Deep Learning

Convolutional Neural Networks(CNN)

Background of Convolutional Neural Networks

CNN, Convolution Neural Network

CNN examples

Case study

Yoon Joong Kim

Department of Computer Engineering, Hanbat National University

yjkim@hanbat.ac.kr

Contents

1. Background of Convolutional Neural Networks

A bit of history:

Hubel & Wiesel,

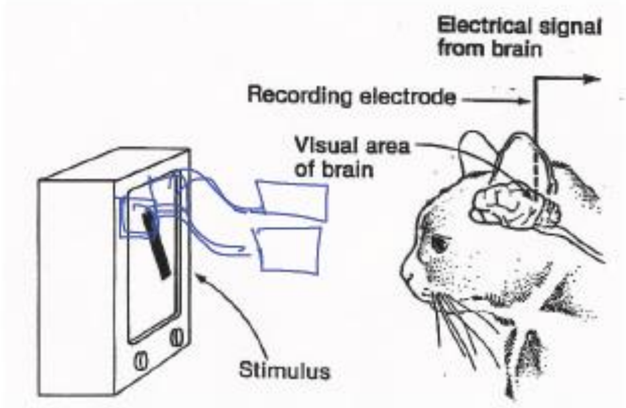
1959

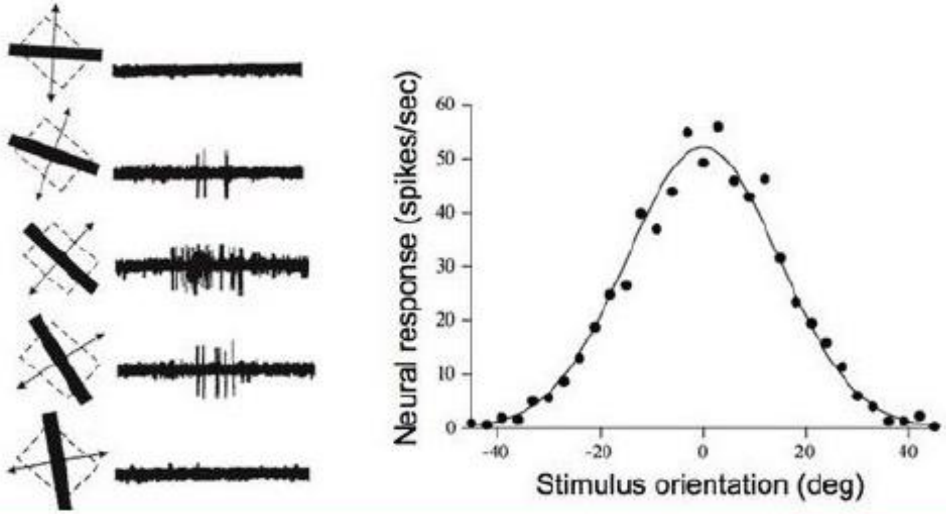
RECEPTIVE FIELDS OF SINGLE NEURONES IN
THE CAT'S STRIATE CORTEX

1962

RECEPTIVE FIELDS, BINOCULAR INTERACTION
AND FUNCTIONAL ARCHITECTURE IN
THE CAT'S VISUAL CORTEX

1968...

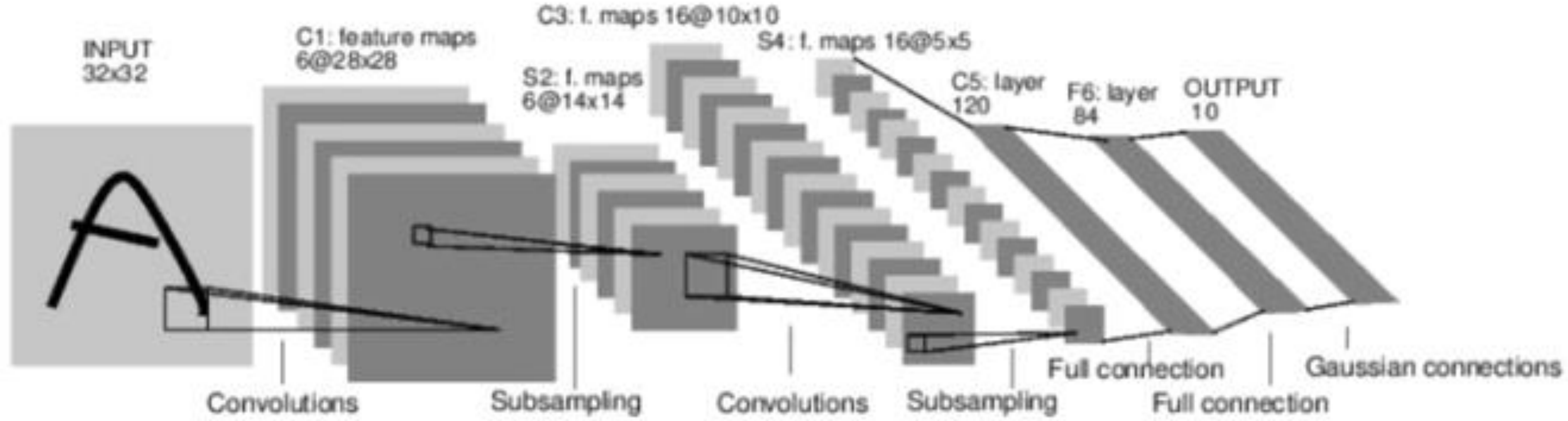




1. Background of Convolutional Neural Networks

Case Study: LeNet-5

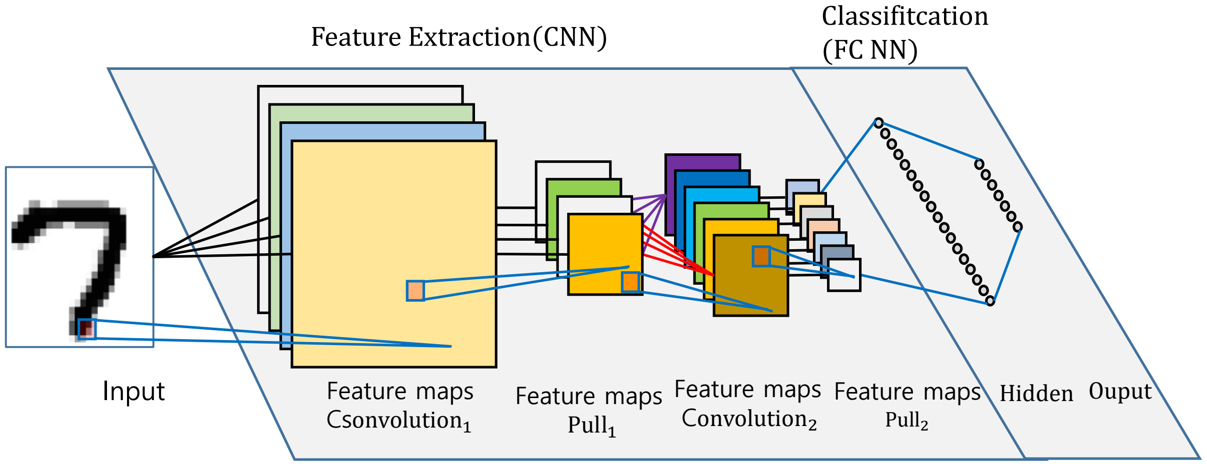
[LeCun et al., 1998]



Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

2. CNN, Convolution Neural Network

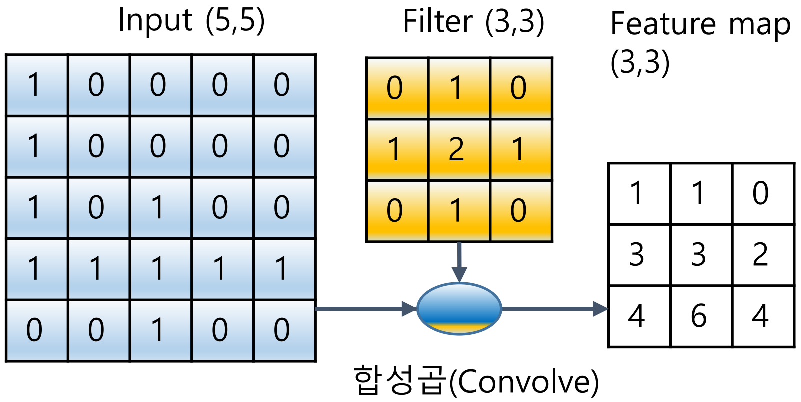
- 개요
  - 전연결신경망(FC NN,Fully Connected Neural Network)을 이용한 이미지 분류의 경우 3차원 이미지를 1차원으로 평면화 하여야 한다. 이미지 공간 정보 유실로 인한 정보 부족으로 인공 신경망이 특징을 추출 및 학습이 비효율적이고 정확도를 높이는데 한계가 있습니다. 이미지의 공간 정보를 유지한 상태로 학습이 가능한 모델이 바로 CNN(Convolutional Neural Network)입니다.
- FC NN에 대한 CNN의 차이점
  - 각 레이어의 입출력 데이터의 형상 유지
  - 이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식
  - 복수의 필터로 이미지의 특징 추출 및 학습
  - 추출한 이미지의 특징을 모으고 강화하는 Pooling 레이어
  - 필터를 공유 파라미터로 사용하기 때문에, 일반 인공 신경망과 비교하여 학습 파라미터가 매우 적음

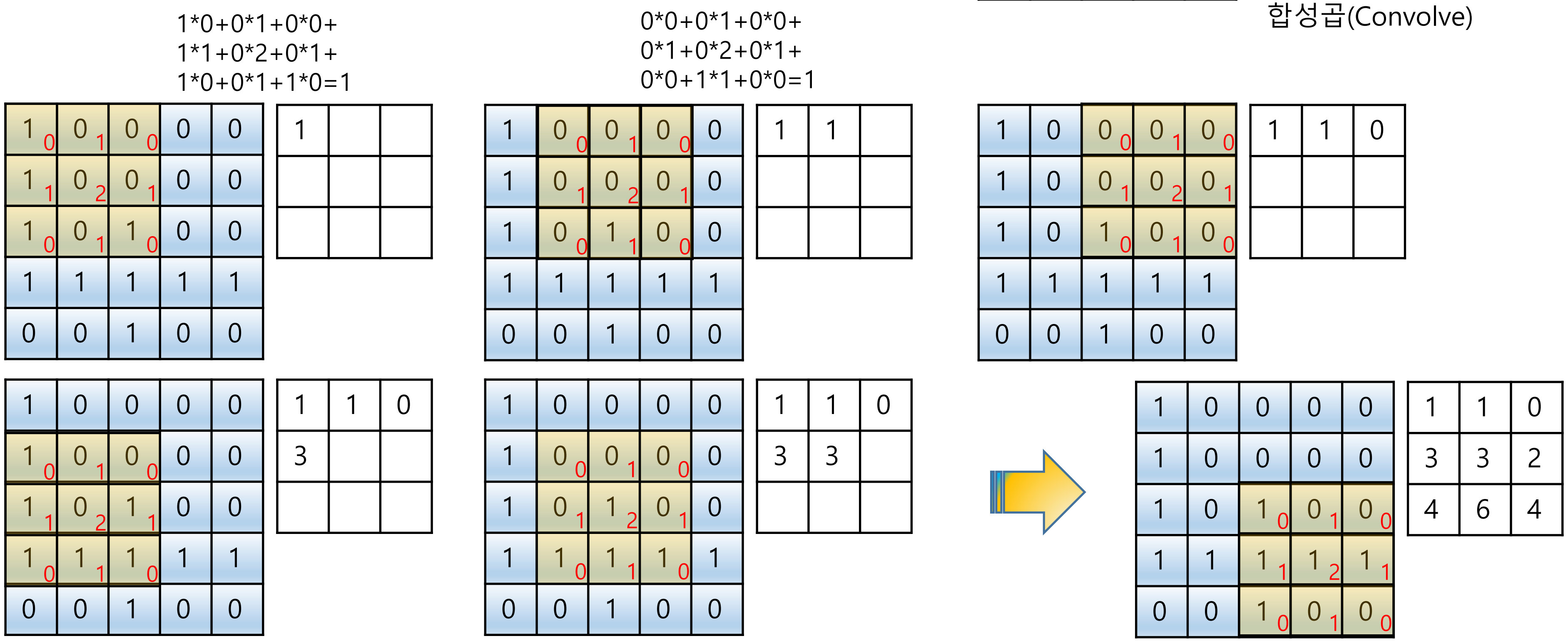


- 1. 주요용어
  - 합성곱(Convolution)
  - 채널(Channel)
  - 필터(Filter)
  - 커널(Kernel)
  - 스트라이드(Stride)
  - 패딩(Padding)
  - 피처맵(Feature map)
  - 액티베이션 맵(Activation Map)
  - 풀링 레이어(Pooling layer)

