**인공지능 영상처리 학기말 프로젝트**

2019451170 / 조민근

!pip install tensorflow==1.14.0

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

import numpy as np

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

X\_train, y\_train = mnist.train.images, mnist.train.labels

X\_valid, y\_valid = mnist.validation.images, mnist.validation.labels

X\_test, y\_test = mnist.test.images, mnist.test.labels

########## Parameters

epochs = 10

batch\_size = 100

########## Functions

def conv2d(input, filters, kernel\_size, strides=1, activation=tf.nn.relu, padding='SAME', name='conv'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.conv2d(input, filters=filters, kernel\_size=kernel\_size,

                           strides=strides, padding=padding, activation=activation, name=name)

    return out

def dense(input, unit, activation=tf.nn.relu, name='dense'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.dense(input, unit, activation=activation, name=name)

  return out

def max\_pool(input, k, s, name='pool'):

  out = tf.nn.max\_pool(input, ksize=[1, k, k, 1], strides=[1, s, s, 1], padding='SAME', name=name)

  return out

########## FC ##########

x = tf.placeholder(tf.float32, [None, 784])

y = tf.placeholder(tf.float32, [None, 10])

fc1 = dense(x, 60, name='fc1')

########## PROJECT 2

# fc2 =

fc\_out = dense(fc1, 10, activation=tf.nn.softmax, name='fc\_out')

cross\_entropy\_FC = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=fc\_out))

optim\_FC = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_FC)

correct\_prediction\_FC = tf.equal(tf.argmax(y, 1), tf.argmax(fc\_out, 1))

accuracy\_FC = tf.reduce\_mean(tf.cast(correct\_prediction\_FC, tf.float32))

########## CNN ##########

x\_image = tf.reshape(x, [-1, 28, 28, 1])

conv1 = conv2d(x\_image, 96, 3, 1, tf.nn.relu, 'SAME', name='conv1')

pool1 = max\_pool(conv1, 2, 2, name='pool1')

########## PROJECT 3

# conv2 =

# pool2 =

########## PROJECT 4

# conv3 =

# pool3 =

# conv4 =

# pool4 =

# conv5 =

# pool5 =

########## PROJECT 5

# short cut (pool1 out & conv5 in)

cnn\_out = tf.reshape(pool1, [batch\_size, tf.size(pool1[0])])

cnn\_out = dense(cnn\_out, 10, activation=tf.nn.softmax, name='cnn\_out')

cross\_entropy\_CNN = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=cnn\_out))

optim\_CNN = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_CNN)

correct\_prediction\_CNN = tf.equal(tf.argmax(cnn\_out,1), tf.argmax(y,1))

accuracy\_CNN = tf.reduce\_mean(tf.cast(correct\_prediction\_CNN, tf.float32))

########## SESSION ##########

init = tf.initialize\_all\_variables()

with tf.Session(config=tf.ConfigProto(allow\_soft\_placement=True, log\_device\_placement=False,

                                      gpu\_options=tf.GPUOptions(allow\_growth=True, visible\_device\_list='0'))) as sess:

  sess.run(init)

  print('Training...')

  for i in range(epochs):

    s = np.random.permutation(len(X\_train))

    X\_train = X\_train[s]

    y\_train = y\_train[s]

    accCNN\_total, accFC\_total = 0, 0

    for offset in range(0, len(X\_train), batch\_size):

      end = offset + batch\_size

      batch\_x, batch\_y = X\_train[offset:end], y\_train[offset:end]

      \_, accFC = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

      \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

      accFC\_total += accFC

      accCNN\_total += accCNN

    accFC\_total /= (len(X\_train) / batch\_size)

    accCNN\_total /= (len(X\_train) / batch\_size)

    print("[TRAIN] epoch: " + str(i) + ", FC Accuracy= " + "{:.4f}".format(accFC\_total) +

          ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))

  print("Optimization Finished!")

  for offset in range(0, len(X\_test), batch\_size):

    end = offset + batch\_size

    batch\_x, batch\_y = X\_test[offset:end], y\_test[offset:end]

