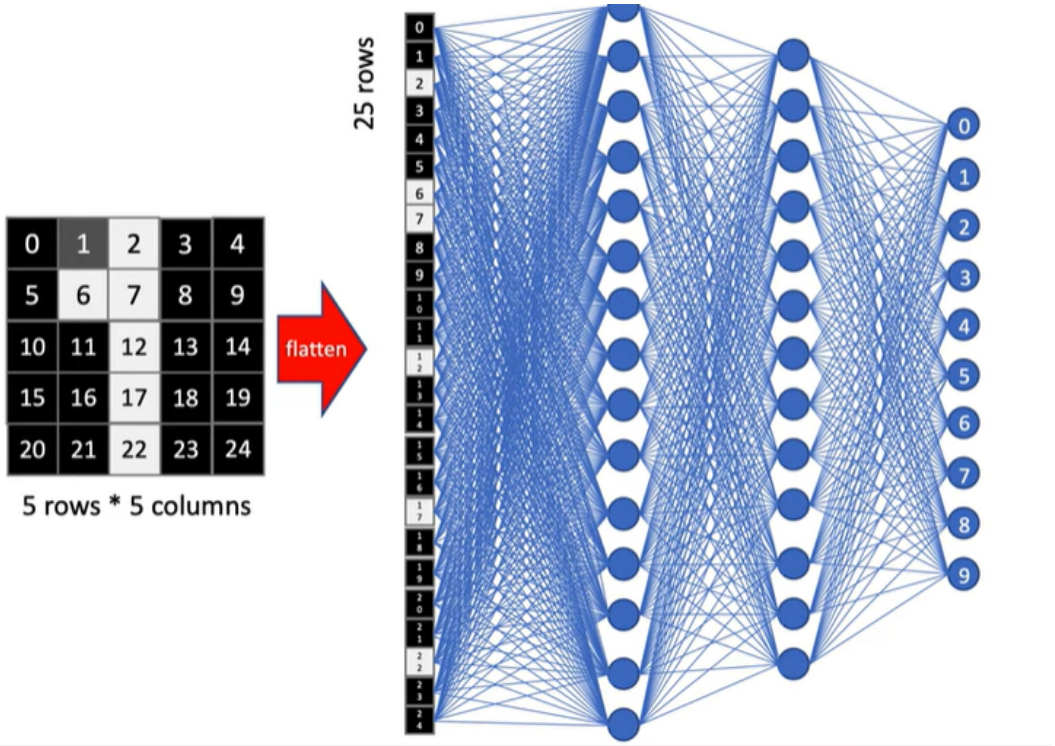
합성곱 신경망 CNN

기존 딥러닝 기법



DNN

2차원 행렬의 이미지 데이터를 1차원으로 나열해 이미지를 분석하는 방식.

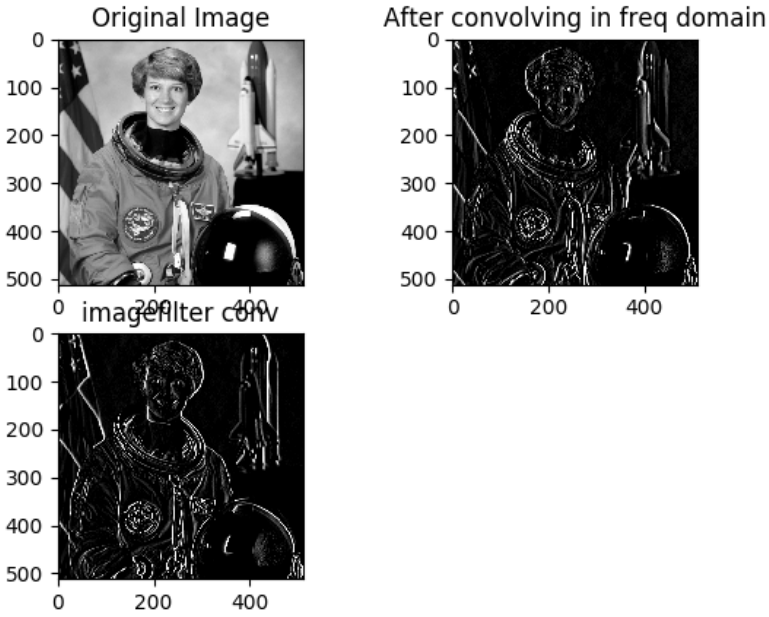
이러한 이유 때문에 1차원에서 정확히 데이터의 이미지를 분류할 수 없는 단점이 존재

DNN의 경우 모든 차원의 함수를 표현이 가능(Universal Approximation Theorem)해 모든 error를 0으로 만드는 너무 높은 자유도를 가지는 문제점이 발생해 과대적합의 원인으로 작용함.

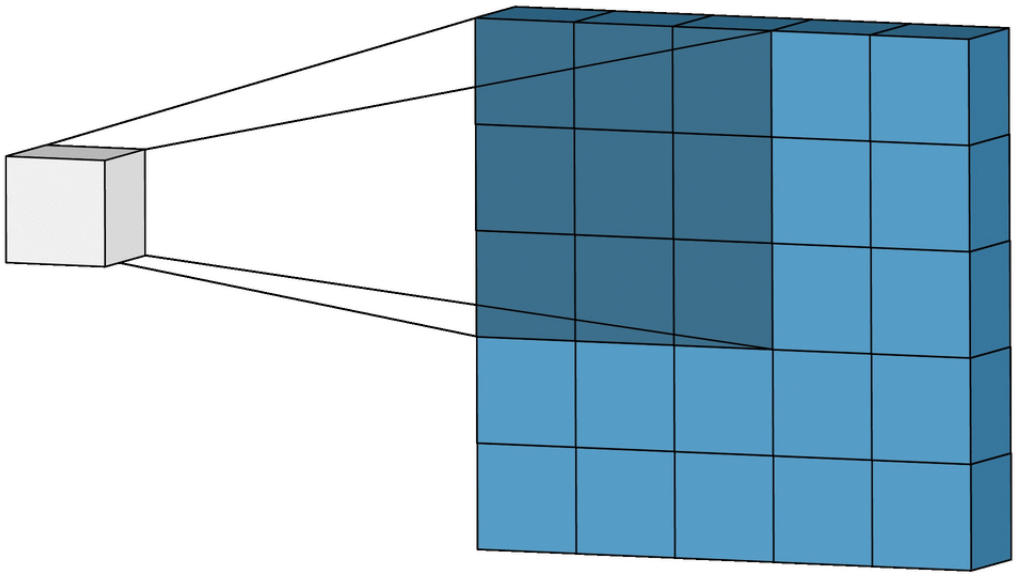
모든 Layer가 Fully Connected Layer 형태를 띄므로 학습속도가 과도하게 오래걸리는 문제점도 존재함.