2.1 convolution

- 2.1 합성곱(Convolution)
  - *합성곱 연산은 두 함수 f, g 가운데 하나의 함수를 반전(reverse), 전이(shift)시킨 다음, 다른 하나의 함수와 곱한 결과를 적분하는 것을 의미한다. 출처: https://ko.wikipedia.org/wiki/%ED%95%A9%EC%84%B1%EA%B3%B1*



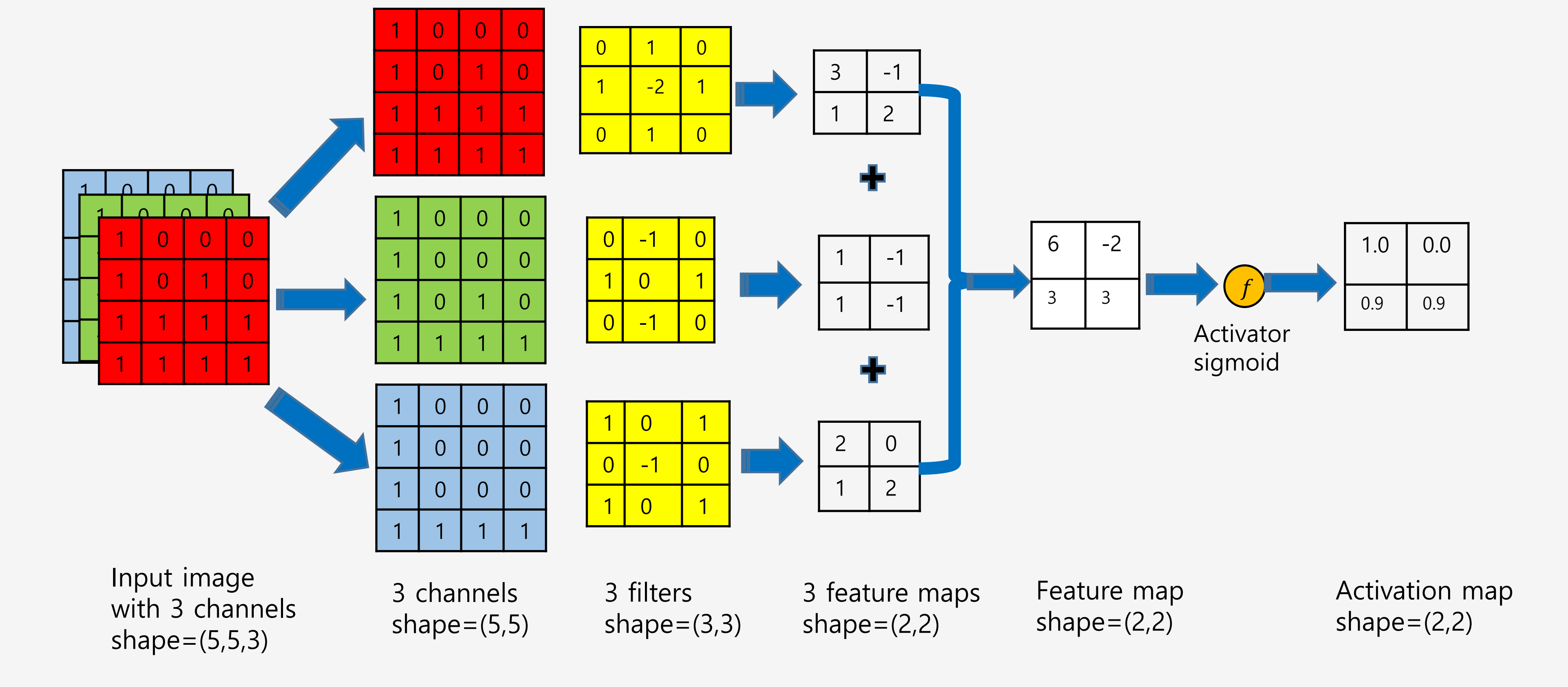


2.2 channel

- 2.2 채널, channel
  - 컬러 이미지 픽셀의 색상은 Red, Green, Blue의 크기로 합성됩니다. 따라서 컬러사의 이미지는 3장의 채널로 구성된다. 컬러 이미지의 shape=(32,32,3) 흑백 이미지는 shape=(32,32,1)이 된다.

2.3 filter, kernel, stride, feature map and activation map

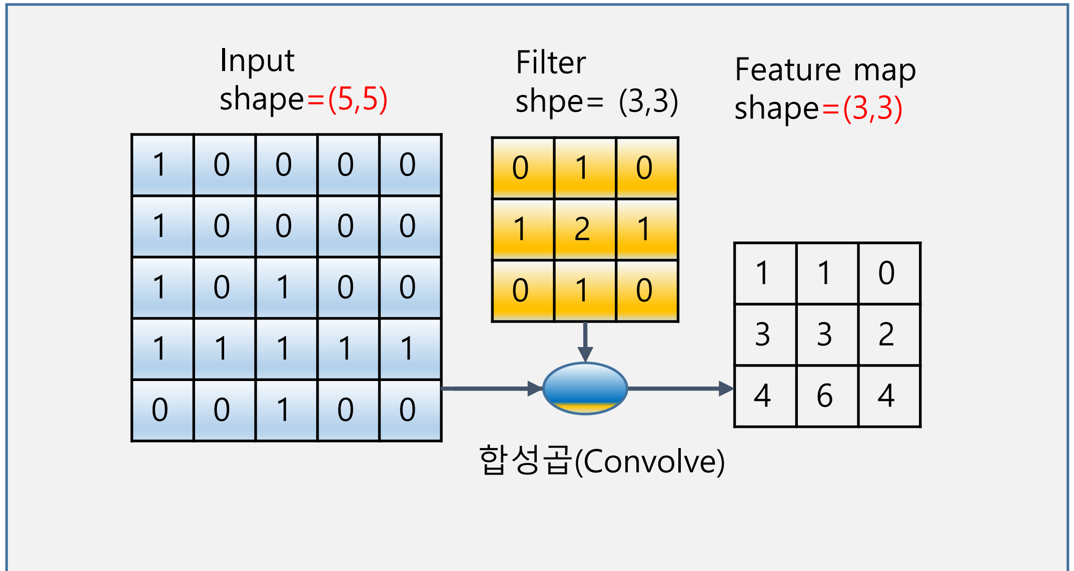
- 2.3 필터,스트라이드, 피처맵(feature map),커널(kernel)
  - 커널 : 필터와 동일, 스트라이드는 필터의 이동 픽셀 수, 피처맵 : 합성곱의 계산결과 이미지
  - 액티베이션 맵 : 피처맵에 액티베이션을 적용한 결과



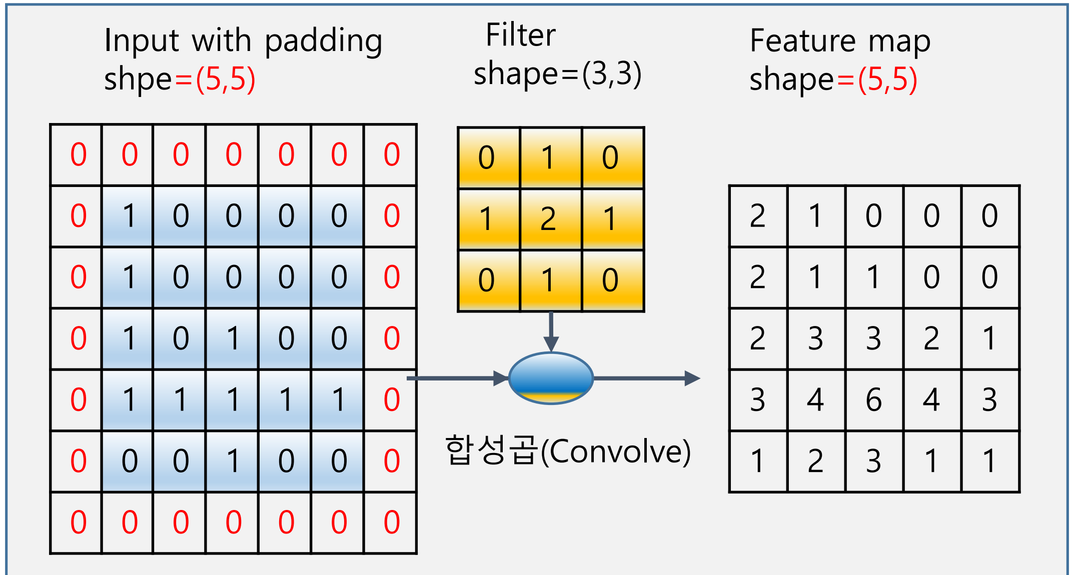
멀티채널 이미지의 합성곱 계산 절차

2.4 padding

- 2.4 패딩
  - 합성곱을 하면 입력에 비하여 피처맵의 사이이즈가 작아진다. 같은 크기가 되도록 입력의 가장자리에 0을 채우는 과정을 말한다.



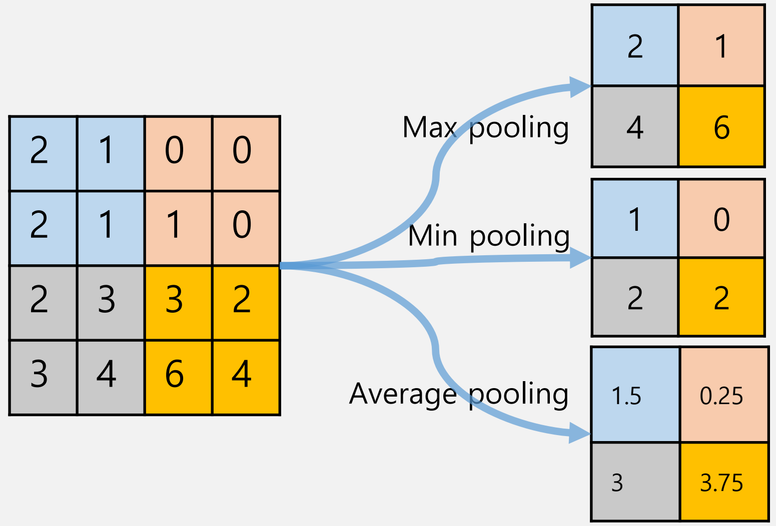
패딩이 없는 합성곱(Convolve)의 예



패딩을 추가한 합성곱(Convolve)의 예

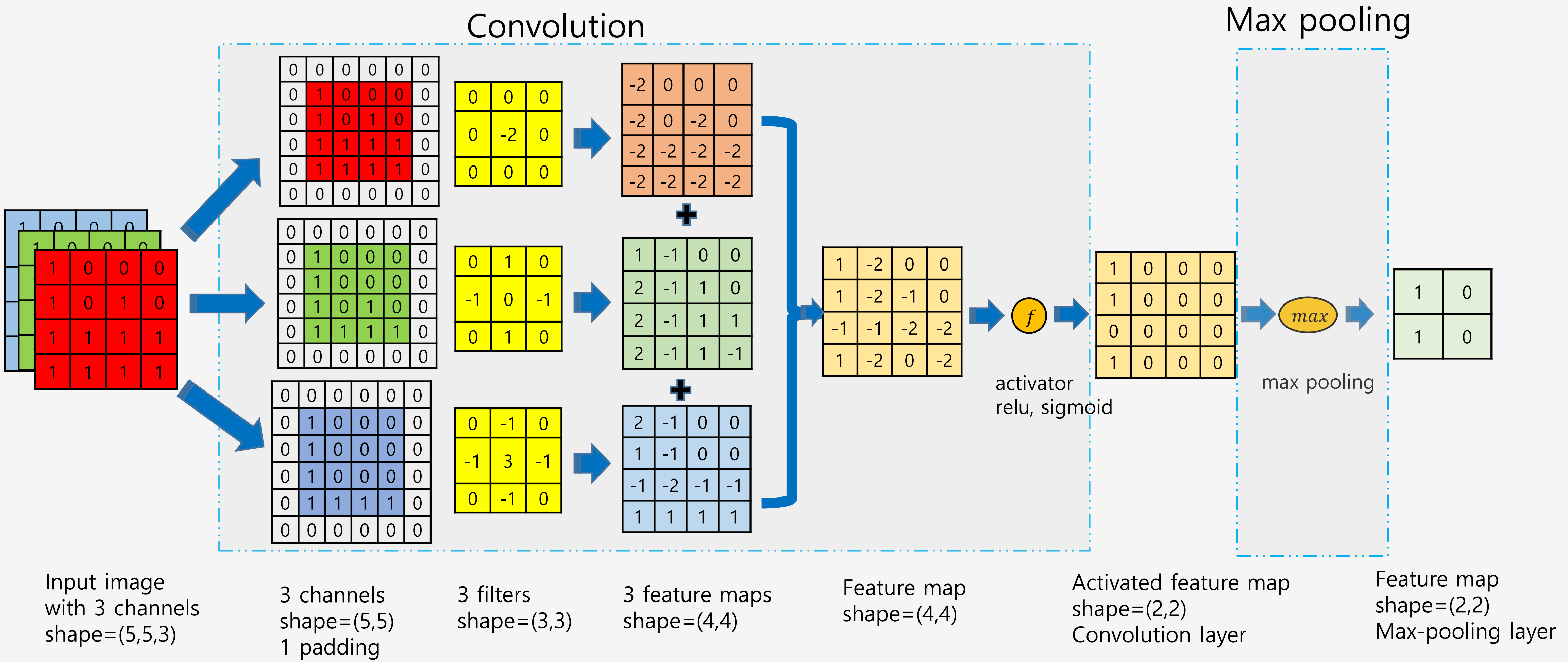
2.5 pooling layer

- 2.5 pooling layer
  - Convolution layer의 출력(activation map)의 일정 값은 강조하거나 크기를 줄이는 목적으로 사용된다. Pooling 방법은 max, min, average pooling이 있다. 일반적으로 pooling size와 stride size로 한다.
  - Pooling 특징
    - 학습대상 파라미터가 없음
    - Pooling 레이어를 통과하면 행렬의 크기 감소
    - Pooling 레이어를 통해서 채널 수 변경 없음



