Title	: HW2 for ECE 281
Name	: Experiment with using neural networks for a recognition task
Team	: Rahul Vishwakarma
Date	: 02 Mar 2018

Objective

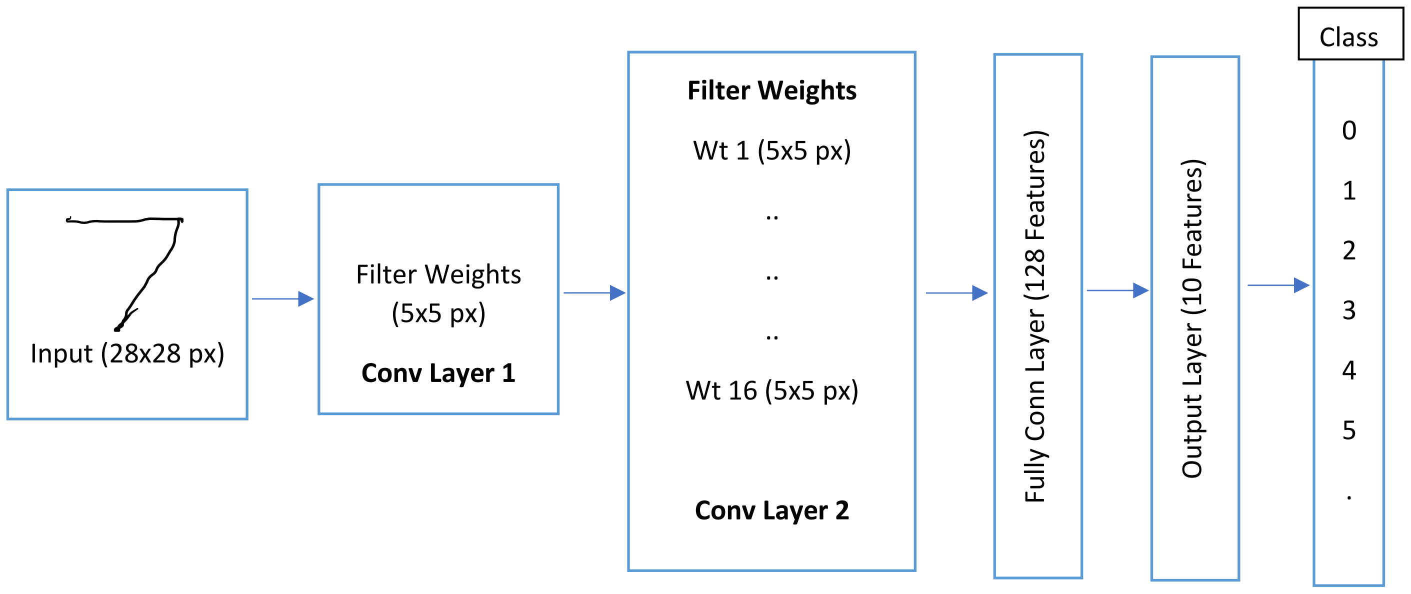
Demonstrate components, process, perform training (with proper batching and validating) and estimate performance (e.g., error rate vs. run time and iterations, resource usage statistics)

Dataset

MNIST http://yann.lecun.com/exdb/mnist

Approach

Used CNN with a choice of number of blocks, number of layers, number of neurons per layer. We know that RNN involves feedback and since we do not see any sequence-based prediction in use we stick to CNN. RNN is favorable if we were dealing with NLP datasets.



Flowchart for the CNN processing in a pure classification task

Execution

``` $ python cnn_recog_mnist.py ```

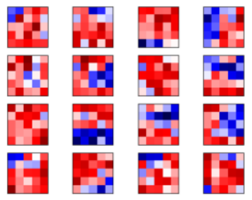
Output can be seen in the attached log file cnn.log

Description

In the given dataset CNN is the best approach since we are looking at each image individually and there is no relation between two images in sequence or batch.

Steps to create CNN Network Graph:

1. 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 from 28x28 to 14x14.



2. These 16 smaller images are processed in the convolutional layer 2. There are 36 output channels so there are a total of 16 x 36 = 576 filters in the second convolutional layer. Images are down-sampled again to 7x7 pixels.
3. The output of the second convolutional layer is 36 images of 7x7 pixels each. These are 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.
4. This feeds into another fully-connected layer with ***10 neurons***, which is used to determine the class of the image.
5. Each convolution layer is operates a 2D convolution, Max Pooling and ReLu function on the input image pixels.

Configuration

The graph is configured and setup before any execution in tensor flow session. Weight is configured as a normal distribution with std deviation as 0.05. Bias is configured to be a constant with value 0.05.

