08 May 2017

Basic of DCNN : AlexNet and VggNet

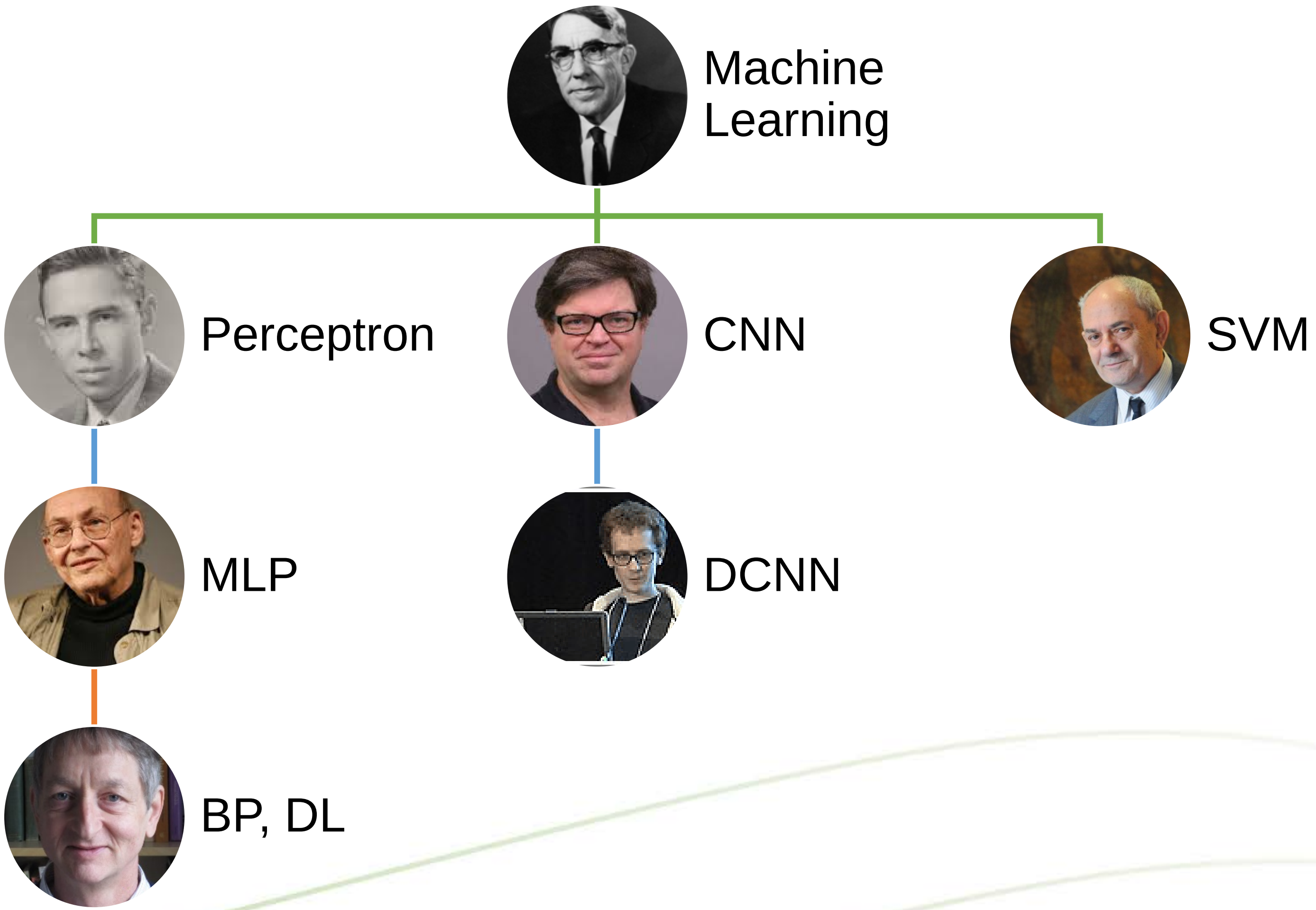
ISL lab Seminar

Han-Sol Kang

Contents

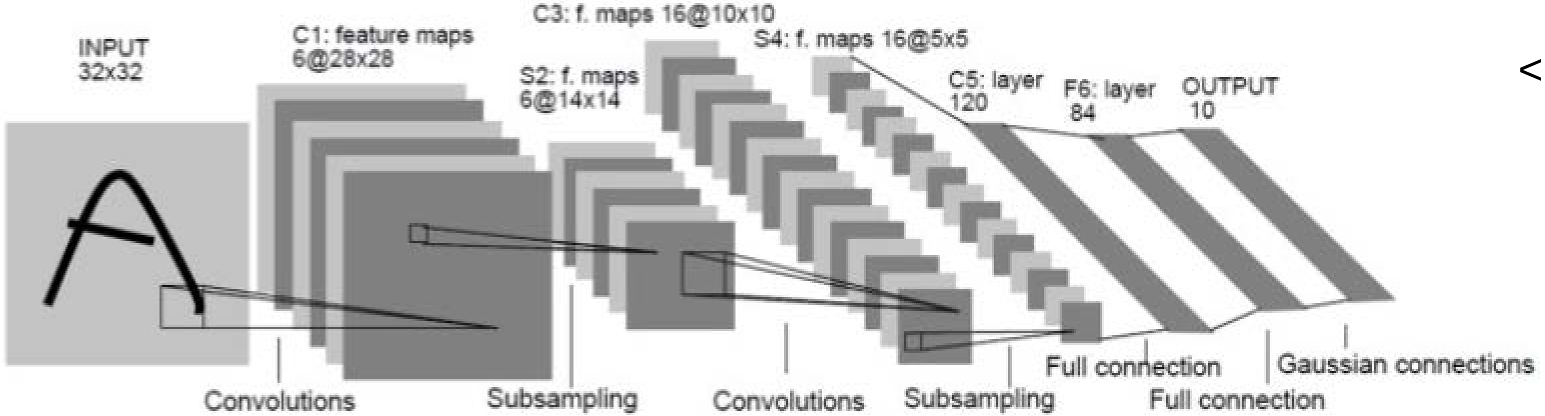
Introduction

- Machine Learning



Introduction

- CNN



<LeNet>