Pooling example, Max pooling, Min pooling, Average pooling

2.5 pooling layer



멀티채널 이미지의 합성곱 layer 및 max pooling layer 계산 예

2.6 출력 레이어의 크기 계산

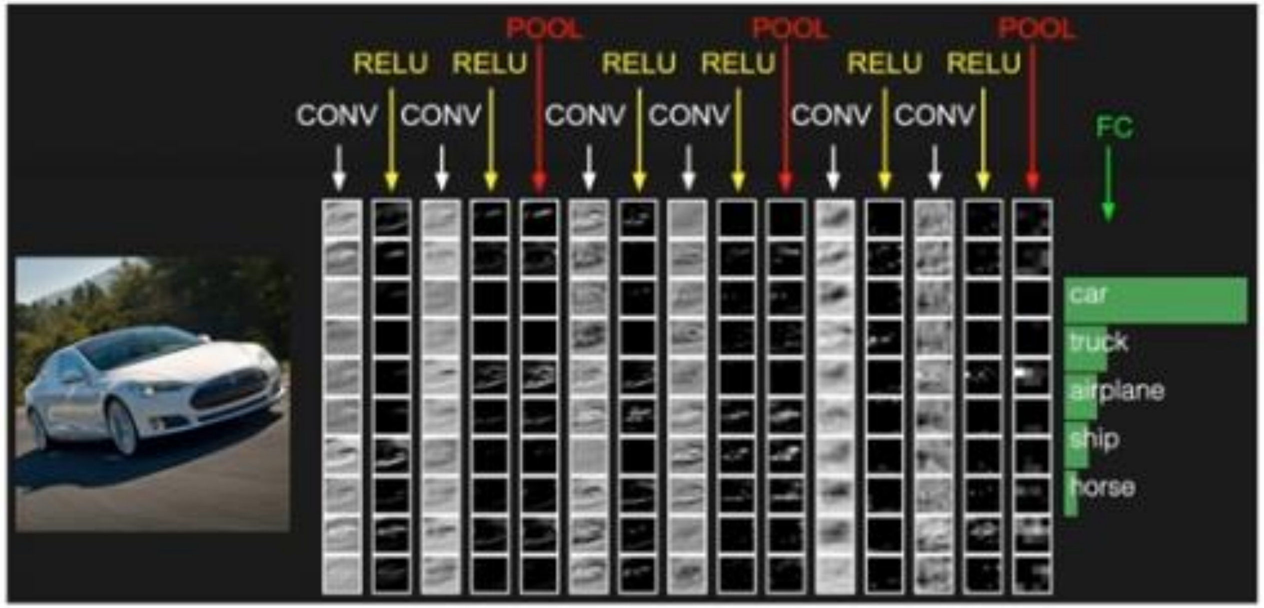
- 2.6 레이어의 출력 크기 계산
  - 입력 데이터 높이: H
  - 입력 데이터 폭: W
  - 필터 높이 : FH
  - 필터 폭 : FW
  - Stride 크기 : S
  - 패딩 사이즈 : P

- Convolution layer 출력의 크기
  - $OuputHeight = oh = \frac{\text{H+2P−FH}}{s} + 1$
  - $OuputWidth = ow = \frac{\text{H+2P−FW}}{s} + 1$
- Pooling layer 의 출력 크기
  - $\text{OuputRowSize} = \frac{\text{InputRowSize}}{\text{PoolingSize}}$
  - $\text{OuputColumnSize} = \frac{\text{InputColumnSize}}{\text{PoolingSize}}$

2.7 Fully Connected Layer(FC layer)

Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks



Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 7 - 58   27 Jan 2016

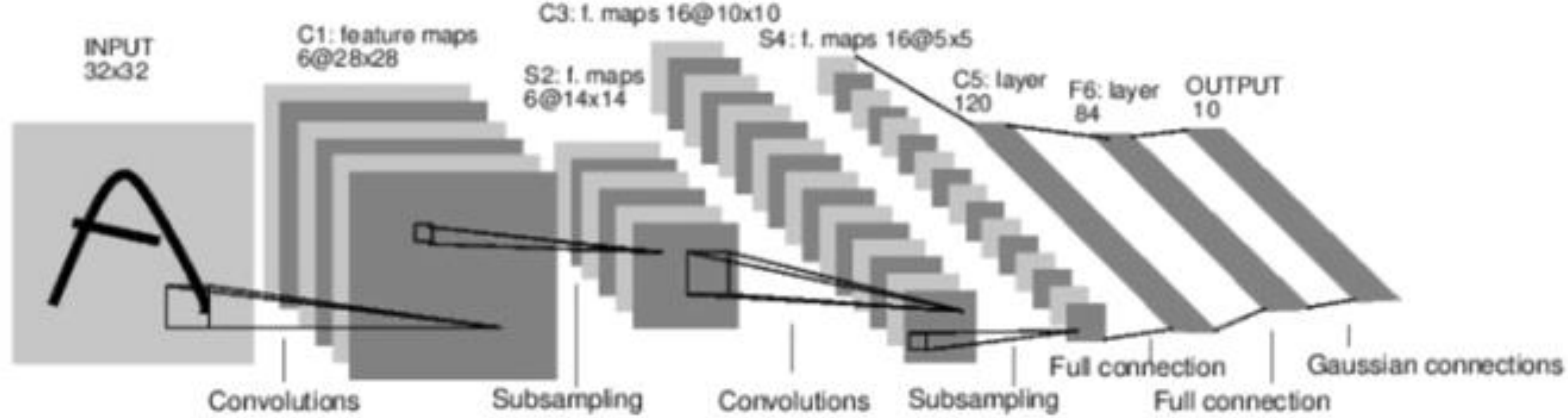
3. Case study

1. LeNet-5
2. AlexNet
3. GoogleNet
4. ResNet
5. Sentence Classification
6. AlphaGo

3.1 LeNet-5

Case Study: LeNet-5

[LeCun et al., 1998]

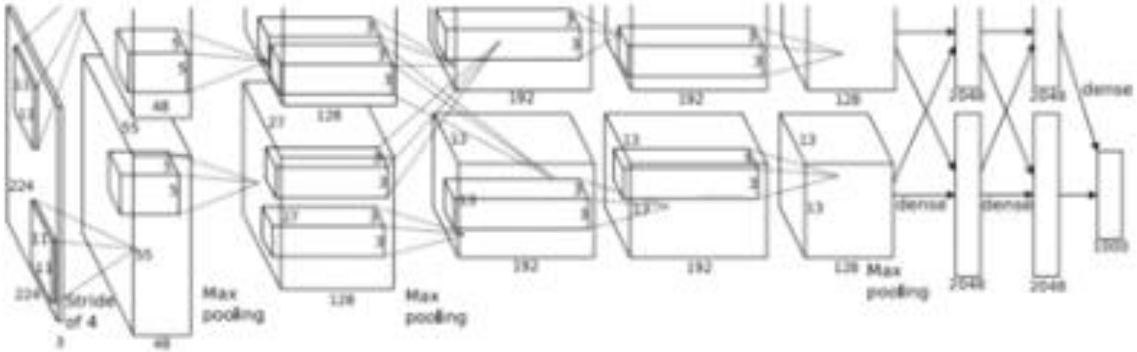


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

3.1 LeNet-5(cont.)

Case Study: AlexNet

[Krizhevsky et al. 2012]



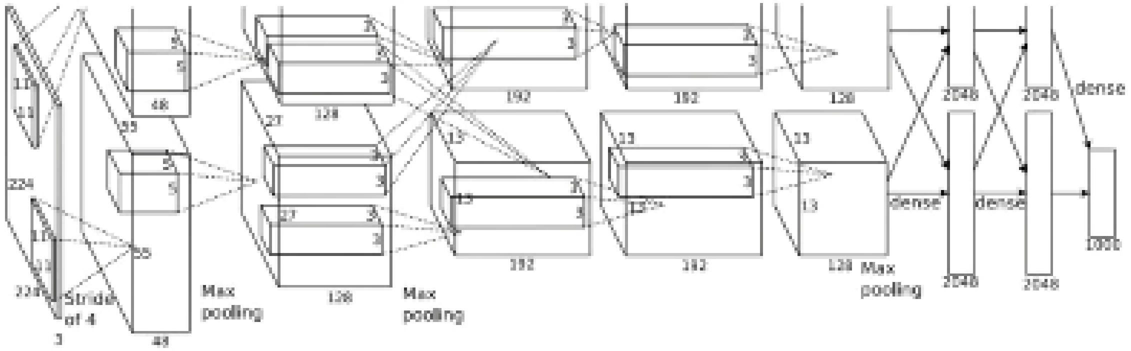
Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

3.2 AlexNet

Case Study: AlexNet

[Krizhevsky et al. 2012]



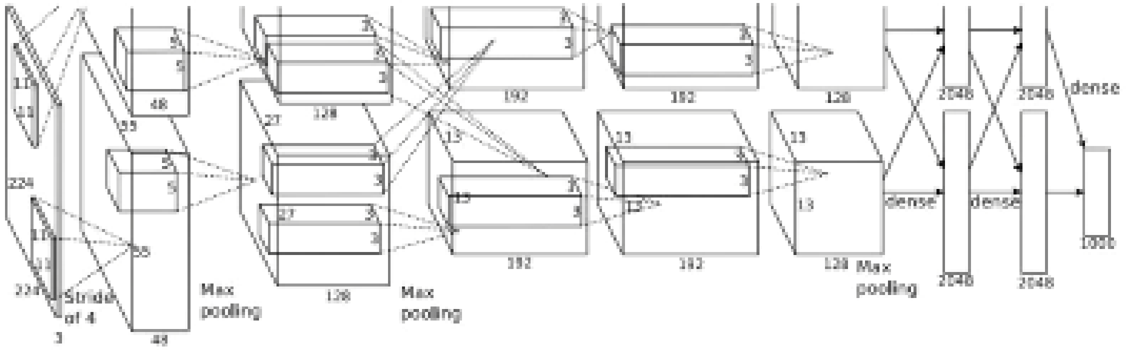
Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

3.2 AlexNet(cont.)

Case Study: AlexNet

[Krizhevsky et al. 2012]

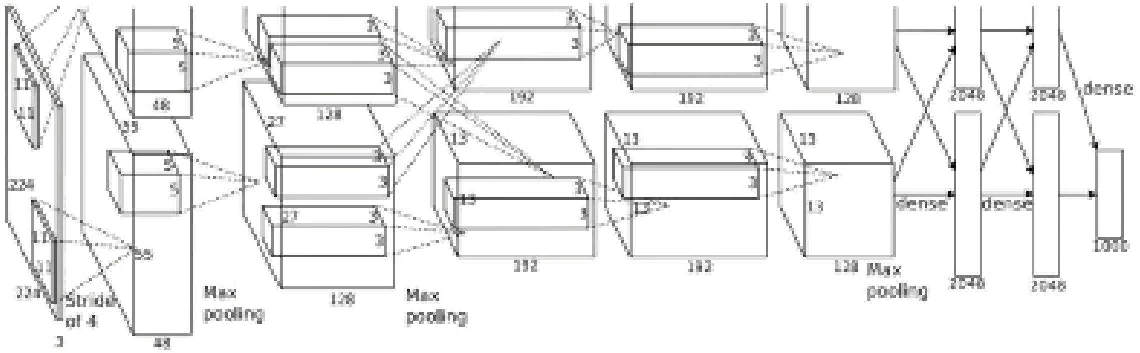


Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

3.2 AlexNet(cont.)

Case Study: AlexNet

[Krizhevsky et al. 2012]