```
# Convolutional Layer 1.
filter_size1 = 5          # Convolution filters are 5 x 5 pixels.
num_filters1 = 16         # There are 16 of these filters.

# Convolutional Layer 2.
filter_size2 = 5          # Convolution filters are 5 x 5 pixels.
```

```
num_filters2 = 36          # There are 36 of these filters.

# Fully-connected layer.
fc_size = 128              # Number of neurons in fully-connected layer.
```

The MNIST data is separated into Training/Test & validation as below

```
Training-set   :       55000
Testing-set    :       10000
Validation-set :       5000

# We know that MNIST images are 28 pixels in each dimension.
img_size = 28

# Number of classes, one class for each of 10 digits.
num_classes = 10
```

Measure

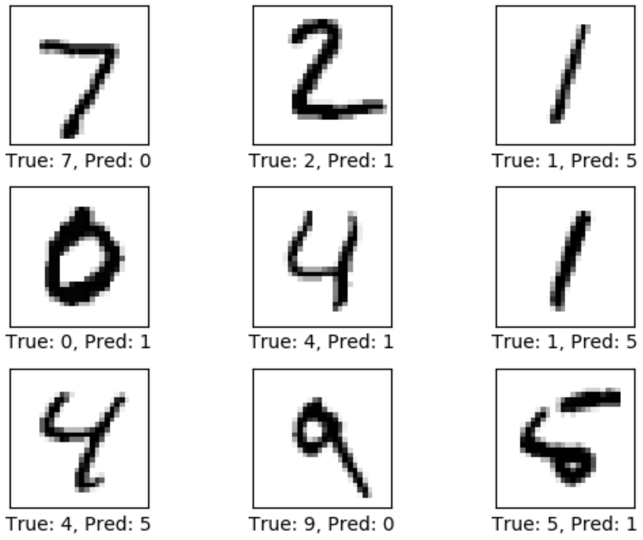
1. TensorFlow has a built-in function for calculating the cross-entropy which is a performance measure used in classification. If the predicted output of the model exactly matches the desired output then the cross-entropy equals zero. Hence the goal of optimization is to minimize cross entropy.
2. We use AdamOptimizer here which is an advanced from of Gradient Decent with *learning rate=**1e-4***
3. Now, we write the optimize function to train the batches of data. The **batch size is 64** and the progress is printed every 100 iterations. In case the validation accuracy is better than ever then we **save this network** in the checkpoints folder.
4. Upon training we run the test and validation data against the best performing network that is saved.
5. A dropout is implemented when we observe **no improvement for 10** consecutive accuracy values.

Performance

Here we can see the results for the 10000 iterations and we compare the provided data set without the training and with the trained network.

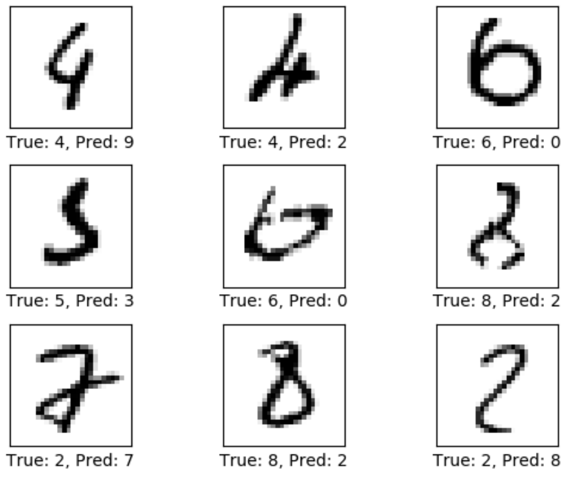
Let’s look at the error rates and the corresponding images that were misclassified.

