COMP 551 Project 3

Ian Tsai, Zhiying Tan Xintong Li

Fall 2020

INTRODUCTION

In this project, our goal is to design a machine learning architecture to compete in a multi-label MNIST competition. Our model will be classifying a modified MNIST dataset for digits recognition, which will be classified using CNN implemented with PyTorch. We also made efforts to achieve our best model by using OpenCV to localize digits or rotate image for data augmentation. Eventually, we were able to achieve 99.5% on the testing dataset on Kaggle. One related work we found on line quite interesting is that classifying MNIST with YOLOV3.

DATA PREPROCESSING & DATA AUGMENTATION

Our first approach to process the multi digits figure is applying KNN with 5 clusters model to split out the digits in the images. However, this approach only correctly separated 85.3% of the cases. Our second approach is to crop each digit out for recognition using OpenCV to localize contours of digits. However, we found out that finding digits with contours would often cut 2 digits into one image or no digit in one image. Thus, we solved this by setting a threshold to the width of each cut and separating the digits by the threshold in some special cases. With some defective blurry cases that CV2 cannot detect, we change our plan to separate each digit by columns with zero black pixels. We also found out that after we properly cropped out all the digits and fill out the image with black pixels to resize all images to 16*16. As for the image with less than 5 digits, we will separate all the digits first, then append a 16*16 totally black images to our dataset. As a result, we got 280,000 subimages (single digit)for training and 70,000 subimages for validation. We then normalized the dataset by coverting the whole dataset to 0 mean and 1 standard deviation.

To improve the performance of our model, we introduced two ways to develop an extended training dataset. As we scouted through the cases that we got wrong on the training dataset, we first figured there were several repetitive mistakes. So, we improved the frequency of theses edge cases by 5 times, so our model could adapt these cases better. Secondly, we noticed that many digits where angled, which might eventually confuse our model so we randomly chose 2000 images from the training dataset and rotated these images by a randomly angle between -20 and 20 degrees and shift in vertical and horizontal direction by a randomly 1 or -1 number of position. Then augmented the transformed images to our training dataset. Eventually, we found out that retraining the mistaken edge cases and angle transformation have improved our validation accuracy from 99.4% to 99.5%.

RESULTS

Our **model design** uses Pytorch to the combine CNN and multi-layer perceptron. At the beginning, we chose to train the digits unseparated dataset, the accuracy was less than 90% on average and the results were very unstable. Training on the digits separated data could bring the overall accuracy to more than 99%. This is a multi-classification problem, so, we chose the cross-entropy loss. In each training epoch, we calculated the cross entropy-loss with l2 loss of the whole batch and apply the automatic differentiation method to backpropogate the gradient. We then update the weight of model according to the gradient. The input data will be trained with 4 trials. For each trial, we have applied the early stopping technique to avoid the problem of loss bounce back. After training each trial, we adjust the optimizer by applying the learning rate annealing technique to decrease the learning rate and the momentum. After completing the training process, we can apply the trained model to predict the validation dataset. Lastly, we merged the single labels together to obtain the overall accuracy. With the expensive training time, we made use of Google Colab's GPU service by importing pycuda to train our model with Tesla T4.

To find out the **best model architecture**, the very first thing is to find the best number of layers, number of channels and number of neurons. For each convolutional layer, we added padding and the output will be applied to maxpool with a size 2 square window. First, we have to decide the padding. By comparing the performance of padding=1 and padding=2, we found out that they have similar training and testing accuracy. To make the model much simpler, we ended up choosing padding=1. The dropout process also included in both CNN layers and fully connected layers to avoid overfitting problem. Between each layer, we applied the batch normalization technique to have a much stable training process. For fully connected layers, we first applied the linear transformation to the input then output the result to relu function. To hold the experiment for models with different height and width, first, we tested the accuracy and cross-entropy loss by training these models with 80% of the data that are randomly selected from the original dataset and the remaining 20% of the data are used for testing. The following 5 models (Table 1) have the highest accuracy in general. First, we varied the number of convolutional layers (1,2 and 3) and fixed the number of dense layers(4 layers), and we found out that 3 layers (training accuracy=99.99% and testing accuracy=97.93%)overfitted and 1 layer (training accuracy=99.91% and testing accuracy=98.58%) underfitted. Thus, we decided to use 2 convolutional layers. By comparing the performance of 6 channels and 10 channels in the first layer as well as 16 and 20 channel in the second layer. We found out that the performance are quite close to each other, so we chose the simplest one. As for the number of fully connected layers, by flattening the output of convolution layer, we ended up with more than 200 neurons and we only kept 11 neurons in the final output, so we chose the dense layers to be between n3 (Table 1)and n4 (Table 1)(by adding a dense layer with same number of input and output neurons). It shows that model N have a much higher training and testing accuracy. The last thing to consider is the number of neurons in the dense layers. By varying the number of neurons between 383 in total and 339 in total, our experiment shows that the one with larger number of neurons can perform much better. Finally, the best model is given by 2 convolutional layers and 4 fully connected layers(See Figure 1).

