Deep Learning #3

15 Jan. 2021

자율주행시스템 개발팀
신 주 석

◆ Problem of the Neural Network

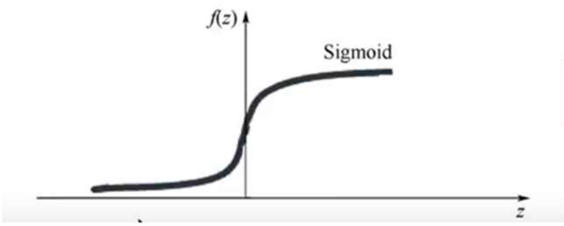
1) 여러 개의 Perceptron으로 특정 문제(e.g. XOR)를 해결할 수 있지만, 해당 파라미터(Weight, Bias)를 학습 시킬 수 있는 방법을 찾을 수 없었음.

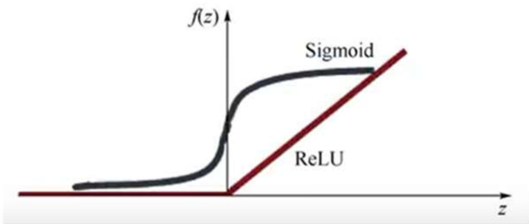
   ➔ **Backpropagation(Chain-Rule)을 이용하여 학습 가능**

2) Vanish Gradient problem (Chain-Rule, Sigmoid(0~1))

   → **ReLu (Rectified Linear Unit)을 이용하여 해결**

Sigmoid!





3) Initialized the weight의 설정

- Not all 0's
- Challenging issue
- Hinton et al. (2006) "A Fast Learning Algorithm for Deep Belief Nets"
  - Restricted Boatman Machine (RBM)

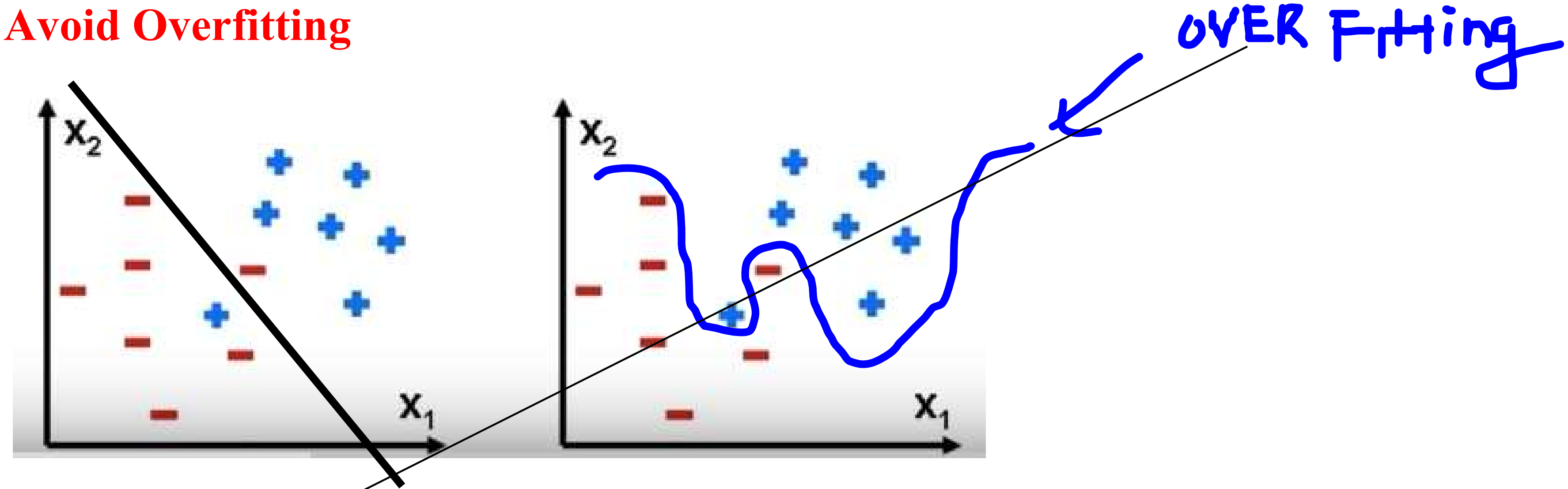
  restricted boltzmann machine

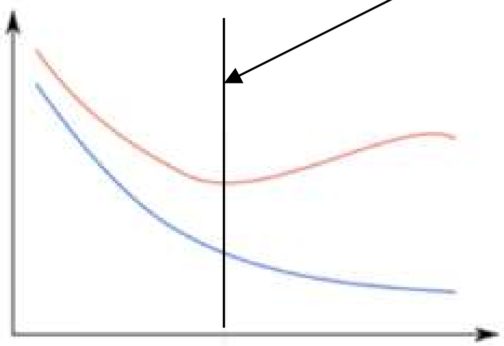
Xavier initialize Method

4) Small labeled dataset, Computing Power

   → **Big Dataset, GPU, etc.**

- **Deep Neural Network**
  - **Drop out**
    - » **Avoid Overfitting**





- Very high accuracy on the training dataset (eg: 0.99)
- Poor accuracy on the test data set (0.85)

Solution for Overfitting

1. More training Data
2. **Regularization**
   - Not have too big numbers in the weight

$$\text{cost} + \lambda \sum W^2$$

```
l2reg = 0.001 * tf.reduce_sum(tf.square(W))
```

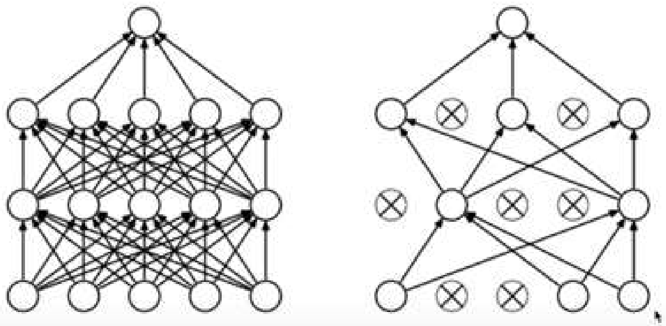
Deep Neural Network

- Dropout
  - **Avoid Overfitting**

Dropout: A Simple Way to Prevent Neural Networks from Overfitting [Srivastava et al. 2014]

Regularization: Dropout

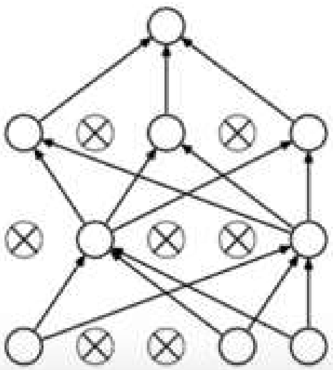
"randomly set some neurons to zero in the forward pass"



(a) Standard Neural Net (b) After applying dropout. *[Srivastava et al., 2014]*

```
dropout_rate = tf.placeholder("float")
_L1 = tf.nn.relu(tf.add(tf.matmul(X, W1), B1))
L1 = tf.nn.dropout(_L1, dropout_rate)
```

```
TRAIN:
 sess.run(optimizer, feed_dict={X: batch_xs, Y: batch_ys,
 dropout_rate: 0.7})

EVALUATION:
 print "Accuracy:", accuracy.eval({X: mnist.test.images, Y:
 mnist.test.labels, dropout_rate: 1})
```



Forces the network to have a redundant representation.

Deep Neural Network

– MNIST Deep NN & ReLu & Xavier Initialization & Dropout

```
W1 = tf.get_variable("W1", shape=[784, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b1 = tf.Variable(tf.random_normal([512]))
L1 = tf.nn.relu(tf.matmul(X, W1) + b1)
L1 = tf.nn.dropout(L1, keep_prob=keep_prob)

W2 = tf.get_variable("W2", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b2 = tf.Variable(tf.random_normal([512]))
L2 = tf.nn.relu(tf.matmul(L1, W2) + b2)
L2 = tf.nn.dropout(L2, keep_prob=keep_prob)

W3 = tf.get_variable("W3", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b3 = tf.Variable(tf.random_normal([512]))
L3 = tf.nn.relu(tf.matmul(L2, W3) + b3)
L3 = tf.nn.dropout(L3, keep_prob=keep_prob)

W4 = tf.get_variable("W4", shape=[512, 512],
                     initializer=tf.contrib.layers.xavier_initializer())
b4 = tf.Variable(tf.random_normal([512]))
L4 = tf.nn.relu(tf.matmul(L3, W4) + b4)
L4 = tf.nn.dropout(L4, keep_prob=keep_prob)

W5 = tf.get_variable("W5", shape=[512, 10],
                     initializer=tf.contrib.layers.xavier_initializer())
b5 = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L4, W5) + b5
```

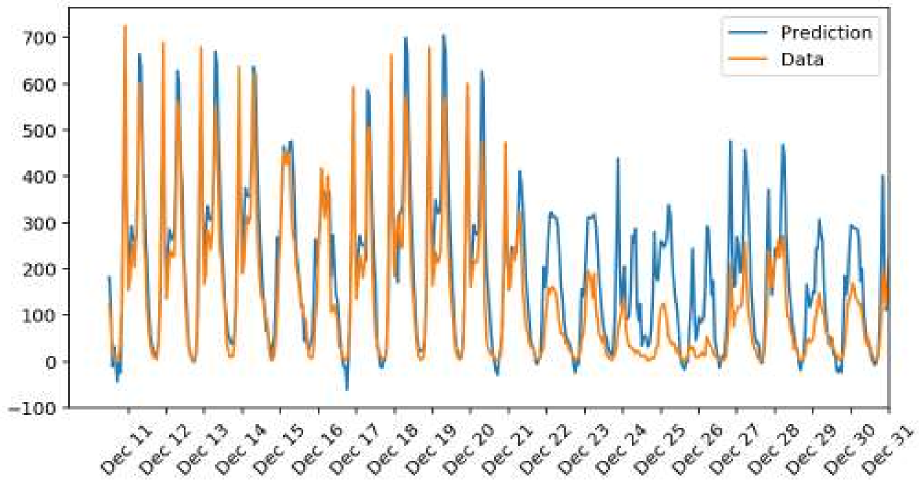
```
Epoch: 0001 cost = 0.314719560
Epoch: 0002 cost = 0.136928339
Epoch: 0003 cost = 0.114664485
Epoch: 0004 cost = 0.105699805
Epoch: 0005 cost = 0.093989542
Epoch: 0006 cost = 0.091876498
Epoch: 0007 cost = 0.079226325
Epoch: 0008 cost = 0.076840967
Epoch: 0009 cost = 0.074611597
Epoch: 0010 cost = 0.064871844
Epoch: 0011 cost = 0.064510580
Epoch: 0012 cost = 0.072228441
Epoch: 0013 cost = 0.067988544
Epoch: 0014 cost = 0.050129582
Epoch: 0015 cost = 0.062997886
Epoch: 0016 cost = 0.054448708
Training Done!!!!
Accuracy:  0.9769
Label:  [2]
Prediction:  [2]
```

```
Epoch: 0001 cost = 0.477981920
Epoch: 0002 cost = 0.174209262
Epoch: 0003 cost = 0.133356226
Epoch: 0004 cost = 0.107630954
Epoch: 0005 cost = 0.097190227
Epoch: 0006 cost = 0.082390315
Epoch: 0007 cost = 0.075062841
Epoch: 0008 cost = 0.069322145
Epoch: 0009 cost = 0.064841361
Epoch: 0010 cost = 0.058239024
Epoch: 0011 cost = 0.056864930
Epoch: 0012 cost = 0.054271612
Epoch: 0013 cost = 0.052183560
Epoch: 0014 cost = 0.046791719
Epoch: 0015 cost = 0.044676678
Learning Finished!
Accuracy: 0.9818
```

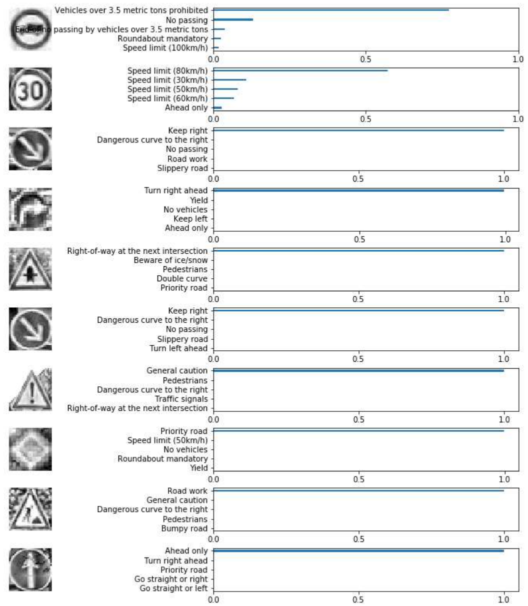
```
for i in range(total_batch):
    batch_xs, batch_ys = mnist.train.next_batch(batch_size)
    feed_dict = {X: batch_xs, Y: batch_ys, keep_prob: 0.7}
    c, _ = sess.run([cost, optimizer], feed_dict=feed_dict)
    avg_cost += c / total_batch
```

```
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, keep_prob: 1}))
```

