Digit Recognizer

Ruoyan Yin

916666619

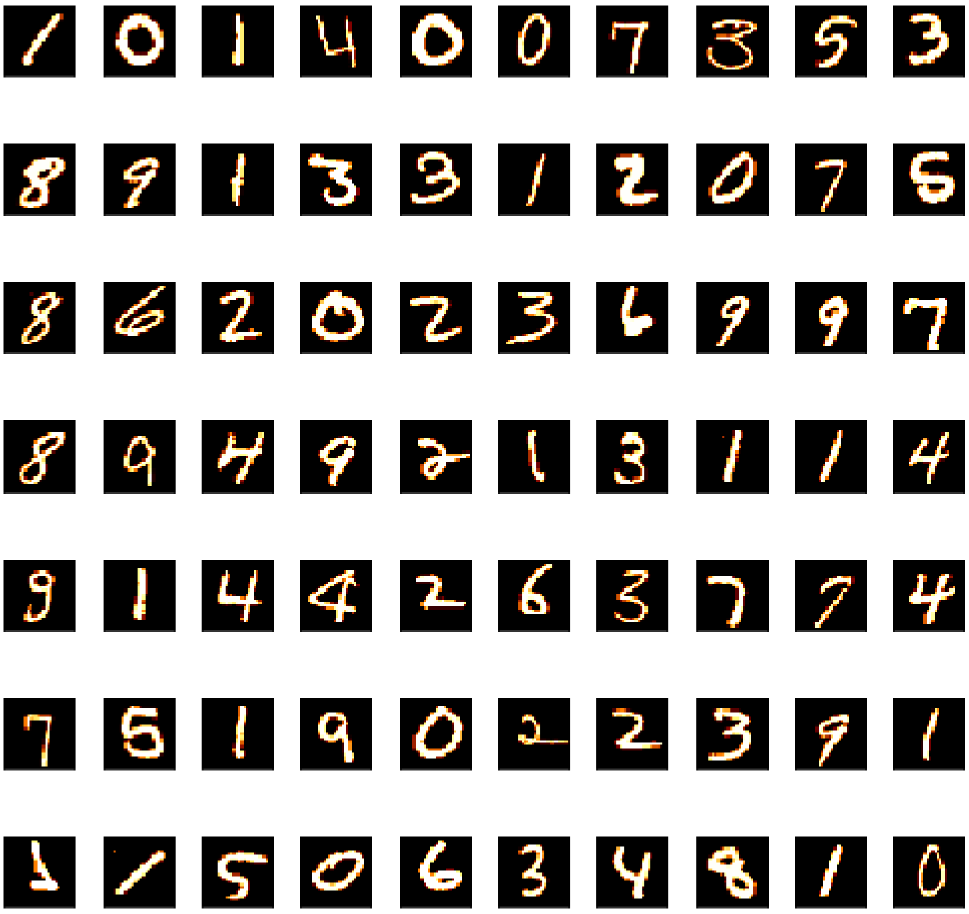
size of group: 1

1. Abstract

In this project, I build up several models to do classification, more specifically, recognition. Overall I implement 5 models including naïve Bayes, decision tree, random forest, SVM and CNN to realize the target. To reduce the dimension of the data without loosing too much information and speed up the calculation, I also use PCA. In some models, there exists hyperparameters (tuning parameters). For these kind of models, I tuned on the parameters to get an optimal model. After implementing each of the models, I do the evaluations on them by using 5-fold cross-validation with the respect of accuracy. I also take the running time into consideration since the efficiency or complexity can also be an important factor of an algorithm.

2. Preprocess data

Before build up models, the very important procedure is to check the format and the composition of the data. The MNIST data is downloaded from Kaggle. And there are 785 columns in total. The first column is the label from 0 to 9 of each picture, and the rest of 784 columns stand for the pixels of the 28*28 grey scale pictures. And the pixel value is between 0 to 255. Higher the value is, darker is the pixel. Then I plot the first 70 digits. To show how it works, I plot the fist 70 digits.



What'smore, first 4 rows of data is shown as below:

label	pixel0	pixel1	pixel2	pixel3	...	pixel780	pixel781	pixel782	pixel783
1	0	0	0	0		0	0	0	0
0	0	0	0	0		0	0	0	0
1	0	0	0	0		0	0	0	0
4	0	0	0	0		0	0	0	0

Clearly, this is a sparse matrix.