CNN 용어

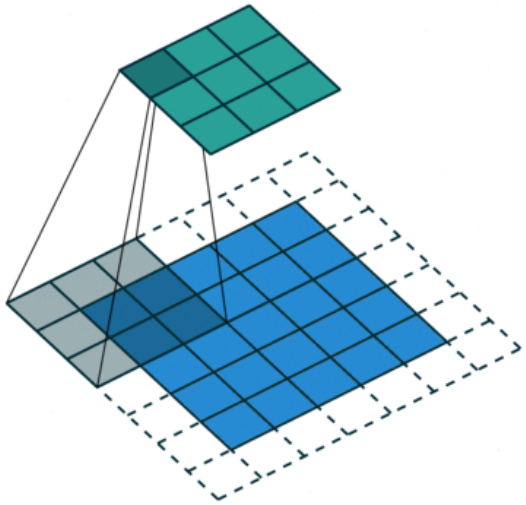
Convolution - 이미지의 윤곽을 따는 과정


Original Image
After convolving in freq domain
imagefilter conv
0
100
200
300
400
500
0
200
400

Conv 과정 예시

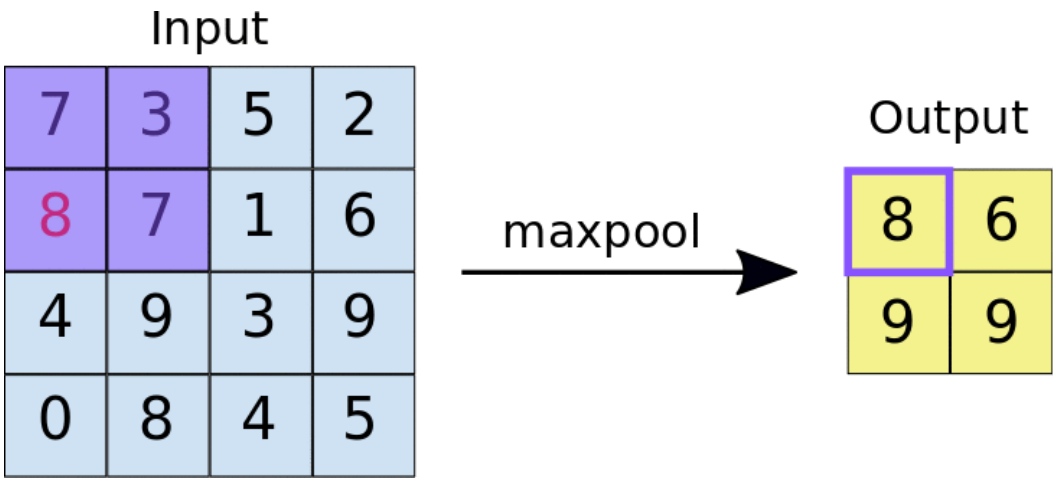


Padding - 출력 크기를 조정할 때 사용하는 방법

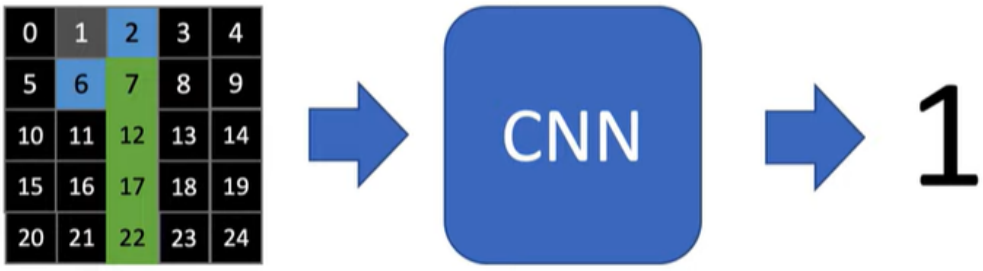


Stride - filter의 이동 칸 수를 의미

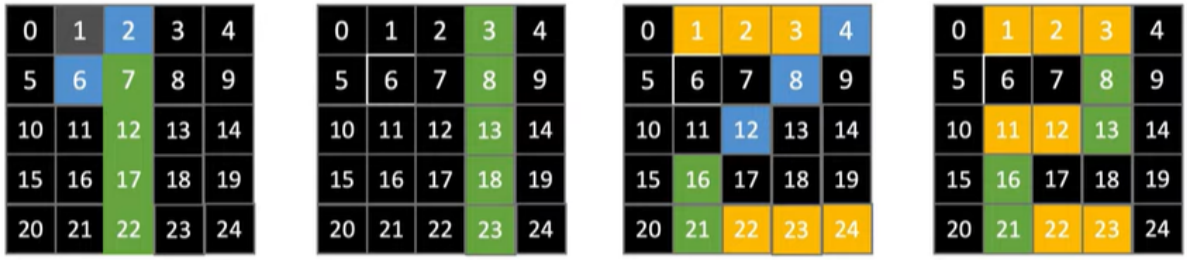
Pooling - 이미지의 특징점을 뽑아 차원수를 줄이는 것(채널은 그대로 유지) (이미지 - stride : 2)



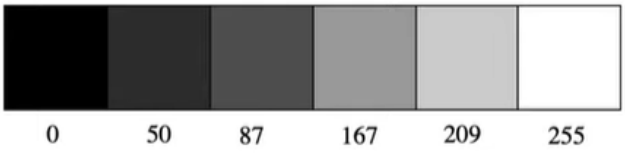
CNN

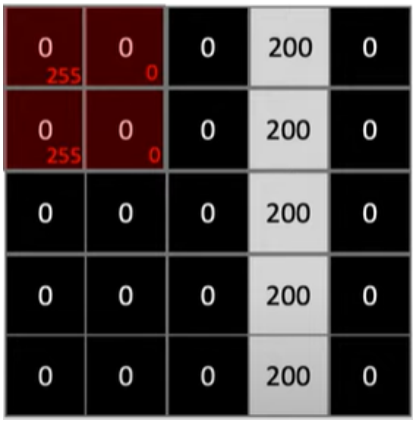


기존의 딥러닝 기법과는 다르게 2차원 행렬에서 특징을 먼저 찾는다.

