

Convolutional Neural Network

구름

도시공학과 일반대학원

한양대학교

1. 1차원 합성곱 신경망

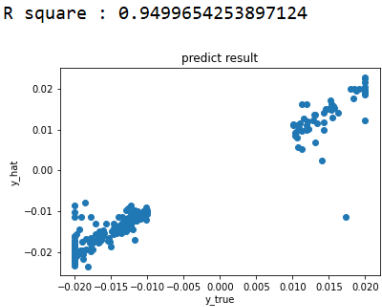
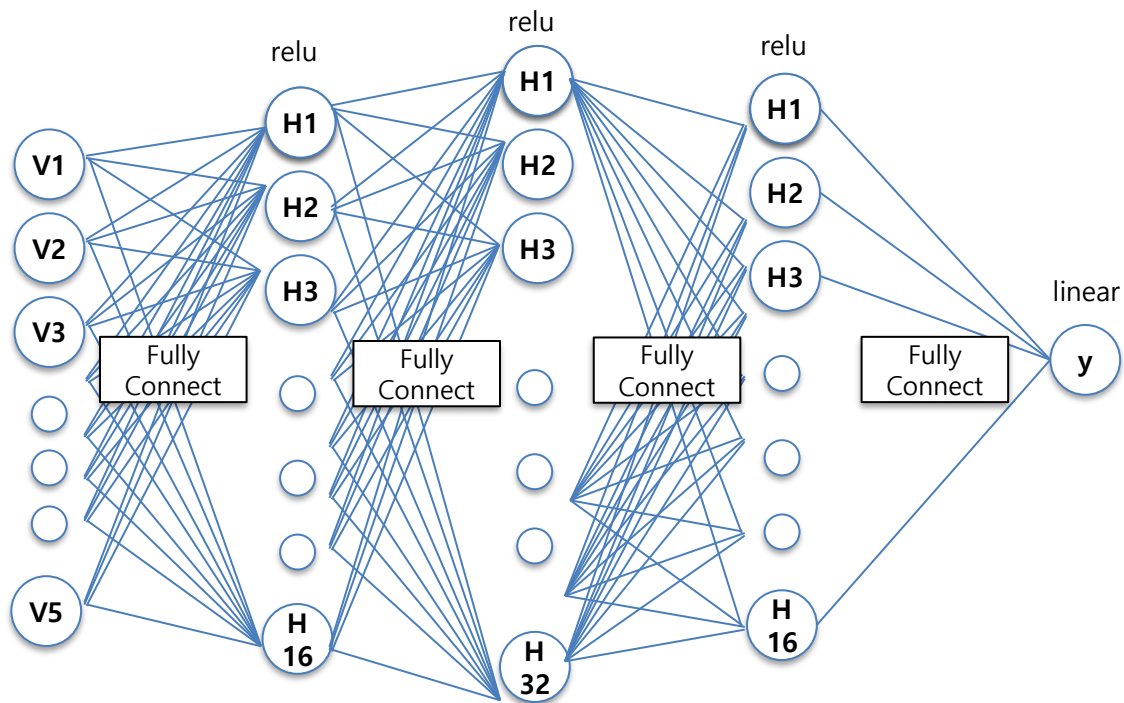
2. 이미지 데이터

3. 2차원 합성곱 신경망

4. 이미지 인식

Dense Layer를 이용한 생활 인구 변화 분석

이전 강의에서 Dense Layer를 통해 Fully Connected된 NN의 학습 과정을 확인



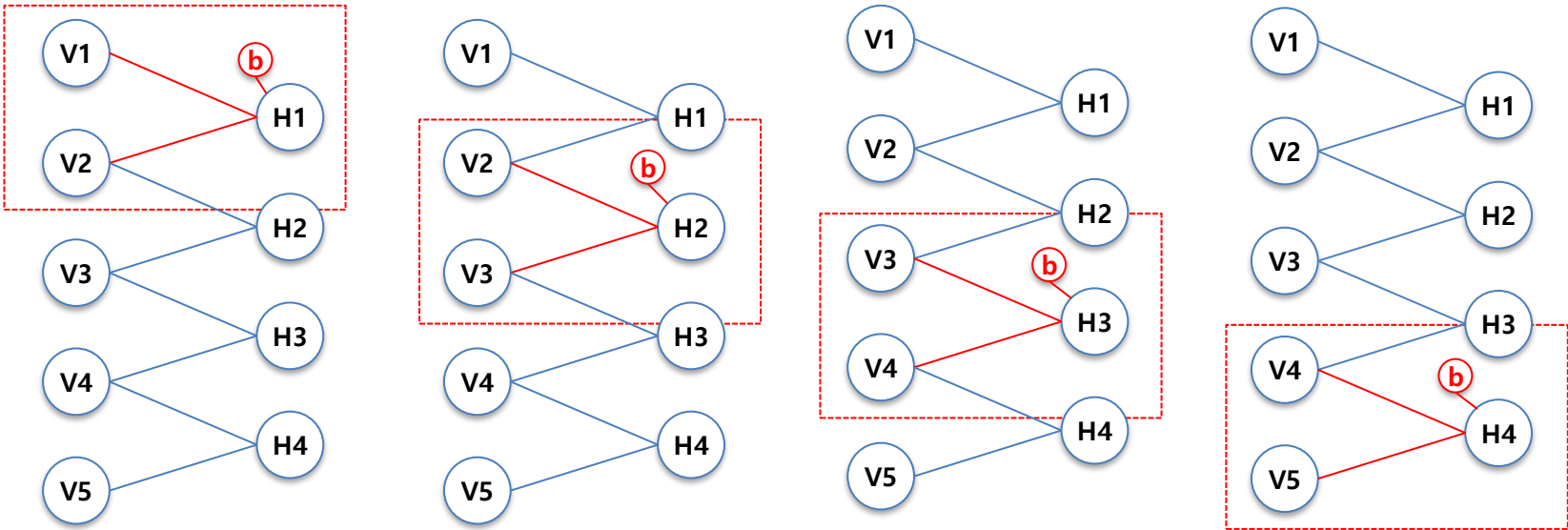
```
In [32]: model.summary()
Model: "sequential_1"
```