$$OH = \frac{H + 2P - FH}{S} + 1$$

$$OW = \frac{W + 2P - FW}{S} + 1$$

	1	2	3	0	
	0	1	2	3	
	3	0	1	2	
	2	3	0	1	

$*$

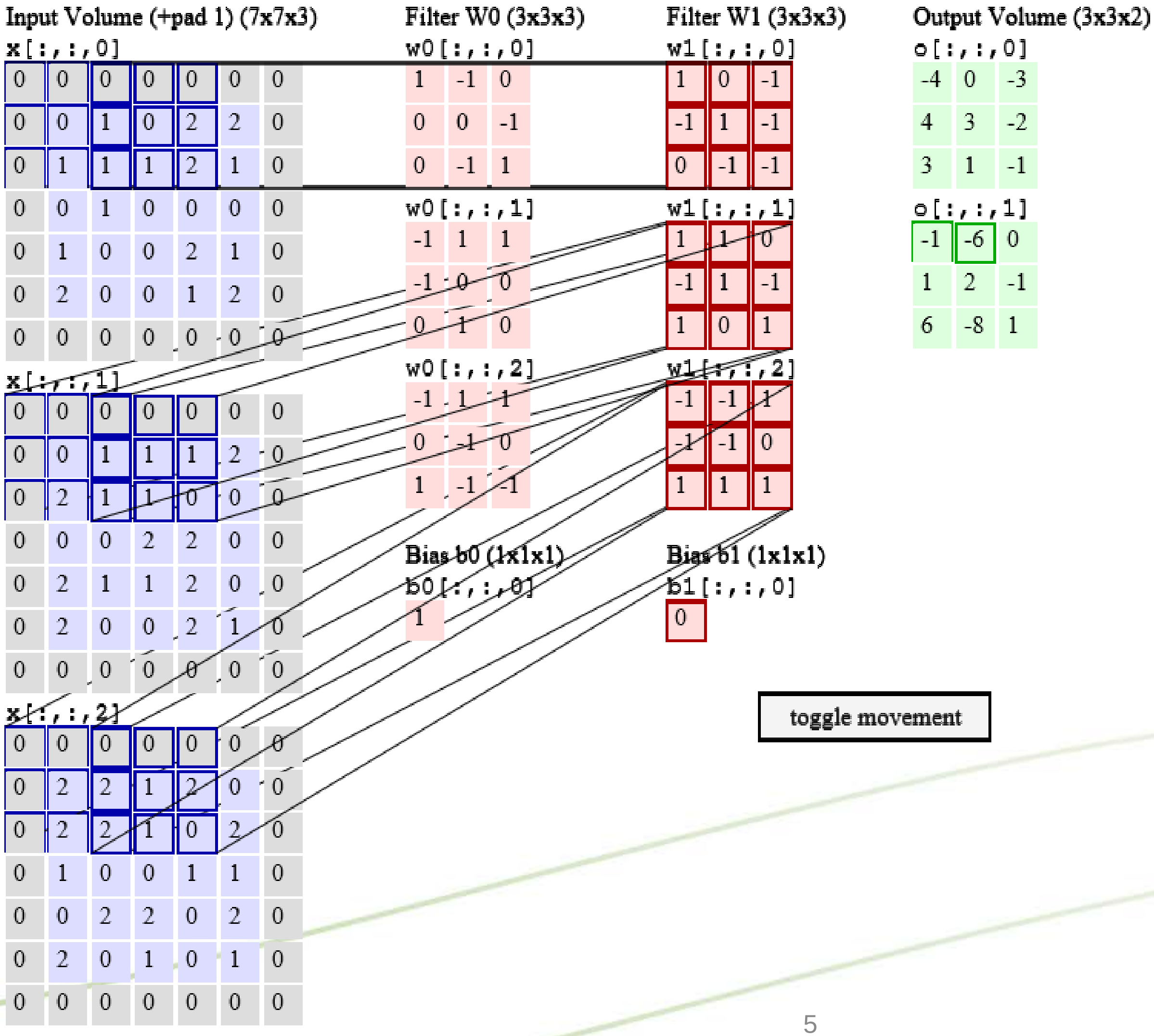
2	0	1
0	1	2
1	0	2

$=$

7	12	10	2
4	15	16	10
10	6	15	6
8	10	4	3

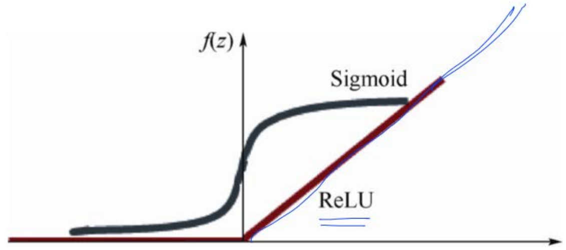
