Intro-week3

> 합성곱 신경망 사용하기

- CNN을 활용한 MNIST 이미지 인식 예제

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

# 1. MNIST 데이터셋 불러오기
(train_images, train_labels), (test_images, test_labels) = datasets.mnist.load_data()

# 2. 데이터 전처리하기
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))
train_images, test_images = train_images / 255.0, test_images / 255.0

# 3. 합성곱 신경망 구성하기
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary()

# 4. Dense 층 추가하기
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.summary()

# 5. 모델 컴파일하기
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# 6. 훈련하기
model.fit(train_images, train_labels, epochs=5)

# 7. 모델 평가하기
loss, acc = model.evaluate(test_images, test_labels, verbose=2)
```

> 데이터 전처리

```
train_images = train_images.reshape((60000, 28, 28, 1))
test_images = test_images.reshape((10000, 28, 28, 1))
train_images, test_images = train_images / 255.0, test_images / 255.0
```

reshape() : 배열의 형태 변경

0~255 사이의 값을 갖는 데이터를 0.0~1.0 사이의 값을 갖도록 255.0으로 나눠줌

합성곱 신경망 구성

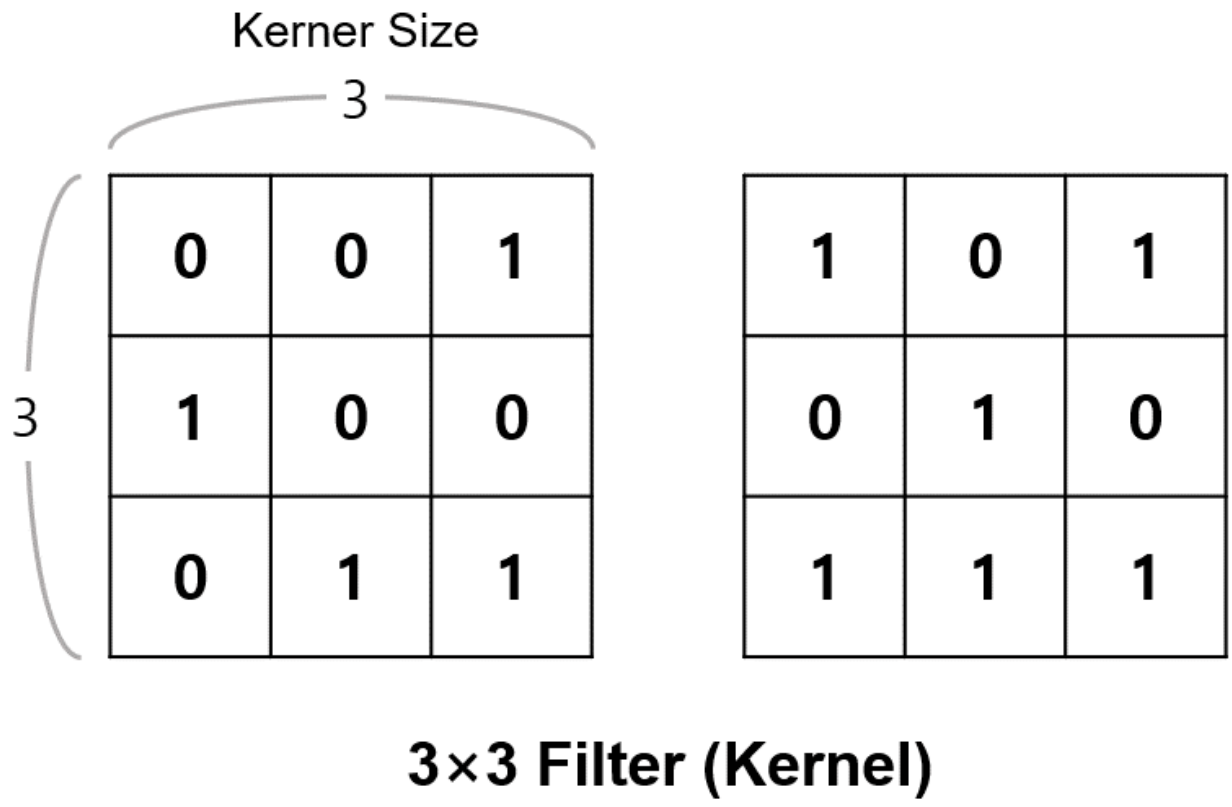
```
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))

