

30분만에 개발하는 이미지 인식 프로그램

- **99.5%** 정확도 달성하기



KYLIus-method



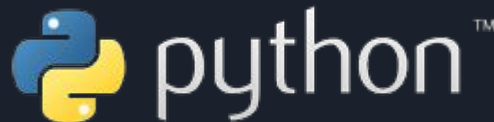
Team KYLius-method

<https://github.com/leedaegon/KYLius-method/wiki>

Member: 김승혁, 곽상욱, 유수원, 이대곤

We do:

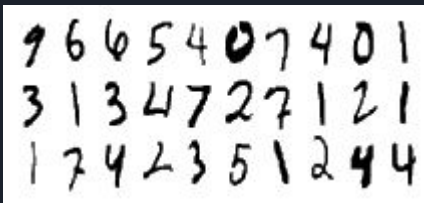
- Research Neural-Network
- Deep learning modeling
- Hyper-parameter tuning
- Automate learn



오늘 이야기할 내용들

1. Data set summary
2. Model
3. Handwriting
4. Ensemble
5. 결론

kaggle

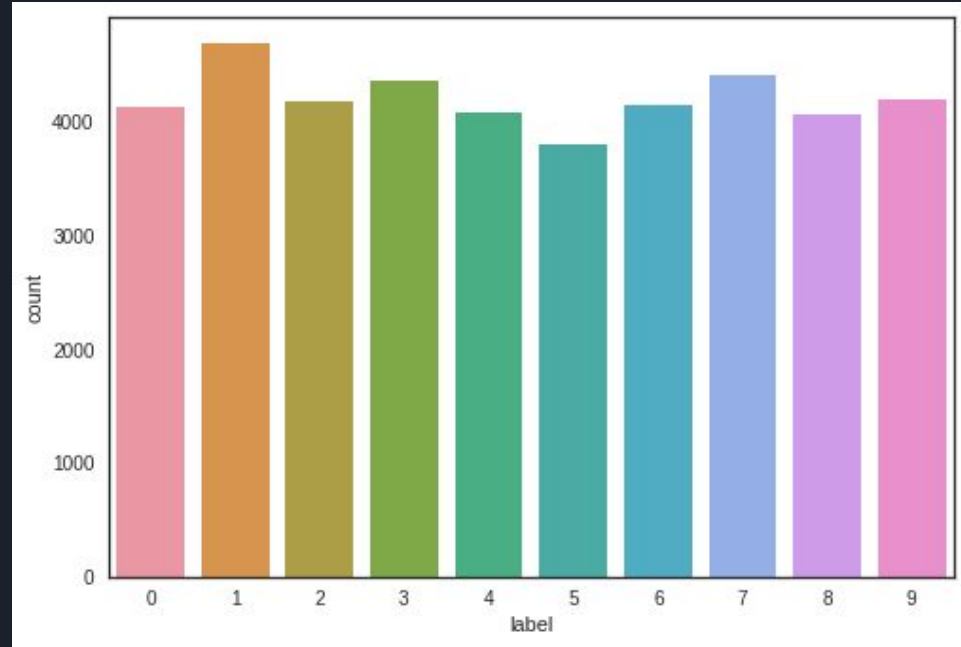
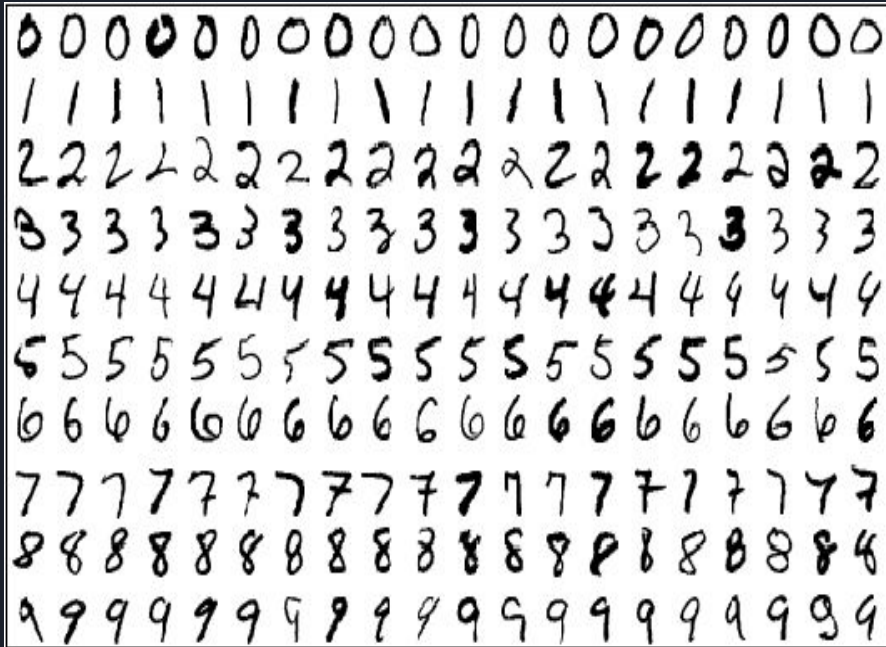




