assignment2

November 15, 2017

Name: Qin Qianqian Student Number: 601098

1 Exercise on Convolutional Neural Networks

In this exercise you will extend and improve the accuracy of the Convolution Neural Network discussed in the demonstration by doing the following things:

Creating a deep Convolutional Neural Network.

Tweaking the hyper-parameters such as the number of channels in each layer, activation functions, mini batch size, etc;

The architecture of the network that we will create is similar to the figure below (This figure will not be displayed when exported to .pdf):

We will begin by first importing the necessary python libraries:

```
In [1]: import os
        import six.moves.cPickle as pickle
        import gzip
        import numpy
        import theano
        import theano.tensor as T
        import matplotlib.pyplot as plt
        import random as rd
        from theano.tensor.signal import pool
        from theano.tensor.nnet import conv2d
        from exercise_helper import load_data, pooling, convLayer
        from exercise_helper import fullyConnectedLayer
        from exercise_helper import negative_log_lik, errors
        from exercise_helper import generate_plot
        %matplotlib inline
        # Setting the random number generator
        rng = numpy.random.RandomState(23455)
        print('***** Import complete *****')

***** Import complete *****
```

Note: If the import of 'exercise_helper' fails, ensure that 'exercise_helper.py' is in the same folder as this notebook.

Now we can create the Theano Computation Graph. We shall partition the data into mini-batches and then create a computation graph for training, validation and testing. The overall logic is very similar to the demonstration. Refer to the demonstration for more details.

```
In [2]: def train_test_conv_net(learning_rate, num_epochs,
                                num_filters, mini_batch_size, activation):
            # Function to create the convolutional neural network, train and
            # evaluate it. This function must be called to run the network.

            # Inputs:
            # learning_rate - Learning rate for Stochastic Gradient Descent
            # num_epochs - Number of training epochs
            # num_filters - Number of kernels for each convolution layer
            #               for e.g. 2 layers - [20, 50].
```

```
#.              layer1 = 20, layer2 = 50
# mini_batch_size - Mini-batch size to be used
# activation - Activation function to use

# Outputs:
# Plot of the cost, prediction errors on validation set and
# visualisation of weights of the first convolutional layer

# Partitioning into mini- batches
n_train_batches = train_set_x.get_value(borrow=True).shape[0]
n_valid_batches = valid_set_x.get_value(borrow=True).shape[0]
n_test_batches = test_set_x.get_value(borrow=True).shape[0]
n_train_batches //= mini_batch_size
n_valid_batches //= mini_batch_size
n_test_batches //= mini_batch_size

print('train: %d batches, test: %d batches,'
      ' validation: %d batches'
      % (n_train_batches, n_test_batches, n_valid_batches))

mb_index = T.lscalar() # mini-batch index
x = T.matrix('x') # rasterised images
y = T.ivector('y') # image labels
layer_weights = [];
print('***** Constructing model ***** ')

# Reshaping matrix of mini_batch_size set of images into a
# 4-D tensor
layer0_input = x.reshape((mini_batch_size, 1, 28, 28))

################# Insert Your Code Here ##################

# Construct first convolution and pooling layer
# Hint: Use the convLayer function. See demonstration.
[layer0_output, layer0_params] = convLayer(
rng,
data_input=layer0_input,
image_spec=(mini_batch_size, 1, 28, 28),
filter_spec=(num_filters[0], 1, 5, 5),
pool_size=(2, 2),
activation=activation)

# Construct second convolution and pooling layer
# Hint: Use the convLayer function. See demonstration.
[layer1_output, layer1_params] = convLayer(
rng,
data_input=layer0_output,
image_spec=(mini_batch_size, num_filters[0], 12, 12),
```

```
filter_spec=(num_filters[1], num_filters[0], 5, 5),
pool_size=(2, 2),
activation=activation)

# Classify the values using the fully-connected
# activation layer.
# Hint: Remember to flatten the output from the
# convolutional layer. Use the fullyConnectedLayer function.
# See demonstration.

fc_layer_input = layer1_output.flatten(2)

[p_y_given_x, y_pred, fc_layer_params] = fullyConnectedLayer(
data_input=fc_layer_input,
num_in=num_filters[1]*4*4,
num_out=10)


##########################################################

# Cost that is minimised during stochastic descent.
cost = negative_log_lik(y=y, p_y_given_x=p_y_given_x)

# Creating a function that computes the mistakes on the test set
# mb_index is the mini_batch_index
test_model = theano.function(
    [mb_index],
    errors(y, y_pred),
    givens={
        x: test_set_x[
            mb_index * mini_batch_size:
            (mb_index + 1) * mini_batch_size],
        y: test_set_y[
            mb_index * mini_batch_size:
            (mb_index + 1) * mini_batch_size]})

# Creating a function that computes the mistakes on the validation
# set
valid_model = theano.function(
    [mb_index],
    errors(y, y_pred),
    givens={
        x: valid_set_x[
            mb_index * mini_batch_size:
            (mb_index + 1) * mini_batch_size],
        y: valid_set_y[
            mb_index * mini_batch_size:
            (mb_index + 1) * mini_batch_size]})
```

```
################ Insert Your Code Here ##################

# Create list of parameters to fit during training.
# Hint: Include the parameters from the two convolution layers
# and activation layer
if len(num_filters)==3:
    params = fc_layer_params+layer2_params+layer1_params+layer0_params
if len(num_filters)==2:
    params = fc_layer_params+layer1_params+layer0_params
if len(num_filters)==1:
    params = fc_layer_params+layer0_params

##########################################################

# Creating a list of gradients
grads = T.grad(cost, params)

# Creating a function that updates the model parameters by SGD.
# The updates list is created by looping over all
# params[i], grads[i]) pairs.
updates = [(param_i, param_i - learning_rate * grad_i)
           for param_i, grad_i in zip(params, grads)]

train_model = theano.function(
    [mb_index],
    cost,
    updates=updates,
    givens={
        x: train_set_x[
            mb_index * mini_batch_size:
            (mb_index + 1) * mini_batch_size],
        y: train_set_y[
            mb_index * mini_batch_size:
            (mb_index + 1) * mini_batch_size]})

epoch = 0
cost_arr = numpy.array([])
valid_score_arr = numpy.array([])
valid_score_arr = numpy.append(valid_score_arr, 1)

print('***** Training model *****')
if (num_epochs < 1):
    print("Too few epochs!")
    return
while (epoch < num_epochs):
    epoch = epoch + 1
    print("Training in epoch: %d / %d" % (epoch, num_epochs),
```

```
                    end='\r')
            for minibatch_index in range(n_train_batches):
                # Computing number of iterations performed or total number
                # of mini-batches executed.
                iter = (epoch - 1) * n_train_batches + minibatch_index
                # cost of each minibatch
                cost_ij = train_model(minibatch_index)
                cost_arr = numpy.append(cost_arr, cost_ij)

            # Computing loss on each validation mini-batch after each epoch
            valid_losses = [valid_model(i) for i in range(n_valid_batches)]
            valid_score_arr = numpy.append(
                                           valid_score_arr,
                                            numpy.mean(valid_losses))
        print('***** Training Complete *****')
        # Computing mean error rate on test set
        test_losses = [test_model(i) for i in range(n_test_batches)]
        test_score = numpy.mean(test_losses)
        print('Prediction error: %f %%' % (test_score * 100.))
        # Generating the plots
        generate_plot(cost_arr, range(1, iter+2),
                      valid_score_arr,
                      range(0, epoch+1),
                      layer0_params[0].get_value())
```

1.0.1 Experiment

Now we shall define some hyper-parameters and evaluate the model. We will compute the final Prediction Error on the test set. To complete this assignment, you must do the following things:

Set your parameters in the cell below

Run the experiment. Then, describe and discuss it in the 'Conclusions' cell below. You must do this for each experiment that you run.

Especially in CSC notebooks, do not forget to restart the kernel after each experiment.

Repeat from step 1 and perform at least 4 experiments.

Run the experiment with the best result again to display the plots in the final submission.

