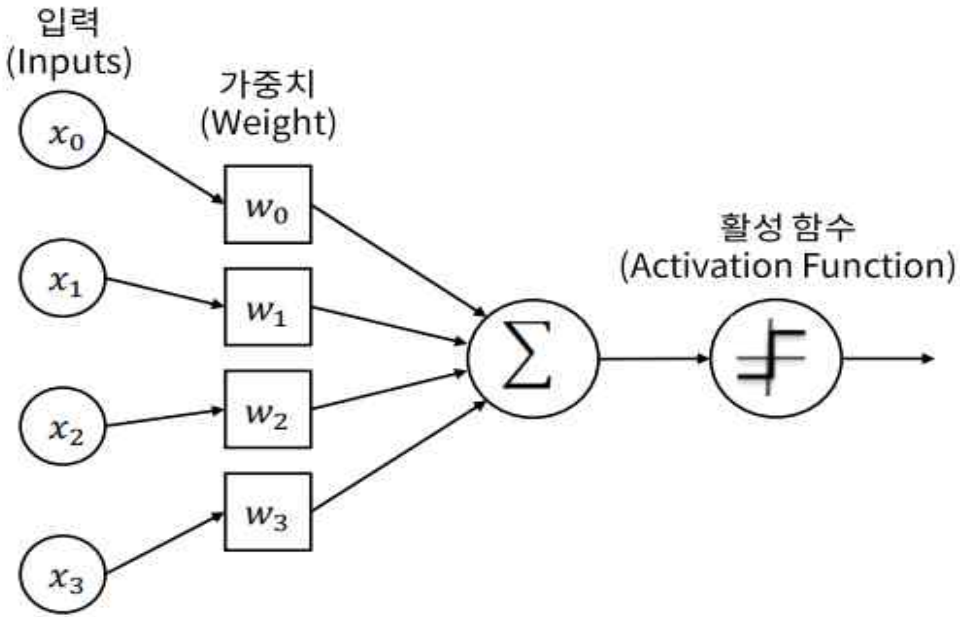
Deep Learning
Study week 5

문구영

이미지 분석과 딥러닝

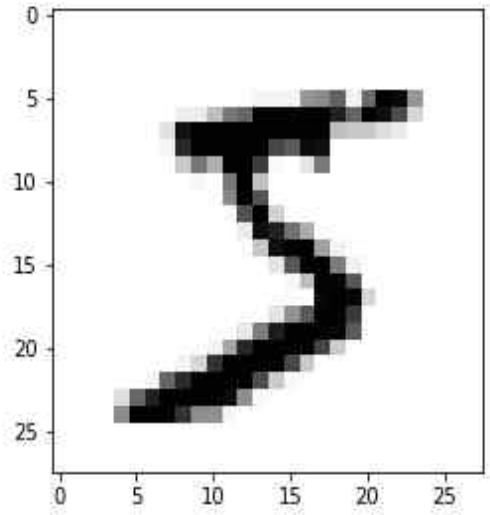


- 이미지의 입력 : width x height 픽셀값

- 기존의 모델 구조를 적용하기에는 한계 o
  - 인접 픽셀간의 상호 연관성 o (독립 x)
  - 픽셀의 위치 정보 중요

→ CNN (Convolutional Neural Network)

이미지 분석을 위한 전처리-입력

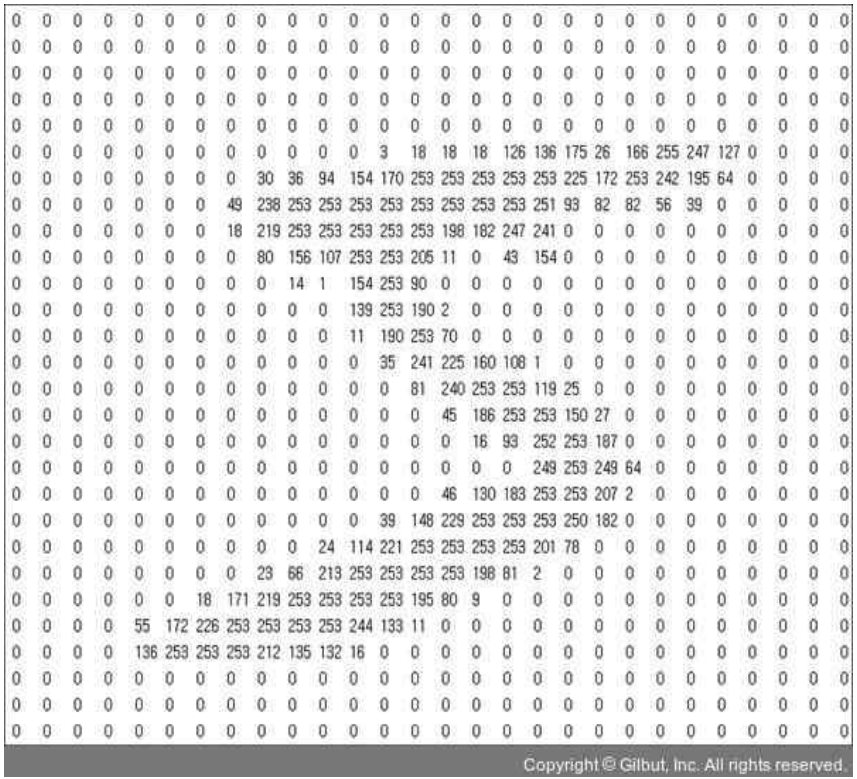


1. 파이썬이 인식할 수 있는 데이터 타입으로 변환
   - 가로 28 x 세로 28 = 총 784개의 픽셀
   - 각 픽셀은 0~255의 값 (밝기 등급에 따라)
   - **정규화** : 0~1사이의 값을 가지도록

2. 모델이 해당 이미지를 **5**라고 인식할 수 있도록 학습

   학습 : 이미지의 유용한 패턴, 표현 등을 학습

이미지 분석을 위한 전처리-입력



- 정규화를 적용시키기 전

- 케라스는 **데이터를 0~1사이의 값으로 변환**한 다음 구동할 때 최적의 성능을 보임

```
train_images
= train_images.astype('float32') / 255
```