Introduction

- CNN

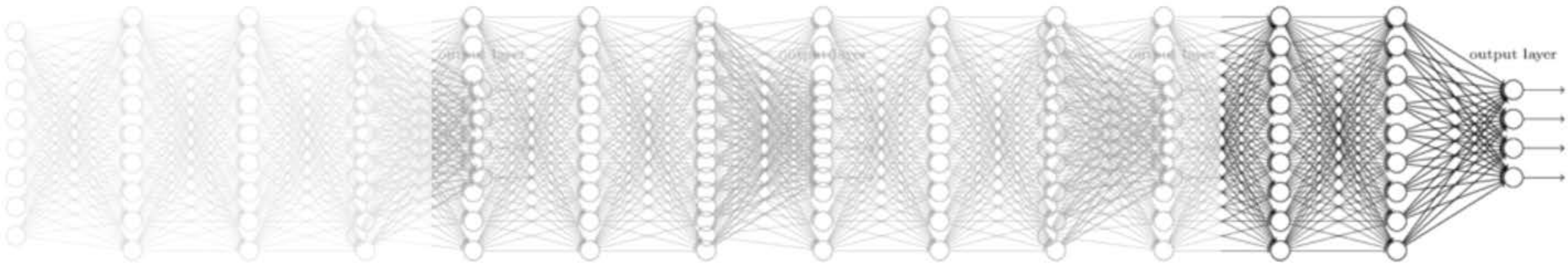


DCNN

- AlexNet



ReLU(Rectified Linear Unit)



"We used the wrong type of non-linearity"

Geoffrey Hinton

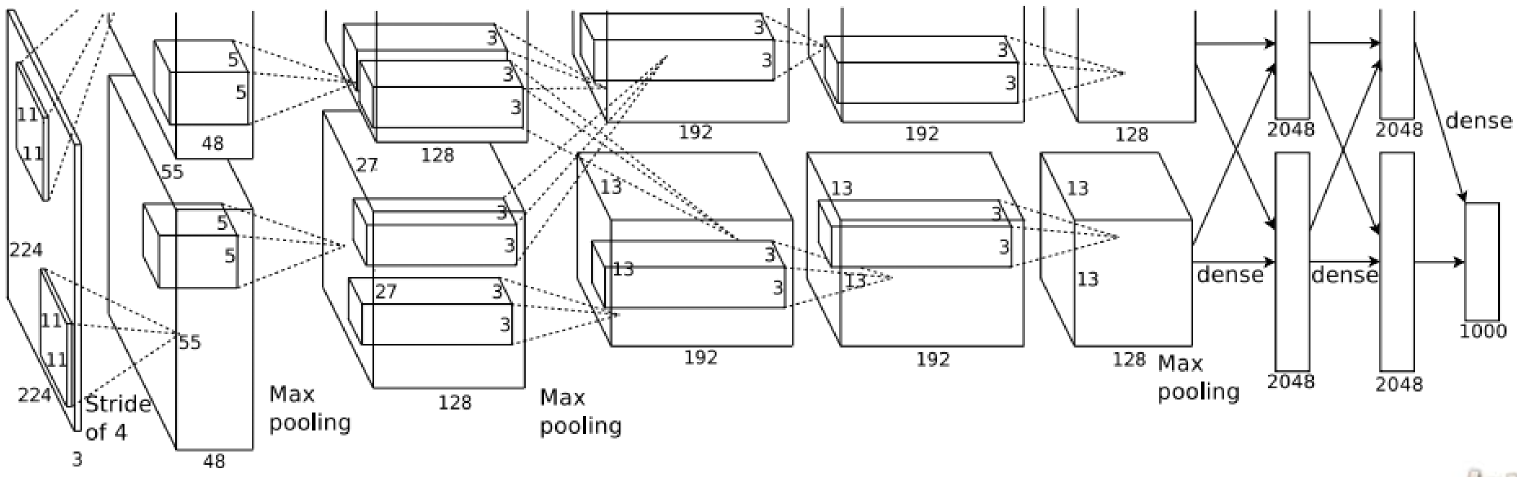
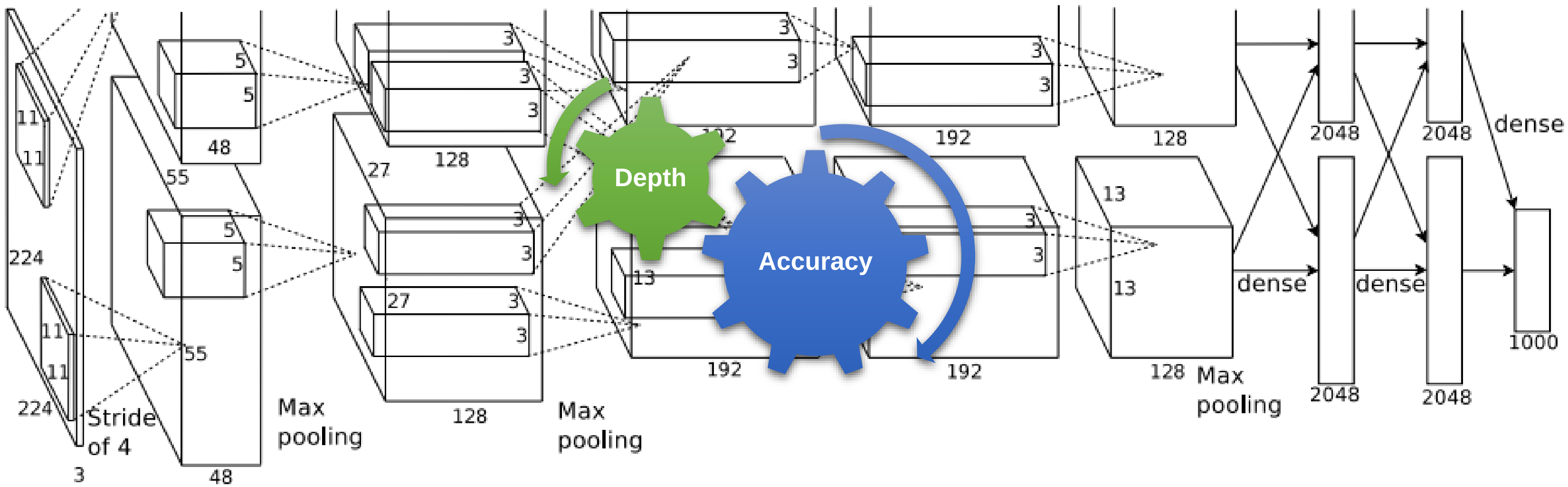