    \_, acc = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

    \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

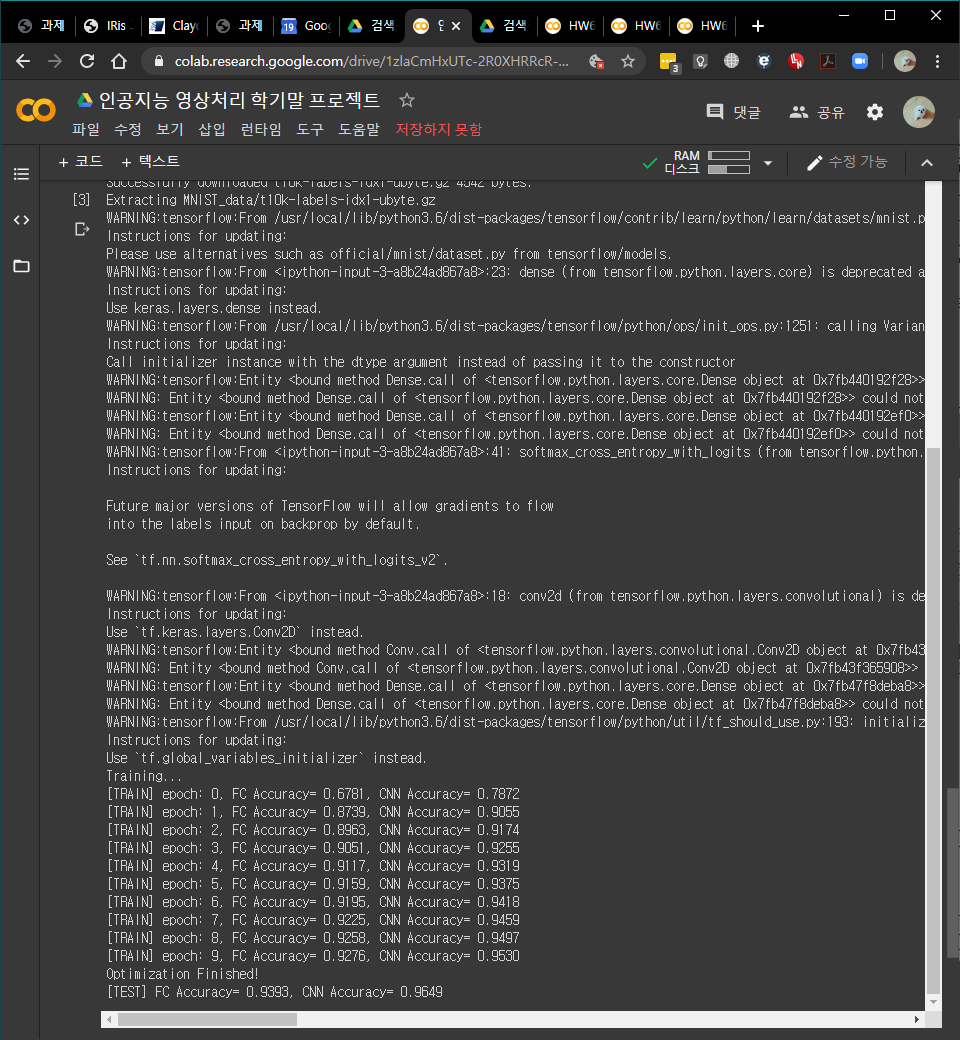
    accFC\_total += accFC

    accCNN\_total += accCNN

  accFC\_total /= (len(X\_test) / batch\_size)

  accCNN\_total /= (len(X\_test) / batch\_size)

  print("[TEST] FC Accuracy= " + "{:.4f}".format(accFC\_total) + ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))



2.

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

import numpy as np

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

X\_train, y\_train = mnist.train.images, mnist.train.labels

X\_valid, y\_valid = mnist.validation.images, mnist.validation.labels

X\_test, y\_test = mnist.test.images, mnist.test.labels

########## Parameters

epochs = 10

batch\_size = 100

########## Functions

def conv2d(input, filters, kernel\_size, strides=1, activation=tf.nn.relu, padding='SAME', name='conv'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.conv2d(input, filters=filters, kernel\_size=kernel\_size,