기존 Data를 학습하여 자전거 대여 대수 예측하기 (using CNN)



Traffic Sign Recognition using CNN



NEXT

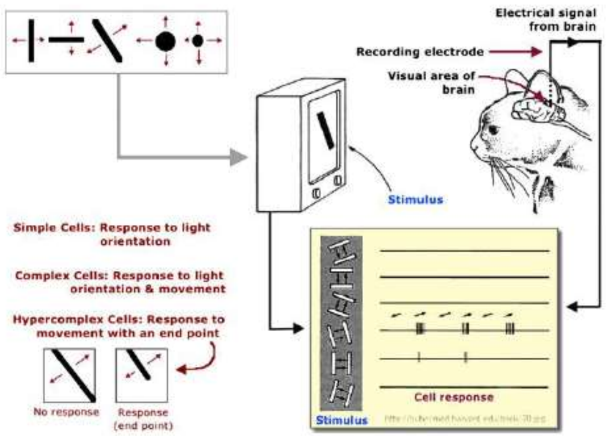
Convolution Neural Network
Recurrent Neural Network

Steering value prediction using CNN

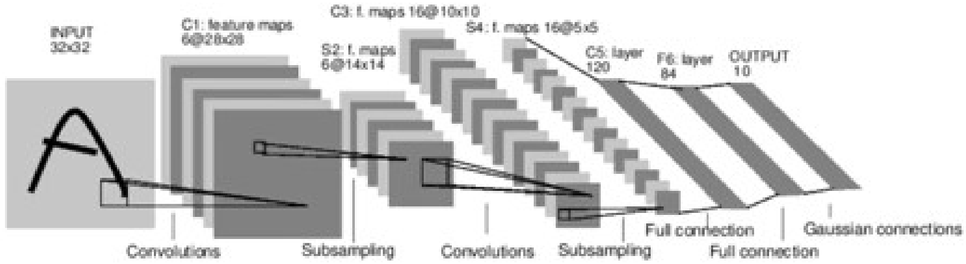
Language Translation using RNN

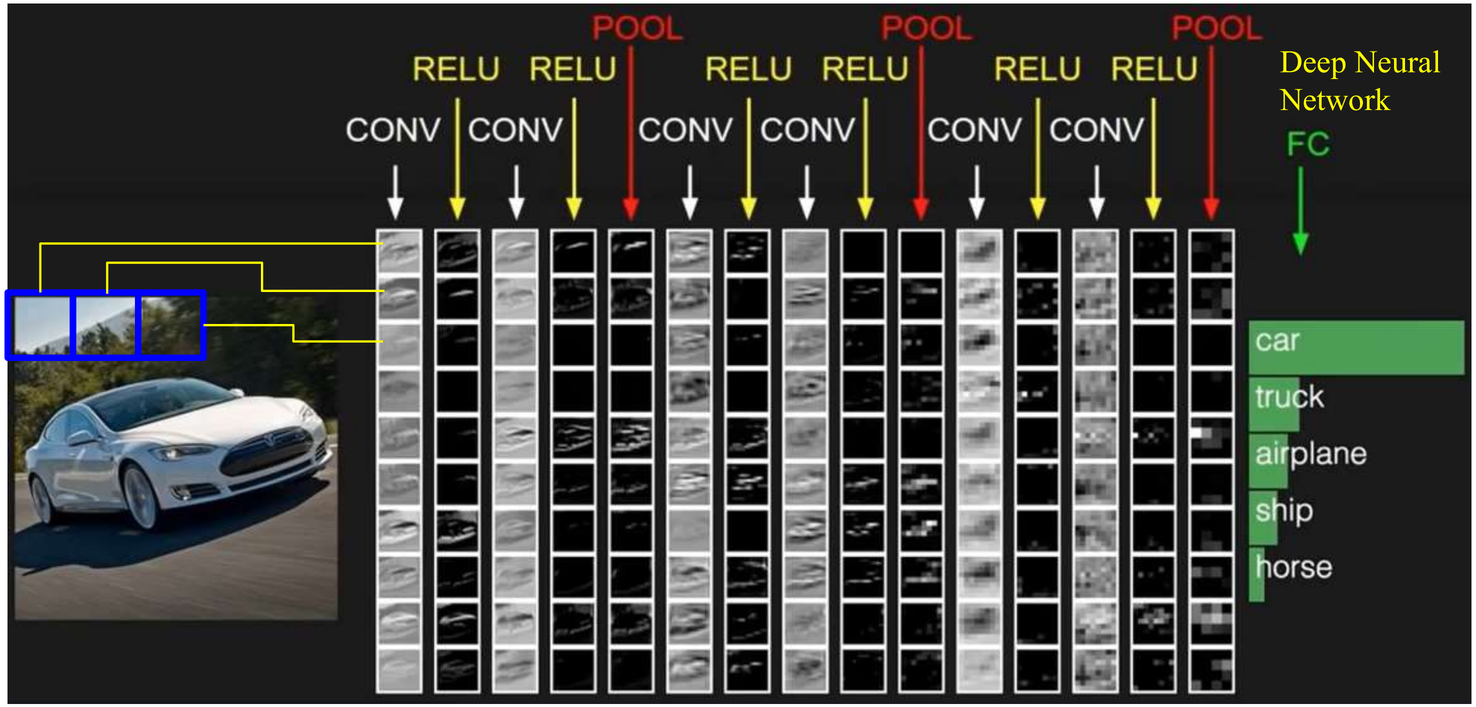
```
Input
  Word Ids:      [208, 68, 203, 32, 9, 95, 129]
  English Words: ['he', 'saw', 'a', 'old', 'yellow', 'truck', '.']

Prediction
  Word Ids:      [288, 175, 27, 141, 209, 293, 10, 325, 1]
  French Words: il a vu un vieux camion jaune . <EOS>
```

◆ Convolution Neural Network



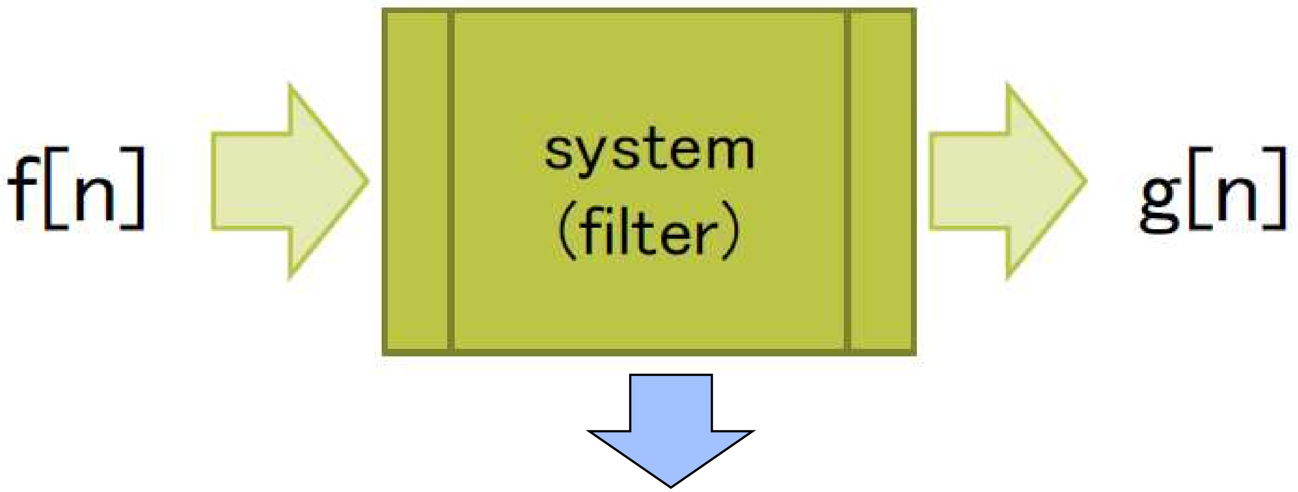
CNN: Convolutional Neural Network





SoftMax

- **Image Filtering**
  - **Filtering: 전자공학 Signal Processing, 시스템 분야로 부터 파생된 개념**
    - » Fourier 변환을 통하여 데이터를 주파수 성분으로 변경한 후, 주파수에 대하여 여러가지 가공 처리를 하기 위해 **Filtering**이란 개념이 나왔음
    - » 이미지의 경우, 입력 신호가 주파수 형태가 아니라 이미지이기 때문에 **Spatial Filtering**



System: 일련의 입력 신호를 처리하여
또 다른 일련의 출력 신호를 만들어 내는 것

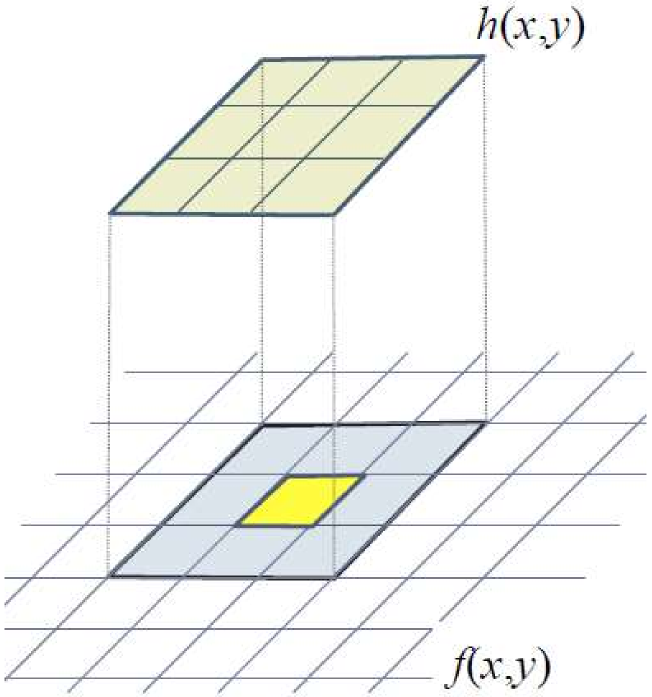
Filter: 시스템의 한 성분으로써, 신호의 일부 성분을 제거하거나
일부 특성을 변경하기 위해 설계된 시스템의 한 종류

- **Convolution**

Mask, filter, template, kernel

$$g(x,y) = h(x,y) \times f(x,y) = \sum_{s=-a}^{a} \sum_{t=-a}^{b} \boxed{h(s,t)} \times f(x+\mathbf{s}, y+\mathbf{t})$$

Kernel Size: m * n
a = (m-1)/2
b = (n-1)/2



a	b	c
d	e	f
g	h	i

$h(x,y)$

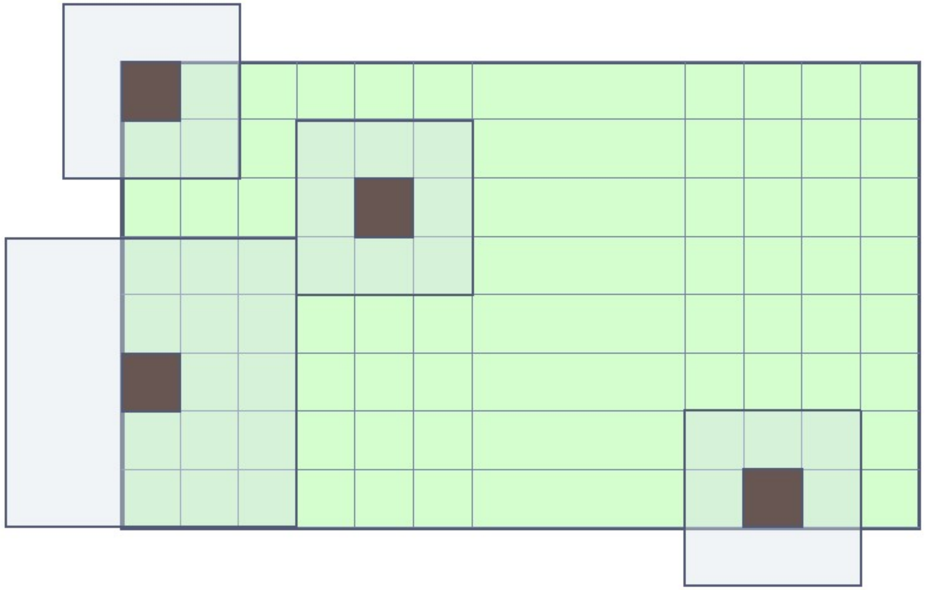
*

r	s	t
u	v	w
x	y	z

$f(x,y)$

$$g = a \cdot z + b \cdot y + c \cdot x + d \cdot w + e \cdot v + f \cdot u + g \cdot t + h \cdot s + i \cdot r$$

◆ Filtering 경계 처리



1. 특정 상수 값 삽입 (e.g. 0)
2. 경계에 있는 픽셀 값을 복사
3. 영상을 주기적인 신호로 해석하여 맞은 편 픽셀 값을 복사 (Wrap-around)
4. 모든 이웃 픽셀이 정의되는 위치에서 Convolution 연산을 시작 (출력 영상의 경계 영역의 값은 입력 영상 값을 그대로 사용하거나 특정 상수 값 사용)