To fit the model, first I split the data set into label vector and pixel matrix. Then for the convenience of evaluation, I split the data into two parts -- training data and test data with size of test data being 0.2. So under this setting, there are 33600 observations in training data and 8400 in test data.

3. Models and Evaluation

In this part, I build up all the models, tune on necessary parameters, and evaluate the models with respect of accuracy by implementing 5-fold cross-validation and efficiency by checking the running time.

3.1 Naïve Bayes

Naïve Bayes is a supervised classifier based on applying Bayes theorem with strong (naïve) independence assumptions between the features. And in my project, I tried two kind of posterior -- Gaussian and multinomial.

3.1.1 Gaussian Naïve Bayes (GNB)

After fitting the model, the result comes as:

```
Model: GaussianNB
Running time: 1.8853528499603271
Accuracy: 0.5977254989416543
```

The efficiency of GNB is quite ideal -- it only takes 1.8853 seconds to train the model. However, the average accuracy on test data using 5-fold cv is just 0.5977. So GNB is not preferred even it runs fast.

3.1.2 Multinomial Naïve Bayes (MNB)

Since the outcome of GNB is quite bad, I tried MNB. But the nothing had been improved as result being:

```
Model: Multinomial Naiive Bayes
Running time: 1.8795948028564453
Accuracy: 0.5977254989416543
```

The accuracy is exactly the same as GNB. So after these two attempt. I draw the conclusion that naïve Bayes is not suitable for this situation and decide to change another type of model.

3.2 Decision Tree

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility.

In this part, I implement the decision tree to do classification.

```
Model: Decision tree
Running time: 5.37388277053833
Accuracy: 0.791057324556544
The training accuracy is 1.0.
The test accuracy is 0.8558333333333333
```

Decision tree also have some problems that cannot be overlooked that is it overfit the training data. As from the output, the accuracy on training data is 1.0 which means all the predictions on training data are correct. However, the performance on test data is poor with average accuracy 0.789. It can be seen that other models can work much better than this in the later part of this report.

3.3 Random forest(RF)

A random forest is simply a collection of decision trees whose results are aggregated into one final result. Their ability to limit overfitting without substantially increasing error due to bias is why they are such powerful models. Just as shown above, overfit is the obvious problem by using decision tree. So, by implementing random forest, things should be improved.

First of all, to see the performance of the **RF**, I initialize the n_estimators = 400 and n_jobs = 4 to have some basic impressions on how well it works. Actually, after fit the model, I got the evaluation as below:

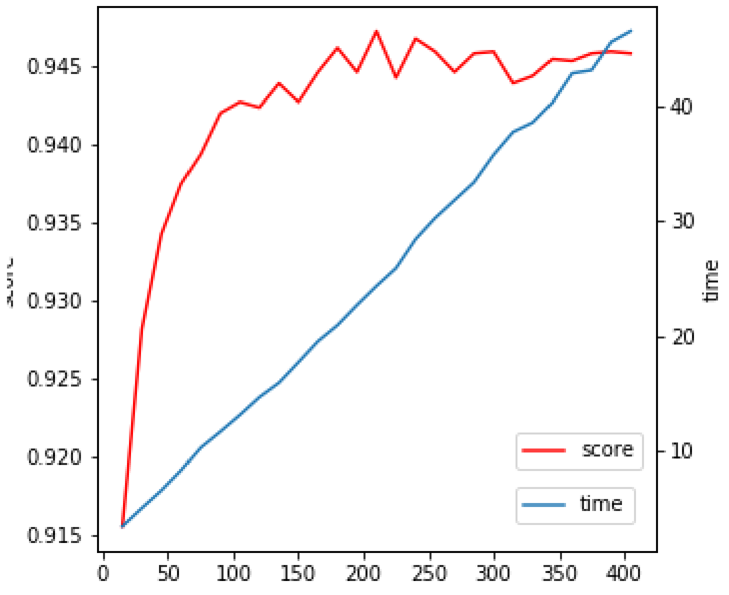
```
Model: Random Forest
Running time: 24.555639028549194
Accuracy: 0.9446425454686327
```

The average accuracy reached 0.945 which is quite a big pace. And though the running time to train the model is about 25 seconds, but considering the size of the training data and the performance of a laptop, this is totally acceptable.