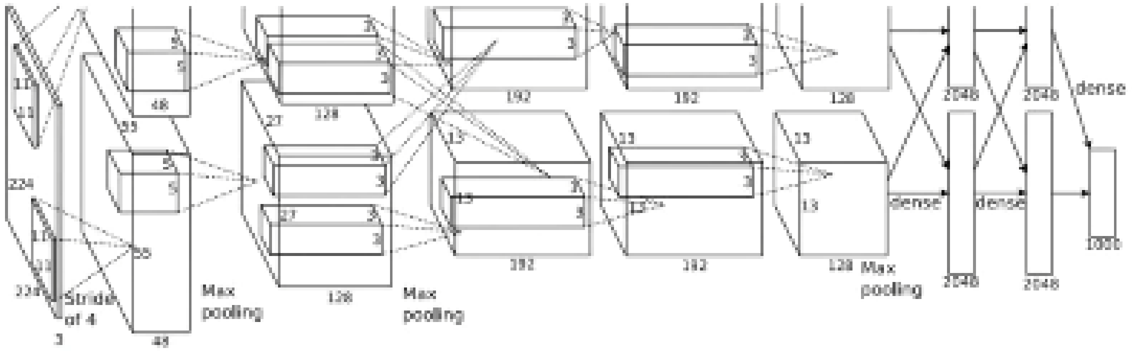


Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

3.3 AlexNet(cont.)

Case Study: AlexNet

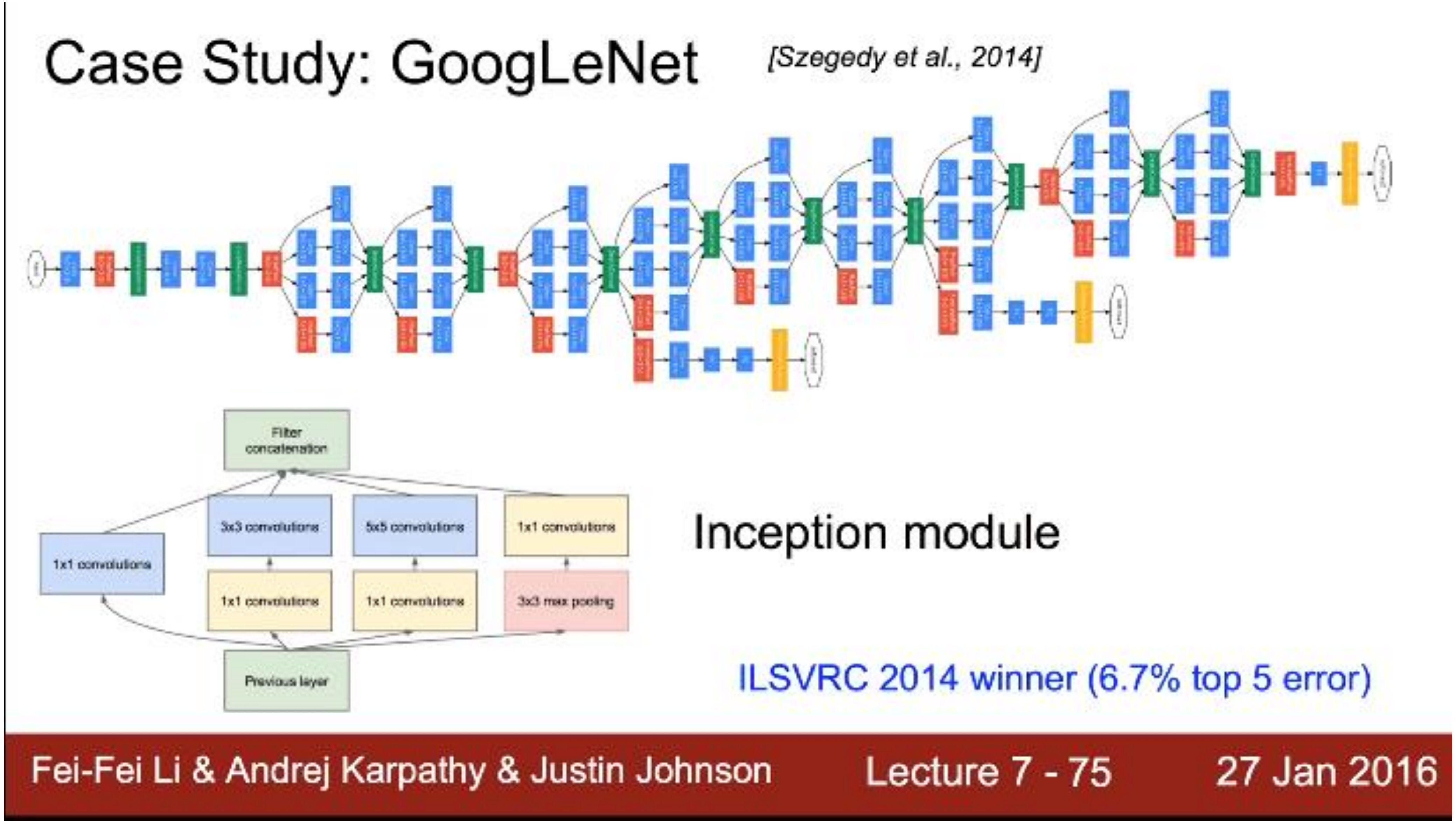
[Krizhevsky et al. 2012]



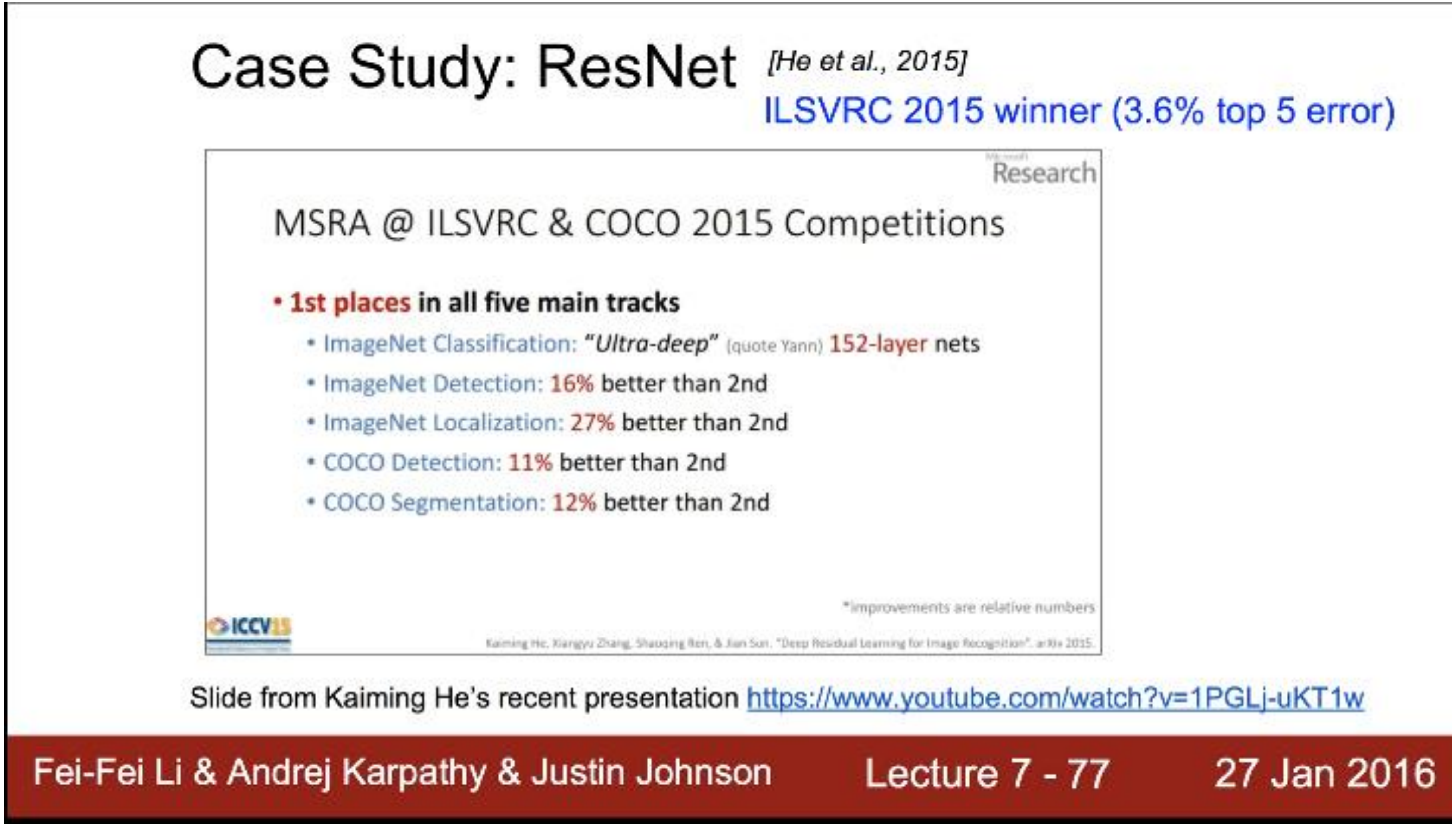
Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

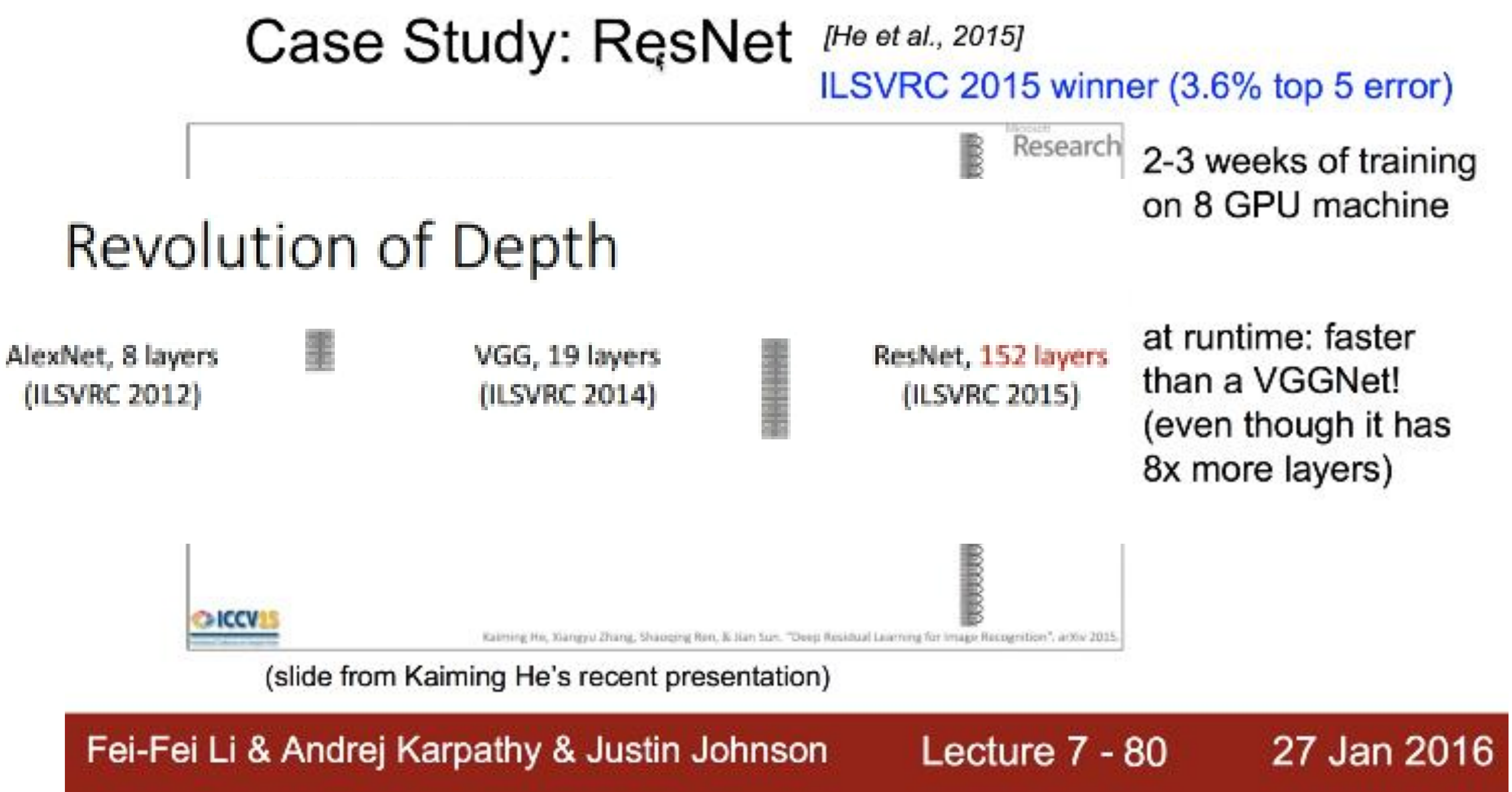
3.3 GoogleNet



3.4 ResNet


Case Study: ResNet
[He et al., 2015]
ILSVRC 2015 winner (3.6% top 5 error)
Research
MSRA @ ILSVRC & COCO 2015 Competitions
• 1st places in all five main tracks
• ImageNet Classification: "Ultra-deep" (quote Yann) 152-layer nets
• ImageNet Detection: 16% better than 2nd
• ImageNet Localization: 27% better than 2nd
• COCO Detection: 11% better than 2nd
• COCO Segmentation: 12% better than 2nd
*improvements are relative numbers
ICCV15
Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.
Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w
Fei-Fei Li & Andrej Karpathy & Justin Johnson
Lecture 7 - 77
27 Jan 2016

