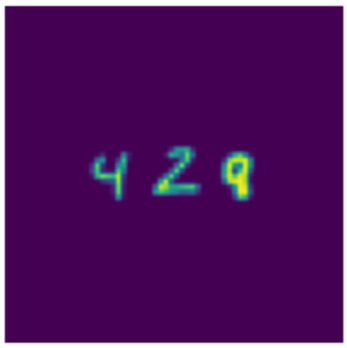


Fig. A1 - Before Image Processing

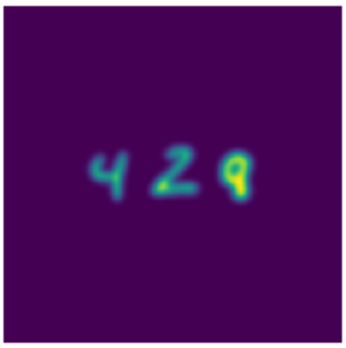


Fig. A2 - After Image Processing

Fig. A3 - After Image Generator

Appendix B - Model Architecture

Network	Batch size	Epoch (preset)	Activation Func	Optimizer	Dropout Rate
AnotherNet	128	100	softmax	adam	0.2 → 0.25 → 0.5

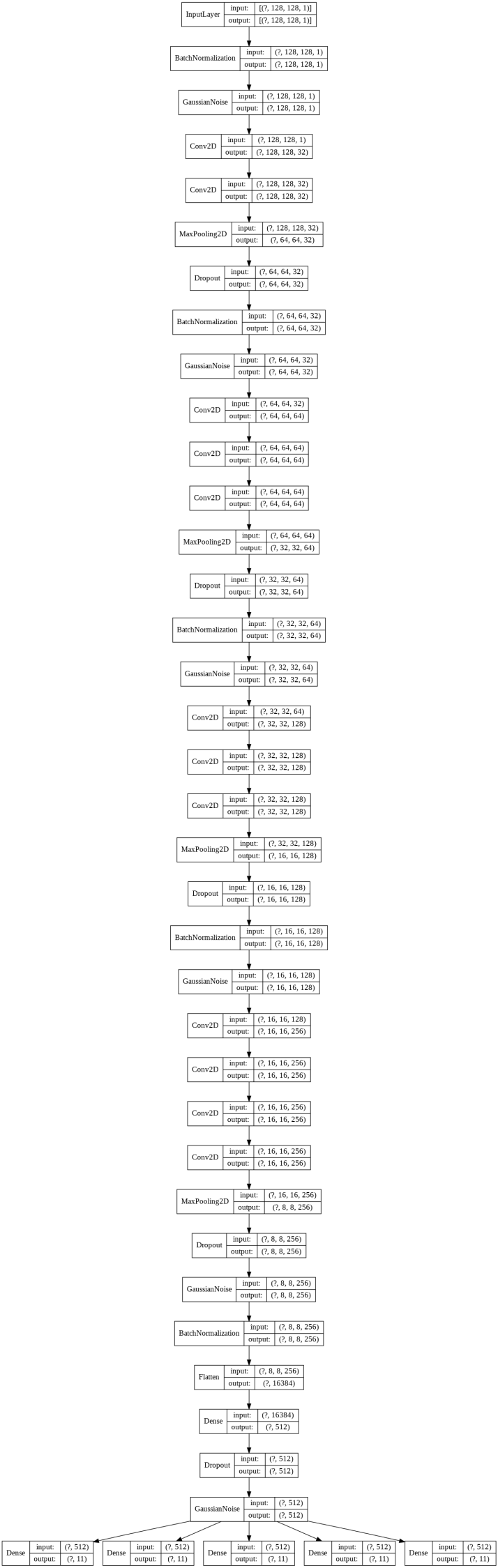
References

[1] T. Sinha, B. Verma, and A. Haidar 2017. Optimization of convolutional neural network parameters for image classification. In 2017 IEEE Symposium Series on Computational Intelligence (SSCI) (pp. 1-7). https://doi.org/10.1109/SSCI.2017.8285338

[2] Seyyed Hossein HasanPour and Mohammad Rouhani and Mohsen Fayyaz and Mohammad Sabokrou (2016). Lets keep it simple, Using simple architectures to outperform deeper and more complex architectures, abs/1608.06037.