1. Data set



Kaggle MNIST Dataset

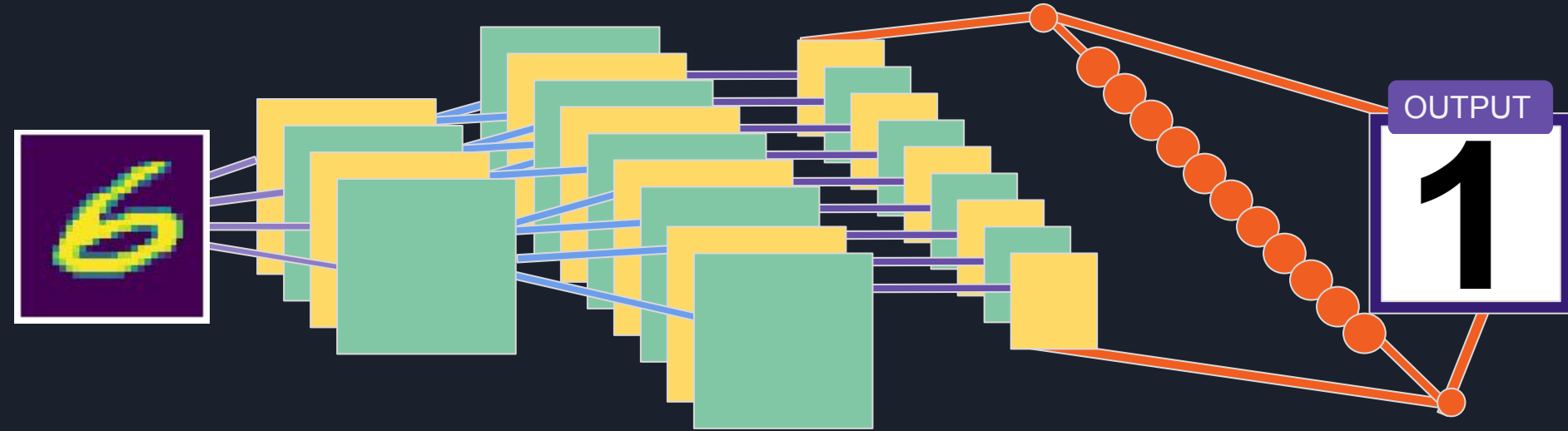




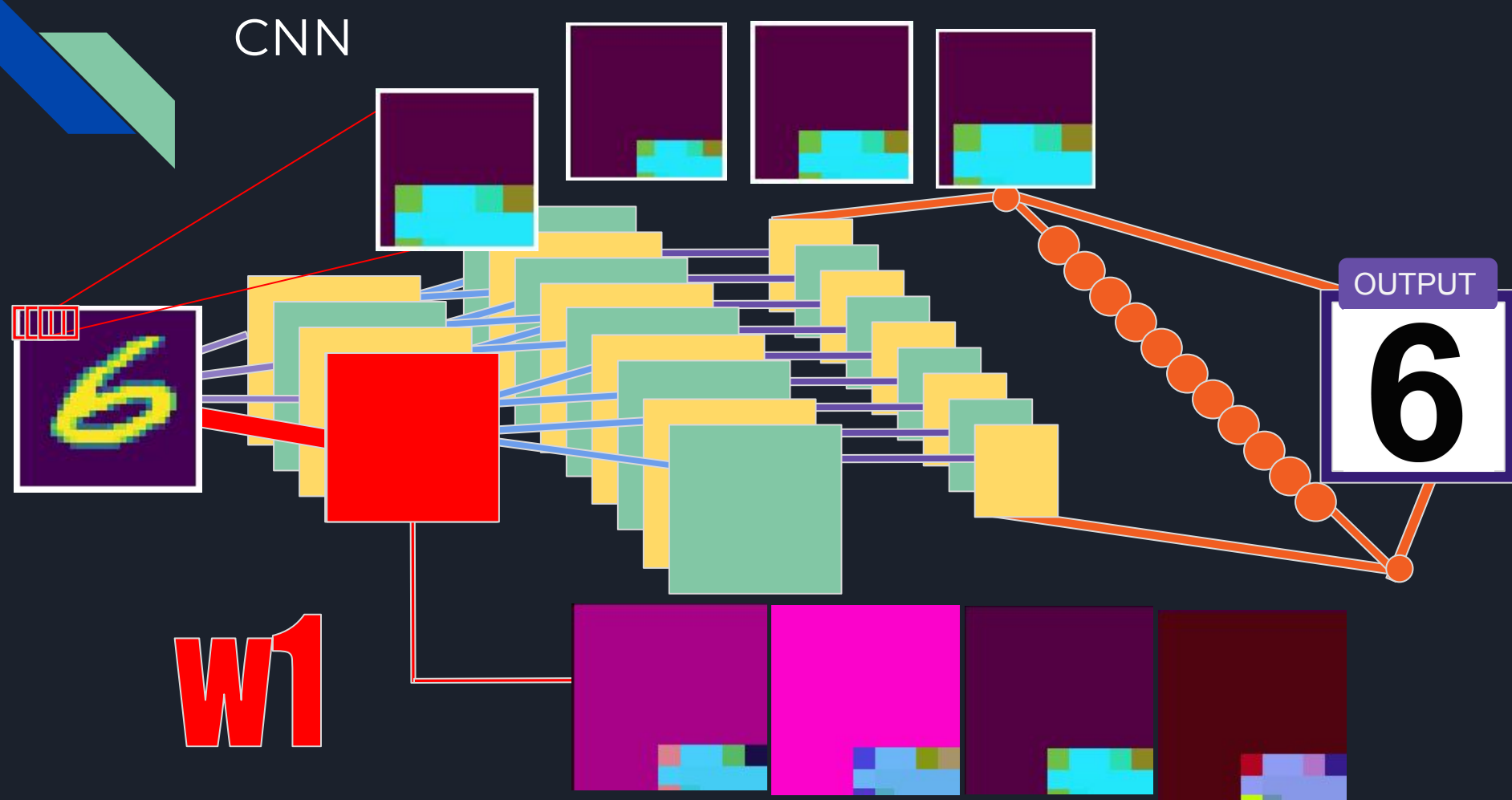
2. Model



CNN



CNN




```
import tensorflow as tf
import numpy as np
tf.reset_default_graph() #그래프 초기화
tf.set_random_seed(777)
import pandas as pd
```

```
#train = pd.read_csv('c:/python/train.csv')
train = pd.read_csv('C:/Users/Nak/train.csv')
```

```
#훈련 세트, validation 세트 나누기(문제은행에서 연습문제 나누기)
from sklearn.model_selection import train_test_split
train_set, validate_set = train_test_split(train, test_size = 0.3)
trainData = train_set.values[:,1:]
validateData = validate_set.values[:,1:]
trainLabel=train_set.values[:,0]
validateLabel=validate_set.values[:,0]
```

```
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1]) #img 28x28x1
(black/white)
Y = tf.placeholder(tf.int32, [None, 1])
Y_onehot=tf.reshape(tf.one_hot(Y, 10), [-1, 10])
p_keep_conv = tf.placeholder(tf.float32)
p_keep_hidden = tf.placeholder(tf.float32)
```

```
# hyper parameters(gpu로 밑에 붙이는 learning_rate)
learning_rate = 0.00008
training_epochs = 300
batch_size = 100
steps_for_validate = 5
keep_prob = tf.placeholder(tf.float32)
```

```
# L1 ImgIn shape=(?, 28, 28, 1)
#W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
W1 = tf.get_variable("W1", shape=[3, 3, 1, 32], initializer=tf.contrib.layers.xavier_initializer())
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
#L1 = tf.nn.relu(L1)
#L1 = tf.nn.elu(L1)
```

```
L1 = tf.nn.leaky_relu(L1, 0.1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME') # l1 shape=(?, 14, 14, 32)
L1 = tf.nn.dropout(L1, p_keep_conv)
```

```
# L2 ImgIn shape=(?, 14, 14, 10)
#W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
W2 = tf.get_variable("W2", shape=[3, 3, 32, 64], initializer=tf.contrib.layers.xavier_initializer())
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
#L2 = tf.nn.relu(L2)
#L2 = tf.nn.elu(L2)
L2 = tf.nn.leaky_relu(L2, 0.1)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME') # l2 shape=(?, 7, 7, 64)
L2 = tf.nn.dropout(L2, p_keep_conv)
```

```
# L3 ImgIn shape=(?, 7, 7, 128)
#W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
W3 = tf.get_variable("W3", shape=[3, 3, 64, 128], initializer=tf.contrib.layers.xavier_initializer())
L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
#L3 = tf.nn.relu(L3)
#L3 = tf.nn.elu(L3)
L3 = tf.nn.leaky_relu(L3, 0.1)
L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME') # l3 shape=(?, 4, 4, 128)
L3 = tf.nn.dropout(L3, p_keep_conv)
L3_flat = tf.reshape(L3, shape=[-1, 128 * 4 * 4]) # reshape to (?, 2048)
# Final FC 4x4x128 inputs -> 10 outputs
#W4 = tf.Variable(tf.random_normal([128 * 4 * 4, 625],
stddev=0.01))
W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625], initializer=tf.contrib.layers.xavier_initializer())
#L4 = tf.nn.relu(tf.matmul(L3_flat, W4))
#L4 = tf.nn.elu(tf.matmul(L3_flat, W4))
L4 = tf.nn.leaky_relu(tf.matmul(L3_flat, W4), 0.1)
L4 = tf.nn.dropout(L4, p_keep_hidden)