Image System Laboratory

VGG Net

- Convolutional network hit



VGG Net

- ConvNet Configuration

A	11 weight layers	Input (224x224 RGB image)	conv3-64	maxpool	conv3-128	maxpool	conv3-256 conv3-256	maxpool	conv3-512 conv3-512	maxpool	conv3-512 conv3-512	maxpool	FC-4096	FC-4096	FC-1000	soft-max
A-LRN	11 weight layers		conv3-64 **LRN**		conv3-128		conv3-256 conv3-256		conv3-512 conv3-512		conv3-512 conv3-512					
B	13 weight layers		conv3-64 **conv3-64**		conv3-128 **conv3-128**		conv3-256 conv3-256		conv3-512 conv3-512		conv3-512 conv3-512					
C	16 weight layers		conv3-64 conv3-64		conv3-128 conv3-128		conv3-256 conv3-256 **conv1-256**		conv3-512 conv3-512 **conv1-512**		conv3-512 conv3-512 **conv1-512**					
D	16 weight layers		conv3-64 conv3-64		conv3-128 conv3-128		conv3-256 conv3-256 **conv3-256**		conv3-512 conv3-512 **conv3-512**		conv3-512 conv3-512 **conv3-512**					
E	19 weight layers		conv3-64 conv3-64		conv3-128 conv3-128		conv3-256 conv3-256 conv3-256 **conv3-256**		conv3-512 conv3-512 conv3-512 **conv3-512**		conv3-512 conv3-512 conv3-512 **conv3-512**					

VGG Net

Training

- Mini-batch gradient descent with momentum (batch size : 256, momentum : 0.9)

$$W \leftarrow W - \eta \frac{\partial L}{\partial W} \qquad v \leftarrow \alpha v - \eta \frac{\partial L}{\partial W}$$

