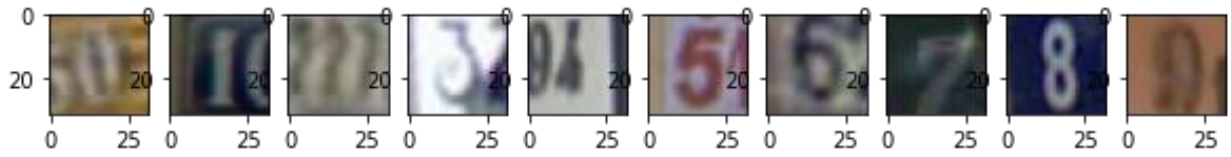
CNN Classifier for Street View House Numbers (SVHN)

Dataset

The Street View House Numbers (SVHN) dataset contains 73257 32*32 RGB images in train set and 26032 32*32 RGB images in test set. It has 10 labels, corresponding to digit "0" to digit "9". The goal is to fit a CNN model on the train set, with at least 91% test accuracy.

Data Preprocessing

The original dataset has a shape of 32*32*3*n, and I transposed dataset to n*32*32*3 for model training. The original dataset contains labels from "1" to "10", which "10" represents digit 0, so I change it to label "0". Below is a random sampling image on each label:



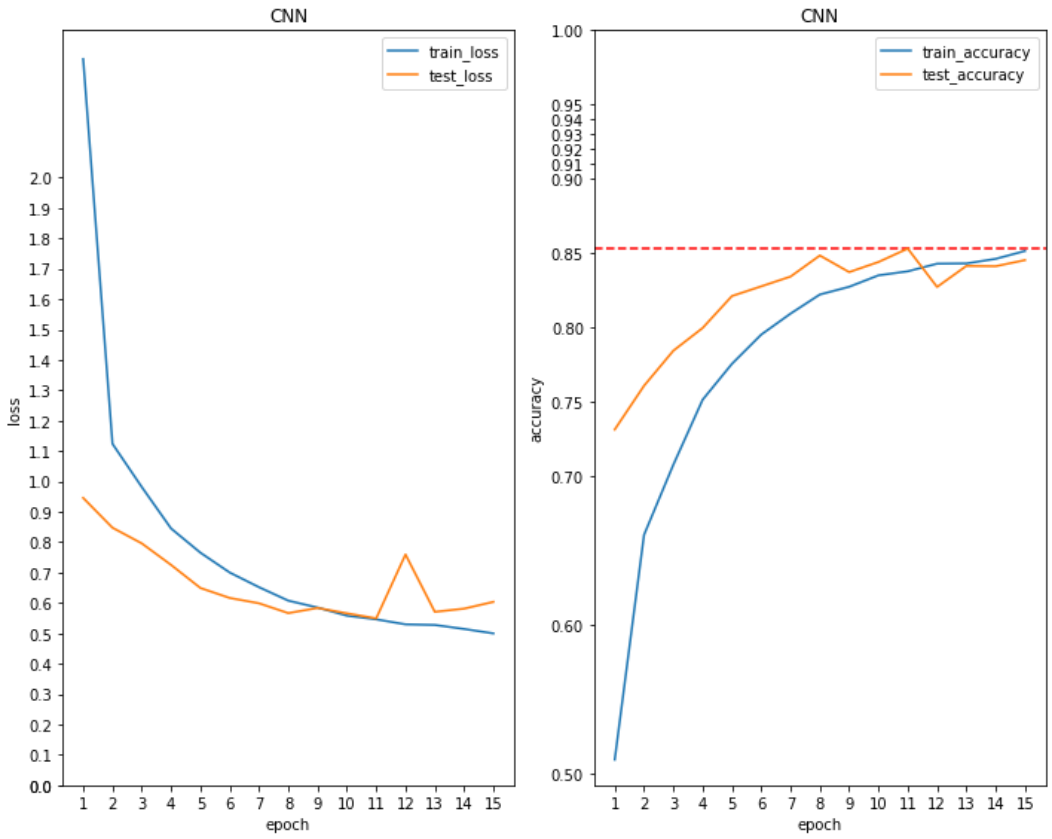
Initial CNN Model:

I start with the CNN model provided by Keras' mnist_convert example:

The CNN model has input layer with same dimension for each image, then has a convolutional 2D layer with 32 filters, kernel size 3*3 and activation function "relu". This will create 32 feature maps, each of them has dimension 30*30 (3*3*3 kernel with default stride = (1,1) on 32*32*3 image). The next layer applies max pooling with size 2*2 on our features map, which reduces the size of feature map by 4. Then it has a second convolutional 2D layer with hyperparameters shown above. It has a dropout layer with 50 percentage, followed by the full connected layer with size 10, corresponding to 10 classes.

