COMP551 MiniProject 3 of Group 99

Zhenyuan Ma 260881867 | Yue Xiao 260740996 | Yihe Zhang 260738383

March 17, 2019

Abstract

Handwritten digits recognition has been a hot research topic in machine learning area and MNIST (Modified National Institute of Standards and Technology) database is one of the most common database used for image classification. There are a wide range of models applied for this topic including linear model, support vector machine and neural networks, and they achieve different performances. In this project, we work on a modified MNIST dataset to find the digit with the largest size and predict its number. After understanding that Inception V3 model is a sequential deep convolution neural network model trained for the ImageNet Large Visual Recognition Challenge, We tried this with different parameters and optimizers and finally achieve a score of 97.3% on Kaggle competition.

1 Introduction

The MNIST (Modified National Institute of Standards and Technology) database is one of the most common database of handwritten digits used for image classification.[1][2] It is often used in "Hello World" programming tutorial for deep learning.[3] In our project, we work on a Modified MNIST dataset with more than one digit appear on each image with grey-scale and fine the label of the digit that occupies the most space. After performing tricky step on image preprocessing, we try deep neural network model inspired Inception V3 model to train the dataset and do prediction. By changing the model structures and hyper-parameters, we achieve a 97.3% score in Kaggle competition.

2 Related Work

Handwritten digits recognition has been a research topic for many years and there are some applications such as recognizing handwritten zip-code[6]. There exist a wide range of methods, some of which are base method[7] and some are combination of classifiers[8, 9, 10]. They use what are commonly used in research areas which include support vector machine, nearest neighbors, neural networks and combination of some of those mentioned. Furthermore, they achieve a good classification error rate. According to [2] and [11], they got a test error rate of 0.23% by using convolutional neural network with width normalization.

3 Dataset and Setup

3.1 Dataset

In this project, we work on a Modified MNIST dataset in which the images contain more than one digit. For given images in packages provided (i.e. *'train_images.pkl'* and *'test_images.pkl'*), each image is represented as a 64 x 64 matrix of pixel values (i.e. the images are grey-scale). And for the labels associated, they are given in *'train_labels.csv'*. We have 50,000 training images with labels and 10,000 test images for label predicting.

3.2 Image preprocessing

Here we perform a trick step for image preprocessing. By using *threshold()* and *findContours()* function in *openCV* package[5], we could eliminate some noisy pixels and find the digit occupied most sapce:

1. Set a threshold and keep pixels larger than 252
2. Find all the contours of digits
3. Calculate the start point (x, y) and size (weight w and height h)
4. Compare $max(w, h)$ and find the largest contour
5. Copy the largest contour and the digit to a new image

After the above steps, we will have a preprocessed image dataset with the digit we want. However, there are some 'bad images' in which two digits may touch each other. In this case, *findContours()* may consider these two digits as one and it should affect the training and prediction result.