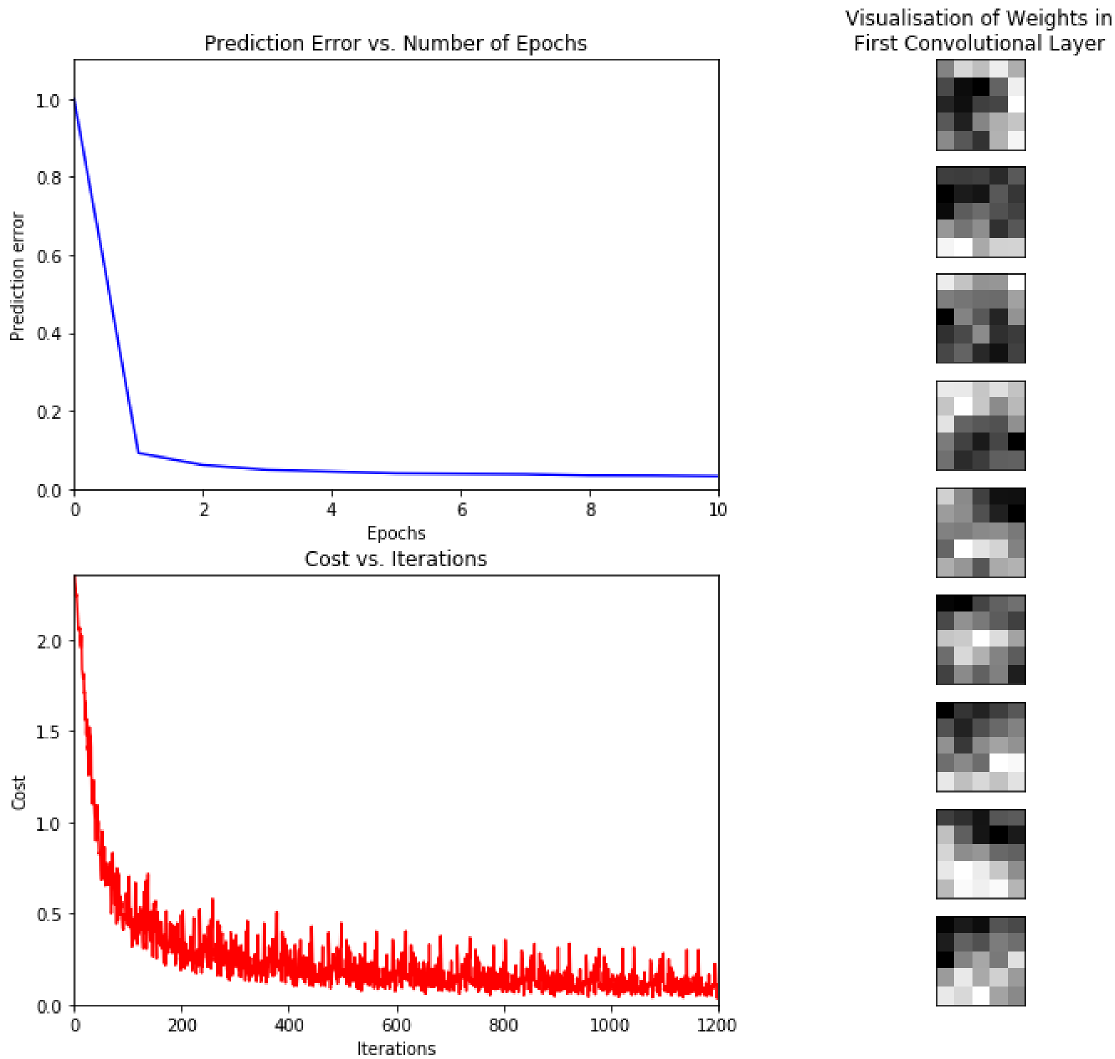
Download the completed assignment as PDF and submit as usual. The final PDF will contain the plots from only your best experiment as well as the discussion of all your experiments in the 'Conclusions' cell.

```
In [3]: ################# Insert Your Code Here ###################

        # Description and examples of parameters to train_test_conv_net()

        # learning_rate - Sets the learning rate for Stochastic Gradient
        #                 Descent. e.g.: 0.1
        # num_epochs - Sets the number of training epochs. E.g.: 10
        # num_filters - Sets the number of kernels for each
        #               convolution layer. for e.g. 2 layers: [4, 8]
```

```
        #                implies layer1 = 4, layer2 = 8
        # mini_batch_size - Sets the mini-batch size to be used in
        #                    the experiment. E.g: 50
        # activation - Sets the activation function to be used.
        #              E.g.: T.tanh
        # train_size - Sets the number of training samples to be used.
        #              E.g. 6000.

        # Note: The Kernel may crash on too large values of the
        # num_epochs, num_filters, mini_batch_size and train_size due to
        # memory limitations. This may happen especially on CSC notebooks
        # as it is a shared resource.
        # If that happens use smaller values!

        learning_rate   = 0.1
        num_epochs      = 10
        num_filters     = [9,9]
        mini_batch_size = 50
        activation      =T.tanh
        train_size      = 6000

        ###########################################################

        # Loading dataset
        # We use only a subset of the full dataset.
        # Validation and test sets will be 1/3 of train
        # set size
        datasets = load_data('mnist.pkl.gz', train_size)
        train_set_x, train_set_y = datasets[0]
        valid_set_x, valid_set_y = datasets[1]
        test_set_x, test_set_y = datasets[2]
        print('Training set: %d samples'
              %(train_set_x.get_value(borrow=True).shape[0]))
        print('Test set: %d samples'
              %(test_set_x.get_value(borrow=True).shape[0]))
        print('Validation set: %d samples'
              %(valid_set_x.get_value(borrow=True).shape[0]))

        # Beginning the training process
        train_test_conv_net(learning_rate, num_epochs,
                            num_filters, mini_batch_size, activation)
```

```
***** Loading data *****
Training set: 6000 samples
Test set: 2000 samples
Validation set: 2000 samples
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
```

```
***** Training model *****
***** Training Complete *****
Prediction error: 4.550000 %
```

