COMP 551 project3: Multi-label classification of image data

Helen Ren (260846901), Yikan Liu (260825744), Jie Min(260846867)

Abstract: In this project, we have practiced building a convolution neural network to classify a handwritten multi-digit MNIST dataset. We built our neural network by using the Pytorch package on the google colab platform. After the investigation of different CNN models, we found our best model achieves an accuracy of 99.47% in Kaggle competition. In this project, we have gained knowledge about implementing CNN models from scratch to predict the dataset from real-world scenarios.

1 Introduction

The task of this project is to train a multi-digit numbers classification model by implementing the convolutional neural network. After feeding the training data, the model should be able to predict an array of 5 numbers, with each position filled with corresponding digits. If some positions are empty, they are filled with 10 in the output array.

The main challenge of this project is to find an efficient way of training models by tuning hyperparameters and arranging proper layers in a CNN model.

2 Dataset

Fact:

The provided data set for this project is a modified MNIST dataset, which contains images (64*64 pixels) of a group of at most 5 handwritten digits. The digits are aligned horizontally with each other. The given dataset can be divided into 3 parts: the training dataset, the test dataset, and the train labels, where the training dataset consists of 56000 figures and the test dataset consists of 14000 figures. Then we split the training dataset into the validation dataset and the training dataset. We choose to keep 80% of data in the training dataset, and 20% of data in the validation dataset.

3 Experiment approach

A. preprocessing

We use openCV to segment each figure into 5 single digits. An example of segmentation is displayed in Appendix 1.

The empty position would correspond to an empty image. Then, we reshape the train labels dataset into a single array containing all the labels of each single digit.

After preprocessing, we get a new dataset with 280000 single-digit images(28*28 pixels) and 280000 labels. We then feed the new training dataset into our built CNN model.

B. Model choose and parameter tuning

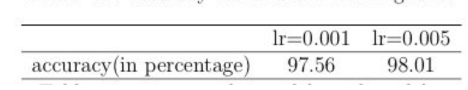
We first implement a 2 layer CNN with the layout as displayed in Appendix 1.

hyperparameter tuning

- Learning rate

We used the learning rate 0.001 and 0.005 for the 2 layers CNN model. The comparison of performance is listed below.

Table 4.2: accuracy for different learning rate.



	lr=0.001	lr=0.005
accuracy(in percentage)	97.56	98.01

As we can see, a learning rate 0.005 would give us worse accuracy. The learning rate controls how quickly the model is adapted to the problem. Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs. However, although larger learning rates bring us shorter training time, it may result in learning a sub-optimal set of weights too fast or an unstable training process.

- Batch size

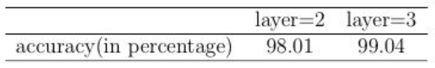
The batch size is a hyperparameter of gradient descent that controls the number of training samples to work through before the model's internal parameters are updated[3]. Batch size controls the accuracy of the estimate of the error gradient when training neural networks. We tried different batch sizes (100, 120 and 150) in our model. We present the test accuracies of our neural network model trained using different batch sizes in Appendix 6. (Orange curves: batch size100, Blue curves: batch size120, purple curves: batch size150).

As we can see, the model with higher batch size has lower asymptotic test accuracy. This is due to the fact that too large of a batch size will lead to poor generalization

Add more layers

Since a deeper model will convolve more input data, we tried to implement a model 3 with 3 layers of CNN (summary layout can be found in Appendix 3).

Table 4.3: accuracy for different models with different layers.



	layer=2	layer=3
accuracy(in percentage)	98.01	99.04

4 Analysis

Training Monitoring : Plotting the Loss-Function and the Accuracy

We plot out the accuracy and loss of our model, and the plots can be found in appendix 4 and 5.