                           strides=strides, padding=padding, activation=activation, name=name)

    return out

def dense(input, unit, activation=tf.nn.relu, name='dense'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.dense(input, unit, activation=activation, name=name)

  return out

def max\_pool(input, k, s, name='pool'):

  out = tf.nn.max\_pool(input, ksize=[1, k, k, 1], strides=[1, s, s, 1], padding='SAME', name=name)

  return out

########## FC ##########

x = tf.placeholder(tf.float32, [None, 784])

y = tf.placeholder(tf.float32, [None, 10])

fc1 = dense(x, 60, name='fc1')

########## PROJECT 2

fc2 = dense(fc1, 30, name='fc2')

fc\_out = dense(fc2, 10, activation=tf.nn.softmax, name='fc\_out')

cross\_entropy\_FC = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=fc\_out))

optim\_FC = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_FC)

correct\_prediction\_FC = tf.equal(tf.argmax(y, 1), tf.argmax(fc\_out, 1))

accuracy\_FC = tf.reduce\_mean(tf.cast(correct\_prediction\_FC, tf.float32))

########## CNN ##########

x\_image = tf.reshape(x, [-1, 28, 28, 1])

conv1 = conv2d(x\_image, 96, 3, 1, tf.nn.relu, 'SAME', name='conv1')

pool1 = max\_pool(conv1, 2, 2, name='pool1')

########## PROJECT 3

# conv2 =

# pool2 =

########## PROJECT 4

# conv3 =

# pool3 =

# conv4 =

# pool4 =

# conv5 =

# pool5 =

########## PROJECT 5

# short cut (pool1 out & conv5 in)

cnn\_out = tf.reshape(pool1, [batch\_size, tf.size(pool1[0])])

cnn\_out = dense(cnn\_out, 10, activation=tf.nn.softmax, name='cnn\_out')

cross\_entropy\_CNN = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=cnn\_out))

optim\_CNN = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_CNN)

correct\_prediction\_CNN = tf.equal(tf.argmax(cnn\_out,1), tf.argmax(y,1))

accuracy\_CNN = tf.reduce\_mean(tf.cast(correct\_prediction\_CNN, tf.float32))

########## SESSION ##########

init = tf.initialize\_all\_variables()

with tf.Session(config=tf.ConfigProto(allow\_soft\_placement=True, log\_device\_placement=False,

                                      gpu\_options=tf.GPUOptions(allow\_growth=True, visible\_device\_list='0'))) as sess:

  sess.run(init)

  print('Training...')

  for i in range(epochs):

    s = np.random.permutation(len(X\_train))

    X\_train = X\_train[s]

    y\_train = y\_train[s]

    accCNN\_total, accFC\_total = 0, 0

    for offset in range(0, len(X\_train), batch\_size):

      end = offset + batch\_size

      batch\_x, batch\_y = X\_train[offset:end], y\_train[offset:end]

      \_, accFC = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

      \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

      accFC\_total += accFC

      accCNN\_total += accCNN

    accFC\_total /= (len(X\_train) / batch\_size)

    accCNN\_total /= (len(X\_train) / batch\_size)

    print("[TRAIN] epoch: " + str(i) + ", FC Accuracy= " + "{:.4f}".format(accFC\_total) +

          ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))

  print("Optimization Finished!")

  for offset in range(0, len(X\_test), batch\_size):

    end = offset + batch\_size

    batch\_x, batch\_y = X\_test[offset:end], y\_test[offset:end]