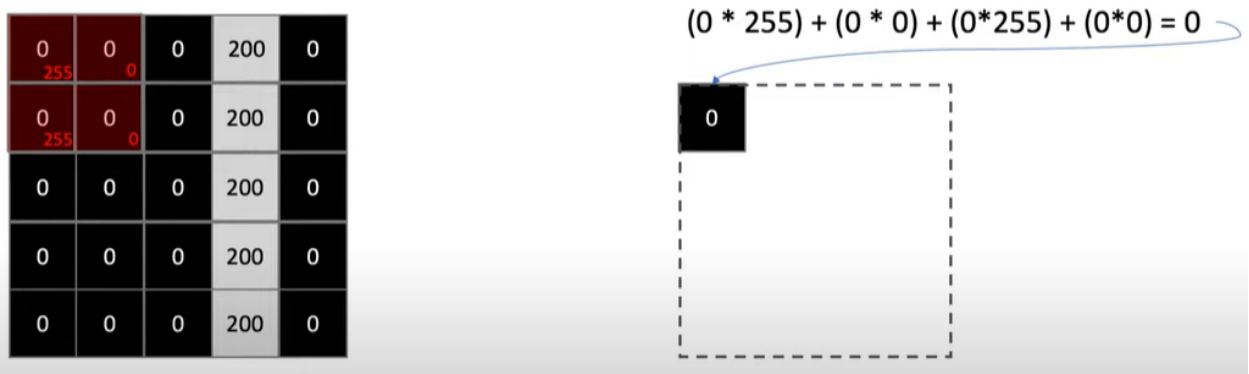


이처럼 2차원 행렬에서 수직선과 수평선을 따로 구분짓는다

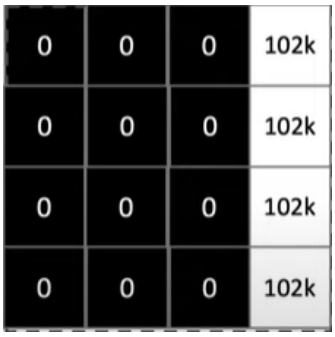




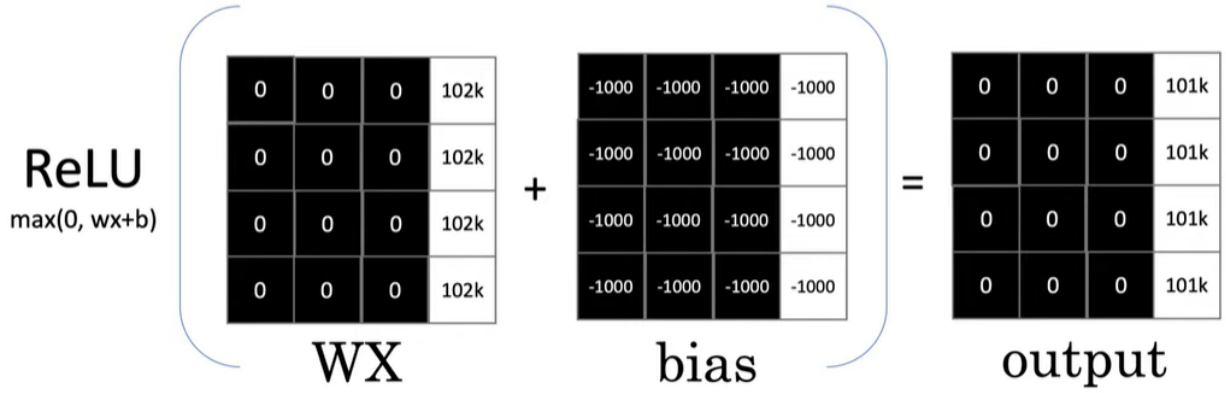
위의 사진처럼 0-255까지의 숫자를 부여하여 명암을 나타내는 척도로 이용한다.



각 타일(4픽셀)의 명암값과 0을 곱해준 후 타일의 합을 입력한다.(픽셀을 다 채울때까지 반복)



결과적으로 위와 같은 그림으로 나타내지며 숫자 1을 나타내는 것을 확인할 수 있다.

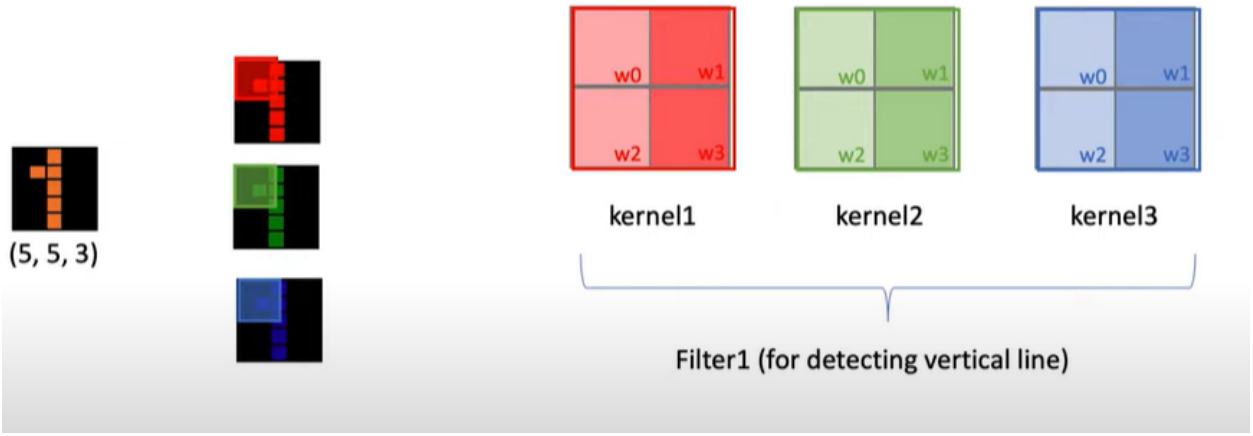


ReLU = Wx + b = output

*이는 흑백 사진의 경우에 사용하는 방법이다.