Plotting these plots helps us understand the convergence of the algorithm. The loss plot is decreasing during the training which is what we want since the goal of the optimization algorithm (Adam) is to minimize the loss function. On the right, the plot shows the evolution of the classification accuracy during the training. The more we train the algorithm, the better the classification accuracy. Notice the fluctuation of the accuracy between ~90 and 100 %. Better tuning of hyper-parameters will provide a precise classification.[1]

5 Creativity

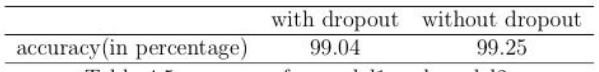
Regularization:

To improve the overall performance we use regularization techniques such as dropout and data augmentation.

- Dropout

  Model1: 2 Layers CNN (without dropout) VS Model2: 2 Layers CNN (with dropout): The summary layouts of Model 1 and Model 2 can be found in Appendix 1 and 2. The comparison of accuracy of these two models is below.

Table 4.4: accuracy for model1 and model2.

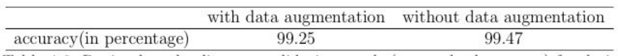


	with dropout	without dropout
accuracy(in percentage)	99.04	99.25

As we can see, the accuracy of the model with drop out is better than the one without dropout. The reason is that dropout, as a regularization technique, plays an important role in preventing overfitting. As we all know, a fully connected layer occupies most of the parameters, and hence, neurons develop co-dependency amongst each other during training which curbs the individual power of each neuron leading to over-fitting of training data. And dropout results in a scenario where at each layer more neurons are forced to learn the multiple characteristics of the neural network.

- Data augmentation

  We also use regularization techniques during our preprocessing stage. After we do geometric transformations (e.g. shift and rotation) and random elastic deformations of the original training examples, we successfully enriched the databases of images. Moreover, in the real world scenario, the digits may have some deformations compared with standard handwritten digits. Thus our model would be more practiced due to the benefit of data augmentation. The accuracy of comparison before and after we do augmentation is displayed as below.



	with data augmentation	without data augmentation
accuracy(in percentage)	99.25	99.47

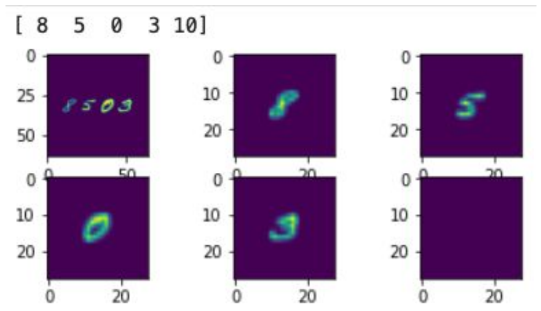
6 Conclusion

In this project, we have discovered lots of useful knowledge about implementing an efficient convolution neural network to classify handwritten digits.

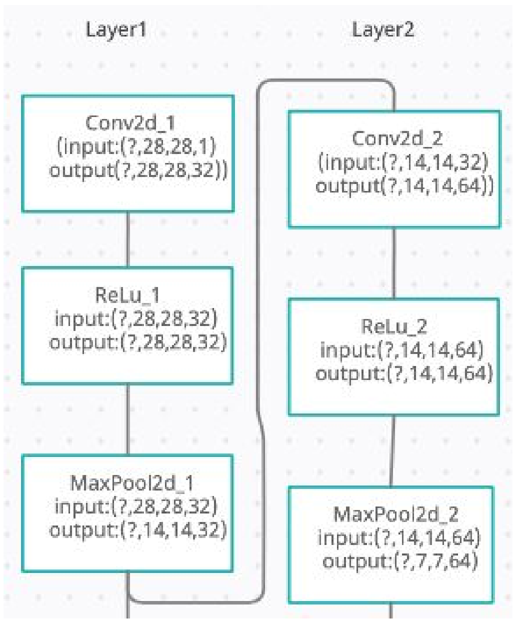
During our training, we investigate that the more epochs we trained, the more accuracy we will get. However, if we train on to many epoches, the accuracy would actually be worse. This is because too many training epochs would cause unavoidable overfitting. Our best model is trained after 130 epoches, with an accuracy of 99.47% in the kaggle competition.