W_o = tf.get_variable("W_o",
shape=[625, 10], initializer=tf.contrib.layers.xavier_initializer())
```

```
b = tf.Variable(tf.random_normal([10]))
logits = tf.matmul(L4, W_o) + b
```

```
# define cost/loss & optimizer
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels=Y_onehot))
#optimizer = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)
optimizer =
tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost) # 아담버전
predict_op = tf.argmax(logits, 1)
```

```
# initialize
sess = tf.Session()
sess.run(tf.global_variables_initializer())
```

```
# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(len(trainData) / batch_size)
    for i in range(total_batch):
        batch_xs = trainData[i*batch_size:(i+1)*batch_size]
        batch_ys = trainLabel[i*batch_size:(i+1)*batch_size].reshape(-1, 1)
        feed_dict = {X: batch_xs, Y: batch_ys, p_keep_conv: .7, p_keep_hidden: .5}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
        print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
        if epoch % steps_for_validate == steps_for_validate-1:
            correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y_onehot, 1))
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
            print('Accuracy:', sess.run(accuracy, feed_dict={X: validateData, Y: validateLabel.reshape(-1, 1), p_keep_conv: 1, p_keep_hidden: 1}))
    print('Finished!')
```

맨 위에서부터 차근차근

```
In [6]:  
  
import tensorflow as tf  
import numpy as np  
tf.reset_default_graph()    #그래프 초기화  
tf.set_random_seed(777)  
import pandas as pd  
  
#train = pd.read_csv('c:/python/train.csv')  
train = pd.read_csv('C:/Users/Nak/train.csv')
```

맨 위에서부터 차근차근

In [6]:

```
import tensorflow as tf
import numpy as np
tf.reset_default_graph() #그래프 초기화
tf.set_random_seed(777)
import pandas as pd

#train = pd.read_csv('c:/python/train.csv')
train = pd.read_csv('C:/Users/Nak/train.csv')
```

맨 위에서부터 차근차근

```
#훈련세트, validation세트 나누기(문제은행에서 연습문제 나누기)
from sklearn.model_selection import train_test_split
train_set, validate_set = train_test_split(train, test_size = 0.3)
trainData = train_set.values[:,1:]
validateData = validate_set.values[:,1:]
trainLabel=train_set.values[:,0]
validateLabel=validate_set.values[:,0]

X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])      # img 28x28x1 (black/white)
Y = tf.placeholder(tf.int32, [None, 1])
Y_onehot=tf.reshape(tf.one_hot(Y, 10), [-1, 10])
p_keep_conv = tf.placeholder(tf.float32)
p_keep_hidden = tf.placeholder(tf.float32)
```

맨 위에서부터 차근차근

#훈련세트, validation세트 나누기(문제은행에서 연습문제 나누기)

```
from sklearn.model_selection import train_test_split
```

```
train_set, validate_set = train_test_split(train, test_size = 0.3)
```

```
trainData = train_set.values[:,1:]
```

```
validateData = validate_set.values[:,1:]
```

```
trainLabel=train_set.values[:,0]
```

```
validateLabel=validate_set.values[:,0]
```

train data = 연습문제(모의고사, 문제은행)
test data = 실전문제(수능, 기사시험)