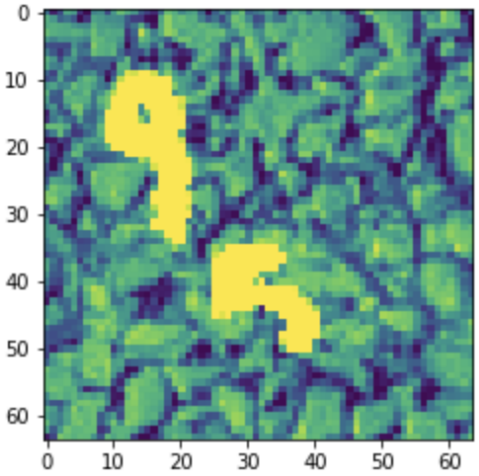

Figure 1: sample image

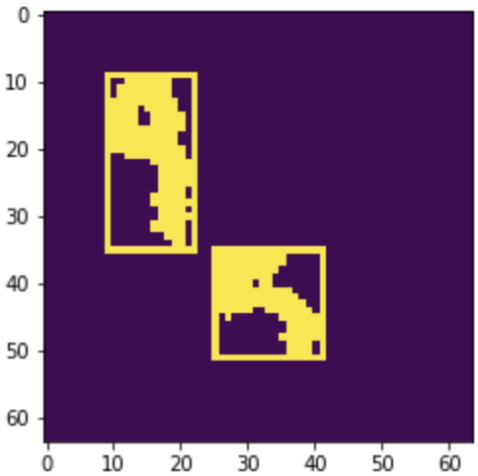

Figure 2: finding contours

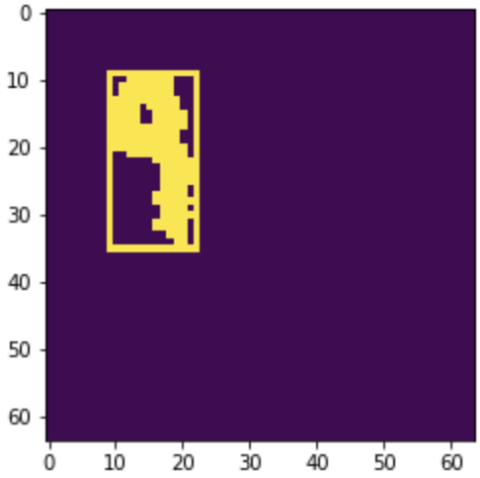

Figure 3: keep the largest digit

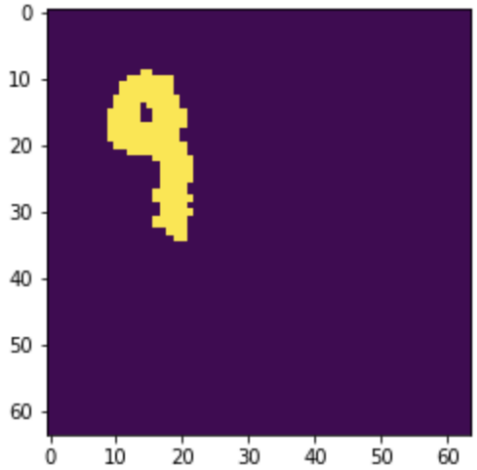

Figure 4: new image

3.3 Setup

We also perform normalization and encode on input and output set. Originally, all the pixels in images are between 0 and 255. Dividing by 255, the input will have values from 0 to 1. For the output y, it is

the label given by an integer. Here we do one hot encode[4] and each output becomes an array of size 10 and $array[i]$ will be 1 if i is the label, otherwise $array[i]$ will be 0.

Since we have 40,000 training images, we applied cross validation to randomly split them into two parts: 90% for training the model and 10% to become validation set.

4 Proposed Approach

4.1 Convolutional Neural Network model

We first try Convolutional Neural Network (CNN) modeling and evaluation. There are so many implementations of CNN on MNIST dataset, and even on Kaggle there are many competitions related. We have viewed some of the works and built our model based on [13, 14, 15]'s implementation. By using *Keras* package[4], we first define a sequential model and then add layers, and figure 5 shows our model with only layers.

After adding layers to our model, we begin to set up optimizer. As we use one hot encode for the output, there will be 10 classes and thus we use *'categorical_crossentropy'* as loss function to measure how our model performs given labels. Moreover, we use *'RMSprop'* optimizer[16] without changing any default parameters. The metric function *'accuracy'* is used to evaluate the performance our model. This metric function is similar to the loss function, except that the results from the metric evaluation are used only for evaluation.

To avoid overfitting, we could find some augmentation techniques. Grayscale is one of the techniques and has already been used before given us the dataset. In this model, we also use *'ImageDataGenerator()'* and its *'flow()'* function to apply some transformations on out training data. Although we do not set any arguments, there are some arguments with default values such as *'zca_epsilon'*, *'batch_size'*, and *'shuffle'*. However, we do not try *'vertical_flip'* nor *'horizontal_flip'* since they may perform wrongly classification on digits such as 6 and 9.

4.2 Inception v3 model

Inception v3 is a widely-used image recognition model. The original model itself is made up of symmetric and asymmetric building blocks, including convolutions, average pooling, max pooling, concats, dropouts, and fully connected layers. Batchnorm is used extensively throughout the model and applied to activation inputs. Loss is computed via Softmax.[12] A high-level diagram of the model is shown in figure 6.

However, in our project, we modified the original Inception v3 model by reducing the number of layers and complexity, since more layers would result in less parameters to train.

Before building models, we generate batches of tensor image data with real-time data augmentation by using Keras preprocessing class[4], and the data will be looped over (in batches).

After preprocessing images and data, we defined a function to apply convolutions 2D and batchnorm through the inception v3 model and built inception v3 model. We then applied cross validation to split the data sets, and trained the sets on the inception v3 model. Moreover, we used ImageDataGenerator from keras to generate new sets of images by randomly rotating, zooming and shifting the input images and fitted those generated images to the inception v3 model.

In order to get a more precise prediction, we built the same model except that we changed the number of epochs in the final classification layer to retrain a pre-trained Inception-v3 model.