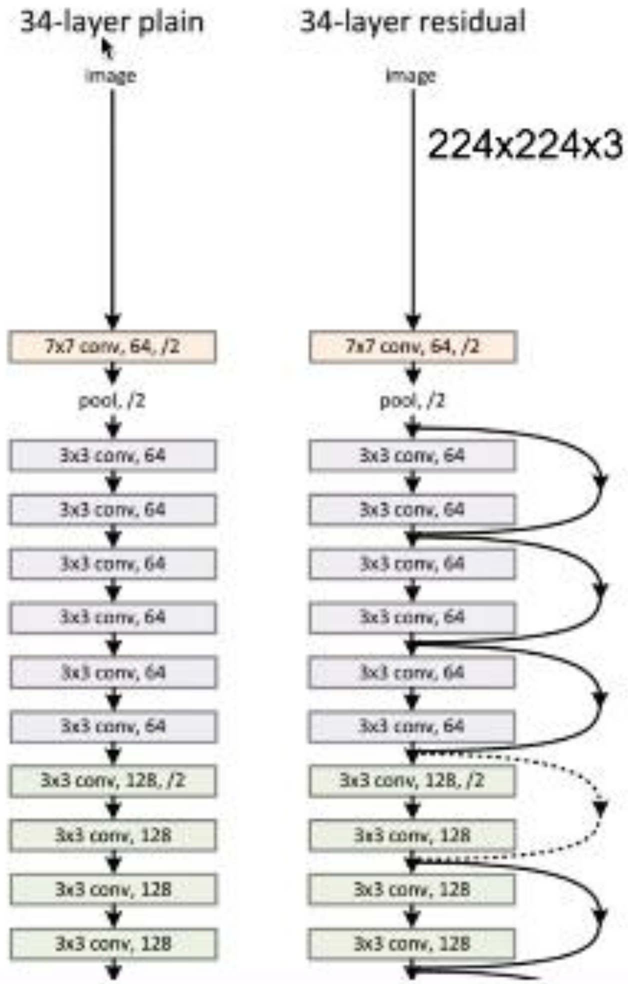
3.4 ResNet (cont.)



3.4 ResNet (cont.)

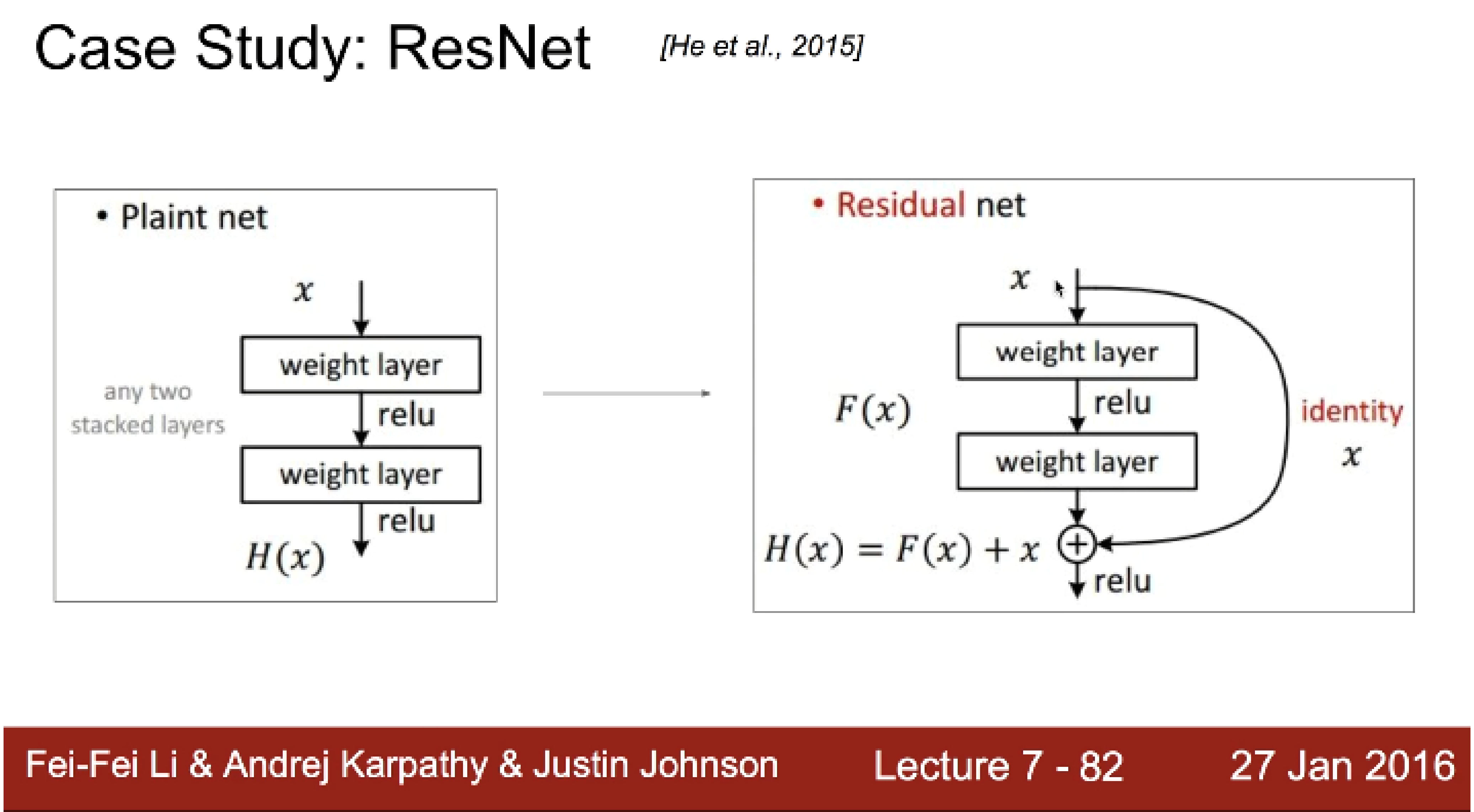
Case Study: ResNet

[He et al., 2015]

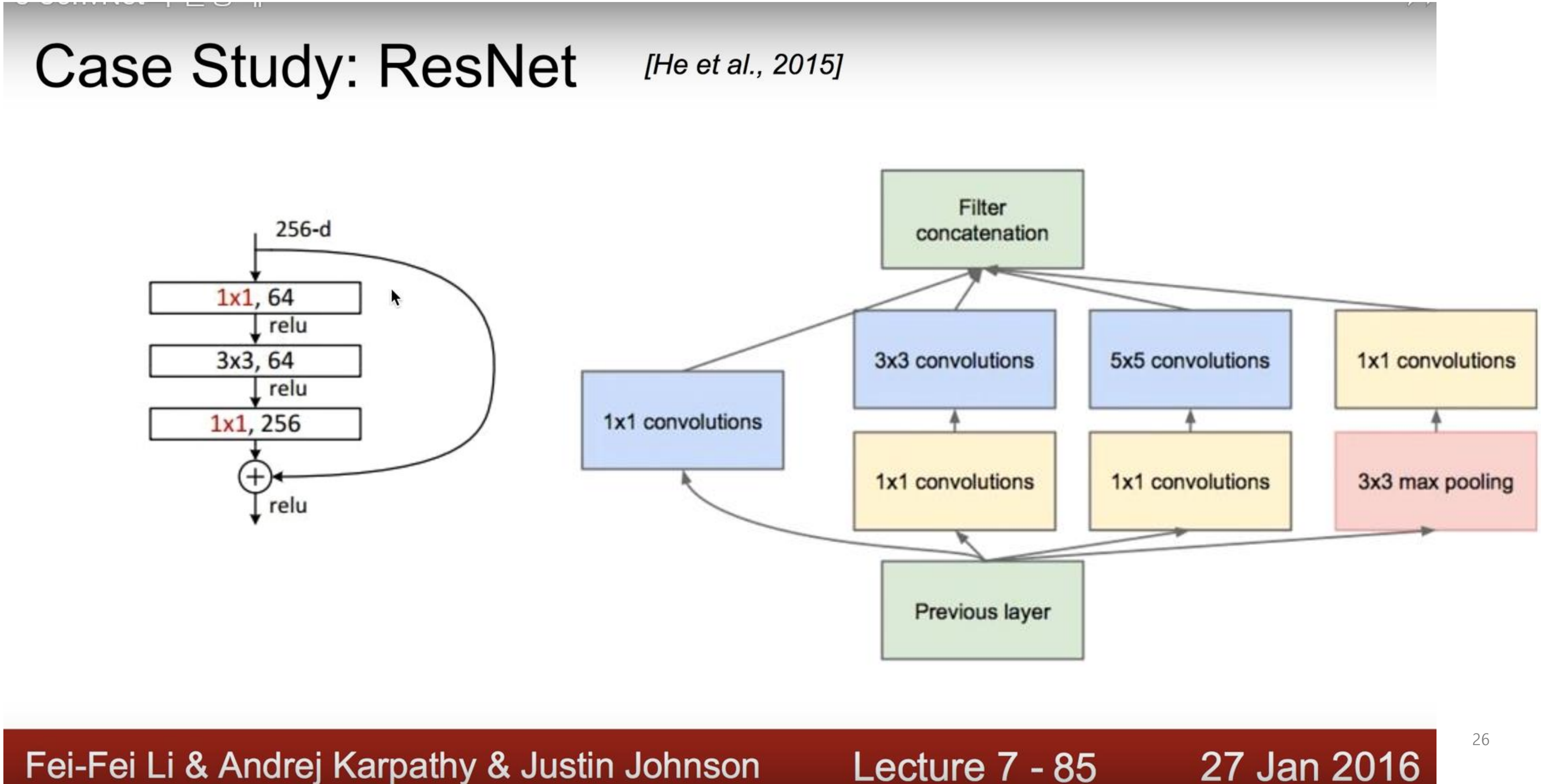


Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 81 27 Jan 2016

3.4 ResNet (cont.)



3.4 ResNet (cont.)



3.5 Sentence Classification

- Convolution neural networks for sentence classification [Yoon Kim, 2014]

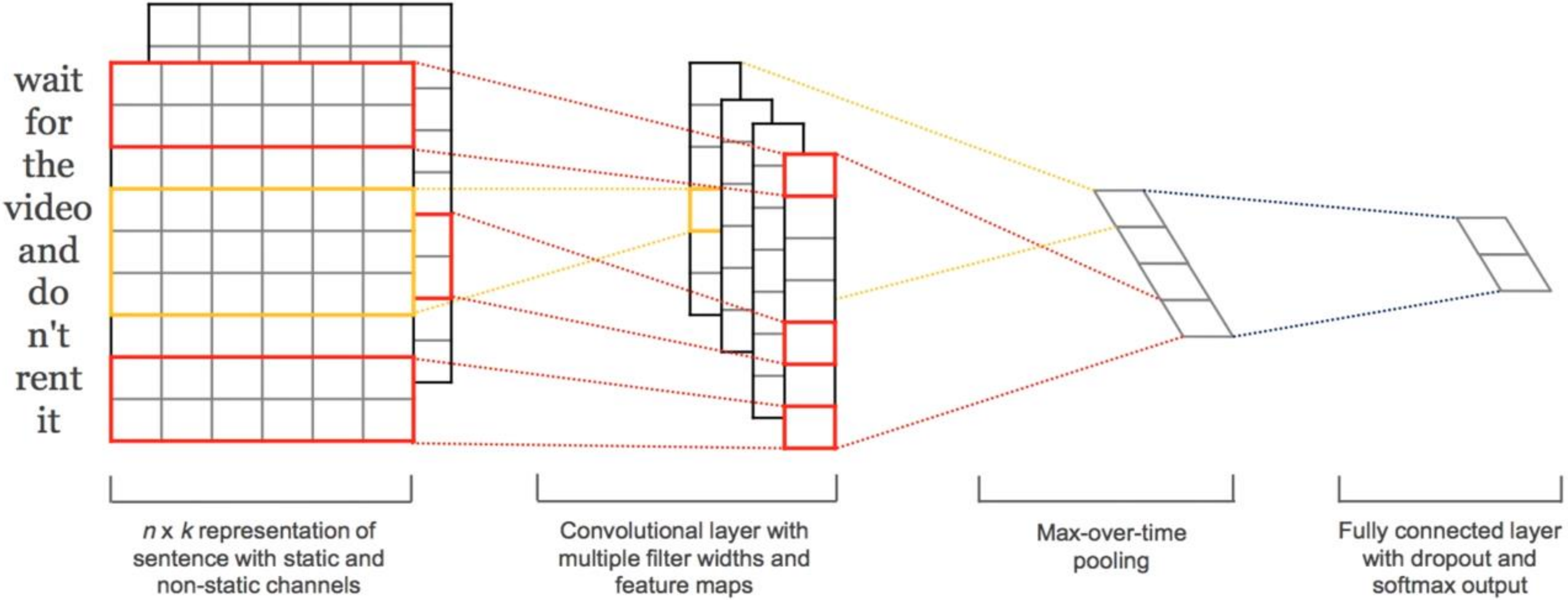
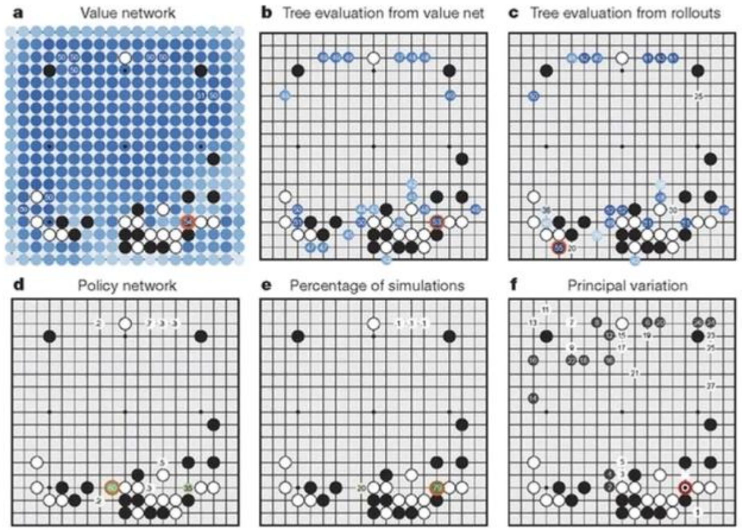


Figure 1: Model architecture with two channels for an example sentence.