model.summary()
```

Sequential() : 인공 신경망 챕터에서 입력층, 은닉층, 출력층에 대해서 배웠다. 케라스에서는 이러한 층을 구성하기 위해 Sequential()을 사용합니다. Sequential()을 model로 선언한 뒤에 model.add()라는 코드를 통해 층을 단계적으로 추가한다. 위의 예시는 model.add()로 층을 추가하는 예제 코드를 보여준다. 괄호 안에는 실제 층을 입력한다. 위의 예시에서는 합성곱 층 Conv2D와 MaxPooling층 MaxPooling2D를 반복해서 구성한다.

첫 번째 Conv2D 층의 첫 번째 인자 32는 **filters**값이다. 합성곱 연산에서 사용되는 filters는 이미지에서 feature를 분리하는 기능을 한다. filters의 값은 합성곱에 사용되는 필터의 종류(개수)이며, 출력 공간의 차원(깊이)을 결정한다.

두 번째 인자인 (3,3)은 **kernel_size**값이다. **kernel_size**는 합성곱에 사용되는 필터, 즉 커널의 크기이다.



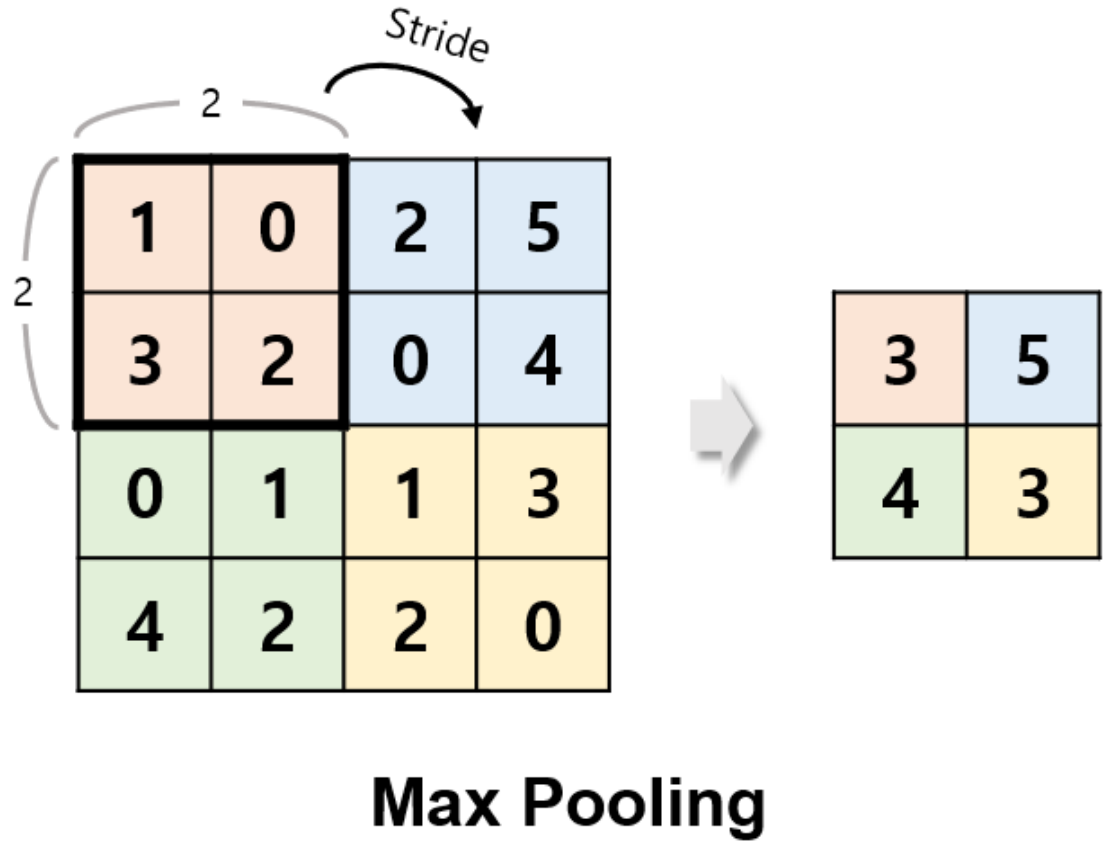
위와 같은 3x3 크기의 필터가 사용된다는 뜻이며, 합성곱 연산이 이루어지고 나면 이미지는 (28,28)크기에서 (26,26)이 된다.

Activation function은 'relu'로 지정하고, input_shape는 (28,28,1)로 설정하였다. relu함수는 은닉층에 주로 사용되는 활성화 함수이다.

Pooling : 합성곱에 의해 얻어진 Feature map으로부터 값을 샘플링하여 정보를 압축하는 과정

Max-pooling : 특정 영역에서 가장 큰 값을 샘플링하는 풀링 방식. 위의 예제에서는 풀링 필터의 크기를 2x2 영역으로 설정함.

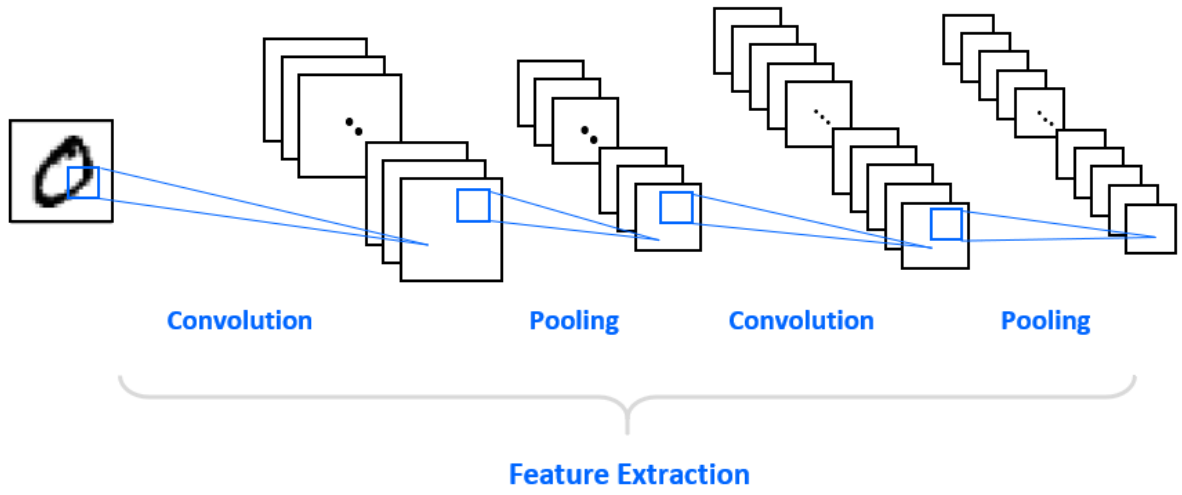
strides : 풀링 필터를 이동시키는 간격. strides를 지정해주지 않으면, 풀링 필터의 크기와 같아서 영역의 오버랩 없이 풀링이 이루어짐. 따라서 풀링이 이루어지고 나면 (26,26)크기의 이미지는 (13,13)이 된다.



summary() : 지금까지 구성한 신경망에 대한 정보를 출력할 수 있음.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D) (None, 5, 5, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D) (None, 1, 1, 64)        0
=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

다음은 위의 신경망의 구조를 그림으로 나타낸 것이다.



이러한 합성곱, 풀링 층은 **특성 추출(Feature Extraction)**을 담당하며, 전체 합성곱 신경망의 앞부분을 구성한다.

Dense층 추가