이미지 분석을 위한 전처리-입력

컬러 이미지는 **RGB 3** 채널로 구성된 **2-D Signal (0~1)**로 표현

(샘플 개수**,** 가로**,** 세로**,** 채널**) : 4D Tensor**

이미지 분석을 위한 전처리-출력

```
Y_train = np.utils.to_categorical(Y_class_train, 10)
Y_test = np.utils.to_categorical(Y_class_train, 10)
```

- (클래스, 클래스의 개수)

```
print(Y_train[0])
```

```
[0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
```

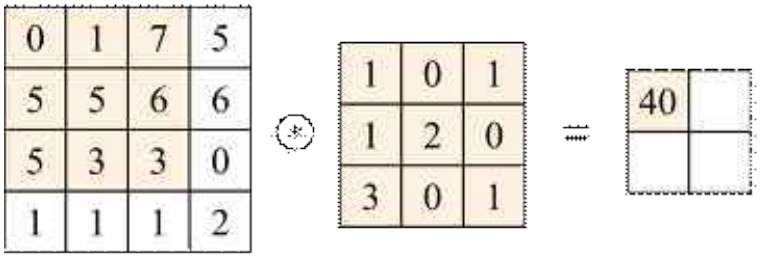
```
[0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
```

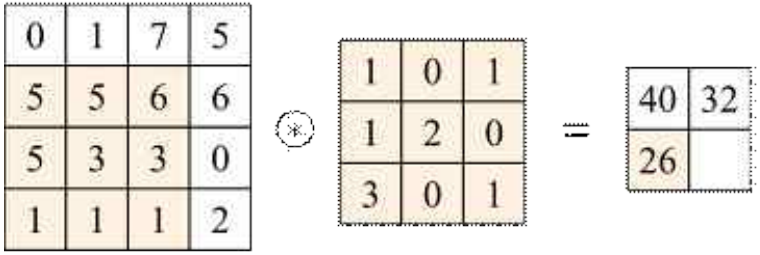
- 3을 표현하는 입력 이미지의 클래스 (타깃)

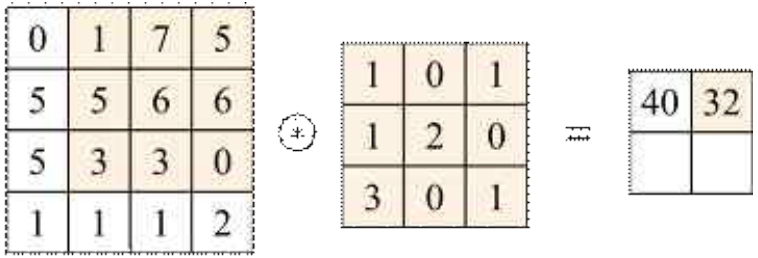
딥러닝의 분류 문제 : **원 핫 인코딩** 적용

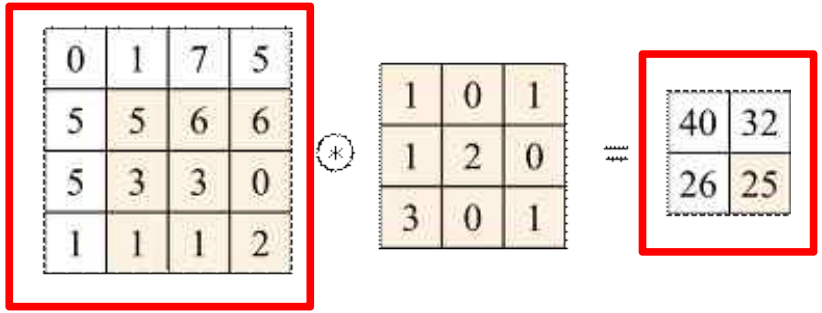
0~9까지의 정수형 값 → **0 or 1로만 이루어진 벡터**로 수정

합성곱 연산







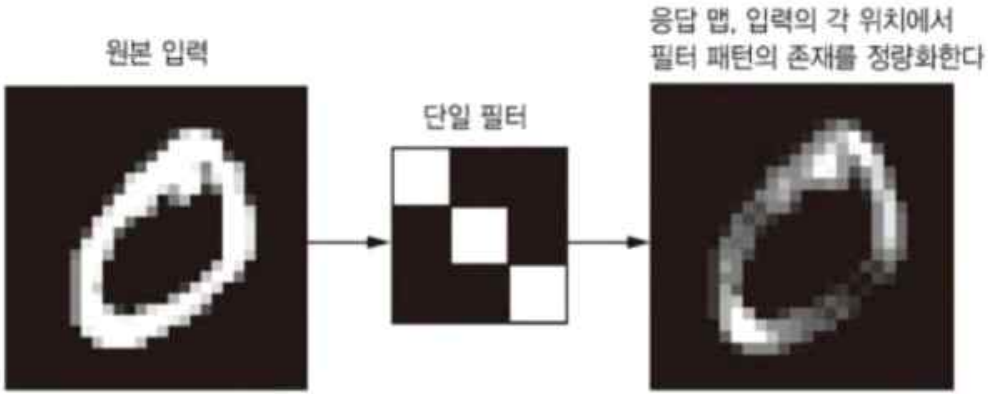


4 x 4 이미지 → **2 x 2** 이미지

3 x 3 필터 / 커널 / 윈도우

- 필터 : 입력의 각 위치에서 패턴, 특성 확인
- 필터의 각 칸 : 가중치

합성곱 연산



- 필터는 입력 데이터의 특성, 패턴을 인코딩
- 특징이 나타나는 위치 탐색
- 응답 맵 : 입력의 각 위치에서 필터에 대한 응답

- 입력으로부터 뽑아낼 패치의 크기 : 필터를 적용한 이후의 출력의 크기 (output_depth)
- 특성 맵의 출력 깊이 : 합성곱으로 계산할 필터의 크기 (height x width)

```
Conv2D(output_depth, (window_height, window_width))
```

