Multi-label Classification of Image Data (Group 85)

Vincent Huang(260761859) Zilong Wang(260823366) Yuelin Liu(260844113)

Nov 20, 2020

1 Introduction

This project serves as an exploration on the utilization of neural networks on multi-label classification on image data. The dataset is a set of 64 * 64 images with no more than 5 handwritten digits which can be fairly accurately splitted into individual digits, so we can process each digit separately and put their predicted labels together at the end. In this way we turned this problem into a classical single digit classification problem. For preprocessing, we used OpenCV to achieve the isolation of the images. Then we implemented and trained a fully connected neural network (FCNN) for single-digit recognition. Our goal is to first split the images accurately, train our model to differentiate the individual digits as correctly as possible, and then integrate them together to obtain the final answer. Using 10-fold cross validation and 5 layers, the predictive accuracy of single digits is 99.89% and the whole accuracy is 99.80%

2 Datasets and Preprocessing

The dataset provided is a modified MNIST dataset where each image is a combination of no more than five single digits. For single-digit recognition, there are extensive resources on the Internet, which is able to exhibit a high performance. We decided to put more weight on data preprocessing where some mechanism could be adopted to split the image into lower-resolution single digits, then the single digits are fed into the neural network.

For splitting the image, we decided to use OpenCV for its simplicity and convenience when processing the dataset. For each image, the goal is to detect how many digits there are and their locations. First, a binary filter was applied to each image to filter out the unwanted noise by turning the pixels into 0, and the areas of interest into 255. Then we have a contrasted border between the digits and the background for extraction. We utilized the implemented border detection algorithm (Suzuki & Be, 1985) supported in OpenCV to find the external contours by the digits. Note that we wanted to ignore the hierarchical relationship between the borders because only the outermost layer is needed to locate a written digit in the image. For each contours detected in the image, we then add paddings to make each contour into 28 * 28 numpy array. Note that this algorithm does not necessarily work all the time because of how the digits are organized in one image. A mechanism is developed to evaluate the performance of the splitting algorithm, where we check the number of contours detected against the number of digits obtained from the label. If they are equal, count them as valid and add the single digits and labels into the constructed new training set. If not, remove this image from the training set completely. In terms of the performance, filtering threshold, contour-finding algorithm are all important factors. After trying out on different sets of parameters, the optimal set is filtering with 0 threshold and then only extracting the outermost layers. This combination yields an extremely high accuracy of splitting with only 47 invalid splits out of 56000 entries in the training set. For the test set, a similar pattern is followed where we split the image into single digits groups, and then predict each digit within the group. Finally, if the number of predictions are less than 5, pad the length into 5 with number 10s. One drawback of this technique is that it provides a performance bottleneck of roughly 99.92% accuracy.

3 Results

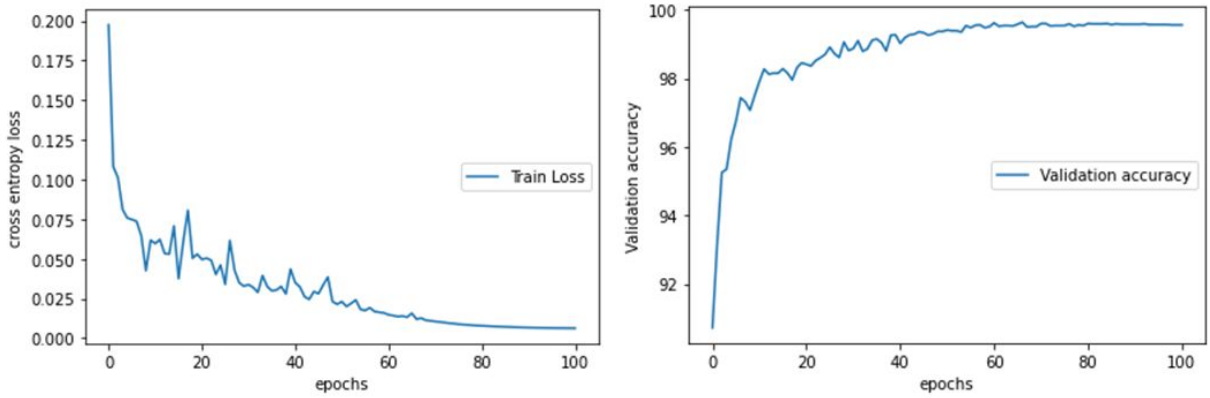
- **Model with justification of choice of hyper-parameters**

We found some optimal combinations of hyperparameters with many trials. For our best model, we chose to use 5 layers and 32 hidden units in each layer. We selected ReLU over tanh and sigmoid function as activation function. Cross entropy is used to calculate the gradient. With 92 epochs (92 * 165728 digits = 15,246,976 digits for training), we achieved the highest accuracy on the validation set which is 99.67%, because while the loss kept decreasing, the validation accuracy started to decrease as well after 92 epochs.

To optimize, we used the Adam optimizer, because it combines momentum and RMSprop optimization and it performs better than the other two after we experimented. The parameter β for momentum is selected to be 0.9 as it is typical. We

adopted a learning rate of 0.005 because it gave us a relatively high accuracy and a fair training time. It has also been scheduled to decay at a rate of 0.0009. In order to overcome overfitting, we used L2 regularization with λ being 0.01, and it performed well. We used 1000 data points for one batch, and in order to regularize the batch and fit it into the neural network, we selected σ^2 and μ to be 1 and 1. Most importantly, the hidden units were changed to 512, which will fit the training more accurately. The **Appendix** shows our best combination of parameters.

- **Validation VS. Train performance**



During the training interactions, the loss decreased from 3.6 to 0.006 at the end of the training, while the validation accuracy at the end of each epoch first increased significantly, and stayed stably around 99.87% in the later epochs. Since we used Adam optimizer, the fluctuation is expected, but the gradient is centered around optimal. Furthermore, we have tried using 4 layers and with different hidden units, and found the lower layer will probably lead to more bias. The hidden units will not have a significant effect on the final accuracy but it requires much more time to train.

4 Discussion and Conclusion

The performance of FCNN proved to be competitive enough, although maybe not as competitive as a fine-tuned CNN model. There are some disadvantages of FCNN, for example, the training time can be long and there are many hyperparameters which makes tuning tedious. In the future, we will explore using CNN models for image classification.

Besides, we found the single digits were reused in different pictures (nearly 90 % of digits), which means we can find the same image of digit in the training set and get its correct label. As such, we created a simple KNN method in check.ipynb to check out results generated by CNN.

5 Statement of Contributions

Zilong helped with model analysis, train-validation comparison and contributed to the report.

Vincent implemented the method to divide digits and analyzed its errors.

Yuelin implemented the CNN method for single digits and analyzed the testing labels.

Appendix

Model architecture	
# layer	**5**
# hidden units	**512**

Adam Optimizer	
Learning rate	**0.005**
λ (L2 regularization)	**0.01**
β (momentum)	**0.9**
Decay_rate (for learning rate)	**0.0009**
mini_batch_size	**1000**
σ2, μ	**1, 1**

References

1. Suzuki, S., & Be, K. (1985). Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing, 30*(1), 32-46. doi:10.1016/0734-189x(85)90016-7
2. Kukreja, Y. (2019, January 03). Recognizing handwritten digits in real life images using CNN. Retrieved December 12, 2020, from https://medium.com/@yash.kukreja.98/recognizing-handwritten-digits-in-real-life-images-using-cnn-3b48a9ae5e3
3. Realize mnist handwritten digit recognition with python's numpy Realize mnist handwritten digit recognition with python's numpy - Programmer Sought