3.6 AlphaGo

Case Study Bonus: DeepMind's AlphaGo

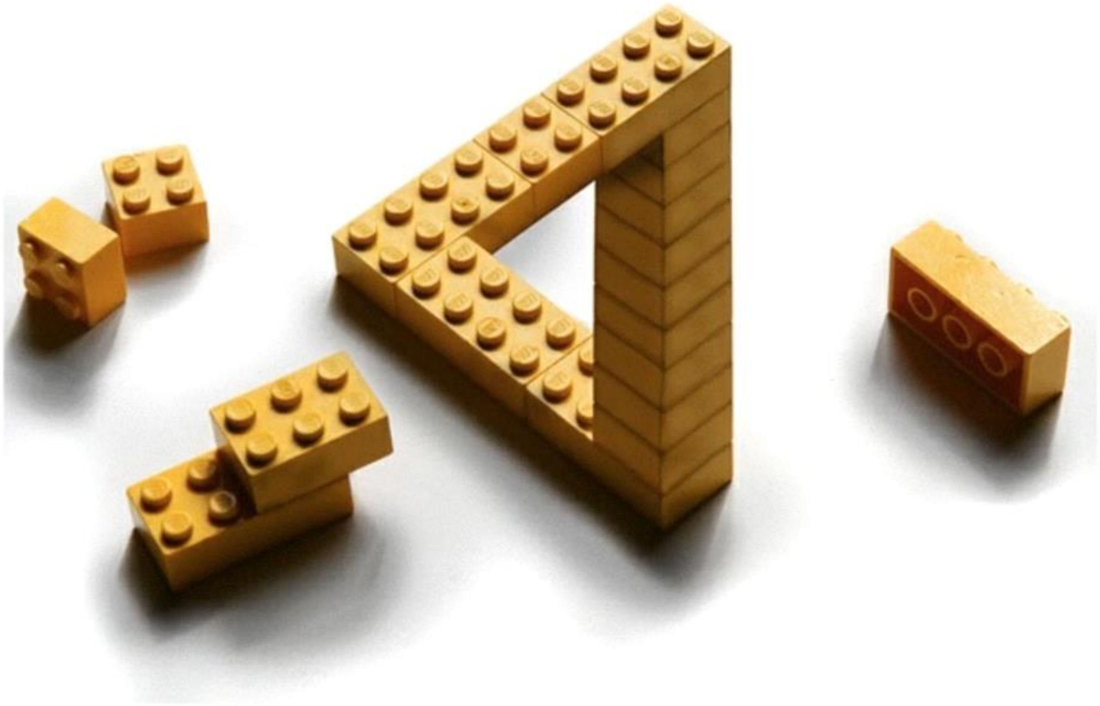


3.6 AlphaGo(cont.)

The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

'The only limit is your imagination'



http://itchyi.squarespace.com/thelatest/2012/5/17/the-only-limit-is-your-imagination.html

4. CNN examples

• 4.1 CNN 구성 예

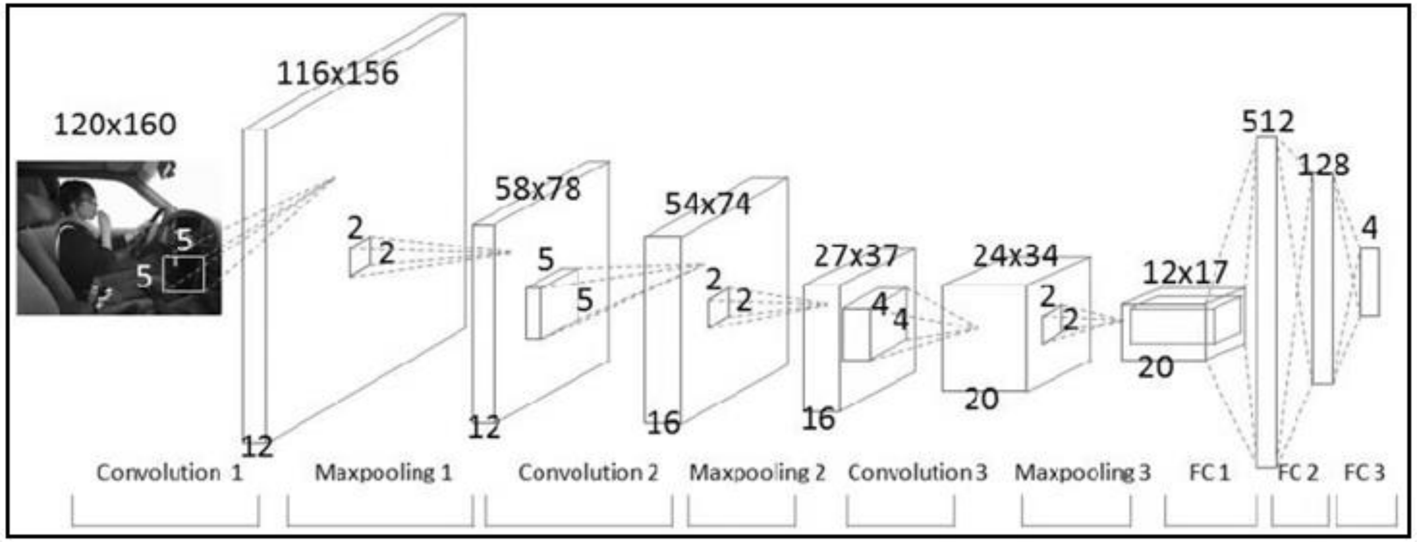
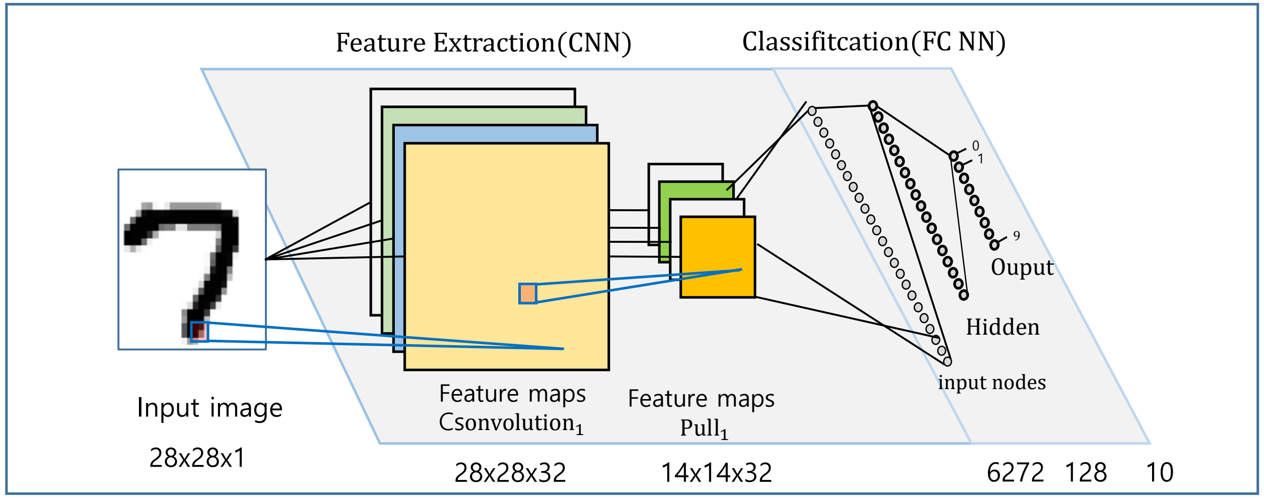