Image Processing for CV: Image Filtering

- **Image Smoothing**
  - 입력영상을 조금 부드럽게 하거나 **잡음 (Noise) 을 제거**하기 위해 사용
  - **Mean, Gaussian, Median Filter**, etc.
  - **Mean Filtering**

$$\frac{1}{9}\left(v_1 + v_2 + v_3 + v_4 + v_5 + v_6 + v_7 + v_8 + v_9\right)$$

$$\frac{1}{9}\boxed{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}} \qquad \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 2 & 1 & 2 \\ 1 & 4 & 1 \\ 2 & 1 & 2 \end{bmatrix} \qquad \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Box Filtering

Image Smoothing

– Mean Filtering

10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	100	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10

Noise

Mean filtering

10	10	10	10	10	10	10	10
10	20	20	20	10	10	10	10
10	20	20	20	10	10	10	10
10	20	20	20	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10
10	10	10	10	10	10	10	10

Image Smoothing

– Mean Filtering

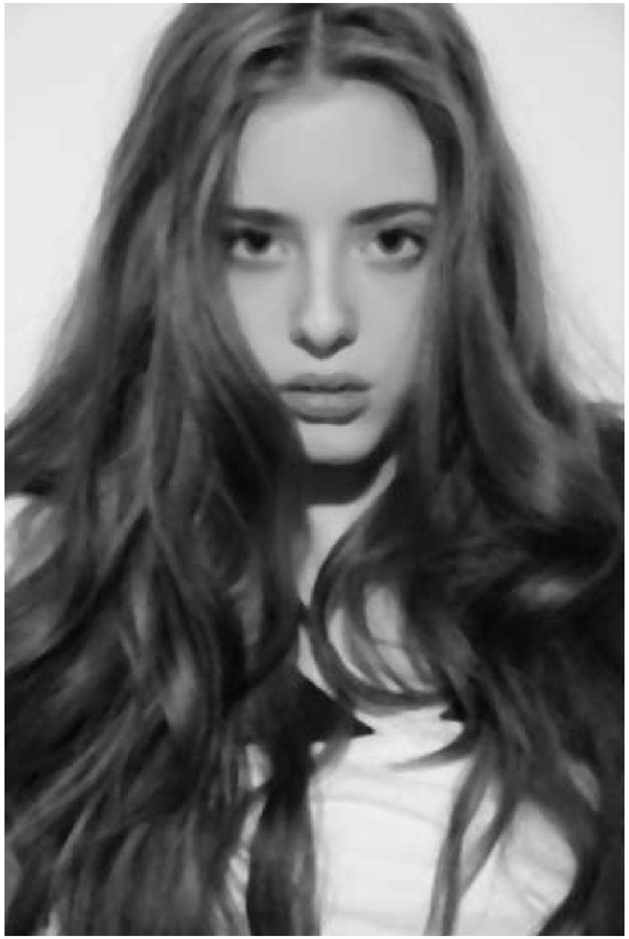
Original image

3*3 Mean filtering

5*5

7*7

- **Image Smoothing**
  - Median Filtering
    - Non-Linear Filter
    - Useful for **removing salt-pepper Noise**



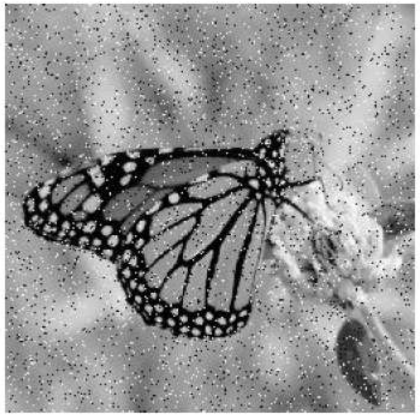
$$\begin{bmatrix} 5 & 5 & 6 \\ 3 & 4 & 5 \\ 3 & 4 & 7 \end{bmatrix}$$

$$(3,3,4,4,\mathbf{5},5,5,6,7)$$

$$\begin{bmatrix} 5 & 5 & 6 \\ 3 & \boxed{5} & 5 \\ 3 & 4 & 7 \end{bmatrix}$$

Image Smoothing

- Median Filtering
  - Non-Linear Filter
  - Useful for **removing salt-pepper Noise**



Original image

Mean Filtering

Median Filtering

◆ Image Smoothing

– Median Filtering: 실습 (MOD & Median Filter (cv::medianBlur(InputArray src, OutputArray dst, int ksize))

» Useful for **removing salt-pepper Noise**

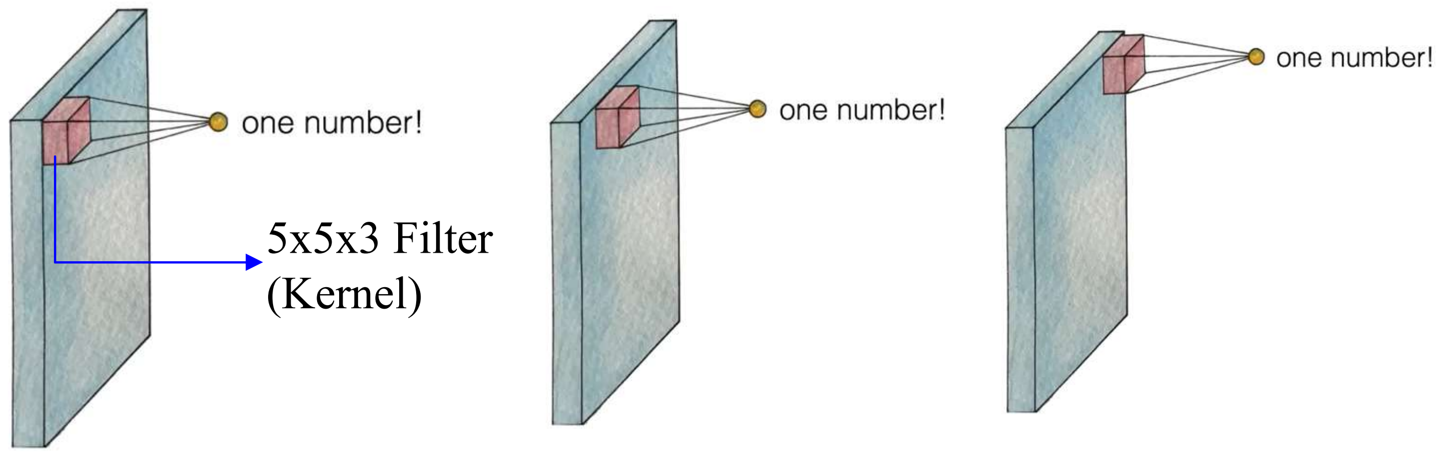

» opencv 라이브러리 사용하지 않고 구현

- Sorting Algorithm 포함

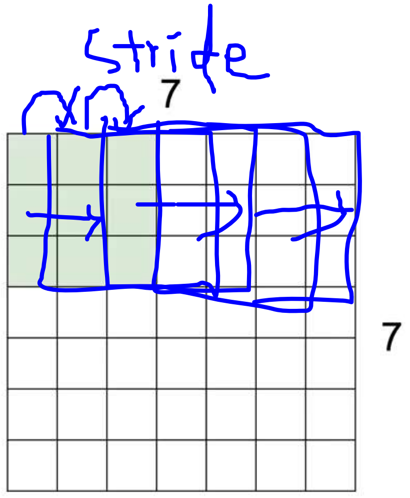
Convolution Neural Network

– **Conv Layer**

동일한 필터(w)를 가지고 이동

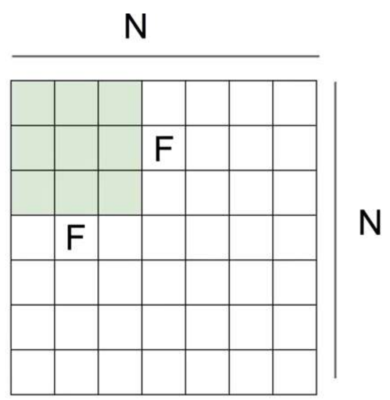


How many numbers can we get?



7x7 input (spatially)
assume 3x3 filter

⇒ 5 x 5 (out)

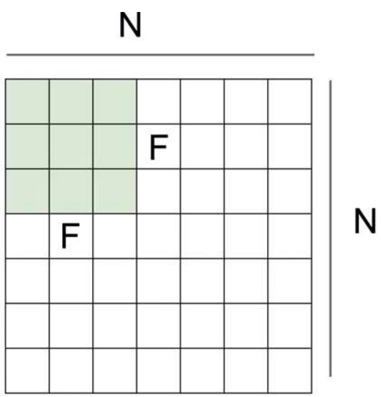


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33

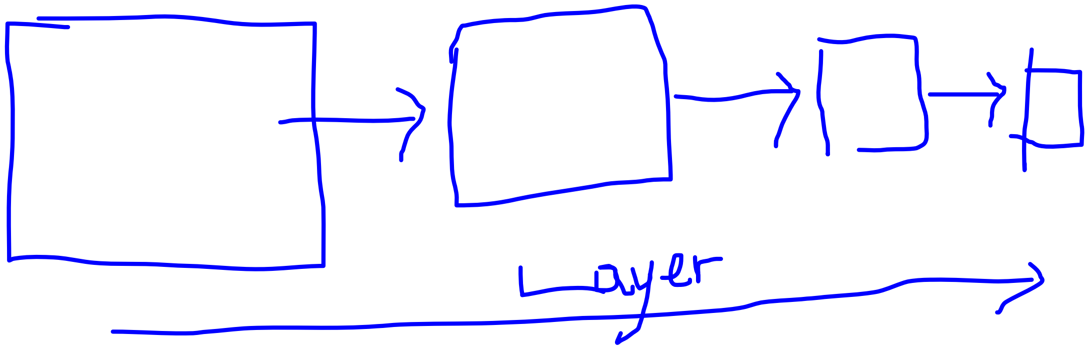
◆ Convolution Neural Network

– Conv Layer

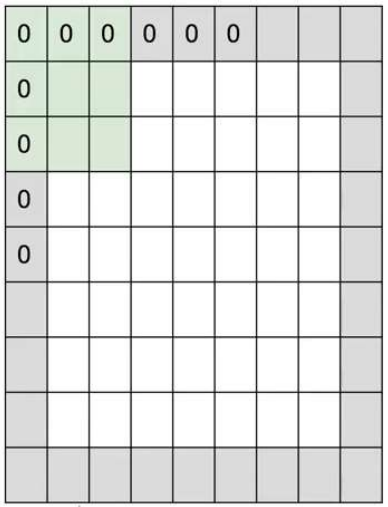


Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33



In practice: Common to zero pad the border

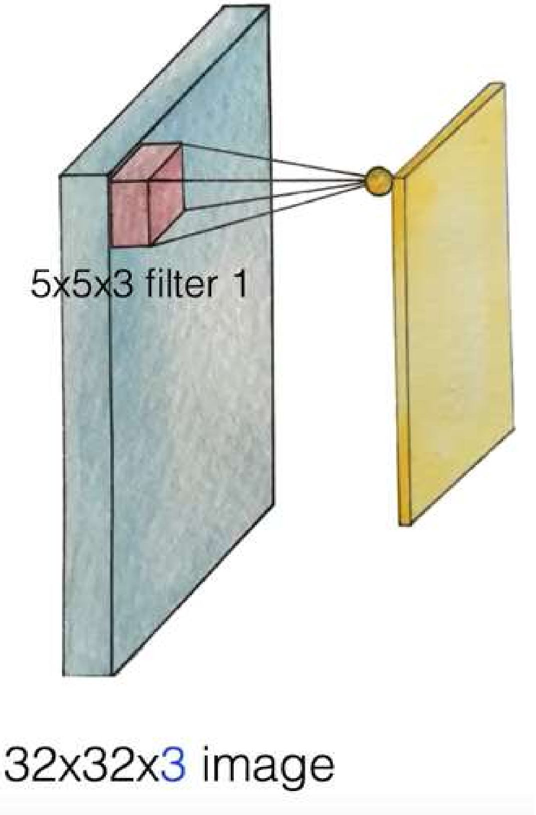


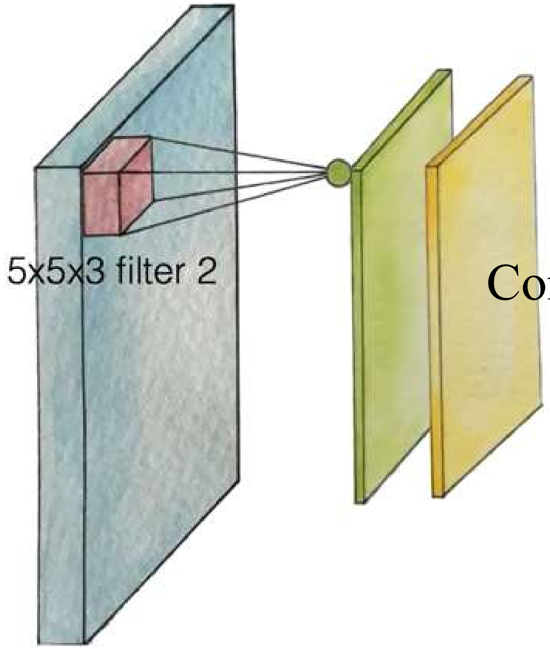
e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Convolution Neural Network