But this model still can be improved by reduce the number of estimators which leads to running time being shortened. This intuition is straightforward. The number of estimators --400-- used in this model is quite large, and hopefully the accuracy actually already converges before it. Under this assumption, I tuning on n_estimators.

3.3.1 Tuning on n_estimators

By changing the n_estimators and fitting the model iteratively, I got the plot as below:



The red curve is the score v.s n_estimators, and the blue one is time consumption v.s n_estimators. Time consumption continuously increase with n_estimators. And the red curve converges at around n_estimators = 115. So under this case, one reasonable final random forest model should be the one with n_estimators to be 115.

3.3.2 Fit the optimized model

Then I fit the model and the result is:

```
Model: Random Forest
Running time: 7.49285101890564
Accuracy: 0.9438126843241047
```

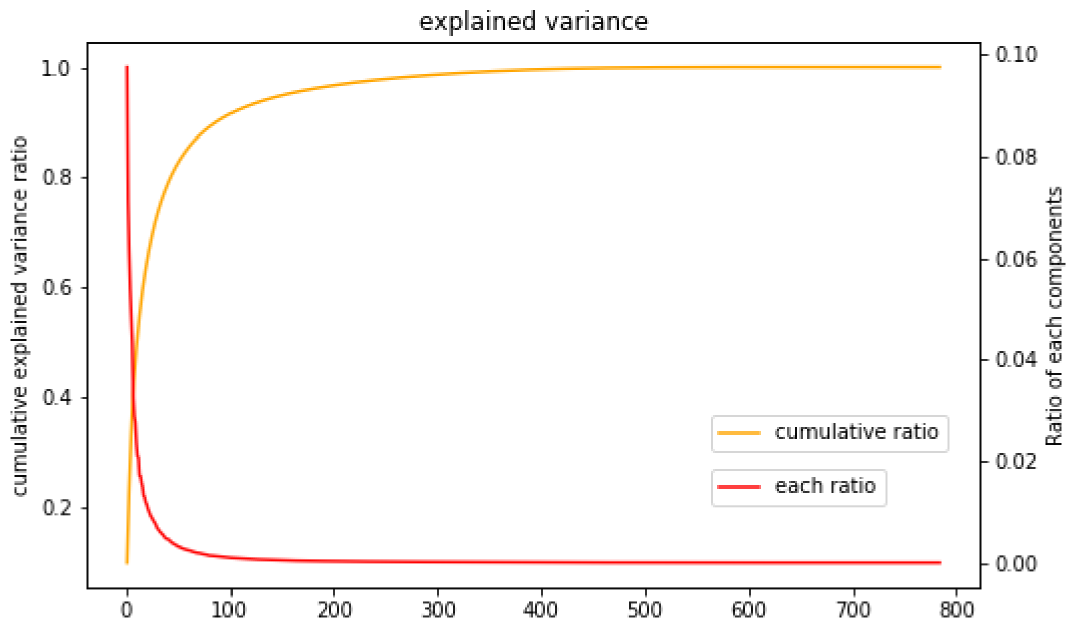
As we can see, the running time dropped from 38.5717 seconds to 10.0390 seconds without at the expense of accuracy.

3.4 PCA and SVM

The advantage of SVM is that it can deal with non-linear classification efficiently using the kernel trick implicitly mapping their inputs into high-dimensional feature spaces. But it also has drawbacks. One of the weakness is that it is very sensible to the dimension meaning if we use the unprocessed data directly, it may take really long time to train the model. So before train SVM model, it is necessary to reduce the dimension of the data. One of the methods is principal components analysis (**PCA**)

3.4.1 Implement of PCA

Two important outputs from PCA is components and explained variance ratio. To decide on the number of components I want to use to transform the data, I will use the cumulative explained variance ratio as criterion. After set the cumulative explained variance ratio to be larger than 0.85, I got the number components is 59 and the corresponding ratio is 0.8508. This plot demonstrate how the cumulative ratio changing with components.



It can be seen that, it is not necessary to include all the components into consideration since most of them carry little information on the data.

3.4.2 Combination of PCA and SVM

After decided on the components, I then transformed the pixel matrix and then train SVC model and used radial basis function kernel (*rbf*).