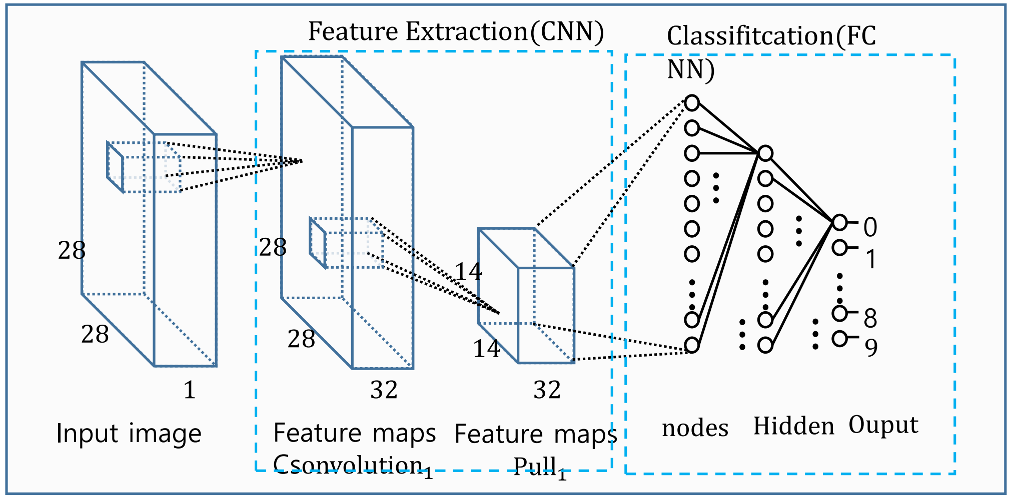
그림 8: 전형적인 CNN, 출처: https://www.researchgate.net/figure/Architecture-of-our-unsupervised-CNN-Network-contains-three-stages-each-of-which_283433254

```
model = Sequential()
model.add(Conv2D(12, kernel_size=(5, 5), activation='relu', input_shape=(120, 160, 1)))      #116x156
model.add(MaxPooling2D(pool_size=(2, 2)))                                                    #  58x78
model.add(Conv2D(16, kernel_size=(5, 5), activation='relu'))          #54x74
model.add(MaxPooling2D(pool_size=(2, 2)))                             #   27x37
model.add(Conv2D(20, kernel_size=(4, 4), activation='relu'))          # 24x34
model.add(MaxPooling2D(pool_size=(2, 2)))                             #   12x17
model.add(Flatten())                                                  #12x17=204
model.add(Dense(128, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

4.2 Mnist digit classifier with 1 Convolution layer

- 1 Convolution NN



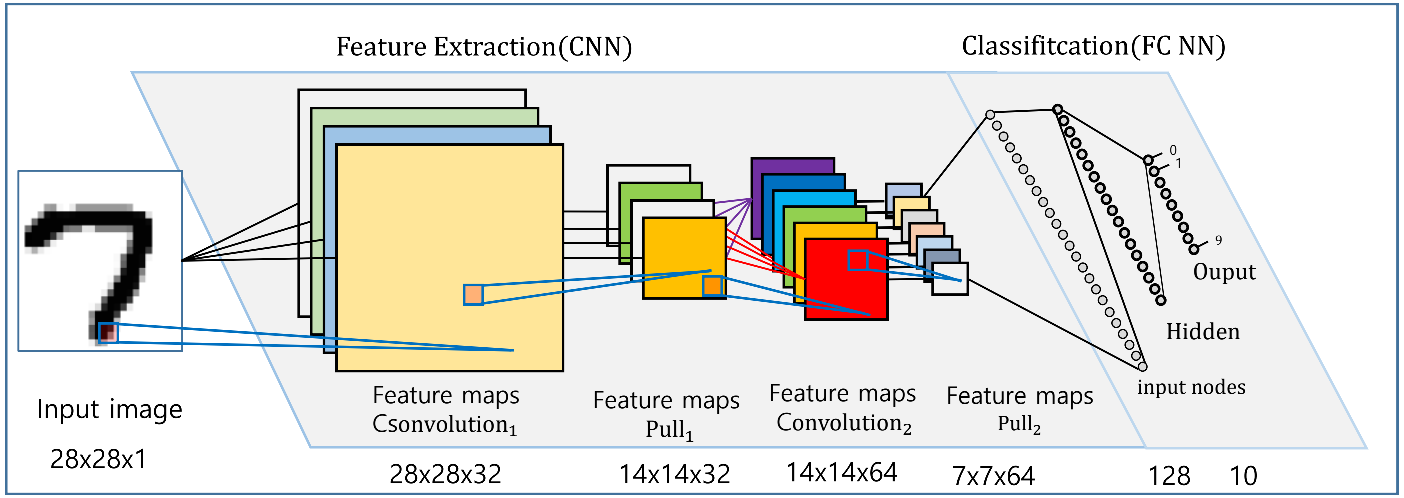


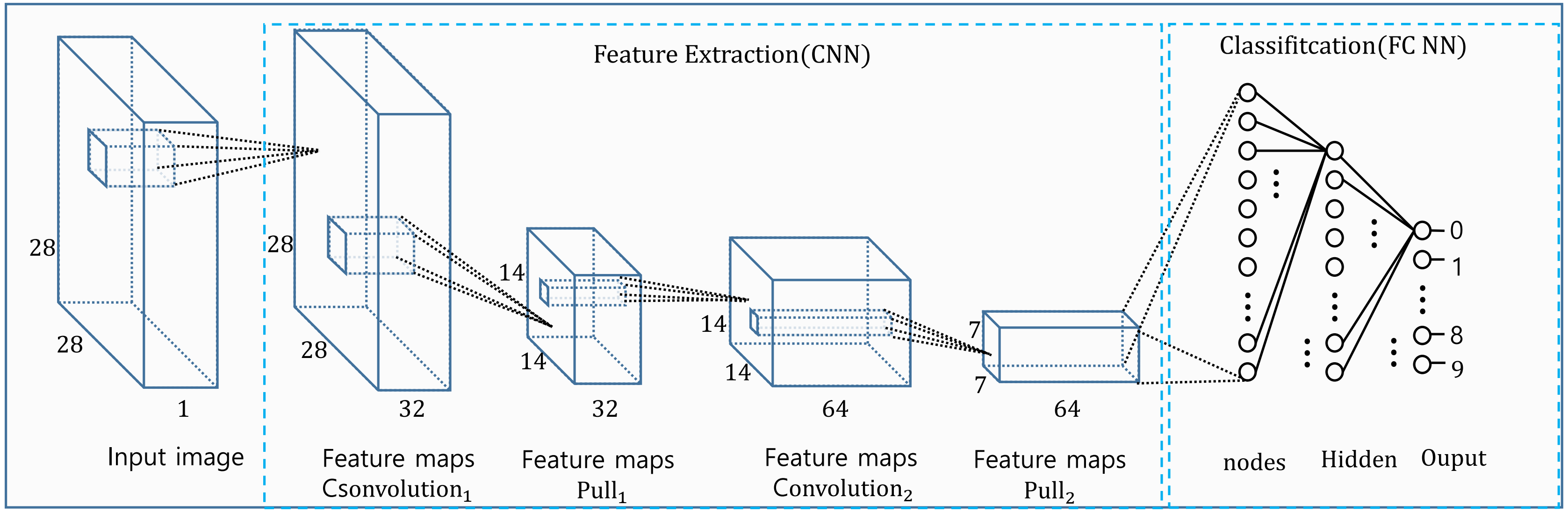

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',input_shape=input_shape)) #26,26,32
model.add(MaxPooling2D(pool_size=(2, 2)))                    #14,14,32
model.add(Dropout(0.25))
model.add(Flatten())                                         #6272 = 14*14*32
model.add(Dense(128, activation='relu'))                     #128
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))          #10
```

```
Epoch 10/12
60000/60000 [==============================] - 4s 73us/step - loss: 0.0459 - acc: 0.9852 - val_loss: 0.0402 - val_acc: 0.9865
Epoch 11/12
60000/60000 [==============================] - 5s 78us/step - loss: 0.0431 - acc: 0.9859 - val_loss: 0.0373 - val_acc: 0.9881
Epoch 12/12
60000/60000 [==============================] - 5s 75us/step - loss: 0.0389 - acc: 0.9873 - val_loss: 0.0357 - val_acc: 0.9880
Test loss: 0.035724134841622436
Test accuracy: 0.988
```

4.2 Mnist digit classifier with deep Convolution layers

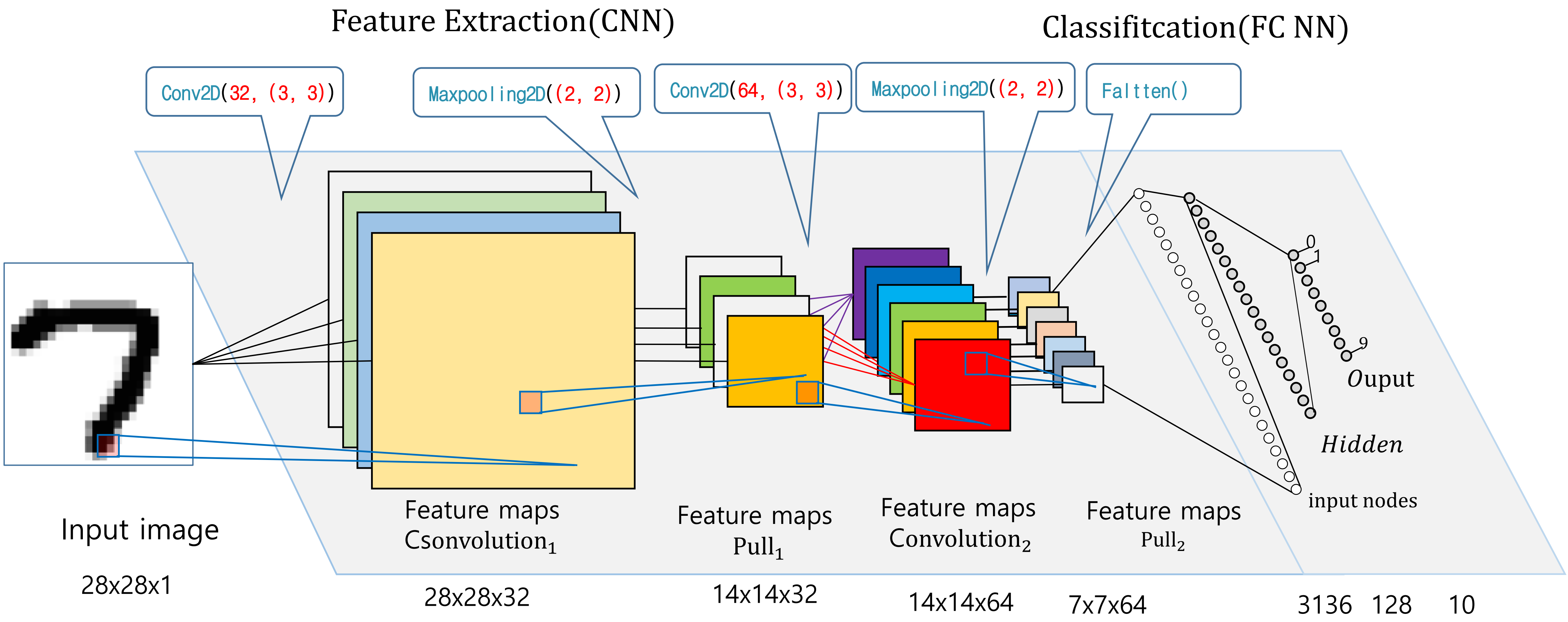
- 2 Convolution NN





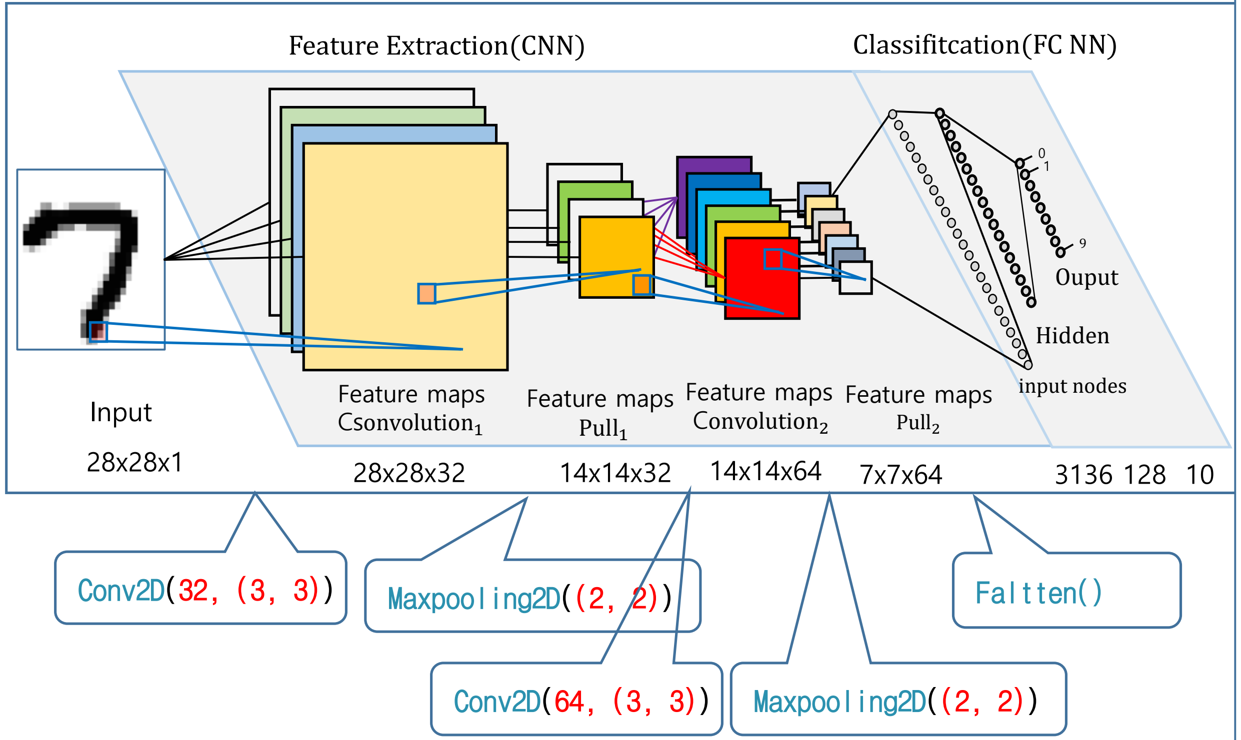
4.2 Mnist digit classifier with deep Convolution layers

- 2 Convolution NN



4.2 Mnist digit classifier with deep Convolution layers

- 2 Convolution NN




```
model = Sequential()
#convolution layer
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                 input_shape=input_shape))                  #26,26,32
model.add(MaxPooling2D(pool_size=(2, 2)))                   #14,14,32
model.add(Dropout(0.25))
model.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) #14,14,64
model.add(MaxPooling2D(pool_size=(2, 2)))                   #7,7,64
model.add(Dropout(0.25))

model.add(Flatten())                                        #3136 = 7*7*64
model.add(Dense(128, activation='relu'))                    #128
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))         #10
```

4.2 Mnist digit classifier with deep Convolution layers

```
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras.utils import to_categorical as ohe

batch_size = 128; num_classes = 10; epochs = 12

# input image dimensions
img_rows, img_cols = 28, 28
input_shape=(img_rows, img_cols,1)

# loead mnist image and train and test datasets
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols,
                1).astype(float)/255
x_test  = x_test.reshape(x_test.shape[0],   img_rows, img_cols,
                1).astype(float)/255
print('x_train shape:{}  y_train shape:{} ', x_train.shape,y_train.shape)
print('x_test shape :{}  y_test shape :{} ', x_test.shape,y_test.shape)

# convert label  to one_hot_encoding(label,10)
y_train = ohe(y_train, num_classes)
y_test  = ohe(y_test, num_classes)
```

```
model = Sequential()
#convolution layer
model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
                input_shape=input_shape))             #26,26,32
model.add(MaxPooling2D(pool_size=(2, 2)))             #14,14,32
model.add(Dropout(0.25))

model.add(Conv2D(64, kernel_size=(3, 3), activation='relu')) #14,14,64
model.add(MaxPooling2D(pool_size=(2, 2)))             #7,7,64
model.add(Dropout(0.25))

model.add(Flatten())                                  #3136 = 7*7*64
model.add(Dense(128, activation='relu'))              #128
model.add(Dropout(0.25))
model.add(Dense(num_classes, activation='softmax')) #10

model.compile(
                loss='categorical_crossentropy',
                optimizer='adam',
                metrics=['accuracy'])

model.fit(x_train, y_train, validation_data=(x_test, y_test),
                batch_size=batch_size, epochs=epochs, verbose=1)

score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

4.2 Mnist digit classifier with deep Convolution layers