– **Conv Layer**

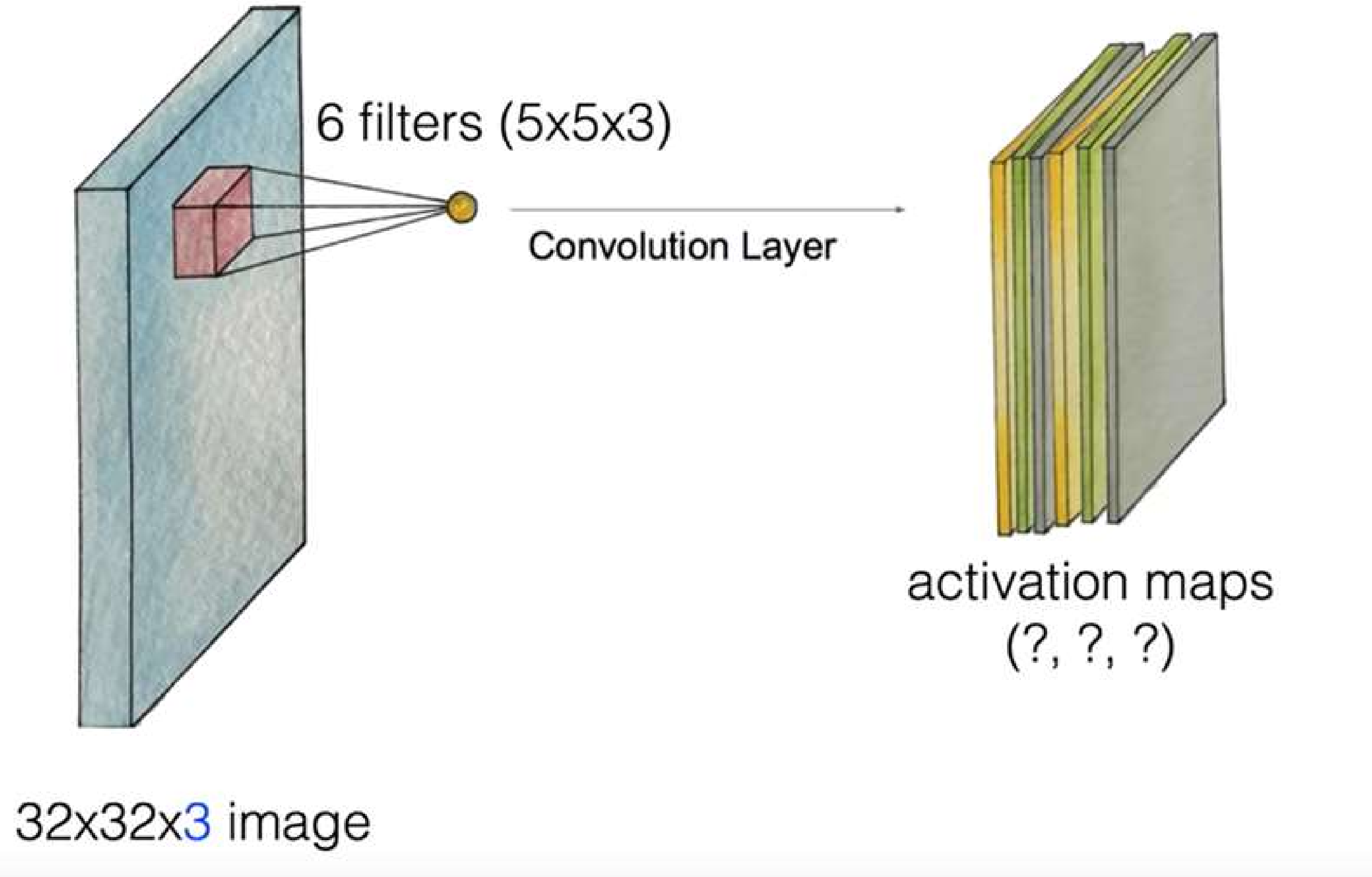




Come from filter 1

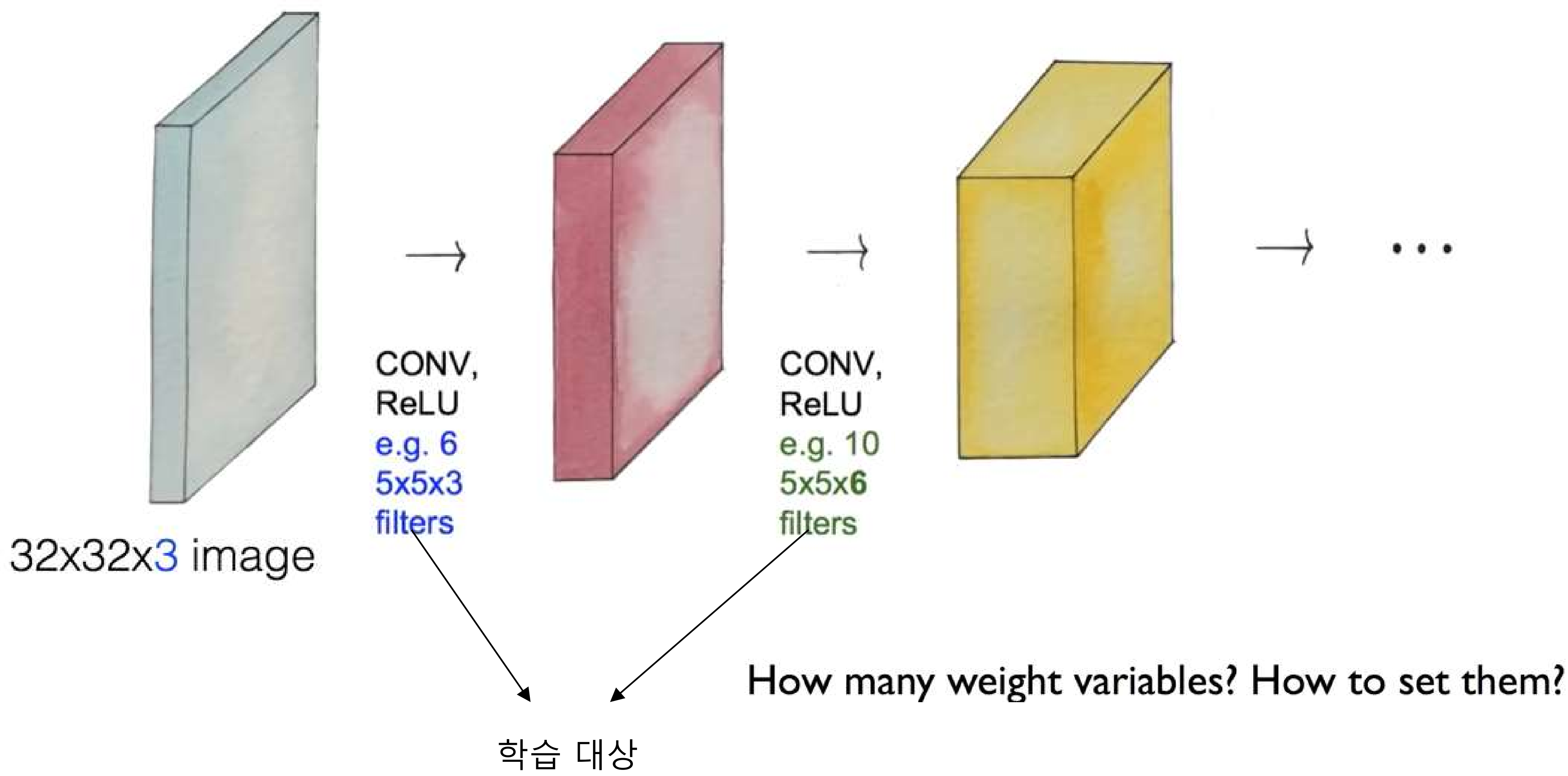
◆ Convolution Neural Network

– **Conv Layer**



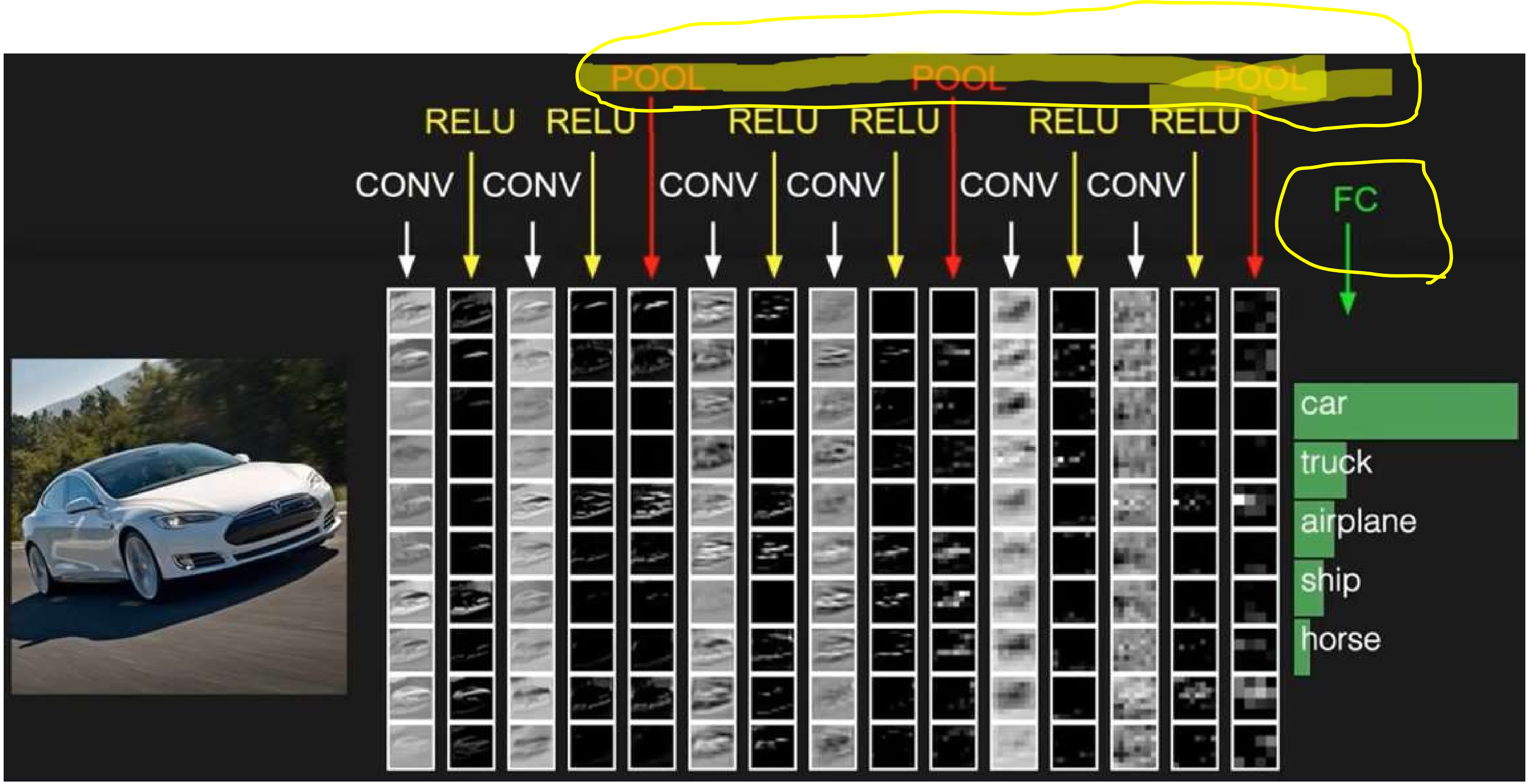
Convolution Neural Network

– **Conv Layer**



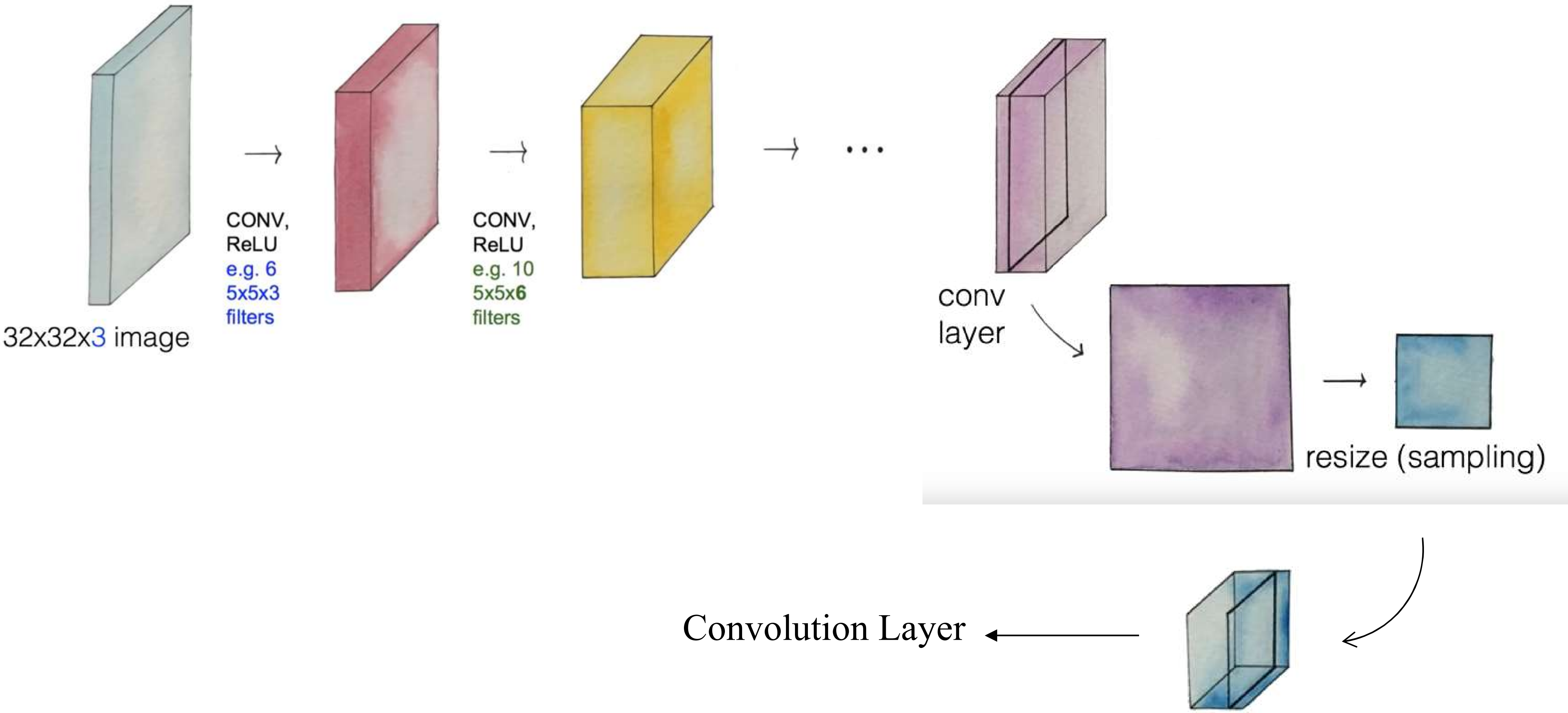
Convolution Neural Network

- Max pooling and others



Convolution Neural Network

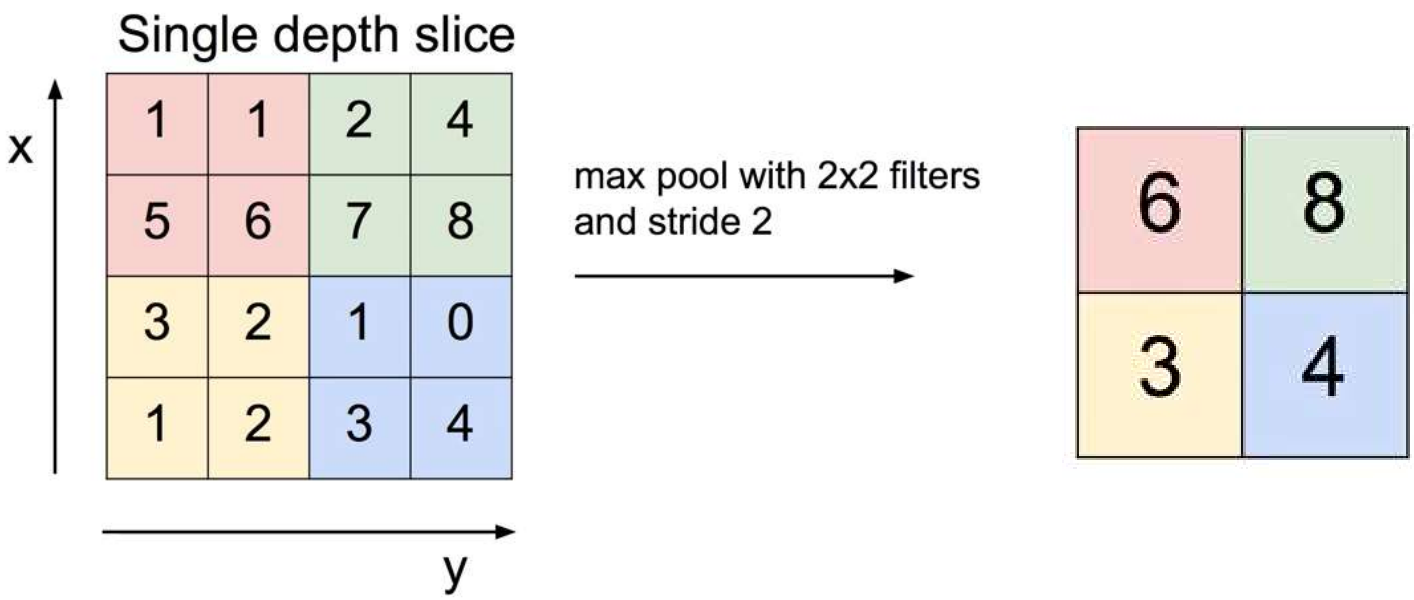