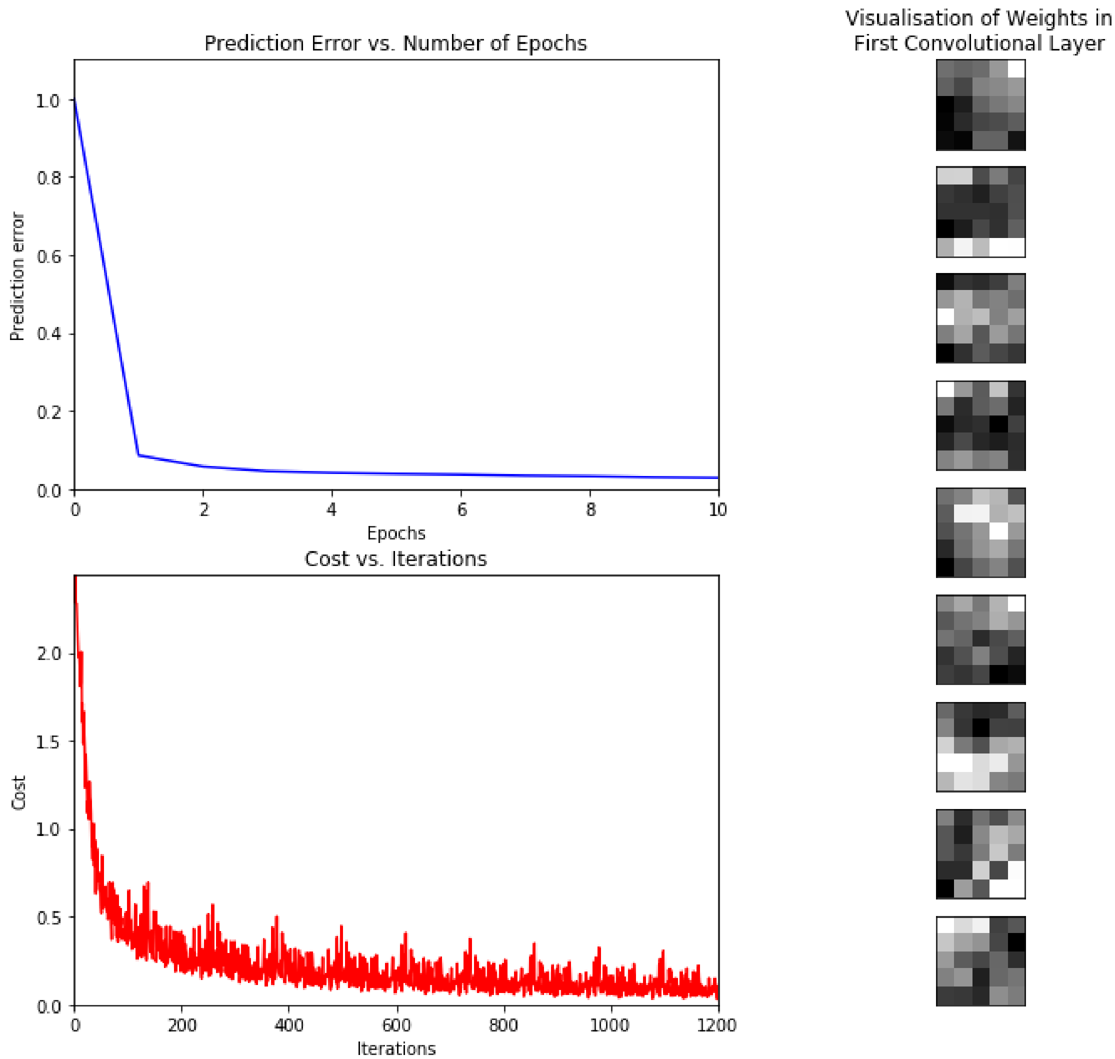


Now go ahead and experiment with different hyper-parameters such as different number of filters, different number of convolution layers, activation functions etc;. Try to find out which configuration gives the best accuracy.

```
In [4]: # experiment with different number of filters
        num_filters1      = [9,12]
        train_test_conv_net(learning_rate, num_epochs,
                            num_filters1, mini_batch_size, activation)
```

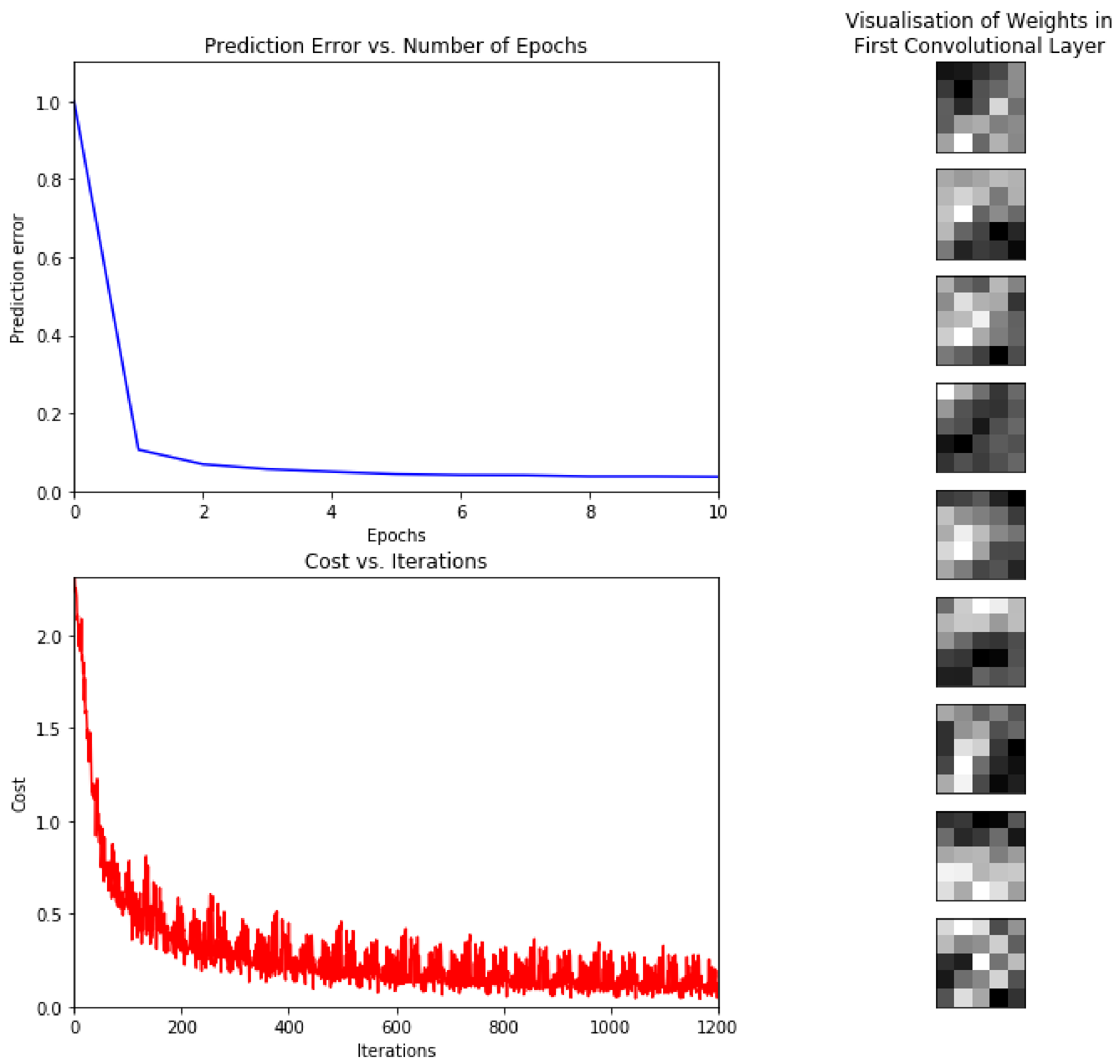
```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
```

```
***** Training model *****
***** Training Complete *****
Prediction error: 4.050000 %
```



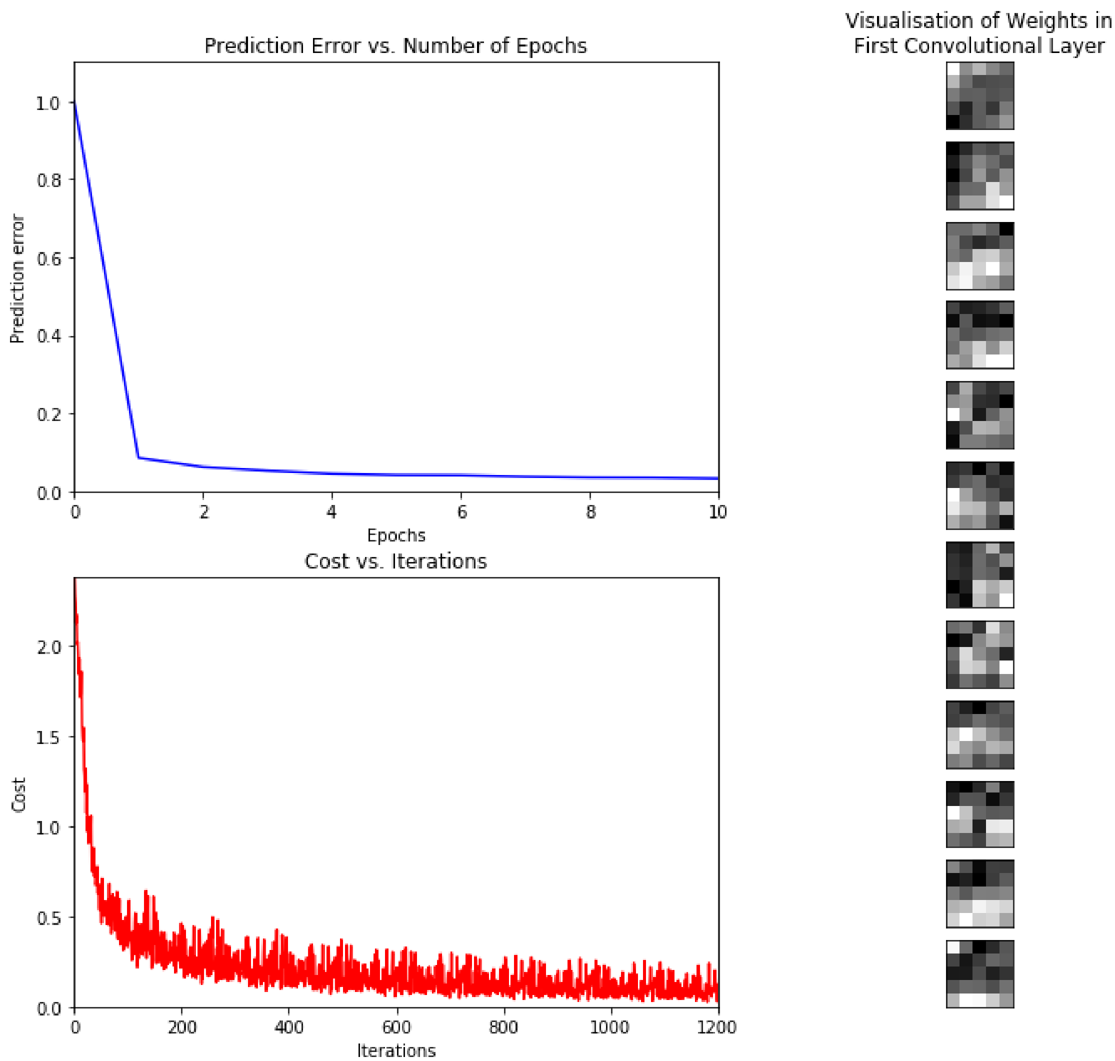

```
In [5]: num_filters2     = [9,6]
        train_test_conv_net(learning_rate, num_epochs,
                            num_filters2, mini_batch_size, activation)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 4.900000 %
```