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

```
X_img = tf.reshape(X, [-1, 28, 28, 1])
```

img 28x28x1 (black/white)

```
Y = tf.placeholder(tf.int32, [None, 1])
```

```
Y_onehot=tf.reshape(tf.one_hot(Y, 10), [-1, 10])
```

```
p_keep_conv = tf.placeholder(tf.float32)
```

```
p_keep_hidden = tf.placeholder(tf.float32)
```

6	7	2	0	2	9	4	7	8	4
6	0	8	4	1	0	4	4	5	3
3	3	6	2	5	5	5	8	3	0
3	2	2	1	6	1	5	3	3	5
9	4	3	7	1	5	5	4	3	7

Hyper parameter

```
# hyper parameters(gpu로 믿어볼이는 learning_rate)
learning_rate = 0.00008
training_epochs = 300
batch_size = 100
steps_for_validate = 5
keep_prob = tf.placeholder(tf.float32)

# L1 imgIn shape=(?, 28, 28, 1)
#W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
W1 = tf.get_variable("W1", shape=[3, 3, 1, 32], initializer=tf.contrib.layers.xavier_initializer())
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
```

Hyper parameter

```
# hyper parameters(gpu로 밑어붙이는 learning_rate)
```

```
learning_rate = 0.00008
```

```
training_epochs = 300
```

```
batch_size = 100
```

```
steps_for_validate = 5
```

```
keep_prob = tf.placeholder(tf.float32)
```

```
# L1 imgIn shape=(?, 28, 28, 1)
```

```
#W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
```

```
W1 = tf.get_variable("W1", shape=[3, 3, 1, 32], initializer=tf.contrib.layers.xavier_initializer())
```

```
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
```

CNN

```
# L1 imgIn shape=(?, 28, 28, 1)
#W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
W1 = tf.get_variable("W1", shape=[3, 3, 1, 32], initializer=tf.contrib.layers.xavier_initializer())
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
#L1 = tf.nn.relu(L1)
#L1 = tf.nn.elu(L1)
L1 = tf.nn.leaky_relu(L1, 0.1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME') # l1 shape=(?, 14, 14, 32)
L1 = tf.nn.dropout(L1, p_keep_conv)

# L2 imgIn shape=(?, 14, 14, 10)
#W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
W2 = tf.get_variable("W2", shape=[3, 3, 32, 64], initializer=tf.contrib.layers.xavier_initializer())
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
```


CNN(2단계 -> 3단계)

```
#L2 = tf.nn.relu(L2)
#L2 = tf.nn.elu(L2)
L2 = tf.nn.leaky_relu(L2, 0.1)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME') # l2 shape=(?, 7, 7, 64)
L2 = tf.nn.dropout(L2, p_keep_conv)

# L3 imgIn shape=(?, 7, 7, 128)
#W3 = tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.01))
W3 = tf.get_variable("W3", shape=[3, 3, 64, 128], initializer=tf.contrib.layers.xavier_initializer())
L3 = tf.nn.conv2d(L2, W3, strides=[1, 1, 1, 1], padding='SAME')
```

CNN(3단계->4단계:마지막 출력)

```
#L3 = tf.nn.relu(L3)
#L3 = tf.nn.elu(L3)
L3 = tf.nn.leaky_relu(L3,0.1)
L3 = tf.nn.max_pool(L3, ksize=[1, 2, 2, 1],strides=[1, 2, 2, 1], padding='SAME') # L3 shape=(?, 4, 4, 128)
L3 = tf.nn.dropout(L3, p_keep_conv)
L3_flat = tf.reshape(L3, shape=[-1, 128 * 4 * 4]) # reshape to (?, 2048)

# Final FC 4x4x128 inputs -> 10 outputs
#W4 = tf.Variable(tf.random_normal([128 * 4 * 4, 625], stddev=0.01))
W4 = tf.get_variable("W4", shape=[128 * 4 * 4, 625], initializer=tf.contrib.layers.xavier_initializer())
```

CNN(4단계 출력)

```
#L4 = tf.nn.relu(tf.matmul(L3_flat, W4))  
#L4 = tf.nn.elu(tf.matmul(L3_flat, W4))  
L4 = tf.nn.leaky_relu(tf.matmul(L3_flat, W4), 0.1)  
L4 = tf.nn.dropout(L4, p_keep_hidden)  
W_o = tf.get_variable("W_o", shape=[625,10], initializer=tf.contrib.layers.xavier_initializer())  
b = tf.Variable(tf.random_normal([10]))  
logits = tf.matmul(L4, W_o) + b  
  
# define cost/loss & optimizer  
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=logits, labels= Y_onehot))
```