| Layer (type) | Output Shape | Param # |
|-------------------------|--------------|---------|
| dense_4 (Dense) | (None, 16) | 96 |
| dense_5 (Dense) | (None, 32) | 544 |
| dense_6 (Dense) | (None, 16) | 528 |
| dense_7 (Dense) | (None, 1) | 17 |
| Total params: 1,185 | | |
| Trainable params: 1,185 | | |
| Non-trainable params: 0 | | |

합성곱 신경망 Conv 1D 구조

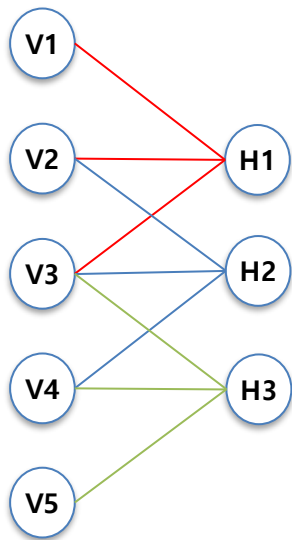
합성곱 신경망은 일정한 크기를 가진 필터를 일정한 간격으로 이동하면서 레이어간 노드를 연결하는 방식
인접 노드간의 특징을 추출하기에 적합

```
inputs = Input(shape=(5,1))
net = Conv1D(filters=1, kernel_size=2, activation='relu')(inputs)
net = Flatten()(net)
net = Dense(units=1, activation='linear')(net)
```

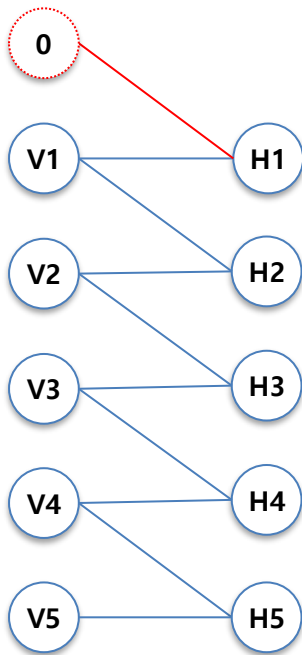


Conv1D 파라미터에 따른 네트워크 변화

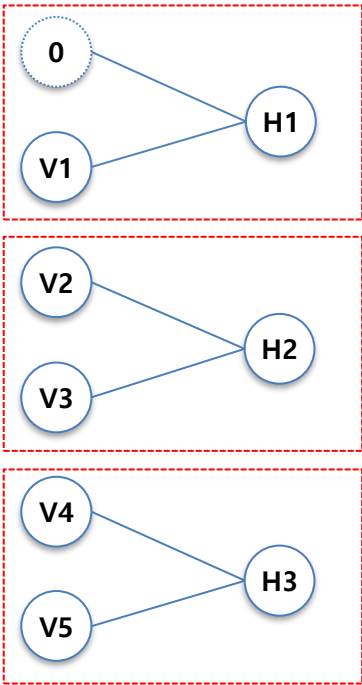
kernel_size=3



padding='same'