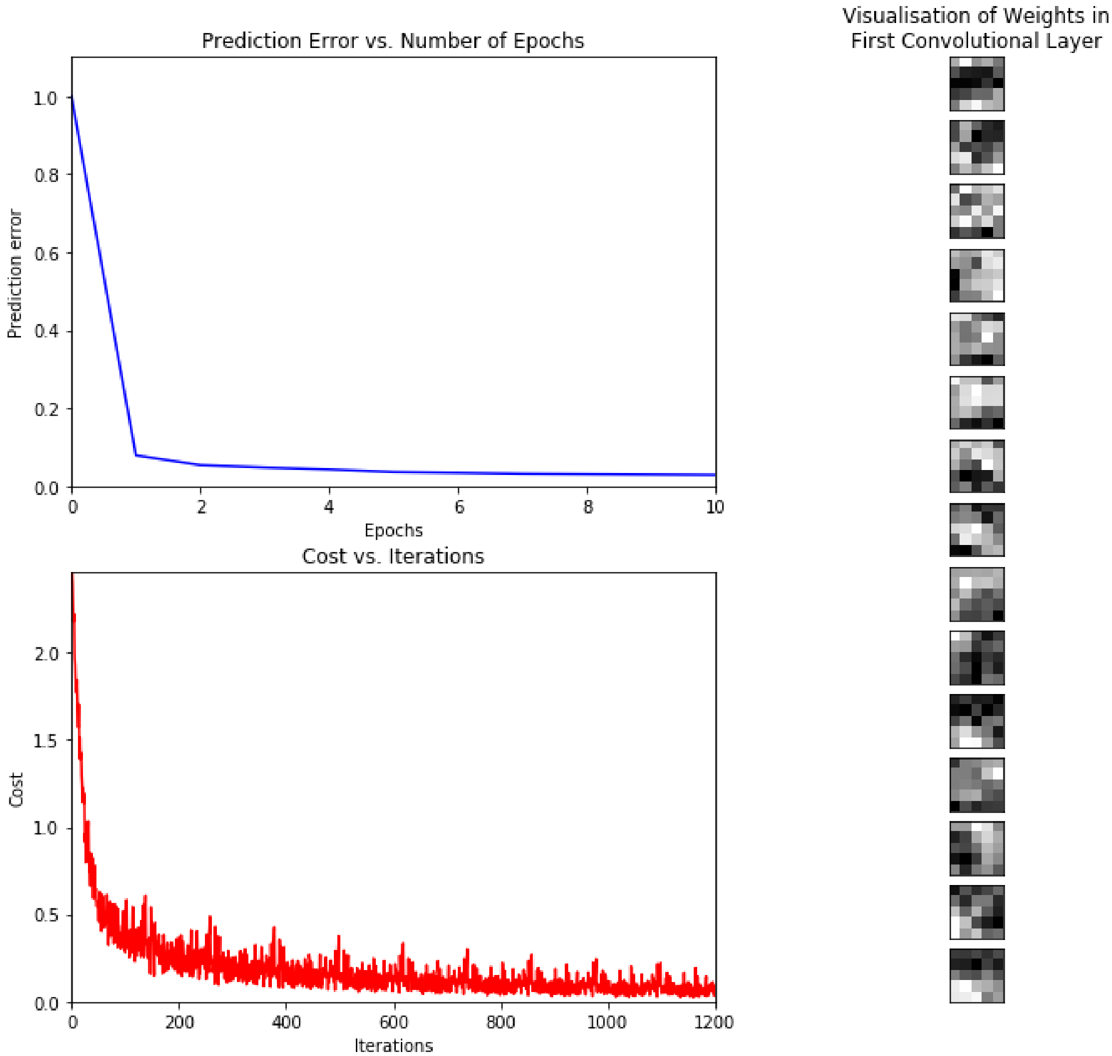

```
In [6]: num_filters3     = [12,12]
        train_test_conv_net(learning_rate, num_epochs,
                            num_filters3, mini_batch_size, activation)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 3.950000 %
```



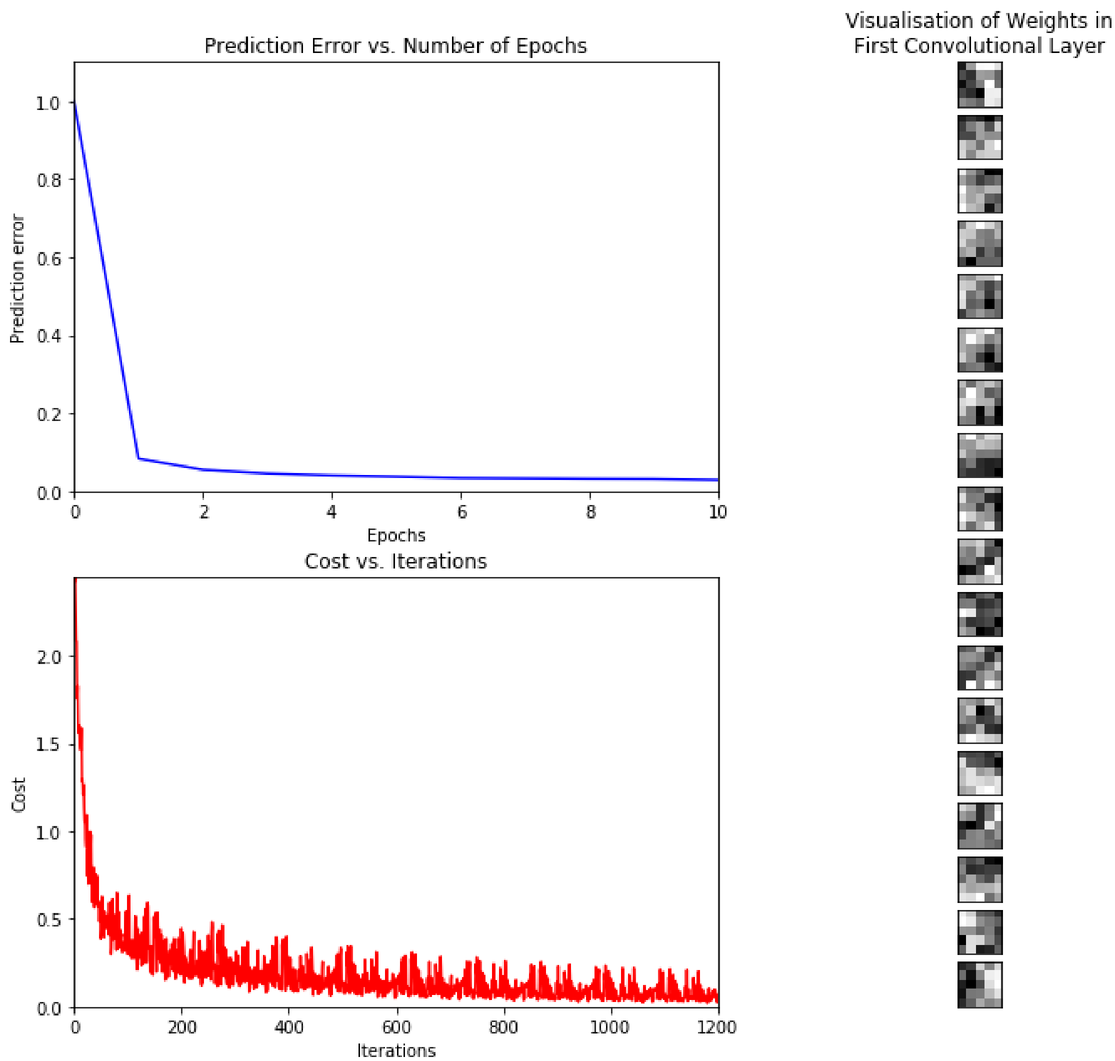

```
In [7]: num_filters4     = [15,15]
        train_test_conv_net(learning_rate, num_epochs,
                            num_filters4, mini_batch_size, activation)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 3.900000 %
```



