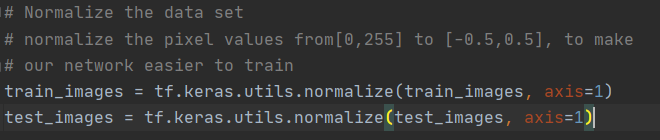
Part 4: practice

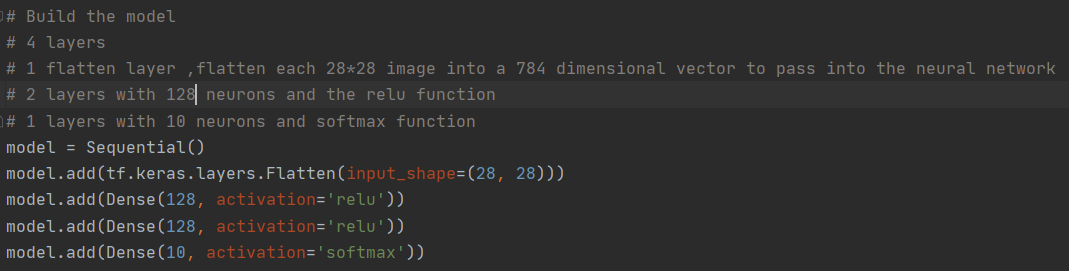
In this part, we tried to implement handwritten numbers recognition in python. Our implementation including Artificial neural network (ANN) and convolutional neural network (CNN), both of them are based on the API provided by Keras and tensorflow. As for the training data set, we both used MNIST in these two implementations.

3.1 Artificial neural network (ANN) implementation

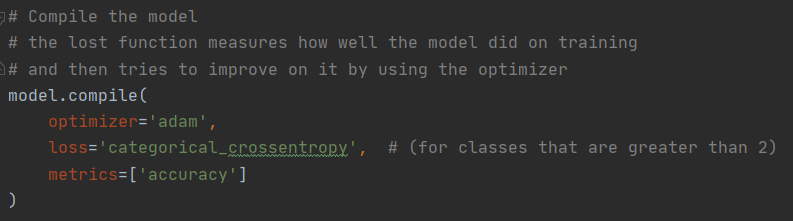
In our implementation of ANN, we first normalized the dataset. In original, the pixel values of the images in MNIST are ranged from 0 to 256. We are supposed to normalize them to -0.5 to 0.5, which can make it easier to training. And in this progress, we used corresponding API provided by Keras.



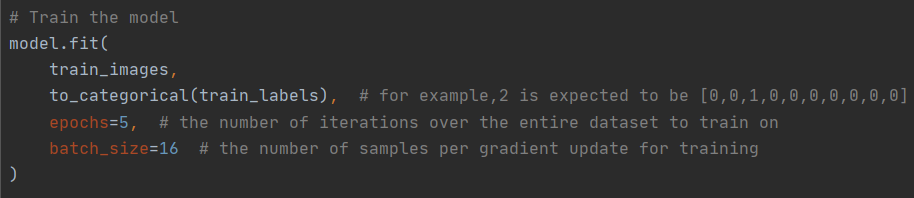
Then we started to build the Artificial neural network training model.

We constructed this ANN with four layers. The first is a flatten layer ,which can flatten each 28\*28 image into a 784 dimensional vector to pass into the neural network. The following 2 layers both have 128 neurons. As for the activation function, we used Rectified Linear Unit (ReLU). The last layer with 10 neurons and used softmax function as ths activation function.

The next step is to compile the model. We choose Adam (Adaptive Moment Estimation) as the optimizer. “Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments.”(Keras API reference) As for the lost function, we used categorical crossentropy, which is suitable for classes large than 2(we have 10 classes needed to be classification, which represent 0 to 9). The lost function measures how well the model did on training and then tries to improve on it by using the optimizer.



Then we are supposed to train the model. We set the epochs equal to 5, which means the entire dataset will be trained on our ANN model for five times. On the other hand, we set the batch size equal to 16, which represented the number of samples per gradient update for training.



After the first iteration of train, the lost value is 0.2314, the accuracy is 0.9307.

