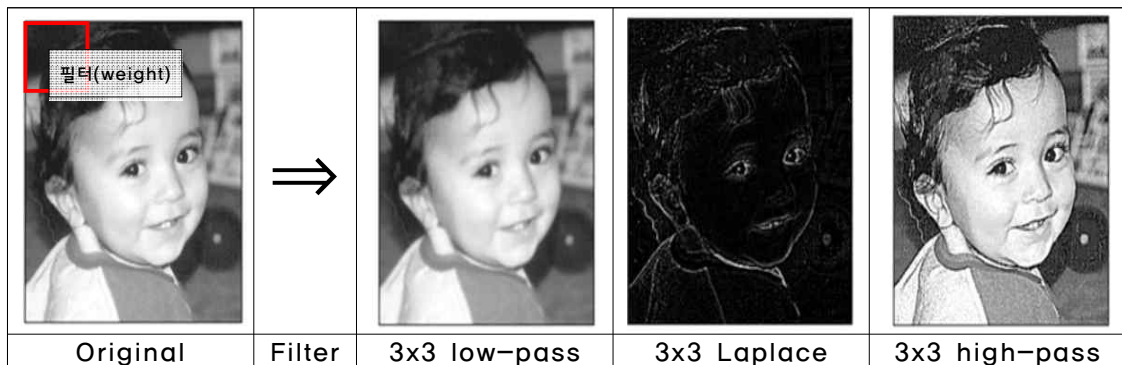


# 합성곱신경망(CNN. Convolutional Neural Network)

변환 불변성에 기초하여 이미지를 분석에 사용하는 깊은 인공신경망의 한 종류  
필터 역할을 스스로 학습하여 상대적으로 전처리를 거의 사용하지 않는다

Convolution: filter 연산에서 사용되어 영상에서 feature를 추출할 때 사용

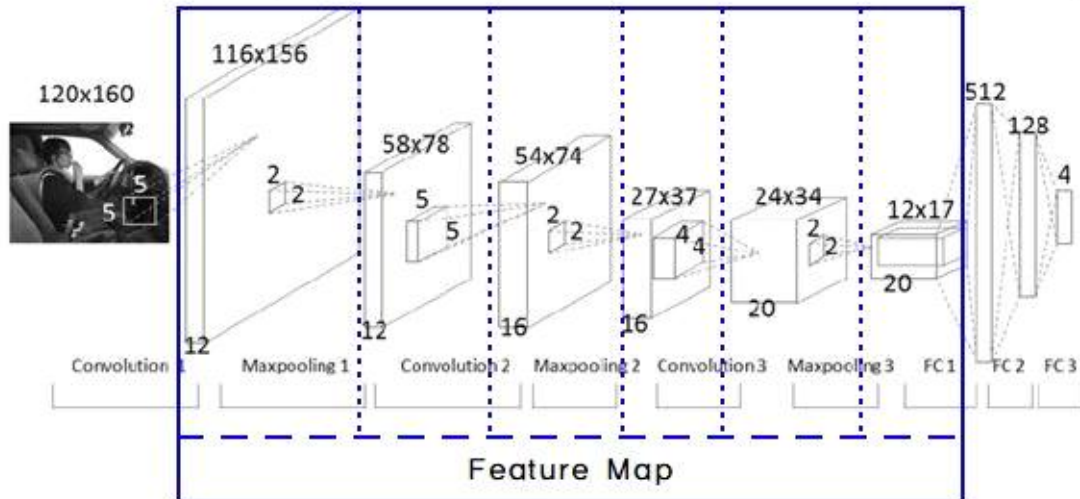


Convolution의 과정	<div> <div> 11100 01110 00111 00110 01100 </div> <div> 101 010 101 </div> <div> </div> </div> (Filter, Kernel, Weight)		
	Image		Convolved Feature
	<div> 1x11x01x100 0x01x11x010 0x10x01x111 00110 01100 </div>	⇒	<div> 4  </div>
	<div> 11x11x00x10 01x01x11x00 00x11x01x11 00110 01100 </div>	⇒	<div> 43  </div>
	⋮		
	Green: 영상 이미지 Yellow: Convolution이 일어나는 영역 Red: Convolution Kernel		

Convolution + NN : Convolution을 사용하는 신경망 연산

2개 이상의 CNN layer

& 입력 영상뿐만 아니라 중간 Feature map에도 Convolution 적용



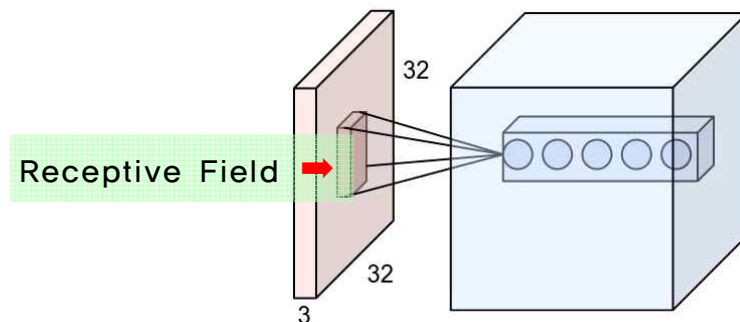
Receptive Field(수용 영역)

: 출력 layer의 뉴런 하나에 영향을 미치는 입력 뉴런들의 공간 크기

외부 자극이 일부 영역에만 영향을 미친다(전체에 영향 X)

