Big Data and Artificial Intelligence for Business (BUDT737)

Project: Digit Recognizer

1. Team Information

- Budt 737 Team12
- Team Member: Shenger Zhang Kaggle: sherrill Zhang
  Ying Chai Kaggle: YingChai0827
  Yifei Li Kaggle: Yifei Li1
  Kai Li Kaggle: Aurelianus

Kaggle Team Name: 737_team12

2. Model and Method

2.1 Introduction

For this project, we used CNN (Convolutional Neural Network) model, which consists of convolutional layers, pooling layers, flatten layers and output layers. The most important advantage CNN has, compared to normal neural networks, is that it can identify and extract the features or patterns in the image. Therefore, it usually works better than general neural networks in image identification projects.

2.2 Data preparation & Preprocessing

We set up the work environment in Kaggle Notebook. After importing the packages needed and reading the dataset, we first separated the features (X) and target (y), then reshaped the data to 3-dimensional data, with n_samples being -1, width and height both being 28, and n_channels being 1. At last, we normalized the data by dividing them by 255.

2.3 CNN model structure

The first layer is a convolutional layer, in which we use a filter (or kernel) to scan over the pixels of the input image so that we get the patterns of it in a resized image. We set up 32 convolutional kernels with size of 5*5 each, “same” padding which tries to pad evenly left and right, and “relu” as activation function.

Then in the pooling layer, in which we shrink the large image down while preserving their important features, we set the pool size to 2*2. And to prevent the model from overfitting, we have a dropout layer after those layers above, which is a regularization method. The dropout parameter is set to 0.25, meaning that 25% of the previous output is temporarily removed from the network, along with all its incoming and outgoing connections.

After the steps above, we did the convolution again, with a different set of parameters, to strengthen the model. We changed the kernel number to 64 and the kernel size to 3*3, while other settings remained the same.

What comes next is the flatten layer, which means transforming the N-dimensional features into 1-dimension array. And for the hidden layer, we added a dense layer with RGB parameter of 256 and “relu” activation function, to actually classify the images based on the output of all preceding layers. Eventually, for the output layer, we had another dense layer with RGB of 10 and activation function of “softmax”, which is commonly used for classification with probability provided.

CNN Model Summary

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 32)	832
conv2d_1 (Conv2D)	(None, 28, 28, 32)	25632
max_pooling2d (MaxPooling2D)	(None, 14, 14, 32)	0
dropout (Dropout)	(None, 14, 14, 32)	0
conv2d_2 (Conv2D)	(None, 14, 14, 64)	18496
conv2d_3 (Conv2D)	(None, 14, 14, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 7, 7, 64)	0
dropout_1 (Dropout)	(None, 7, 7, 64)	0
flatten (Flatten)	(None, 3136)	0
dense (Dense)	(None, 256)	803072
dropout_2 (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 10)	2570

2.4 Model training and validation

So far we've had a set-up CNN model, and what is needed next is to choose the proper optimizer, loss function, and metric. We used "sparse_categorical_crossentropy()" as the loss function, in case we have an integer-dependent variable. And we chose "adam" for the optimizer as it outperformed other optimizers.

Now we're ready for the training model. In this part, we use all of the training datasets, and used the ".fit()" method to fit the model, with the batch_size being 60, epochs being 20, and validation_split being 0.2. As we can see here, the model automatically splits an assigned proportion of data for validation, so we only need to choose the validation size we want, which is 20%.

3. Results and reports

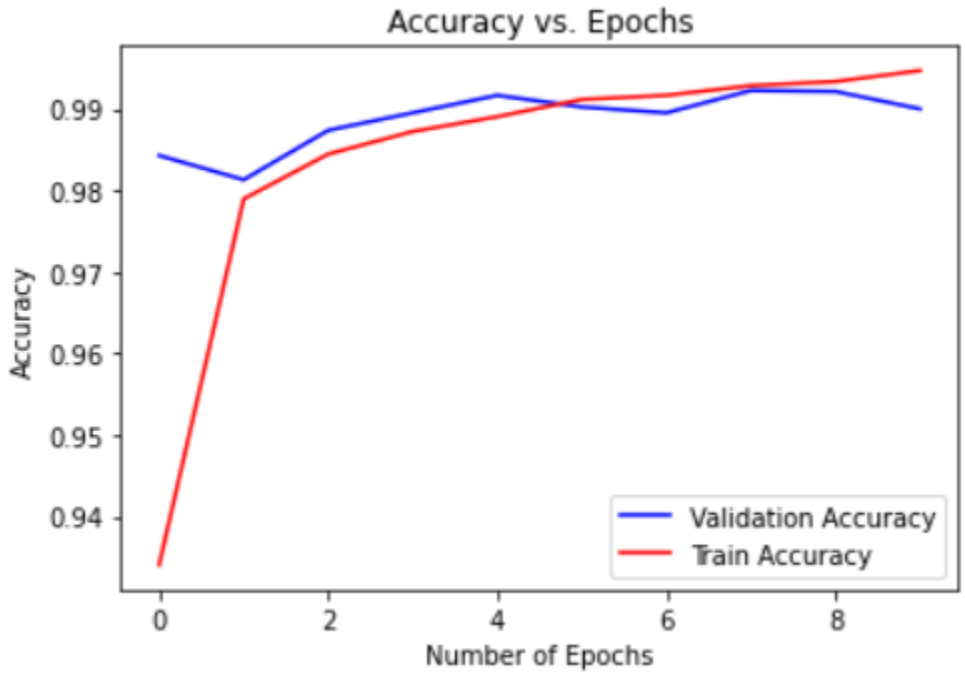
We first looked at the head of the training data set and saw that the data provided by Kaggle was not the same as the data imported from the MNIST package. Every single item has been pre-processed into a 28*28 pixel, with each pixel representing a color.

