

Deep Learning

Lab Session 2 - 3 Hours

Convolutional Neural Network (CNN) for Handwritten Digits Recognition

Student 1: CANALE **Student 2:** ELLENA

The aim of this session is to practice with Convolutional Neural Networks. Answers and experiments should be made by groups of one or two students. Each group should fill and run appropriate notebook cells.

Once you have completed all of the code implementations and successfully answered each question above, you may finalize your work by exporting the iPython Notebook as an pdf document using print as PDF (Ctrl+P). Do not forget to run all your cells before generating your final report and do not forget to include the names of all participants in the group. The lab session should be completed by May 29th 2017.

Send you pdf file to benoit.huet@eurecom.fr and olfa.ben-ahmed@eurecom.fr using **[DeepLearning_lab2]** as Subject of your email.

Introduction

In the last Lab Session, you built a Multilayer Perceptron for recognizing hand-written digits from the MNIST data-set. The best achieved accuracy on testing data was about 97%. Can you do better than these results using a deep CNN ? In this Lab Session, you will build, train and optimize in TensorFlow one of the early Convolutional Neural Networks: **LeNet-5** to go to more than 99% of accuracy.

Load MNIST Data in TensorFlow

Run the cell above to load the MNIST data that comes with TensorFlow. You will use this data in **Section 1** and **Section 2**.

```
In [1]: import tensorflow as tf
        from tensorflow.examples.tutorials.mnist import input_data
        mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
        X_train, y_train = mnist.train.images, mnist.train.labels
        X_validation, y_validation = mnist.validation.images, mnist.validation.labels
        X_test, y_test = mnist.test.images, mnist.test.labels
        print("Image Shape: {}".format(X_train[0].shape))
        print("Training Set: {} samples".format(len(X_train)))
        print("Validation Set: {} samples".format(len(X_validation)))
        print("Test Set: {} samples".format(len(X_test)))
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
Image Shape: (784,)
Training Set: 55000 samples
Validation Set: 5000 samples
Test Set: 10000 samples
```

Section 1 : My First Model in TensorFlow

Before starting with CNN, let's train and test in TensorFlow the example : $y = \text{softmax}(Wx + b)$ seen in the DeepLearning course last week.

This model reaches an accuracy of about 92 %. You will also learn how to launch the tensorBoard

https://www.tensorflow.org/get_started/summaries_and_tensorboard

(https://www.tensorflow.org/get_started/summaries_and_tensorboard) to visualize the computation graph, statistics and learning curves.

Part 1 : Read carefully the code in the cell below. Run it to perform training.

```
In [2]: from __future__ import print_function
import tensorflow as tf

#STEP 1

# Parameters
learning_rate = 0.01
training_epochs = 100
batch_size = 128
display_step = 1
logs_path = 'log_files/' # useful for tensorboard

# tf Graph Input: mnist data image of shape 28*28=784
x = tf.placeholder(tf.float32, [None, 784], name='InputData')
# 0-9 digits recognition, 10 classes
y = tf.placeholder(tf.float32, [None, 10], name='LabelData')
```

```

# Set model weights
W = tf.Variable(tf.zeros([784, 10]), name='Weights')
b = tf.Variable(tf.zeros([10]), name='Bias')

# Construct model and encapsulating all ops into scopes, making Tensorboard's Graph visualization more convenient
with tf.name_scope('Model'):
    # Model
    pred = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax
with tf.name_scope('Loss'):
    # Minimize error using cross entropy
    cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
with tf.name_scope('SGD'):
    # Gradient Descent
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
with tf.name_scope('Accuracy'):
    # Accuracy
    acc = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
    acc = tf.reduce_mean(tf.cast(acc, tf.float32))

# Initializing the variables
init = tf.global_variables_initializer()
# Create a summary to monitor cost tensor
tf.summary.scalar("Loss", cost)
# Create a summary to monitor accuracy tensor
tf.summary.scalar("Accuracy", acc)
# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()

#STEP 2

# Launch the graph for training
with tf.Session() as sess:
    sess.run(init)
    # op to write logs to Tensorboard
    summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())
    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # Run optimization op (backprop), cost op (to get loss value)

            # and summary nodes
            _, c, summary = sess.run([optimizer, cost, merged_summary_op],
                                     feed_dict={x: batch_xs, y: batch_ys})

```

```
        # Write logs at every iteration
        summary_writer.add_summary(summary, epoch * total_batch + i)
        # Compute average loss
        avg_cost += c / total_batch
    # Display logs per epoch step
    if (epoch+1) % display_step == 0:
        print("Epoch: ", '%02d' % (epoch+1), " =====> Loss=", "{:.9f}".format(avg_cost))

    print("Optimization Finished!")

# Test model
# Calculate accuracy
print("Accuracy:", acc.eval({x: mnist.test.images, y: mnist.test.labels}))
```

```
Epoch: 01 =====> Loss= 1.286880517
Epoch: 02 =====> Loss= 0.731984088
Epoch: 03 =====> Loss= 0.600053700
Epoch: 04 =====> Loss= 0.536617899
Epoch: 05 =====> Loss= 0.497729340
Epoch: 06 =====> Loss= 0.470953372
Epoch: 07 =====> Loss= 0.451454671
Epoch: 08 =====> Loss= 0.436050117
Epoch: 09 =====> Loss= 0.423322550
Epoch: 10 =====> Loss= 0.413169499
Epoch: 11 =====> Loss= 0.404188006
Epoch: 12 =====> Loss= 0.396906343
Epoch: 13 =====> Loss= 0.390203831
Epoch: 14 =====> Loss= 0.384435246
Epoch: 15 =====> Loss= 0.379411392
Epoch: 16 =====> Loss= 0.374507659
Epoch: 17 =====> Loss= 0.370433932
Epoch: 18 =====> Loss= 0.366441450
Epoch: 19 =====> Loss= 0.362923098
Epoch: 20 =====> Loss= 0.359755657
Epoch: 21 =====> Loss= 0.356627841
Epoch: 22 =====> Loss= 0.353912697
Epoch: 23 =====> Loss= 0.350857394
Epoch: 24 =====> Loss= 0.348639083
Epoch: 25 =====> Loss= 0.346385159
Epoch: 26 =====> Loss= 0.344261822
Epoch: 27 =====> Loss= 0.342352411
Epoch: 28 =====> Loss= 0.340091097
Epoch: 29 =====> Loss= 0.338452283
Epoch: 30 =====> Loss= 0.336743982
Epoch: 31 =====> Loss= 0.335173390
Epoch: 32 =====> Loss= 0.333633489
Epoch: 33 =====> Loss= 0.332001159
Epoch: 34 =====> Loss= 0.330275257
Epoch: 35 =====> Loss= 0.329148997
Epoch: 36 =====> Loss= 0.327851887
Epoch: 37 =====> Loss= 0.326581550
Epoch: 38 =====> Loss= 0.325475444
Epoch: 39 =====> Loss= 0.324036331
Epoch: 40 =====> Loss= 0.323037213
Epoch: 41 =====> Loss= 0.322044352
Epoch: 42 =====> Loss= 0.321038089
Epoch: 43 =====> Loss= 0.320074725
Epoch: 44 =====> Loss= 0.318792509
Epoch: 45 =====> Loss= 0.317947309
Epoch: 46 =====> Loss= 0.316920819
Epoch: 47 =====> Loss= 0.316344473
Epoch: 48 =====> Loss= 0.315395182
Epoch: 49 =====> Loss= 0.314128211
Epoch: 50 =====> Loss= 0.313477824
Epoch: 51 =====> Loss= 0.312522907
Epoch: 52 =====> Loss= 0.312000528
Epoch: 53 =====> Loss= 0.311596174
Epoch: 54 =====> Loss= 0.310733500
Epoch: 55 =====> Loss= 0.310072478
Epoch: 56 =====> Loss= 0.309392135
Epoch: 57 =====> Loss= 0.308561868
```

```
Epoch: 58 =====> Loss= 0.307863509
Epoch: 59 =====> Loss= 0.307470363
Epoch: 60 =====> Loss= 0.306724633
Epoch: 61 =====> Loss= 0.306077459
Epoch: 62 =====> Loss= 0.305593763
Epoch: 63 =====> Loss= 0.304914057
Epoch: 64 =====> Loss= 0.304340292
Epoch: 65 =====> Loss= 0.303780586
Epoch: 66 =====> Loss= 0.303413475
Epoch: 67 =====> Loss= 0.302843422
Epoch: 68 =====> Loss= 0.302373371
Epoch: 69 =====> Loss= 0.301496408
Epoch: 70 =====> Loss= 0.301173913
Epoch: 71 =====> Loss= 0.300857027
Epoch: 72 =====> Loss= 0.300336870
Epoch: 73 =====> Loss= 0.299399064
Epoch: 74 =====> Loss= 0.299432497
Epoch: 75 =====> Loss= 0.298780698
Epoch: 76 =====> Loss= 0.298585278
Epoch: 77 =====> Loss= 0.297873315
Epoch: 78 =====> Loss= 0.297437032
Epoch: 79 =====> Loss= 0.297368031
Epoch: 80 =====> Loss= 0.296844116
Epoch: 81 =====> Loss= 0.296352289
Epoch: 82 =====> Loss= 0.296155450
Epoch: 83 =====> Loss= 0.295431316
Epoch: 84 =====> Loss= 0.295379238
Epoch: 85 =====> Loss= 0.294928055
Epoch: 86 =====> Loss= 0.294508771
Epoch: 87 =====> Loss= 0.294325627
Epoch: 88 =====> Loss= 0.293421298
Epoch: 89 =====> Loss= 0.293454468
Epoch: 90 =====> Loss= 0.293272546
Epoch: 91 =====> Loss= 0.292701247
Epoch: 92 =====> Loss= 0.292564908
Epoch: 93 =====> Loss= 0.291923672
Epoch: 94 =====> Loss= 0.291831442
Epoch: 95 =====> Loss= 0.291606335
Epoch: 96 =====> Loss= 0.291287685
Epoch: 97 =====> Loss= 0.290778177
Epoch: 98 =====> Loss= 0.290533946
Epoch: 99 =====> Loss= 0.290248456
Epoch: 100 =====> Loss= 0.290133316
Optimization Finished!
Accuracy: 0.9203
```

Part 2 : Using Tensorboard, we can now visualize the created graph, giving you an overview of your architecture and how all of the major components are connected. You can also see and analyse the learning curves.

To launch tensorBoard:

- Go to the **TP2** folder,
- Open a Terminal and run the command line "**tensorboard --logdir= log_files/**", it will generate an http link ,ex <http://666.6.6.6:6006> (<http://666.6.6.6:6006>),
- Copy this link into your web browser

Enjoy It !!

Section 2 : The 99% MNIST Challenge !

Part 1 : LeNet5 implementation

One you are now familiar with **tensorFlow** and **tensorBoard**, you are in this section to build, train and test the baseline LeNet-5 (<http://yann.lecun.com/exdb/lenet/>) model for the MNIST digits recognition problem.

In more advanced step you will make some optimizations to get more than 99% of accuracy. The best model can get to over 99.7% accuracy!

For more information, have a look at this list of results :

http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html
(http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html)

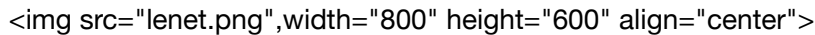


Figure 1: Lenet 5

The LeNet architecture accepts a $32 \times 32 \times C$ image as input, where C is the number of color channels. Since MNIST images are grayscale, C is 1 in this case.

Layer 1: Convolutional. The output shape should be $28 \times 28 \times 6$. **Activation.** sigmoid **Pooling.** The output shape should be $14 \times 14 \times 6$.

Layer 2: Convolutional. The output shape should be $10 \times 10 \times 16$. **Activation.** sigmoid **Pooling.** The output shape should be $5 \times 5 \times 16$.

Flatten. Flatten the output shape of the final pooling layer such that it's 1D instead of 3D. You may need to use `*flatten` from `tensorflow.contrib.layers` import `flatten`

Layer 3: Fully Connected. This should have 120 outputs. **Activation.** sigmoid

Layer 4: Fully Connected. This should have 84 outputs. **Activation.** sigmoid

Layer 5: Fully Connected. This should have 10 outputs. **Activation.** softmax

Question 2.1.1 Implement the Neural Network architecture described above. For that, you will use classes and functions from https://www.tensorflow.org/api_docs/python/tf/nn (https://www.tensorflow.org/api_docs/python/tf/nn).