– **Pooling Layer (Sampling과 유사)**



Convolution Neural Network

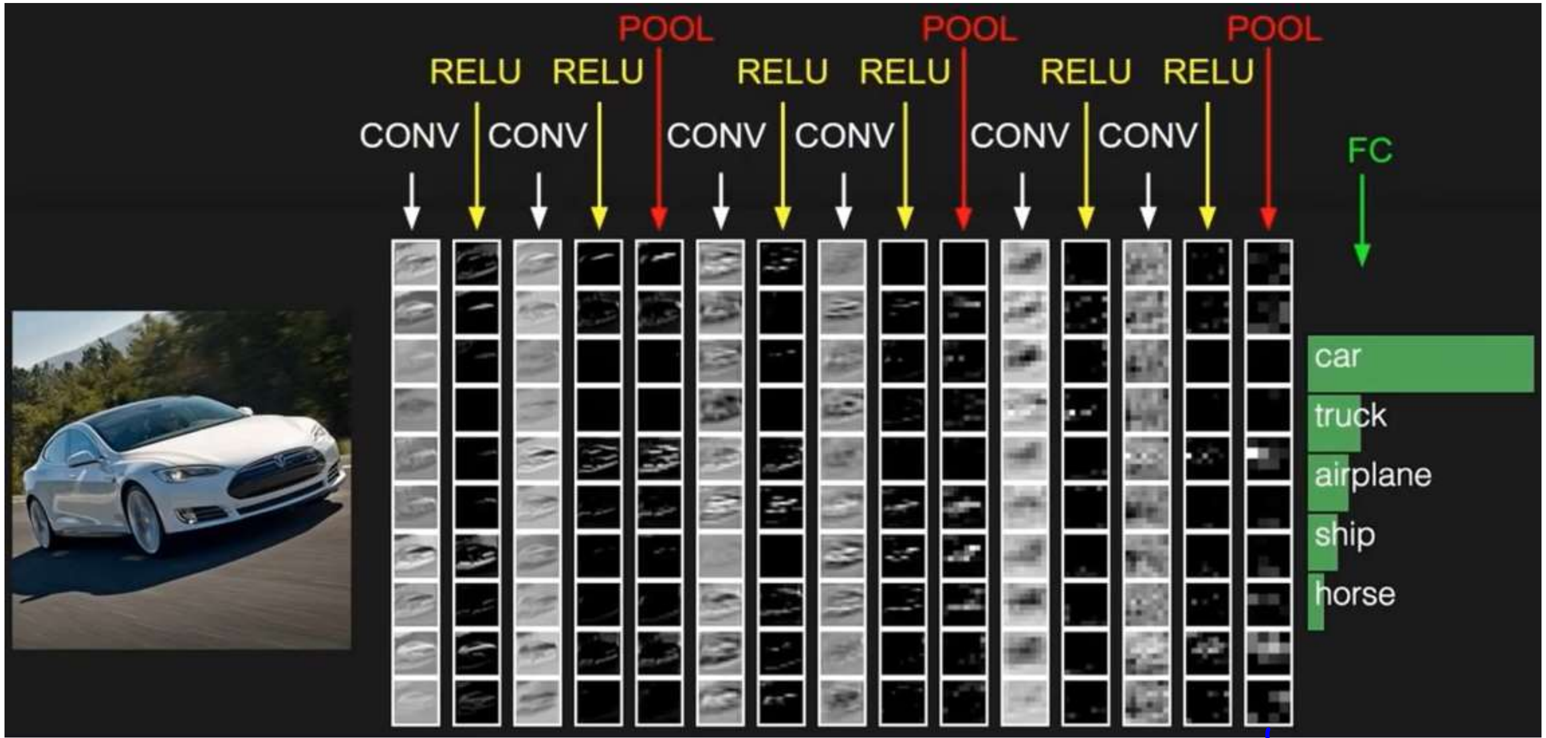
– Pooling Layer

» Max Pooling

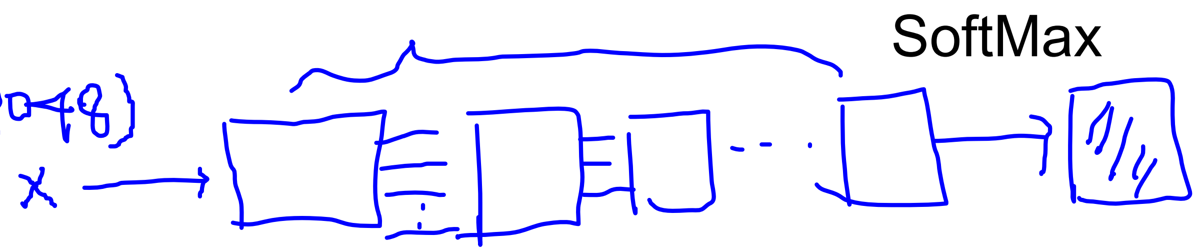


◆ Convolution Neural Network

– **FC (Fully Connected) Layer**



$4 \times 4 \times 128 \ (= 2048)$



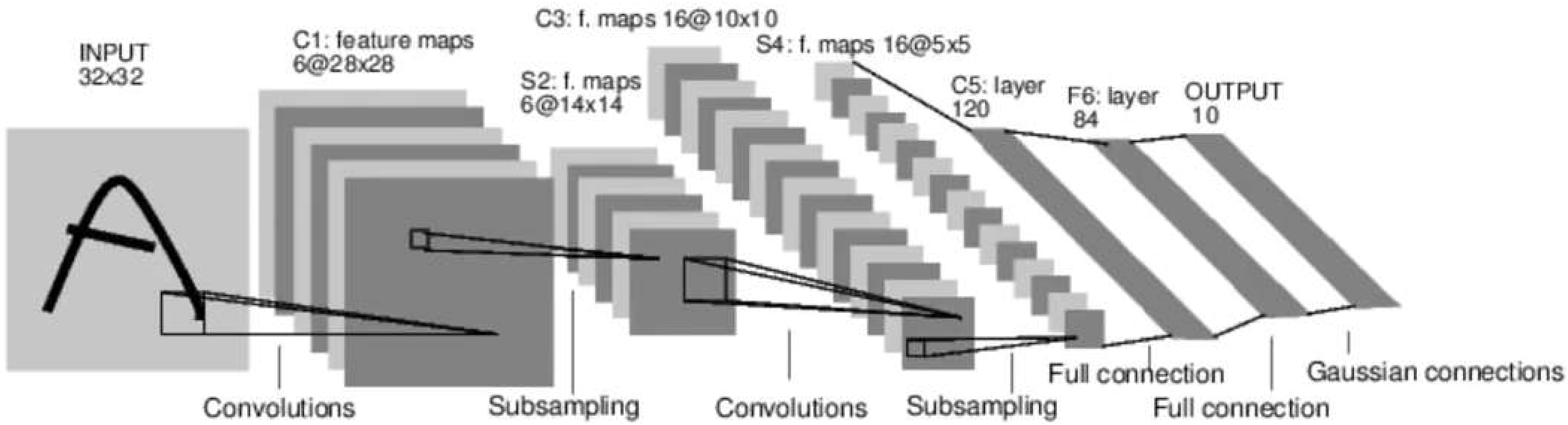
Deep Neural Network

◆ Convolution Neural Network

– **CNN Case Study**

» LeNet-5

[LeCun et al., 1998]