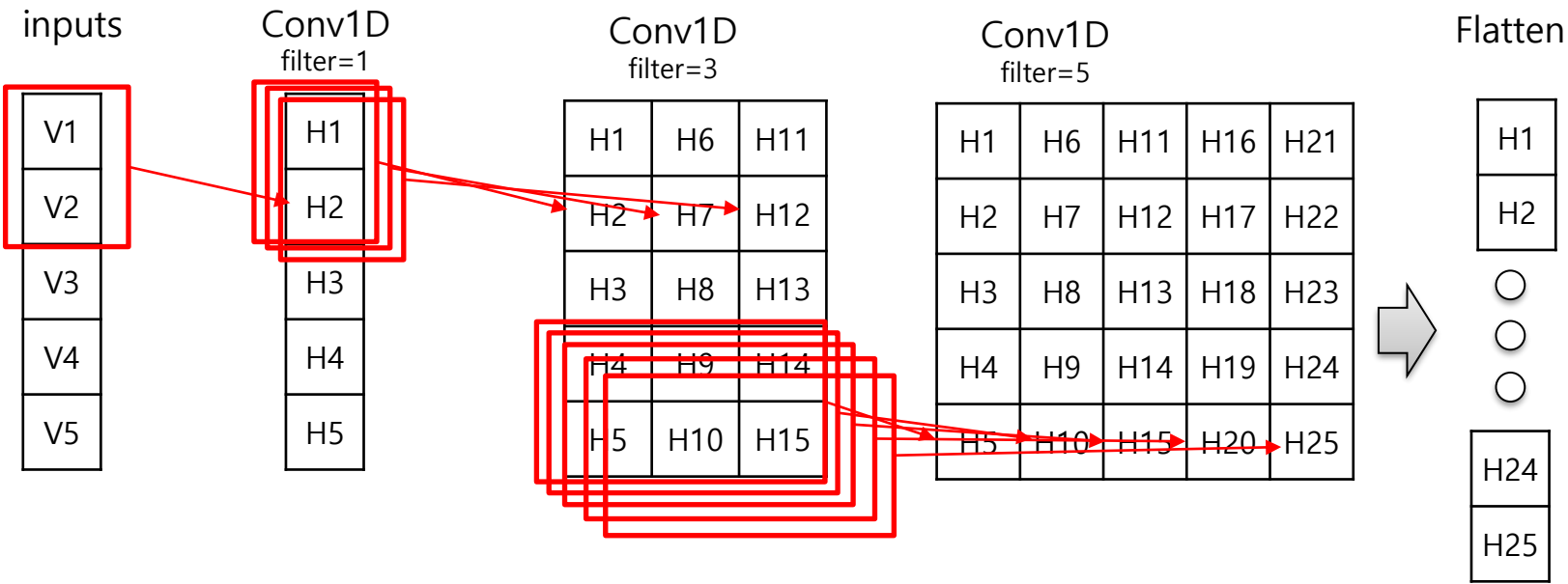


Strides=2



Filter 개수 만큼 가중치 생성

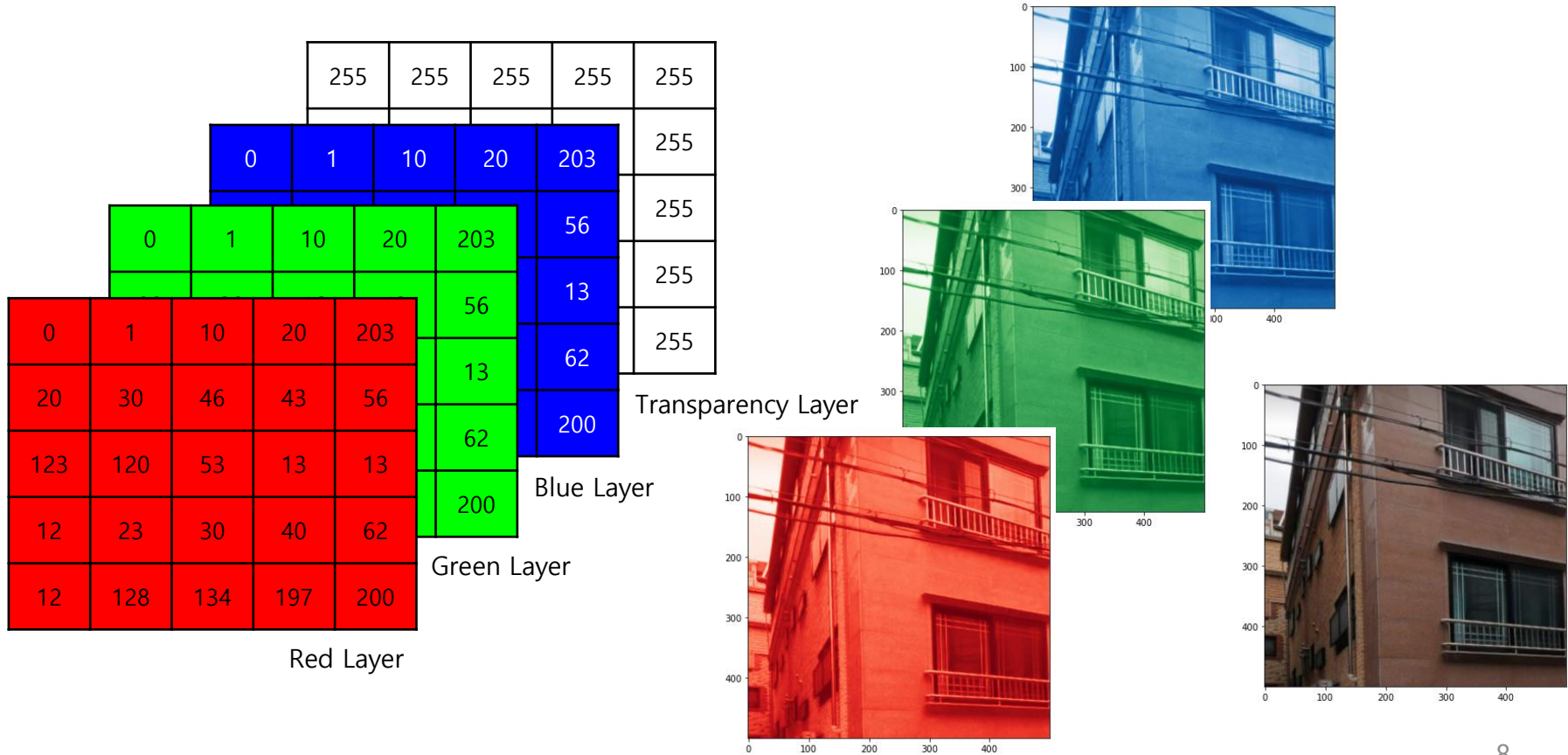
```
inputs = Input(shape=(5,1))
net = Conv1D(filters=1, kernel_size=2, padding='same', activation='relu')(inputs)
net = Conv1D(filters=3, kernel_size=2, padding='same', activation='relu')(net)
net = Conv1D(filters=5, kernel_size=2, padding='same', activation='relu')(net)
net = Flatten()(net)
net = Dense(units=1, activation='linear')(net)
```