합성곱 연산-예시

$$* \frac{1}{273} \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} =$$

- **Gaussian Filter :** 잡음 제거

합성곱 연산-예시

*

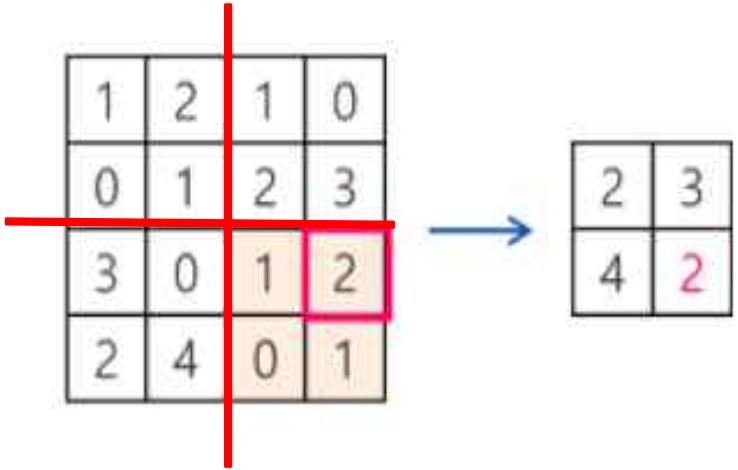
+1	+2	+1
0	0	0
-1	-2	-1

=

- **Sobel Filter :** 해당 방향의 **edge** 성분 추출

맥스 풀링

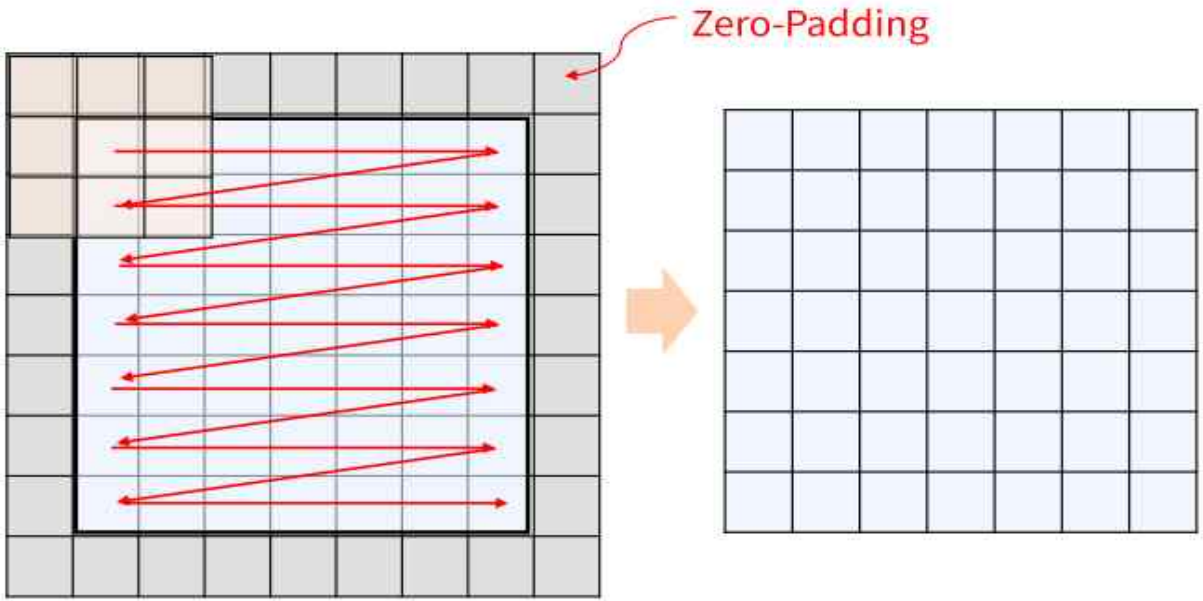
- Pooling : 강제적으로 특성 맵을 **다운 샘플링 (subsampling)** → 특성 맵의 가중치 개수를 감소
- Max Pooling : 정해진 구역 안에서 **가장 큰 값만 다음 층으로 넘기고 나머지는 버림**



- Max Pooling 2D : stride 2
- stride : 합성곱 연산에서 커널을 이동시키는 거리
  크게 하면, 출력의 크기가 작아짐

패딩

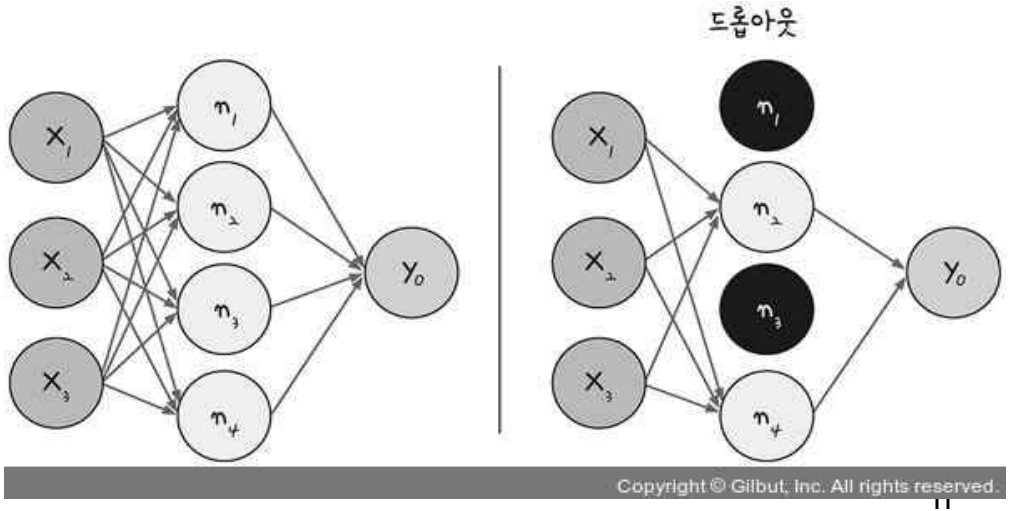
- 합성곱 연산시, 필터를 거치며 이미지의 크기가 줄어드는 문제 발생
- 크기가 원본 이미지의 상하좌우에 **Zero-Padding** 적용