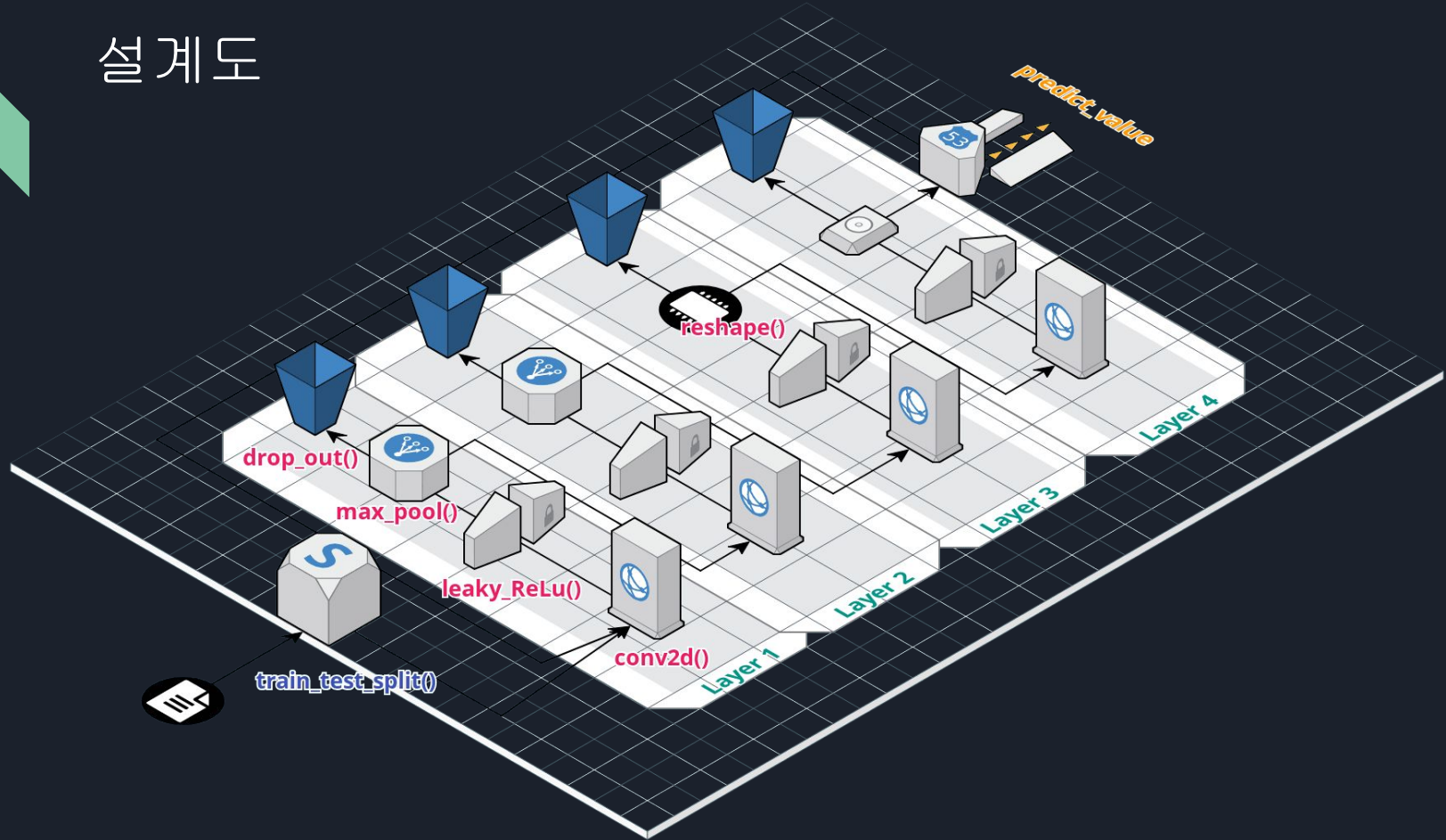
Optimizer?

```
#optimizer = tf.train.RMSPropOptimizer(0.001, 0.9).minimize(cost)  
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost) # 아담버전  
predict_op = tf.argmax(logits, 1)  
  
# initialize  
sess = tf.Session()  
sess.run(tf.global_variables_initializer())
```

돌려봅시다!

```
# train my model
print('Learning started. It takes sometime.')
for epoch in range(training_epochs):
    avg_cost = 0
    total_batch = int(len(trainData) / batch_size)
    for i in range(total_batch):
        batch_xs = trainData[i*batch_size:(i+1)*batch_size]
        batch_ys = trainLabel[i*batch_size:(i+1)*batch_size].reshape(-1, 1)
        feed_dict = {X: batch_xs, Y: batch_ys, p_keep_conv: .7, p_keep_hidden: .5}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch
    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))
    if epoch % steps_for_validate == steps_for_validate-1:
        correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y_onehot, 1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
        print('Accuracy:', sess.run(accuracy, feed_dict={
            X: validateData, Y: validateLabel.reshape(-1, 1), p_keep_conv: 1, p_keep_hidden: 1}))
print('Finished!')
```

설계도



3. HandWriting



3. HandWriting

<https://github.com/teedaegon/KYLIus-method>

실제 손글씨를 얼마나 알아볼 수 있을까

train_optimizer.py

1. 모델(trained model)의 학습결과를 저장

```
In [ ]: # 변수 선언 후
sess = tf.Session()
sess.run(tf.global_variables_initializer())
saver = tf.train.Saver()
# 트레이닝 후
save_path = saver.save(sess,
"/Users/kimseunghyuck/desktop/git/daegon/KYLIus-method/승혁/opt2/opt2")
```


3. HandWriting

<https://github.com/teedaegon/KYLIus-method>

실제 손글씨를 얼마나 알아볼 수 있을까?

img_pred.py

class img_pred

2. 저장된 모델을 로드

```
# graph 의 메타 구조를 import
saver=tf.train.import_meta_graph(opt_addr+".meta")
sess = tf.InteractiveSession()
print("Meta_Graph Imported")
# 변수값들을 로드
saver.restore(sess,
tf.train.get_checkpoint_state(opt_addr2).model_checkpoint_path)
print("Parameters Restored")
```

3. HandWriting

<https://github.com/teedaegon/KYLIus-method>

실제 손글씨를 얼마나 알아볼 수 있을까?

img_pred.py

class img_pred

2. 저장된 모델을 로드

```
# 변수들을 호출 가능한 형태로 만들  
graph=tf.get_default_graph()  
X=graph.get_tensor_by_name('X:0')  
pred=graph.get_tensor_by_name('pred:0')  
p_keep_conv=graph.get_tensor_by_name('p_keep_conv:0')  
p_keep_hidden=graph.get_tensor_by_name('p_keep_hidden:0')  
print("Variables Saved")
```

3. HandWriting

<https://github.com/leedaegon/KYLIus-method>

실제 손글씨를 얼마나 알아볼 수 있을까?

img_pred.py

class img_pred

function number

3. 손글씨 이미지를 csv 형태로 변환

```
import matplotlib.pyplot as plt
from PIL import Image
im=Image.open(arg1)
img = np.array(im.resize((28, 28), Image.ANTIALIAS).convert("L"))
data = img.reshape([1, 784])
data = 255 - data
```

3. HandWriting

<https://github.com/leedaegon/KYLIus-method>

실제 손글씨를 얼마나 알아볼 수 있을까?