컬러 사진의 경우

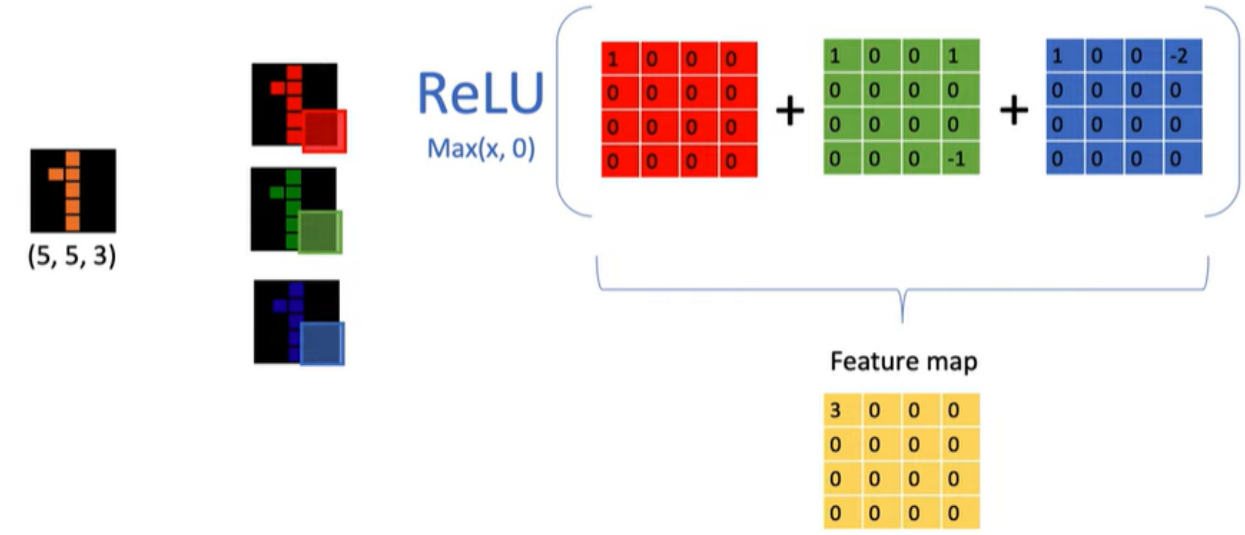
Kernel과 Filter



***1 은 사실 1,1,1의 모든 색을 합친 색이다.**

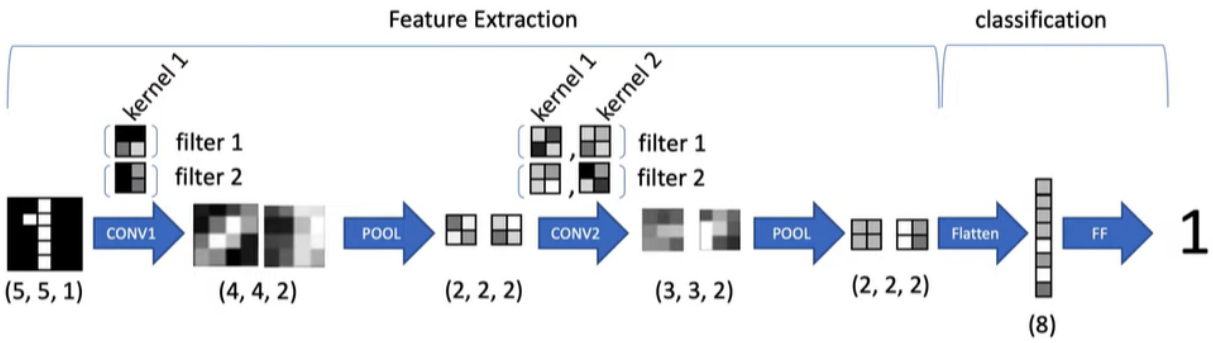
Kernel - 2차원 행렬에서 돌아다니는 타일

Filter - 하나의 특징을 찾기 위해서 이미지의 개수 만큼 kernal을 가진 집합체



CNN 분석 과정

흑백

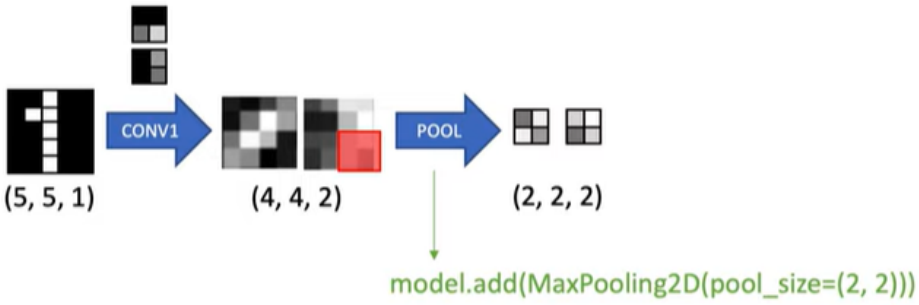


흑백 이미지이기 때문에 한 개의 행렬이 입력값으로 Filter에 한 개의 Kernel만 존재

1.

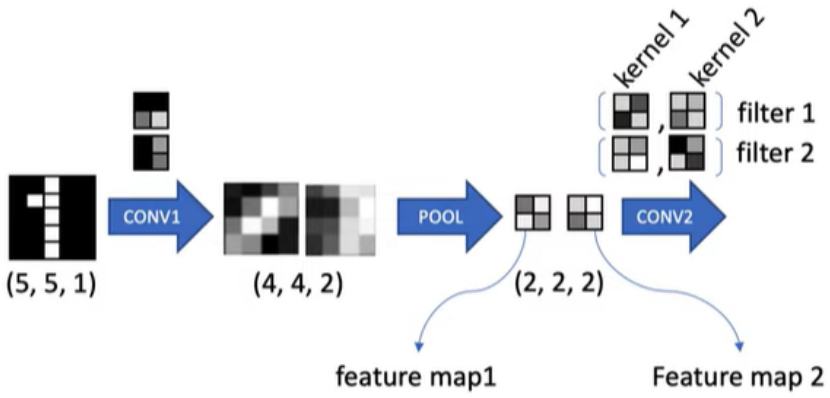

```
model.add(Conv2D(2, kernel_size=(2,2), activation='relu',input_shape=(5,5,1)))

Conv2D - 2차원 데이터를 처리하는 함수
kernel_size=(2,2) - kernal의 사이즈가 2*2 행렬
activation='relu' - 활성화 함수로 ReLU 적용
input_shape=(5,5,1))) - 흑백 이미지 이므로 5*5 행렬 1개만 존재
```

2.



```
model.add(MaxPooling2D(pool_size=(2,2)))

Maxpooling2D - 각 타일의 최대값만 뽑아 축소시키는 함수
pool_size - Pooling을 거친 행렬의 크기를 2*2행렬로 축소시키는 것
행렬의 크기를 줄여 파라미터 개수와 계산시간을 줄이고 과대적합을 방지한다.
```

3.



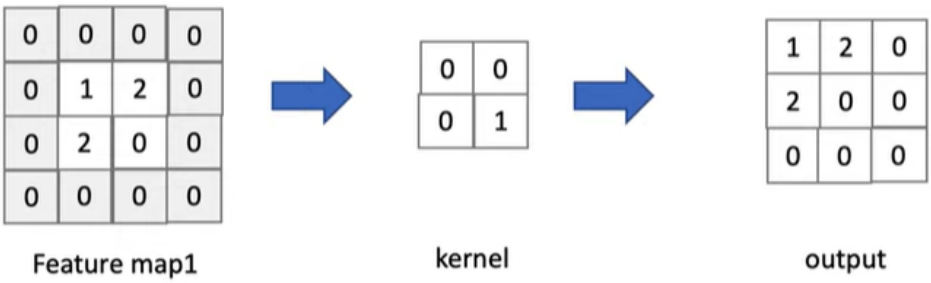
*Conv2D를 진행해야 하지만 Feature Map의 크기가 너무 작아 Filter가 움직일 공간이지 못하는 현상

이런 경우에 zero padding을 사용한다.

Zero Padding

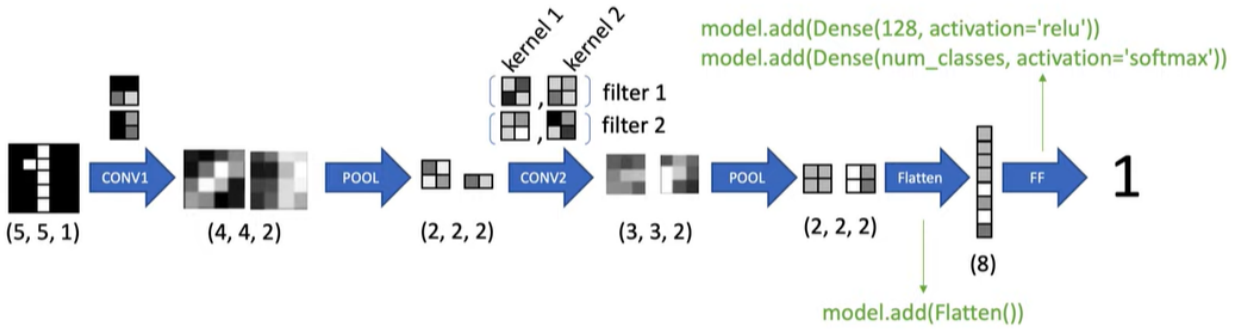
Feature Map에 0으로 둘러쌓은 패딩을 입혀 Conv2D를 다시 적용한다.