$$W \leftarrow W + v$$

- Weight decay(L_2 , $5 \cdot 10^{-4}$) & dropout (0.5) regularization

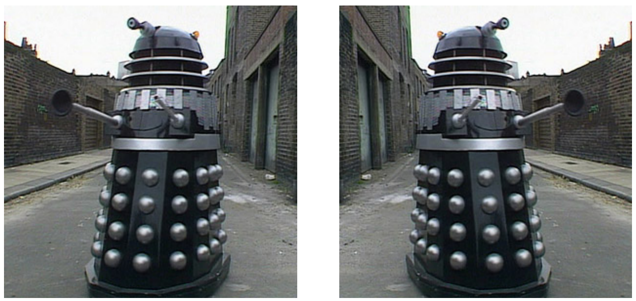
- 초기 learning rate는 10^{-2}로 설정

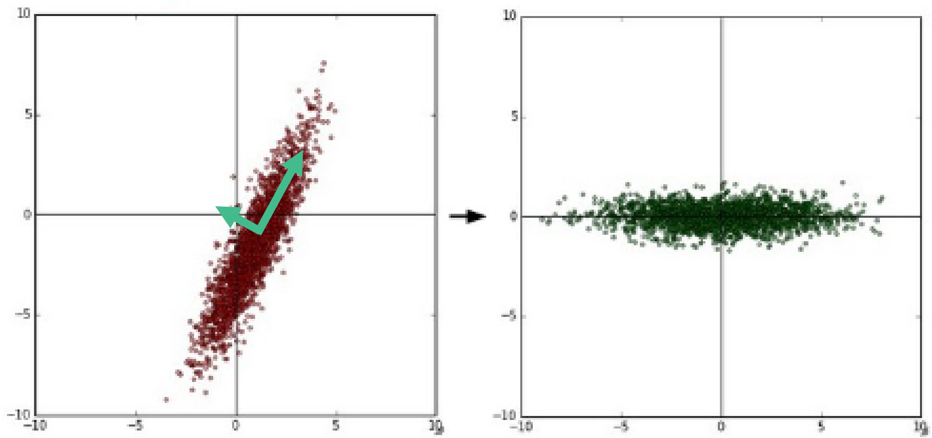
- A 네트워크 트레이닝 ⟶ 깊은 네트워크 트레이닝(초기 4개 Conv, 3개의 FC)

VGG Net

☆ Training

- Data augmentation(flip, RGB color shift, rescaling)





Single scale training : S를 고정(256 & 384)

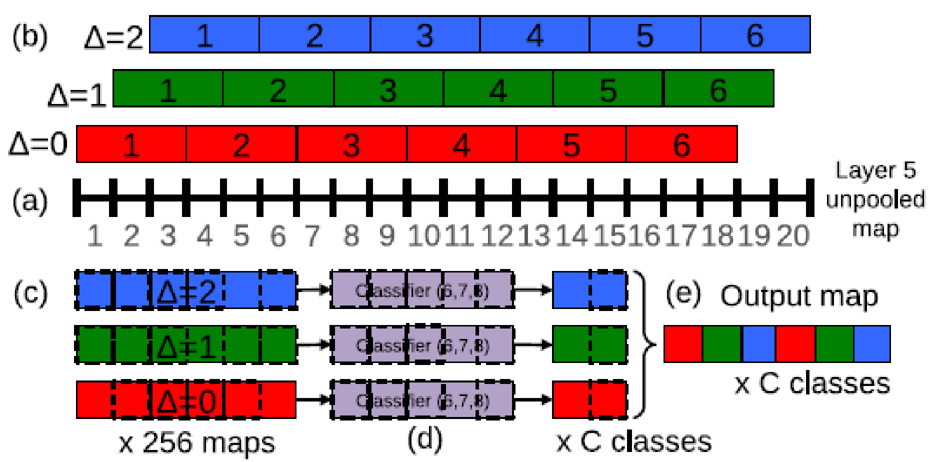
Multi-scale training : S를 일정 범위 안에서 랜덤으로 지정$[S_{min}, S_{max}]$,
$(S_{min} = 256, S_{max} = 512)$

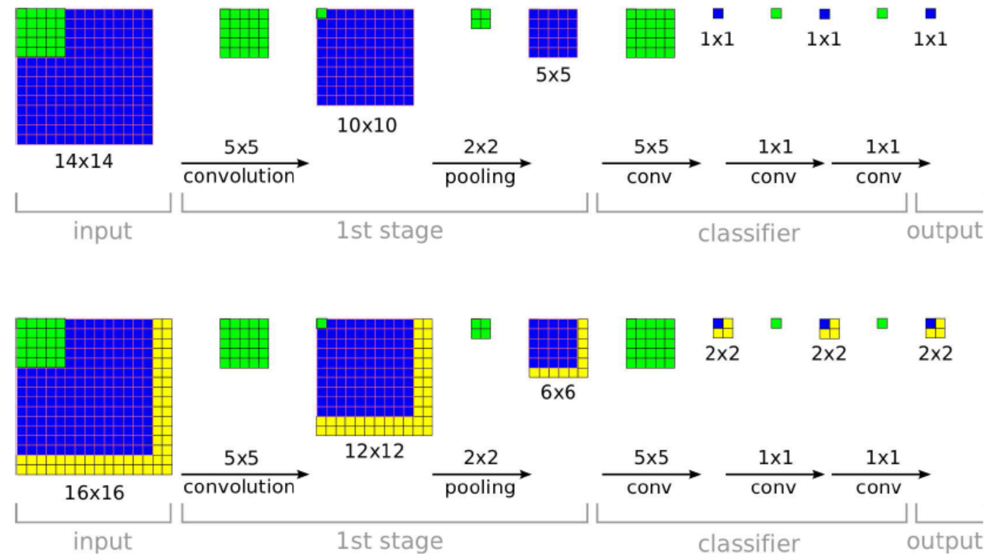
* S : 트레이닝 스케일, 입력 이미지의 비율을 유지하면서 스케일링 했을 때 가장 작은 면.