```
from keras.data
from keras.mod
from keras.laye
from keras.laye
from keras.utils

batch_size = 1

# input image
img_rows, img_
input_shape=(i

# loead mnist
(x_train, y_train
x_train = x_trai
            1)
x_test  = x_test
            1)
print('x_train sh
print('x_test sha

# convert labe
y_train = ohe(y
y_test  = ohe(y_test, num_classes)
```

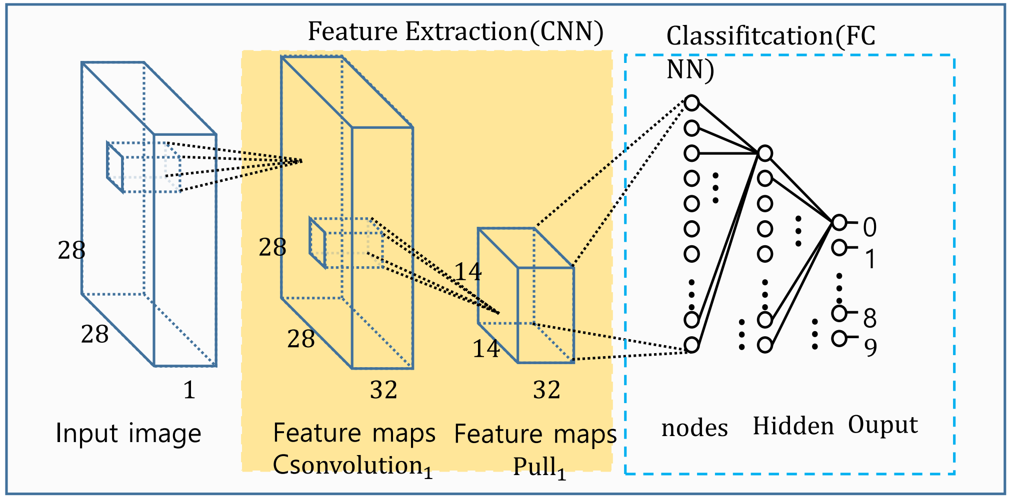
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python36_64\python.exe

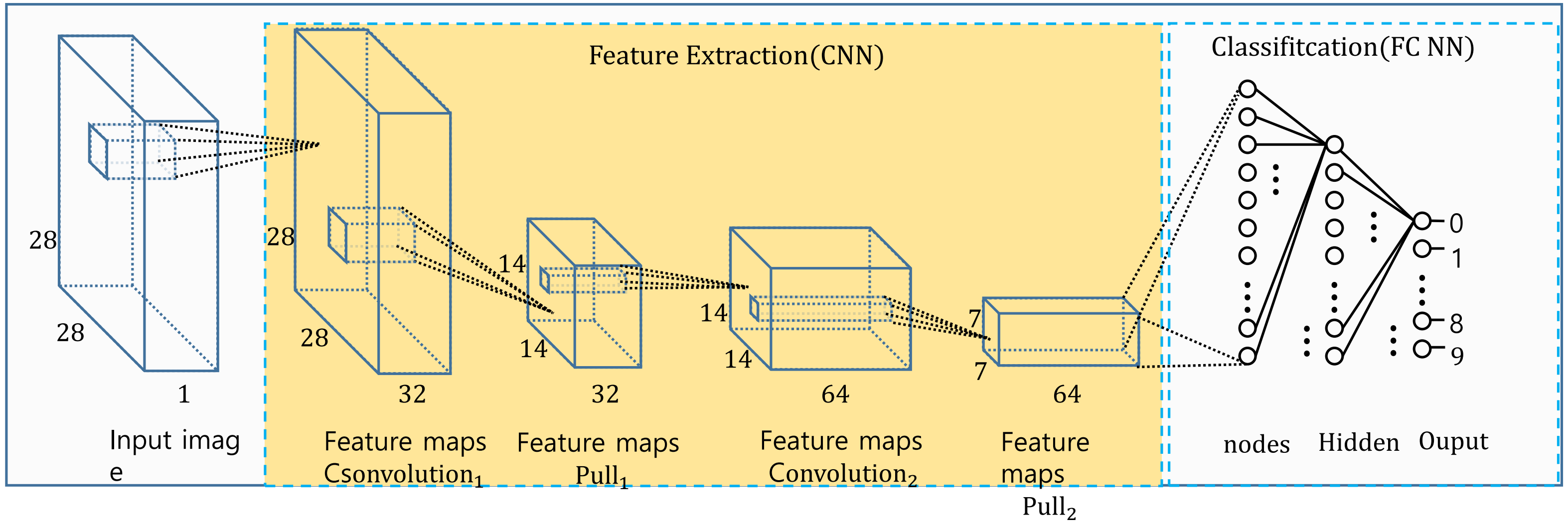
```
Using TensorFlow backend.
x_train shape:{}  y_train shape:{}  (60000, 28, 28, 1) (60000,)
x_test shape :{}  y_test shape :{}  (10000, 28, 28, 1) (10000,)
Train on 60000 samples, validate on 10000 samples
Epoch 1/12
60000/60000 [==============================] - 10s 166us/step - loss: 0.2960 - acc: 0.9077 - val_loss: 0.0724 - val_acc: 0.9766
Epoch 2/12
60000/60000 [==============================] - 5s 87us/step - loss: 0.0897 - acc: 0.9727 - val_loss: 0.0442 - val_acc: 0.9859
Epoch 3/12
60000/60000 [==============================] - 5s 88us/step - loss: 0.0655 - acc: 0.9799 - val_loss: 0.0331 - val_acc: 0.9882
Epoch 4/12
60000/60000 [==============================] - 5s 88us/step - loss: 0.0526 - acc: 0.9835 - val_loss: 0.0307 - val_acc: 0.9896
Epoch 5/12
60000/60000 [==============================] - 5s 89us/step - loss: 0.0474 - acc: 0.9855 - val_loss: 0.0278 - val_acc: 0.9906
Epoch 6/12
60000/60000 [==============================] - 5s 88us/step - loss: 0.0409 - acc: 0.9871 - val_loss: 0.0245 - val_acc: 0.9912
Epoch 7/12
60000/60000 [==============================] - 5s 90us/step - loss: 0.0359 - acc: 0.9886 - val_loss: 0.0249 - val_acc: 0.9916
Epoch 8/12
60000/60000 [==============================] - 5s 89us/step - loss: 0.0340 - acc: 0.9889 - val_loss: 0.0250 - val_acc: 0.9912
Epoch 9/12
60000/60000 [==============================] - 5s 91us/step - loss: 0.0299 - acc: 0.9904 - val_loss: 0.0235 - val_acc: 0.9917
Epoch 10/12
60000/60000 [==============================] - 5s 79us/step - loss: 0.0271 - acc: 0.9914 - val_loss: 0.0232 - val_acc: 0.9924
Epoch 11/12
60000/60000 [==============================] - 5s 91us/step - loss: 0.0254 - acc: 0.9915 - val_loss: 0.0207 - val_acc: 0.9932
Epoch 12/12
60000/60000 [==============================] - 5s 87us/step - loss: 0.0255 - acc: 0.9917 - val_loss: 0.0203 - val_acc: 0.9929
Test loss: 0.020317405389965288
Test accuracy: 0.9929
```

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
```

```
Test loss: 0.03572413484162436
Test accuracy: 0.988
```

4.2 Mnist digit classifier with deep Convolution layers

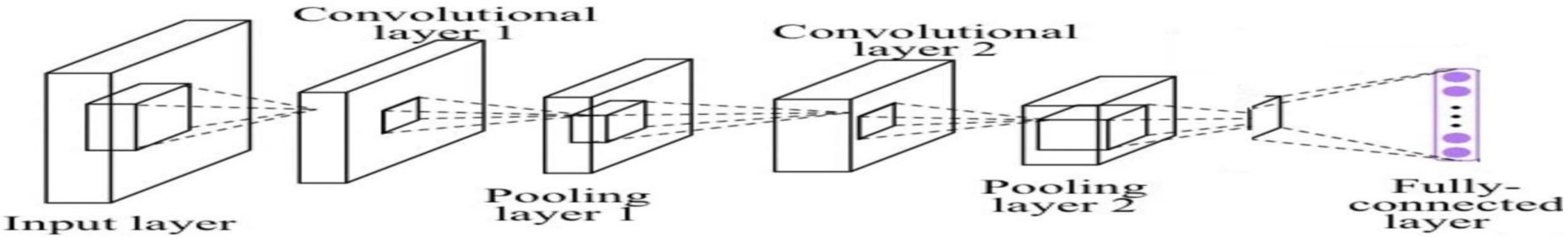





```
Test loss: 0.020317405389965528
Test accuracy: 0.9929
```

4.3 exercise

- 다음 CNN구조로 minist image 인식 시스템을 구현화여 99.3%이상의 인식률을 얻을 수 있음을 확인 하시오



784: 28x28x1 28x28x32 14x14x32 14x14x64 7x7x64 : 3136

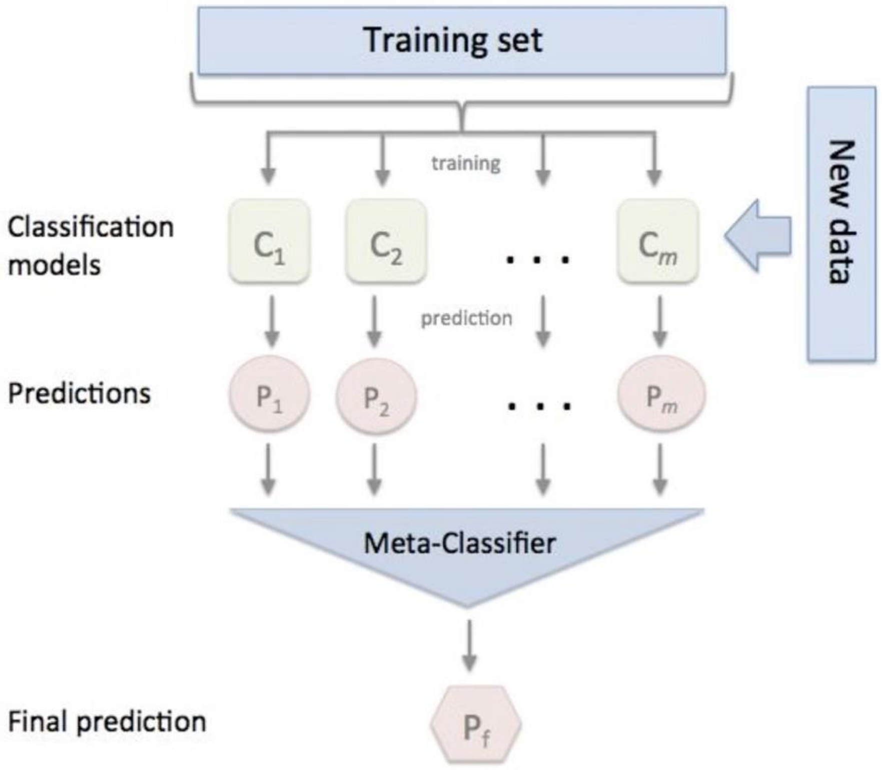
```
Epoch 12/12
60000/60000 [========================
Test loss: 0.02079462225716561
Test accuracy: 0.9935
Press any key to continue . . .
```

4.4 CIFAR-10

- [ConvNetJS demo: training on CIFAR-10]

  http://cs.Stanford.edu/people/karpathy/convnetjs/demo/cifar10.html

4.4 Ensemble



http://rasbt.github.io/mlxtend/user_guide/classifier/StackingClassifier/

$P_1 = C_1.predict(y_{train})$
$P_2 = C_2.predict(y_{train})$
:
$P_n = C_n.predict(y_{train})$

5.3 Ensemble

Ensemble prediction

	0	1	2	3	4	5	6	7	8	9
C_1 →	0.1	0.01	0.02	0.8	...	...	...	...	...	...
C_2 →	0.01	0.5	0.02	0.4	...	...	...	...	...	...
. . .										
C_m →	0.01	0.01	0.1	0.7	...	...	...	...	...	...
Sum	0.12	0.52	0.14	**1.9**	...	...	...	...	...	...

argmax

```
import numpy as np

predictions=np.zeros(10,dtype=float)
for i, model in enumerate(models):
        acc=model.evaluate(X_train,Y_train_ohe)
        print(' model[{}] acc:{}'.format(i,acc))
        p=mode.predict(X_train) #p=[0.1, 0.3, 0.2, .₩0.5,,,]
        predictions =predictions+p  #[0..9] =[0..9]+[0..9]

ensenble_predicts=np.equal(np.argmax(predictions,1),
        np.argmax(Y_train_ohe,1))
ensenble_accuracy=ensenble_predicts.mean()
print('Ensenble accuracy : ',ensenble_accuracy)
```

The End