이 방법을 이용하면 외각에 대한 정보를 모델에게 알려주고 Conv2가 움직일 수 있게 해준다.



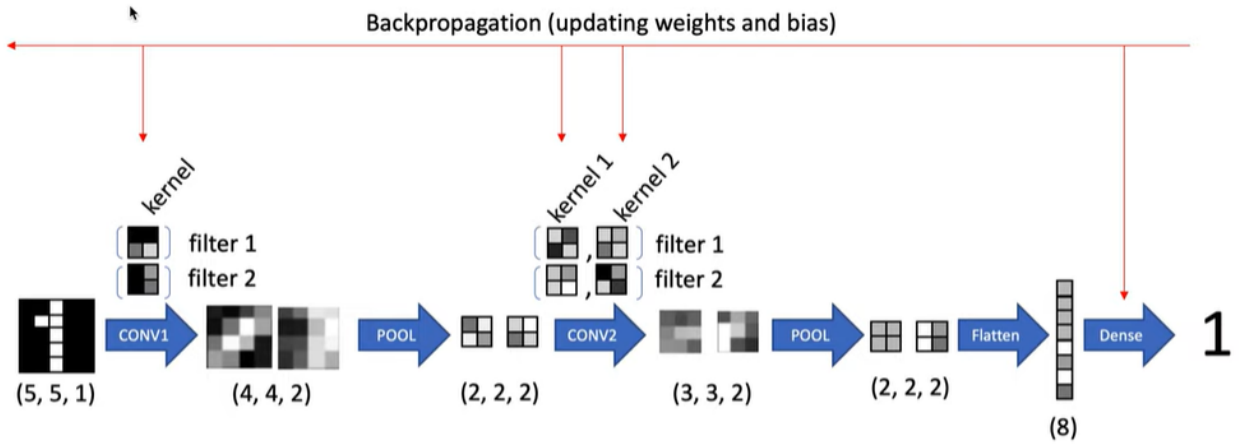
output = Feature Map의 Conv2D 진행 한 kernal + bias → ReLU 적용

4.




```
model.add(Flatten()) - 모델 1차원으로 flat적용

model.add(Dense(128, activation='relu'))
model.add(num_classes, activation='softmax'))

Dense - 노드들로 구성된 하나의 리스트
```

5.



역전파를 활용해서 가중치와 평행값을 최적화 시켜준다.

***학습 초기에는 각각의 Filter가 어떤 특징을 잡아야 할지 모르기 때문이다.**

코드 총정리

```
from IPython.display import Image

from __future__ import absolute_import, division, print_function, unicode_literals

try:
  %tensorflow_version 2.x
except Exception:
  pass

import tensorflow as tf
```

```
tf.random.set_seed(1)

import tensorflow as tf
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras import backend as K
from tensorflow.keras.losses import categorical_crossentropy
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Preprocessing

# collect MNIST data
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

# sample to show gray scale values
print(x_train[0][8])

# sample to show labels for first train data to 10th train data
print(y_train[0:9])

print("test data has " + str(x_test.shape[0]) + " samples")
print("every test data is " + str(x_test.shape[1])
      + " * " + str(x_test.shape[2]) + " image")

# reshape data
import numpy as np
x_train = np.reshape(x_train, (60000,28,28,1))
x_test = np.reshape(x_test, (10000,28,28,1))

print(x_train.shape)
print(x_test.shape)

# normalization
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

gray_scale = 255
x_train /= gray_scale
x_test /= gray_scale

# change label to one hot encoding
num_classes = 10
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)

### We will implement below CNN

Image(url= "https://raw.githubusercontent.com/captainchargers/deeplearning/master/img/practice_cnn.png", width=800, height=200)

model = Sequential()
model.add(Conv2D(16, kernel_size=(5, 5),
                 activation='relu',
                 input_shape=(28,28,1),padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, kernel_size=(5, 5), activation='relu',padding='same'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))

model.summary()

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

callbacks = [EarlyStopping(monitor='val_accuracy', patience=2, restore_best_weights=False),
             ModelCheckpoint(filepath='best_model.h5', monitor='val_accuracy', save_best_only=True)]

model.fit(x_train, y_train,
```

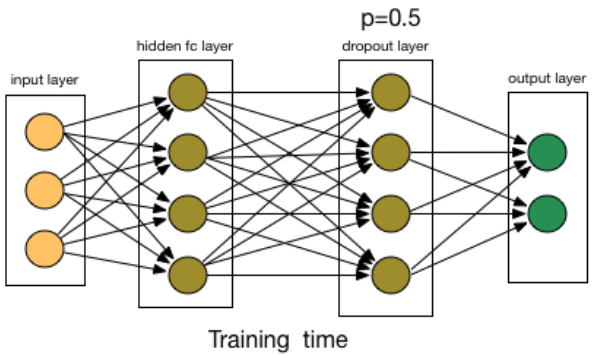
```
          batch_size=500,
          epochs=10,
          verbose=1,
          validation_split = 0.1,
          callbacks=callbacks)

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

```
import os
import PIL
import PIL.Image
import tensorflow_datasets as tfds
import tensorflow as tf
from tensorflow import keras

import pathlib
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = keras.utils.get_file(origin=dataset_url, #url 주소에서 파일 불러오기
                                fname='flower_photos', #파일이름 설정
                                untar=True) #압축해제 여부
data_dir = pathlib.Path(data_dir)

#.jpg 확장자 파일만 가져오기
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)

#장미 데이터만 모으기
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))

batch_size=32
img_height=180
img_width=180

#디렉토리에서 이미지 가져오기 - 1
train_ds = keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset='training',  #train, test를 구분짓는 파라미터
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

#디렉토리에서 이미지 가져오기 - 2
val_ds = keras.preprocessing.image_dataset_from_directory(
    data_dir,
    validation_split=0.2,
    subset='validation',
    seed=123,
    image_size=(img_height, img_width),
    batch_size=batch_size)

class_names = train_ds.class_names

plt.figure(figsize=(10,10))
for images, labels in train_ds.take(1):
    for i in range(9):
        ax = plt.subplot(3,3,i+1)
        plt.imshow(images[i].numpy().astype('uint8'))  #numpy() - tensor 행렬을  numpy행렬로 변환,uint8(0~255)
        plt.axis('off')

for image_batch, labels_batch in train_ds:
    print(image_batch.shape)
    print(labels_batch.shape)
    break

#0~1 사이로 정규화
```

```
from tensorflow.keras import layers
normalization_layer = layers.experimental.preprocessing.Rescaling(1./255)

normalization_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalization_ds))
first_image = image_batch[0]
print(np.min(first_image), np.max(first_image))

#train_test split
#autotune = buffer_size 동적 변환으로 변경
#buffer_size = 한 신호에 설정된 값 만큼 데이터를 전송하는 것 (32의 배수 형태)
AUTOTUNE = tf.data.experimental.AUTOTUNE
train_ds = train_ds.cache().prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