영상에서 특정 위치에 있는 픽셀들은 주변에 있는 일부 픽셀들과만

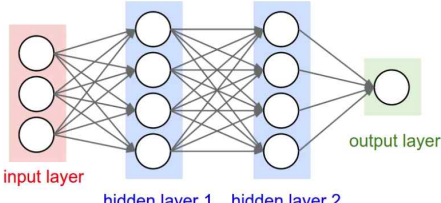
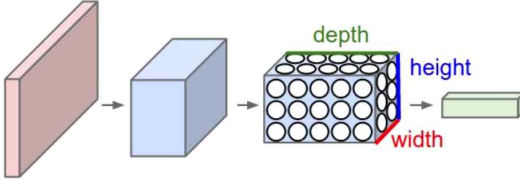
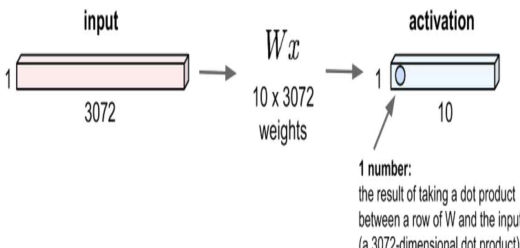
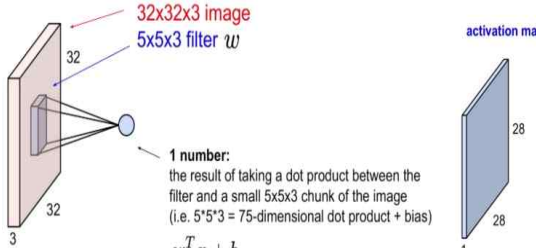
correlation이 높다 → 거리가 멀수록 영향이 감소한다


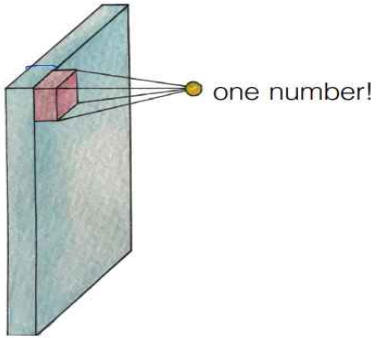



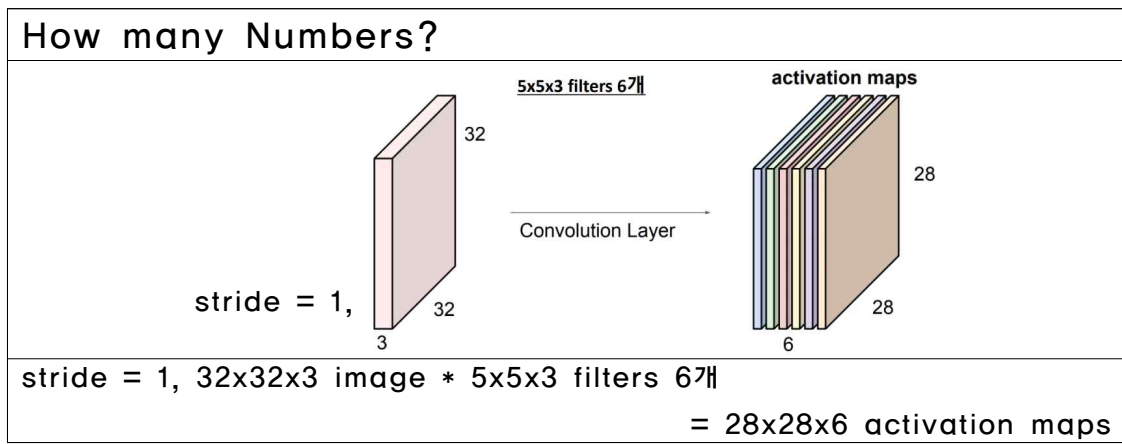
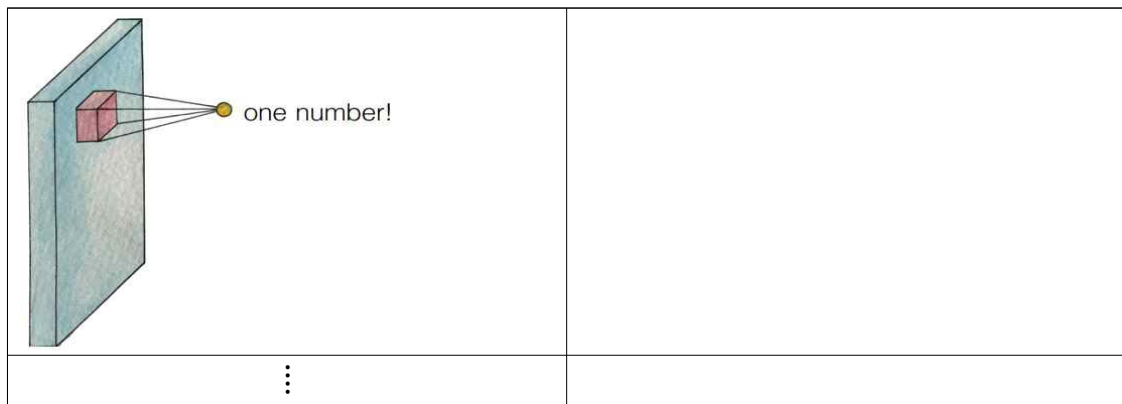
CNN의 장점