We give you some helper functions for weights and bias initialization. Also you can refer to section 1.

```
In [2]: #Helper functions for weights and bias initialization
def weight_variable(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)
```

```
In [3]: # https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
# https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

def LeNet5_Model(data,activation_function=tf.nn.sigmoid):

    # layer 1 param
    conv1_weights = weight_variable([5,5,1,6])
    conv1_bias = bias_variable([6])

    # layer 2 param
    conv2_weights = weight_variable([5,5,6,16])
    conv2_bias = bias_variable([16])
```

```

# layer 3 param
layer3_weights = weight_variable([400, 120])
layer3_bias = bias_variable([120])

# layer 4 param
layer4_weights = weight_variable([120, 84])
layer4_bias = bias_variable([84])

# layer 5 param
layer5_weights = weight_variable([84, 10])
layer5_bias = bias_variable([10])

with tf.name_scope('Model'):
    with tf.name_scope('Layer1'):
        conv1 =
tf.nn.conv2d(input=data,filter=conv1_weights,strides=
[1,1,1,1],padding='SAME')
        print(conv1.shape)
        sigmoid1 = activation_function(conv1 + conv1_bias)
        pool1 = tf.nn.max_pool(sigmoid1,ksize=[1, 2, 2, 1],strides=
[1, 2, 2, 1],padding='VALID')
        print(pool1.shape)

        with tf.name_scope('Layer2'):
            conv2 = tf.nn.conv2d(input=pool1,filter=conv2_weights,stride
s=[1,1,1,1],padding='VALID')
            print(conv2.shape)
            sigmoid2 = activation_function(conv2 + conv2_bias)
            pool2 = tf.nn.max_pool(sigmoid2,ksize=[1, 2, 2, 1],strides=
[1, 2, 2, 1],padding='VALID')
            print(pool2.shape)

        with tf.name_scope('Flatten'):
            flat_inputs = tf.contrib.layers.flatten(pool2)
            print(flat_inputs.shape)

        with tf.name_scope('Layer3'):
            out3 = activation_function(tf.matmul(flat_inputs, layer3_w
ights) + layer3_bias)

        with tf.name_scope('Layer4'):
            out4 = activation_function(tf.matmul(out3, layer4_weights) +
layer4_bias)

        with tf.name_scope('Layer5'):
            pred = tf.nn.softmax(tf.matmul(out4, layer5_weights) + layer
5_bias) # Softmax
        return pred

```

Question 2.1.2. Calculate the number of parameters of this model

```
In [13]: total_parameters = 0
for variable in tf.trainable_variables():
    # shape is an array of tf.Dimension
    shape = variable.get_shape()
    print(shape)
    variable_parameters = 1
    for dim in shape:
        variable_parameters *= dim.value
    print(variable_parameters)
    total_parameters += variable_parameters
print(total_parameters)
```

```
(5, 5, 1, 6)
150
(6,)
6
(5, 5, 6, 16)
2400
(16,)
16
(400, 120)
48000
(120,)
120
(120, 84)
10080
(84,)
84
(84, 10)
840
(10,)
10
61706
```

```
In [15]: layer1 = 5*5*1*6 + 6
layer2 = 5*5*6*16 + 16
layer3 = 400*120 + 120
layer4 = 120*84 + 84
layer5 = 84*10 + 10
tot = layer1 + layer2 + layer3 + layer4 + layer5
print('total number of parameters: %d' % tot)

total number of parameters: 61706
```

Your answer goes here in details

Question 2.1.3. Start the training with the parameters cited below:

Learning rate =0.1
Loss Fucntion : Cross entropy
Optimisateur: SGD
Number of training iterations= 100
The batch size =128

```

In [18]: from __future__ import print_function
import tensorflow as tf
from numpy import array
import numpy as np
#STEP 1
tf.reset_default_graph()

# Parameters
learning_rate = 0.1
training_epochs = 100
batch_size = 128
display_step = 1
logs_path = 'log_files/' # useful for tensorboard

# tf Graph Input: mnist data image of shape 28*28=784
x = tf.placeholder(tf.float32, [batch_size,28, 28,1], name='InputData')
# 0-9 digits recognition, 10 classes
y = tf.placeholder(tf.float32, [batch_size, 10], name='LabelData')

# Construct model and encapsulating all ops into scopes, making Tensorboard's Graph visualization more convenient
with tf.name_scope('Model'):
    # Model
    pred = LeNet5_Model(data=x)
with tf.name_scope('Loss'):
    # Minimize error using cross entropy
    cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
with tf.name_scope('SGD'):
    # Gradient Descent
    optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
with tf.name_scope('Accuracy'):
    # Accuracy
    acc = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
    acc = tf.reduce_mean(tf.cast(acc, tf.float32))

# Initializing the variables
init = tf.global_variables_initializer()
# Create a summary to monitor cost tensor
tf.summary.scalar("Loss", cost)
# Create a summary to monitor accuracy tensor
tf.summary.scalar("Accuracy", acc)
# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()

```

```

#STEP 2

# Launch the graph for training
with tf.Session() as sess:
    sess.run(init)
    # op to write logs to Tensorboard
    summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_defau
lt_graph())
    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # Loop over all batches
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)

            batch_xs = array(batch_xs).reshape(batch_size, 28,28,1)
            #print(batch_xs.shape)
            #print(batch_xs.dtype)
            # Run optimization op (backprop), cost op (to get loss valu
e)

            # and summary nodes
            _, c, summary = sess.run([optimizer, cost,
merged_summary_op],
                                     feed_dict={x: batch_xs, y:
batch_ys})

            # Write logs at every iteration
            summary_writer.add_summary(summary, epoch * total_batch + i)
            # Compute average loss
            avg_cost += c / total_batch
        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            print("Epoch: ", '%02d' % (epoch+1), " =====> Loss=", "{:.9
f}".format(avg_cost))

        print("Optimization Finished!")

```

```
(128, 28, 28, 6)
(128, 14, 14, 6)
(128, 10, 10, 16)
(128, 5, 5, 16)
(128, 400)
Epoch: 01 =====> Loss= 2.306496096
Epoch: 02 =====> Loss= 2.305118391
Epoch: 03 =====> Loss= 2.304803859
Epoch: 04 =====> Loss= 2.303603085
Epoch: 05 =====> Loss= 2.302692528
Epoch: 06 =====> Loss= 2.301137964
Epoch: 07 =====> Loss= 2.296393236
Epoch: 08 =====> Loss= 2.276063018
Epoch: 09 =====> Loss= 1.980490430
Epoch: 10 =====> Loss= 1.061656557
Epoch: 11 =====> Loss= 0.673400137
Epoch: 12 =====> Loss= 0.457144566
Epoch: 13 =====> Loss= 0.350934934
Epoch: 14 =====> Loss= 0.285083193
Epoch: 15 =====> Loss= 0.246945218
Epoch: 16 =====> Loss= 0.216544041
Epoch: 17 =====> Loss= 0.191632318
Epoch: 18 =====> Loss= 0.175332371
Epoch: 19 =====> Loss= 0.161357922
Epoch: 20 =====> Loss= 0.148372099
Epoch: 21 =====> Loss= 0.134611330
Epoch: 22 =====> Loss= 0.130380134
Epoch: 23 =====> Loss= 0.120058484
Epoch: 24 =====> Loss= 0.112938234
Epoch: 25 =====> Loss= 0.110061851
Epoch: 26 =====> Loss= 0.105592581
Epoch: 27 =====> Loss= 0.098736952
Epoch: 28 =====> Loss= 0.094830746
Epoch: 29 =====> Loss= 0.092585727
Epoch: 30 =====> Loss= 0.089760418
Epoch: 31 =====> Loss= 0.083930924
Epoch: 32 =====> Loss= 0.084534728
Epoch: 33 =====> Loss= 0.080185655
Epoch: 34 =====> Loss= 0.077936677
Epoch: 35 =====> Loss= 0.075848382
Epoch: 36 =====> Loss= 0.075313177
Epoch: 37 =====> Loss= 0.072877646
Epoch: 38 =====> Loss= 0.072251406
Epoch: 39 =====> Loss= 0.066627552
Epoch: 40 =====> Loss= 0.071041026
Epoch: 41 =====> Loss= 0.065217602
Epoch: 42 =====> Loss= 0.062069521
Epoch: 43 =====> Loss= 0.066116210
Epoch: 44 =====> Loss= 0.060936432
Epoch: 45 =====> Loss= 0.062576688
Epoch: 46 =====> Loss= 0.059032645
Epoch: 47 =====> Loss= 0.057685662
Epoch: 48 =====> Loss= 0.057808492
Epoch: 49 =====> Loss= 0.056264729
Epoch: 50 =====> Loss= 0.055786627
Epoch: 51 =====> Loss= 0.055408733
Epoch: 52 =====> Loss= 0.052068538
```



```
Epoch: 53 =====> Loss= 0.052856209
Epoch: 54 =====> Loss= 0.051636685
Epoch: 55 =====> Loss= 0.050580791
Epoch: 56 =====> Loss= 0.051136808
Epoch: 57 =====> Loss= 0.046678293
Epoch: 58 =====> Loss= 0.049320392
Epoch: 59 =====> Loss= 0.048037040
Epoch: 60 =====> Loss= 0.046233684
Epoch: 61 =====> Loss= 0.046614292
Epoch: 62 =====> Loss= 0.045592060
Epoch: 63 =====> Loss= 0.043955584
Epoch: 64 =====> Loss= 0.045034743
Epoch: 65 =====> Loss= 0.041108692
Epoch: 66 =====> Loss= 0.045543321
Epoch: 67 =====> Loss= 0.040326549
Epoch: 68 =====> Loss= 0.042093843
Epoch: 69 =====> Loss= 0.041527857
Epoch: 70 =====> Loss= 0.039806241
Epoch: 71 =====> Loss= 0.040189555
Epoch: 72 =====> Loss= 0.040268243
Epoch: 73 =====> Loss= 0.037503375
Epoch: 74 =====> Loss= 0.038824848
Epoch: 75 =====> Loss= 0.038723698
Epoch: 76 =====> Loss= 0.036885229
Epoch: 77 =====> Loss= 0.036568665
Epoch: 78 =====> Loss= 0.036027519
Epoch: 79 =====> Loss= 0.035885228
Epoch: 80 =====> Loss= 0.035430162
Epoch: 81 =====> Loss= 0.034851235
Epoch: 82 =====> Loss= 0.034437052
Epoch: 83 =====> Loss= 0.034559765
Epoch: 84 =====> Loss= 0.033277507
Epoch: 85 =====> Loss= 0.033500814
Epoch: 86 =====> Loss= 0.034352661
Epoch: 87 =====> Loss= 0.031346059
Epoch: 88 =====> Loss= 0.032277952
Epoch: 89 =====> Loss= 0.031845599
Epoch: 90 =====> Loss= 0.031653468
Epoch: 91 =====> Loss= 0.030574811
Epoch: 92 =====> Loss= 0.030923009
Epoch: 93 =====> Loss= 0.029608658
Epoch: 94 =====> Loss= 0.030911727
Epoch: 95 =====> Loss= 0.028381630
Epoch: 96 =====> Loss= 0.028625046
Epoch: 97 =====> Loss= 0.030301574
Epoch: 98 =====> Loss= 0.028793503
Epoch: 99 =====> Loss= 0.027472138
Epoch: 100 =====> Loss= 0.028038724
Optimization Finished!
```

Question 2.1.4. Implement the evaluation function for accuracy computation

```
In [4]: def evaluate(model, y):  
        #your implementation goes here  
        correct_prediction = tf.equal(tf.argmax(model,1), tf.argmax(y,1))  
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))  
        #print(accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels}))  
  
        return accuracy
```

Question 2.1.5. Implement training pipeline and run the training data through it to train the model.

- Before each epoch, shuffle the training set.
- Print the loss per mini batch and the training/validation accuracy per epoch. (Display results every 100 epochs)
- Save the model after training
- Print after training the final testing accuracy

```

In [5]: import numpy as np
        # Initializing the variables
        def train(learning_rate, training_epochs, batch_size, display_step, optimizer_method=tf.train.GradientDescentOptimizer, activation_function=tf.nn.sigmoid):
            tf.reset_default_graph()
            # Initializing the session

            logs_path = 'log_files/' # useful for tensorboard

            # tf Graph Input: mnist data image of shape 28*28=784
            x = tf.placeholder(tf.float32, [None, 28, 28, 1], name='InputData')
            # 0-9 digits recognition, 10 classes
            y = tf.placeholder(tf.float32, [None, 10], name='LabelData')

            # Construct model and encapsulating all ops into scopes, making TensorBoard's Graph visualization more convenient
            with tf.name_scope('Model'):
                # Model
                pred = LeNet5_Model(data=x, activation_function=activation_function)

            with tf.name_scope('Loss'):
                # Minimize error using cross entropy
                # Minimize error using cross entropy
                if activation_function == tf.nn.sigmoid:
                    cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
                else:
                    cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(tf.clip_by_value(pred, -1.0, 1.0)), reduction_indices=1))
                    #cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
            with tf.name_scope('SGD'):
                # Gradient Descent
                optimizer = optimizer_method(learning_rate).minimize(cost)
            with tf.name_scope('Accuracy'):
                # Accuracy

```

```

acc = evaluate(pred, y)

# Initializing the variables
init = tf.global_variables_initializer()
# Create a summary to monitor cost tensor
tf.summary.scalar("Loss", cost)
# Create a summary to monitor accuracy tensor
tf.summary.scalar("Accuracy", acc)
# Merge all summaries into a single op
merged_summary_op = tf.summary.merge_all()

saver = tf.train.Saver()

print ("Start Training!")
t0 = time()
X_train,Y_train = mnist.train.images.reshape((-1,28,28,1)), mnist.train.labels
X_val,Y_val = mnist.validation.images.reshape((-1,28,28,1)), mnist.validation.labels

# Launch the graph for training
with tf.Session() as sess:
    sess.run(init)
    # op to write logs to Tensorboard
    summary_writer = tf.summary.FileWriter(logs_path, graph=tf.get_default_graph())

    # Training cycle
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)

        # Loop over all batches
        for i in range(total_batch):
            # train_next_batch shuffle the images by default
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            batch_xs = batch_xs.reshape((-1,28,28,1))

            # Run optimization op (backprop), cost op (to get loss value)
            # and summary nodes
            _, c, summary = sess.run([optimizer, cost, merged_summary_op],
                                     feed_dict={x: batch_xs,
                                                 y: batch_ys})

            # Write logs at every iteration
            summary_writer.add_summary(summary, epoch * total_batch + i)

            # Compute average loss
            avg_cost += c / total_batch

        # Display logs per epoch step
        if (epoch+1) % display_step == 0:
            print("Epoch: ", '%02d' % (epoch+1), "====> Loss=", "