    \_, acc = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

    \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

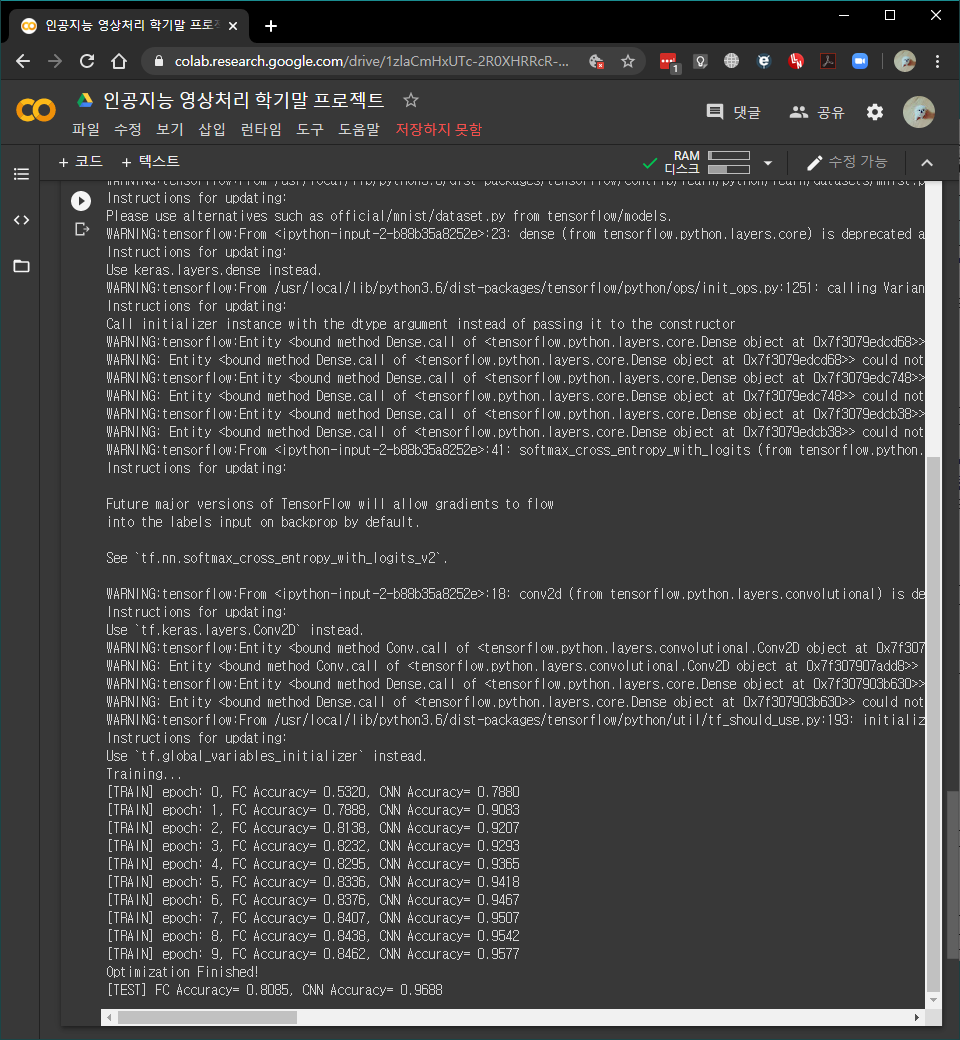
    accFC\_total += accFC

    accCNN\_total += accCNN

  accFC\_total /= (len(X\_test) / batch\_size)

  accCNN\_total /= (len(X\_test) / batch\_size)

  print("[TEST] FC Accuracy= " + "{:.4f}".format(accFC\_total) + ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))



3.

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

import numpy as np

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

X\_train, y\_train = mnist.train.images, mnist.train.labels

X\_valid, y\_valid = mnist.validation.images, mnist.validation.labels

X\_test, y\_test = mnist.test.images, mnist.test.labels

########## Parameters

epochs = 10

batch\_size = 100

########## Functions

def conv2d(input, filters, kernel\_size, strides=1, activation=tf.nn.relu, padding='SAME', name='conv'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.conv2d(input, filters=filters, kernel\_size=kernel\_size,

                           strides=strides, padding=padding, activation=activation, name=name)

    return out

def dense(input, unit, activation=tf.nn.relu, name='dense'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.dense(input, unit, activation=activation, name=name)

  return out

def max\_pool(input, k, s, name='pool'):

  out = tf.nn.max\_pool(input, ksize=[1, k, k, 1], strides=[1, s, s, 1], padding='SAME', name=name)

  return out

########## FC ##########

x = tf.placeholder(tf.float32, [None, 784])

y = tf.placeholder(tf.float32, [None, 10])

fc1 = dense(x, 60, name='fc1')

########## PROJECT 2

fc2 = dense(fc1, 30, name='fc2')

fc\_out = dense(fc2, 10, activation=tf.nn.softmax, name='fc\_out')

cross\_entropy\_FC = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=fc\_out))

optim\_FC = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_FC)

correct\_prediction\_FC = tf.equal(tf.argmax(y, 1), tf.argmax(fc\_out, 1))

accuracy\_FC = tf.reduce\_mean(tf.cast(correct\_prediction\_FC, tf.float32))