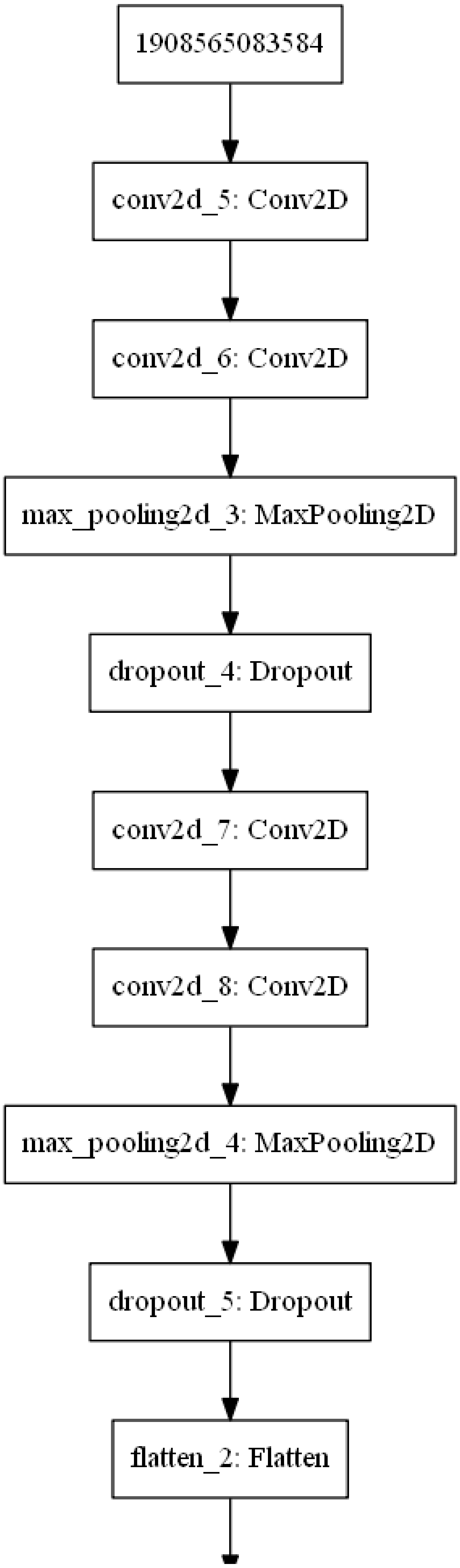
```
Model: SVC
Running time: 22.626350164413452
Accuracy: 0.9625042564683233
```

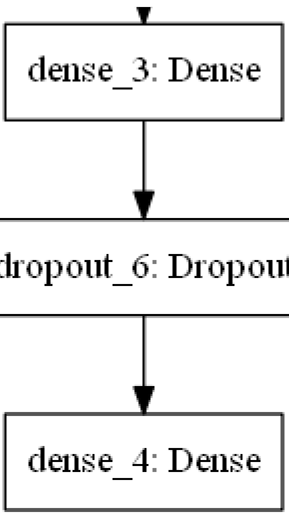
This time, it did not take really long to train the model - just 15 seconds with 0.9629 accuracy. The outcome also validate that the choice of the components was reasonable and the transformed data kept most information from the original data.

3.5 Convolution Neural Nets - CNN

CNN now is a popular method in machine learning area and is widely used in image recognition.

I create a model with structure:

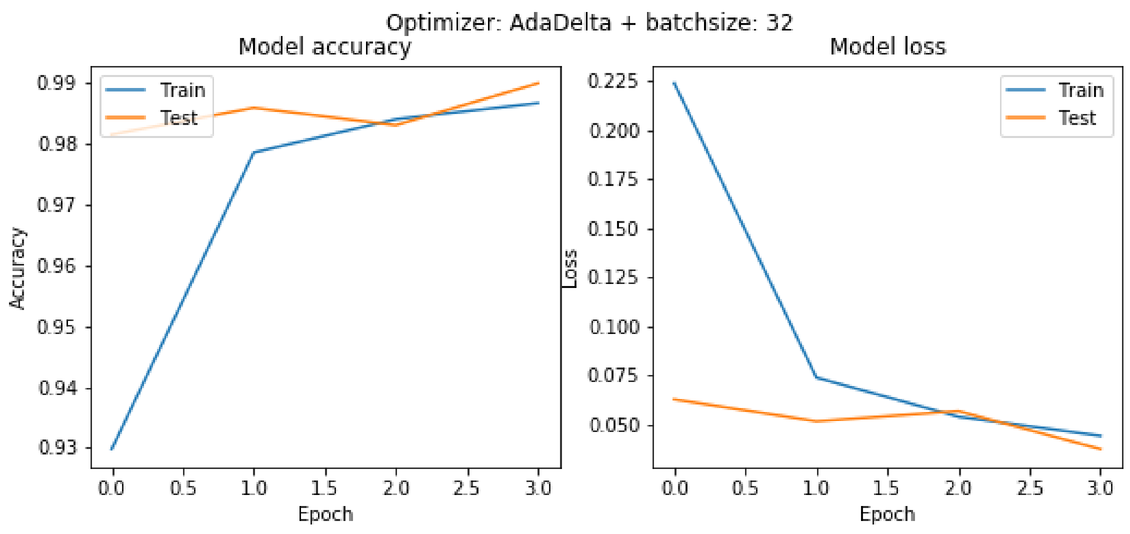

1908565083584
conv2d_5: Conv2D
conv2d_6: Conv2D
max_pooling2d_3: MaxPooling2D
dropout_4: Dropout
conv2d_7: Conv2D
conv2d_8: Conv2D
max_pooling2d_4: MaxPooling2D
dropout_5: Dropout
flatten_2: Flatten