영상을 2D에서 1D로 평탄화하지 않으므로, 형상을 유지한다

∴ 입출력 모두 3차원 데이터로 처리하기 때문에 공간적 정보를 유지할 수 있다

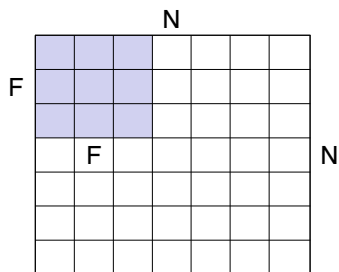
DNN	CNN
 <p>input layer hidden layer 1 hidden layer 2 output layer</p>	 <p>depth height width</p>
 <p>input 3072 <math>Wx</math> 10 x 3072 weights activation 10</p> <p>1 number: the result of taking a dot product between a row of <math>W</math> and the input (a 3072-dimensional dot product)</p>	 <p>32x32x3 image 5x5x3 filter <math>w</math> activation map 28 28 1</p> <p>1 number: the result of taking a dot product between the filter and a small 5x5 chunk of the image (i.e. <math>5 \times 5 \times 3 = 75</math>-dimensional dot product + bias)</p> <p><math>w^T x + b</math></p>
10x3072 weights	5x5x3 filter $w$
많은 수의 가중치	적은 수의 가중치 - 부분의 합으로 판단하는 브레인 모사

 <p>32x32x3 image</p>	<p>Image: width x height x depth depth = 1:1 채널 = 흑백 depth = 3:3 채널 = 칼라(RGB)</p>
 <p>32x32x3 image</p> <p>one number!</p>	 <p>5x5x3 filter</p> <p>image * filter = <math>Wx + b</math> = one number (<math>W</math> = filter, <math>b</math> = bias)</p>
⋮	



Stride: Filter가 움직이는 정보  
 stride = n : n칸씩 움직인다

### Output size 구하기



$$(N - F) / \text{stride} + 1$$

ex)  $N = 7, F = 3$

stride = 1 :  $(7-3)/1+1 = 5$  (ok)

stride = 2 :  $(7-3)/2+1 = 3$  (ok)

stride = 3 :  $(7-3)/3+1 = 2.33$  (X)

stride = 4 :  $(7-3)/4+1 = 2$  (ok)

## Padding: 영상 사이즈 유지하기

ex) input 7x7 image

pad with 1 pixel border(회색)

0	0	0	0	0				
0								
0								
0								

if. pad의 값 = 0 : zero pad

⇐ zero pad with 1

Kernel에 따른 Padding의 크기

=  $(F / 2)$ 의 소수점 첫째 자리 올림

ex)

$F = 3$  : zero pad with 1

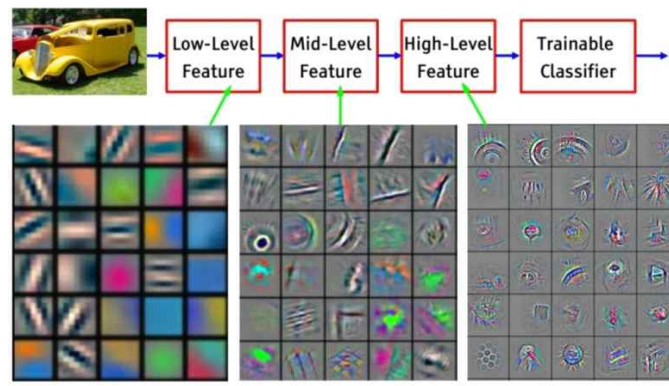
$F = 5$  : zero pad with 2

$F = 7$  : zero pad with 3

## Convolution 결과의 Size 구하기

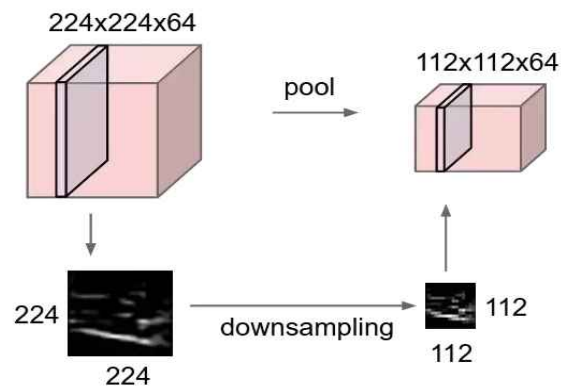
Input: $W_1 \times H_1 \times D_1$	Parameter Filter의 개수: $K$ Filter의 한 변 크기: $F$ Stride: $S$ Zero Pad의 개수: $P$
Output: $W_2 \times H_2 \times D_2$	$W_2 = (W_1 - F + 2P) / S + 1$ $H_2 = (H_1 - F + 2P) / S + 1$ $D_2 = K$ → filter 개수로 output의 depth 정할 수 있다
with parameter sharing, $F \cdot F \cdot D_1$ weight per filter total $\Rightarrow (F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases	

## 1. Visualization of Activation Map(Feature Map)



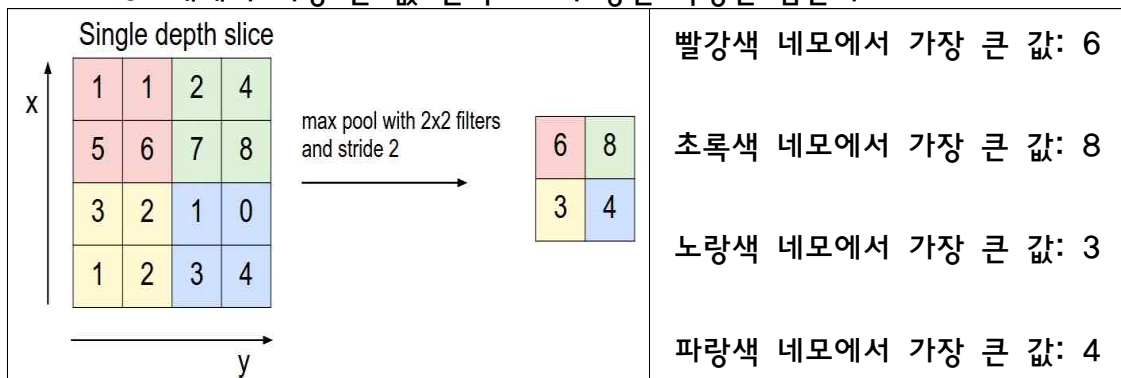
## 2. Pooling Layer(Sampling)

