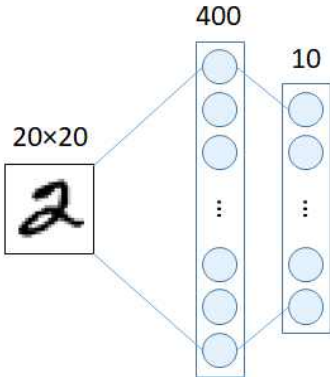
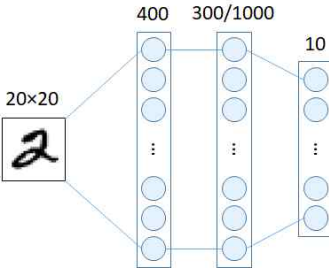
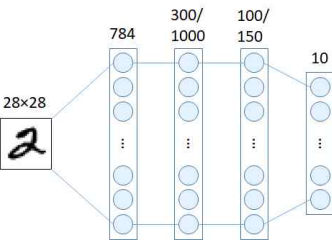


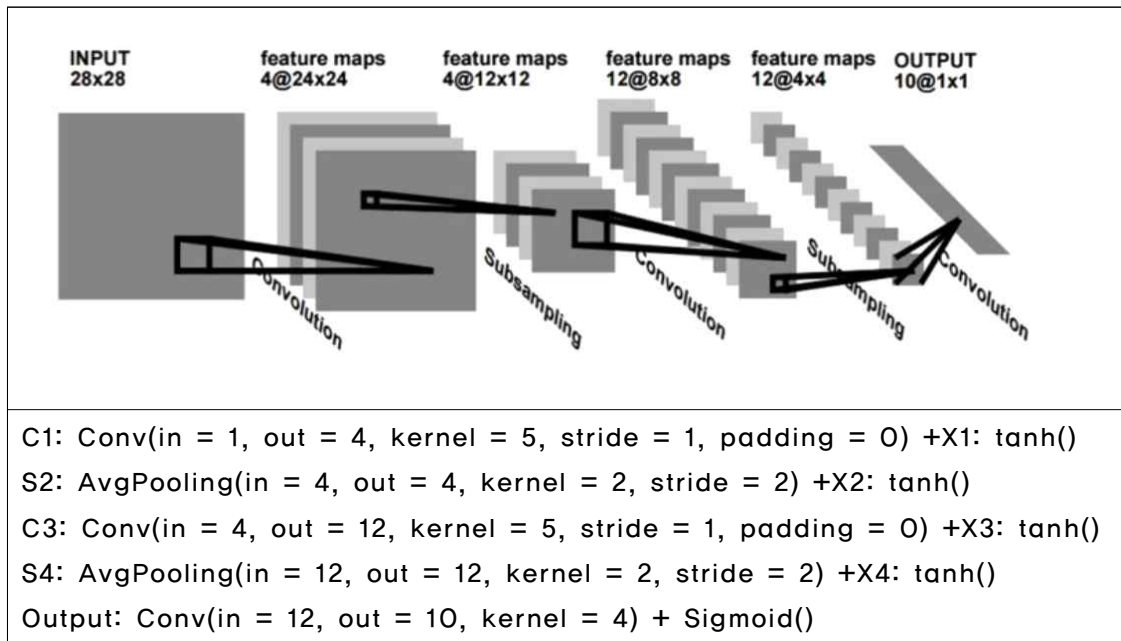
CNN 영상 분류기

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%

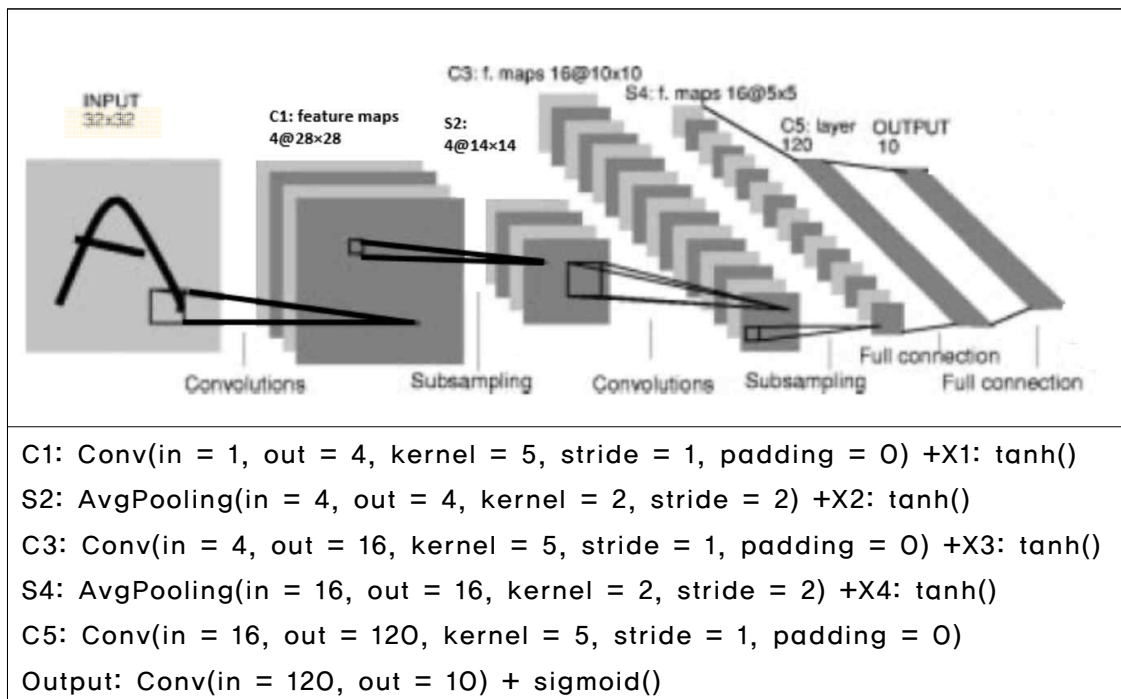
Baseline Linear Classifier	One-Hidden-Layer Fully Connected