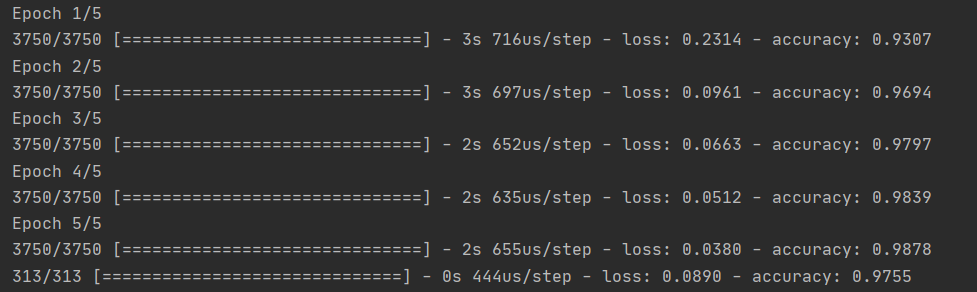
After the second iteration of train, the lost value is 0.0961, the accuracy is 0.9694.

After the third iteration of train, the lost value is 0.0663, the accuracy is 0.9797.

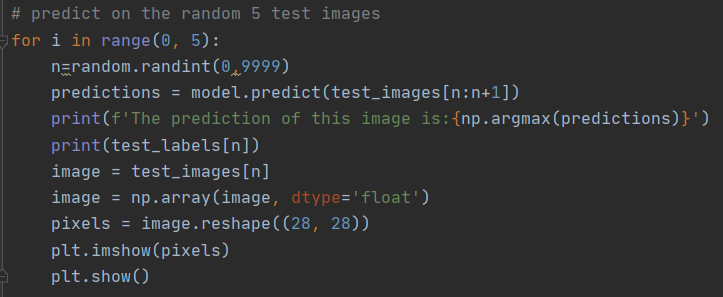
After the fourth iteration of train, the lost value is 0.0512, the accuracy is 0.9839.

After the fifth iteration of train, the lost value is 0.0380, the accuracy is 0.9878.

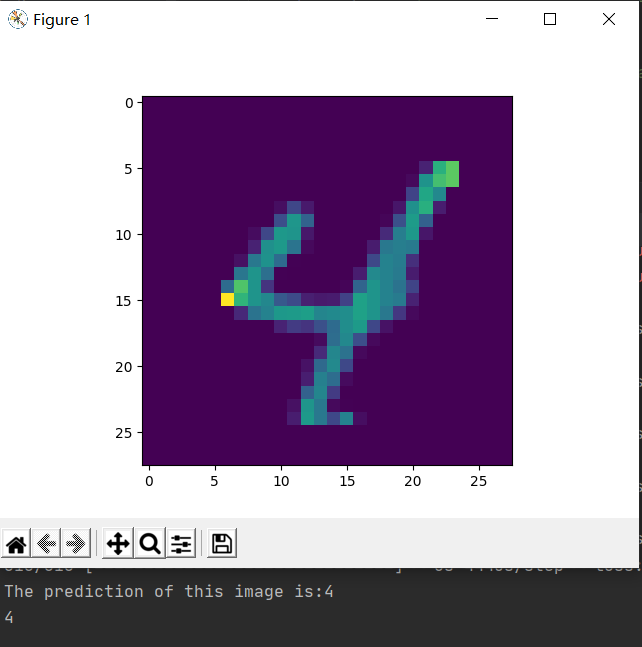
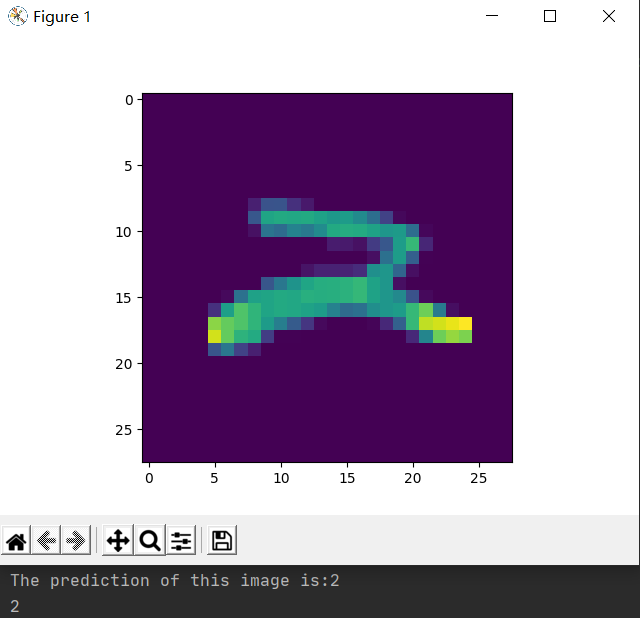
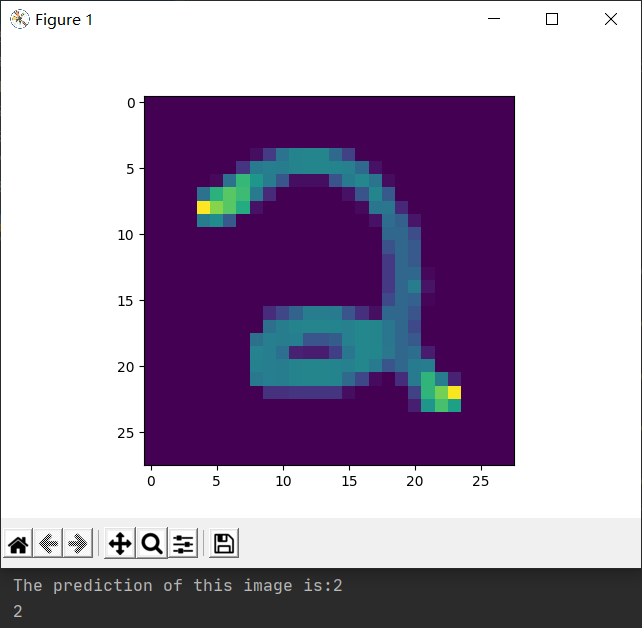
The result on the test dataset is lost value equals to 0.089, and the accuracy is 0.9755.