VGG Net

✩ Testing

- 테스팅 스케일 Q를 사용
- 첫번째 FC layer는 7x7 conv.layer 마지막 두 개의 FC layer는 1x1 conv.layer
- Dense evaluation을 이용. (multi-crop 방식과 같이 사용시 성능 향상)





VGG Net

- Classification experiments

ConvNet config.	smallest image side		top-1 val. error(%)	top-5 val. error(%)
	train(S)	test(Q)		
A	256	256	29.6	10.4
A-LRN	256	256	29.7	10.5
B	256	256	28.7	9.9
C	256	256	28.1	9.4
	384	384	28.1	9.3
	[256;512]	384	27.3	8.8
D	256	256	27.0	8.8
	384	384	26.8	8.7
	[256;512]	384	25.6	8.1
E	256	256	27.3	9.0
	384	384	26.9	8.7
	[256;512]	384	**25.5**	**8.0**

ConvNet config.	smallest image side		top-1 val. error(%)	top-5 val. error(%)
	train(S)	test(Q)		
B	256	224, 256, 288	28.2	9.6
C	256	224, 256, 288	27.7	9.2
	384	352, 384, 416	27.8	9.2
	[256;512]	256, 384, 512	26.3	8.2
D	256	224, 256, 288	26.6	8.6
	384	352, 384, 416	26.5	8.6
	[256;512]	256, 384, 512	**24.8**	**7.5**
E	256	224, 256, 288	26.9	8.7
	384	352, 384, 416	26.7	8.6
	[256;512]	256, 384, 512	**24.8**	**7.5**

VGG Net

- Classification experiments

ConvNet config.	Evaluation method	top-1 val. error(%)	top-5 val. error(%)
D	dense	24.8	7.5
	multi-crop	24.6	7.5
	multi-crop & dense	**24.4**	**7.2**
E	dense	24.8	7.5
	multi-crop	24.6	7.4
	multi-crop & dense	**24.4**	**7.1**

Method	top-1 val. error(%)	top-5 val. error(%)	top-5 test error(%)
VGG(2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG(1 net, multi-crop & dense eval.)	24.4	7.1	7
VGG(ILSVRC submission, 7 nets, dense eval.)	2.47	7.5	7.3
GoogLeNet(1net)		7.9	
GoogLeNet(7 nets)		**6.7**	
MSRA(11 nets)			8.1
MSRA(1 net)	27.9	9.1	9.1
Clarifai(multiple nets)			11.7
Clarifai(1 net)			12.5
ZF Net(6nets)	36.0	14.7	14.8
ZF Net(1net)	37.5	16	16.1
OverFeat(7 nets)	34.0	13.2	13.6
OverFeat(1 nets)	35.7	14.2	
AlexNet(5 nets)	38.1	16.4	16.4
AlexNet(1 net)	40.7	18.2	

VGG Net

✩ Conclusion

- 3x3의 아주 작은 컨볼루션 필터를 이용해 깊은 네트워크 구조를 평가.
- 네트워크의 깊이가 깊어질수록 분류 정확도에 도움을 주는 것을 확인.
- 전통적인 ConvNet 구조에서 깊이를 증가시켜 좋은 성능을 확인.
- VGG-16 & VGG-19 모델 공개

Implementation

```
import tensorflow as tf
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

learning_rate = 0.001
training_epochs = 15
batch_size = 100

X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])
Y = tf.placeholder(tf.float32, [None, 10])

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

L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

#두번째 레이어
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
L2 = tf.nn.conv2d(L1, W2, strides=[1, 1, 1, 1], padding='SAME')
L2 = tf.nn.relu(L2)
L2 = tf.nn.max_pool(L2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')
L2 = tf.reshape(L2, [-1, 7*7*64]) #FC 연결하기 위해 벡터로

W3 = tf.get_variable("W3", shape=[7*7*64,10], initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2,W3) + b

cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

```
sess = tf.Session()
sess.run(tf.global_variables_initializer())
print('Learning started. It takes sometime')
for epoch in range(training_epochs):
    avg_cost=0
    total_batch = int(mnist.train.num_examples / batch_size)
    for i in range(total_batch):
        batch_xs , batch_ys = mnist.train.next_batch(batch_size)
        feed_dict={X:batch_xs, Y:batch_ys}
        c,_ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost+=c/total_batch
    print('Epoch:', '%04d' % (epoch+1), 'cost =', '{:.9f}'.format(avg_cost))
print('Learning finished')

correct_prediction = tf.equal(tf.argmax(hypothesis,1), tf.argmax(Y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