By comparing the performance of **SGD, Adam and RMSprop**, we found out that Adam has the best performance(highest accuracy). In the comparison experiment, we found that training with SGD is much slower than Adam and RMSprop. Adam has the lowest loss and it is more stable than the other two optimizers so we chose Adam optimizer in the model training process. (Table 2 of comparing different optimizer) Among all sorts of hyperaparameters, learning rate, weight decay and momentum are very essential. In this part, we used simple grid search to find the best pairs of hyper-param for our best model. By randomly selecting 80,000 data as the training dataset and randomly taking 20,000 data as the testing dataset, we can calculate the training and testing accuracy respectively. Initially, we implemented simple grid search for learning rate, weight decay and momentum. We found that the pair with 0.001 learning rate, e-2 weight decay, and momentum in 0.95 and 0.99 have the best accuracy, but this result was not used in our final model.
Weight decay decides the weight of l2 loss. Higher weight might cause underfitting and lower weight can cause overfitting, so we decide to choose 1e-3. For learning rate, we found that smaller learning rate can make the training process relatively slow and large learning rate will underfit(the local or global minimum are missed). As for large momentum, the training process is unable to stop at the minimum while small momentum makes the training process pretty unstable and may be affected by the outliers. As a result, we arise with an idea of annealing learning rate and momentum. With this techniques (Table 3), we no longer have a fixed pair of momentum and learning rate. After early stopping of a trail, we will decrease the learning rate and momentum accordingly. Finally, we find out the following optimal combination of hyperparameter by grid search algorithm again.

A **CNN model** contains many **hyperparameters** such as the batch size, dropout rate, early stopping etc. As for the dropout rate for both CNN and fully connected layers, the following table shows the experimental result of both training and testing data(Table 4). It is shown that, dropout rate of more than 0.01 in convolution layers have underfitting problem. As for the dropout rate in fully connected layers, the dropout rate have relatively smaller effect on the model fitting. All in all, we decide to choose 0.001 as the 2dDropout rate and no dropout for fully connected layers. (Table 4) Additionally, We ran another simple grid search on the model hyper-parameters. We experimented batch size[N, N/10,N/100], number of epoch [100,500,1000], and early stopping[0(bounce back),30,50]. We found that the training time takes longer as we increase the number of batch. Also if there are no early stopping, the training loss will first increase then decrease again. The most optimal model hyper-parameters we found for our model is batch size= N/100, epoch = 1000 and early stopping =30.

CONCLUSION & DISCUSSION

Figure 2 represents the single label cross entropy loss trend after after early stopping at 278 epochs. We noticed that the cross entropy loss decreased drastically in the first 50 epochs while the testing trend tends to always have a high loss than the training trend.

Overall, our final model achieved a 99.99% of training accuracy and the accuracy of submitted dataset is 99.5%. For future work, we may use GAN to generate an extended training data or we could propose an enhanced digits finding algorithm. Further investigations such as image rotation angles or augmented data selection could be done.

STATEMENT OF CONTRIBUTION

Ian and Zhiying worked on the report and the project. Xintong worked on hyper-parameter selection and cases verification.