```
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax')) #softmax : 셋 이상을 분류하는 다중 클래스 분류 문제에서 출력층에 주로 사용하는 활성화함수

model.summary()
```

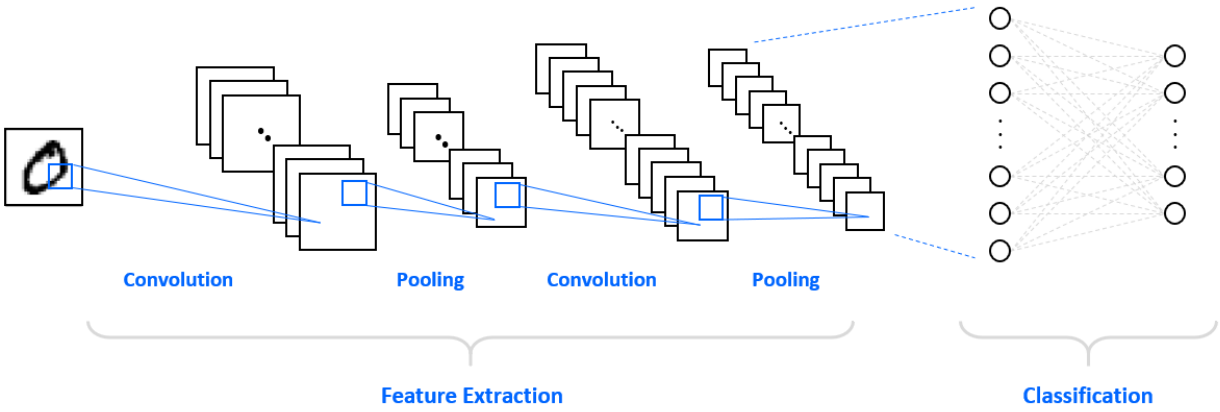
Flatten() : reshape(1,-1)과 같다. 다차원배열을 1차원으로 펼쳐준다. Dense층은 벡터(1D)를 입력으로 받지만, 현재 출력은 3D이기 때문에 사용한다.

Dense(Fully-connected layer) : 분류를 담당하는 층.

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv2D(32, (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.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(2, 2),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(10, activation='softmax')
])

model.summary()
```

우리가 공부한 코드는 이것과 같은데, 앞서 설명한 합성곱 신경망과 Dense층을 구성하는 또다른 방식이다.



그림으로 표현하면 위와 같다.

모델 컴파일

```
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

compile() 메서드를 이용하여 optimizer, 손실함수, 지표를 각각 설정해주었다.

optimizer : 훈련 과정을 설정하는 옵티마이저를 설정합니다. 'adam'이나 'sgd'와 같이 문자열로 지정할 수도 있다.
loss : 훈련 과정에서 사용할 손실 함수(loss function)를 설정한다.
metrics : 훈련을 모니터링하기 위한 지표를 선택한다.

훈련

```
model.fit(train_images, train_labels, epochs=5)
```

fit() 메서드를 이용하여 train_images, train_labels, epochs를 입력하고 훈련을 진행하였다.

모델 평가

```
loss, acc = model.evaluate(test_images, test_labels, verbose=2)
```

evaluate() 메서드에 test_images와 test_labels를 입력함으로써 모델을 평가할 수 있다.