드롭아웃, 플래튼

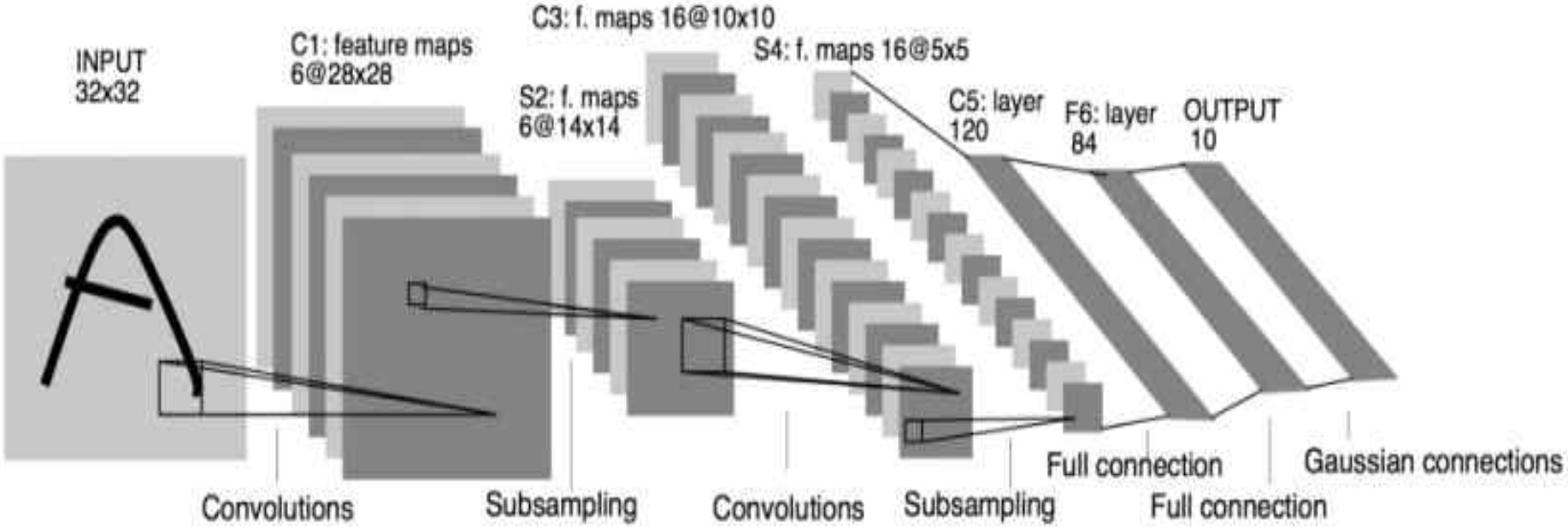
- 노드, 층이 많아진다고 반드시 학습 성능이 개선되지는 않음 : **과적합**
- 딥러닝 모델의 성능 개선 : 과적합의 문제를 얼마나 효과적으로 해결하는가
- **드롭아웃 : 은닉층에 배치된 노드 중 일부를 임의로 꺼줌**

  학습 데이터에 지나치게 치우쳐서 학습되는 과적합 방지

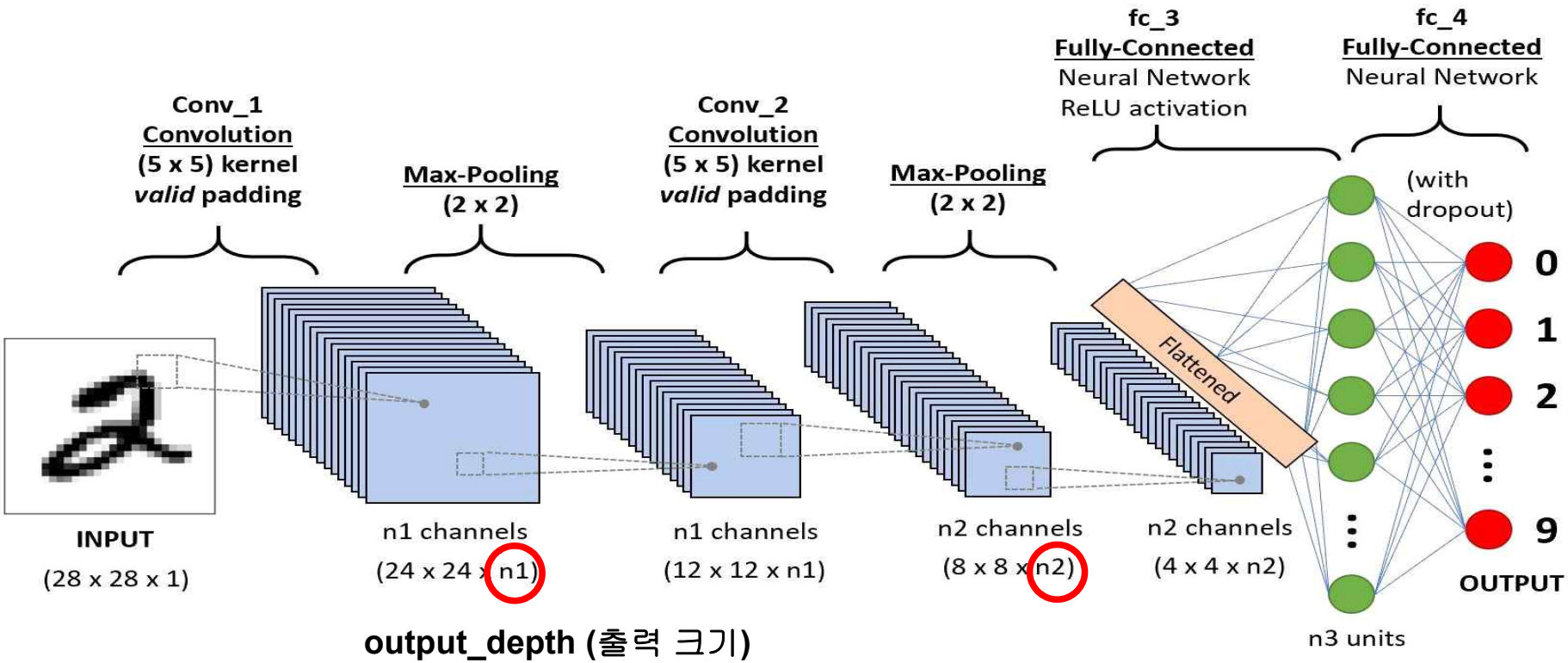


- `model.add(Dropout(0.5))`

컨브넷 구조


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

컨브넷 구조


Conv_1
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
Conv_2
Convolution
(5 x 5) kernel
valid padding
Max-Pooling
(2 x 2)
fc_3
Fully-Connected
Neural Network
ReLU activation
fc_4
Fully-Connected
Neural Network
(with dropout)
Flattened
INPUT
(28 x 28 x 1)
n1 channels
(24 x 24 x n1)
n1 channels
(12 x 12 x n1)
n2 channels
(8 x 8 x n2)
n2 channels
(4 x 4 x n2)
n3 units
0
1
2
9
OUTPUT
output_depth (출력 크기)

n

컨브넷의 특징

컨브넷 (합성곱 층)