: resizing Conv layer



### Max Pooling

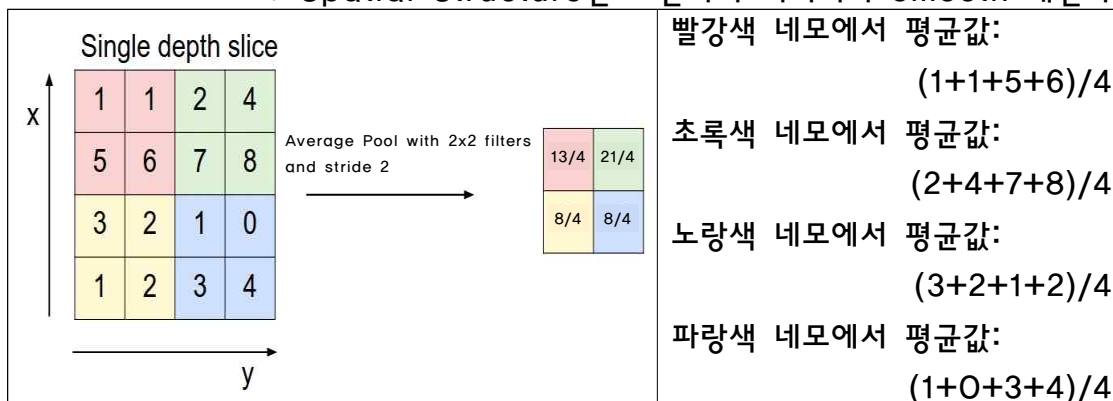
Filter 내에서 가장 큰 값 선택  $\Rightarrow$  더 강한 특징만 남는다



## Average Pooling

Filter 내에서 평균값 선택

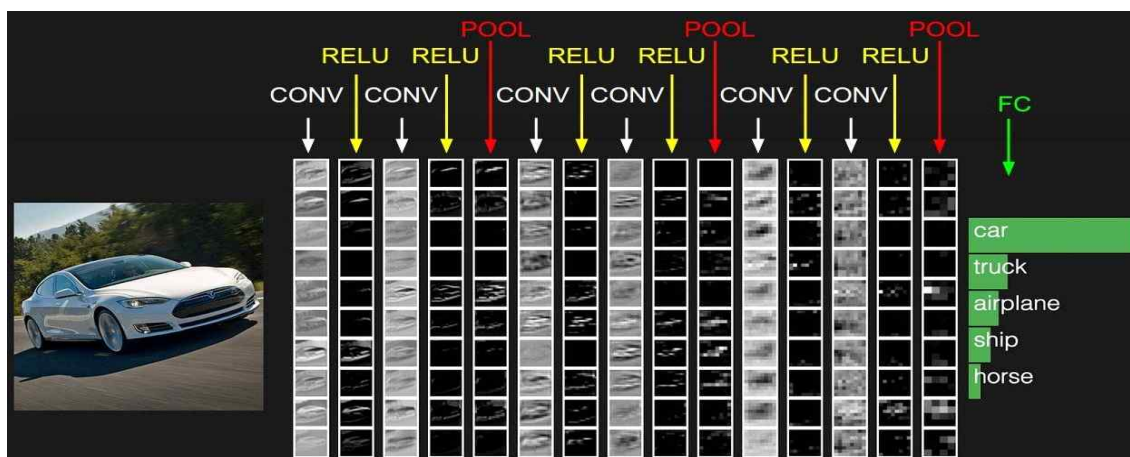
⇒ Spatial Structure만 보존하여 이미지가 smooth 해진다



- (2x2 filter의 경우) 전체 데이터의 75%를 버리고 25%만 선택  
→ Computational Complexity 감소한다
- Depth를 줄이지 않고 Spatially하게만 줄인다(Height & Width)  
 $32 \times 32 \times 3 \rightarrow 16 \times 16 \times 3$
- Q) Stride와 Pooling 모두 down-sampling인데 어느 것 사용?  
A) 최근 CNN 아키텍처는 stride를 사용하는 경우가 많다(stride 추천)

## 3. FC layer(Fully Connected Layer)

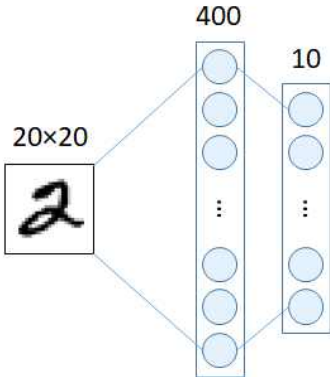
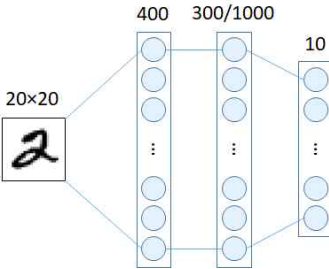
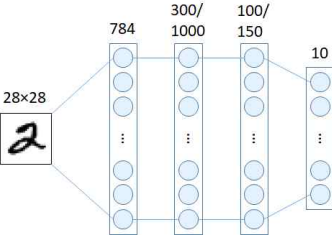
마지막 Pooling layer를 통과한 데이터가 1x1024 feature를 갖는다면  
FC layer with W=1024x5를 통과하여 1x5의 Output이 나온다