```

```

{: .9f}").format(avg_cost))

        acc_train = acc.eval({x: X_train, y: Y_train})
        print("Epoch: ", '%02d' % (epoch+1), "=====> Accuracy Train=", "{: .9f}").format(acc_train))

        acc_val = acc.eval({x: X_val, y: Y_val})
        print("Epoch: ", '%02d' % (epoch+1), "=====> Accuracy Validation=", "{: .9f}").format(acc_val))
        print ("Training Finished!")
        t1 = time()
        # Save the variables to disk.
        save_path = saver.save(sess, "model.ckpt")
        print("Model saved in file: %s" % save_path)

        #Your implementation for testing accuracy after training goes here

        X_test,Y_test = mnist.test.images.reshape((-1,28,28,1)),mnist.test.labels

        acc_test = acc.eval({x: X_test, y: Y_test})
        print("Accuracy Test=", "{: .9f}").format(acc_test))

        return acc_train,acc_val,acc_test,t1-t0

```

```

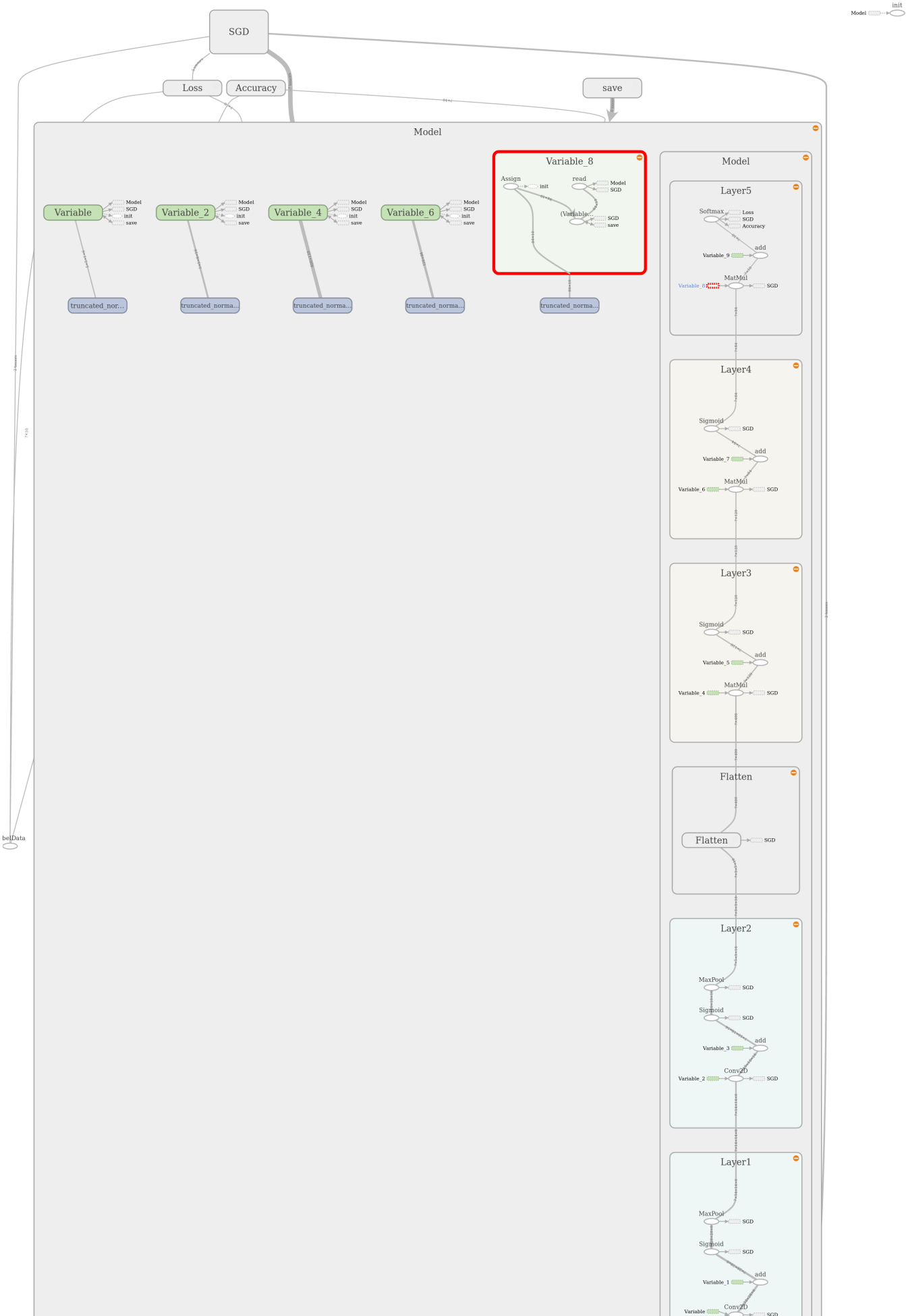
In [87]: %time train (0.1,100,128,10,optimizer_method=tf.train.GradientDescentOpt
im�izer)

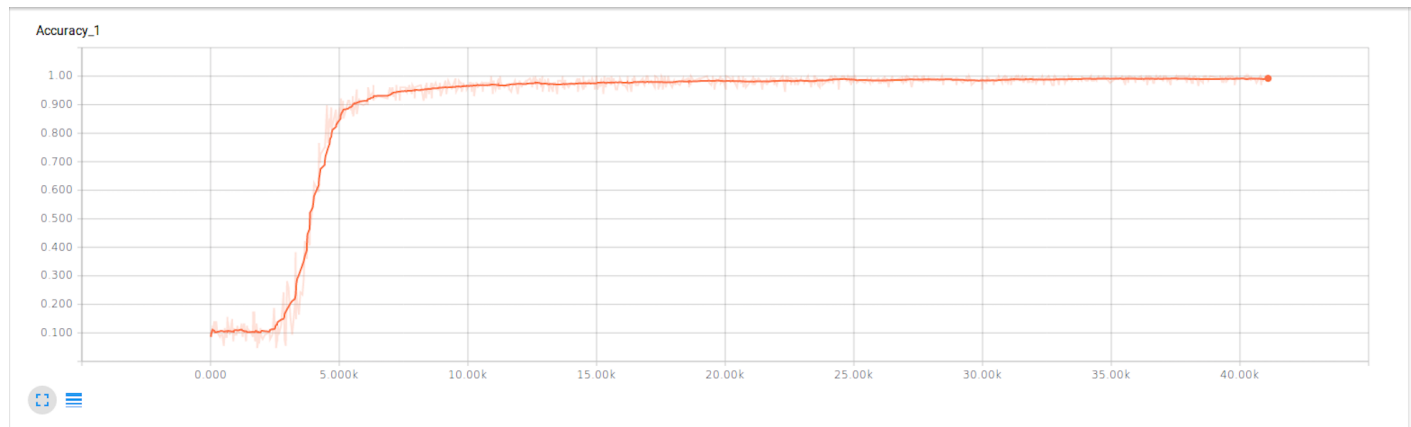
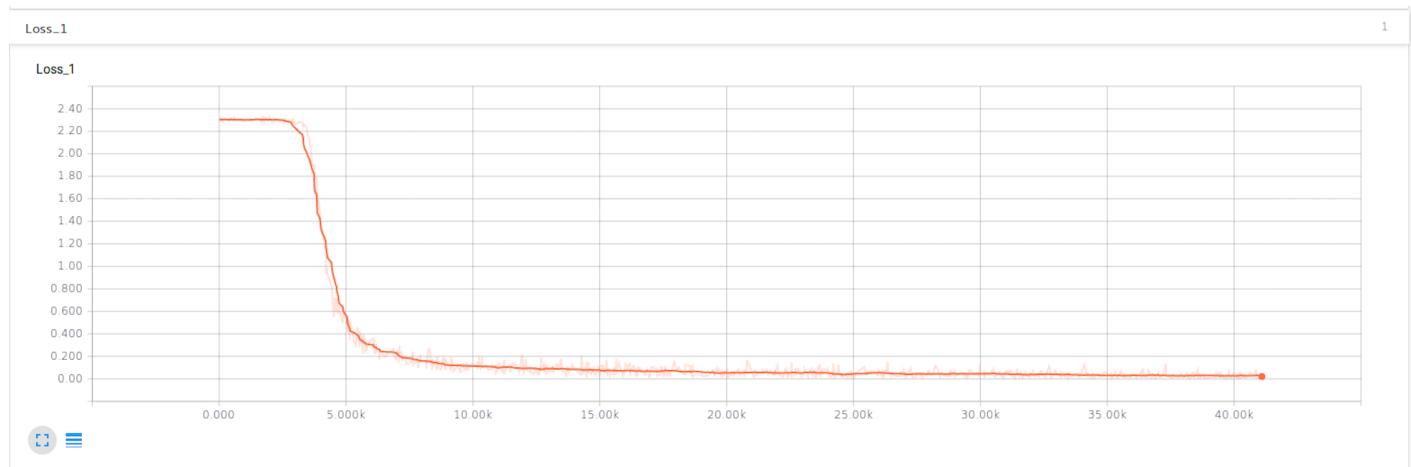
(?, 28, 28, 6)
(?, 14, 14, 6)
(?, 10, 10, 16)
(?, 5, 5, 16)
(?, 400)
Start Training!
Epoch: 10  TOTAL  =====> Loss= 2.098373385
Epoch: 10  TOTAL  =====> Accuracy Train= 0.409581810
Epoch: 10  TOTAL  =====> Accuracy Validation= 0.407400012
Epoch: 20  TOTAL  =====> Loss= 0.165306511
Epoch: 20  TOTAL  =====> Accuracy Train= 0.953509092
Epoch: 20  TOTAL  =====> Accuracy Validation= 0.957799971
Epoch: 30  TOTAL  =====> Loss= 0.094643849
Epoch: 30  TOTAL  =====> Accuracy Train= 0.972872734
Epoch: 30  TOTAL  =====> Accuracy Validation= 0.972400010
Epoch: 40  TOTAL  =====> Loss= 0.068466610
Epoch: 40  TOTAL  =====> Accuracy Train= 0.977836370
Epoch: 40  TOTAL  =====> Accuracy Validation= 0.977400005
Epoch: 50  TOTAL  =====> Loss= 0.056492984
Epoch: 50  TOTAL  =====> Accuracy Train= 0.983418167
Epoch: 50  TOTAL  =====> Accuracy Validation= 0.980199993
Epoch: 60  TOTAL  =====> Loss= 0.047570020
Epoch: 60  TOTAL  =====> Accuracy Train= 0.986472726
Epoch: 60  TOTAL  =====> Accuracy Validation= 0.982800007
Epoch: 70  TOTAL  =====> Loss= 0.041231574
Epoch: 70  TOTAL  =====> Accuracy Train= 0.988545477
Epoch: 70  TOTAL  =====> Accuracy Validation= 0.984399974
Epoch: 80  TOTAL  =====> Loss= 0.036321188
Epoch: 80  TOTAL  =====> Accuracy Train= 0.990109086
Epoch: 80  TOTAL  =====> Accuracy Validation= 0.986000001
Epoch: 90  TOTAL  =====> Loss= 0.032033988
Epoch: 90  TOTAL  =====> Accuracy Train= 0.991127253
Epoch: 90  TOTAL  =====> Accuracy Validation= 0.985199988
Epoch: 100 TOTAL  =====> Loss= 0.027192902
Epoch: 100 TOTAL  =====> Accuracy Train= 0.992200017
Epoch: 100 TOTAL  =====> Accuracy Validation= 0.986800015
Training Finished!
Model saved in file: model.ckpt
TOTAL  =====> Accuracy Test= 0.986699998
CPU times: user 1h 26min 32s, sys: 14min 28s, total: 1h 41min
Wall time: 35min 38s

Out[87]: (0.99220002, 0.98680001, 0.9867, 2137.679314851761)

```

Question 2.1.6 : Use tensorBoard to visualise and save the LeNet5 Graph and all learning curves. Save all obtained figures in the folder "TP2/MNIST_99_Challenge_Figures"





Comment:

Here we see how the accuracy rapidly increases after few epochs and then increases at an always slower rate. Regarding the accuracy, we see that it gets always nearer to zero epoch after epoch. From the graph we can have a confirm of our network architecture.

Part 2 : LeNET 5 Optimization

Question 2.2.1 Change the sigmoid function with a Relu :

- Retrain your network with SGD and AdamOptimizer and then fill the table above :

Optimizer	Gradient Descent	AdamOptimizer
Validation Accuracy	0.99180001	0.097599998
Testing Accuracy	0.99089998	0.1032
Training Time	36min	36min

- Try with different learning rates for each Optimizer (0.0001 and 0.001) and different Batch sizes (50 and 128) for 20000 Epochs.
- For each optimizer, plot (on the same curve) the **testing accuracies** function to **(learning rate, batch size)**
- Did you reach the 99% accuracy ? What are the optimal parametres that gave you the best results?