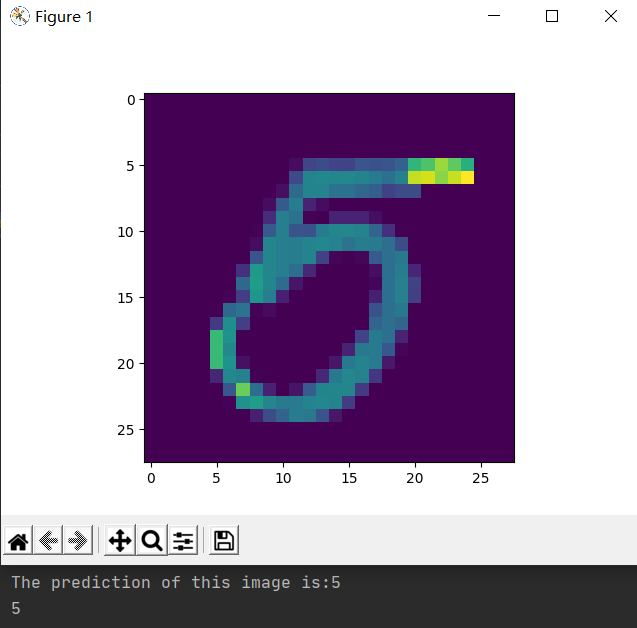


In order to verify our training model further, we random picked five pictures from the test dataset to see whether the prediction given by our model is the same as the label of the picture, which represent the correct answer of the recognition output.

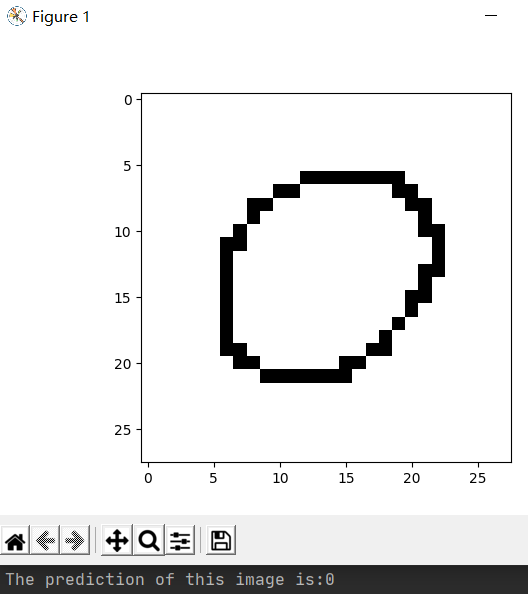
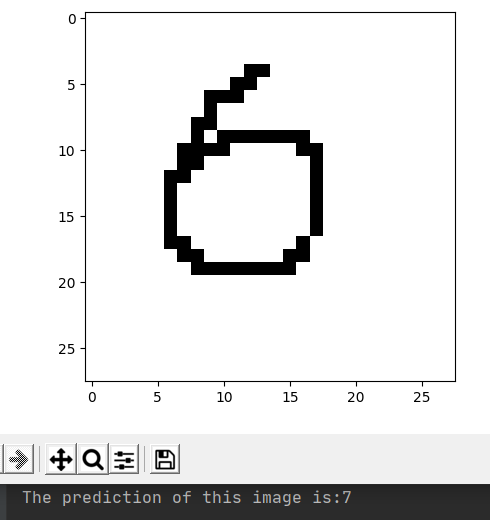