########## CNN ##########

x\_image = tf.reshape(x, [-1, 28, 28, 1])

conv1 = conv2d(x\_image, 96, 3, 1, tf.nn.relu, 'SAME', name='conv1')

pool1 = max\_pool(conv1, 2, 2, name='pool1')

########## PROJECT 3

 conv2 = conv2d(pool1, 96, 7, 1, tf.nn.relu, 'SAME', name='conv2')

 pool2 = max\_pool(conv2, 8, 2, name='pool2')

########## PROJECT 4

# conv3 =

# pool3 =

# conv4 =

# pool4 =

# conv5 =

# pool5 =

########## PROJECT 5

# short cut (pool1 out & conv5 in)

cnn\_out = tf.reshape(pool1, [batch\_size, tf.size(pool1[0])])

cnn\_out = dense(cnn\_out, 10, activation=tf.nn.softmax, name='cnn\_out')

cross\_entropy\_CNN = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=cnn\_out))

optim\_CNN = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_CNN)

correct\_prediction\_CNN = tf.equal(tf.argmax(cnn\_out,1), tf.argmax(y,1))

accuracy\_CNN = tf.reduce\_mean(tf.cast(correct\_prediction\_CNN, tf.float32))

########## SESSION ##########

init = tf.initialize\_all\_variables()

with tf.Session(config=tf.ConfigProto(allow\_soft\_placement=True, log\_device\_placement=False,

                                      gpu\_options=tf.GPUOptions(allow\_growth=True, visible\_device\_list='0'))) as sess:

  sess.run(init)

  print('Training...')

  for i in range(epochs):

    s = np.random.permutation(len(X\_train))

    X\_train = X\_train[s]

    y\_train = y\_train[s]

    accCNN\_total, accFC\_total = 0, 0

    for offset in range(0, len(X\_train), batch\_size):

      end = offset + batch\_size

      batch\_x, batch\_y = X\_train[offset:end], y\_train[offset:end]

      \_, accFC = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

      \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

      accFC\_total += accFC

      accCNN\_total += accCNN

    accFC\_total /= (len(X\_train) / batch\_size)

    accCNN\_total /= (len(X\_train) / batch\_size)

    print("[TRAIN] epoch: " + str(i) + ", FC Accuracy= " + "{:.4f}".format(accFC\_total) +

          ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))

  print("Optimization Finished!")

  for offset in range(0, len(X\_test), batch\_size):

    end = offset + batch\_size

    batch\_x, batch\_y = X\_test[offset:end], y\_test[offset:end]

    \_, acc = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

    \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

    accFC\_total += accFC

    accCNN\_total += accCNN

  accFC\_total /= (len(X\_test) / batch\_size)

  accCNN\_total /= (len(X\_test) / batch\_size)

  print("[TEST] FC Accuracy= " + "{:.4f}".format(accFC\_total) + ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))

3.

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

import numpy as np

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

X\_train, y\_train = mnist.train.images, mnist.train.labels

X\_valid, y\_valid = mnist.validation.images, mnist.validation.labels

X\_test, y\_test = mnist.test.images, mnist.test.labels

########## Parameters

epochs = 10

batch\_size = 100

########## Functions

def conv2d(input, filters, kernel\_size, strides=1, activation=tf.nn.relu, padding='SAME', name='conv'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.conv2d(input, filters=filters, kernel\_size=kernel\_size,

                           strides=strides, padding=padding, activation=activation, name=name)

    return out

def dense(input, unit, activation=tf.nn.relu, name='dense'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.dense(input, unit, activation=activation, name=name)

  return out

def max\_pool(input, k, s, name='pool'):

  out = tf.nn.max\_pool(input, ksize=[1, k, k, 1], strides=[1, s, s, 1], padding='SAME', name=name)

  return out

########## FC ##########

x = tf.placeholder(tf.float32, [None, 784])

y = tf.placeholder(tf.float32, [None, 10])

fc1 = dense(x, 60, name='fc1')