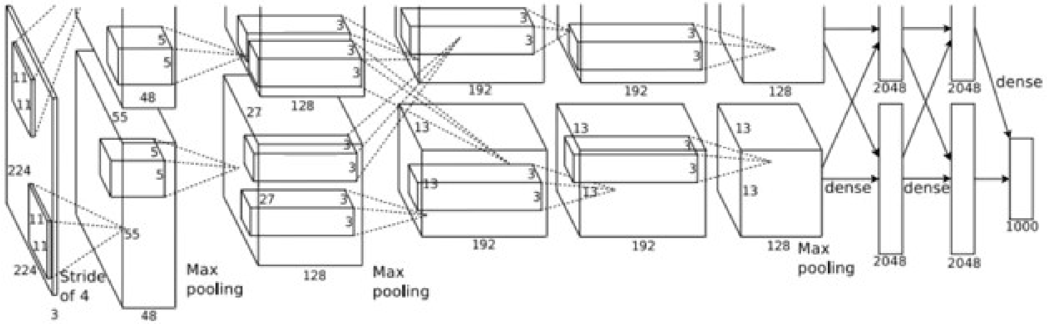


Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Convolution Neural Network

CNN Case Study

» AlexNet [Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4 x3
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

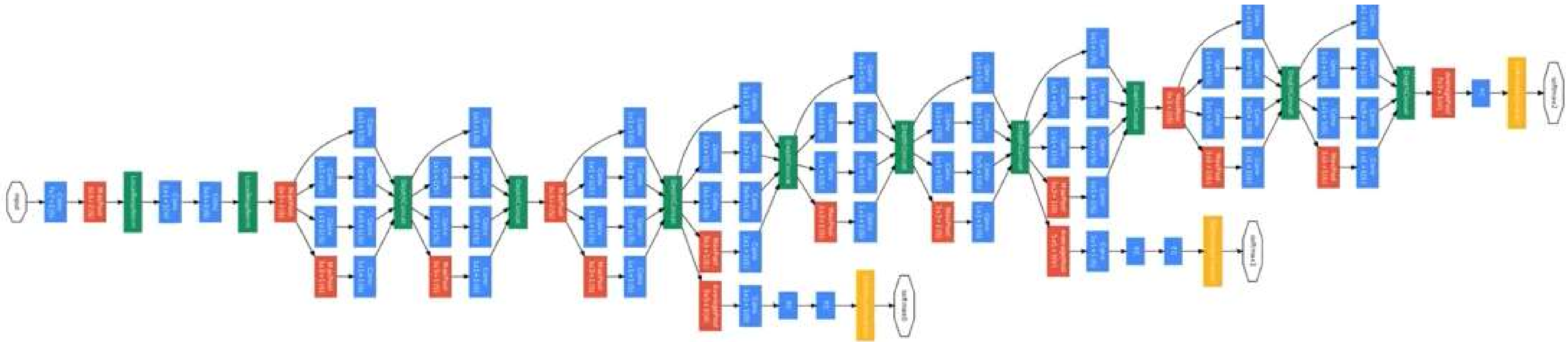
Input: 227x227x3 images
After CONV1: 55x55x96

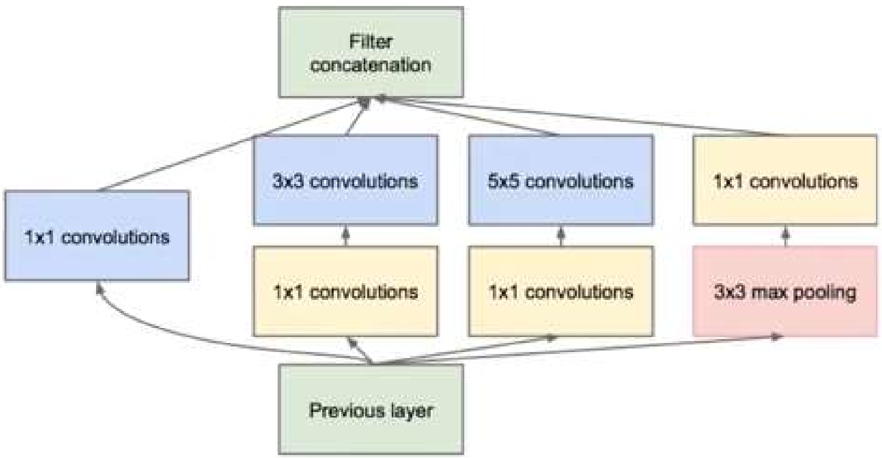
Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

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%

- **Convolution Neural Network**
  - **CNN Case Study**
    - » GoogLeNet [Szegedy et al. 2014]





Inception module

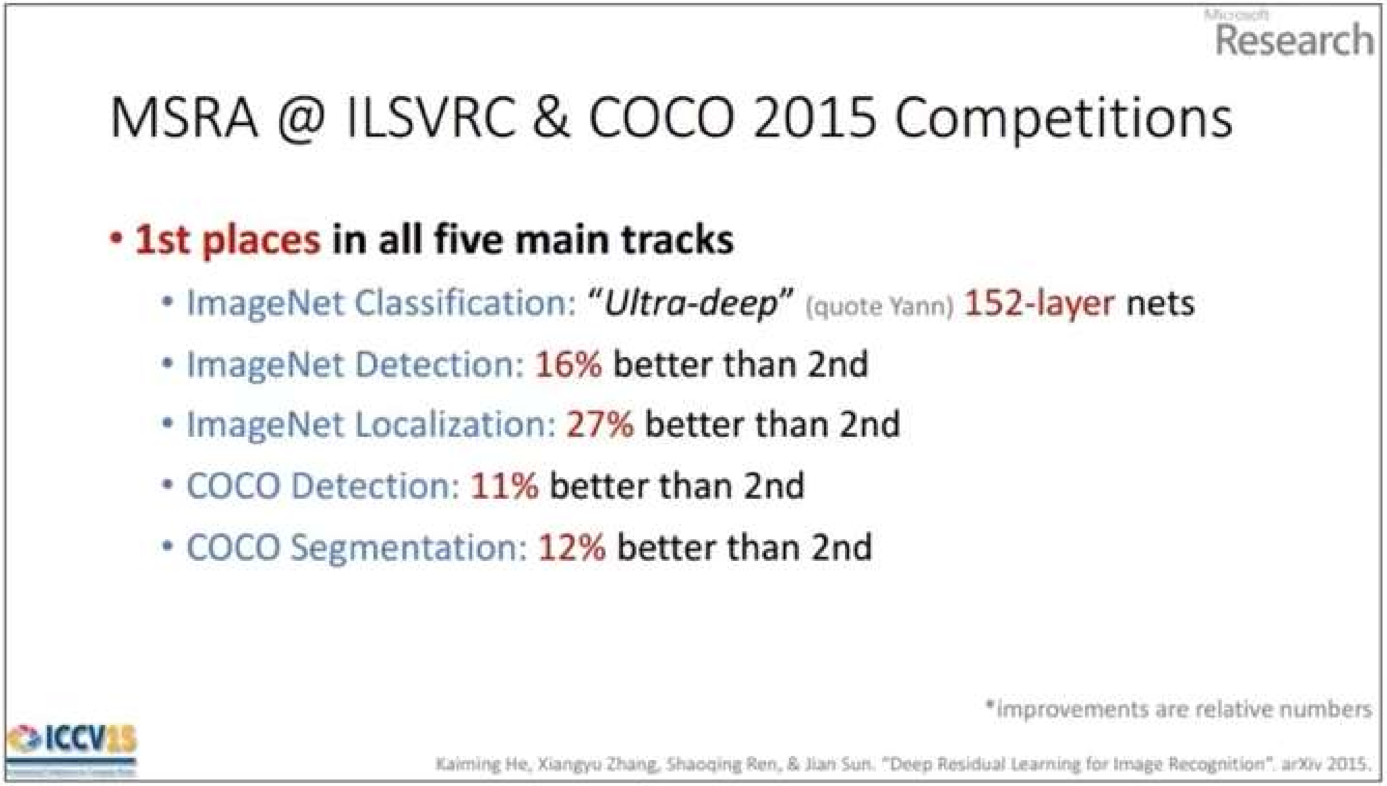
ILSVRC 2014 winner (6.7% top 5 error)

Convolution Neural Network

– CNN Case Study

» ResNet [He et al. 2015]

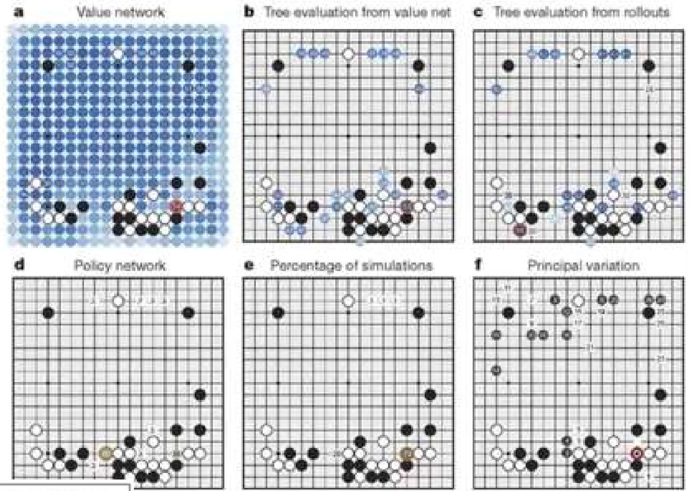
ILSVRC 2015 winner (3.6% top 5 error)



Convolution Neural Network

– CNN Case Study

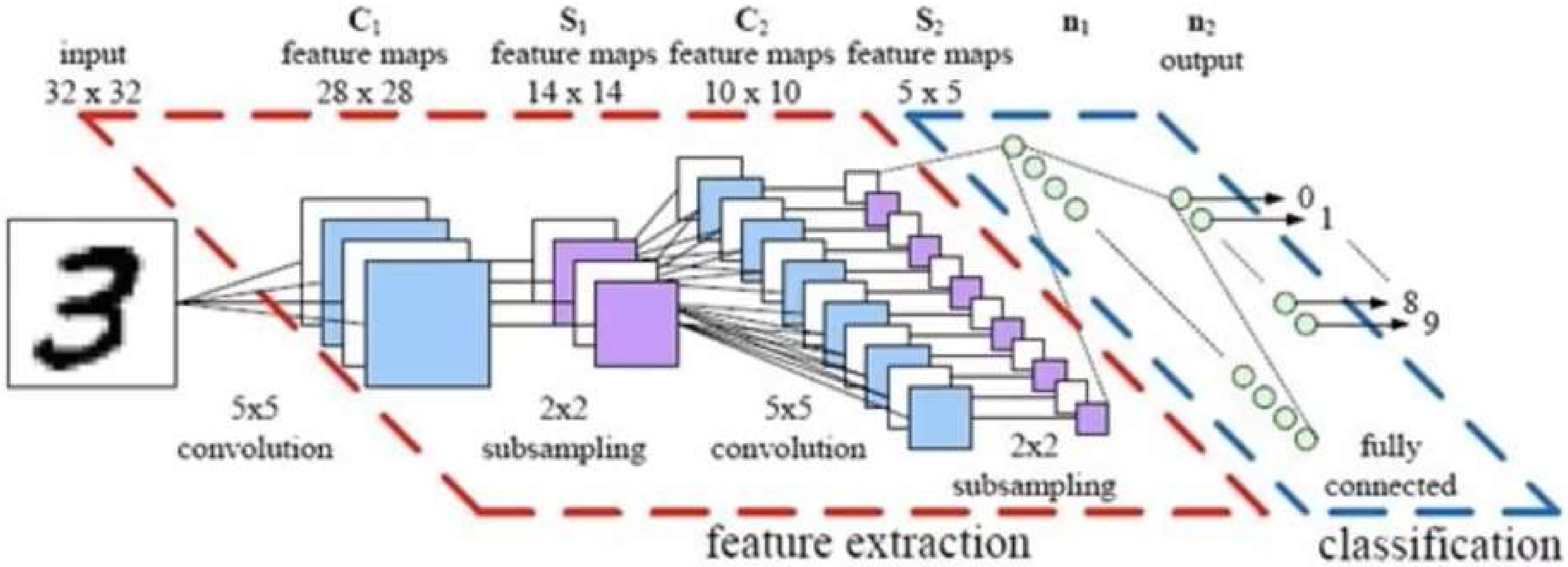
» Deep Mind's AlphaGo



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)*

- **Convolution Neural Network**
  - **MNIST 99% using CNN**

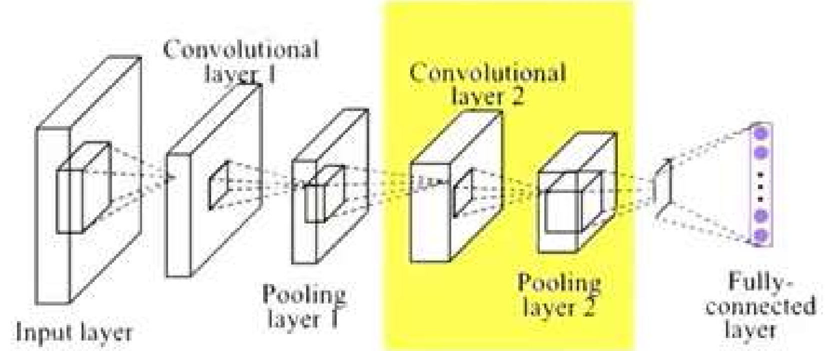


Convolution Neural Network

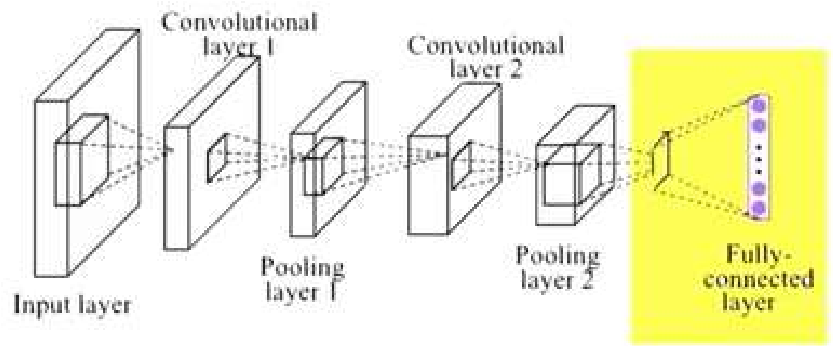
– **실습: MNIST 99% using CNN**

» Simple CNN : Layer 3

```
# L2 ImgIn shape=(?, 14, 14, 32)
W2 = tf.Variable(tf.random_normal([3, 3, 32, 64], stddev=0.01))
#    Conv      ->(?, 14, 14, 64)
#    Pool      ->(?, 7, 7, 64)
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])
```




```
L2 = tf.reshape(L2, [-1, 7 * 7 * 64])

# Final FC 7x7x64 inputs -> 10 outputs
W3 = tf.get_variable("W2", shape=[7 * 7 * 64, 10],
initializer=tf.contrib.layers.xavier_initializer())
b = tf.Variable(tf.random_normal([10]))
hypothesis = tf.matmul(L2, W3) + b
```