- 특정 크기의 필터를 이용, **지역 패턴**을 학습

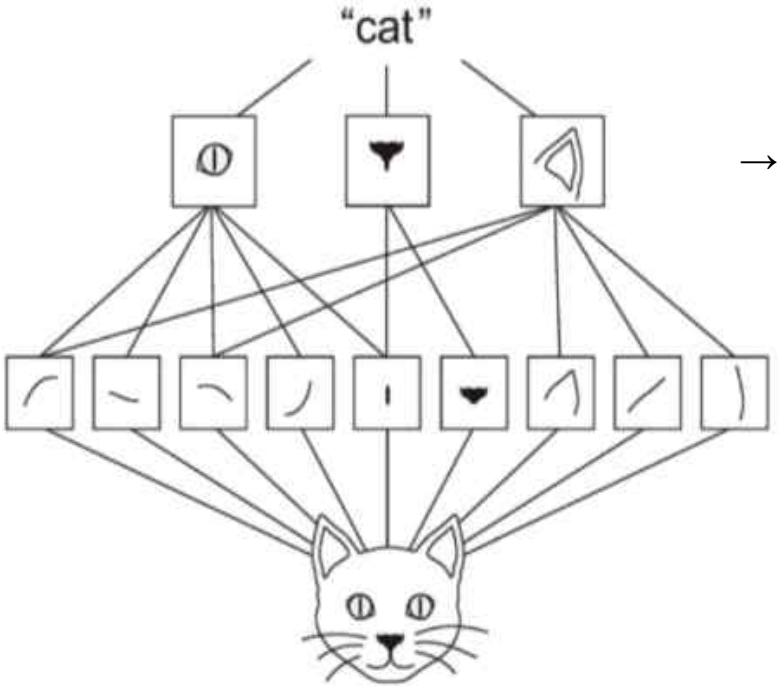
완전 연결 층 (Dense)

- 입력 특성 공간의 **전역 패턴** (모든 픽셀에 걸친 패턴)을 학습

- **학습 패턴의 평행이동 불변성**
  - 왼쪽 모서리에서 패턴 A 학습 → 오른쪽 모서리에서 패턴 A 인식 가능
  - 완전 연결 층 : 새로운 위치에서 나타나 패턴은 다른 패턴으로 인식

→ 일반화 능력 (우리가 보는 세상은 평행이동으로 인해 다르게 인식되지 않음)

컨브넷의 특징-공간적 계층 구조 학습

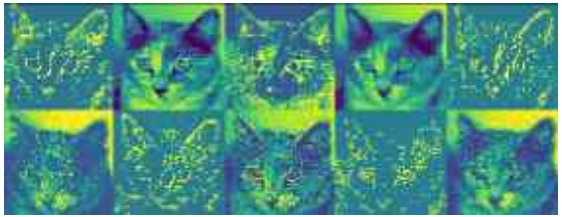


→ **구체적**, 국부적인 지역 패턴 학습 (타깃에 대한 정보)

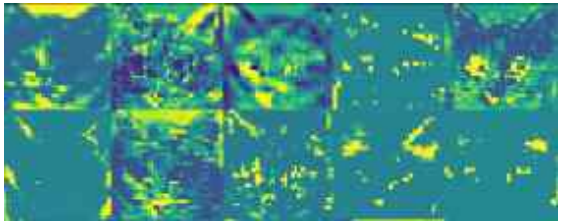
→ **일반적**, 작은 지역 패턴 학습 (시각적 정보)