# CNN 영상 분류기

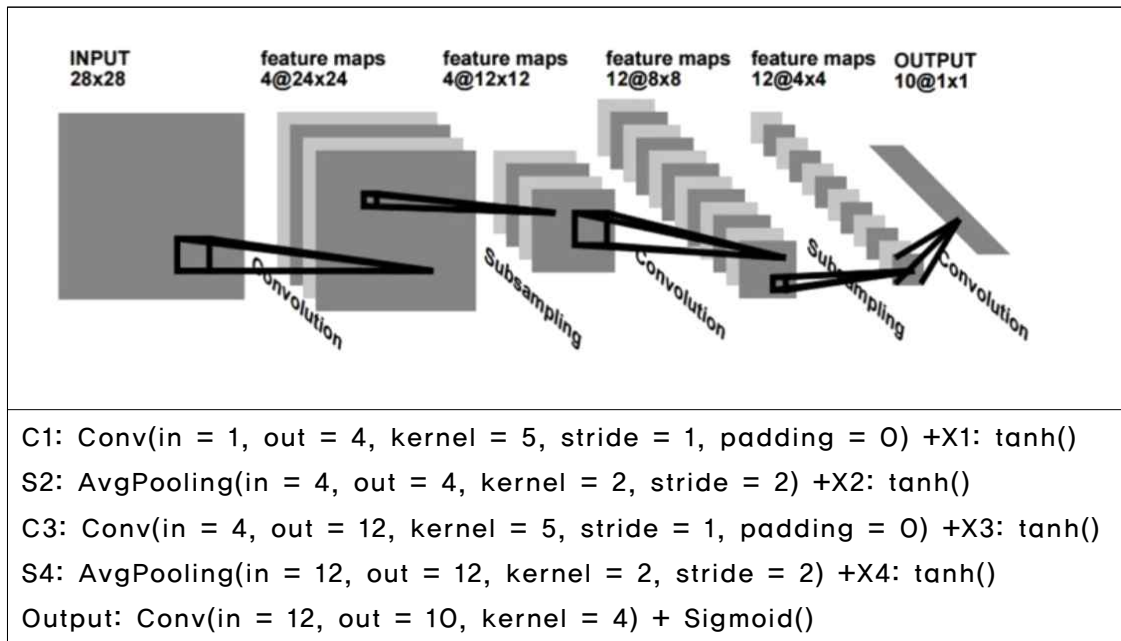
## MNIST: Image Classification

DNN	Baseline Linear Classifier		One-Hidden-Layer Fully Connected Multi-layer NN		Two-Hidden-Layer Fully Connected Multi-layer NN	
Error	8.4%		3.6% to 3.8%		2.95% to 3.05%	
CNN	LeNet-1	LeNet-4		LeNet-5		Boosted LeNet-4
Error	1.7%	1.1%		0.95%		0.7%

Baseline Linear Classifier	One-Hidden-Layer Fully Connected Multi-layer NN	Two-Hidden-Layer Fully Connected Multi-layer NN
20x20 → 10: error 8.4%	20x20 → 1000 → 10: error 3.8% 20x20 → 300 → 10: error 3.6%	28x28 → 300 → 100 → 10: error 3.05% 28x28 → 1000 → 150 → 10: error 2.95%
		

