7주차

▼ 8.1 합성곱 신경망의 구성 요소

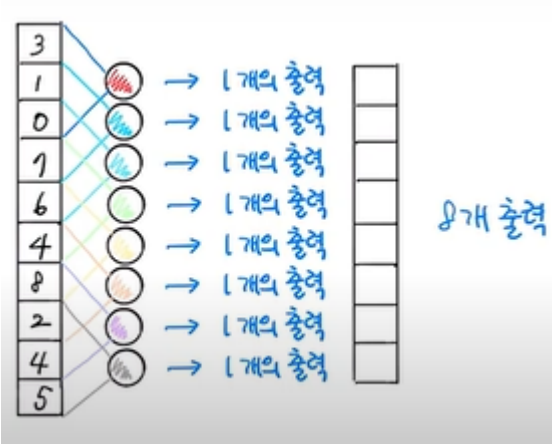
- 밀집층

  각 특성에 뉴런의 가중치(w) 곱하고 절편 더하기

  2차원 이미지를 1차원으로 펼치는 것이 비효율적

  → 처리 방식도 2차원으로 유지하는 것이 합성곱 신경망

- 합성곱



  입력 > 출력

  적은 개수의 가중치를 사용해 입력 위를 슬라이딩하며 연산 수행

- 커널=필터=가중치

- 2차원 합성곱

  합성곱을 이용해 이미지를 2차원으로 처리 가능

  합성곱 커널의 크기 < 입력

합성곱 연산 = 입력*합성곱 커널 가중치(w)+절편

- 특성 맵

  4X4 → 3X3 커널 → 2X2 출력 (특성맵, feature map)

  특성맵 = 활성화 출력 (relu 함수) = 합성곱 층 출력

- 여러 개의 필터

  필터마다 다른 가중치 가짐

  항상 절편이 붙음

  각 필터는 2X2 특성 맵을 만듦

  특성 맵을 쌓아 3차원 특성맵이 이루어짐

  필터 개수 = 차원 수 (배열)

- 케라스 합성곱 층

  keras.layers.ConvD(10, kernel_size=(3,3), activation='relu')

- 패딩의 목적

  패딩 - 주변에 하나의 픽셀을 덧붙임

  필터가 슬라이딩하는 영역을 늘림

  더 큰 특성맵이 만들어짐

  세임 패딩 : 입력 = 출력 패딩 방식 → 많이 사용

- 패딩의 목적

  연산 기여도를 높여 주변 정보를 잘 감지

- 케라스의 패딩 설정

  keras.layers.Conv2D(10, kernel_size=(3, 3), activation='relu', padding='same')

  valid 패딩 - 특성 맵 줄어듦

- 스트라이드

  이동 크기 (픽셀 단위)

  keras.layers.Conv2D(10, kernel_size=(3, 3), activation='relu', padding='same', stride=1)

- 풀링

  풀링층 : 동일한 크기의 특성 맵 크기를 절반으로 줄임

  4X4 → 2X2 → 3차원

  영역에 도장을 찍으며 슬라이딩

  - 평균 풀링 : 입력 특성 맵에 해당되는 영역에서 평균을 냄
  - 최대 풀링 : 그중 가장 큰 값을 사용

  풀링 층에는 가중치가 없음 → 합성곱과의 차이점

  채널 차원 - 변경 X

- 최대 풀링

  스트라이드 2칸씩 이동 (2X2)

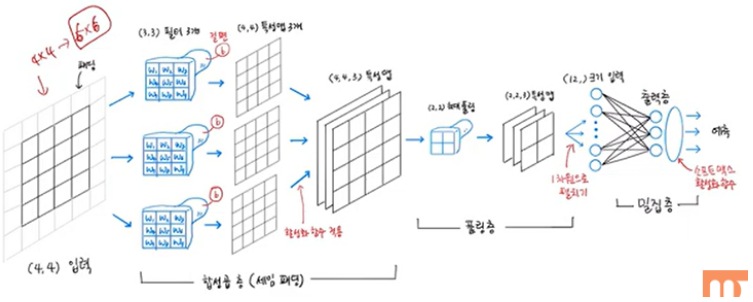
  풀링은 겹치지 않고 슬라이딩

- 케라스의 풀링 층

  keras.layers.MaxPooling2D(2)