DropOut

일부 노드를 학습에서 무시하는 것을 의미함

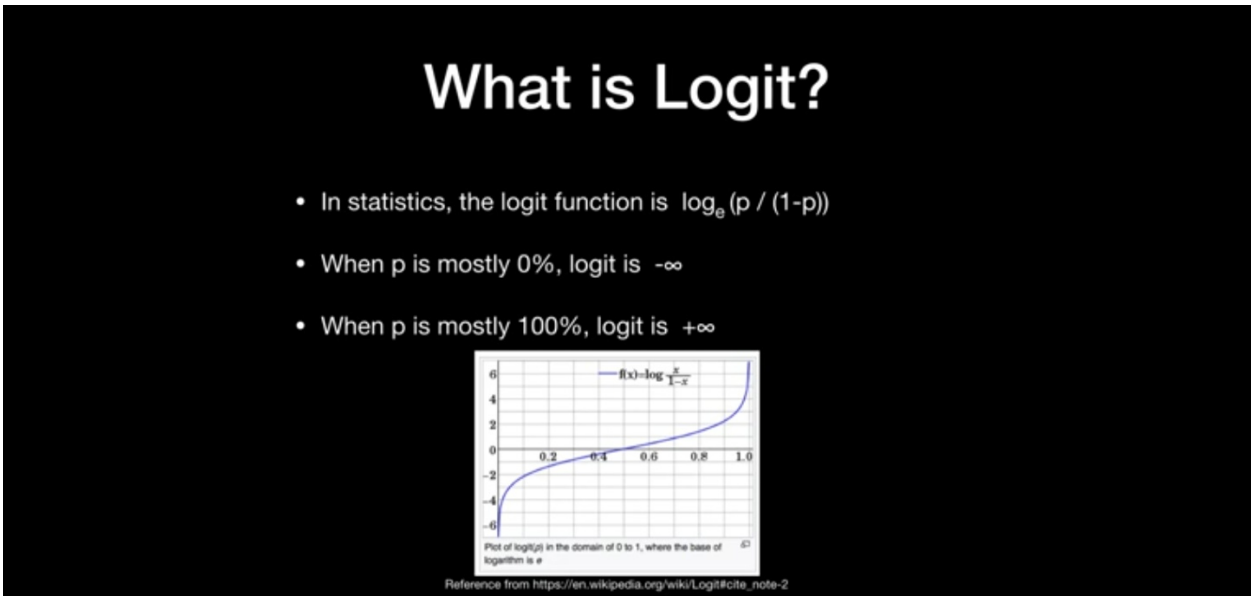


위의 좌측의 신경망 그림처럼 p의 확률로 은닉층 노드를 제거하는 것을 말한다.

이를 통해 Low variance를 얻어 과대적합을 방지할 수 있다.

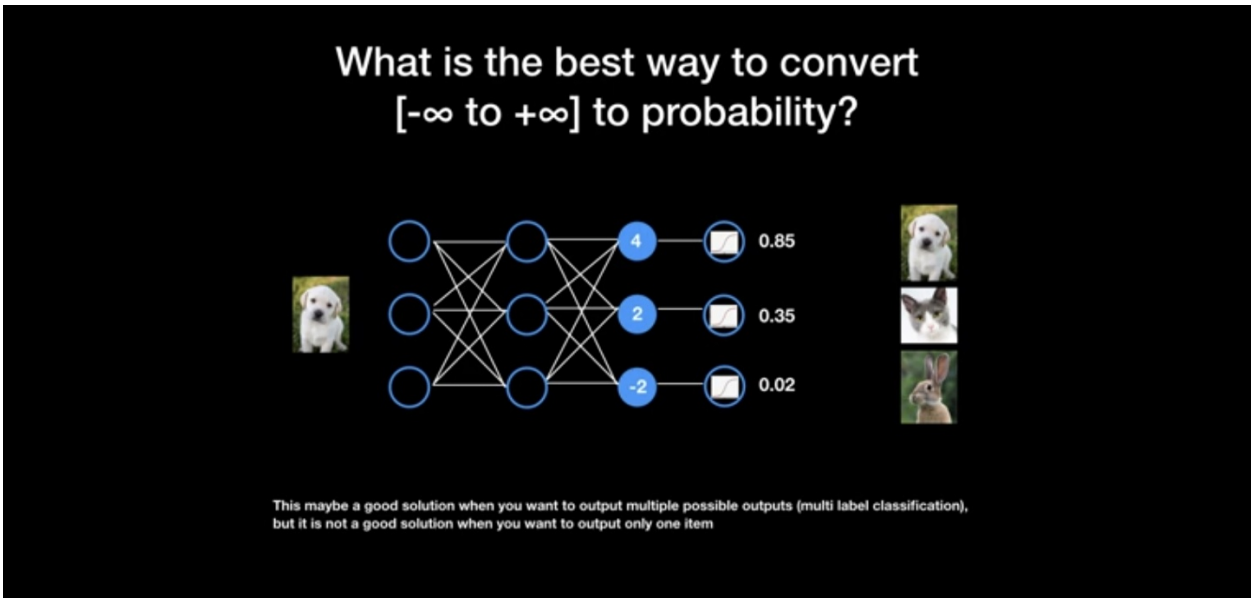
***일반적으로 train모델에만 사용하며 test에서는 더 확실한 결과를 얻기 위해 쓰지 않는다.**

logit

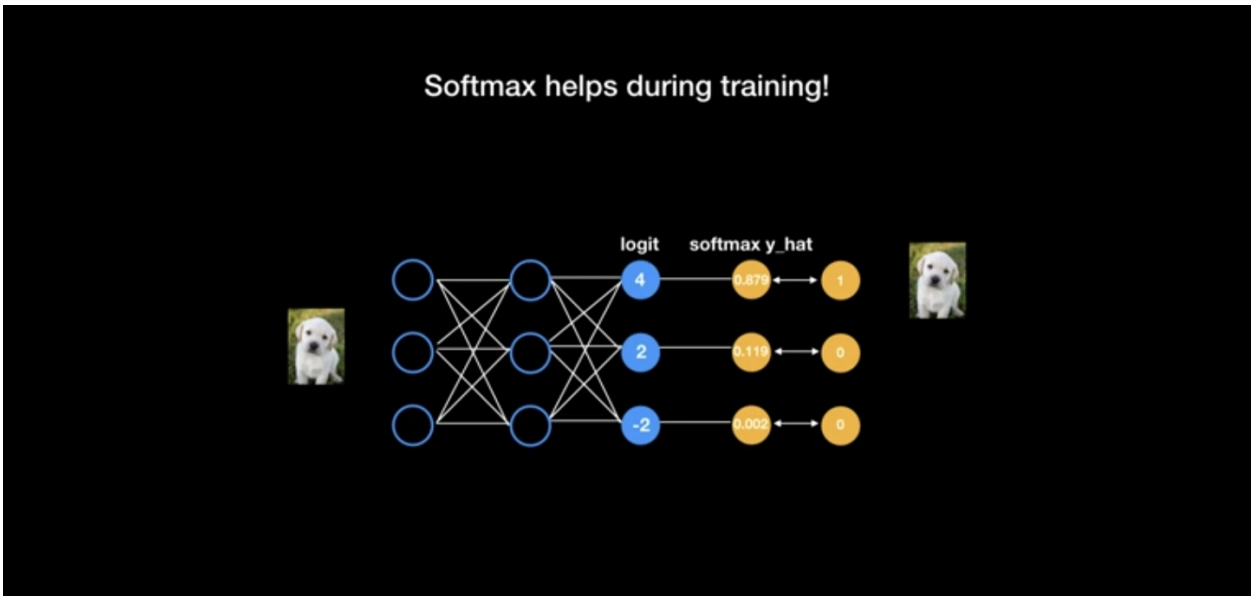


$$logit = log_e$$

이러한 이유 때문에 logit을 확률 값으로 변환하여 1차적으로 적용을 한다.