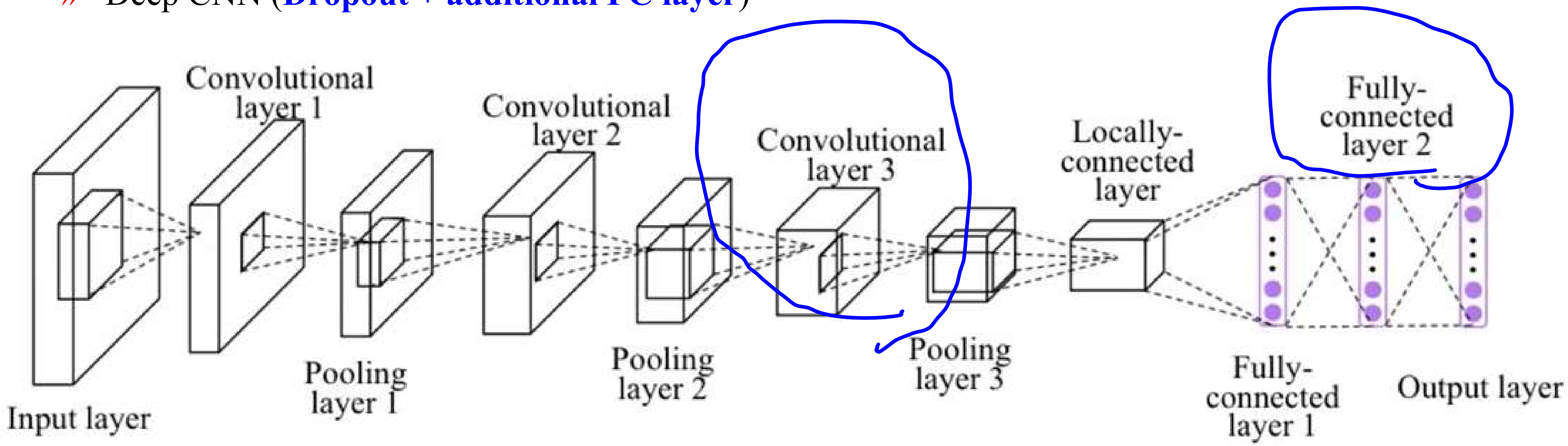



```
# define cost/loss & optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=hypothesis, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

Accuracy: 98.85%

◆ Convolution Neural Network

- 실습: **MNIST 99% using CNN**
  - » Deep CNN (**Dropout + additional FC layer**)



Accuracy: 99.38%

Convolution Neural Network

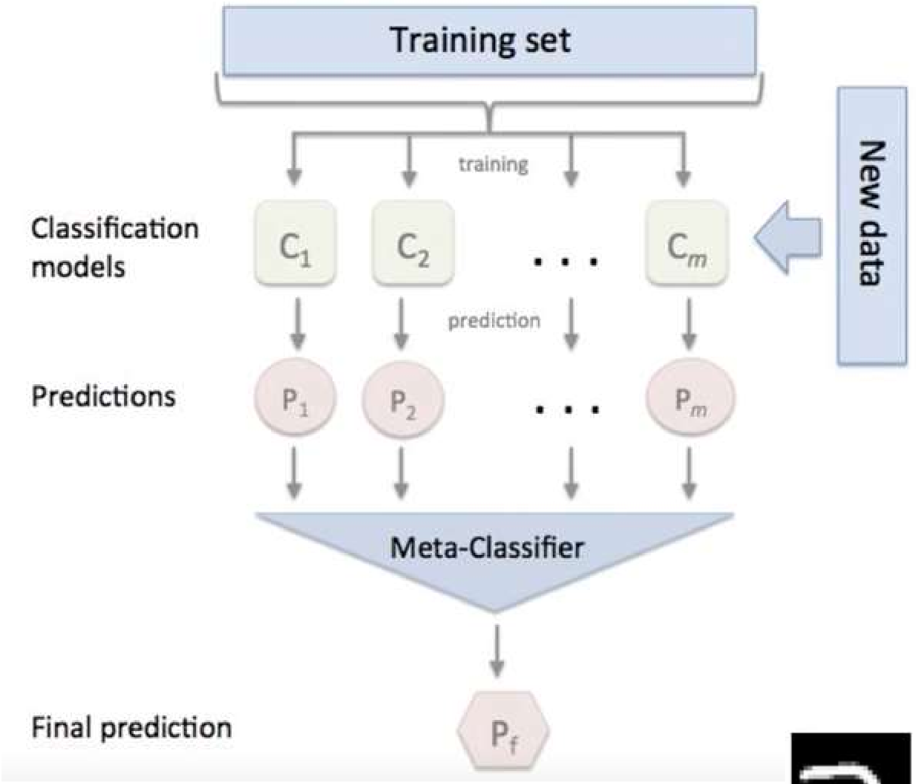
- 실습: MNIST 99% using CNN
  - Deep CNN (**Dropout + additional FC layer**)+**Callback**

```
class callback_Chk_ACC(tf.keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('accuracy')>0.99):
            print("\nAccuracy is 99%")
            self.model.stop_training = True

callbacks = callback_Chk_ACC()
```

```
tf.model.fit(x_train, y_train, batch_size=batch_size, epochs=training_epochs, callbacks=[callbacks])
```

◆ Convolution Neural Network

– **Ensemble**



Accuracy: 99.52%

	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**	...	...	...	...	...	...

Convolution Neural Network

– Fashion MNIST

```
import tensorflow as tf
print(tf.__version__)
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
training_images=training_images.reshape(60000, 28, 28, 1)
training_images=training_images / 255.0
test_images = test_images.reshape(10000, 28, 28, 1)
test_images=test_images/255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(28, 28, 1)),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2,2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
model.fit(training_images, training_labels, epochs=1)
test_loss = model.evaluate(test_images, test_labels)
```

- **TSR using CNN**
  - **Build a Traffic Sign Recognition Project**
    - » Pip install pandas
    - » Pip install sklearn
    - » Pip install scikit-image

TSR using CNN

- **Build a Traffic Sign Recognition Project**
  - Load the data set (German Traffic Sign: http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset)
    - Using provided “pickle” files

```
# Load pickled data
import pickle

# TODO: Fill this in based on where you saved the training and testing data

training_file = 'traffic-signs-data/train.p'
validation_file = 'traffic-signs-data/valid.p'
testing_file = 'traffic-signs-data/test.p'

with open(training_file, mode='rb') as f:
    train = pickle.load(f)
with open(validation_file, mode='rb') as f:
    valid = pickle.load(f)
with open(testing_file, mode='rb') as f:
    test = pickle.load(f)

X_train, y_train = train['features'], train['labels']
X_valid, y_valid = valid['features'], valid['labels']
X_test, y_test = test['features'], test['labels']
```

Containing raw pixel data of the traffic sign images

  - Explore, Summarize and visualize the data set
  - Design, Train and Test a CNN Model architecture
  - Use the model to make predictions on new images
  - Analyze the softmax probabilities of the new images

Project #1: TSR(Traffic Sign Recognition) using CNN

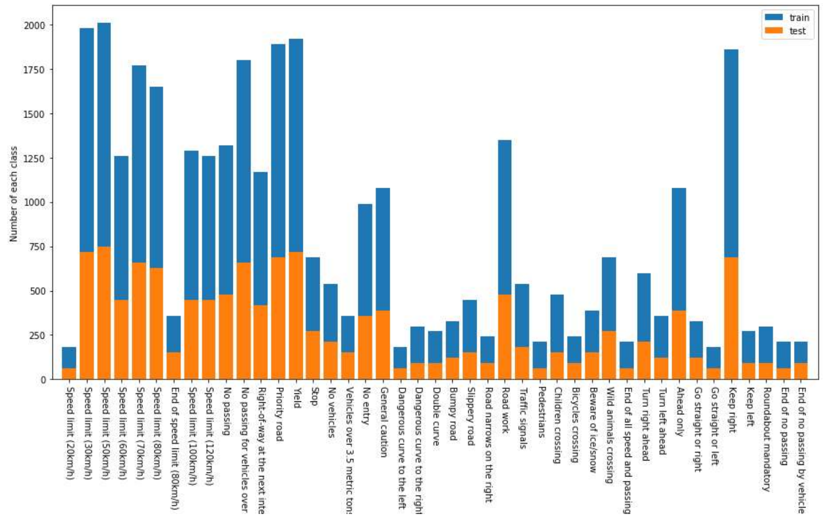
TSR using CNN

- Build a Traffic Sign Recognition Project
  - Load the data set
  - Explore, Summarize and visualize the data set
    - The size of training/validation/test set is 34799/4410/12630.
    - The shape of a traffic sign images is (32, 32, 3). `image_shape = X_train[0].shape`
    - The number of unique classes/labels in the data set is 43. `n_classes = len(np.unique(y_train))`

```
Number of training examples = 34799
Number of validation examples = 4410
Number of testing examples = 12630
Image data shape = (32, 32, 3)
Number of classes = 43
```

 # TODO: make code

 # TODO: Reference the provided code



  - Design, Train and Test a CNN Model architecture
  - Use the model to make predictions on new images
  - Analyze the softmax probabilities of the new images

TSR using CNN

- **Build a Traffic Sign Recognition Project**
  - » Load the data set
  - » Explore, Summarize and visualize the data set

 # TODO: Reference the provided code

(1)

```
### Data exploration visualization code goes here.
### Feel free to use as many code cells as needed.
import matplotlib.pyplot as plt
import pandas as pd
import random as rnd
import cv2
# Visualizations will be shown in the notebook.
%matplotlib inline

readfile = pd.read_csv('signnames.csv')
sign_name = readfile['SignName'].values

train_classes, train_class_cnt = np.unique(y_train, return_counts = True)
test_classes, test_class_cnt = np.unique(y_test, return_counts = True)

fig, axis = plt.subplots(2,4, figsize=(15,6))
fig.subplots_adjust(hspace=0.2, wspace=0.2)
axis = axis.ravel()
for i in range(8):
    idx = rnd.randint(0, n_train)
    img = X_train[idx]
    axis[i].axis('off')
    axis[i].set_title(sign_name[y_train[idx]])
    axis[i].imshow(img)
```

(2)