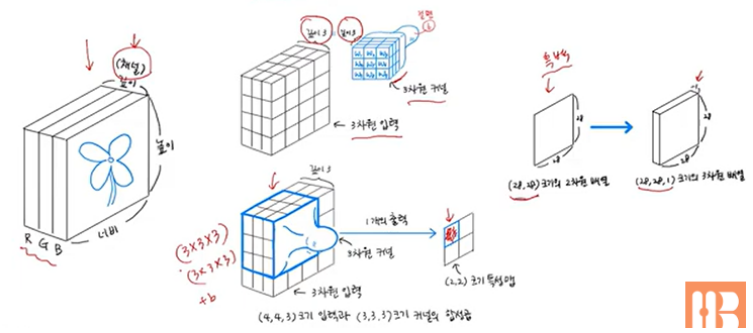
  keras.layers.MaxPooling2D(2, strides=2, padding='valid')

- 합성곱 신경망



CNN 구조

- 3차원 합성곱

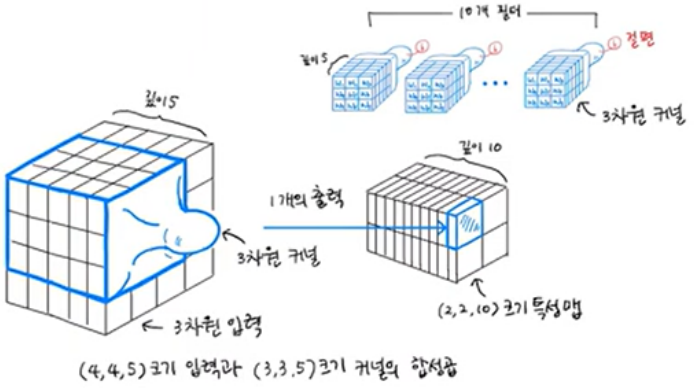


너비, 높이, 깊이

많은 가중치 필요, 계산량 증가

하나의 값으로 출력

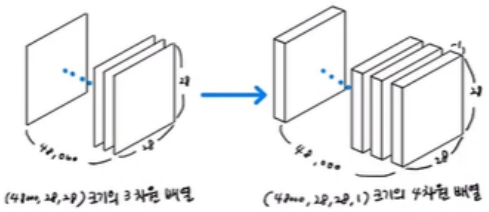
- 여러 개의 필터가 있는 3차원 합성곱



▼ 8.2 합성곱 신경망을 사용한 이미지 분류

- 패션 MNIST 데이터

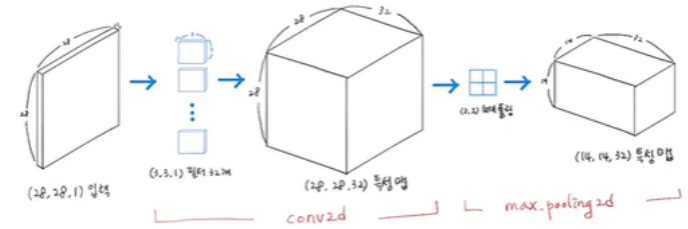
```
(train_input, train_target), (test_input, test_target) =
keras.datasets.fashion_mnist.load_data()

train_scaled = train_input.reshape(-1, 28, 28, 1) / 255.0

train_scaled, val_scaled, train_target, val_target = train_test_split(
    train_scaled, train_target, test_size=0.2, random_state=42)
```