[3] Karen Simonyan, & Andrew Zisserman. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition, abs/1409.1556.

[4] Chih-Ting Liu and Yi-Heng Wu and Yu-Sheng Lin and Shao-Yi Chien (2017). A Kernel Redundancy Removing Policy for Convolutional Neural Network, abs/1705.10748.


InputLayer
input: [(?, 128, 128, 1)]
output: [(?, 128, 128, 1)]
BatchNormalization
input: (?, 128, 128, 1)
output: (?, 128, 128, 1)
GaussianNoise
input: (?, 128, 128, 1)
output: (?, 128, 128, 1)
Conv2D
input: (?, 128, 128, 1)
output: (?, 128, 128, 32)
Conv2D
input: (?, 128, 128, 32)
output: (?, 128, 128, 32)
MaxPooling2D
input: (?, 128, 128, 32)
output: (?, 64, 64, 32)
Dropout
input: (?, 64, 64, 32)
output: (?, 64, 64, 32)
BatchNormalization
input: (?, 64, 64, 32)
output: (?, 64, 64, 32)
GaussianNoise
input: (?, 64, 64, 32)
output: (?, 64, 64, 32)
Conv2D
input: (?, 64, 64, 32)
output: (?, 64, 64, 64)
Conv2D
input: (?, 64, 64, 64)
output: (?, 64, 64, 64)
Conv2D
input: (?, 64, 64, 64)
output: (?, 64, 64, 64)
MaxPooling2D
input: (?, 64, 64, 64)
output: (?, 32, 32, 64)
Dropout
input: (?, 32, 32, 64)
output: (?, 32, 32, 64)
BatchNormalization
input: (?, 32, 32, 64)
output: (?, 32, 32, 64)
GaussianNoise
input: (?, 32, 32, 64)
output: (?, 32, 32, 64)
Conv2D
input: (?, 32, 32, 64)
output: (?, 32, 32, 128)
Conv2D
input: (?, 32, 32, 128)
output: (?, 32, 32, 128)
Conv2D
input: (?, 32, 32, 128)
output: (?, 32, 32, 128)
MaxPooling2D
input: (?, 32, 32, 128)
output: (?, 16, 16, 128)
Dropout
input: (?, 16, 16, 128)
output: (?, 16, 16, 128)
BatchNormalization
input: (?, 16, 16, 128)
output: (?, 16, 16, 128)
GaussianNoise
input: (?, 16, 16, 128)
output: (?, 16, 16, 128)
Conv2D
input: (?, 16, 16, 128)
output: (?, 16, 16, 256)
Conv2D
input: (?, 16, 16, 256)
output: (?, 16, 16, 256)
Conv2D
input: (?, 16, 16, 256)
output: (?, 16, 16, 256)
Conv2D
input: (?, 16, 16, 256)
output: (?, 16, 16, 256)
MaxPooling2D
input: (?, 16, 16, 256)
output: (?, 8, 8, 256)
Dropout
input: (?, 8, 8, 256)
output: (?, 8, 8, 256)
GaussianNoise
input: (?, 8, 8, 256)
output: (?, 8, 8, 256)
BatchNormalization
input: (?, 8, 8, 256)
output: (?, 8, 8, 256)
Flatten
input: (?, 8, 8, 256)
output: (?, 16384)
Dense
input: (?, 16384)
output: (?, 512)
Dropout
input: (?, 512)
output: (?, 512)
GaussianNoise
input: (?, 512)
output: (?, 512)
Dense
input: (?, 512)
output: (?, 11)
Dense
input: (?, 512)
output: (?, 11)
Dense
input: (?, 512)
output: (?, 11)
Dense
input: (?, 512)
output: (?, 11)
Dense
input: (?, 512)
output: (?, 11)