7 Appendix

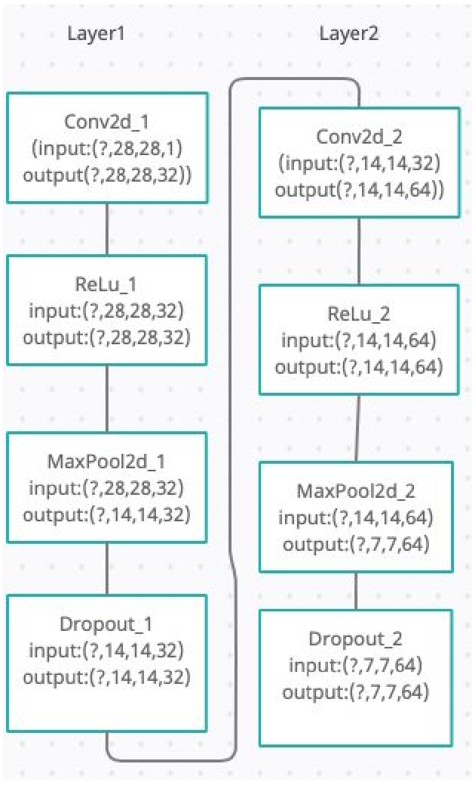
1a.



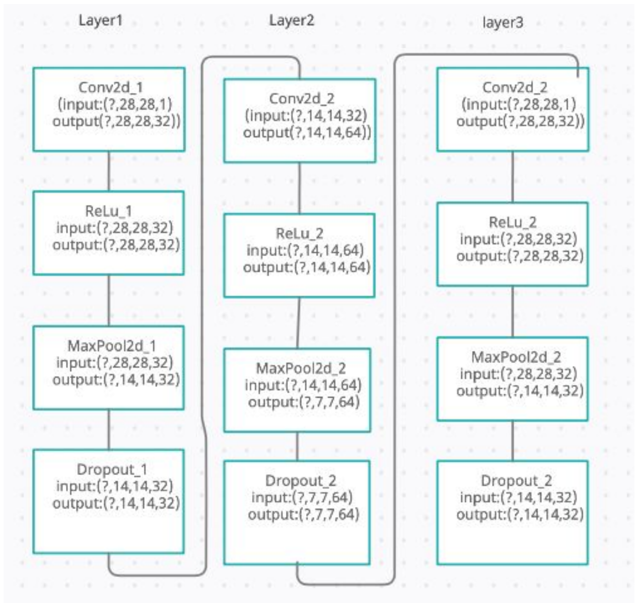
1.


Layer1
Layer2
Conv2d_1 (input:(?,28,28,1) output(?,28,28,32))
ReLu_1 input:(?,28,28,32) output:(?,28,28,32)
MaxPool2d_1 input:(?,28,28,32) output:(?,14,14,32)
Conv2d_2 (input:(?,14,14,32) output(?,14,14,64))
ReLu_2 input:(?,14,14,64) output:(?,14,14,64)
MaxPool2d_2 input:(?,14,14,64) output:(?,7,7,64)

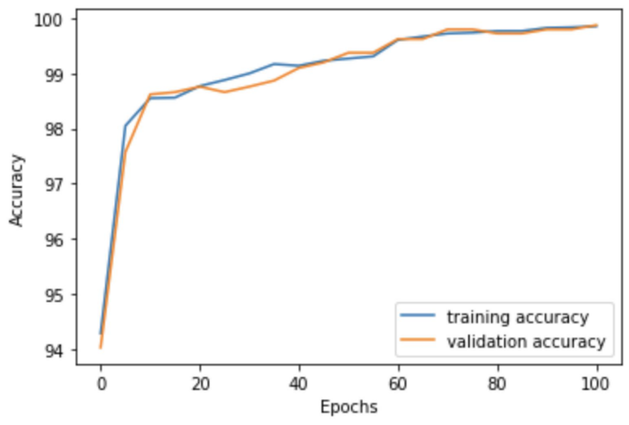
2.