Comment:

- Relu: when we use the relu we need to change the cost function, in fact we need to do gradient clipping otherwise our network will crash.
- The Adam optimizer gives very bad results when used with high learning rate, in fact it works better with a low learning rate, such as 0.001
- When we use the stochastic gradient descent with the relu we obtain really good results: more than 99% test accuracy. So we could stop here.

```
In [6]: from time import time
```

```
In [14]: %time train (0.1,100,128,10,optimizer_method=tf.train.GradientDescentOpt
imazer,activation_function=tf.nn.relu)
```

```
(?, 28, 28, 6)
(?, 14, 14, 6)
(?, 10, 10, 16)
(?, 5, 5, 16)
(?, 400)
Start Training!
('Epoch: ', '10', ' TOTAL =====> Loss=', '0.022426698')
('Epoch: ', '10', ' TOTAL =====> Accuracy Train=', '0.993163645')
('Epoch: ', '10', ' TOTAL =====> Accuracy Validation=', '0.989000022')
('Epoch: ', '20', ' TOTAL =====> Loss=', '0.006640168')
('Epoch: ', '20', ' TOTAL =====> Accuracy Train=', '0.998363614')
('Epoch: ', '20', ' TOTAL =====> Accuracy Validation=', '0.990400016')
('Epoch: ', '30', ' TOTAL =====> Loss=', '0.001571395')
('Epoch: ', '30', ' TOTAL =====> Accuracy Train=', '0.999836385')
('Epoch: ', '30', ' TOTAL =====> Accuracy Validation=', '0.991400003')
('Epoch: ', '40', ' TOTAL =====> Loss=', '0.005297473')
('Epoch: ', '40', ' TOTAL =====> Accuracy Train=', '0.999490917')
('Epoch: ', '40', ' TOTAL =====> Accuracy Validation=', '0.989799976')
('Epoch: ', '50', ' TOTAL =====> Loss=', '0.000140374')
('Epoch: ', '50', ' TOTAL =====> Accuracy Train=', '0.999981821')
('Epoch: ', '50', ' TOTAL =====> Accuracy Validation=', '0.991800010')
('Epoch: ', '60', ' TOTAL =====> Loss=', '0.000417097')
('Epoch: ', '60', ' TOTAL =====> Accuracy Train=', '0.999981821')
('Epoch: ', '60', ' TOTAL =====> Accuracy Validation=', '0.991599977')
('Epoch: ', '70', ' TOTAL =====> Loss=', '0.000396061')
('Epoch: ', '70', ' TOTAL =====> Accuracy Train=', '0.999981821')
('Epoch: ', '70', ' TOTAL =====> Accuracy Validation=', '0.991999984')
('Epoch: ', '80', ' TOTAL =====> Loss=', '0.000382105')
('Epoch: ', '80', ' TOTAL =====> Accuracy Train=', '0.999981821')
('Epoch: ', '80', ' TOTAL =====> Accuracy Validation=', '0.991800010')
('Epoch: ', '90', ' TOTAL =====> Loss=', '0.000373299')
('Epoch: ', '90', ' TOTAL =====> Accuracy Train=', '0.999981821')
('Epoch: ', '90', ' TOTAL =====> Accuracy Validation=', '0.991800010')
('Epoch: ', '100', ' TOTAL =====> Loss=', '0.000031499')
('Epoch: ', '100', ' TOTAL =====> Accuracy Train=', '0.999981821')
('Epoch: ', '100', ' TOTAL =====> Accuracy Validation=', '0.99180001
0')
Training Finished!
Model saved in file: model.ckpt
('TOTAL =====> Accuracy Test=', '0.990899980')
CPU times: user 1h 24min 38s, sys: 13min 29s, total: 1h 38min 7s
Wall time: 36min 13s
```

```
Out[14]: (0.99998182, 0.99180001, 0.99089998, 2172.412855863571)
```

```
In [15]: %time train (0.1,100,128,10,optimizer_method=tf.train.AdamOptimizer,activation_function=tf.nn.relu)
```

```
(?, 28, 28, 6)
(?, 14, 14, 6)
(?, 10, 10, 16)
(?, 5, 5, 16)
(?, 400)
Start Training!
('Epoch: ', '10', ' TOTAL =====> Loss=', '16.597466315')
('Epoch: ', '10', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '10', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '20', ' TOTAL =====> Loss=', '16.558553229')
('Epoch: ', '20', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '20', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '30', ' TOTAL =====> Loss=', '16.557546838')
('Epoch: ', '30', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '30', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '40', ' TOTAL =====> Loss=', '16.589079859')
('Epoch: ', '40', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '40', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '50', ' TOTAL =====> Loss=', '16.595118102')
('Epoch: ', '50', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '50', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '60', ' TOTAL =====> Loss=', '16.606188205')
('Epoch: ', '60', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '60', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '70', ' TOTAL =====> Loss=', '16.561907793')
('Epoch: ', '70', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '70', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '80', ' TOTAL =====> Loss=', '16.606859145')
('Epoch: ', '80', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '80', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '90', ' TOTAL =====> Loss=', '16.592769930')
('Epoch: ', '90', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '90', ' TOTAL =====> Accuracy Validation=', '0.097599998')
('Epoch: ', '100', ' TOTAL =====> Loss=', '16.586396231')
('Epoch: ', '100', ' TOTAL =====> Accuracy Train=', '0.099454544')
('Epoch: ', '100', ' TOTAL =====> Accuracy Validation=', '0.097599998')
Training Finished!
Model saved in file: model.ckpt
('TOTAL =====> Accuracy Test=', '0.103200004')
CPU times: user 1h 24min 53s, sys: 13min 50s, total: 1h 38min 43s
Wall time: 36min 27s
```

```
Out[15]: (0.099454544, 0.097599998, 0.1032, 2185.8693709373474)
```

In [7]: *# your answer goes here*

```

columns = ['optimizer','learning_rate','activation_function','batch_size',
'training_accuracy','validation_accuracy','test_accuracy','elapsed_time']
optimizer_options = {'gradient_descent':tf.train.GradientDescentOptimizer,
'adam':tf.train.AdamOptimizer}
learning_options = [0.001,0.0001]
activation_options = {'sigmoid':tf.nn.sigmoid,'relu':tf.nn.relu}
batch_options = [50,128]

final_results = []
for optimizer_label in optimizer_options:
    optimizer = optimizer_options[optimizer_label]
    for learning_rate in learning_options:
        for activation_label in activation_options:
            activation_function = activation_options[activation_label]
            for batch_size in batch_options:
                #TO DEFINE TrainAndTest
                training_accuracy,validation_accuracy,test_accuracy,elapsed_time = train(
                    learning_rate = learning_rate,
                    training_epochs=100,
                    batch_size = batch_size,
                    display_step = 10,
                    optimizer_method = optimizer,
                    activation_function = activation_function
                )
                obj_test = {'optimizer':optimizer_label,
                    'learning_rate':learning_rate,
                    'activation_function':activation_label,
                    'batch_size':batch_size,
                    'training_accuracy':training_accuracy,
                    'validation_accuracy':validation_accuracy,
                    'test_accuracy':test_accuracy,
                    'elapsed_time': elapsed_time
                }

                final_results.append(obj_test)

```

```
(?, 28, 28, 6)
(?, 14, 14, 6)
(?, 10, 10, 16)
(?, 5, 5, 16)
(?, 400)
Start Training!
('Epoch: ', '10', '=====> Loss=', '0.013496247')
('Epoch: ', '10', '=====> Accuracy Train=', '0.995527267')
('Epoch: ', '10', '=====> Accuracy Validation=', '0.989600003')
('Epoch: ', '20', '=====> Loss=', '0.006858599')
('Epoch: ', '20', '=====> Accuracy Train=', '0.999272704')
('Epoch: ', '20', '=====> Accuracy Validation=', '0.991199970')
('Epoch: ', '30', '=====> Loss=', '0.005198982')
('Epoch: ', '30', '=====> Accuracy Train=', '0.998963654')
('Epoch: ', '30', '=====> Accuracy Validation=', '0.992399991')
('Epoch: ', '40', '=====> Loss=', '0.006224660')
('Epoch: ', '40', '=====> Accuracy Train=', '0.998654544')
('Epoch: ', '40', '=====> Accuracy Validation=', '0.990999997')
('Epoch: ', '50', '=====> Loss=', '0.005244983')
('Epoch: ', '50', '=====> Accuracy Train=', '0.998981833')
('Epoch: ', '50', '=====> Accuracy Validation=', '0.988200009')
('Epoch: ', '60', '=====> Loss=', '0.002950049')
('Epoch: ', '60', '=====> Accuracy Train=', '0.999109089')
('Epoch: ', '60', '=====> Accuracy Validation=', '0.989199996')
('Epoch: ', '70', '=====> Loss=', '0.002159522')
('Epoch: ', '70', '=====> Accuracy Train=', '0.998290896')
('Epoch: ', '70', '=====> Accuracy Validation=', '0.988600016')
('Epoch: ', '80', '=====> Loss=', '0.002705119')
('Epoch: ', '80', '=====> Accuracy Train=', '0.999872744')
('Epoch: ', '80', '=====> Accuracy Validation=', '0.990400016')
('Epoch: ', '90', '=====> Loss=', '0.002768753')
('Epoch: ', '90', '=====> Accuracy Train=', '0.999400020')
('Epoch: ', '90', '=====> Accuracy Validation=', '0.990800023')
('Epoch: ', '100', '=====> Loss=', '0.003820146')
('Epoch: ', '100', '=====> Accuracy Train=', '0.999563634')
('Epoch: ', '100', '=====> Accuracy Validation=', '0.990400016')
```

Training Finished!

Model saved in file: model.ckpt

```
('Accuracy Test=', '0.989799976')
```

```
(?, 28, 28, 6)
(?, 14, 14, 6)
(?, 10, 10, 16)
(?, 5, 5, 16)
(?, 400)
```

Start Training!

```
('Epoch: ', '10', '=====> Loss=', '0.020895901')
('Epoch: ', '10', '=====> Accuracy Train=', '0.995745480')
('Epoch: ', '10', '=====> Accuracy Validation=', '0.988600016')
('Epoch: ', '20', '=====> Loss=', '0.007990904')
('Epoch: ', '20', '=====> Accuracy Train=', '0.997436345')
('Epoch: ', '20', '=====> Accuracy Validation=', '0.990199983')
('Epoch: ', '30', '=====> Loss=', '0.006926079')
('Epoch: ', '30', '=====> Accuracy Train=', '0.997981846')
('Epoch: ', '30', '=====> Accuracy Validation=', '0.987999976')
('Epoch: ', '40', '=====> Loss=', '0.007150749')
('Epoch: ', '40', '=====> Accuracy Train=', '0.999199986')
('Epoch: ', '40', '=====> Accuracy Validation=', '0.992399991')
```

```
( 'Epoch: ', '50', '=====> Loss=', '0.003432998')
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.998854518')
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.990400016')
( 'Epoch: ', '60', '=====> Loss=', '0.001935950')
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.999545455')
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.991400003')
( 'Epoch: ', '70', '=====> Loss=', '0.002249211')
( 'Epoch: ', '70', '=====> Accuracy Train=', '0.999436378')
( 'Epoch: ', '70', '=====> Accuracy Validation=', '0.991800010')
( 'Epoch: ', '80', '=====> Loss=', '0.002597852')
( 'Epoch: ', '80', '=====> Accuracy Train=', '0.999236345')
( 'Epoch: ', '80', '=====> Accuracy Validation=', '0.989799976')
( 'Epoch: ', '90', '=====> Loss=', '0.002687383')
( 'Epoch: ', '90', '=====> Accuracy Train=', '0.998981833')
( 'Epoch: ', '90', '=====> Accuracy Validation=', '0.990599990')
( 'Epoch: ', '100', '=====> Loss=', '0.000002554')
( 'Epoch: ', '100', '=====> Accuracy Train=', '1.000000000')
( 'Epoch: ', '100', '=====> Accuracy Validation=', '0.991599977')
```

Training Finished!