Multi-layer NN	Two-Hidden-Layer Fully Connected Multi-layer NN
20x20 → 10: error 8.4%	20x20 → 1000 → 10: error 3.8% 20x20 → 300 → 10: error 3.6%	28x28 → 300 → 100 → 10: error 3.05% 28x28 → 1000 → 150 → 10: error 2.95%
		

LeNet-1



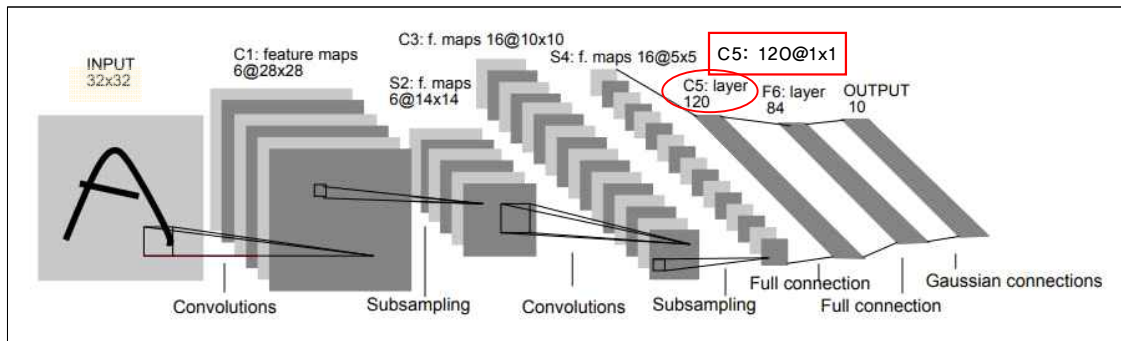
- Conv(in = 12, out = 10, kernel = 4) + Sigmoid() == FC Layer