1. 1차원 합성곱 신경망
- 2. 이미지 데이터**
3. 2차원 합성곱 신경망
4. 이미지 인식

이미지 파일 구조 이해

PIL(Python Imaging Library) 라이브러리를 이용하여 이미지를 불러낸 후 Numpy 배열로 변경
일반적인 이미지는 RGB 3개 층이 데이터를 보유
PNG의 경우에는 투명도를 포함하는 1개 레이어를 추가로 보유
배열 값을 변경하여 이미지 변환 가능



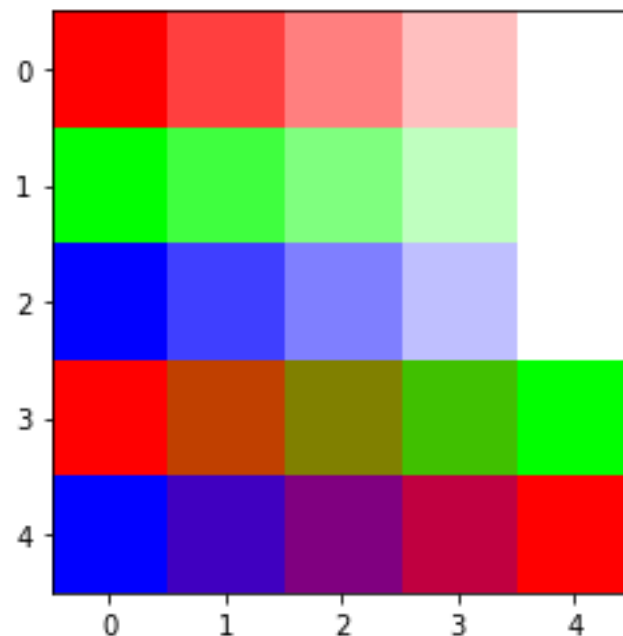
이미지 만들기

넘파일 array를 shape[5,5,3] 형태로 생성하여 직접 숫자를 입력하면 이미지 객체를 만들 수 있다.

```
r = np.array([[255, 255, 255, 255, 255],
              [0, 0, 0, 0, 0],
              [0, 0, 0, 0, 0],
              [255, 192, 128, 64, 0],
              [0, 64, 128, 192, 255]])
g = np.array([[0, 0, 0, 0, 0],
              [255, 255, 255, 255, 255],
              [0, 0, 0, 0, 0],
              [0, 64, 128, 192, 255],
              [0, 0, 0, 0, 0]])
b = np.array([[0, 0, 0, 0, 0],
              [0, 0, 0, 0, 0],
              [255, 255, 255, 255, 255],
              [0, 0, 0, 0, 0],
              [255, 192, 128, 64, 0]])
t = np.array([[255, 192, 128, 64, 0],
              [255, 192, 128, 64, 0],
              [255, 192, 128, 64, 0],
              [255, 255, 255, 255, 255],
              [255, 255, 255, 255, 255]])
```

```
img = np.zeros([5,5,4]).astype('int')
img[:,0] = r
img[:,1] = g
img[:,2] = b
img[:,3] = t
```

```
plt.imshow(img)
```



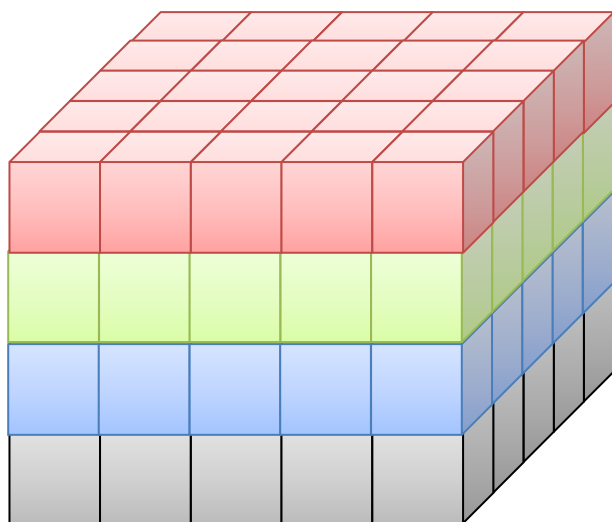
1. 1차원 합성곱 신경망
2. 이미지 데이터
- 3. 2차원 합성곱 신경망**
4. 이미지 인식

Conv2D Layer 이해하기

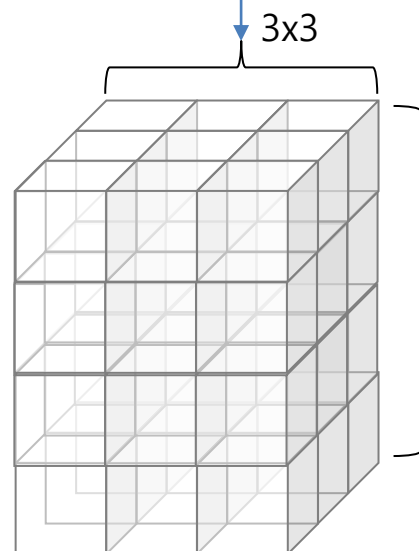
생성된 이미지를 입력으로 받아서 Conv2D 네트워크를 생성

```
net = layers.Conv2D( 5, kernel_size=3)(inputs)
```