- 학습된 **일반적** 표현 : 재사용 가능 **(전이학습의 근거)**
- 다른 분류 문제에 적용 가능 **(일반화)**

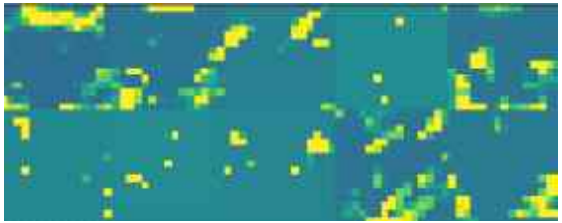
컨브넷 학습 시각화



- **초기 층** : 원본의 거의 모든 시각적 정보 유지

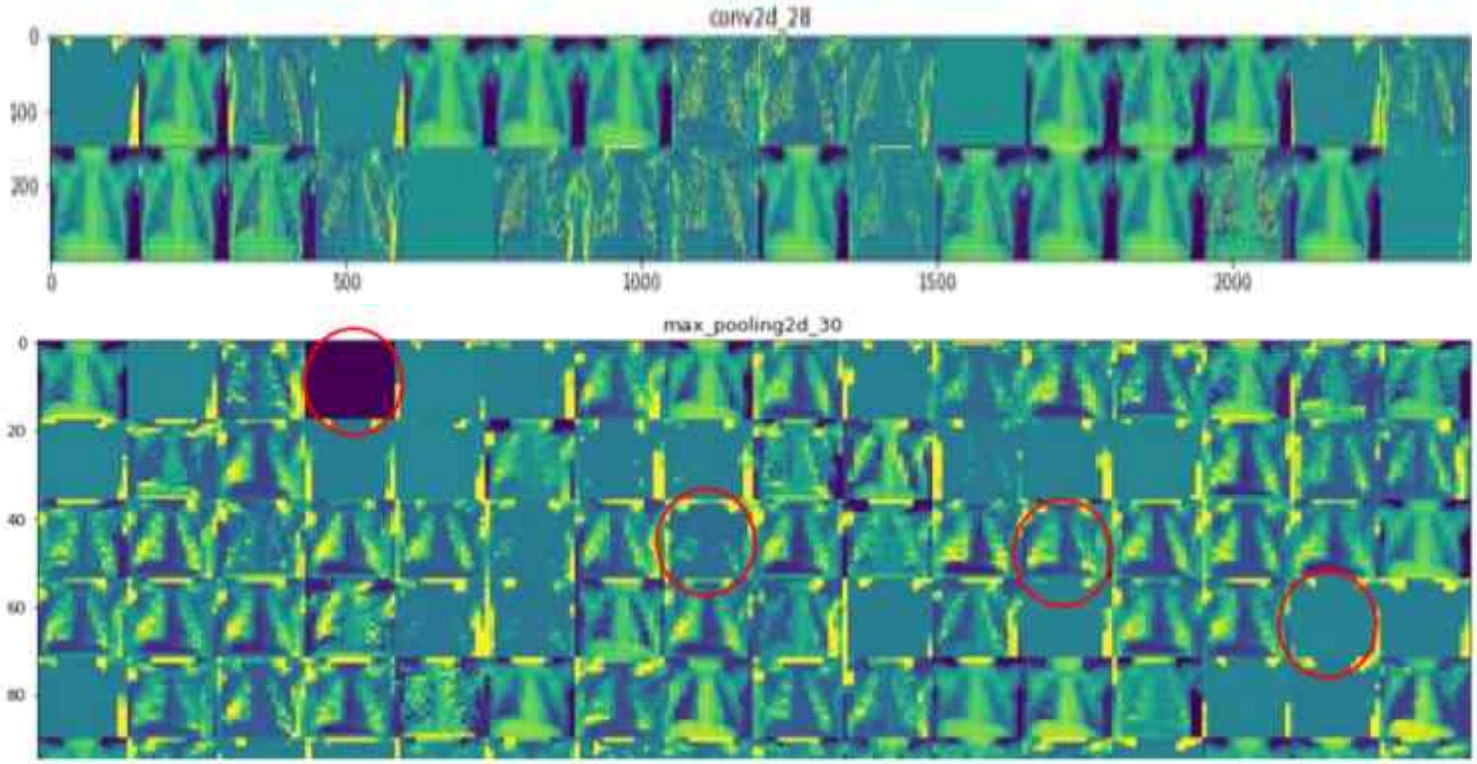


- **중간 층** : 고수준, 구체적 개념 인코딩 (귀, 눈)
  유용한 표현, 패턴을 학습해나가는 과정

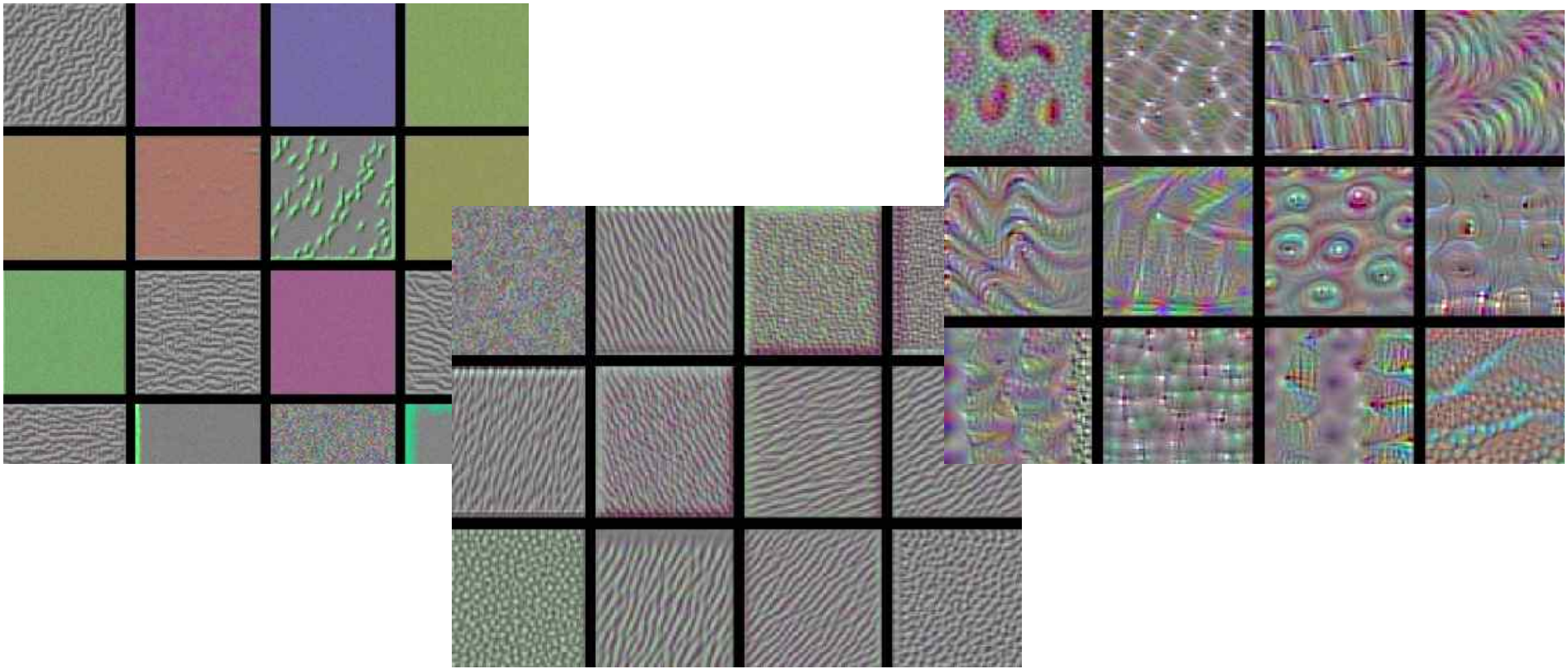


- **마지막 층** : 추상적, 시각적으로 이해하기 어려워짐
  타깃(클래스)에 관한 정보

컨브넷의 학습 시각화


conv2d_28
0
100
200