LeNet-4

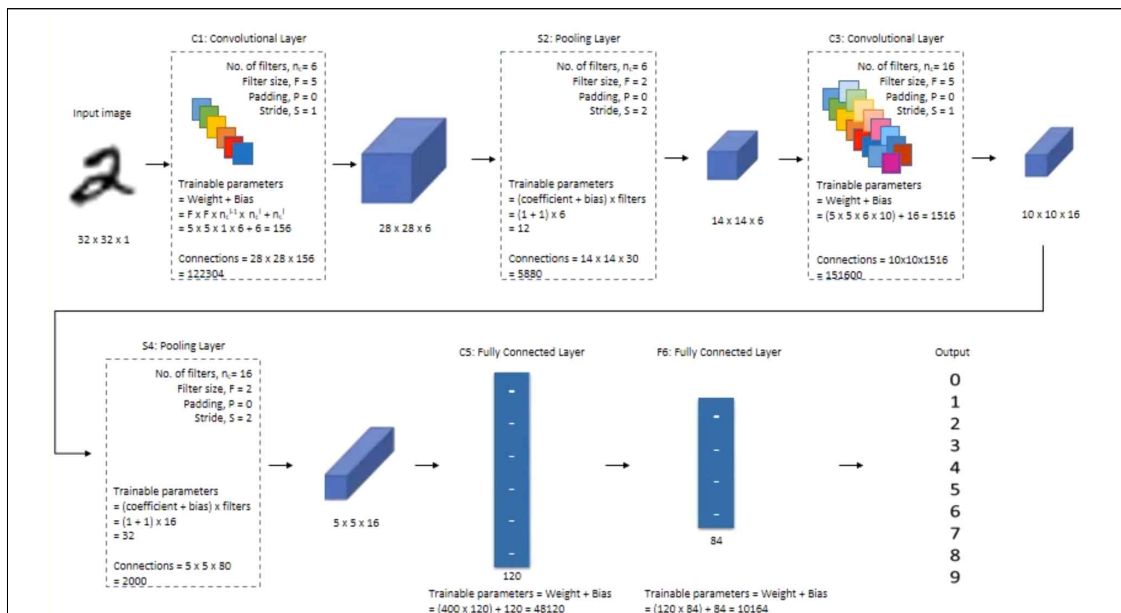


※ FC Layer 2개: C5, Output

LeNet-5



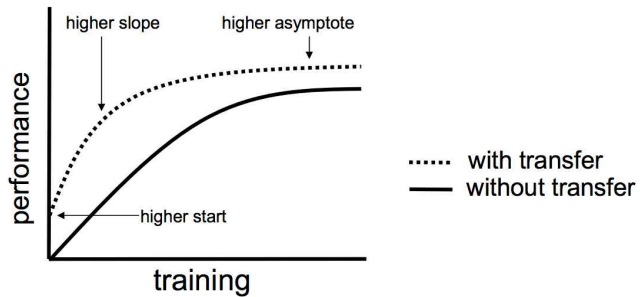
C1: Conv(in = 1, out = 6, kernel = 5, stride = 1, padding = 0) +X1: tanh()
 S2: AvgPooling(in = 6, out = 6, kernel = 2, stride = 2) +X2: tanh()
 C3: Conv(in = 6, out = 16, kernel = 5, stride = 1, padding = 0) +X3: tanh()
 S4: AvgPooling(in = 16, out = 16, kernel = 2, stride = 2) +X4: tanh()
 C5: Conv(in = 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

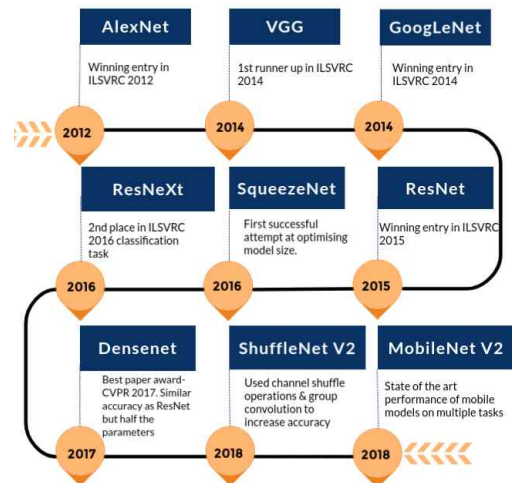
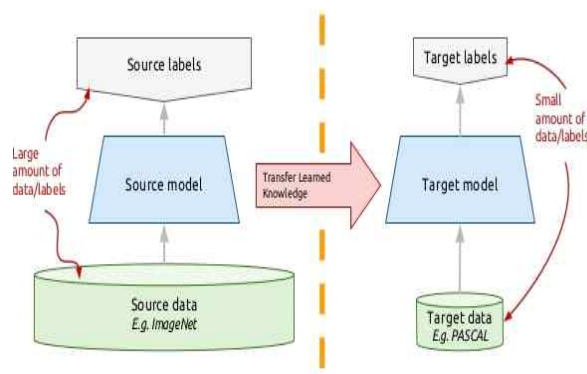