아래
커널을
5개 만듦



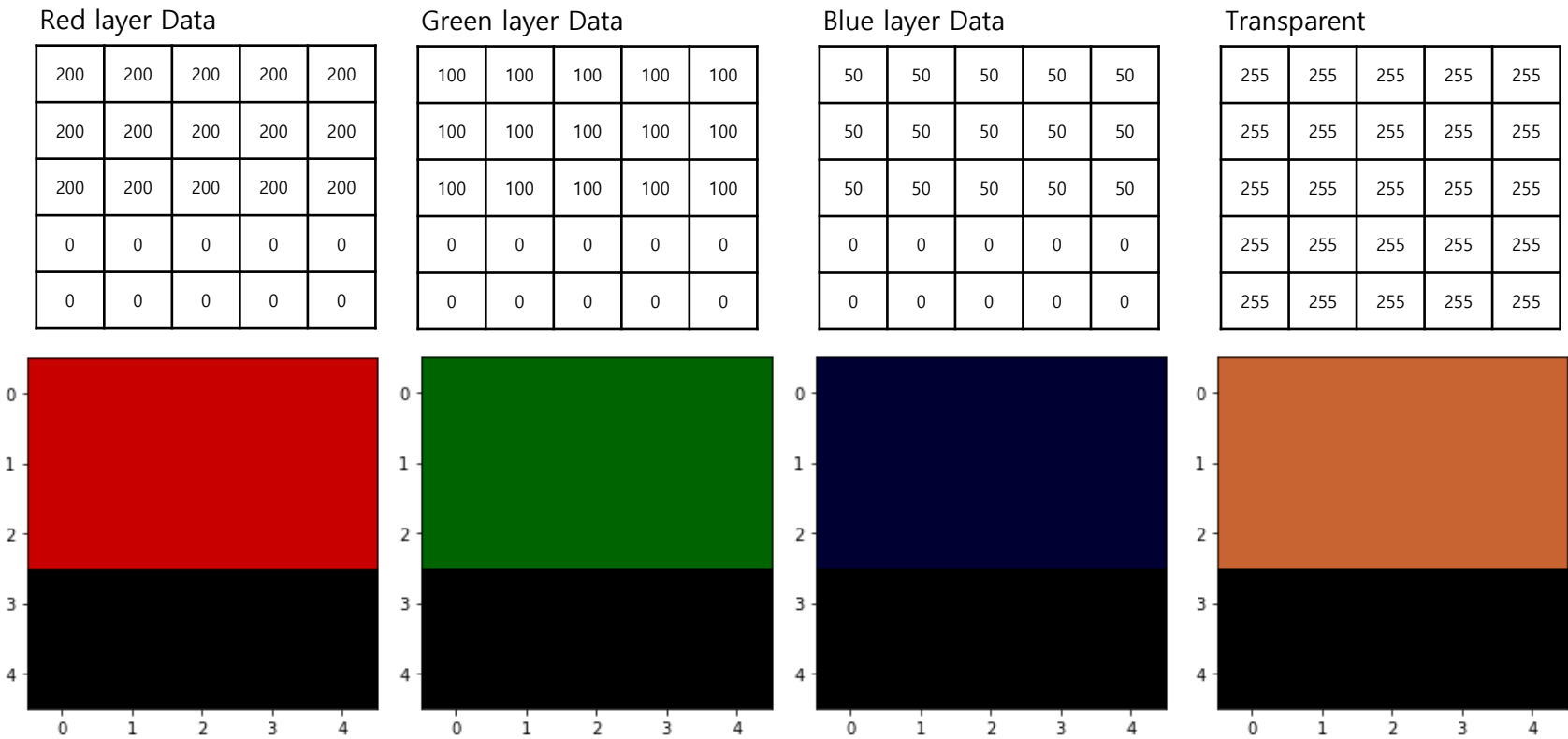
입력 데이터



이 크기는
입력 레이어에서
이미 결정

임의의 이미지 데이터셋을 생성

Conv2D input으로 활용하기 위해 값을 지정



Conv2D Layer 이해하기

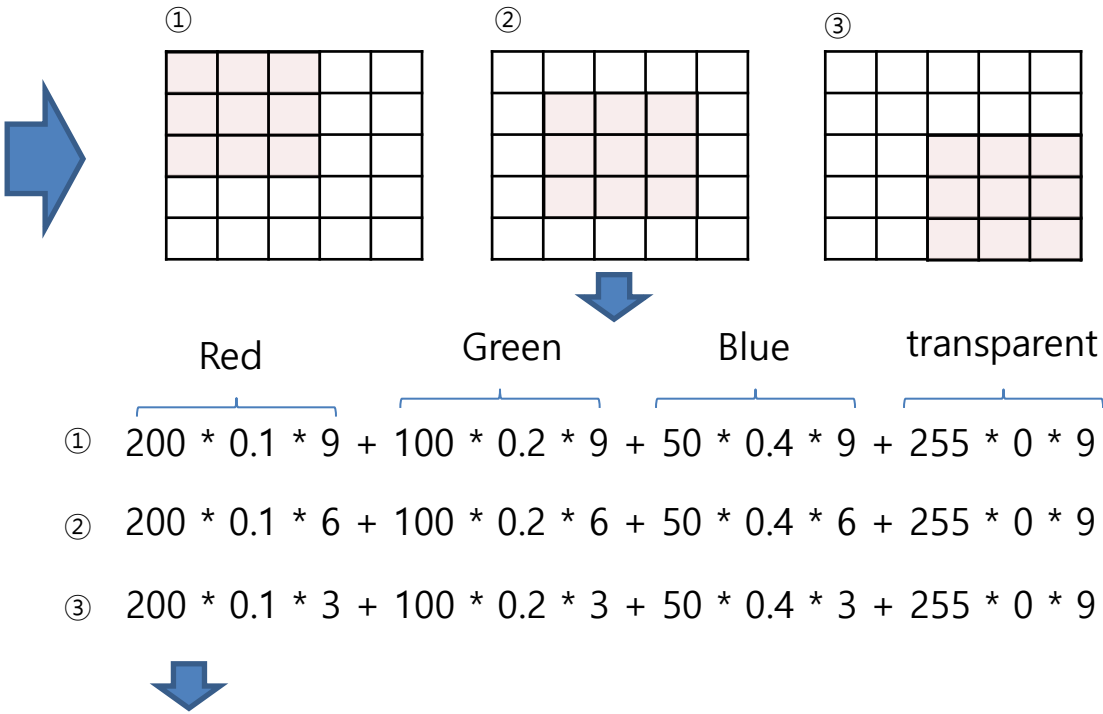
Conv2D는 Input X Weight 구조를 직접 확인

1번 Weights = filter R

| | | |
|-----|-----|-----|
| 0.1 | 0.1 | 0.1 |
| 0.1 | 0.1 | 0.1 |
| 0.1 | 0.1 | 0.1 |

Red layer Data

| | | | | |
|-----|-----|-----|-----|-----|
| 200 | 200 | 200 | 200 | 200 |
| 200 | 200 | 200 | 200 | 200 |
| 200 | 200 | 200 | 200 | 200 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |









Conv1 Output Data

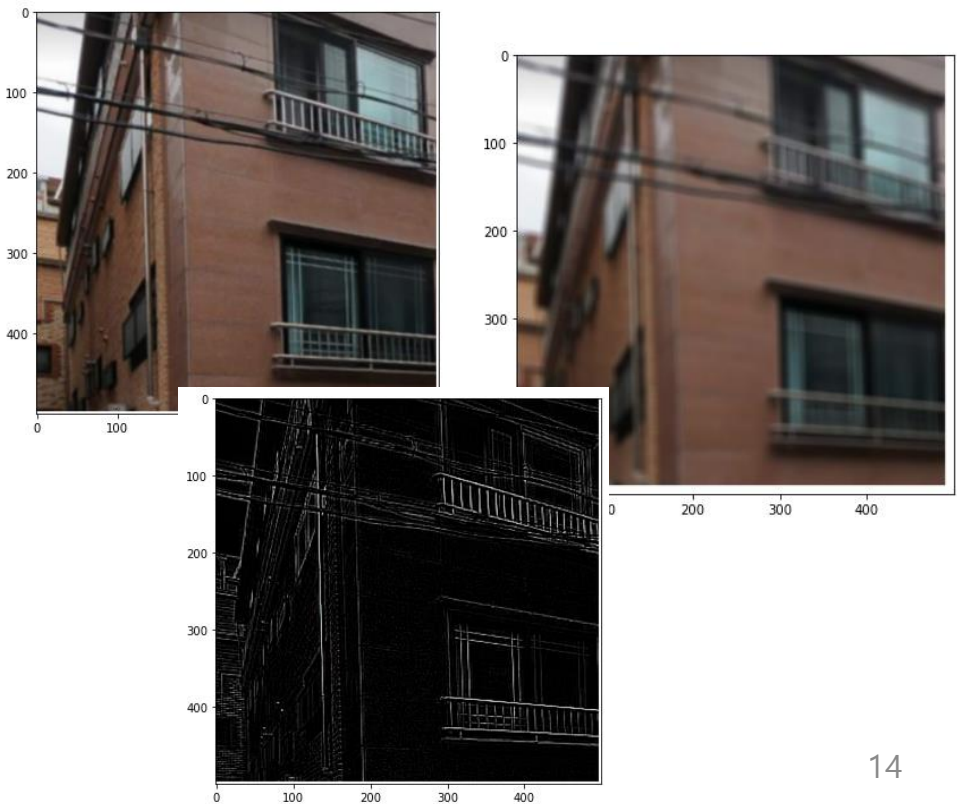
| | | | | | |
|---|---|---|-----|-----|-----|
| ① | | | 540 | 540 | 540 |
| | ② | | 360 | 360 | 360 |
| | | ③ | 180 | 180 | 180 |

이미지 Filter

이미지 필터링은 이미지 처리에서 많이 쓰이는 기법
필터의 계산 방법이 이미지와 합성곱으로 계산되는 방식

$$G_{ij} = (F * X)(i, j) = \sum_{m=0}^{F_H-1} \sum_{n=0}^{F_W-1} F_{m,n} X_{(i-m),(j-n)} \cdot$$

| Operation | Filter | Filtered image |
|--------------------------|---|---|
| Identity | $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ |  |
| Edge detection | $\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$ |  |
| | $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$ |  |
| | $\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$ |  |
| Sharpen | $\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$ |  |
| Box blur (normalized) | $\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$ |  |