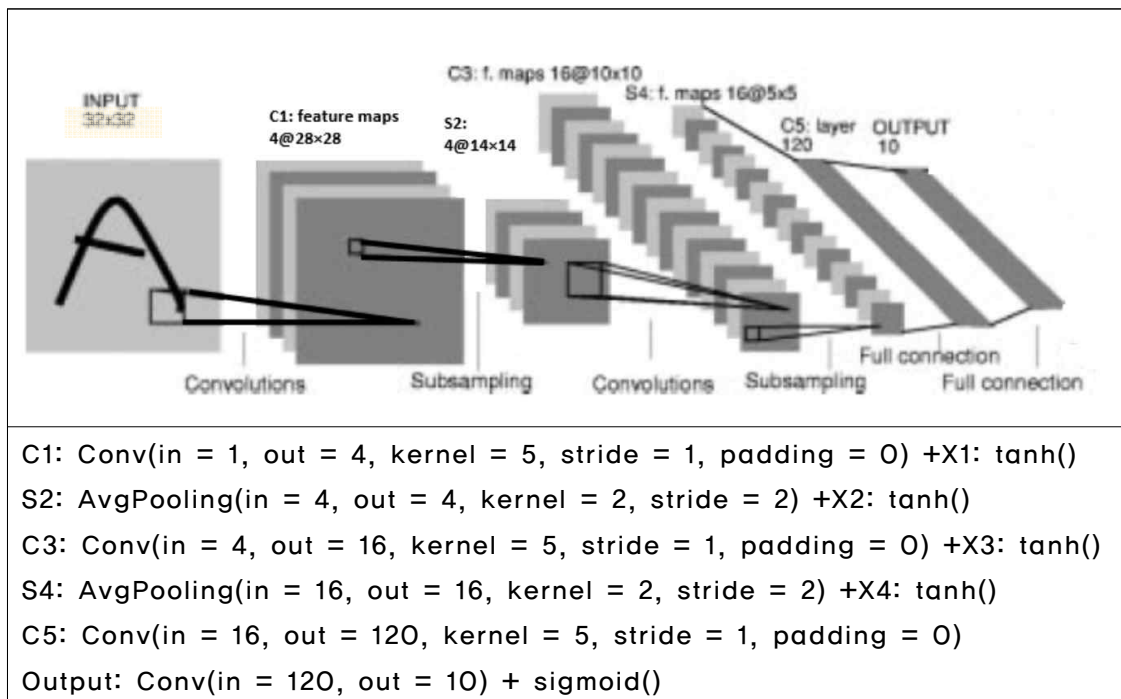


## LeNet-1



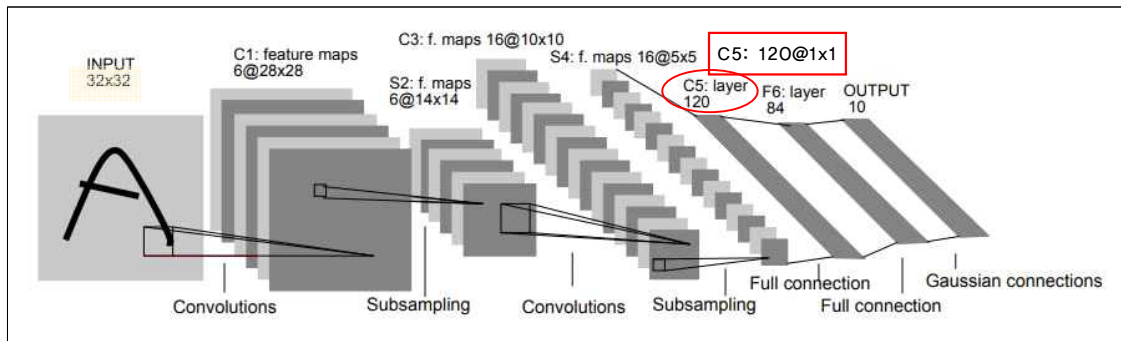
- Conv(in = 12, out = 10, kernel = 4) + Sigmoid() == FC Layer

## LeNet-4

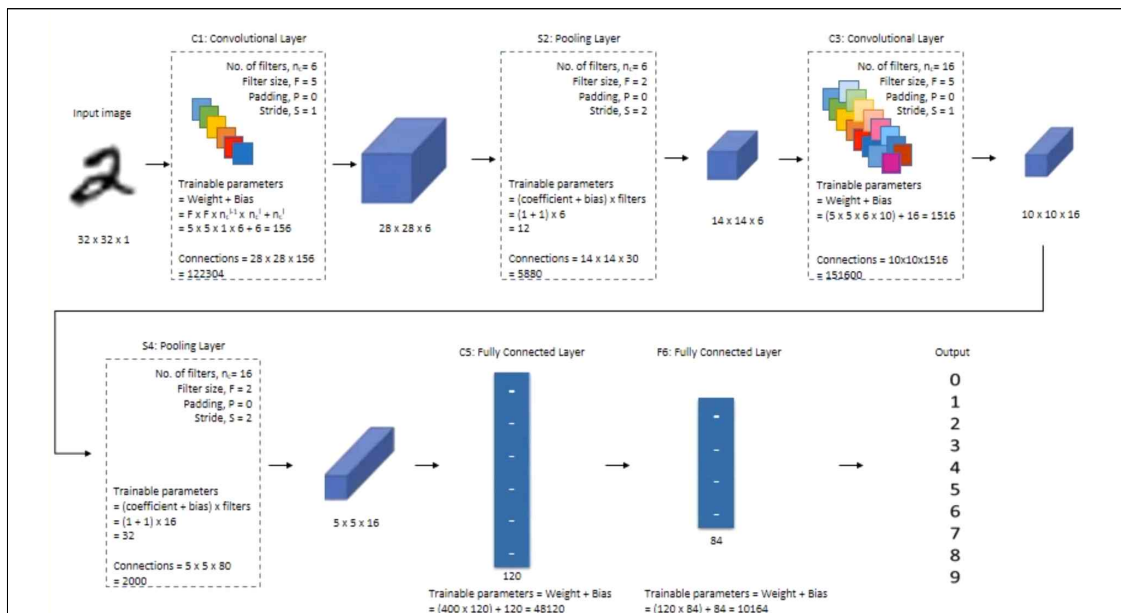


※ FC Layer 2개: C5, Output

## LeNet-5



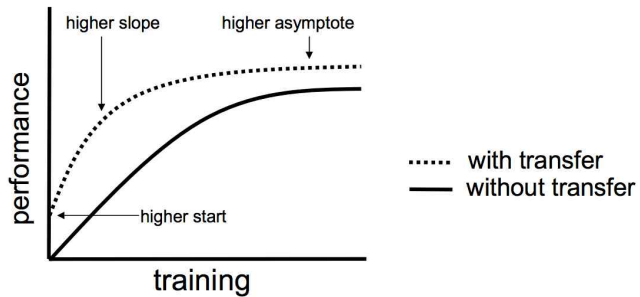
C1: Conv(in = 1, out = 6, kernel = 5, stride = 1, padding = 0) +X1: tanh()  
 S2: AvgPooling(in = 6, out = 6, kernel = 2, stride = 2) +X2: tanh()  
 C3: Conv(in = 6, out = 16, kernel = 5, stride = 1, padding = 0) +X3: tanh()  
 S4: AvgPooling(in = 16, out = 16, kernel = 2, stride = 2) +X4: tanh()  
 C5: Conv(in = 16, out = 120) +X5: tanh()  
 F6: FC(in = 120, out = 84) +X6: tanh()  
 Output: FC(in = 84, out = 10) + sigmoid()



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	—	—	—
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	—	84	—	—	tanh
Output	FC	—	10	—	—	softmax