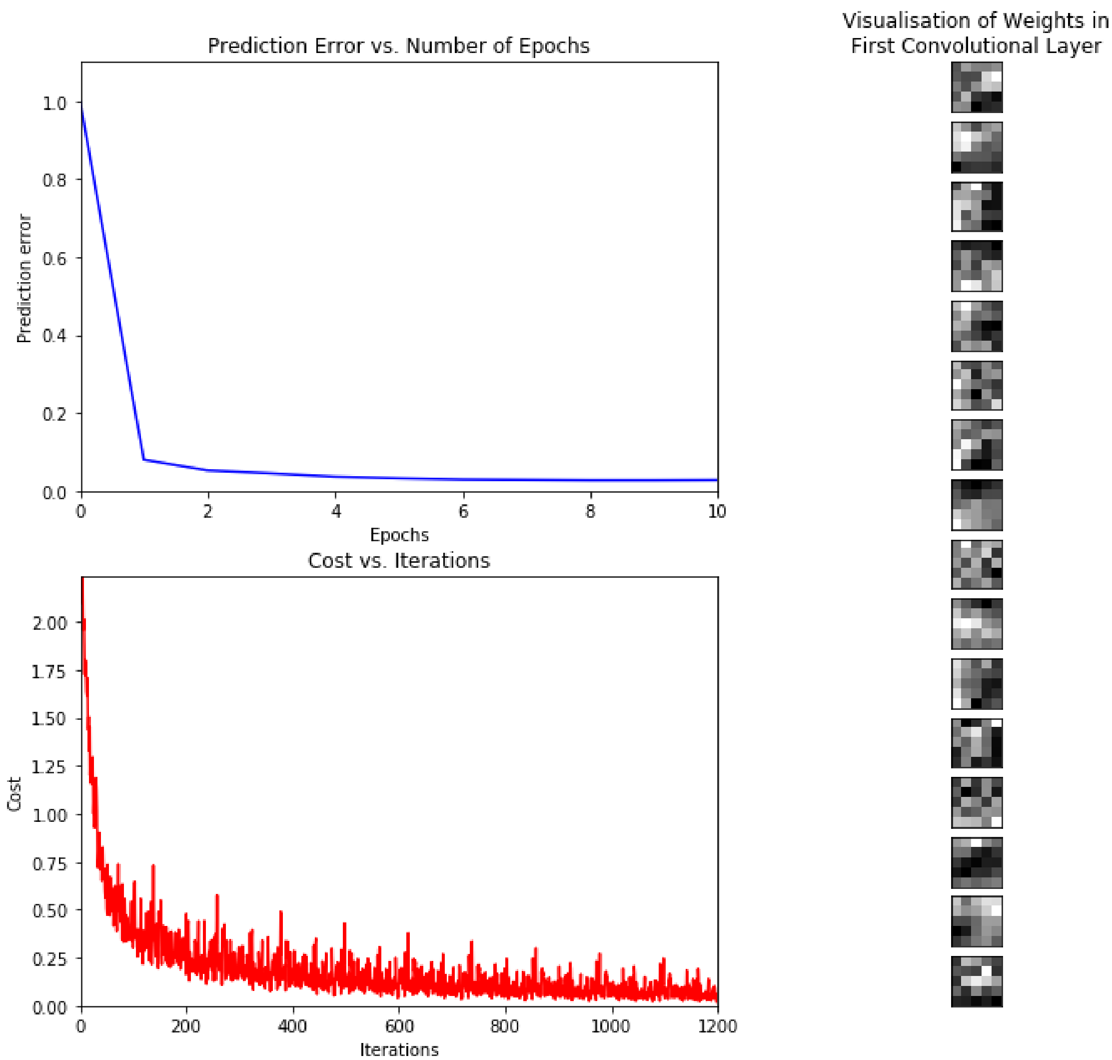

```
In [8]: num_filters5    = [18,18]
        train_test_conv_net(learning_rate, num_epochs,
                            num_filters5, mini_batch_size, activation)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 4.100000 %
```



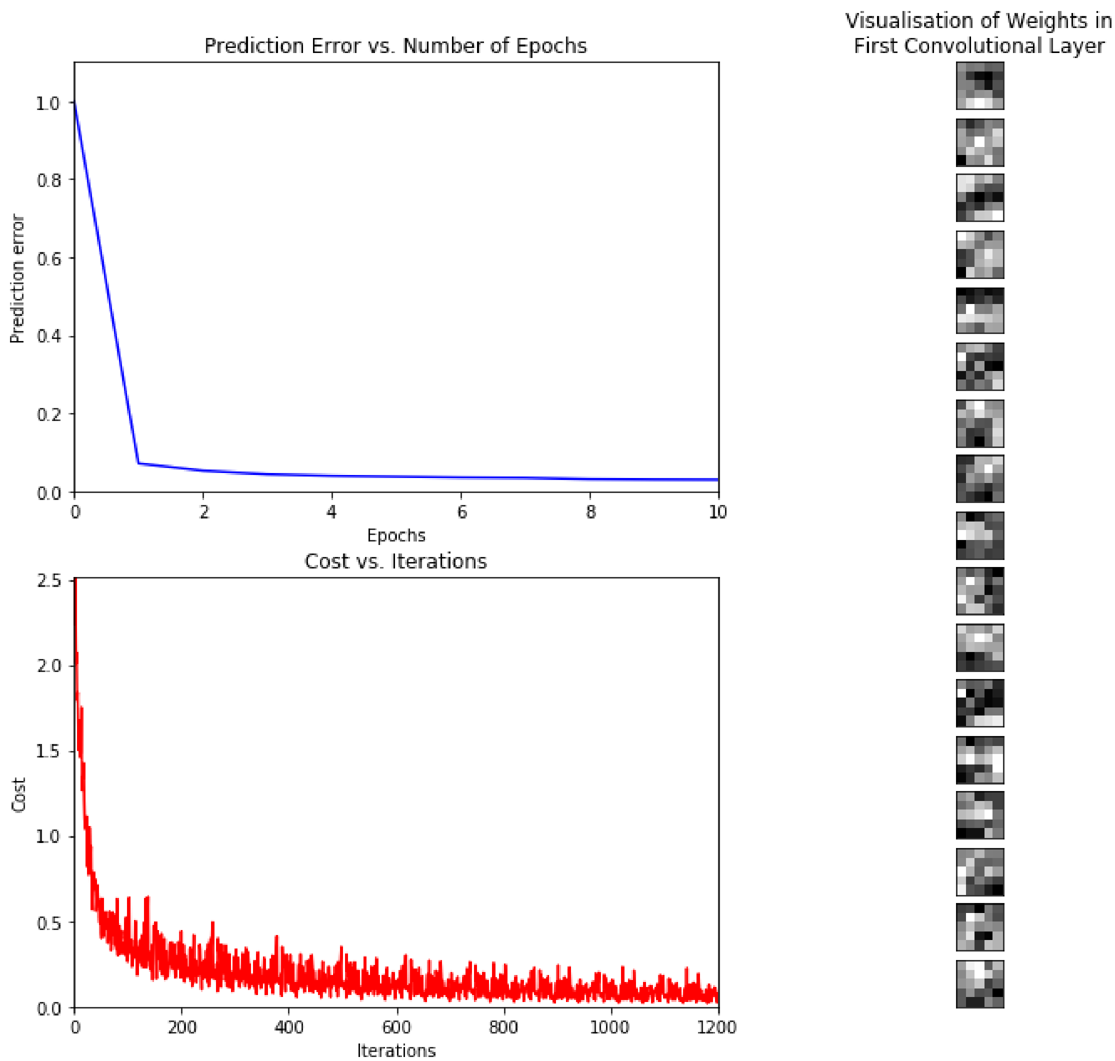

```
In [9]: num_filters6    = [16,16]
        train_test_conv_net(learning_rate, num_epochs,
                            num_filters6, mini_batch_size, activation)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 3.650000 %
```