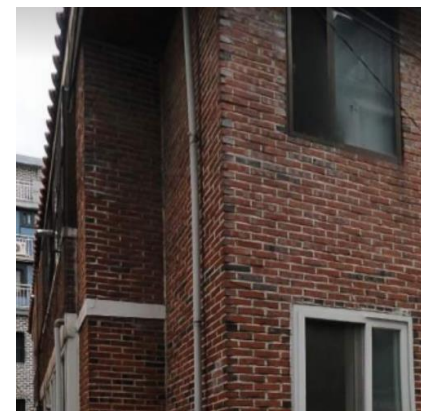
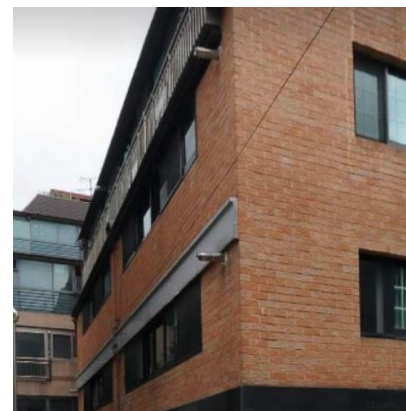
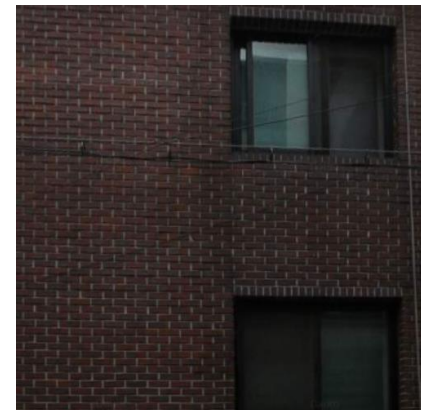
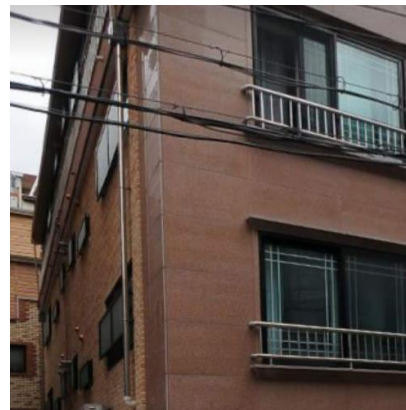