print('Accuracy', sess.run(accuracy, feed_dict={X:mnist.test.images, Y:mnist.test.labels}))
```

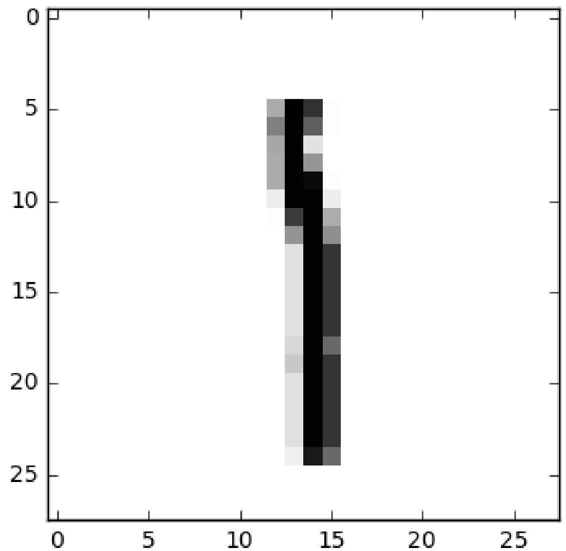
```
Epoch: 0001 cost = 0.366208560
Epoch: 0002 cost = 0.091067037
Epoch: 0003 cost = 0.067395312
Epoch: 0004 cost = 0.054241491
Epoch: 0005 cost = 0.046002268
Epoch: 0006 cost = 0.039577450
Epoch: 0007 cost = 0.034572003
Epoch: 0008 cost = 0.030414227
Epoch: 0009 cost = 0.026961391
Epoch: 0010 cost = 0.024227326
Epoch: 0011 cost = 0.020874776
Epoch: 0012 cost = 0.018590417
Epoch: 0013 cost = 0.016660221
Epoch: 0014 cost = 0.014668066
Epoch: 0015 cost = 0.012948724
Learning finished
Accuracy 0.9884
```

Implementation

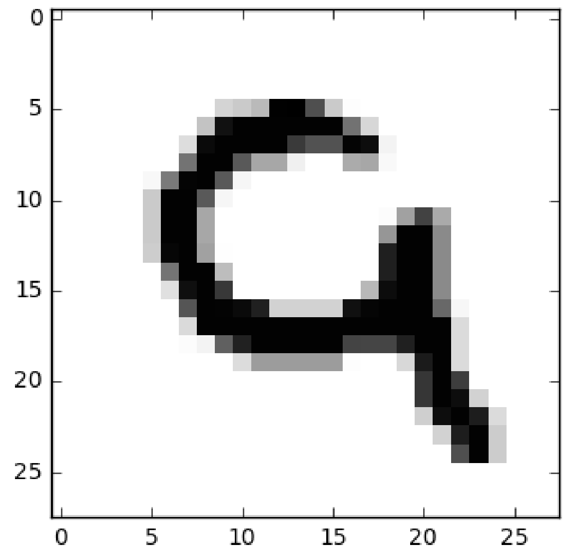
```
In [8]: # Get one and predict
        import random
        r = random.randint(0, mnist.test.num_examples - 1)
        print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
        print("Prediction: ", sess.run(
            tf.argmax(hypothesis, 1), feed_dict={X: mnist.test.images[r:r + 1]}))

        plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), cmap='Greys', interpolation='nearest')
        plt.show()
```

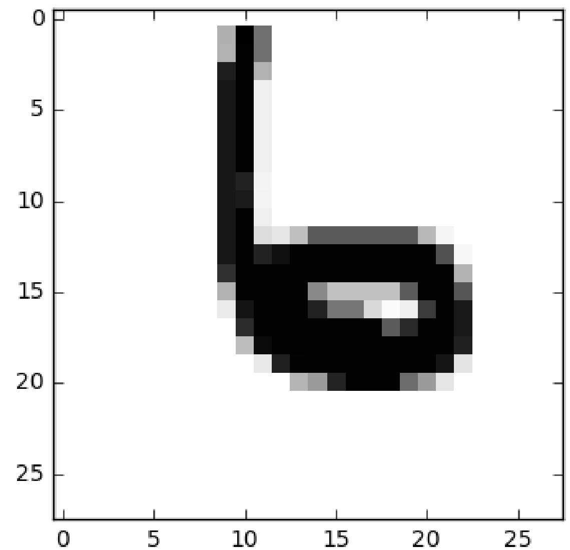
```
Label:  [1]
Prediction:  [1]
```



```
Label:  [9]
Prediction:  [9]
```



```
Label:  [6]
Prediction:  [6]
```



Implementation

```python
import tensorflow as tf
import random
import matplotlib.pyplot as plt

from tensorflow.examples.tutorials.mnist import input_data

tf.set_random_seed(777)  # reproducibility

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# hyper parameters
learning_rate = 0.001
training_epochs = 15
batch_size = 100

# dropout (keep_prob) rate  0.7~0.5 on training, but should be 1 for testing
keep_prob = tf.placeholder(tf.float32)

# input place holders
X = tf.placeholder(tf.float32, [None, 784])
X_img = tf.reshape(X, [-1, 28, 28, 1])   # img 28x28x1 (black/white)
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))
L1 = tf.nn.conv2d(X_img, W1, strides=[1, 1, 1, 1], padding='SAME')
L1 = tf.nn.relu(L1)
L1 = tf.nn.max_pool(L1, ksize=[1, 2, 2, 1],
                    strides=[1, 2, 2, 1], padding='SAME')
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)
                                              ...
# L5 Final FC 625 inputs -> 10 outputs
W5 = tf.get_variable("W5", shape=[625, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
logits = tf.matmul(L4, W5) + b5
```

```python
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
# 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(mnist.train.num_examples / batch_size)

    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
        c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
        avg_cost += c / total_batch

    print('Epoch:', '%04d' % (epoch + 1), 'cost =', '{:.9f}'.format(avg_cost))

print('Learning Finished!')