Appendix: Supplementary Data and Plots

"Overall" represents the accuracy of full images with five possible digits and "single" represents the accuracy of one single digit classification

index	Experimented Architectures	Testing(Overall/single)	Training(Overall/single)
Best model	2 CNN(6,16) +4 linear layer(383 neurons)	(99.60%/99.99%)	(99.99%/99.99)
n1	1 CNN(6,16) +4 linear layer(383 neurons)	(99.25%/99.85%)	(99.88%/99.98)
n2	3 CNN(6,16) + 4 linear layer (383 neurons)	(99.40%/99.89%)	(99.96%/99.99)
n3	2 CNN(10,20) +4 linear layer(383 neurons)	(99.45%/99.99%)	(99.98%/99.99)
n4	2 CNN(6,16) +3 linear layer	(99.25%/99.85%)	(99.90%/99.98)
n5	2 CNN(6,16) + 4 linear layer (339 neurons)	(99.30%/99.85%)	(99.98%/99.99)

Table 1: Detailed information about our proposed model

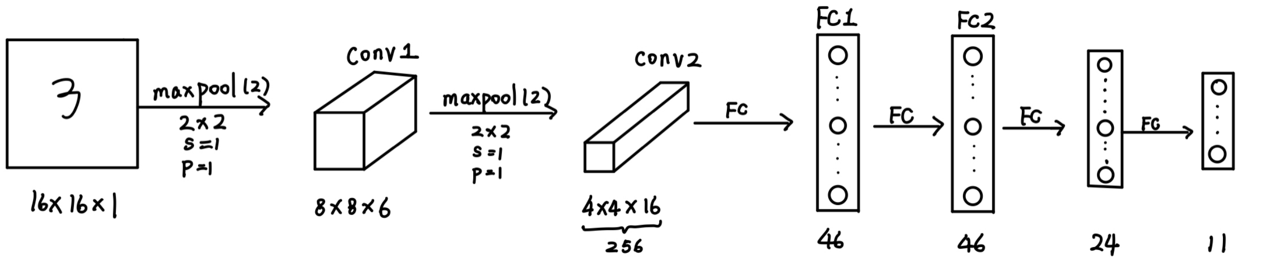

Figure 1: Visualization of our CNN architecture

Optimizer(batch size=N,weight decay=e-3, learning rate=0.0005)	Training accuracy	Testing accuracy
Adam(momentum=(0.9,0.99))	99.88%	99.38%
SGD(momentum=0.95)	94.78%	93.81%
RMSprop(momentum=0.95)	99.25%	97.25%

Table 2: From the above table we found out that Adam has the best overall accuracy out of the three popular optimizers

first trial	lr=0.001	momentum=(0.9,0.99)	weight decay=1e-3
second trial	lr=0.0003	momentum=(0.9,0.96)	weight decay=1e-3
third trial	lr = 0.0001	momentum=(0.9,0.94)	weight decay=1e-3
fourth trial	lr =0.00005	momentum=(0.9,0.92)	weight decay=1e-3

Table 3: Details of each trials for early stop annealing

Dropout 1D (columns) Dropout 2D (rows)	0.00015	0.001	0.010
0.5	(99.45%/99.89%)	(99.55%/99.91%)	(90.45%/95.89%)
0.7	(99.25%/99.85%)%	(99.55%/99.90%)	(90.32%/95.77%)

Table 4: Performance (overall/single) of Dropout 1D X Dropout 2D

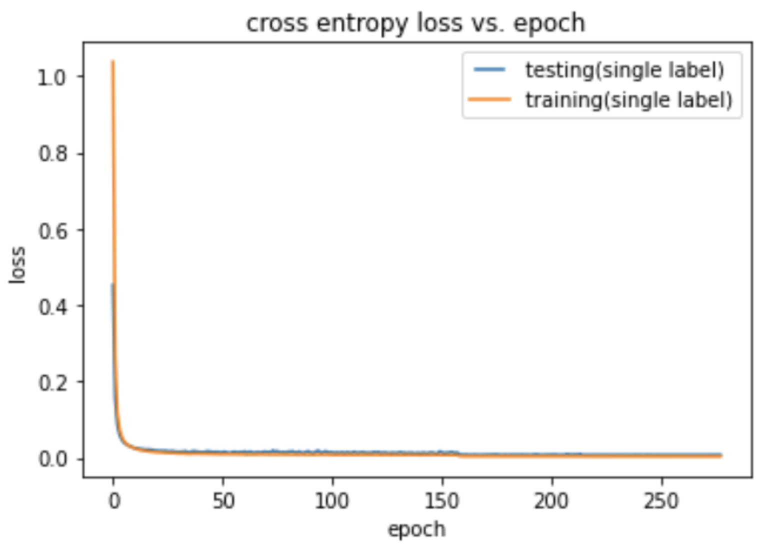

Figure 2: cross entropy loss vs epoch