Model saved in file: model.ckpt

```
( 'Accuracy Test=', '0.992600024')
```

```
(?, 28, 28, 6)
```

```
(?, 14, 14, 6)
```

```
(?, 10, 10, 16)
```

```
(?, 5, 5, 16)
```

```
(?, 400)
```

Start Training!

```
( 'Epoch: ', '10', '=====> Loss=', '0.033932604')
( 'Epoch: ', '10', '=====> Accuracy Train=', '0.990909100')
( 'Epoch: ', '10', '=====> Accuracy Validation=', '0.987800002')
( 'Epoch: ', '20', '=====> Loss=', '0.011420819')
( 'Epoch: ', '20', '=====> Accuracy Train=', '0.994490921')
( 'Epoch: ', '20', '=====> Accuracy Validation=', '0.986800015')
( 'Epoch: ', '30', '=====> Loss=', '0.004249394')
( 'Epoch: ', '30', '=====> Accuracy Train=', '0.998563647')
( 'Epoch: ', '30', '=====> Accuracy Validation=', '0.990000010')
( 'Epoch: ', '40', '=====> Loss=', '0.003515815')
( 'Epoch: ', '40', '=====> Accuracy Train=', '0.998709083')
( 'Epoch: ', '40', '=====> Accuracy Validation=', '0.990000010')
( 'Epoch: ', '50', '=====> Loss=', '0.000881494')
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.999654531')
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.989199996')
( 'Epoch: ', '60', '=====> Loss=', '0.006840039')
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.998618186')
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.988600016')
( 'Epoch: ', '70', '=====> Loss=', '0.000045993')
( 'Epoch: ', '70', '=====> Accuracy Train=', '1.000000000')
( 'Epoch: ', '70', '=====> Accuracy Validation=', '0.990999997')
( 'Epoch: ', '80', '=====> Loss=', '0.000018666')
( 'Epoch: ', '80', '=====> Accuracy Train=', '1.000000000')
( 'Epoch: ', '80', '=====> Accuracy Validation=', '0.991199970')
( 'Epoch: ', '90', '=====> Loss=', '0.000010945')
( 'Epoch: ', '90', '=====> Accuracy Train=', '1.000000000')
( 'Epoch: ', '90', '=====> Accuracy Validation=', '0.991400003')
( 'Epoch: ', '100', '=====> Loss=', '0.000007403')
( 'Epoch: ', '100', '=====> Accuracy Train=', '1.000000000')
( 'Epoch: ', '100', '=====> Accuracy Validation=', '0.991800010')
```

Training Finished!

Model saved in file: model.ckpt
 ('Accuracy Test=', '0.990300000')
 (?, 28, 28, 6)
 (?, 14, 14, 6)
 (?, 10, 10, 16)
 (?, 5, 5, 16)
 (?, 400)

Start Training!

('Epoch: ', '10', '=====> Loss=', '0.050955012')
 ('Epoch: ', '10', '=====> Accuracy Train=', '0.986490905')
 ('Epoch: ', '10', '=====> Accuracy Validation=', '0.985599995')
 ('Epoch: ', '20', '=====> Loss=', '0.021428636')
 ('Epoch: ', '20', '=====> Accuracy Train=', '0.994727254')
 ('Epoch: ', '20', '=====> Accuracy Validation=', '0.988600016')
 ('Epoch: ', '30', '=====> Loss=', '0.009640858')
 ('Epoch: ', '30', '=====> Accuracy Train=', '0.995000005')
 ('Epoch: ', '30', '=====> Accuracy Validation=', '0.988600016')
 ('Epoch: ', '40', '=====> Loss=', '0.004495570')
 ('Epoch: ', '40', '=====> Accuracy Train=', '0.999090910')
 ('Epoch: ', '40', '=====> Accuracy Validation=', '0.988600016')
 ('Epoch: ', '50', '=====> Loss=', '0.001847927')
 ('Epoch: ', '50', '=====> Accuracy Train=', '0.999454558')
 ('Epoch: ', '50', '=====> Accuracy Validation=', '0.989400029')
 ('Epoch: ', '60', '=====> Loss=', '0.000261703')
 ('Epoch: ', '60', '=====> Accuracy Train=', '1.000000000')
 ('Epoch: ', '60', '=====> Accuracy Validation=', '0.990599990')
 ('Epoch: ', '70', '=====> Loss=', '0.005501417')
 ('Epoch: ', '70', '=====> Accuracy Train=', '0.999618173')
 ('Epoch: ', '70', '=====> Accuracy Validation=', '0.990199983')
 ('Epoch: ', '80', '=====> Loss=', '0.000674863')
 ('Epoch: ', '80', '=====> Accuracy Train=', '0.999963641')
 ('Epoch: ', '80', '=====> Accuracy Validation=', '0.989799976')
 ('Epoch: ', '90', '=====> Loss=', '0.000250022')
 ('Epoch: ', '90', '=====> Accuracy Train=', '0.999599993')
 ('Epoch: ', '90', '=====> Accuracy Validation=', '0.990000010')
 ('Epoch: ', '100', '=====> Loss=', '0.000033286')
 ('Epoch: ', '100', '=====> Accuracy Train=', '1.000000000')
 ('Epoch: ', '100', '=====> Accuracy Validation=', '0.991199970')

Training Finished!

Model saved in file: model.ckpt
 ('Accuracy Test=', '0.989899993')
 (?, 28, 28, 6)
 (?, 14, 14, 6)
 (?, 10, 10, 16)
 (?, 5, 5, 16)
 (?, 400)

Start Training!

('Epoch: ', '10', '=====> Loss=', '0.060508789')
 ('Epoch: ', '10', '=====> Accuracy Train=', '0.983727276')
 ('Epoch: ', '10', '=====> Accuracy Validation=', '0.982999980')
 ('Epoch: ', '20', '=====> Loss=', '0.032258125')
 ('Epoch: ', '20', '=====> Accuracy Train=', '0.988799989')
 ('Epoch: ', '20', '=====> Accuracy Validation=', '0.986599982')
 ('Epoch: ', '30', '=====> Loss=', '0.019336368')
 ('Epoch: ', '30', '=====> Accuracy Train=', '0.995309114')
 ('Epoch: ', '30', '=====> Accuracy Validation=', '0.987999976')


```

('Epoch: ', '40', '=====> Loss=', '0.011388161')
('Epoch: ', '40', '=====> Accuracy Train=', '0.997672737')
('Epoch: ', '40', '=====> Accuracy Validation=', '0.987600029')
('Epoch: ', '50', '=====> Loss=', '0.007056366')
('Epoch: ', '50', '=====> Accuracy Train=', '0.998600006')
('Epoch: ', '50', '=====> Accuracy Validation=', '0.988399982')
('Epoch: ', '60', '=====> Loss=', '0.004877941')
('Epoch: ', '60', '=====> Accuracy Train=', '0.999381840')
('Epoch: ', '60', '=====> Accuracy Validation=', '0.989400029')
('Epoch: ', '70', '=====> Loss=', '0.003708944')
('Epoch: ', '70', '=====> Accuracy Train=', '0.999690890')
('Epoch: ', '70', '=====> Accuracy Validation=', '0.989799976')
('Epoch: ', '80', '=====> Loss=', '0.003286625')
('Epoch: ', '80', '=====> Accuracy Train=', '0.998581827')
('Epoch: ', '80', '=====> Accuracy Validation=', '0.987600029')
('Epoch: ', '90', '=====> Loss=', '0.001272148')
('Epoch: ', '90', '=====> Accuracy Train=', '0.999709070')
('Epoch: ', '90', '=====> Accuracy Validation=', '0.990000010')
('Epoch: ', '100', '=====> Loss=', '0.003630497')
('Epoch: ', '100', '=====> Accuracy Train=', '0.999909103')
('Epoch: ', '100', '=====> Accuracy Validation=', '0.989799976')

```

Training Finished!

Model saved in file: model.ckpt

```
('Accuracy Test=', '0.988600016')
```

```
(?, 28, 28, 6)
```

```
(?, 14, 14, 6)
```

```
(?, 10, 10, 16)
```

```
(?, 5, 5, 16)
```

```
(?, 400)
```

Start Training!

```

('Epoch: ', '10', '=====> Loss=', '0.086312188')
('Epoch: ', '10', '=====> Accuracy Train=', '0.974909067')
('Epoch: ', '10', '=====> Accuracy Validation=', '0.974799991')
('Epoch: ', '20', '=====> Loss=', '0.050865723')
('Epoch: ', '20', '=====> Accuracy Train=', '0.985581815')
('Epoch: ', '20', '=====> Accuracy Validation=', '0.982599974')
('Epoch: ', '30', '=====> Loss=', '0.036149178')
('Epoch: ', '30', '=====> Accuracy Train=', '0.989600003')
('Epoch: ', '30', '=====> Accuracy Validation=', '0.985199988')
('Epoch: ', '40', '=====> Loss=', '0.026947668')
('Epoch: ', '40', '=====> Accuracy Train=', '0.992600024')
('Epoch: ', '40', '=====> Accuracy Validation=', '0.987399995')
('Epoch: ', '50', '=====> Loss=', '0.019950968')
('Epoch: ', '50', '=====> Accuracy Train=', '0.992545426')
('Epoch: ', '50', '=====> Accuracy Validation=', '0.987399995')
('Epoch: ', '60', '=====> Loss=', '0.015298240')
('Epoch: ', '60', '=====> Accuracy Train=', '0.995927274')
('Epoch: ', '60', '=====> Accuracy Validation=', '0.990199983')
('Epoch: ', '70', '=====> Loss=', '0.011793830')
('Epoch: ', '70', '=====> Accuracy Train=', '0.997309089')
('Epoch: ', '70', '=====> Accuracy Validation=', '0.989400029')
('Epoch: ', '80', '=====> Loss=', '0.008663446')
('Epoch: ', '80', '=====> Accuracy Train=', '0.998527288')
('Epoch: ', '80', '=====> Accuracy Validation=', '0.989400029')
('Epoch: ', '90', '=====> Loss=', '0.006750681')
('Epoch: ', '90', '=====> Accuracy Train=', '0.999054551')
('Epoch: ', '90', '=====> Accuracy Validation=', '0.988600016')

```

```

('Epoch: ', '100', '=====> Loss=', '0.004504250')
('Epoch: ', '100', '=====> Accuracy Train=', '0.999236345')
('Epoch: ', '100', '=====> Accuracy Validation=', '0.988600016')
Training Finished!
Model saved in file: model.ckpt
('Accuracy Test=', '0.988699973')
(?, 28, 28, 6)
(?, 14, 14, 6)
(?, 10, 10, 16)
(?, 5, 5, 16)
(?, 400)
Start Training!
('Epoch: ', '10', '=====> Loss=', '0.193546795')
('Epoch: ', '10', '=====> Accuracy Train=', '0.947363615')
('Epoch: ', '10', '=====> Accuracy Validation=', '0.950399995')
('Epoch: ', '20', '=====> Loss=', '0.100437300')
('Epoch: ', '20', '=====> Accuracy Train=', '0.970163643')
('Epoch: ', '20', '=====> Accuracy Validation=', '0.973200023')
('Epoch: ', '30', '=====> Loss=', '0.072692497')
('Epoch: ', '30', '=====> Accuracy Train=', '0.978454530')
('Epoch: ', '30', '=====> Accuracy Validation=', '0.977599978')
('Epoch: ', '40', '=====> Loss=', '0.057596801')
('Epoch: ', '40', '=====> Accuracy Train=', '0.983600020')
('Epoch: ', '40', '=====> Accuracy Validation=', '0.981599987')
('Epoch: ', '50', '=====> Loss=', '0.046975700')
('Epoch: ', '50', '=====> Accuracy Train=', '0.986872733')
('Epoch: ', '50', '=====> Accuracy Validation=', '0.984600008')
('Epoch: ', '60', '=====> Loss=', '0.039299161')
('Epoch: ', '60', '=====> Accuracy Train=', '0.989109099')
('Epoch: ', '60', '=====> Accuracy Validation=', '0.984000027')
('Epoch: ', '70', '=====> Loss=', '0.032884745')
('Epoch: ', '70', '=====> Accuracy Train=', '0.990872741')
('Epoch: ', '70', '=====> Accuracy Validation=', '0.985800028')
('Epoch: ', '80', '=====> Loss=', '0.027923221')
('Epoch: ', '80', '=====> Accuracy Train=', '0.992036343')
('Epoch: ', '80', '=====> Accuracy Validation=', '0.986999989')
('Epoch: ', '90', '=====> Loss=', '0.023686793')
('Epoch: ', '90', '=====> Accuracy Train=', '0.993581831')
('Epoch: ', '90', '=====> Accuracy Validation=', '0.986999989')
('Epoch: ', '100', '=====> Loss=', '0.020125311')
('Epoch: ', '100', '=====> Accuracy Train=', '0.994636357')
('Epoch: ', '100', '=====> Accuracy Validation=', '0.988200009')
Training Finished!
Model saved in file: model.ckpt
('Accuracy Test=', '0.987399995')
(?, 28, 28, 6)
(?, 14, 14, 6)
(?, 10, 10, 16)
(?, 5, 5, 16)
(?, 400)
Start Training!
('Epoch: ', '10', '=====> Loss=', '0.492886786')
('Epoch: ', '10', '=====> Accuracy Train=', '0.875945449')
('Epoch: ', '10', '=====> Accuracy Validation=', '0.880999982')
('Epoch: ', '20', '=====> Loss=', '0.204249872')
('Epoch: ', '20', '=====> Accuracy Train=', '0.941454530')
('Epoch: ', '20', '=====> Accuracy Validation=', '0.949800014')

```

```
( 'Epoch: ', '30', '=====> Loss=', '0.131304923')
( 'Epoch: ', '30', '=====> Accuracy Train=', '0.961654544')
( 'Epoch: ', '30', '=====> Accuracy Validation=', '0.966000021')
( 'Epoch: ', '40', '=====> Loss=', '0.098910232')
( 'Epoch: ', '40', '=====> Accuracy Train=', '0.971036375')
( 'Epoch: ', '40', '=====> Accuracy Validation=', '0.972400010')
( 'Epoch: ', '50', '=====> Loss=', '0.080679185')
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.976000011')
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.975799978')
( 'Epoch: ', '60', '=====> Loss=', '0.068668278')
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.980072737')
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.978399992')
( 'Epoch: ', '70', '=====> Loss=', '0.059532278')
( 'Epoch: ', '70', '=====> Accuracy Train=', '0.982290924')
( 'Epoch: ', '70', '=====> Accuracy Validation=', '0.981000006')
( 'Epoch: ', '80', '=====> Loss=', '0.052414630')
( 'Epoch: ', '80', '=====> Accuracy Train=', '0.984436393')
( 'Epoch: ', '80', '=====> Accuracy Validation=', '0.983399987')
( 'Epoch: ', '90', '=====> Loss=', '0.046660844')
( 'Epoch: ', '90', '=====> Accuracy Train=', '0.985581815')
( 'Epoch: ', '90', '=====> Accuracy Validation=', '0.983600020')
( 'Epoch: ', '100', '=====> Loss=', '0.041655485')
( 'Epoch: ', '100', '=====> Accuracy Train=', '0.987999976')
( 'Epoch: ', '100', '=====> Accuracy Validation=', '0.984000027')
```

Training Finished!