Layer1
Layer2
Conv2d_1 (input:(?,28,28,1) output(?,28,28,32))
ReLu_1 input:(?,28,28,32) output:(?,28,28,32)
MaxPool2d_1 input:(?,28,28,32) output:(?,14,14,32)
Dropout_1 input:(?,14,14,32) output:(?,14,14,32)
Conv2d_2 (input:(?,14,14,32) output(?,14,14,64))
ReLu_2 input:(?,14,14,64) output:(?,14,14,64)
MaxPool2d_2 input:(?,14,14,64) output:(?,7,7,64)
Dropout_2 input:(?,7,7,64) output:(?,7,7,64)

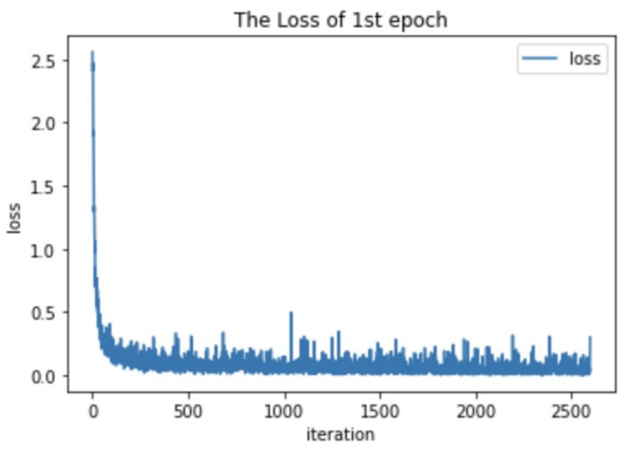
3.


Layer1
Layer2
layer3
Conv2d_1 (input:(?,28,28,1) output(?,28,28,32))
ReLu_1 input:(?,28,28,32) output:(?,28,28,32)
MaxPool2d_1 input:(?,28,28,32) output:(?,14,14,32)
Dropout_1 input:(?,14,14,32) output:(?,14,14,32)
Conv2d_2 (input:(?,14,14,32) output(?,14,14,64))
ReLu_2 input:(?,14,14,64) output:(?,14,14,64)
MaxPool2d_2 input:(?,14,14,64) output:(?,7,7,64)
Dropout_2 input:(?,7,7,64) output:(?,7,7,64)
Conv2d_2 (input:(?,28,28,1) output(?,28,28,32))
ReLu_2 input:(?,28,28,32) output:(?,28,28,32)
MaxPool2d_2 input:(?,28,28,32) output:(?,14,14,32)
Dropout_2 input:(?,14,14,32) output:(?,14,14,32)

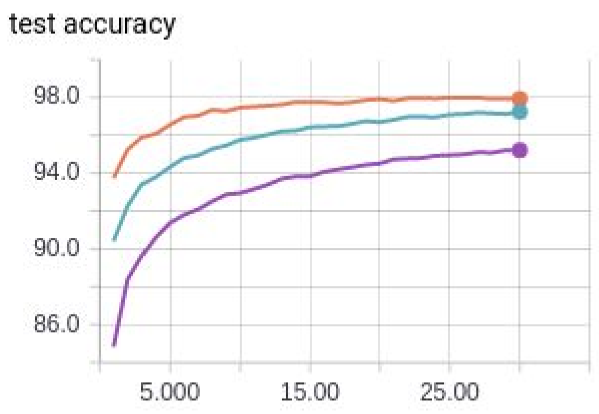
4.



5.



6.



Reference:

[1]https://towardsdatascience.com/convolutional-neural-network-for-image-classification-with-implementation-on-python-using-pytorch-7b88342c9ca9
[2]https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
[3]https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
[4]https://medium.com/mini-distill/effect-of-batch-size-on-training-dynamics-21c14f7a716e