컴퓨터 비전에서의 전이 학습 = 사전 학습된 모델(pre-trained model) 이용

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

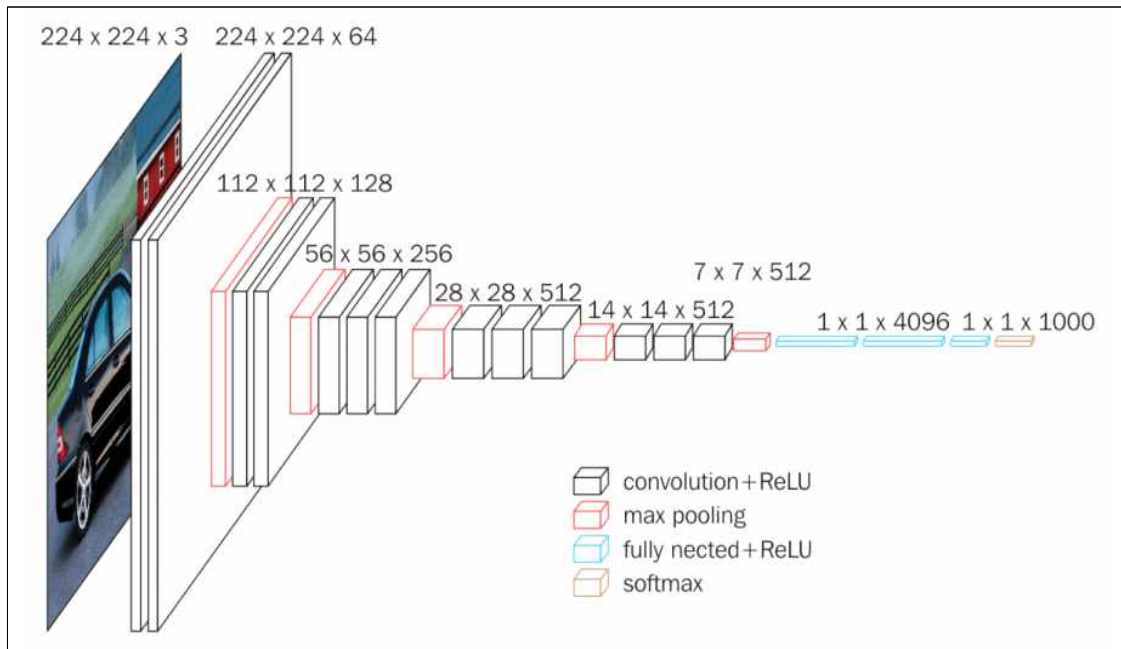
Transfer learning: idea



※ 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)

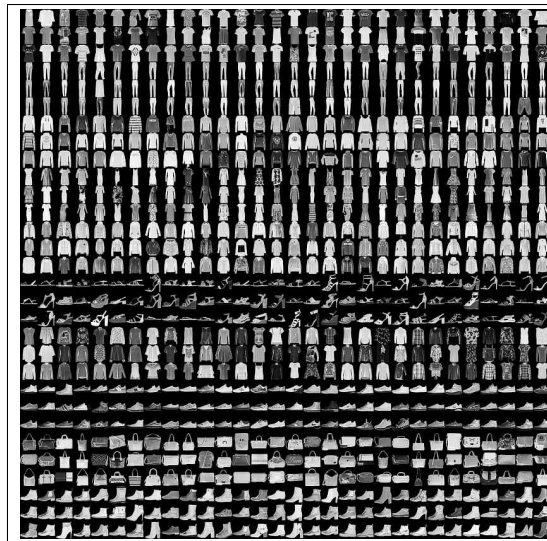
```

영상 분류 데이터셋

1. MNIST Dataset

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

2. Fashion-MNIST Dataset



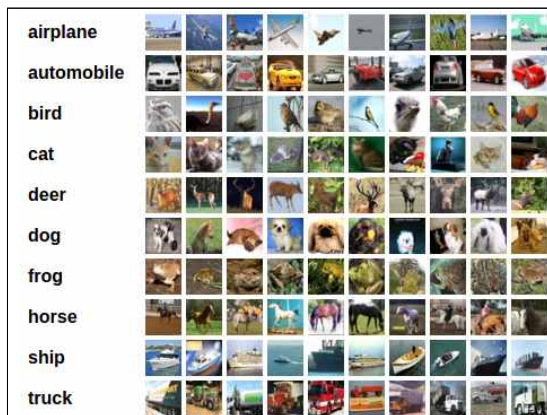
10개의 카테고리 범주

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

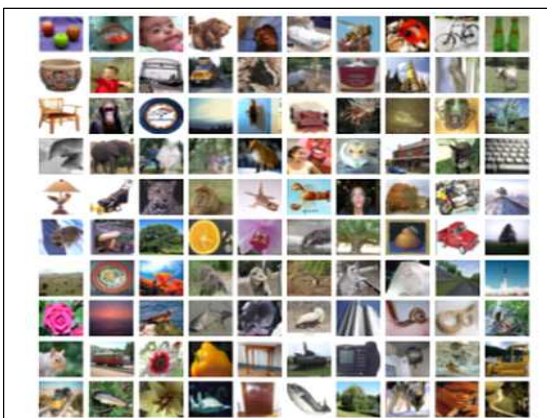
클래스당: 6,000장

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