Training the model with optimizer "adam", 15 epochs and batch size 128, then evaluating its performance on the test set. We can see the test accuracy is stopping around 0.85, the visualized result is shown below:

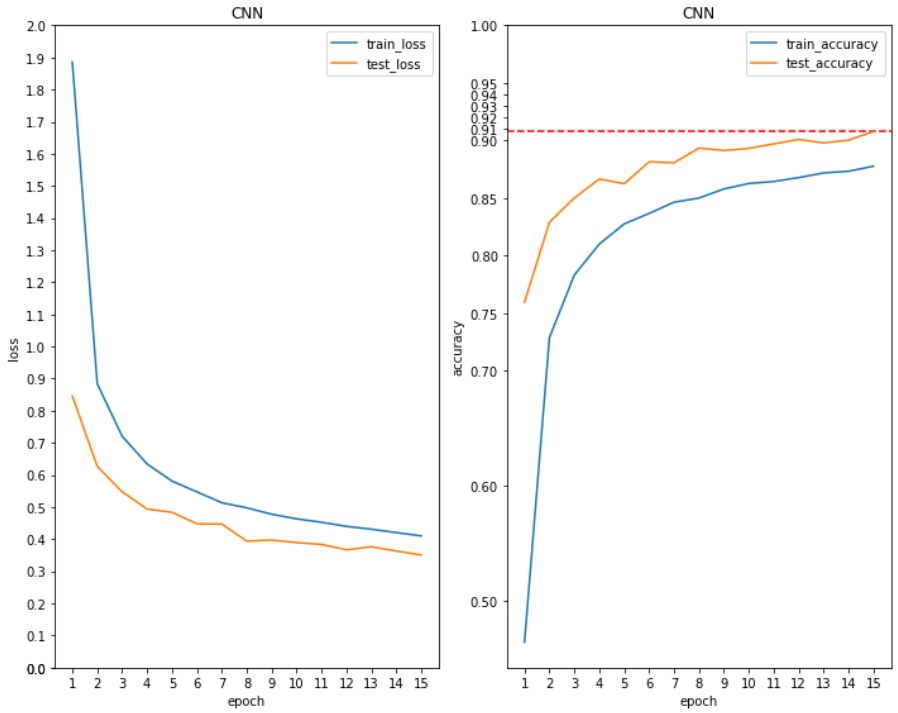
```
cnn = models.Sequential()
cnn.add(Input(shape=(32, 32, 3)))
cnn.add(layers.Conv2D(32, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn.add(layers.Flatten())
cnn.add(layers.Dropout(0.5))
cnn.add(layers.Dense(10, activation="softmax"))
```



Model iteration:

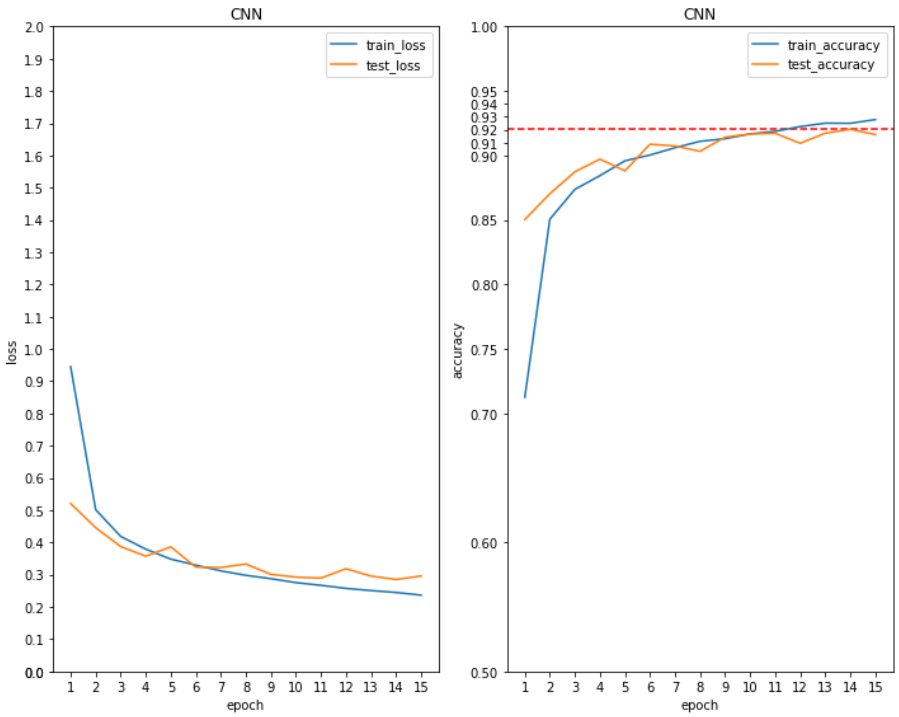
From the result of initial CNN model, we can see the test loss didn't decrease constantly after epoch 5 but not on the train set, so I guess it might have problem of overfitting on train set. Since the train accuracy cannot exceed 0.85 after 15 epochs, this probably because we didn't exact the image feature sufficiently. As a result, I add a convolution layer after the last one filter size 128, followed by 2*2 max pooling, which will help CNN model extract more features. The new model with its visualized results is shown below:

```
cnn = models.Sequential()
cnn.add(Input(shape=(32, 32, 3)))
cnn.add(layers.Conv2D(32, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.Flatten())
cnn.add(layers.Dropout(0.5))
cnn.add(layers.Dense(10, activation="softmax"))
```