5 Results

5.1 Results

Here we attach tables with training information of different models we have tried (some of which are not appeared in the code but we record the result):

Model 1. CNN model, 26 epochs, 450 steps per training epoch, 50 steps per validation, *'RMSprop'* optimizer

epoch	training time	loss	accuracy	val_loss	val_accuracy
1	270s	1.9795	0.3008	1.5337	0.4406
2	287s	0.9681	0.6986	0.6248	0.8137
...	...	...	...	...	...
25	286s	0.2041	0.9494	0.2374	0.9369
26	284s	0.1784	0.9544	0.2773	0.9369

Table 1: Experiment result of CNN model

Model 2. Inception v3 model, 150 epochs, 1000 steps per training epoch

epoch	training time	loss	accuracy	val_loss	val_accuracy
1	79s	1.1247	0.6328	2.4216	0.4382
2	79s	0.4467	0.8719	0.5125	0.8477
...	...	...	...	...	...
149	218s	0.1564	0.9603	0.1177	0.9680
150	218s	0.1543	0.9605	0.1165	0.9720

Table 2: Experiment result of Inception v3 model

The final validation accuracy of each model is shown below:

model setting	val_accuracy
Model 1	0.9385
Model 2	0.9723

Table 3: Experiment result of different model settings

5.2 Leaderboard result

On Kaggle leaderboard, we achieved a score of 0.9730 for the test set by using Inception v3 model.

6 Discussion and Conclusion

From this project, we are impressed by what we can do with MNIST database. Besides our Modified MNIST, there are still some other expansion such as Fashion MNIST and M2NIST[1]. Also, by applying different models (or ensemble of some models) we could achieve nearly perfect prediction[2] on the dataset. However, due to the computational cost and time consumption, we could not try models with more and complex layers and a very large number of epochs. Hopefully, we will have chance to get in touch with more generations of MNIST dataset and more choices of models in future research life.

7 Statement of Contributions

Zhenyuan Ma: image preprocessing, model building, report writing

Yue Xiao: model testing, report writing

Yihe Zhang: model building, model testing, report writing

References

[1] Wikipedia, MNIST database. *https://en.wikipedia.org/wiki/MNIST_database*

[2] LeCun, Y. et. al. The MNIST Database of Handwritten Digits. *http://yann.lecun.com/exdb/mnist*

[3] Taus Priyo, Shehzad Noor. Image Classification Using Deep Learning: Hello WorldTutorial. Medium. 06 Mar 2017. Accessed 08 July 2018. *Https://medium.com/@sntaus/image-classification-using-deep-learning-hello-world-tutoriala47d02fd9db1*

[4] Keras: The Python Deep Learning library. *https://keras.io/*

[5] Open Source Computer Vision Library. *https://docs.opencv.org*

[6] Daniel Keysers Comparison and Combination of State-of-the-art Techniques for Handwritten Character Recognition: Topping the MNIST Benchmark *CoRR arXiv:0710.2231*, 2007.

[7] S. Belongie, J. Malik, and J. Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 24(4):509522, April 2002.

[8] U. Bhattacharya, S. Vajda, A. Mallick, B. B. Chaudhuri, and A. Belaid. On the Choice of Training Set, Architecture and Combination Rule of Multiple MLP Classifiers for Multiresolution Recognition of Handwritten Characters. In *International Workshop on Frontiers in Handwriting Recognition (IWFHR04)*, pages 419424, Tokyo, Japan, October 2004.

[9] K. Chellapilla, M. Shilman, and P. Simard. Combining Multiple Classifiers for Faster Optical Character Recognition. In *DAS 2006, Int. Workshop Document Analysis Systems,* volume 3872 of LNCS, pages 358367, Nelson, New Zealand, February 2006.

[10] J. Dahmen, D. Keysers, and H. Ney. Combined Classification of Handwritten Digits using the Virtual Test Sample Method In *MCS 2001, 2nd Int. Workshop on Multiple Classifier Systems, volume 2096 of Lecture Notes in Computer Science,* pages 109118, Cambridge, UK, May 2001. Springer.

[11] Dan C. Ciresan and Ueli Meier and Jürgen Schmidhuber Multi-column Deep Neural Networks for Image Classification *CoRR arXiv:1202.2745*, 2012.

[12] Advanced Guide to Inception v3 on Cloud TPU *https://cloud.google.com/tpu/docs/inception-v3-advanced*

[13] Aditya Soni MNIST with Keras for Beginners(.99457) *https://www.kaggle.com/adityaecdrid/mnist-with-keras-for-beginners-99457*

[14] Yassine Ghouzam Introduction to CNN Keras - 0.997 (top 6%) *https://www.kaggle.com/yassineghouzam/introduction-to-cnn-keras-0-997-top-6*

[15] Poonam Ligade Deep Neural Network Keras way *https://www.kaggle.com/poonaml/deep-neural-network-keras-way*

[16] Geoffrey Hinton and Nitish Srivastava and Kevin Swersky rmsprop: Divide the gradient by a running average of its recent magnitude http://www.cs.toronto.edu/tijmen/csc321/slides/lecture_slides_lec6.pdf