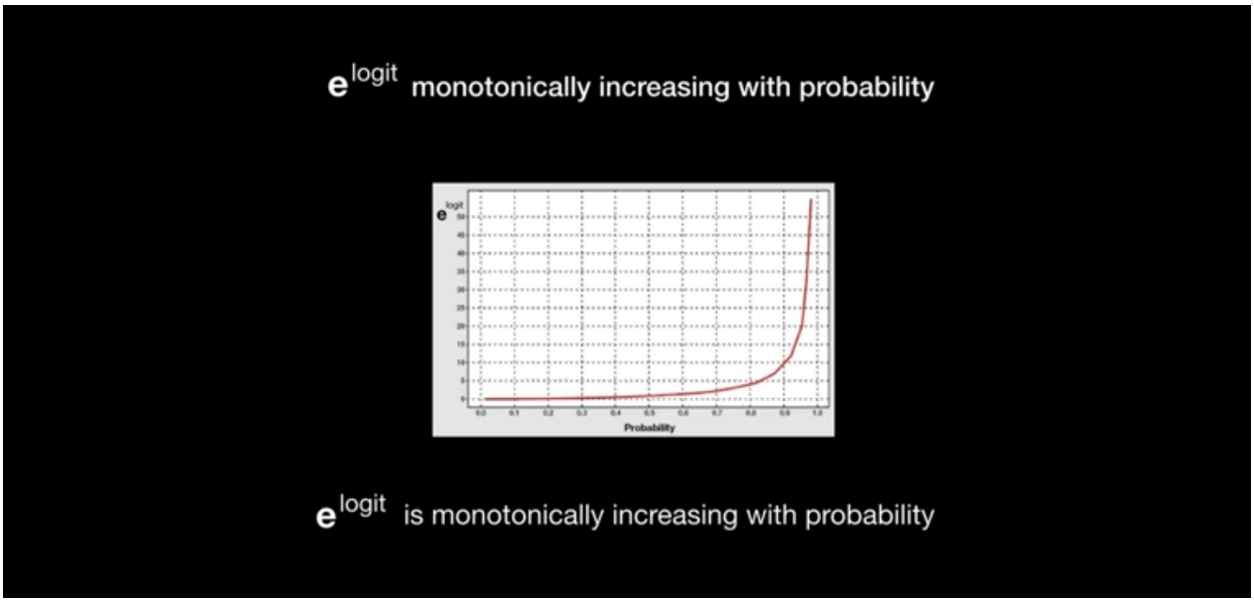


이후 softmax를 적용해 모든 값들의 확률로 변환해준다.



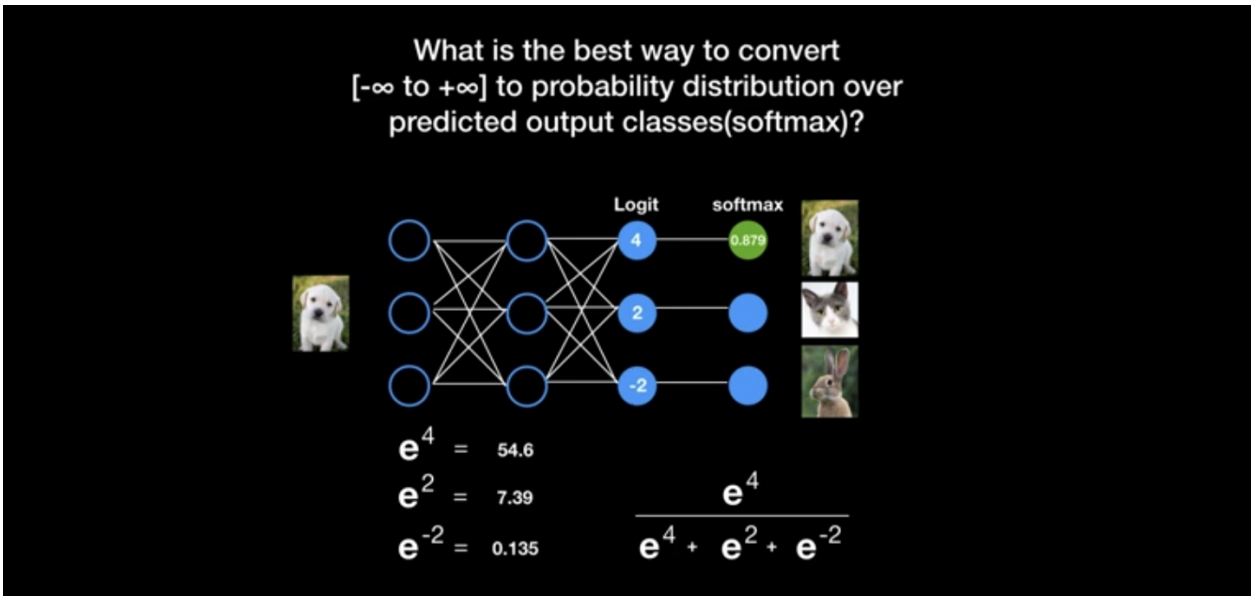
e^logit이 확률이 높아질 때마다 값이 커진다는 특성을 이용한 것.