########## PROJECT 2

fc2 = dense(fc1, 30, name='fc2')

fc\_out = dense(fc2, 10, activation=tf.nn.softmax, name='fc\_out')

cross\_entropy\_FC = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=fc\_out))

optim\_FC = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_FC)

correct\_prediction\_FC = tf.equal(tf.argmax(y, 1), tf.argmax(fc\_out, 1))

accuracy\_FC = tf.reduce\_mean(tf.cast(correct\_prediction\_FC, tf.float32))

########## CNN ##########

x\_image = tf.reshape(x, [-1, 28, 28, 1])

conv1 = conv2d(x\_image, 96, 3, 1, tf.nn.relu, 'SAME', name='conv1')

pool1 = max\_pool(conv1, 2, 2, name='pool1')

########## PROJECT 3

conv2 = conv2d(pool1, 96, 7, 1, tf.nn.relu, 'SAME', name='conv2')

pool2 = max\_pool(conv2, 8, 2, name='pool2')

########## PROJECT 4

# conv3 =

# pool3 =

# conv4 =

# pool4 =

# conv5 =

# pool5 =

########## PROJECT 5

# short cut (pool1 out & conv5 in)

cnn\_out = tf.reshape(pool2, [batch\_size, tf.size(pool2[0])])

cnn\_out = dense(cnn\_out, 10, activation=tf.nn.softmax, name='cnn\_out')

cross\_entropy\_CNN = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=cnn\_out))

optim\_CNN = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_CNN)

correct\_prediction\_CNN = tf.equal(tf.argmax(cnn\_out,1), tf.argmax(y,1))

accuracy\_CNN = tf.reduce\_mean(tf.cast(correct\_prediction\_CNN, tf.float32))

########## SESSION ##########

init = tf.initialize\_all\_variables()

with tf.Session(config=tf.ConfigProto(allow\_soft\_placement=True, log\_device\_placement=False,

                                      gpu\_options=tf.GPUOptions(allow\_growth=True, visible\_device\_list='0'))) as sess:

  sess.run(init)

  print('Training...')

  for i in range(epochs):

    s = np.random.permutation(len(X\_train))

    X\_train = X\_train[s]

    y\_train = y\_train[s]

    accCNN\_total, accFC\_total = 0, 0

    for offset in range(0, len(X\_train), batch\_size):

      end = offset + batch\_size

      batch\_x, batch\_y = X\_train[offset:end], y\_train[offset:end]

      \_, accFC = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

      \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

      accFC\_total += accFC

      accCNN\_total += accCNN

    accFC\_total /= (len(X\_train) / batch\_size)

    accCNN\_total /= (len(X\_train) / batch\_size)

    print("[TRAIN] epoch: " + str(i) + ", FC Accuracy= " + "{:.4f}".format(accFC\_total) +

          ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))

  print("Optimization Finished!")

  for offset in range(0, len(X\_test), batch\_size):

    end = offset + batch\_size

    batch\_x, batch\_y = X\_test[offset:end], y\_test[offset:end]