img_pred.py

class img_pred

function number

4. csv 데이터로 결과 출력

```
sess.run(pred, feed_dict={X: data,  
p_keep_conv: 1.0, p_keep_hidden: 1.0})
```

잘 나오는지 한번 볼까요?
(시연)



4. Ensemble



부록 - Ensemble

```
Epoch: 0281 cost = 0.004309580
Epoch: 0282 cost = 0.003279742
Epoch: 0283 cost = 0.003534236
Epoch: 0284 cost = 0.004005449
Epoch: 0285 cost = 0.005078060
Accuracy: 0.993651
Epoch: 0286 cost = 0.003409503
Epoch: 0287 cost = 0.002734759
Epoch: 0288 cost = 0.002700005
Epoch: 0289 cost = 0.002742919
Epoch: 0290 cost = 0.003816546
Accuracy: 0.994206
Epoch: 0291 cost = 0.003566688
Epoch: 0292 cost = 0.003698584
Epoch: 0293 cost = 0.005130288
Epoch: 0294 cost = 0.002316941
Epoch: 0295 cost = 0.003286639
Accuracy: 0.994524
Epoch: 0296 cost = 0.003097230
Epoch: 0297 cost = 0.003816188
Epoch: 0298 cost = 0.004105472
Epoch: 0299 cost = 0.003956134
Epoch: 0300 cost = 0.004233572
Accuracy: 0.994841
Finished!
```

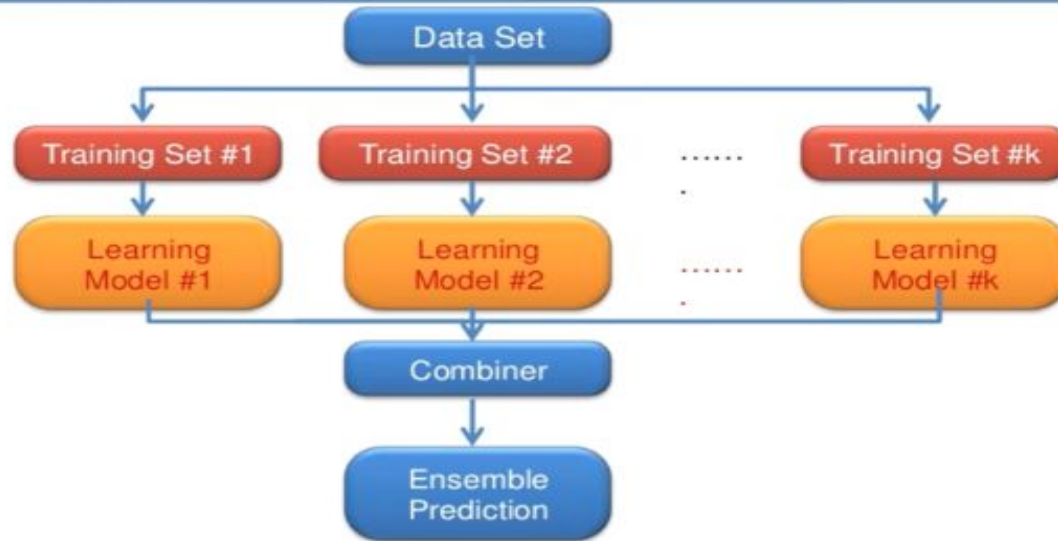
```
Python console
Console 1/A X
Accuracy: 0.994303
Epoch: 0250 cost = 0.003892478
Epoch: 0257 cost = 0.004079010
Epoch: 0258 cost = 0.004339220
Epoch: 0259 cost = 0.003897957
Epoch: 0260 cost = 0.006007245
Accuracy: 0.993571
Epoch: 0261 cost = 0.004118198
Epoch: 0262 cost = 0.004173137
Epoch: 0263 cost = 0.004512424
Epoch: 0264 cost = 0.005312191
Epoch: 0265 cost = 0.003564031
Accuracy: 0.994603
Epoch: 0266 cost = 0.004549257
Epoch: 0267 cost = 0.003209255
Epoch: 0268 cost = 0.003961666
Epoch: 0269 cost = 0.002745414
Epoch: 0270 cost = 0.003396001
Accuracy: 0.994048
Epoch: 0271 cost = 0.004785818
Epoch: 0272 cost = 0.002469373
Epoch: 0273 cost = 0.003543403
Epoch: 0274 cost = 0.004545365
Epoch: 0275 cost = 0.003999185
Accuracy: 0.993889
Epoch: 0276 cost = 0.004005012
Epoch: 0277 cost = 0.004646602
Epoch: 0278 cost = 0.004044836
Epoch: 0279 cost = 0.003262609
Epoch: 0280 cost = 0.003785761
Accuracy: 0.994206
Epoch: 0281 cost = 0.004309580
Epoch: 0282 cost = 0.003279742
Epoch: 0283 cost = 0.003534236
Epoch: 0284 cost = 0.004005449
Epoch: 0285 cost = 0.005078060
Accuracy: 0.993651
Epoch: 0286 cost = 0.003409503
Epoch: 0287 cost = 0.002734759
Epoch: 0288 cost = 0.002700005
Epoch: 0289 cost = 0.002742919
Epoch: 0290 cost = 0.003816546
Accuracy: 0.994206
Epoch: 0291 cost = 0.003566688
Epoch: 0292 cost = 0.003698584
Epoch: 0293 cost = 0.005130288
Epoch: 0294 cost = 0.002316941
Epoch: 0295 cost = 0.003286639
Accuracy: 0.994524
Epoch: 0296 cost = 0.003097230
Epoch: 0297 cost = 0.003816188
Epoch: 0298 cost = 0.004105472
Epoch: 0299 cost = 0.003956134
Epoch: 0300 cost = 0.004233572
Accuracy: 0.994841
Finished!
```

In [35]:

Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 26 Column: 24 Memory: 32 %

부록 - Ensemble

What is Ensemble?