Almost all the predictions are correct.



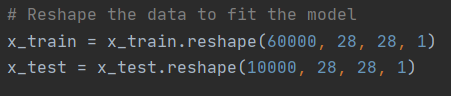


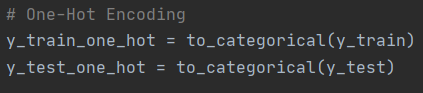
However, when we use this model to recognition our own handwritten images, the accuracy is much lower than our expectation, which is about 0.60. The reasons of this problem may be that the images have not been preprocessed and our model has been overfitting on the training dataset.

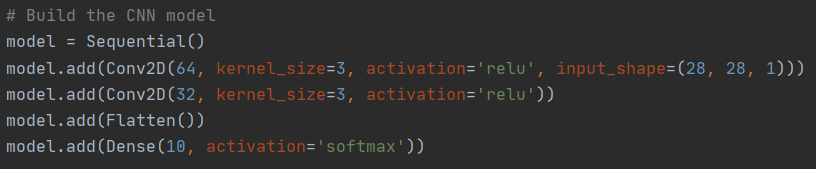
3.2 Convolutional Neural Network (CNN) implementation

In our implementation of CNN training model, we also first normalized the training dataset and the test dataset to make it compatible with our CNN training model. Moreover, we are supposed to one-hot encoding the training labels and the test labels, which can also make it easier for model training.

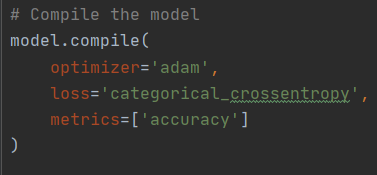




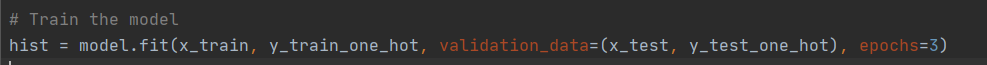
Then we started to build our CNN training model. This model is also composed of four layers. The first two layers are convolution layers. “2D convolution layer, this layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs.”(Keras API reference) We set the filter numbers equal to 64 and 32 separately, and the kernel size equal to 3. We also used Rectified Linear Unit (ReLU) as the activation function. The next layer is a flatten layer, which can connect the convolution layers and the dense layer. The last layer is the same as the last layer we used in ANN training model, it is with 10 neurons and used softmax function as ths activation function.



The compiling step is exactly the same as we did in ANN, we used Adam (Adaptive Moment Estimation) as the optimizer and categorical crossentropy as the lost function.



The following step is model training. We set the epochs equal to 3.



After the first iteration of train, the lost value is 0.2262, the accuracy is 0.9589.

The result on the test dataset is lost value equals to 0.1093, and the accuracy is 0.9680.

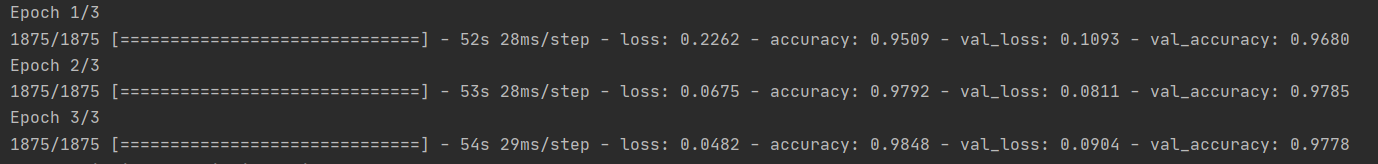
After the second iteration of train, the lost value is 0.0675, the accuracy is 0.9792.

The result on the test dataset is lost value equals to 0.0811, and the accuracy is 0.9785.