입력 데이터를 3차원으로 변경

train_scaled, val_scaled, train_target, val_target으로 데이터 나눔

- 첫 번째 합성곱 층

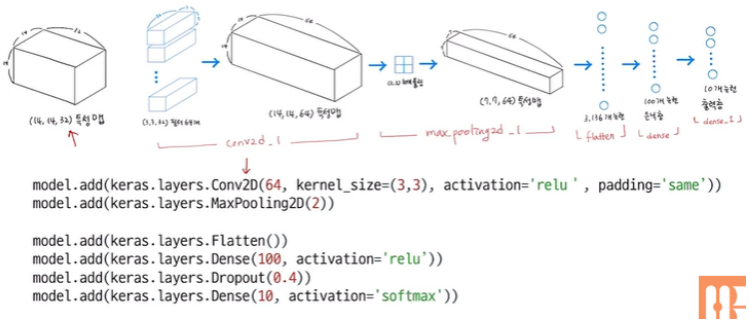



```
model = keras.Sequential()

model.add(keras.layers.Conv2D(32, kernel_size=3, activation='relu',
          padding='same', input_shape=(28,28,1)))

model.add(keras.layers.MaxPooling2D(2))
```

입력 데이터의 채널과 동일한 크기의 커널 차원이 만들어짐

풀링의 너비 크기 2, 깊이는 동일하게 유지

- 두 번째 합성곱 층 + 완전 연결 층



```
model.add(keras.layers.Conv2D(64, kernel_size=(3,3), activation='relu ', padding='same'))
model.add(keras.layers.MaxPooling2D(2))

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(100, activation='relu'))
model.add(keras.layers.Dropout(0.4))
model.add(keras.layers.Dense(10, activation='softmax'))
```

커널 사이즈의 높이와 너비는 같게 지정

MaxPooling2D : 절반으로

relu, softmax dense 층 사이 과대적합을 막기 위해dropout 층 추가

- 모델 요약

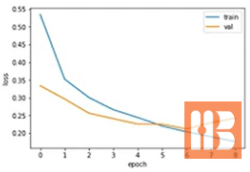
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
dense (Dense)                (None, 100)               313700
_________________________________________________________________
dropout (Dropout)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010
=================================================================
Total params: 333,526
Trainable params: 333,526
Non-trainable params: 0
_________________________________________________________________
```

- plot_model()

  keras.util.plot_model(model, show_shape=True)

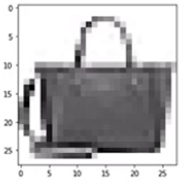
- 컴파일과 훈련

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

checkpoint_cb = keras.callbacks.ModelCheckpoint('best-cnn-model.h5')

early_stopping_cb = keras.callbacks.EarlyStopping(patience=2,
                                                  restore_best_weights=True)

history = model.fit(train_scaled, train_target, epochs=20,
                    validation_data=(val_scaled, val_target),
                    callbacks=[checkpoint_cb, early_stopping_cb])
```



- 평가와 예측

```
model.evaluate(val_scaled, val_target)
375/375 [==============================] - 1s 2ms/step - loss: 0.2118 - accuracy: 0.9212

[0.2118101418018341, 0.9212499856948853]

plt.imshow(val_scaled[0].reshape(28, 28), cmap='gray_r')
plt.show()

preds = model.predict(val_scaled[0:1])
print(preds)

[[1.4339859e-16 8.4426813e-24 2.2553201e-18 4.8564068e-16 3.6903781e-15
  6.4212738e-20 3.1580050e-15 9.1121079e-19 1.0000000e+00 1.0684053e-20]]
```