***e^logit은 확률이 극단적으로 나타난다(높은 값은 아주 높게, 낮은 값은 아주 낮게)**

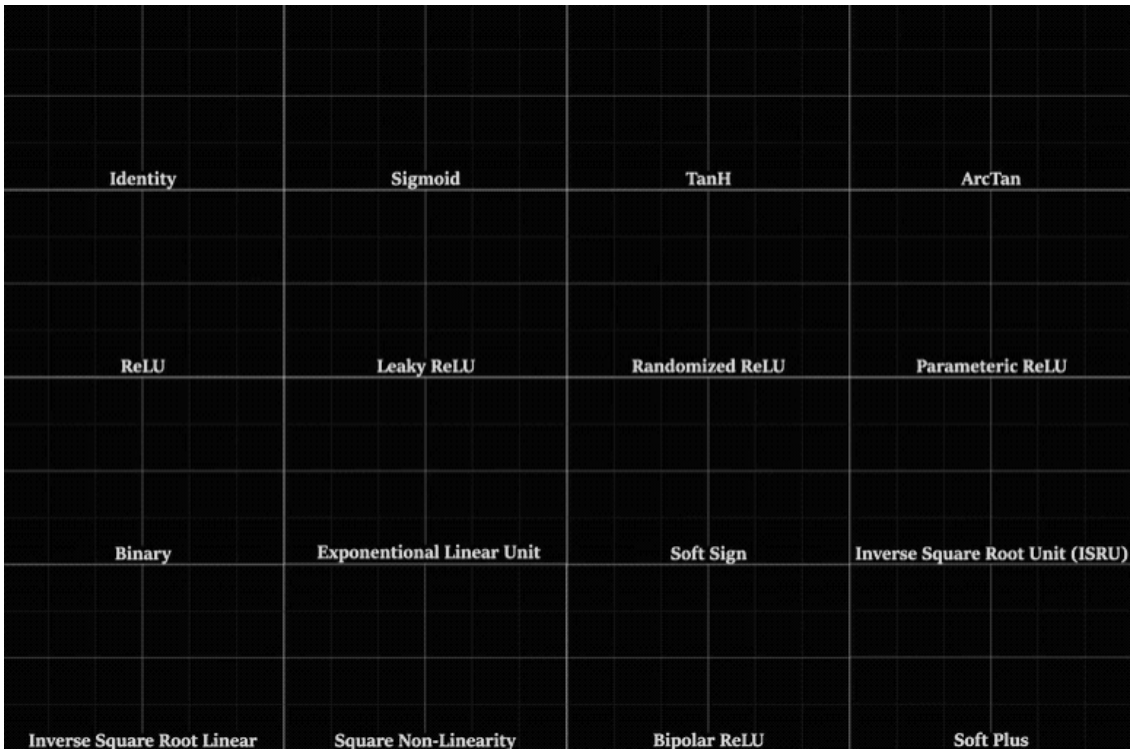


***e를 사용하는 이유 - 기반이 logit이기 때문에 계산이 간단해지기 때문**

$$e^{\log_e(p / (1-p))} = p / (1-p)$$



활성화 함수

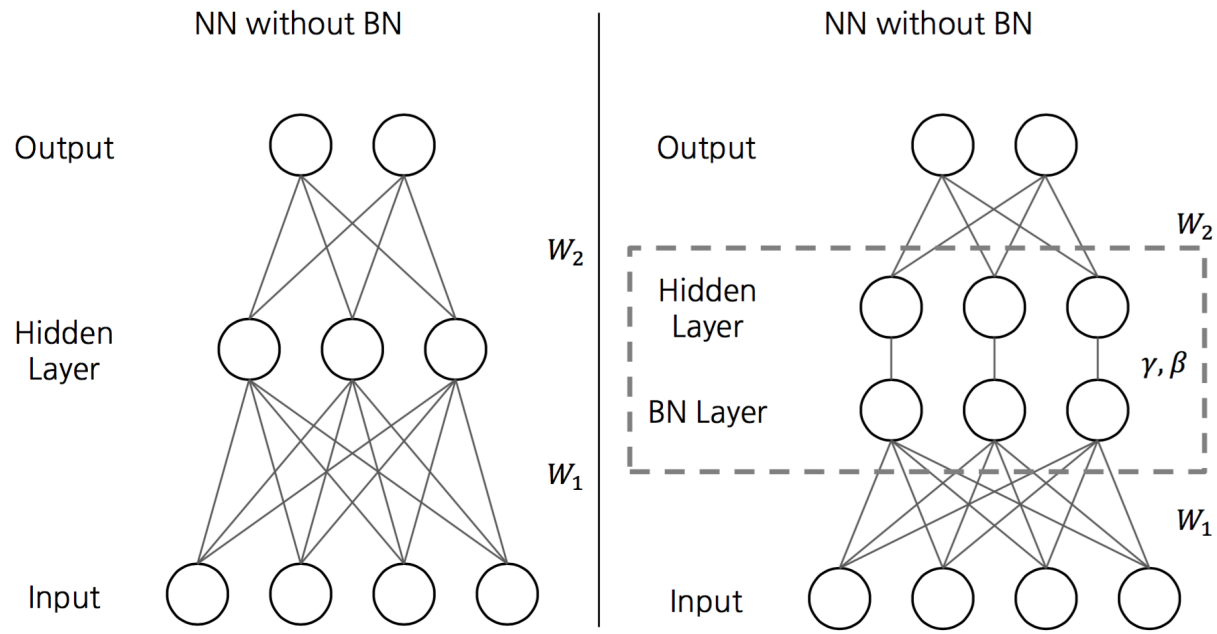


퍼셉트론의 경우 활성함 함수로 시그모이드 함수를 적용하여 사용하는데
여기서 시그모이드 함수는 출력값의 확률이 각각의 개별 노드들의 확률이기 때문에
모든 라벨의 확률의 합이 1이 되지 않아 적합하지 못한 방법이다.

Batch Norm - 배치 정규화

활성화 함수를 통해 출력값의 크기가 매우 작아져 다음 layer에서의 변화량이 급격하게 감소한다.

이를 해결하기 위해 학습 과정 자체에 정규화를 통해 학습 속도를 가속화하고 안정화해주는 것.



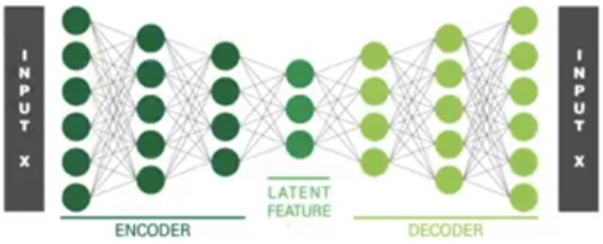

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28*28, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 10)
        self.dropout_prob = 0.5
        self.batch_norm1 = nn.BatchNorm1d(512) #배치 정규화 1
        self.batch_norm2 = nn.BatchNorm1d(256) #배치 정규화 2

    def forward(self, x):
        x = x.view(-1, 28*28)
        x = self.fc1(x)
        x = self.batch_norm1(x) #활성화 함수 이전에 정규화를 미리 해주는 것
        x = F.relu(x)
        x = F.dropout(x, training = self.training, p = self.dropout_prob)
        x = self.fc2(x)
        x = self.batch_norm2(x)
        x = F.relu(x)
        x = F.dropout(x, training = self.training, p = self.dropout_prob)
        x = self.fc3(x)
        x = F.log_softmax(x, dim=1)
        return x
```

Auto Encoder - 다층구조 학습

차원축소를 통해 표현학습 및 특징학습을 비지도 학습 형태로 학습하는 신경망

***표현학습/지도학습 이란?**

특징 탐지나 분류를 위해 필요한 특징들을 자동으로 발견하는 시스템적 기법들의 총체



위의 그림처럼 차원을 축소하여 Latent Feature를 알아내기 위함(decoder는 일종의 확인 작업)

***Auto Encoder를 사용하는 이유 - 차원의 저주(빅데이터의 저주)**

차원의 저주

저차원의 공간에서 나타나지 않던 문제들이 고차원으로 갈 때 발생하는 현상

주로 머신러닝에서는 차원이 증가할 경우 방대한 양의 데이터가 요구되는 현상을 의미.