correct_prediction = tf.equal(tf.argmax(logits, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print('Accuracy:', sess.run(accuracy, feed_dict={
      X: mnist.test.images, Y: mnist.test.labels, keep_prob: 1}))
```

Epoch: 0001 cost = 0.409386985	Epoch: 0006 cost = 0.047962842	Epoch: 0011 cost = 0.032899911
Epoch: 0002 cost = 0.100627775	Epoch: 0007 cost = 0.042300057	Epoch: 0012 cost = 0.031550007
Epoch: 0003 cost = 0.072903002	Epoch: 0008 cost = 0.039930305	Epoch: 0013 cost = 0.028447655
Epoch: 0004 cost = 0.060526004	Epoch: 0009 cost = 0.034254246	Epoch: 0014 cost = 0.028178741
Epoch: 0005 cost = 0.052039743	Epoch: 0010 cost = 0.033424444	Epoch: 0015 cost = 0.027132071

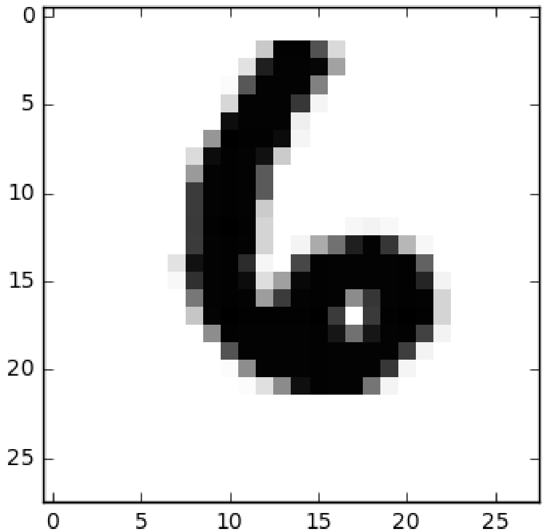
Learning Finished!
Accuracy: 0.9939

Implementation

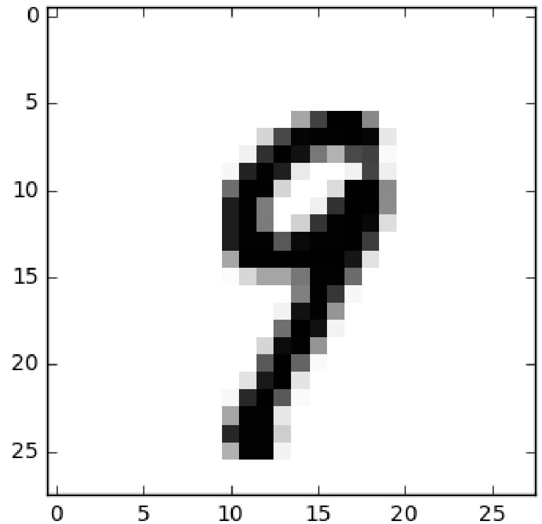
In [2]:
```
r = random.randint(0, mnist.test.num_examples - 1)
print("Label: ", sess.run(tf.argmax(mnist.test.labels[r:r + 1], 1)))
print("Prediction: ", sess.run(
    tf.argmax(logits, 1), feed_dict={X: mnist.test.images[r:r + 1], keep_prob: 1}))

plt.imshow(mnist.test.images[r:r + 1].reshape(28, 28), cmap='Greys', interpolation='nearest')
plt.show()
```

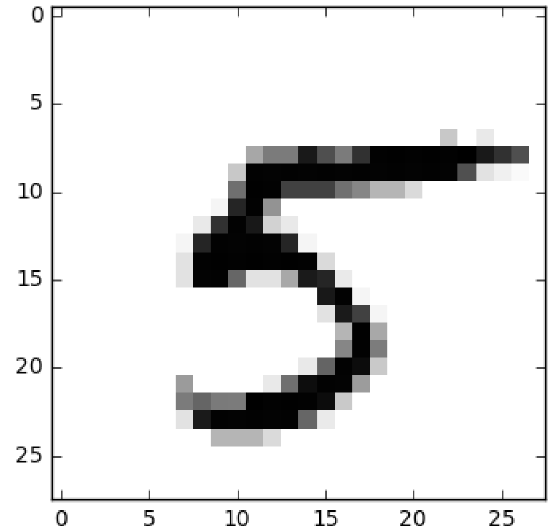
```
Label:  [6]
Prediction:  [6]
```



```
Label:  [9]
Prediction:  [9]
```



```
Label:  [5]
Prediction:  [5]
```



Implementation

In [1]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import cv2


#레이블 불러오기
label = np.loadtxt('food-101/food-101/meta/labels.txt', delimiter=',', dtype=np.str)
label_sz = np.shape(label)
print('Label 개수 :',label_sz[0]) #레이블 확인
```

```
Label 개수 : 101
```

In [2]:
```
train = np.loadtxt('food-101/food-101/meta/train.txt', delimiter=',', dtype='bytes').astype(str)
print(train)
```

```
['apple_pie/1005649' 'apple_pie/1014775' 'apple_pie/1026328' ...,
 'waffles/982668' 'waffles/995085' 'waffles/999047']
```

In [ ]:
```
train_sz = np.shape(train)

train_img=[]

for i in range(train_sz[0]):
    label_temp = train[i].split('/')[0]
    train_temp=cv2.imread('food-101/food-101/images/'+ train[i] + '.jpg')
    train_temp=cv2.resize(train_temp,(256,256))

    train_img.append([train_temp,label_temp])

np.shape(train_img)
```

In [14]:
```
label_train = train[0].split('/')[0]
train_img=[]
train_img.append([temp, label_train])
np.shape(train_img)
```

Out[14]:
```
(1, 2)
```

Q & A

Thank You!!!