부록 - Ensemble

이전 코드

```
# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.get_variable("W1", shape=[3, 3, 1,
32], initializer=tf.contrib.layers.xavier_initializer())
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
#L1 = tf.nn.relu(L1)
L1 = tf.nn.leaky_relu(L1, 0.1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME') # l1 shape=(?, 14, 14, 32)
L1 = tf.nn.dropout(L1, p_keep_conv)

# L2 ImgIn shape=(?, 14, 14, 10)
W2 = tf.get_variable("W2", shape=[3, 3, 32,
64], initializer=tf.contrib.layers.xavier_initializer())
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
#L1 = tf.nn.relu(L1)
L2 = tf.nn.leaky_relu(L2, 0.1)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1],
padding='SAME') # l2 shape=(?, 7, 7, 64)
```

...

...

...



Class code

class Model:

```
def __init__(self, sess, name):
    self.sess = sess
    self.name = name
    self._build_net()
```

def _build_net(self): //여기에 모든 레이어를 넣습니다.

```
with tf.variable_scope(self.name):
    self.X = tf.placeholder(tf.float32, [None, 784])
    X_img = tf.reshape(self.X, [-1, 28, 28, 1])
    self.Y = tf.placeholder(tf.float32, [None, 10])
```

```
# L1 ImgIn shape=(?, 28, 28, 1)
W1 = tf.Variable(tf.random_normal([3, 3, 1, 32], stddev=0.01))
...
```

```
def predict(self, x_test, keep_prop=1.0):
    return self.sess.run(self.logits,
        feed_dict={self.X: x_test, self.keep_prob: keep_prop})
```


부록 - Ensemble

이전 코드 코스트 출력

Epoch: 0001 Cost = 8.49241807
Epoch: 0002 Cost = 2.00249927
Epoch: 0003 Cost = 1.05088007
Epoch: 0004 Cost = 0.66475868
Epoch: 0005 Cost = 0.59724291
Epoch: 0006 Cost = 0.49182685
Epoch: 0007 Cost = 0.37189295
Epoch: 0008 Cost = 0.36213403
Epoch: 0009 Cost = 0.27789087
Epoch: 0010 Cost = 0.29027444
Accuracy: 0.9341



ensemble 코드 코스트 출력
모델 5개

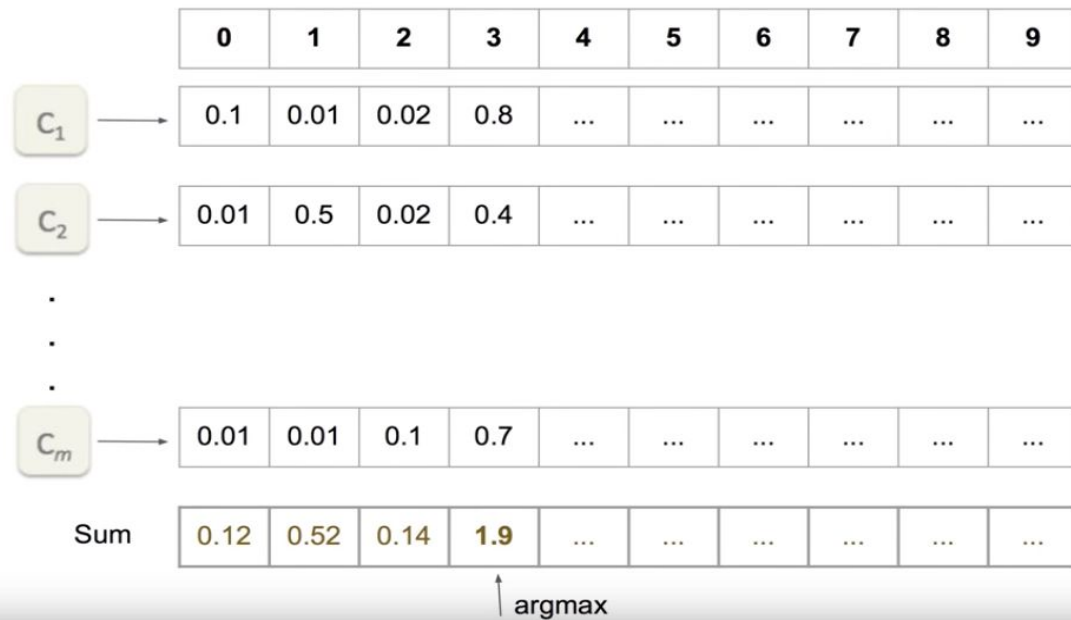
Epoch: 0001 Cost = [8.49241807 10.87485297 10.52818911 7.12766369 7.58521522]
Epoch: 0002 Cost = [1.66913546 2.0372428 2.00249927 1.47342024 1.64121001]
Epoch: 0003 Cost = [1.05441222 1.29278263 1.25971556 0.9297188 1.05088007]
Epoch: 0004 Cost = [0.77209901 0.96274401 0.89902738 0.66475868 0.77866484]
Epoch: 0005 Cost = [0.59724291 0.75306771 0.70474437 0.53224141 0.58912528]
Epoch: 0006 Cost = [0.49759393 0.6228686 0.5576628 0.43235828 0.49182685]
Epoch: 0007 Cost = [0.42757112 0.52155101 0.46283212 0.37189295 0.41670503]
Epoch: 0008 Cost = [0.36458481 0.45986743 0.40814632 0.32401 0.36213403]
Epoch: 0009 Cost = [0.32700458 0.38760419 0.35965106 0.27789087 0.31364639]
Epoch: 0010 Cost = [0.29027444 0.3512799 0.31902656 0.2587427 0.27308713]
Accuracy: [0.93215 0.924485 0.941115 0.9315481 0.945112]

Epoch: 0011 Cost = 0.31032155
Epoch: 0012 Cost = 0.27787288
Epoch: 0013 Cost = 0.20681616
Epoch: 0014 Cost = 0.1766747
Epoch: 0015 Cost = 0.18023166
Epoch: 0016 Cost = 0.16275373
Epoch: 0017 Cost = 0.1439912
Epoch: 0018 Cost = 0.16074292
Epoch: 0019 Cost = 0.129944
Epoch: 0020 Cost = 0.11760519