Model saved in file: model.ckpt

```
( 'Accuracy Test=', '0.984899998')
```

```
(?, 28, 28, 6)
```

```
(?, 14, 14, 6)
```

```
(?, 10, 10, 16)
```

```
(?, 5, 5, 16)
```

```
(?, 400)
```

Start Training!

```
( 'Epoch: ', '10', '=====> Loss=', '0.199729827')
( 'Epoch: ', '10', '=====> Accuracy Train=', '0.943654537')
( 'Epoch: ', '10', '=====> Accuracy Validation=', '0.946600020')
( 'Epoch: ', '20', '=====> Loss=', '0.122782507')
( 'Epoch: ', '20', '=====> Accuracy Train=', '0.964781821')
( 'Epoch: ', '20', '=====> Accuracy Validation=', '0.966799974')
( 'Epoch: ', '30', '=====> Loss=', '0.094191241')
( 'Epoch: ', '30', '=====> Accuracy Train=', '0.971581817')
( 'Epoch: ', '30', '=====> Accuracy Validation=', '0.972000003')
( 'Epoch: ', '40', '=====> Loss=', '0.079302760')
( 'Epoch: ', '40', '=====> Accuracy Train=', '0.976181805')
( 'Epoch: ', '40', '=====> Accuracy Validation=', '0.976199985')
( 'Epoch: ', '50', '=====> Loss=', '0.069194937')
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.979818165')
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.978600025')
( 'Epoch: ', '60', '=====> Loss=', '0.061677245')
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.981054544')
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.979799986')
( 'Epoch: ', '70', '=====> Loss=', '0.056220320')
( 'Epoch: ', '70', '=====> Accuracy Train=', '0.983690917')
( 'Epoch: ', '70', '=====> Accuracy Validation=', '0.981199980')
( 'Epoch: ', '80', '=====> Loss=', '0.051395045')
( 'Epoch: ', '80', '=====> Accuracy Train=', '0.984781802')
( 'Epoch: ', '80', '=====> Accuracy Validation=', '0.983200014')
```

```
( 'Epoch: ', '90', '=====> Loss=', '0.047391328' )
( 'Epoch: ', '90', '=====> Accuracy Train=', '0.985981822' )
( 'Epoch: ', '90', '=====> Accuracy Validation=', '0.981400013' )
( 'Epoch: ', '100', '=====> Loss=', '0.043857428' )
( 'Epoch: ', '100', '=====> Accuracy Train=', '0.986090899' )
( 'Epoch: ', '100', '=====> Accuracy Validation=', '0.981800020' )
```

Training Finished!

Model saved in file: model.ckpt

```
( 'Accuracy Test=', '0.982699990' )
```

```
( ?, 28, 28, 6 )
```

```
( ?, 14, 14, 6 )
```

```
( ?, 10, 10, 16 )
```

```
( ?, 5, 5, 16 )
```

```
( ?, 400 )
```

Start Training!

```
( 'Epoch: ', '10', '=====> Loss=', '0.431872474' )
( 'Epoch: ', '10', '=====> Accuracy Train=', '0.882218182' )
( 'Epoch: ', '10', '=====> Accuracy Validation=', '0.888599992' )
( 'Epoch: ', '20', '=====> Loss=', '0.249007264' )
( 'Epoch: ', '20', '=====> Accuracy Train=', '0.927090883' )
( 'Epoch: ', '20', '=====> Accuracy Validation=', '0.933200002' )
( 'Epoch: ', '30', '=====> Loss=', '0.187337504' )
( 'Epoch: ', '30', '=====> Accuracy Train=', '0.945490897' )
( 'Epoch: ', '30', '=====> Accuracy Validation=', '0.950600028' )
( 'Epoch: ', '40', '=====> Loss=', '0.151950350' )
( 'Epoch: ', '40', '=====> Accuracy Train=', '0.955072701' )
( 'Epoch: ', '40', '=====> Accuracy Validation=', '0.959200025' )
( 'Epoch: ', '50', '=====> Loss=', '0.129838842' )
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.961418211' )
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.963199973' )
( 'Epoch: ', '60', '=====> Loss=', '0.114747240' )
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.965781808' )
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.966000021' )
( 'Epoch: ', '70', '=====> Loss=', '0.103926300' )
( 'Epoch: ', '70', '=====> Accuracy Train=', '0.968963623' )
( 'Epoch: ', '70', '=====> Accuracy Validation=', '0.969600022' )
( 'Epoch: ', '80', '=====> Loss=', '0.095046490' )
( 'Epoch: ', '80', '=====> Accuracy Train=', '0.971181810' )
( 'Epoch: ', '80', '=====> Accuracy Validation=', '0.972000003' )
( 'Epoch: ', '90', '=====> Loss=', '0.088118315' )
( 'Epoch: ', '90', '=====> Accuracy Train=', '0.972018182' )
( 'Epoch: ', '90', '=====> Accuracy Validation=', '0.971599996' )
( 'Epoch: ', '100', '=====> Loss=', '0.082138759' )
( 'Epoch: ', '100', '=====> Accuracy Train=', '0.975436389' )
( 'Epoch: ', '100', '=====> Accuracy Validation=', '0.974600017' )
```

Training Finished!

Model saved in file: model.ckpt

```
( 'Accuracy Test=', '0.975099981' )
```

```
( ?, 28, 28, 6 )
```

```
( ?, 14, 14, 6 )
```

```
( ?, 10, 10, 16 )
```

```
( ?, 5, 5, 16 )
```

```
( ?, 400 )
```

Start Training!

```
( 'Epoch: ', '10', '=====> Loss=', '2.301367530' )
( 'Epoch: ', '10', '=====> Accuracy Train=', '0.112345457' )
( 'Epoch: ', '10', '=====> Accuracy Validation=', '0.112599999' )
```

```
( 'Epoch: ', '20', '=====> Loss=', '2.301346316')
( 'Epoch: ', '20', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '20', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '30', '=====> Loss=', '2.301324679')
( 'Epoch: ', '30', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '30', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '40', '=====> Loss=', '2.301269561')
( 'Epoch: ', '40', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '40', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '50', '=====> Loss=', '2.301277550')
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '60', '=====> Loss=', '2.301219232')
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '70', '=====> Loss=', '2.301250912')
( 'Epoch: ', '70', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '70', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '80', '=====> Loss=', '2.301198761')
( 'Epoch: ', '80', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '80', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '90', '=====> Loss=', '2.301161070')
( 'Epoch: ', '90', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '90', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '100', '=====> Loss=', '2.301107464')
( 'Epoch: ', '100', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '100', '=====> Accuracy Validation=', '0.112599999')
```

Training Finished!

Model saved in file: model.ckpt

```
( 'Accuracy Test=', '0.113499999')
```

```
(?, 28, 28, 6)
```

```
(?, 14, 14, 6)
```

```
(?, 10, 10, 16)
```

```
(?, 5, 5, 16)
```

```
(?, 400)
```

Start Training!

```
( 'Epoch: ', '10', '=====> Loss=', '2.301164296')
( 'Epoch: ', '10', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '10', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '20', '=====> Loss=', '2.301161988')
( 'Epoch: ', '20', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '20', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '30', '=====> Loss=', '2.301127632')
( 'Epoch: ', '30', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '30', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '40', '=====> Loss=', '2.301125973')
( 'Epoch: ', '40', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '40', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '50', '=====> Loss=', '2.301113508')
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '60', '=====> Loss=', '2.301083235')
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '70', '=====> Loss=', '2.301081216')
( 'Epoch: ', '70', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '70', '=====> Accuracy Validation=', '0.112599999')
```

```
('Epoch: ', '80', '=====> Loss=', '2.301062383')
('Epoch: ', '80', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '80', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '90', '=====> Loss=', '2.301049383')
('Epoch: ', '90', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '90', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '100', '=====> Loss=', '2.301056336')
('Epoch: ', '100', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '100', '=====> Accuracy Validation=', '0.112599999')
```

Training Finished!

Model saved in file: model.ckpt

```
('Accuracy Test=', '0.113499999')
```

```
(?, 28, 28, 6)
```

```
(?, 14, 14, 6)
```

```
(?, 10, 10, 16)
```

```
(?, 5, 5, 16)
```

```
(?, 400)
```

Start Training!

```
('Epoch: ', '10', '=====> Loss=', '2.159965737')
('Epoch: ', '10', '=====> Accuracy Train=', '0.481400013')
('Epoch: ', '10', '=====> Accuracy Validation=', '0.488200009')
('Epoch: ', '20', '=====> Loss=', '1.218465333')
('Epoch: ', '20', '=====> Accuracy Train=', '0.750472724')
('Epoch: ', '20', '=====> Accuracy Validation=', '0.753000021')
('Epoch: ', '30', '=====> Loss=', '0.526025555')
('Epoch: ', '30', '=====> Accuracy Train=', '0.858127296')
('Epoch: ', '30', '=====> Accuracy Validation=', '0.858600020')
('Epoch: ', '40', '=====> Loss=', '0.393398697')
('Epoch: ', '40', '=====> Accuracy Train=', '0.887636364')
('Epoch: ', '40', '=====> Accuracy Validation=', '0.892400026')
('Epoch: ', '50', '=====> Loss=', '0.336847887')
('Epoch: ', '50', '=====> Accuracy Train=', '0.901527286')
('Epoch: ', '50', '=====> Accuracy Validation=', '0.910399973')
('Epoch: ', '60', '=====> Loss=', '0.301628755')
('Epoch: ', '60', '=====> Accuracy Train=', '0.910018206')
('Epoch: ', '60', '=====> Accuracy Validation=', '0.917400002')
('Epoch: ', '70', '=====> Loss=', '0.275532730')
('Epoch: ', '70', '=====> Accuracy Train=', '0.916545451')
('Epoch: ', '70', '=====> Accuracy Validation=', '0.922999978')
('Epoch: ', '80', '=====> Loss=', '0.254295892')
('Epoch: ', '80', '=====> Accuracy Train=', '0.922709107')
('Epoch: ', '80', '=====> Accuracy Validation=', '0.927999973')
('Epoch: ', '90', '=====> Loss=', '0.236312885')
('Epoch: ', '90', '=====> Accuracy Train=', '0.928418159')
('Epoch: ', '90', '=====> Accuracy Validation=', '0.934400022')
('Epoch: ', '100', '=====> Loss=', '0.221003828')
('Epoch: ', '100', '=====> Accuracy Train=', '0.933418155')
('Epoch: ', '100', '=====> Accuracy Validation=', '0.938600004')
```

Training Finished!

Model saved in file: model.ckpt

```
('Accuracy Test=', '0.936600029')
```

```
(?, 28, 28, 6)
```

```
(?, 14, 14, 6)
```

```
(?, 10, 10, 16)
```

```
(?, 5, 5, 16)
```

```
(?, 400)
```

Start Training!

```
( 'Epoch: ', '10', '=====> Loss=', '2.234068918')
( 'Epoch: ', '10', '=====> Accuracy Train=', '0.225818187')
( 'Epoch: ', '10', '=====> Accuracy Validation=', '0.226799995')
( 'Epoch: ', '20', '=====> Loss=', '2.126468317')
( 'Epoch: ', '20', '=====> Accuracy Train=', '0.418672740')
( 'Epoch: ', '20', '=====> Accuracy Validation=', '0.428799987')
( 'Epoch: ', '30', '=====> Loss=', '1.867886189')
( 'Epoch: ', '30', '=====> Accuracy Train=', '0.557472706')
( 'Epoch: ', '30', '=====> Accuracy Validation=', '0.574599981')
( 'Epoch: ', '40', '=====> Loss=', '1.326568398')
( 'Epoch: ', '40', '=====> Accuracy Train=', '0.702836335')
( 'Epoch: ', '40', '=====> Accuracy Validation=', '0.717199981')
( 'Epoch: ', '50', '=====> Loss=', '0.849893839')
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.793236375')
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.805400014')
( 'Epoch: ', '60', '=====> Loss=', '0.623423636')
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.834145427')
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.842000008')
( 'Epoch: ', '70', '=====> Loss=', '0.510477006')
( 'Epoch: ', '70', '=====> Accuracy Train=', '0.857763648')
( 'Epoch: ', '70', '=====> Accuracy Validation=', '0.866800010')
( 'Epoch: ', '80', '=====> Loss=', '0.444419014')
( 'Epoch: ', '80', '=====> Accuracy Train=', '0.873145461')
( 'Epoch: ', '80', '=====> Accuracy Validation=', '0.883400023')
( 'Epoch: ', '90', '=====> Loss=', '0.400024359')
( 'Epoch: ', '90', '=====> Accuracy Train=', '0.884018183')
( 'Epoch: ', '90', '=====> Accuracy Validation=', '0.894200027')
( 'Epoch: ', '100', '=====> Loss=', '0.367043349')
( 'Epoch: ', '100', '=====> Accuracy Train=', '0.892545462')
( 'Epoch: ', '100', '=====> Accuracy Validation=', '0.902199984')
```

Training Finished!