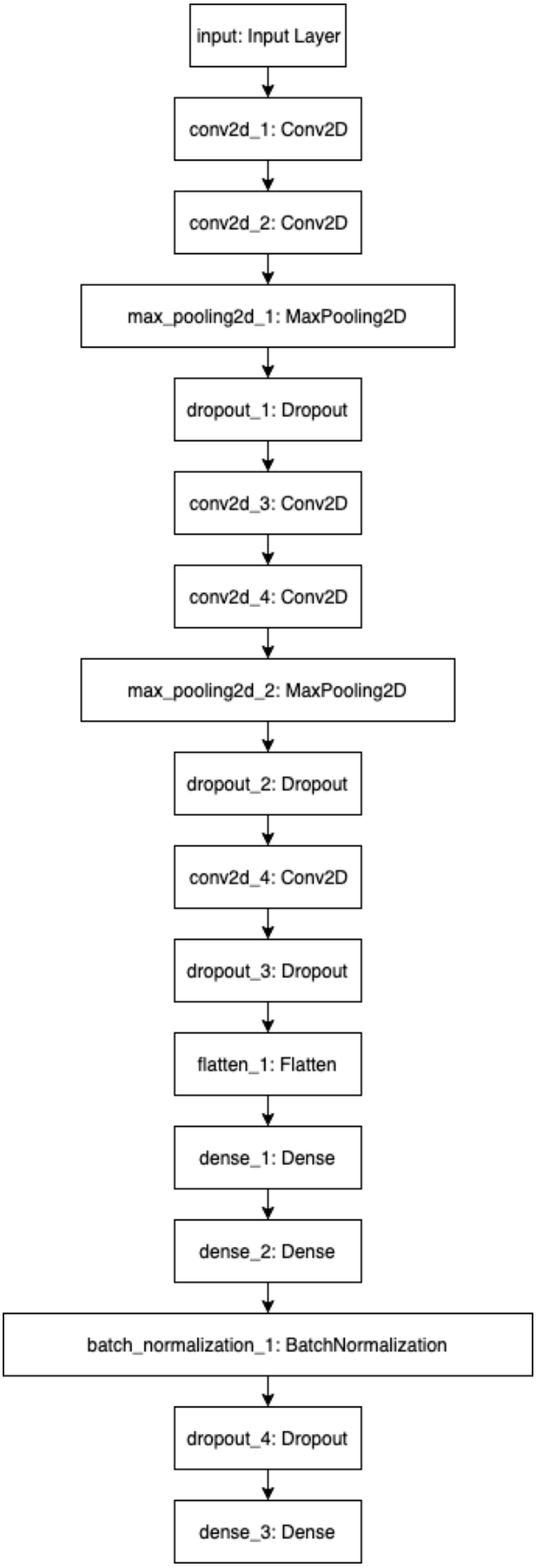

Figure 5: CNN model structure

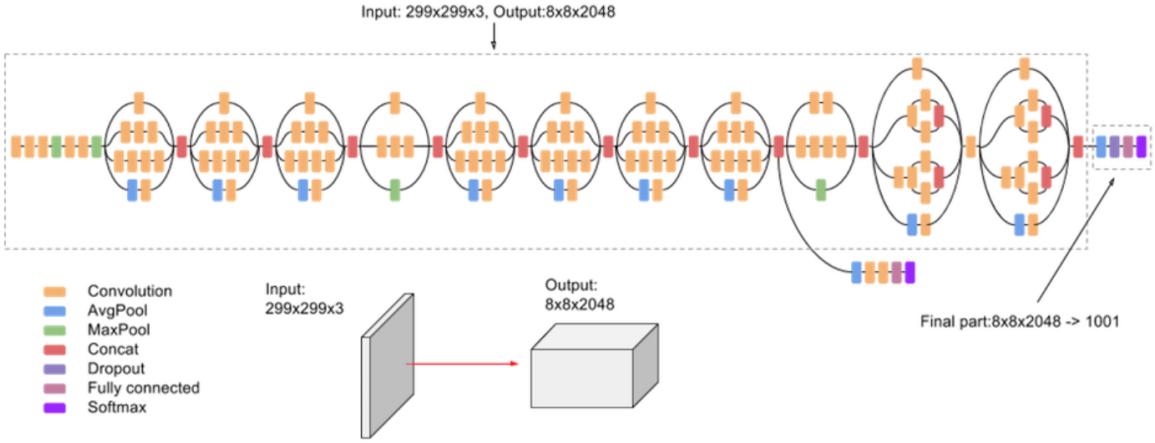

Figure 6: A high-level diagram of Inception v3 model