Epoch: 0011 Cost = [0.25922384 0.31032155 0.28886831 0.23514648 0.26196254]
Epoch: 0012 Cost = [0.23628862 0.27787288 0.26135749 0.20697285 0.22838042]
Epoch: 0013 Cost = [0.21110017 0.25410617 0.23468112 0.18942689 0.20681616]
Epoch: 0014 Cost = [0.19857974 0.22948367 0.21368499 0.1766747 0.19431507]
Epoch: 0015 Cost = [0.18023166 0.21097862 0.20243013 0.1701023 0.17910747]
Epoch: 0016 Cost = [0.16610203 0.19096776 0.1876079 0.15377879 0.16275373]
Epoch: 0017 Cost = [0.15477573 0.18003848 0.16935547 0.1439912 0.15484631]
Epoch: 0018 Cost = [0.1430705 0.16939201 0.16074292 0.13027417 0.14399578]
Epoch: 0019 Cost = [0.1379196 0.15855069 0.15091801 0.1279273 0.129944]
Epoch: 0020 Cost = [0.12886279 0.14519003 0.13762941 0.11760519 0.12391298]

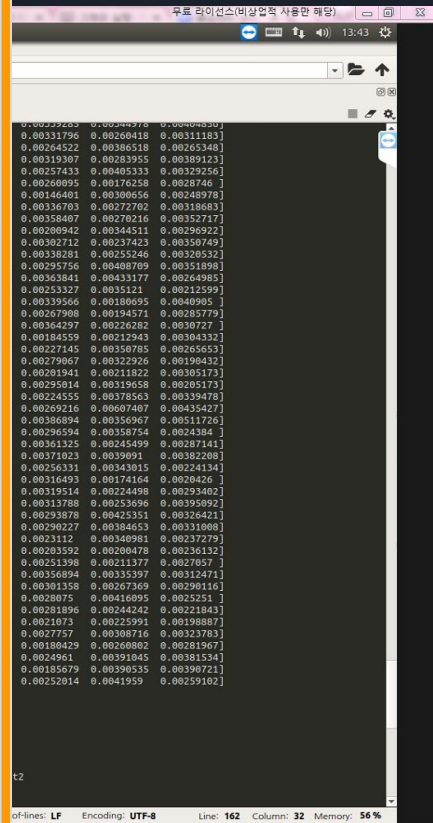
부록 - Ensemble

Ensemble prediction



부록 - Ensemble

```
Epoch: 0375 Cost = [ 0.00337258 0.00291041 0.00295014 0.00319658 0.00205173 ]
Epoch: 0376 Cost = [ 0.00225759 0.00334605 0.00224555 0.00378563 0.00339478 ]
Epoch: 0377 Cost = [ 0.00245181 0.00256908 0.00269216 0.00607407 0.00435427 ]
Epoch: 0378 Cost = [ 0.00268352 0.00258506 0.00386894 0.00356967 0.00511726 ]
Epoch: 0379 Cost = [ 0.0032349 0.00357136 0.00296594 0.00358754 0.0024384 ]
Epoch: 0380 Cost = [ 0.00264649 0.00341906 0.00361325 0.00245499 0.00287141 ]
Epoch: 0381 Cost = [ 0.00202985 0.00264583 0.00371023 0.0039091 0.00382208 ]
Epoch: 0382 Cost = [ 0.00460582 0.00275744 0.00256331 0.00343015 0.00224134 ]
Epoch: 0383 Cost = [ 0.00268927 0.00252092 0.00316493 0.00174164 0.0020426 ]
Epoch: 0384 Cost = [ 0.00217946 0.00187081 0.00319514 0.00224498 0.00293402 ]
Epoch: 0385 Cost = [ 0.00331092 0.00194117 0.00313788 0.00253696 0.00395092 ]
Epoch: 0386 Cost = [ 0.00235657 0.00280586 0.00293878 0.00425351 0.00326421 ]
Epoch: 0387 Cost = [ 0.00228023 0.00277434 0.00290227 0.00384653 0.00331008 ]
Epoch: 0388 Cost = [ 0.00136978 0.00184256 0.0023112 0.00340981 0.00237279 ]
Epoch: 0389 Cost = [ 0.00249313 0.00380039 0.00203592 0.00200478 0.00236132 ]
Epoch: 0390 Cost = [ 0.00382729 0.00223138 0.00251398 0.00211377 0.0027057 ]
Epoch: 0391 Cost = [ 0.00238345 0.00315367 0.00356894 0.00335397 0.00312471 ]
Epoch: 0392 Cost = [ 0.00259086 0.00359748 0.00301358 0.00267369 0.00290116 ]
Epoch: 0393 Cost = [ 0.00316236 0.00390281 0.0028075 0.00416095 0.0025251 ]
Epoch: 0394 Cost = [ 0.00355389 0.003331 0.00281896 0.00244242 0.00221843 ]
Epoch: 0395 Cost = [ 0.00353289 0.00343076 0.0021073 0.00225991 0.00198887 ]
Epoch: 0396 Cost = [ 0.00282651 0.0030055 0.0027757 0.00308716 0.00323783 ]
Epoch: 0397 Cost = [ 0.002623 0.00305262 0.00180429 0.00260802 0.00281967 ]
Epoch: 0398 Cost = [ 0.00245636 0.00267282 0.0024961 0.00391045 0.00381534 ]
Epoch: 0399 Cost = [ 0.00225432 0.00339072 0.00185679 0.00390535 0.00390721 ]
Epoch: 0400 Cost = [ 0.00162053 0.00268204 0.00252014 0.0041959 0.00259102 ]
Training Finished
0 Accuracy: 0.994048
1 Accuracy: 0.993571
2 Accuracy: 0.993889
3 Accuracy: 0.994524
4 Accuracy: 0.994365
Ensemble accuracy: 0.995079
Model saved to /home/itwill03/다운로드/opt2/opt2
```



5. 결론!



감사합니다

