Lab 1 (Part 2)

Introduction

Convolutional Networks work by moving small filters across the input image. This means the filters are re-used for recognizing patterns throughout the entire input image. This makes the Convolutional Networks much more powerful than Fully-Connected networks with the same number of variables. This in turn makes the Convolutional Networks faster to train.

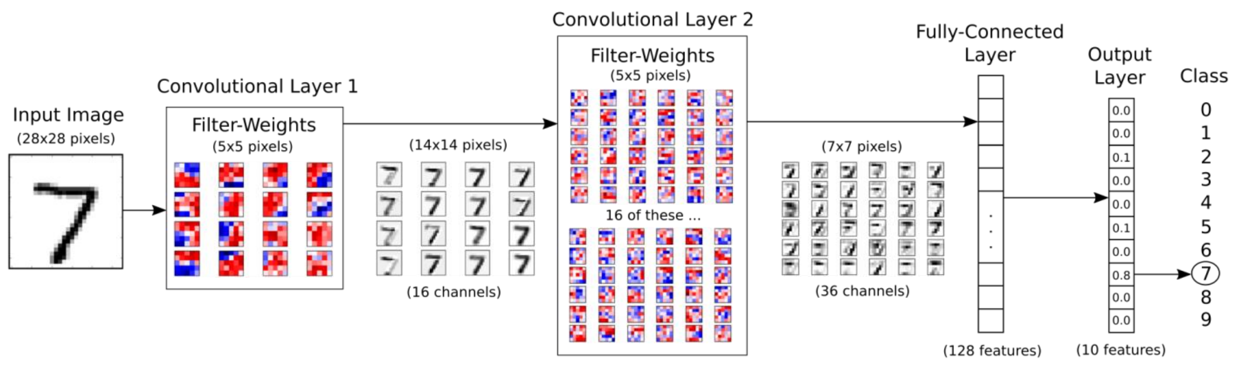
Libraries

In this project I have used following libraries:
1. Tensorflow
2. Numpy
3. Sklearn
4. Keras
5. Opencv(cv2)

And the program is written in python 3.6. I think you need to install opencv(cv2) manually.

Flowchart

The following chart shows roughly how the data flows in the Convolutional Neural Network that is implemented below.



(this picture is just for demo purpose)

The input image is processed in the first convolutional layer using the filter-weights. This results in 16 new images, one for each filter in the convolutional layer. The images are also down-sampled so the image resolution is decreased from 28x28 to 14x14.

These 16 smaller images are then processed in the second convolutional layer. We need filter-weights for each of these 16 channels, and we need filter-weights for each output channel of this layer. There are 36 output channels so there are a total of 16 x 36 = 576 filters in the second convolutional layer. The resulting images are down-sampled again to 7x7 pixels.

The output of the second convolutional layer is 36 images of 7x7 pixels each. These are then flattened to a single vector of length 7 x 7 x 36 = 1764, which is used as the input to a fully-connected layer with 128 neurons (or elements). This feeds into another fully-connected layer with 10 neurons, one for each of the classes, which is used to determine the class of the image, that is, which number is depicted in the image.

The convolutional filters are initially chosen at random, so the classification is done randomly. The error between the predicted and true class of the input image is measured as the so-called cross-entropy. The optimizer then automatically propagates this error back through the Convolutional Network using the chain-rule of differentiation and updates the filter-weights so as to improve the classification error. This is done iteratively thousands of times until the classification error is sufficiently low.

These particular filter-weights and intermediate images are the results of one optimization run and may look different if you re-run this Notebook.

Note that the computation in TensorFlow is actually done on a batch of images instead of a single image, which makes the computation more efficient. This means the flowchart actually has one more data-dimension when implemented in TensorFlow.

Code Explanation

First need to give the path of the data.

```
data_path = PATH + '/data'
```

Then need the set the image rows and columns. And the we also need to define how many iteration we need

```
img_rows=128
img_cols=128
num_channel=1
num_epoch=20
```

We also need to define the class or in other words number of training objects. For our lab this number should be 14.

```
num_classes = 14
```

Next we have to process the images for the training. So we loop through the file path and process all the data we have.

After that we need to set labels.

```
labels[0:255]=1
labels[255:510]=2
. . . . . . . . . .
```

Then give names to the training objects.

```
names = [...,'guo', 'jia',.., 'zhi']
```

In order to get rid of encoding problem I used pinyin name instead of hanzi.

Now we need to add all these thing to the model for training.

#Training

```
hist = model.fit(X_train, y_train, batch_size=16, nb_epoch=num_epoch, verbose=1,
validation_data=(X_test, y_test))
```

The following code gives the result for a test image.

```
print(model.predict(test_image))
print(model.predict_classes(test_image))
```

We can also test new image also,

```
test_image = cv2.imread('/1/1.bmp')
. . . . . . . . . .
```

If everything runs fine the three figure will appear. And in the console the result will be visible. To get better performance it is recommended to set the *num_epoch to 20.*

Conclusion

We have seen that a Convolutional Neural Network works much better at recognizing hand-written digits than the simple linear model. The Convolutional Network gets a classification accuracy of about 99%, or even more if you make some adjustments.

(NB- Since I couldn’t access the ftp I submitted the lab on 30/10/2017. Please don’t conside it as late submission.)