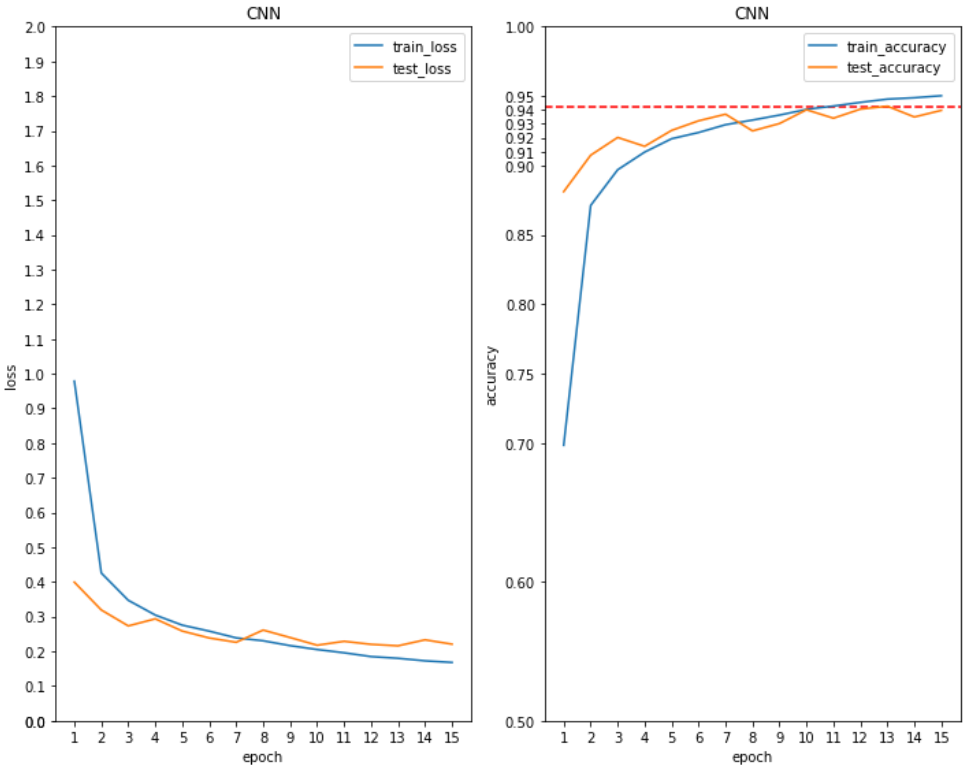
From the result above, we can see the adding convolutional layer successfully promote the performance on feature extraction, that both train loss and test loss decreases significantly. The maximum test accuracy is 0.9064 on epoch 15. I observed that the main reason now might not be the overfitting, since there's only slight difference between test loss and train loss as shown in the above plot. Focusing on feature extraction, I added batch normalization layer after each convolutional layer this time and observed a better performance on feature extraction. The visualized result is shown below:

```
cnn = models.Sequential()
cnn.add(Input(shape=(32, 32, 3)))
cnn.add(layers.Conv2D(32, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Flatten())
cnn.add(layers.Dropout(0.5))
cnn.add(layers.Dense(10, activation="softmax"))
```



During the first two iterations, I find that adding convolutional layers and batch normalization have significant effects on promoting feature extraction's performance. With this idea, I added more layers to let the CNN model convolute slowly, and get my final CNN model. The visualized result is shown below:

```
cnn = models.Sequential()
cnn.add(Input(shape=(32, 32, 3)))
cnn.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Dropout(0.2))
cnn.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Conv2D(64, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Dropout(0.2))
cnn.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Conv2D(128, (3, 3), activation='relu'))
cnn.add(layers.MaxPooling2D((2, 2)))
cnn.add(layers.BatchNormalization())
cnn.add(layers.Flatten())
cnn.add(layers.Dropout(0.5))
cnn.add(layers.Dense(10, activation="softmax"))
```



Above is my final CNN model and the test accuracy reaches 0.94 after 10 epochs. Besides the technics I used in previous iteration, I also use same padding on the first convolutional layer to extract more features, and apply dropout on convolutional layers to prevent overfitting. I create a table to record the results of the iterations above corresponding to their evaluation:

Model	Train Loss	Train Accuracy	Test Loss	Test Accuracy
Initial	0.49	0.847	0.61	0.845
Iteration 1	0.45	0.87	0.37	0.906
Iteration 2	0.27	0.93	0.33	0.921
Final	0.1680	0.9502	0.2207	0.9397

The total number of parameters is high because of the large number of layers, so we'd better need GPU to implement it.

I tried different combination of dropout layers, batch normalization and adding convolutional layers with new filter sizes, but find difficulties in increasing the test accuracy to 0.95. I believe one way to further increase the accuracy is adding part in data preprocessing such as image segmentation before CNN model training.