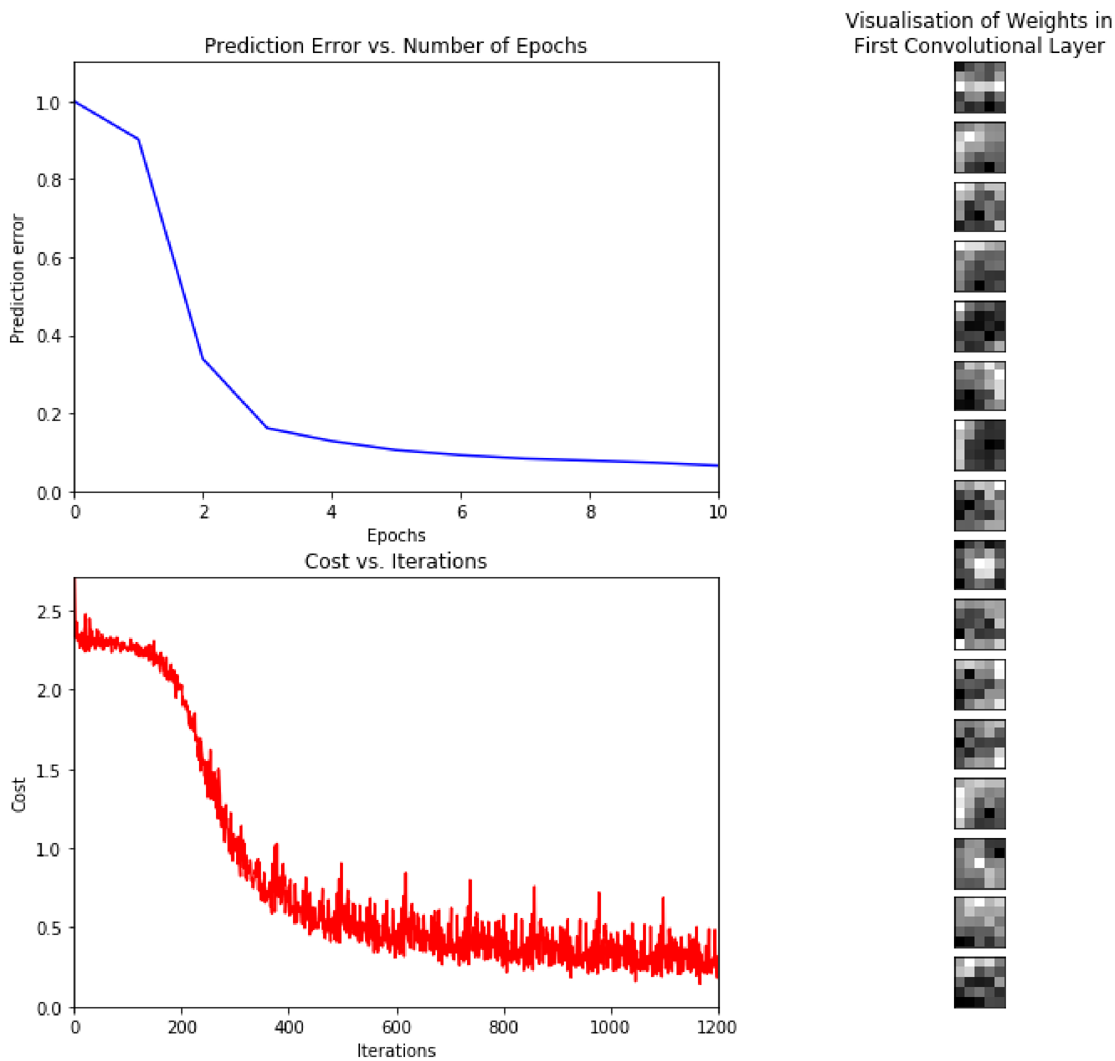

```
In [11]: num_filters7    = [17,17]
         train_test_conv_net(learning_rate, num_epochs,
                             num_filters7, mini_batch_size, activation)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 3.700000 %
```



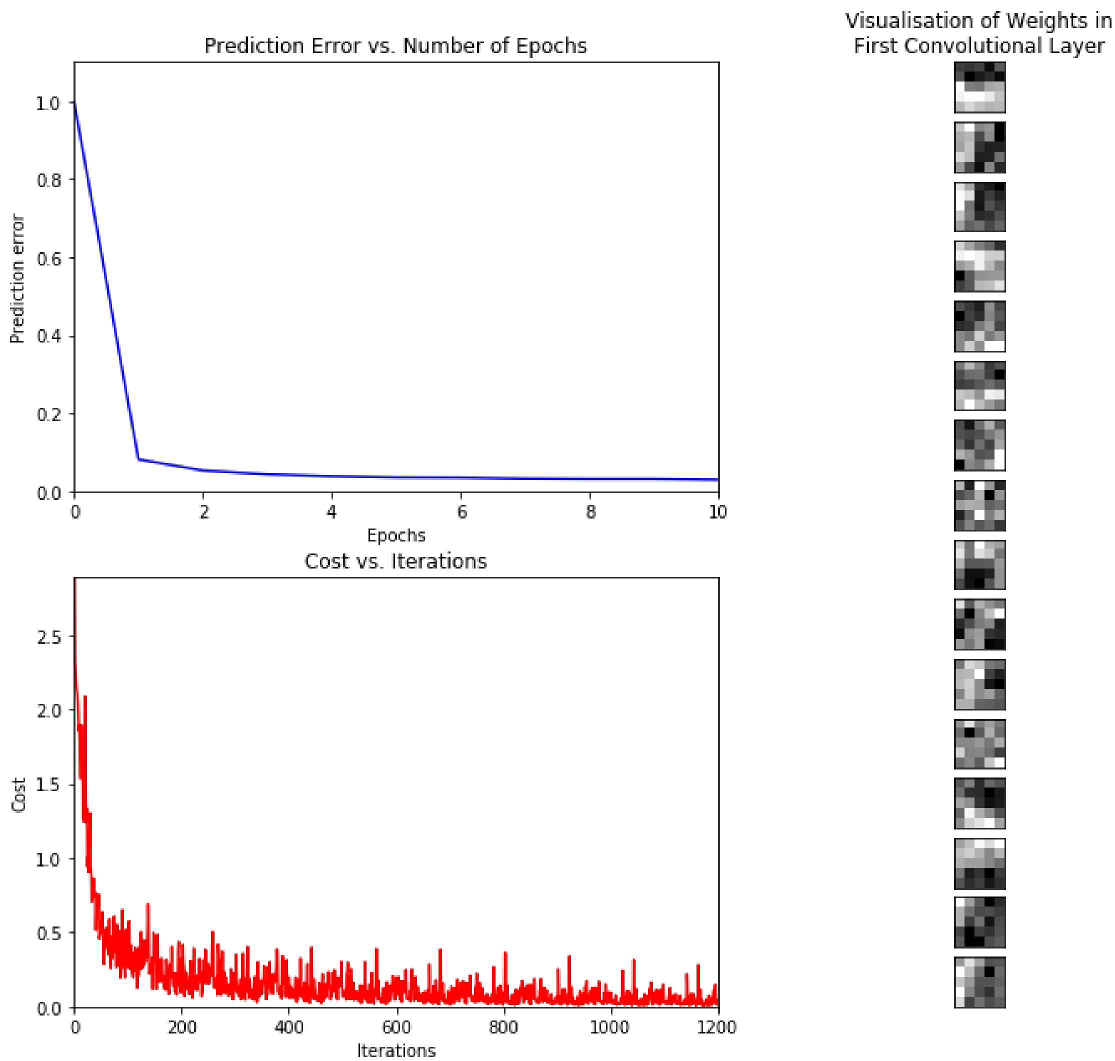

```
In [13]: # After testing several filters,  num_filters6 = [16,16] is choosen
         # Experiment with different activation functions
         activation2=T.nnet.sigmoid
         train_test_conv_net(learning_rate, num_epochs,
                             num_filters6, mini_batch_size, activation2)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 9.200000 %
```