```
fig0 = plt.figure(figsize=(13,10))
plt.bar(np.arange(n_classes), train_class_cnt, align='center', label='train')
plt.bar(np.arange(n_classes), test_class_cnt, align='center', label='test')
plt.xlabel('Class: Name of Traffic sign')
plt.ylabel('Number of each class')
plt.xlim([-1, n_classes])
plt.xticks(np.arange(n_classes), sign_name, rotation=270)
plt.legend()
plt.tight_layout()

plt.show()
```

  - » Design, Train and Test a CNN Model architecture
  - » Use the model to make predictions on new images
  - » Analyze the softmax probabilities of the new images

Project #1: TSR(Traffic Sign Recognition) using CNN

TSR using CNN

– **Build a Traffic Sign Recognition Project**
  - » Load the data set
  - » Explore, Summarize and visualize the data set
  - » Design, Train and Test a CNN Model architecture
    - Pre-processing image data
      - ✓ Color channel images & normalize

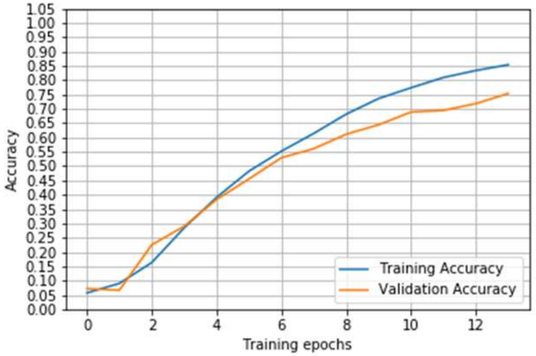
```python
def normalize_image(image_data):
    return (image_data - 128)/ 128
```

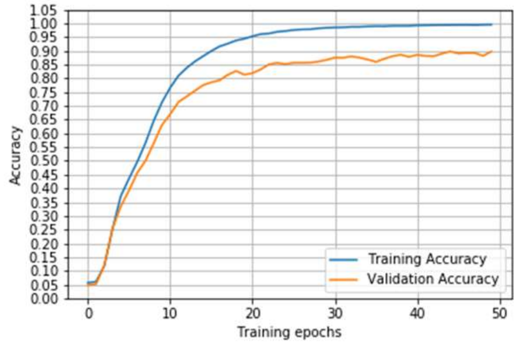
Layer	Description
Input	32x32x3 (Color & Normalize)
Convolution 3x3	1x1 stride, same padding, outputs 32x32x32
RELU	
Max pooling	2x2 stride, outputs 16x16x32
Convolution 3x3	1x1 stride, same padding, outputs 16x16x64
RELU	
Max pooling	2x2 stride, outputs 8x8x64
Convolution 3x3	1x1 stride, same padding, outputs 8x8x64
RELU	
Dropout	0.6
Convolution 3x3	1x1 stride, same padding, outputs 8x8x96
RELU	
Dropout	0.6
Convolution 3x3	1x1 stride, same padding, outputs 8x8x128
RELU	
Dropout	0.6

Flatten	8x8x128 = 8192
Fully connected	(8192, 256)
Dropout	0.6
Fully connected	(256, 128)
Dropout	0.6
Fully connected	(128, 84)
Dropout	0.6
Fully connected	(84, 43)

```python
def plot_train(train_acc, valid_acc):
    fig, ax = plt.subplots()
    ax.plot(range(len(train_acc)), train_acc, label="Training Accuracy")
    ax.plot(range(len(valid_acc)), valid_acc, label="Validation Accuracy")

    ax.set_xlabel('Training epochs')
    ax.set_ylabel('Accuracy')
    ax.legend(loc=4)
    ax.set_ylim([0,1])
    plt.yticks(np.arange(0, 1.1, 0.05))
    plt.grid(True)
    plt.show()

    fig.savefig('train_valid_graph_color_norm.png')
```

 # TODO: Make CNN Architecture & plot using provided code





  - » Use the model to make predictions on new images
  - » Analyze the softmax probabilities of the new images

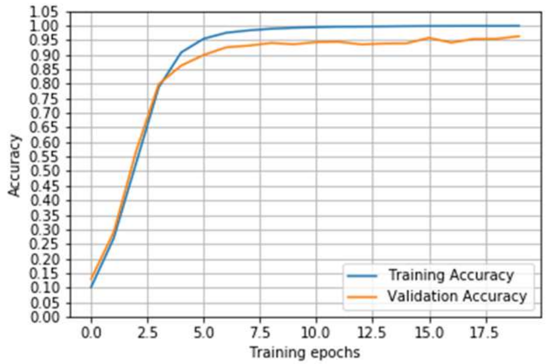
TSR using CNN

– Build a Traffic Sign Recognition Project

- » Load the data set
- » Explore, Summarize and visualize the data set
- » Design, Train and Test a CNN Model architecture
  - ▪ Pre-processing image data
    - ✓ Grayscale images & normalize

 # TODO: Convert Color RGB to Grayscale using provided code And Plotting

```
def norm (img_data):
#     return (img_data - 128)/ 128
#     return img_data / np.max(img_data)
    return img_data / 255

def gray_scale(X):
    X = 0.299 * X[:, :, :, 0] + 0.587 * X[:, :, :, 1] + 0.114 * X[:, :, :, 2]
    X = X.reshape(X.shape + (1,))
    return X
```




```
# X_train = norm(gray_scale(X_train))
# X_valid = norm(gray_scale(X_valid))
# X_test = norm(gray_scale(X_test))
```

The result of the accuracy of the validation set is **about 96%.**

- » Use the model to make predictions on new images
- » Analyze the softmax probabilities of the new images

Project #1: TSR(Traffic Sign Recognition) using CNN

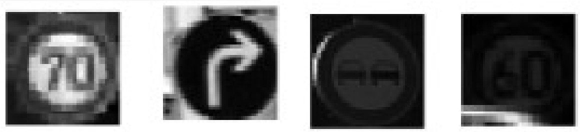
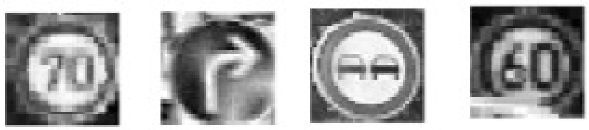
TSR using CNN

- Build a Traffic Sign Recognition Project
  - » Load the data set
  - » Explore, Summarize and visualize the data set
  - » Design, Train and Test a CNN Model architecture
    - Pre-processing image data
      - ✓ Grayscale images & normalize

 # TODO: Apply CLAHE (Contrast Limited Adaptive Histogram Equalization) to Grayscale using provided code and Plotting CLAHE images

```
X_train_gray = []
X_train_CLAHE = []
X_valid_gray = []
X_valid_CLAHE = []
X_test_gray = []
X_test_CLAHE = []

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(4,4))
for i in range(n_train):
    X_train_gray.append(cv2.cvtColor(X_train[i], cv2.COLOR_RGB2GRAY))
    X_train_CLAHE.append(clahe.apply(X_train_gray[i]))
for i in range(n_validation):
    X_valid_gray.append(cv2.cvtColor(X_valid[i], cv2.COLOR_RGB2GRAY))
    X_valid_CLAHE.append(clahe.apply(X_valid_gray[i]))
for i in range(n_test):
    X_test_gray.append(cv2.cvtColor(X_test[i], cv2.COLOR_RGB2GRAY))
    X_test_CLAHE.append(clahe.apply(X_test_gray[i]))

fig, axis = plt.subplots(2,4, figsize=(15,6))
fig.subplots_adjust(hspace=0.2, wspace=0.2)
axis = axis.ravel()
```

```
for i in range(8):
    idx = rnd.randint(0, n_train)
    img = X_train_CLAHE[idx]
    axis[i].axis('off')
    axis[i].set_title(sign_name[y_train[idx]])
    axis[i].imshow(img, 'gray')

X_train_arr = np.array(X_train_CLAHE)
X_valid_arr = np.array(X_valid_CLAHE)
X_test_arr = np.array(X_test_CLAHE)
X_train_arr = X_train_arr.reshape(X_train_arr.shape + (1,))
X_valid_arr = X_valid_arr.reshape(X_valid_arr.shape + (1,))
X_test_arr = X_test_arr.reshape(X_test_arr.shape + (1,))
X_train = norm(X_train_arr)
X_valid = norm(X_valid_arr)
X_test = norm(X_test_arr)
```

  - » Use the model to make predictions on new images
  - » Analyze the softmax probabilities of the new images

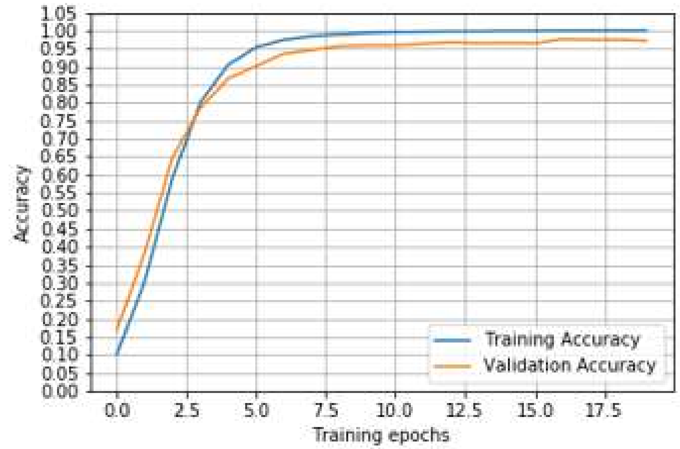
Project #1: TSR(Traffic Sign Recognition) using CNN

- **TSR using CNN**
  - **Build a Traffic Sign Recognition Project**
    - » Load the data set
    - » Explore, Summarize and visualize the data set
    - » Design, Train and Test a CNN Model architecture
      - Pre-processing image data
        - ✓ Grayscale images & normalize

 # TODO: Make CNN Architecture and plot the accuracy

Layer	Description
Input	32x32x1 (CLAHE & Normalize)
Convolution 3x3	1x1 stride, same padding, outputs 32x32x96
RELU	
Max pooling	2x2 stride, outputs 16x16x96
Convolution 4x4	1x1 stride, same padding, outputs 16x16x128
RELU	
Max pooling	2x2 stride, outputs 8x8x128
Convolution 3x3	1x1 stride, same padding, outputs 8x8x256
RELU	
Max pooling	2x2 stride, outputs 4x4x256
Convolution 4x4	1x1 stride, same padding, outputs 4x4x256
RELU	
Dropout	0.5
Flatten	4x4x256 = 4096
Fully connected	(4096, 1024)
Dropout	0.5
Fully connected	(1024, 256)
Dropout	0.5
Fully connected	(256, 43)

- Training set accuracy of 99%
- Validation set accuracy of 97.3%
- Test set accuracy of 95.6%



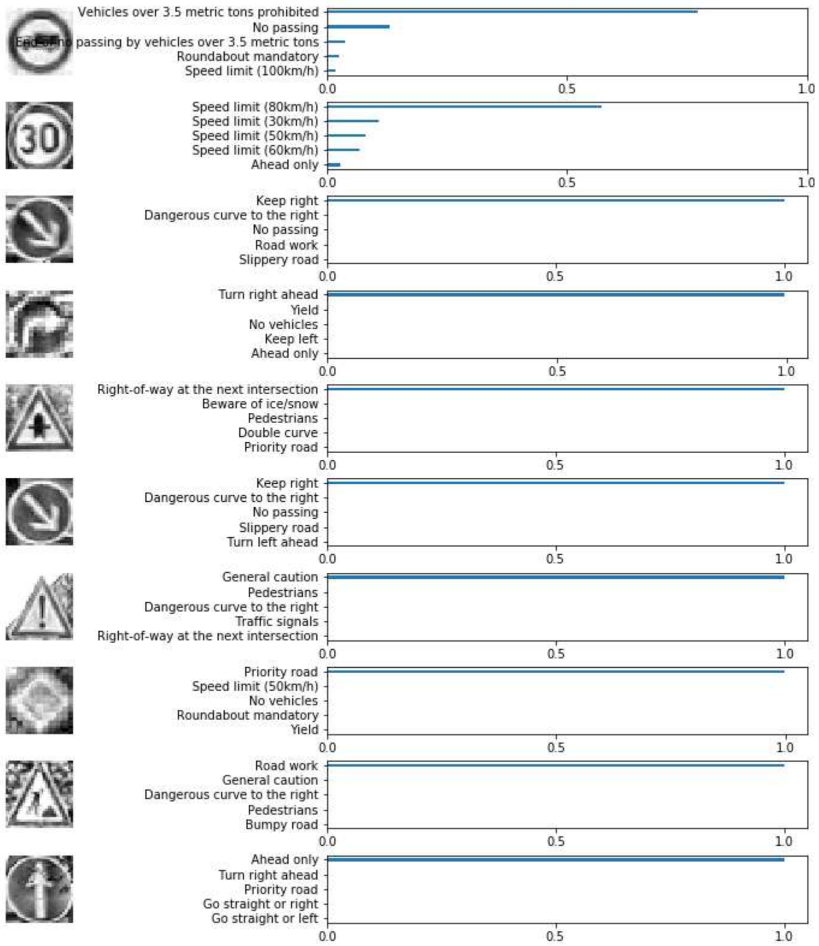
- » Use the model to make predictions on new images
- » Analyze the softmax probabilities of the new images

Project #1: TSR(Traffic Sign Recognition) using CNN

- **TSR using CNN**
  - **Build a Traffic Sign Recognition Project**
    - Load the data set
    - Explore, Summarize and visualize the data set
    - Design, Train and Test a CNN Model architecture
    - Use the model to make predictions on new images
    - Analyze the softmax probabilities of the new images

 # TODO: Reference provided code and some test data

```python
def plot_test_images(images,n):
    fig, axes = plt.subplots(1, n, figsize=(13,5))
    fig.subplots_adjust(hspace=0.1, wspace=0.1)

    for i, ax in enumerate(axes.flat):
        ax.imshow(images[i])
        ax.set_title(i+1)
        ax.set_xticks([])
        ax.set_yticks([])
#     fig.savefig('in5.png')
### Load the images
from skimage import io
imgs = [ io.imread('test0/test{}.png'.format(i + 11)) for i in range(10) ]

plot_test_images(imgs,10)
```



Thank you & Good luck !