    \_, acc = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

    \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

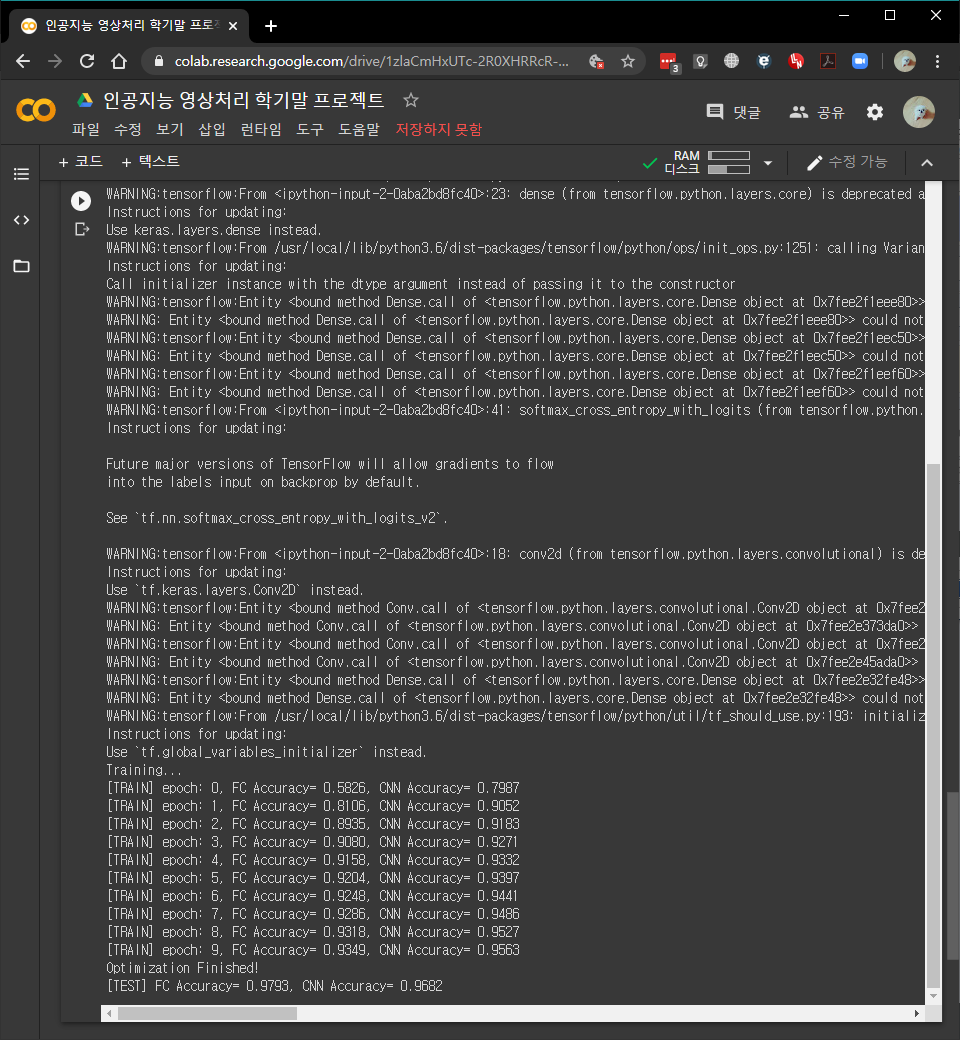
    accFC\_total += accFC

    accCNN\_total += accCNN

  accFC\_total /= (len(X\_test) / batch\_size)

  accCNN\_total /= (len(X\_test) / batch\_size)

  print("[TEST] FC Accuracy= " + "{:.4f}".format(accFC\_total) + ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))



4.

import tensorflow as tf

from tensorflow.examples.tutorials.mnist import input\_data

import numpy as np

mnist = input\_data.read\_data\_sets("MNIST\_data/", one\_hot=True)

X\_train, y\_train = mnist.train.images, mnist.train.labels

X\_valid, y\_valid = mnist.validation.images, mnist.validation.labels

X\_test, y\_test = mnist.test.images, mnist.test.labels

########## Parameters

epochs = 10

batch\_size = 100

########## Functions

def conv2d(input, filters, kernel\_size, strides=1, activation=tf.nn.relu, padding='SAME', name='conv'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.conv2d(input, filters=filters, kernel\_size=kernel\_size,

                           strides=strides, padding=padding, activation=activation, name=name)

    return out

def dense(input, unit, activation=tf.nn.relu, name='dense'):

  with tf.variable\_scope(name, reuse=False):

    out = tf.layers.dense(input, unit, activation=activation, name=name)

  return out

def max\_pool(input, k, s, name='pool'):

  out = tf.nn.max\_pool(input, ksize=[1, k, k, 1], strides=[1, s, s, 1], padding='SAME', name=name)

  return out

'''

## 이게 1 resnet

def res\_block(x, num, filter\_num, kernel\_size, strides, name='res\_block'):

  with tf.variable\_scope(name, reuse=False):

    x\_ = x

    for i in range(num ‐ 1):

      x\_ = conv2d(x, filter\_num, kernel\_size, strides, tf.nn.relu,'SAME', name=name+'%s' % i)

    x\_ = conv2d(x\_, filter\_num, kernel\_size, strides, None, 'SAME’,name=name+'%s' % num)

    out = tf.nn.relu(x + x\_)

  return out

  '''

########## FC ##########

x = tf.placeholder(tf.float32, [None, 784])

y = tf.placeholder(tf.float32, [None, 10])

fc1 = dense(x, 60, name='fc1')