Model saved in file: model.ckpt

```
( 'Accuracy Test=', '0.900099993')
```

```
(?, 28, 28, 6)
```

```
(?, 14, 14, 6)
```

```
(?, 10, 10, 16)
```

```
(?, 5, 5, 16)
```

```
(?, 400)
```

Start Training!

```
( 'Epoch: ', '10', '=====> Loss=', '2.301447420')
( 'Epoch: ', '10', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '10', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '20', '=====> Loss=', '2.301223905')
( 'Epoch: ', '20', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '20', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '30', '=====> Loss=', '2.301208252')
( 'Epoch: ', '30', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '30', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '40', '=====> Loss=', '2.301218171')
( 'Epoch: ', '40', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '40', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '50', '=====> Loss=', '2.301205788')
( 'Epoch: ', '50', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '50', '=====> Accuracy Validation=', '0.112599999')
( 'Epoch: ', '60', '=====> Loss=', '2.301189613')
( 'Epoch: ', '60', '=====> Accuracy Train=', '0.112345457')
( 'Epoch: ', '60', '=====> Accuracy Validation=', '0.112599999')
```

```
('Epoch: ', '70', '=====> Loss=', '2.301207670')
('Epoch: ', '70', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '70', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '80', '=====> Loss=', '2.301218076')
('Epoch: ', '80', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '80', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '90', '=====> Loss=', '2.301189397')
('Epoch: ', '90', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '90', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '100', '=====> Loss=', '2.301171967')
('Epoch: ', '100', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '100', '=====> Accuracy Validation=', '0.112599999')
```

Training Finished!

Model saved in file: model.ckpt

```
('Accuracy Test=', '0.113499999')
```

```
(?, 28, 28, 6)
```

```
(?, 14, 14, 6)
```

```
(?, 10, 10, 16)
```

```
(?, 5, 5, 16)
```

```
(?, 400)
```

Start Training!

```
('Epoch: ', '10', '=====> Loss=', '2.309031929')
('Epoch: ', '10', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '10', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '20', '=====> Loss=', '2.302276206')
('Epoch: ', '20', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '20', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '30', '=====> Loss=', '2.301395521')
('Epoch: ', '30', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '30', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '40', '=====> Loss=', '2.301304812')
('Epoch: ', '40', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '40', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '50', '=====> Loss=', '2.301294195')
('Epoch: ', '50', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '50', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '60', '=====> Loss=', '2.301276709')
('Epoch: ', '60', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '60', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '70', '=====> Loss=', '2.301261499')
('Epoch: ', '70', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '70', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '80', '=====> Loss=', '2.301266454')
('Epoch: ', '80', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '80', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '90', '=====> Loss=', '2.301277130')
('Epoch: ', '90', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '90', '=====> Accuracy Validation=', '0.112599999')
('Epoch: ', '100', '=====> Loss=', '2.301284024')
('Epoch: ', '100', '=====> Accuracy Train=', '0.112345457')
('Epoch: ', '100', '=====> Accuracy Validation=', '0.112599999')
```

Training Finished!

Model saved in file: model.ckpt

```
('Accuracy Test=', '0.113499999')
```



```
In [9]: final_results
```

```
Out[9]: [{'activation_function': 'relu',
          'batch_size': 50,
          'elapsed_time': 2719.1444079875946,
          'learning_rate': 0.001,
          'optimizer': 'adam',
          'test_accuracy': 0.98979998,
          'training_accuracy': 0.99956363,
          'validation_accuracy': 0.99956363},
         {'activation_function': 'relu',
          'batch_size': 128,
          'elapsed_time': 2140.7165479660034,
          'learning_rate': 0.001,
          'optimizer': 'adam',
          'test_accuracy': 0.99260002,
          'training_accuracy': 1.0,
          'validation_accuracy': 1.0},
         {'activation_function': 'sigmoid',
          'batch_size': 50,
          'elapsed_time': 2590.7476279735565,
          'learning_rate': 0.001,
          'optimizer': 'adam',
          'test_accuracy': 0.9903,
          'training_accuracy': 1.0,
          'validation_accuracy': 1.0},
         {'activation_function': 'sigmoid',
          'batch_size': 128,
          'elapsed_time': 2174.403846025467,
          'learning_rate': 0.001,
          'optimizer': 'adam',
          'test_accuracy': 0.98989999,
          'training_accuracy': 1.0,
          'validation_accuracy': 1.0},
         {'activation_function': 'relu',
          'batch_size': 50,
          'elapsed_time': 2707.127268075943,
          'learning_rate': 0.0001,
          'optimizer': 'adam',
          'test_accuracy': 0.98860002,
          'training_accuracy': 0.9999091,
          'validation_accuracy': 0.9999091},
         {'activation_function': 'relu',
          'batch_size': 128,
          'elapsed_time': 2144.91255903244,
          'learning_rate': 0.0001,
          'optimizer': 'adam',
          'test_accuracy': 0.98869997,
          'training_accuracy': 0.99923635,
          'validation_accuracy': 0.99923635},
         {'activation_function': 'sigmoid',
          'batch_size': 50,
          'elapsed_time': 2667.0797259807587,
          'learning_rate': 0.0001,
          'optimizer': 'adam',
          'test_accuracy': 0.9874,
          'training_accuracy': 0.99463636,
          'validation_accuracy': 0.99463636},
         {'activation_function': 'sigmoid',
```

```
'batch_size': 128,
'elapsed_time': 2166.4854328632355,
'learning_rate': 0.0001,
'optimizer': 'adam',
'test_accuracy': 0.9849,
'training_accuracy': 0.98799998,
'validation_accuracy': 0.98799998},
{'activation_function': 'relu',
'batch_size': 50,
'elapsed_time': 2705.517254114151,
'learning_rate': 0.001,
'optimizer': 'gradient_descent',
'test_accuracy': 0.98269999,
'training_accuracy': 0.9860909,
'validation_accuracy': 0.9860909},
{'activation_function': 'relu',
'batch_size': 128,
'elapsed_time': 2159.0516889095306,
'learning_rate': 0.001,
'optimizer': 'gradient_descent',
'test_accuracy': 0.97509998,
'training_accuracy': 0.97543639,
'validation_accuracy': 0.97543639},
{'activation_function': 'sigmoid',
'batch_size': 50,
'elapsed_time': 2701.9426329135895,
'learning_rate': 0.001,
'optimizer': 'gradient_descent',
'test_accuracy': 0.1135,
'training_accuracy': 0.11234546,
'validation_accuracy': 0.11234546},
{'activation_function': 'sigmoid',
'batch_size': 128,
'elapsed_time': 2153.0241179466248,
'learning_rate': 0.001,
'optimizer': 'gradient_descent',
'test_accuracy': 0.1135,
'training_accuracy': 0.11234546,
'validation_accuracy': 0.11234546},
{'activation_function': 'relu',
'batch_size': 50,
'elapsed_time': 2709.723870038986,
'learning_rate': 0.0001,
'optimizer': 'gradient_descent',
'test_accuracy': 0.93660003,
'training_accuracy': 0.93341815,
'validation_accuracy': 0.93341815},
{'activation_function': 'relu',
'batch_size': 128,
'elapsed_time': 2159.801533937454,
'learning_rate': 0.0001,
'optimizer': 'gradient_descent',
'test_accuracy': 0.90009999,
'training_accuracy': 0.89254546,
'validation_accuracy': 0.89254546},
{'activation_function': 'sigmoid',
'batch_size': 50,
```

```
'elapsed_time': 2715.5786321163177,
'learning_rate': 0.0001,
'optimizer': 'gradient_descent',
'test_accuracy': 0.1135,
'training_accuracy': 0.11234546,
'validation_accuracy': 0.11234546},
{'activation_function': 'sigmoid',
'batch_size': 128,
'elapsed_time': 2195.4368810653687,
'learning_rate': 0.0001,
'optimizer': 'gradient_descent',
'test_accuracy': 0.1135,
'training_accuracy': 0.11234546,
'validation_accuracy': 0.11234546}]
```

Comment

Here we have seen that the relu performs the best. Then, we also seen that the the sigmoid, combined with a low learning rate, takes an infinite amount of time to reach the optimum.

```
{'activation_function': 'relu',
'batch_size': 128,
'elapsed_time': 2140.7165479660034,
'learning_rate': 0.001,
'optimizer': 'adam',
'test_accuracy': 0.99260002,
'training_accuracy': 1.0,
'validation_accuracy': 1.0},
```

Regarding the batch size, the best configuration (learning rate=0.001, adam optimizer) achieved a better result with a larger batch size. Generally, we have not seen great differences. The most important thing is that with a littler batch size we need more time to train.

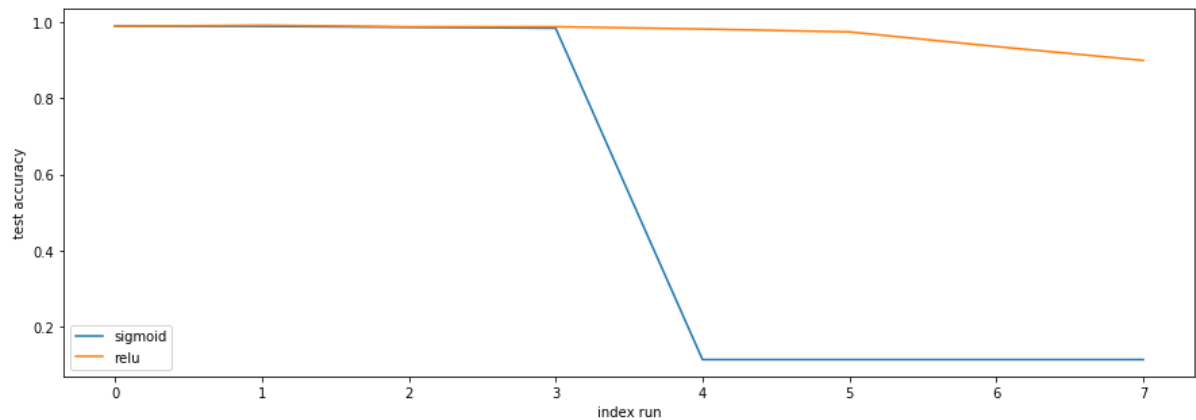
Regarding the learning rate, we achieved the best result with a learning rate of 0.001, but also in this case the answer is not certain. The final accuracy depends from a combination of factors and we can't find something like a proportional behaviour.

Special cases: when we use the stochastic gradient descent with a low learning rate and the sigmoid activation function, the accuracy is always really low.

As a general rule, the adam optimizer can achieve a better acuracy in a lower amount of time, but we need to take care of carefully choosing the learning rate.

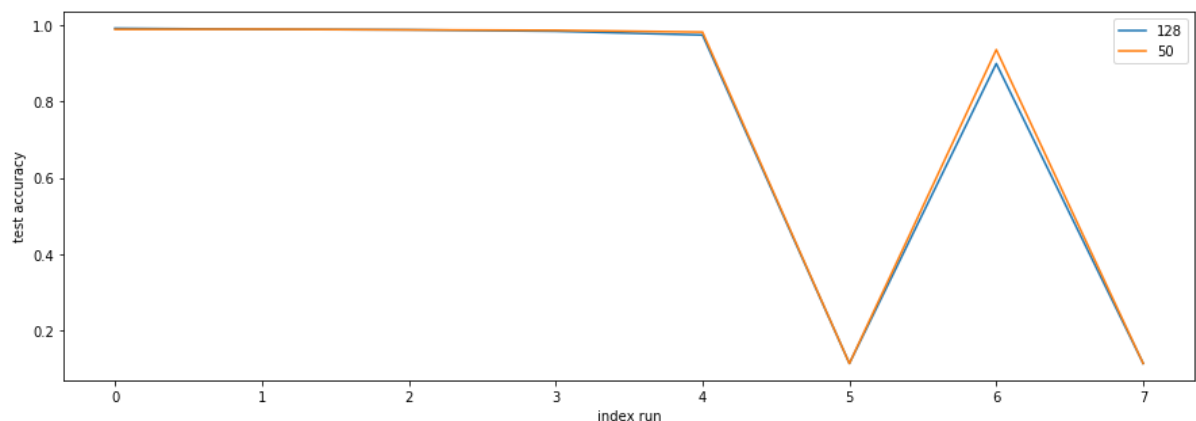
```
In [23]: with open('json.json','r') as input_fp:
         results = json.load(input_fp)
```

```
In [17]: import matplotlib.pyplot as plt
sigmoid = [x for x in results if x['activation_function']=='sigmoid']
relu = [x for x in results if x['activation_function']!='sigmoid']
plt.figure(figsize=(15,5))
plt.plot(range(len(sigmoid)),[x['test_accuracy'] for x in sigmoid])
plt.plot(range(len(sigmoid)),[x['test_accuracy'] for x in relu])
plt.legend(['sigmoid','relu'])
plt.ylabel('test accuracy')
plt.xlabel('index run')
plt.show()
```