After we defined the input x and output y of training data, we knew that the number of images in the training data set was 42000, and the number of images in the test data set was 28000.

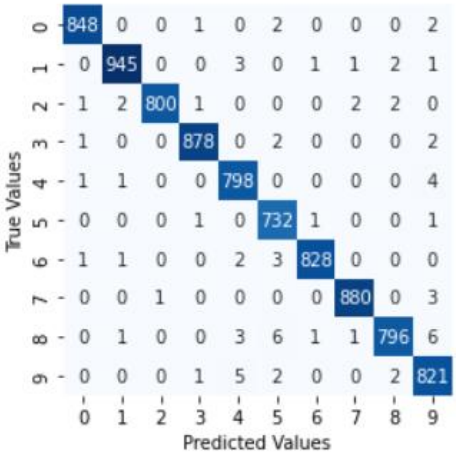
Setting the batch 60, epochs 10 and validation split 0.2, we ran the model on the training data set.

Finally, we had an accuracy which was 0.9913. We predicted on test data and got a test accuracy score of 0.99039.

We also plotted the accuracy curve of the training and validation data set. As the number of epochs increased, the training accuracy also increased and started increasing at the value of 0.9842. The validation accuracy also increased as the number of epochs increased. Unlike the training accuracy, the validation accuracy started at a value higher than 0.98 and fluctuated as the epoch increased. Thus, we assume that there should be a trade-off between accuracy and epochs. With epochs under 10, we've got the desired accuracy, and the increase of epoch doesn't increase much accuracy.



We calculated the confusion matrix between true values and predicted values of the validation data set.



From the confusion matrix we observe that each row represented a number of true classes and each column represented a number of predicted classes. The model may probably misclassify images 8 to 4, 5 and 9; images 9 to 4. Overall, We found that the model was very accurate at classifying from 0 to 9.

Later on, we created the visualization for the output of 28000 images in the test dataset. Most of the time, the predicted number matched the images shown to us.

4. Conclusion

In this process, the convolutional neural network is implemented, which applies a filter to input to create a feature map that summarizes the presence of detected features in the input. Then Sequential model, Conv2D layer, pooling layer, dropout layer, flatten layer and dense layer are added. The Conv2D layer extracts the features of samples. The MaxPool2D layer compresses image features. The dropout layer can prevent overfitting and finally we use the flatten layer to convert the elements into a 1-dimensional array for input to the dense layer. Then Activation function in the dense layer is used for the transformation of the input and finally, the elements are classified into 10 types.

What we have learned in this project:

- The first step to do machine learning is to know about the data well. We could use some functions and visualization methods to learn about data format, data type, and data structure.
- Before we build models and train models, we need to do data preparations, like cleaning data, normalizing data, and modifying data structure.

- To build an optimal model, we need to try many times. There are different logics behind different layers, optimizers and we need to get an understanding of them. The second part is we need to adjust the parameters in the model many times, to make sure the model has good performance.
- In this process, we know how to refer to the official keras document and look up the examples offered in it, which will assist us to do more in the future.

Reference:

1. Keras API reference,https://keras.io/api/

2. THE MNIST DATABASE of handwritten digits, http://yann.lecun.com/exdb/mnist/index.html

3.Yassine Ghouzam(2017) Introduction to CNN Keras - 0.997 (top 6%),https://www.kaggle.com/code/yassineghouzam/introduction-to-cnn-keras-0-997-top-6

4. Data Reshaping for CNN using Keras https://datascience.stackexchange.com/questions/60126/data-reshaping-for-cnn-using-keras

5. Introduction to CNN Keras - 0.997 (top 6%) https://www.kaggle.com/code/yassineghouzam/introduction-to-cnn-keras-0-997-top-6