## 전이 학습(Transfer Learning)

: 이미 학습된 신경망 능력을 유사하거나 전혀 새로운 분야의 신경망 학습에 사용  
높은 정확도를 비교적 짧은 시간 내에 달성 가능

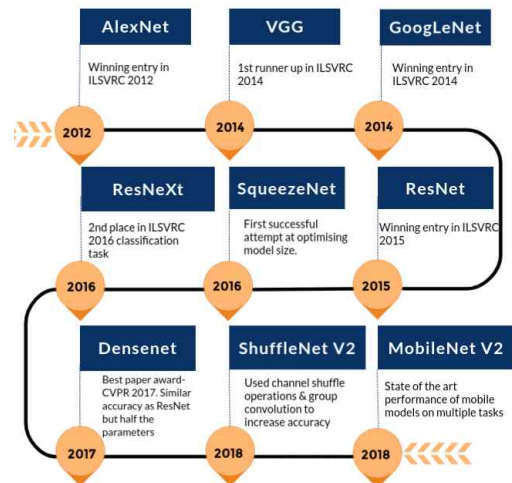
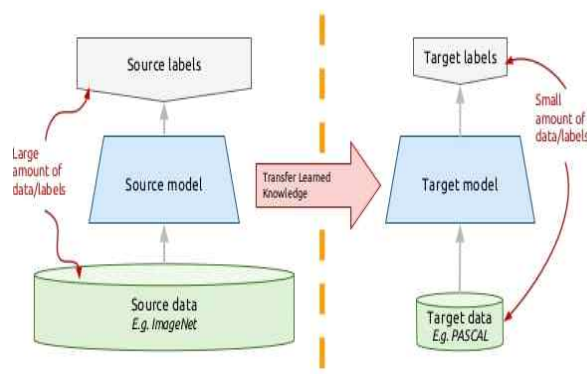


컴퓨터 비전에서의 전이 학습 = 사전 학습된 모델(pre-trained model) 이용

## 사전 학습된 모델(Pretrained Model)

: 풀고자 하는 문제와 비슷하고 사이즈가 큰 데이터로 이미 학습이 된 모델  
오랜 시간과 연산량으로 학습되어 있다  
어려운 문제를 잘 풀면, 쉬운 문제도 잘 풀 것으로 기대한다

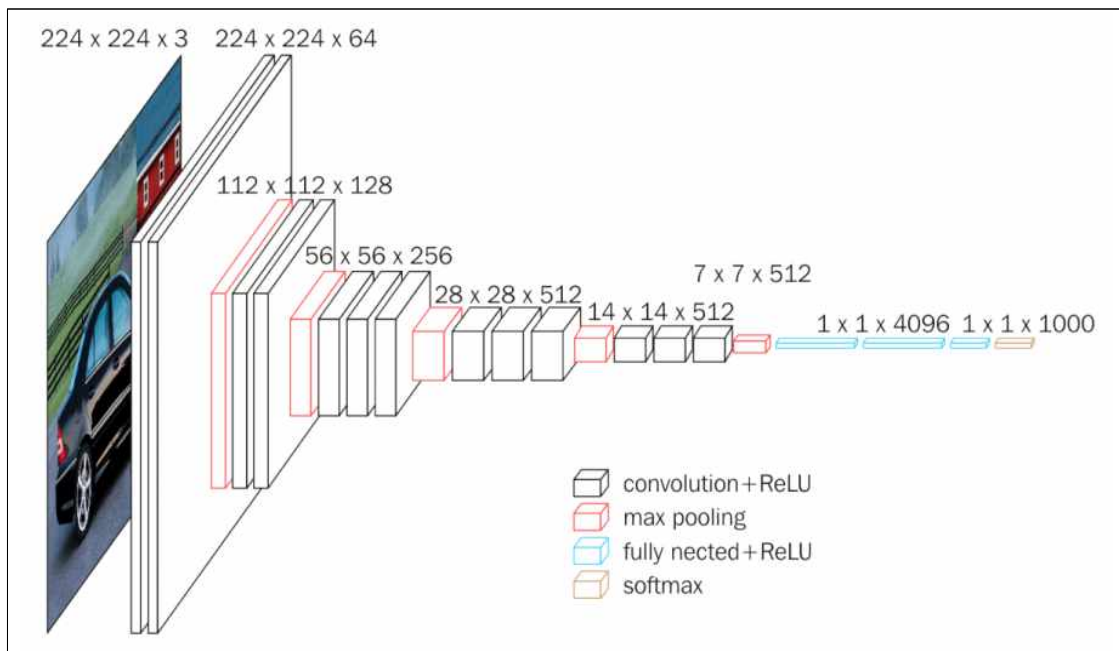
### Transfer learning: idea



※ Pretrained Model의 입력 사이즈에 맞게

입력 영상 사이즈를 Resize 해야 한다

## VGG-16



VGG(

(features): Sequential(

C0: Conv2d(in = 3, out = 64, kernel = 3, stride = 1, padding = 1)

X1: ReLU(inplace = True)

C2: Conv2d(in = 64, out = 64, kernel = 3, stride = 1, padding = 1)

X3: ReLU(inplace = True)

S4: MaxPool2d(kernel = 2, stride = 2, padding = 0, dilation = 1, ceil\_mode = False)

C5: Conv2d(in = 64, out = 128, kernel = 3, stride = 1, padding = 1)

X6: ReLU(inplace = True)

C7: Conv2d(in = 128, out = 128, kernel = 3, stride = 1, padding = 1)

X8: ReLU(inplace = True)

S9: MaxPool2d(kernel = 2, stride = 2, padding = 0, dilation = 1, ceil\_mode = False)

C10: Conv2d(in = 128, out = 256, kernel = 3, stride = 1, padding = 1)

X11: ReLU(inplace = True)

C12: Conv2d(in = 256, out = 256, kernel = 3, stride = 1, padding = 1)

X13: ReLU(inplace = True)

C14: Conv2d(in = 256, out = 256, kernel = 3, stride = 1, padding = 1)

X15: ReLU(inplace = True)

S16: MaxPool2d(kernel = 2, stride = 2, padding = 0, dilation = 1, ceil\_mode = False)

C17: Conv2d(in = 256, out = 512, kernel = 3, stride = 1, padding = 1)

X18: ReLU(inplace = True)

C19: Conv2d(in = 512, out = 512, kernel = 3, stride = 1, padding = 1)

X20: ReLU(inplace = True)

C21: Conv2d(in = 512, out = 512, kernel = 3, stride = 1, padding = 1)

X22: ReLU(inplace = True)