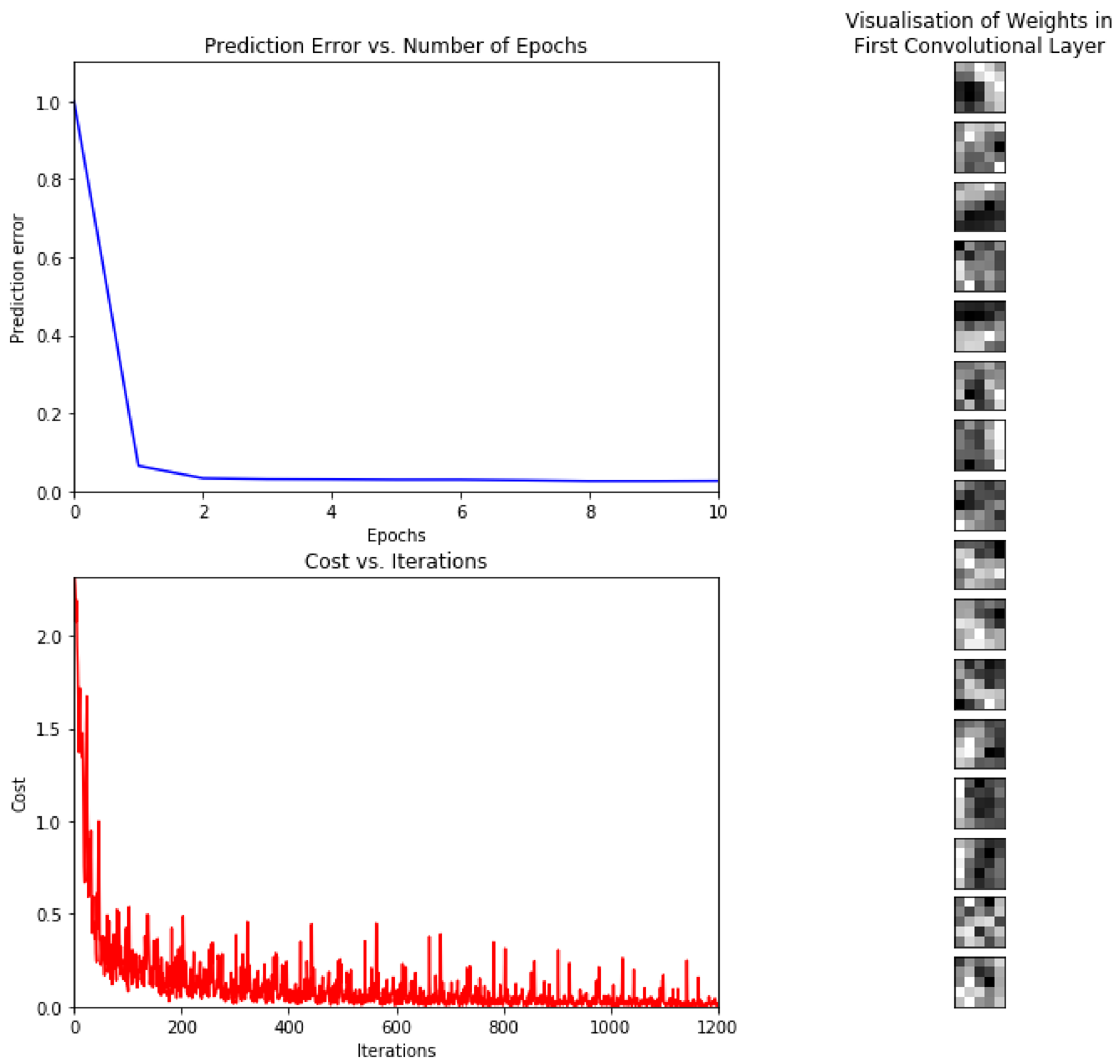

```
In [14]: activation3=T.nnet.relu
         train_test_conv_net(learning_rate, num_epochs,
                             num_filters6, mini_batch_size, activation3)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 3.350000 %
```



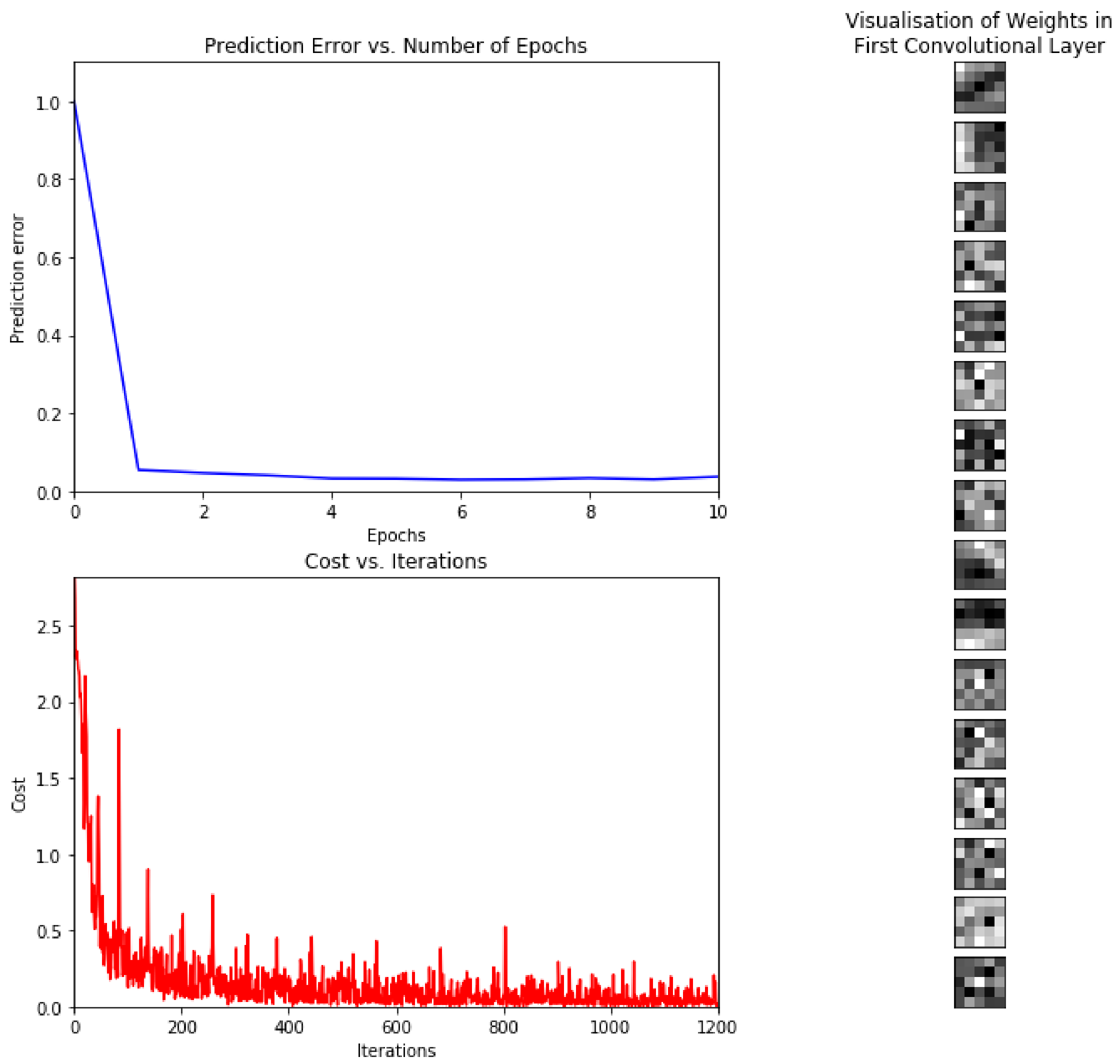
In [15]:
```
# T.nnet.relu should be choosen
# Experiment with different learning rate
learning_rate2 = 0.2
train_test_conv_net(learning_rate2, num_epochs,
                    num_filters6, mini_batch_size, activation3)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 2.600000 %
```