0
500
1000
1500
2000
max_pooling2d_30
0
20
40
60
80

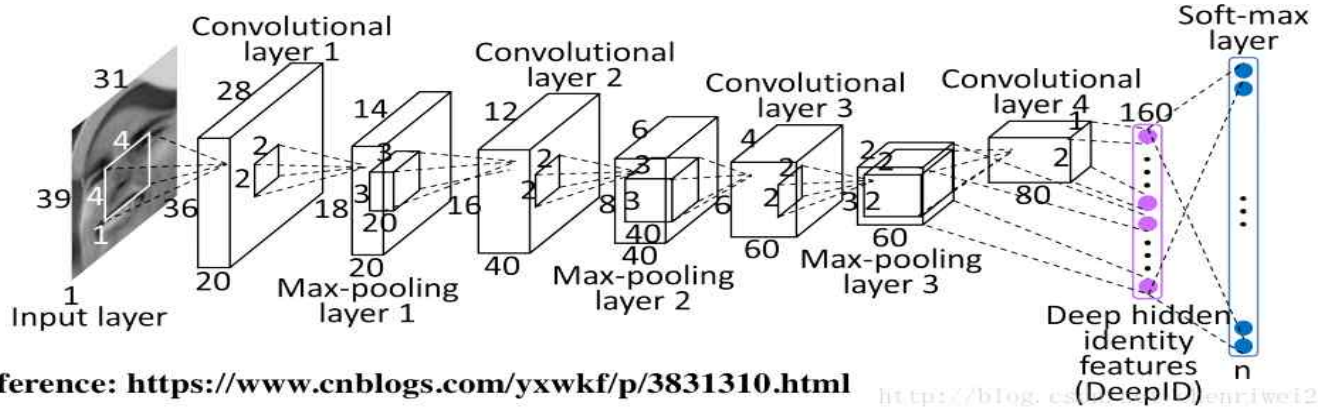
컨브넷 필터 시각화



n

컨브넷의 학습

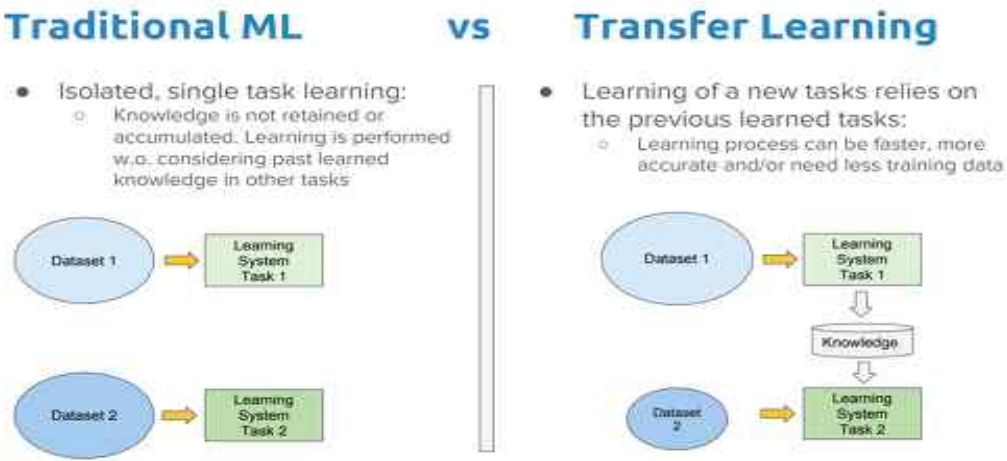
- 층에서 추출한 특성, 패턴은 **층의 깊이가 깊어짐에 따라 점점 더 추상적**
- 층의 깊이가 깊어짐에 따라 시각적 정보는 감소, **타깃에 관한 정보 증가**
- **지역적, 평행이동으로 변하지 않는 특성** 학습 : 지각 문제에 있어 데이터를 효율적으로 사용
- 이미지의 반복적, 연속적 변환 → **유용한 정보가 강조, 개선**
- 컨브넷 : 입력되는 원본 데이터의 정보 정제, 변환 파이프라인