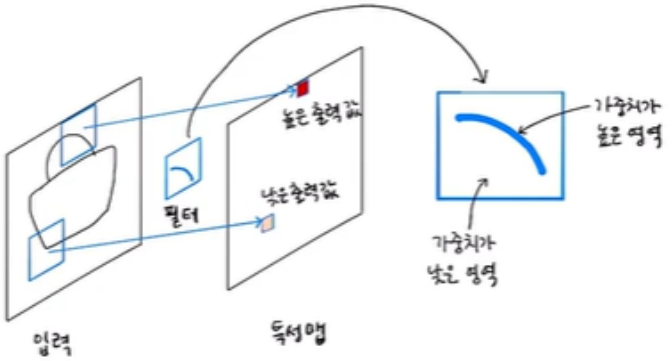
10개의 확률 출력

- 테스트 세트 점수

  test set에서 확인 - 검증 성능보다 낮음

▼ 8.3 합성곱 신경망의 시각화

- 가중치 시각화

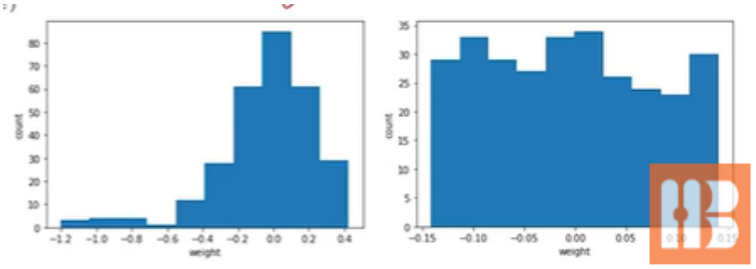


필터 학습 → 가중치가 높은 영역 : 큰 활성화 출력

- 층의 가중치 분포

  파이썬 객체에 합성곱 필터, 가중치, 절편 포함

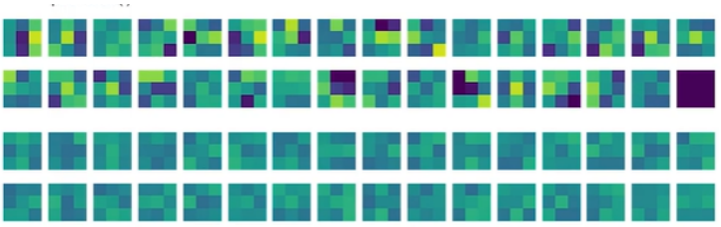
  가중치 시각화



  균등분포를 이용해 가중치 패턴 감지

- 층의 가중치 시각화

  imshow : 배열 값을 받아 이미지 출력

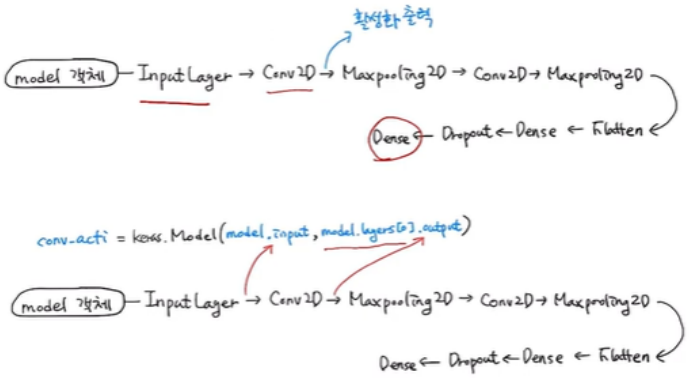


- 함수형 API

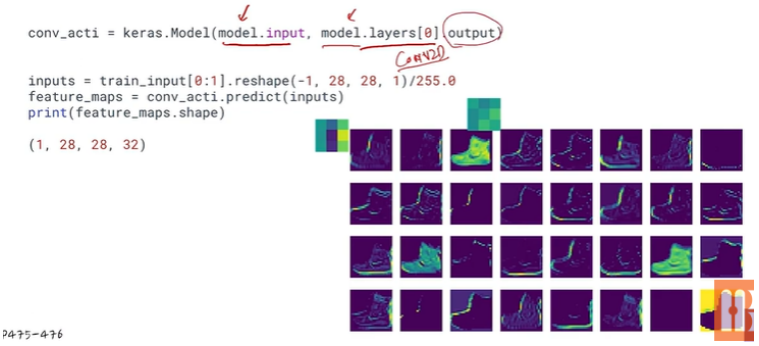
  dense 객체를 함수처럼 호출 가능

  호출되었을 때 특정 계산 수행

- 모델 객체의 층



- 첫 번째 특성 맵 시각화



신발 부분이 밝게 활성화되어 출력

배경 부분을 감지하는 가중치 - 학습을 추측

- 두 번째 특성 맵 시각화

  - 저수준 학습
  - 고수준 학습 → 추상적 개념 학습