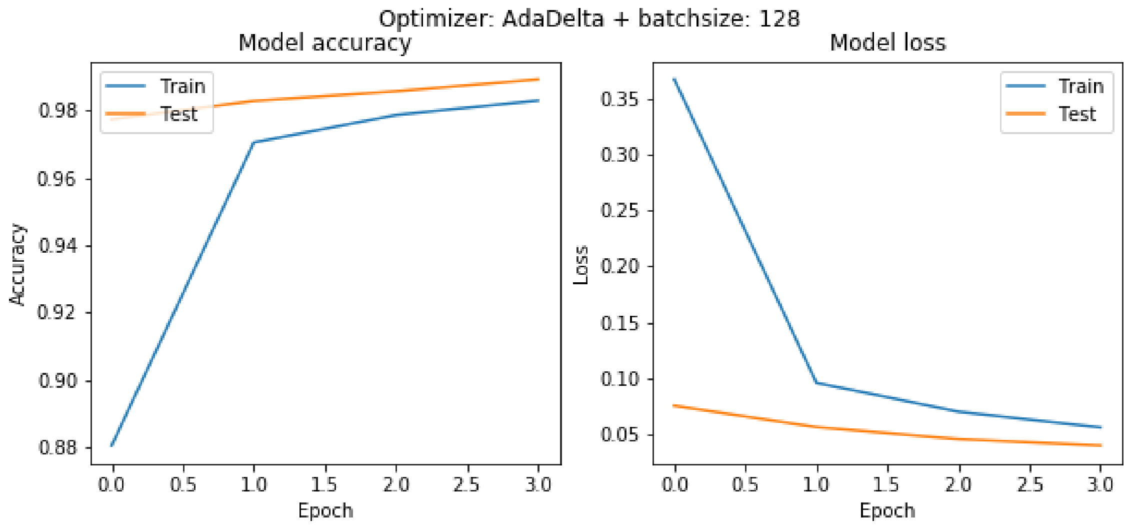


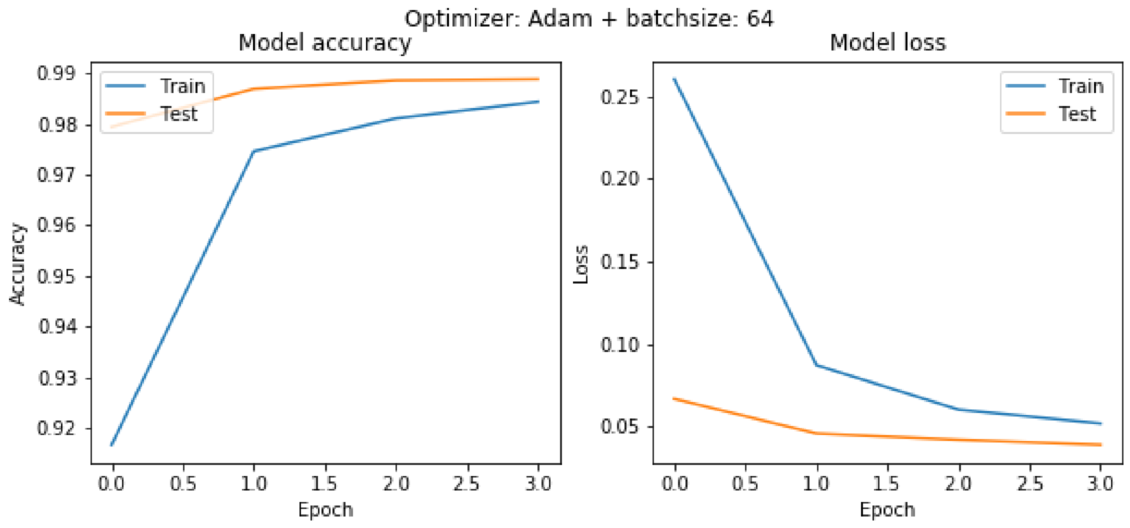
In my CNN model, I choose categorical_crossentropy as my loss function. Then I tried different combinations of optimizer and batch size. The result is listed as below:

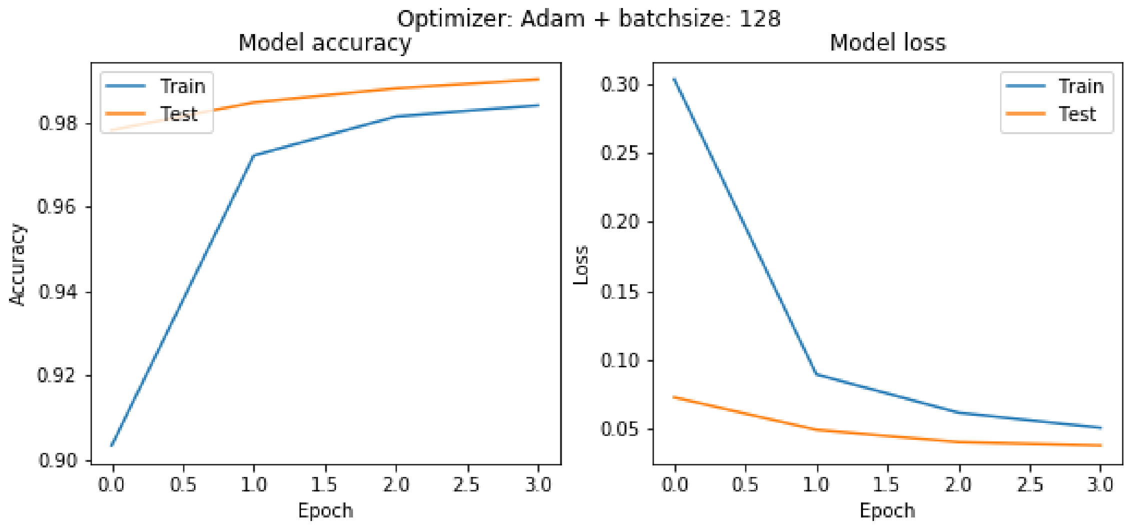
	AdaDelta + 128	Adadelta+ 32	Adam + 128	Adam +64
Test loss	0.0395927	0.03037407	0.03818897	0.03890653
Test accuracy	0.9892	0.99	0.99011	0.9888
speed	3ms/step	3ms/step	3ms/step	3ms/step

The plots below is the curves which demonstrate the performance of each combination. And as we can see, the accuracy get to 0.99 very fast for all the models and the accuracy of them are quite impressive.









Conclusion

After training all the models and making comparisons between them, I conclude that:

- Naïve Bayes is not suitable for this kind of problem.

- Random forest can fix the overfitting problem in decision tree which is the often case, and the accuracy is higher than decision tree as well.
- PCA is really helpful when dealing sparse matrix and speed up some dimension dependent algorithms. By combining PCA and SVM, it also gets an impressive accuracy. It is even attractive when considering its speed.
- By making comparisons between models, CNN has the highest accuracy- around 0.9, and for my device, it did not really take a long time to train - about 60 seconds per epoch. Further more, four combinations of optimizer and batchsize get similar performances which indicates CNN is quite an ideal model for this problem. But it also has some drawbacks. One of them is using CNN on a device without Nvidia GPU may cause the algorithm takes really long to calculate since tensorflow is specially optimized on GPU.