########## PROJECT 2

fc2 = dense(fc1, 30, name='fc2')

fc\_out = dense(fc2, 10, activation=tf.nn.softmax, name='fc\_out')

cross\_entropy\_FC = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=fc\_out))

optim\_FC = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_FC)

correct\_prediction\_FC = tf.equal(tf.argmax(y, 1), tf.argmax(fc\_out, 1))

accuracy\_FC = tf.reduce\_mean(tf.cast(correct\_prediction\_FC, tf.float32))

########## CNN ##########

x\_image = tf.reshape(x, [-1, 28, 28, 1])

conv1 = conv2d(x\_image, 96, 3, 1, tf.nn.relu, 'SAME', name='conv1')

pool1 = max\_pool(conv1, 2, 2, name='pool1')

########## PROJECT 3

#conv2 = conv2d(pool1, 96, 7, 1, tf.nn.relu, 'SAME', name='conv2')

#pool2 = max\_pool(conv2, 8, 2, name='pool2')

########## PROJECT 4

conv3 = conv2d(pool1, 96, 3, 1, tf.nn.relu, 'SAME', name='conv3')

#pool3 = max\_pool(conv3, 2, 2, name='pool3')

conv4 = conv2d(conv3, 96, 3, 1, tf.nn.relu, 'SAME', name='conv4')

#pool4 = max\_pool(conv4, 2, 2, name='pool4')

########## PROJECT 5

# short cut (pool1 out & conv5 in)

out = conv4+pool1

conv5 = conv2d(out , 96, 3, 1, tf.nn.relu, 'SAME', name='conv5')

pool5 = max\_pool(conv5, 2, 2, name='pool5')

##바꿔주기

cnn\_out = tf.reshape(pool5, [batch\_size, tf.size(pool5[0])])

cnn\_out = dense(cnn\_out, 10, activation=tf.nn.softmax, name='cnn\_out')

cross\_entropy\_CNN = tf.reduce\_mean(tf.nn.softmax\_cross\_entropy\_with\_logits(labels=y, logits=cnn\_out))

optim\_CNN = tf.train.AdamOptimizer(1e-4).minimize(cross\_entropy\_CNN)

correct\_prediction\_CNN = tf.equal(tf.argmax(cnn\_out,1), tf.argmax(y,1))

accuracy\_CNN = tf.reduce\_mean(tf.cast(correct\_prediction\_CNN, tf.float32))

########## SESSION ##########

init = tf.initialize\_all\_variables()

with tf.Session(config=tf.ConfigProto(allow\_soft\_placement=True, log\_device\_placement=False,

                                      gpu\_options=tf.GPUOptions(allow\_growth=True, visible\_device\_list='0'))) as sess:

  sess.run(init)

  print('Training...')

  for i in range(epochs):

    s = np.random.permutation(len(X\_train))

    X\_train = X\_train[s]

    y\_train = y\_train[s]

    accCNN\_total, accFC\_total = 0, 0

    for offset in range(0, len(X\_train), batch\_size):

      end = offset + batch\_size

      batch\_x, batch\_y = X\_train[offset:end], y\_train[offset:end]

      \_, accFC = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

      \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

      accFC\_total += accFC

      accCNN\_total += accCNN

    accFC\_total /= (len(X\_train) / batch\_size)

    accCNN\_total /= (len(X\_train) / batch\_size)

    print("[TRAIN] epoch: " + str(i) + ", FC Accuracy= " + "{:.4f}".format(accFC\_total) +

          ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))

  print("Optimization Finished!")

  for offset in range(0, len(X\_test), batch\_size):

    end = offset + batch\_size

    batch\_x, batch\_y = X\_test[offset:end], y\_test[offset:end]

    \_, acc = sess.run([optim\_FC, accuracy\_FC], feed\_dict={x: batch\_x, y: batch\_y})

    \_, accCNN = sess.run([optim\_CNN, accuracy\_CNN], feed\_dict={x: batch\_x, y: batch\_y})

    accFC\_total += accFC

    accCNN\_total += accCNN

  accFC\_total /= (len(X\_test) / batch\_size)

  accCNN\_total /= (len(X\_test) / batch\_size)

  print("[TEST] FC Accuracy= " + "{:.4f}".format(accFC\_total) + ", CNN Accuracy= " + "{:.4f}".format(accCNN\_total))