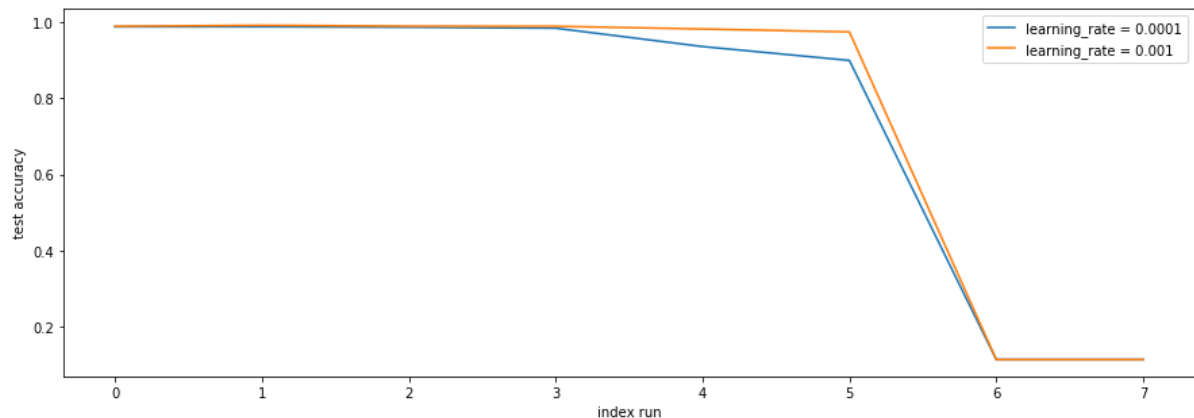
The relu is always equal or better than the sigmoid

```
In [16]: import matplotlib.pyplot as plt
a = [x for x in results if x['batch_size']== 128]
b = [x for x in results if x['batch_size']!=128]
plt.figure(figsize=(15,5))
plt.plot(range(len(a)),[x['test_accuracy'] for x in a])
plt.plot(range(len(a)),[x['test_accuracy'] for x in b])
plt.legend(['128','50'])
plt.ylabel('test accuracy')
plt.xlabel('index run')
plt.show()
```



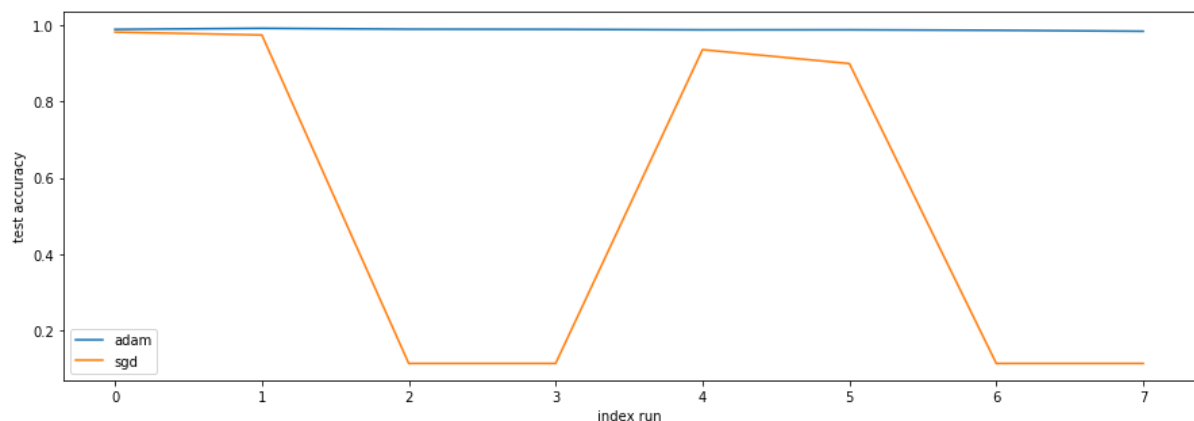
Test accuracy does not change when we change batch size

```
In [18]: import matplotlib.pyplot as plt
a = [x for x in results if x['learning_rate']== 0.0001]
b = [x for x in results if x['learning_rate'] !=0.0001]
plt.figure(figsize=(15,5))
plt.plot(range(len(a)),[x['test_accuracy'] for x in a])
plt.plot(range(len(a)),[x['test_accuracy'] for x in b])
plt.legend(['learning_rate = 0.0001','learning_rate = 0.001'])
plt.ylabel('test accuracy')
plt.xlabel('index run')
plt.show()
```



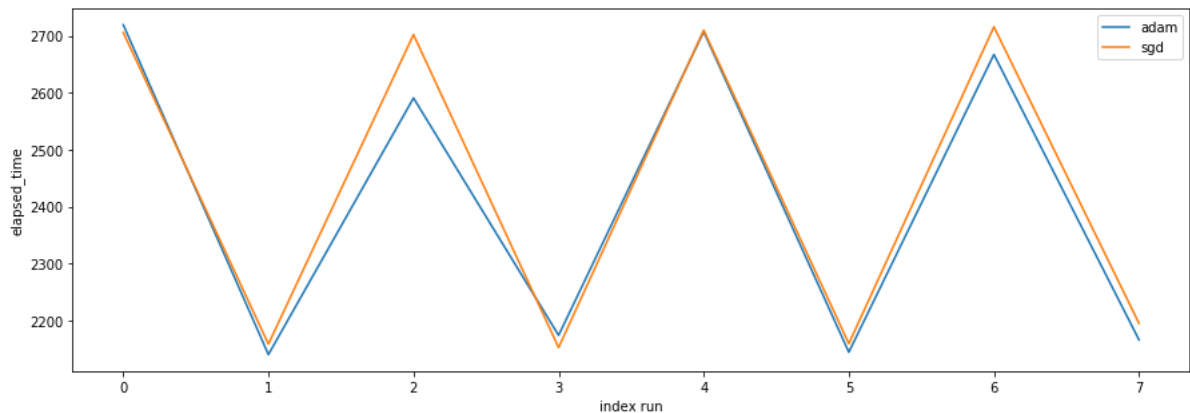
An higher learning rate is better, in this specifi case. Later we will see that this is not always the case.

```
In [19]: import matplotlib.pyplot as plt
a = [x for x in results if x['optimizer']== 'adam']
b = [x for x in results if x['optimizer'] != 'adam']
plt.figure(figsize=(15,5))
plt.plot(range(len(a)),[x['test_accuracy'] for x in a])
plt.plot(range(len(a)),[x['test_accuracy'] for x in b])
plt.legend(['adam','sgd'])
plt.ylabel('test accuracy')
plt.xlabel('index run')
plt.show()
```



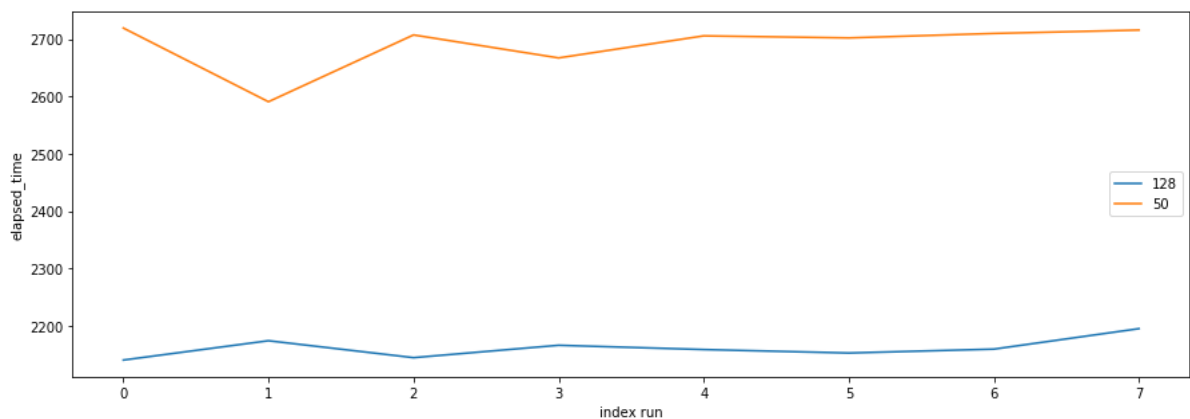
Adam is always better than Stochastic gradient descent

```
In [21]: import matplotlib.pyplot as plt
a = [x for x in results if x['optimizer'] == 'adam']
b = [x for x in results if x['optimizer'] != 'adam']
plt.figure(figsize=(15,5))
plt.plot(range(len(a)), [x['elapsed_time'] for x in a])
plt.plot(range(len(a)), [x['elapsed_time'] for x in b])
plt.legend(['adam', 'sgd'])
plt.ylabel('elapsed_time')
plt.xlabel('index run')
plt.show()
```



Sometimes adam is vaster than SGD

```
In [22]: import matplotlib.pyplot as plt
a = [x for x in results if x['batch_size'] == 128]
b = [x for x in results if x['batch_size'] != 128]
plt.figure(figsize=(15,5))
plt.plot(range(len(a)), [x['elapsed_time'] for x in a])
plt.plot(range(len(a)), [x['elapsed_time'] for x in b])
plt.legend(['128', '50'])
plt.ylabel('elapsed_time')
plt.xlabel('index run')
plt.show()
```



Bigger batches mean less training time, this is actually a good news if we consider that the batch dimension does not has a big influence on the final accuracy

Question 2.2.2 What about applying a dropout layer on the Fully connected layer and then retraining the model with the best Optimizer and parameters(Learning rate and Batch size) obtained in *Question 2.2.1* ? (probability to keep units=0.75). For this stage ensure that the keep prob is set to 1.0 to evaluate the performance of the network including all nodes.


```

In [16]: # https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide
-To-Understanding-Convolutional-Neural-Networks/
# https://adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/

def LeNet5_Model(data,keep_prob,activation_function=tf.nn.sigmoid):

    # layer 1 param
    conv1_weights = weight_variable([5,5,1,6])
    conv1_bias = bias_variable([6])

    # layer 2 param
    conv2_weights = weight_variable([5,5,6,16])
    conv2_bias = bias_variable([16])

    # layer 3 param
    layer3_weights = weight_variable([400, 120])
    layer3_bias = bias_variable([120])

    # layer 4 param
    layer4_weights = weight_variable([120, 84])
    layer4_bias = bias_variable([84])

    # layer 5 param
    layer5_weights = weight_variable([84, 10])
    layer5_bias = bias_variable([10])

    with tf.name_scope('Model'):
        with tf.name_scope('Layer1'):
            conv1 =
tf.nn.conv2d(input=data,filter=conv1_weights,strides=
[1,1,1,1],padding='SAME')
            print(conv1.shape)
            sigmoid1 = activation_function(conv1 + conv1_bias)
            pool1 = tf.nn.max_pool(sigmoid1,ksize=[1, 2, 2, 1],strides=
[1, 2, 2, 1],padding='VALID')
            print(pool1.shape)

            with tf.name_scope('Layer2'):
                conv2 = tf.nn.conv2d(input=pool1,filter=conv2_weights,stride
s=[1,1,1,1],padding='VALID')
                print(conv2.shape)
                sigmoid2 = activation_function(conv2 + conv2_bias)
                pool2 = tf.nn.max_pool(sigmoid2,ksize=[1, 2, 2, 1],strides=
[1, 2, 2, 1],padding='VALID')
                print(pool2.shape)

            with tf.name_scope('Flatten'):
                flat_inputs = tf.contrib.layers.flatten(pool2)

```

```
        print(flat_inputs.shape)

    with tf.name_scope('Layer3'):
        out3 = activation_function(tf.matmul(flat_inputs, layer3_weights) + layer3_bias)

    with tf.name_scope('Layer4'):
        out4 = activation_function(tf.matmul(out3, layer4_weights) + layer4_bias)

    with tf.name_scope('Layer5'):
        out_drop = tf.nn.dropout(out4, keep_prob)
        pred = tf.nn.softmax(tf.matmul(out_drop, layer5_weights) + layer5_bias) # Softmax
    return pred
```

```

In [19]: import numpy as np
# Initializing the variables
def train(learning_rate, training_epochs, batch_size, display_step, optimizer_method=tf.train.GradientDescentOptimizer, activation_function=tf.nn.sigmoid):
    tf.reset_default_graph()
    # Initializing the session

    logs_path = 'log_files/' # useful for tensorboard

    # tf Graph Input: mnist data image of shape 28*28=784
    x = tf.placeholder(tf.float32, [None, 28, 28, 1], name='InputData')
    # 0-9 digits recognition, 10 classes
    y = tf.placeholder(tf.float32, [None, 10], name='LabelData')

    keep_prob = tf.placeholder(tf.float32)

    # Construct model and encapsulating all ops into scopes, making Tensorboard's Graph visualization more convenient
    with tf.name_scope('Model'):
        # Model
        pred = LeNet5_Model(x, keep_prob, activation_function=activation_function)
    with tf.name_scope('Loss'):
        # Minimize error using cross entropy
        # Minimize error using cross entropy
        if activation_function == tf.nn.sigmoid:
            cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(pred), reduction_indices=1))
        else:
            cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(tf.clip_by_value(pred, -1.0, 1.0)), reduction_indices=1))
            #cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred, labels=y))
    with tf.name_scope('SGD'):
        # Gradient Descent
        optimizer = optimizer_method(learning_rate).minimize(cost)
    with tf.name_scope('Accuracy'):
        # Accuracy
        acc = evaluate(pred, y)

```