```
Accuracy on Vald-Set: 10.1% (503 / 5000)
Accuracy on Test-Set: 9.9% (992 / 10000)
```



Misclassifications/Errors without Training

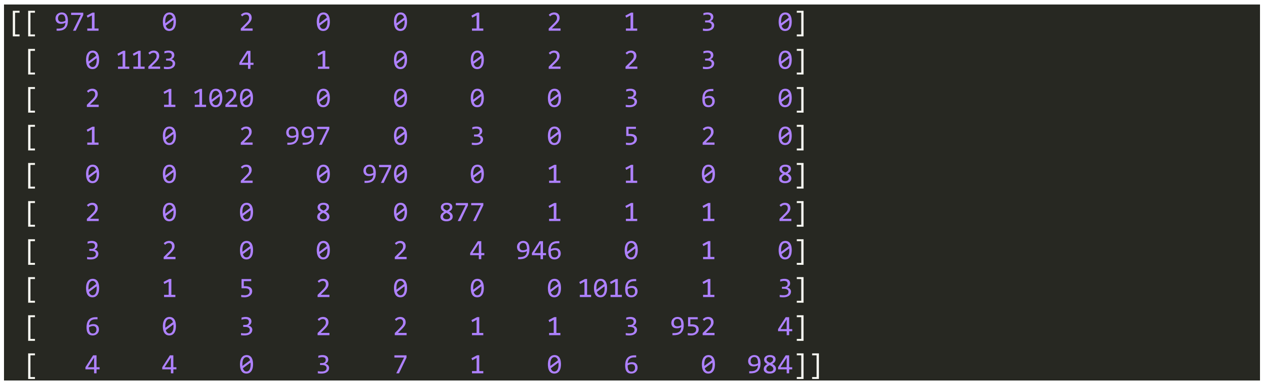
```
Accuracy on Vald-Set: 98.7% (4933 / 5000)
Accuracy on Test-Set: 98.6% (9856 / 10000)
```



Misclassifications/Errors with training (10000 iterations)

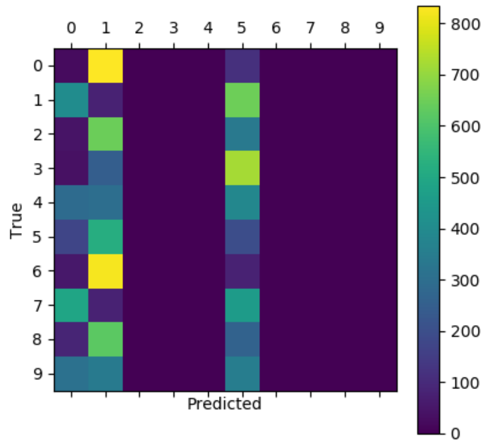
Improvement seen here since images are a little obscure for good classification.

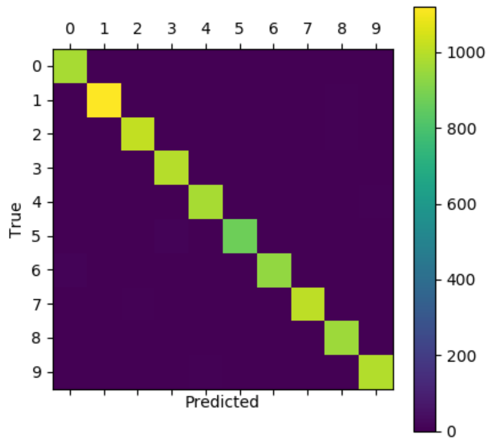
We finally see the Confusion Matrix to see how well the values in the diagonal matrix are well separated from others.



```
[[ 971    0    2    0    0    1    2    1    3    0]
 [   0 1123    4    1    0    0    2    2    3    0]
 [   2    1 1020    0    0    0    0    3    6    0]
 [   1    0    2  997    0    3    0    5    2    0]
 [   0    0    2    0  970    0    1    1    0    8]
 [   2    0    0    8    0  877    1    1    1    2]
 [   3    2    0    0    2    4  946    0    1    0]
 [   0    1    5    2    0    0    0 1016    1    3]
 [   6    0    3    2    2    1    1    3  952    4]
 [   4    4    0    3    7    1    0    6    0  984]]
```

The Confusion Matrix improved from image on the left to the one on the right





1. **Before training**

2. **After training**

Overall CNN is a in this architecture performs well but it can improve if we use a known set of weights early on rather than the random normals.