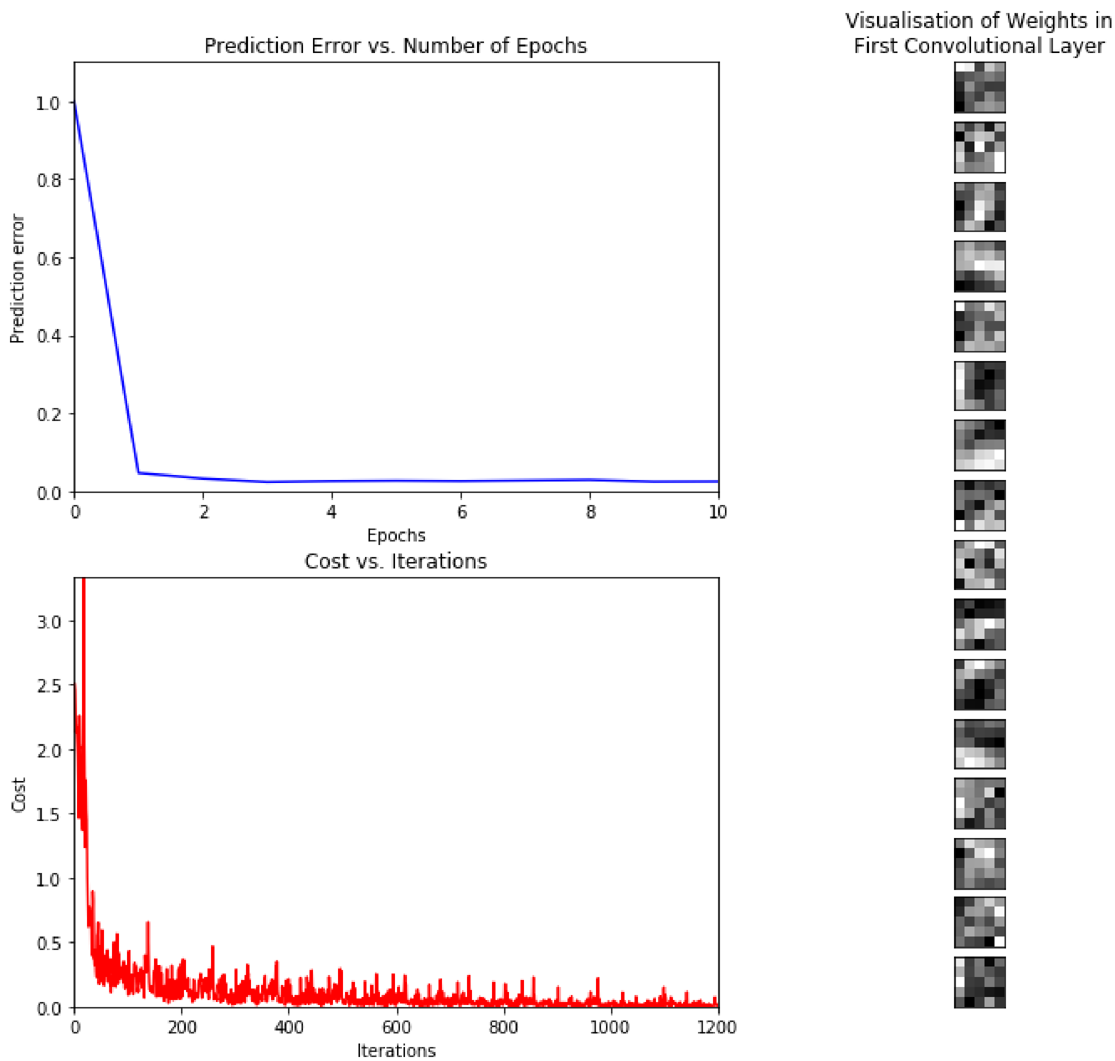

```
In [16]: learning_rate3 = 0.4
         train_test_conv_net(learning_rate3, num_epochs,
                             num_filters6, mini_batch_size, activation3)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 3.950000 %
```



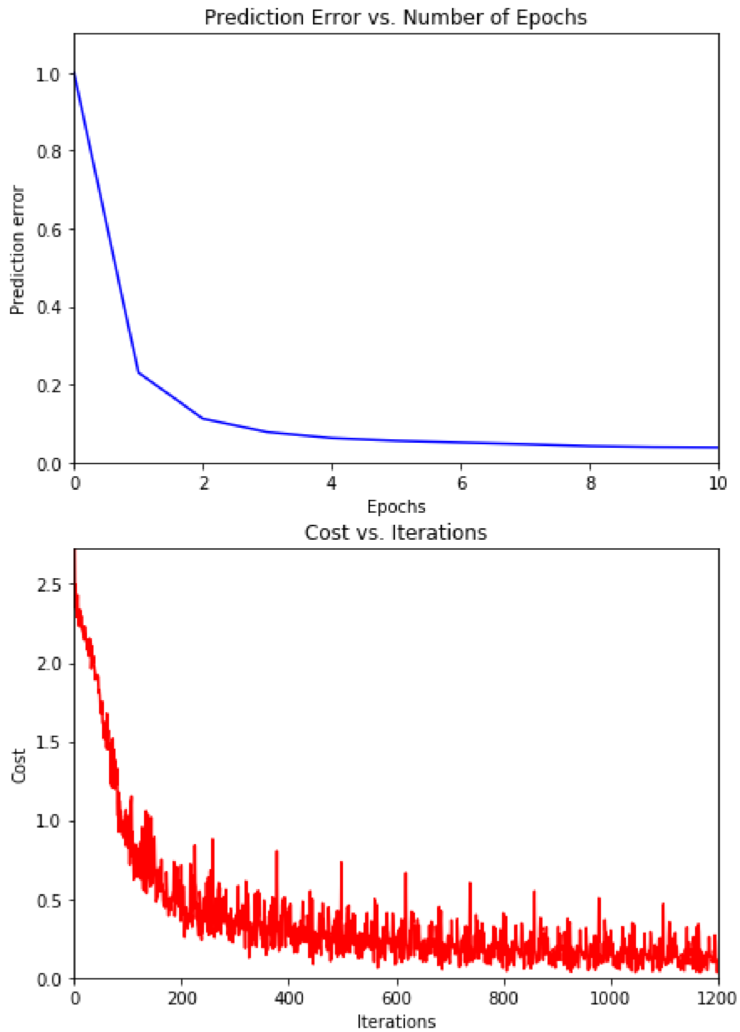

```
In [17]: learning_rate4 = 0.3
         train_test_conv_net(learning_rate4, num_epochs,
                             num_filters6, mini_batch_size, activation3)
```

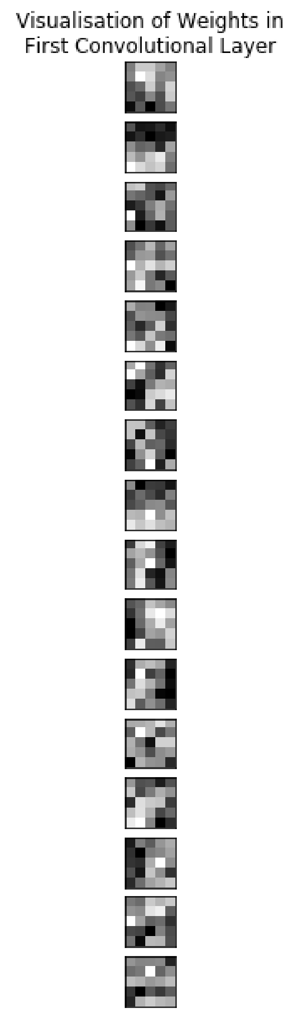
```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 2.550000 %
```




```
In [18]: learning_rate5= 0.02
         train_test_conv_net(learning_rate5, num_epochs,
                             num_filters6, mini_batch_size, activation3)
```

```
train: 120 batches, test: 40 batches, validation: 40 batches
***** Constructing model *****
***** Training model *****
***** Training Complete *****
Prediction error: 5.000000 %
```





1.0.2 Conclusions

After experimenting with different hyper-parameters, the following hyper-parameters gives the best accuracy among my experiments: learning_rate=0.3, num_epochs=10, num_filters=[16,16], mini_batch_size=50, activation function=T.nnet.relu

Smaller learning_rate or more filters does not mean better accuracy. To get best accuracy, we need to do searches many times. There are some search methods to select appropriate hyper-parameters.

Many activation function can be choosen. Deciding which one to choose needs analyzing and experimenting as well.

I notice that convergence curve does not change much when changing number of filters. It changes relative obviously when changing activation function and learning_rate.