After the third iteration of train, the lost value is 0.0482, the accuracy is 0.9848.

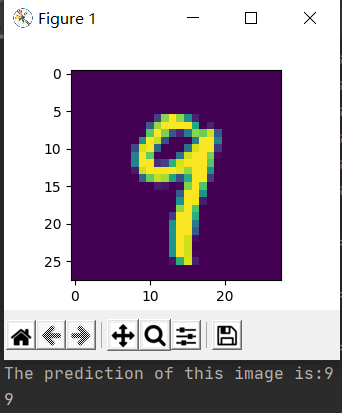
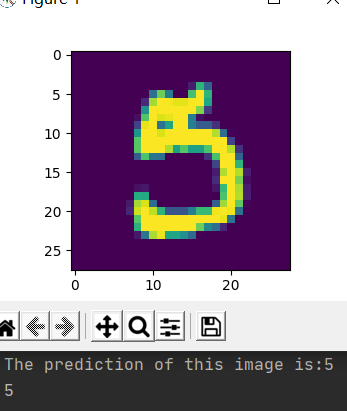
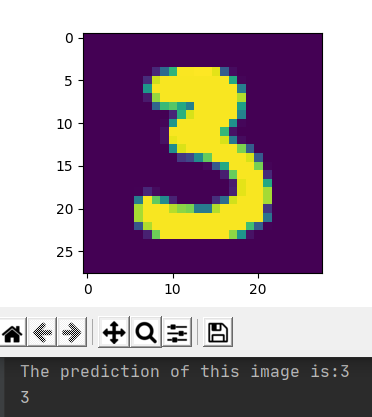
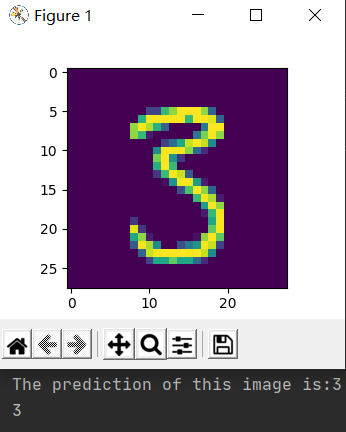
The result on the test dataset is lost value equals to 0.0904, and the accuracy is 0.9778.

The output is better than ANN training model.

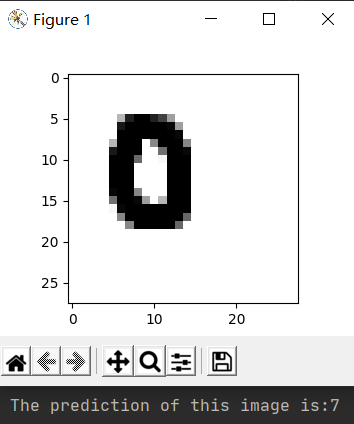
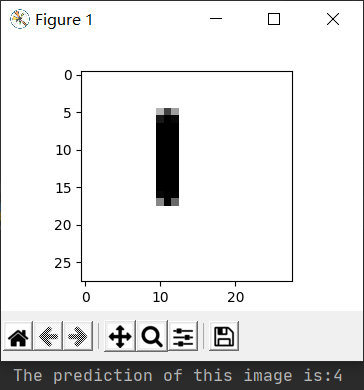
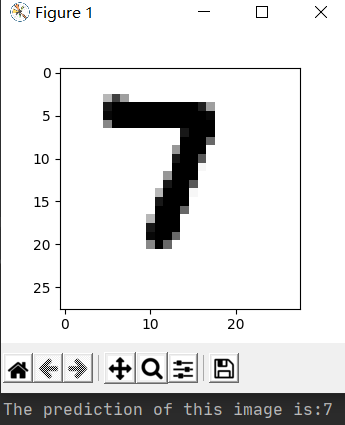


In order to verify our training model further, we also random picked five pictures from the test dataset to see whether the prediction given by our model is the same as the label of the picture.

Almost all the predictions are correct.

However, when we use this model to recognition our own handwritten images, the accuracy is much lower than our expectation, which is about 0.60. The same as the ANN model, we think this may cause by the images have not been preprocessed and our model has been overfitting on the training dataset.

References:

1. Keras API reference about activation function adam <https://keras.io/api/optimizers/adam/>

2. Keras API reference about 2D convolution layer https://keras.io/api/layers/convolution\_layers/convolution2d/