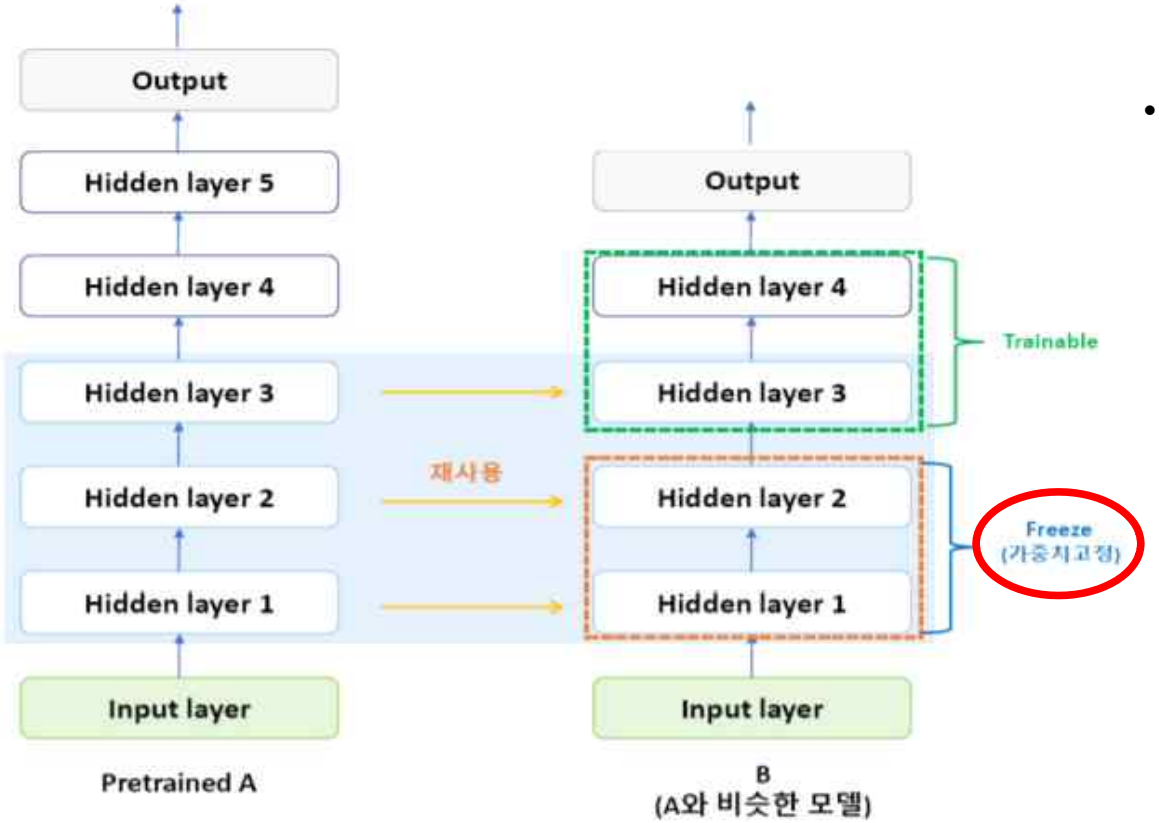
전이 학습-사전 훈련된 컨브넷

사전 훈련된 컨브넷

- 대규모 이미지 분류 문제를 위해 대량의 데이터셋에서 **미리 훈련되어 저장**된 네트워크
- 사전 훈련된 컨브넷에서 학습된 특성을 **다른 분류 문제에 적용** (유연성, 일반화)
- 새로운 문제가 원래 작업과 완전히 다른 클래스에 대한 것이더라도 유용한 성능을 보임 (원본 데이터셋이 충분히 크고 일반적이라면)

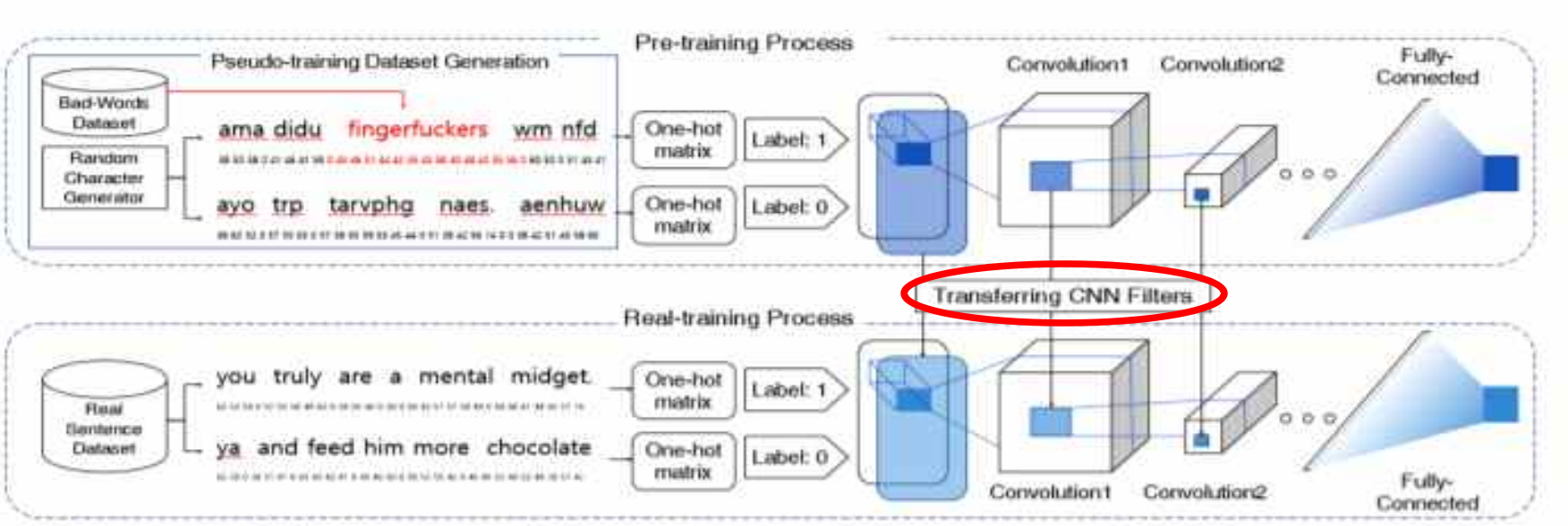


전이학습-사전 훈련된 컨브넷



- 대량의 분류 문제에 사용된 컨브넷 네트워크의 일부분을 새로운 분류 문제에 적용

전이 학습-사전 훈련된 컨브넷



- 사전 훈련된 컨브넷의 필터를 새로운 분류 문제에 적용 (이미 학습된 필터의 가중치 이용)

전이 학습-특성 추출

- 사전에 학습된 네트워크의 표현을 재사용 → 새로운 샘플에서 유용한 특성, 패턴 추출

  → 추출된 특성을 바탕으로 새로운 분류기 훈련
- **사전 훈련된 컨브넷** : 합성곱 기반 층 / 완전 연결 층 (분류기)
- **합성곱 기반 층만 재사용**

  - 합성곱 층 : 일반적인 표현 학습

  - 완전 연결 층 : 특정 분류 문제의 클래스 집합에 특화된 표현, 위치 정보 x
- **합성곱 기반 층의 하위 층 몇 개만 재사용**

  - 하위 층 : 일반적인 표현, 특성 학습 (에지, 색깔, 질감)

  - 상위 층 : 추상적인 개념 (눈, 귀, 코)

전이 학습

```
from tensorflow.keras.applications import VGG16
from keras import models
from keras import layers

conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(150, 150, 3))

model = models.Sequential()
model.add(conv_base)

model.add(layers.Flatten())
model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

conv_base.trainable = False
```

- 사전 훈련된 네트워크(VGG16)의 합성곱 기반 층 불러오기
- 합성곱 기반 층을 새로운 모델에 추가
- 새로운 모델의 완전 연결 분류기
- **합성곱 기반 층 동결 (사전 훈련)**

미세 조정

- 동결 모델 (사전 훈련된 네트워크의 합성곱 기반 층)의 **층 몇 개를 동결 해제**
  → 모델에 새로 추가한 층 (완전 연결 분류기)와 함께 훈련, 가중치 업데이트

1. 사전 훈련된 기반 네트워크 위에 새로운 네트워크 추가 (완전 연결 분류기)
2. 기반 네트워크 동결 (훈련 x, 가중치 업데이트 x)
3. 기반 네트워크 일부 층의 동결 해제 (훈련 o, 가중치 업데이트 o)
4. **동결을 해제한 기반 네트워크의 층, 새로 추가한 층을 함께 훈련**

- **보통 최상위 2~3개 층만 미세조정 하는 것이 효과적**