S23: MaxPool2d(kernel = 2, stride = 2, padding = 0, dilation = 1, ceil\_mode = False)

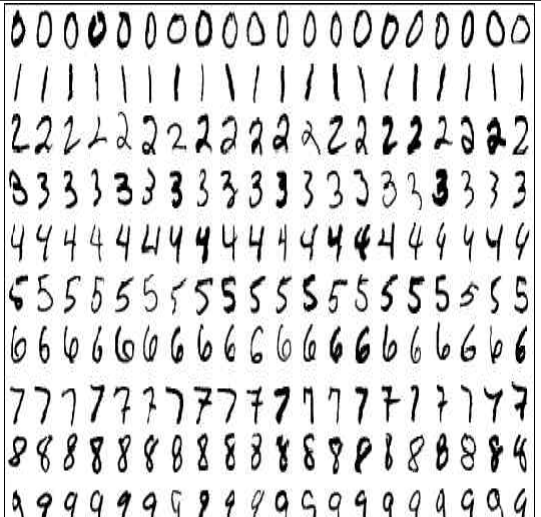
```

C24: Conv2d(in = 512, out = 512, kernel = 3, stride = 1, padding 1)
X25: ReLU(inplace = True)
C26: Conv2d(in = 512, out = 512, kernel = 3, stride = 1, padding 1)
X27: ReLU(inplace = True)
C28: Conv2d(in = 512, out = 512, kernel = 3, stride = 1, padding 1)
X29: ReLU(inplace = True)
S30: MaxPool2d(kernel = 2, stride = 2, padding = 0, dilation = 1, ceil_mode = False)
)
(avgpool): AdaptiveAvgPool2d(output_size = (7, 7))
(classifier): Sequential(
  (0): Linear(in = 25088, out = 4096, bias = True)
  (1): ReLU(inplace = True)
  (2): Dropout(p = 0.5, inplace = False)
  (3): Linear(in = 4096, out = 4096, bias = True)
  (4): ReLU(inplace = True)
  (5): Dropout(p = 0.5, inplace = False)
  (6): Linear(in = 4096, out = 10000, bias = True)
)
)
import torchvision.models as models
vgg16 = models.vgg16(pretrained = True).to(device)

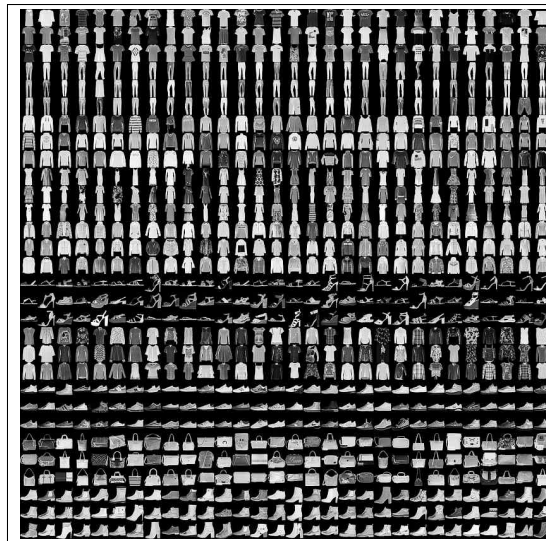
```

## 영상 분류 데이터셋

### 1. MNIST Dataset

	<p>클래스: 10개</p> <p>학습데이터: 60,000장</p> <p>테스트데이터: 10,000장</p> <p>해상도: 28x28</p>
---	--

## 2. Fashion-MNIST Dataset



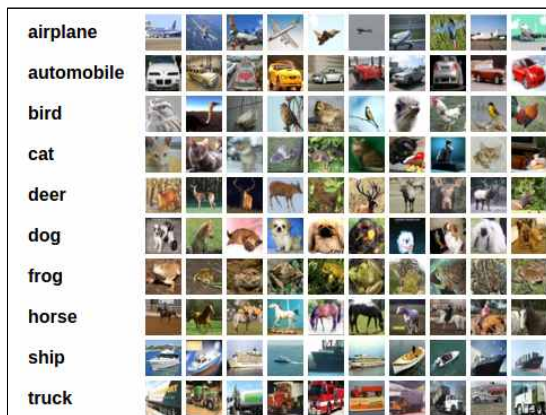
10개의 카테고리 범주

70,000개의 흑백 이미지

28x28 이미지 해상도

레이블	클래스
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

## 3. CIFAR-10



클래스: 10개

클래스당: 6,000장

학습 데이터: 50,000장

테스트 데이터: 10,000장

해상도: 32x32

## 4. CIFAR-100



클래스: 100개

클래스당: 600장

학습 데이터: 50,000장

테스트 데이터: 10,000장

해상도: 32x32



## 5. Caltech101, 2003년

	<p>클래스: 101개 + 배경 전체 이미지: 9144장 해상도: 300x200</p>
---	--

## 6. Caltech256, 2006년

	<p>클래스: 256개 + 배경 전체 이미지: 30,608장 클래스별: 80~827장 해상도: 300x200</p>
--	--

## 7. ImageNet, 2009년

	<p>클래스: 1000개 전체 이미지: 14백만장</p> <p>학습 데이터: 138G 테스트 데이터: 6.3G</p>
---	---

## Benchmark

### 1. CIFAR-10

Model	Accuracy
VGG16	92.64%
ResNet18	93.02%
ResNet50	93.62%
ResNet101	93.75%
RegNetX 200MF	94.24%
RegNetY 400MF	94.29%
MobileNetV2	94.43%
ResNeXt29(32x4d)	94.73%
ResNeXt29(2x64d)	94.82%
DenseNet121	95.04%
PreActResNet18	95.11%
DPN92	95.16%

### 2. ImageNet