1. 1차원 합성곱 신경망
2. 이미지 데이터
3. 2차원 합성곱 신경망
- 4. 이미지 인식**

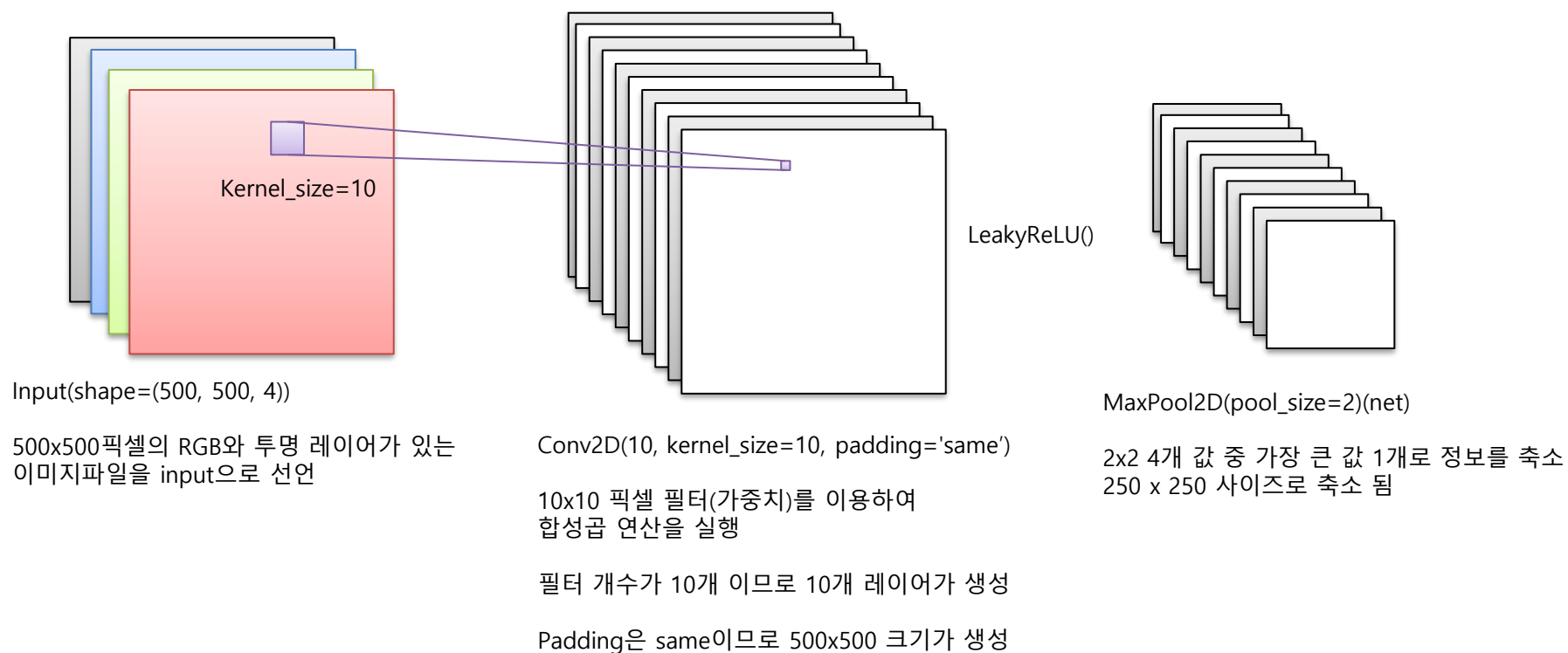
외벽 이미지를 벽돌/대리석 구분

이미지와 labeling 파일 이용하여 학습 데이터셋 만들기

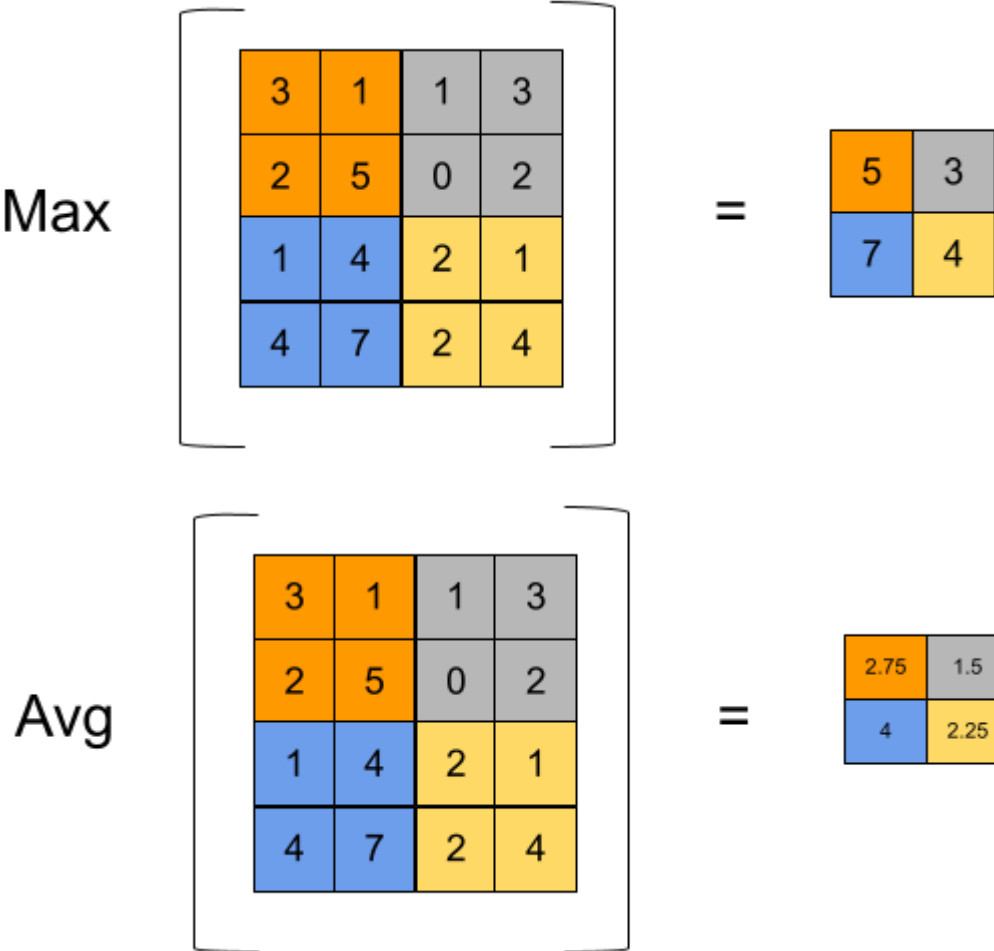
| Index | image | type | y |
|-------|---------------------------|------|---|
| 0 | BV1171010400100090001.png | 대리석 | 1 |
| 1 | BV1171010400100090008.png | 벽돌 | 0 |
| 2 | BV1171010400100090011.png | 벽돌 | 0 |
| 3 | BV1171010400100090016.png | 벽돌 | 0 |
| 4 | BV1171010400100100000.png | 벽돌 | 0 |
| 5 | BV1171010400100100001.png | 대리석 | 1 |
| 6 | BV1171010400100100002.png | 벽돌 | 0 |
| 7 | BV1171010400100100004.png | 벽돌 | 0 |
| 8 | BV1171010400100100005.png | 벽돌 | 0 |
| 9 | BV1171010400100100006.png | 대리석 | 1 |
| 10 | BV1171010400100100008.png | 대리석 | 1 |
| 11 | BV1171010400100100010.png | 벽돌 | 0 |
| 12 | BV1171010400100100011.png | 벽돌 | 0 |
| 13 | BV1171010400100100014.png | 벽돌 | 0 |
| 14 | BV1171010400100100015.png | 벽돌 | 0 |
| 15 | BV1171010400100100016.png | 벽돌 | 0 |
| 16 | BV1171010400100100018.png | 벽돌 | 0 |
| 17 | BV1171010400100110003.png | 벽돌 | 0 |
| 18 | BV1171010400100110004.png | 벽돌 | 0 |
| 19 | BV1171010400100110006.png | 벽돌 | 0 |



Conv2D Layer 이해하기

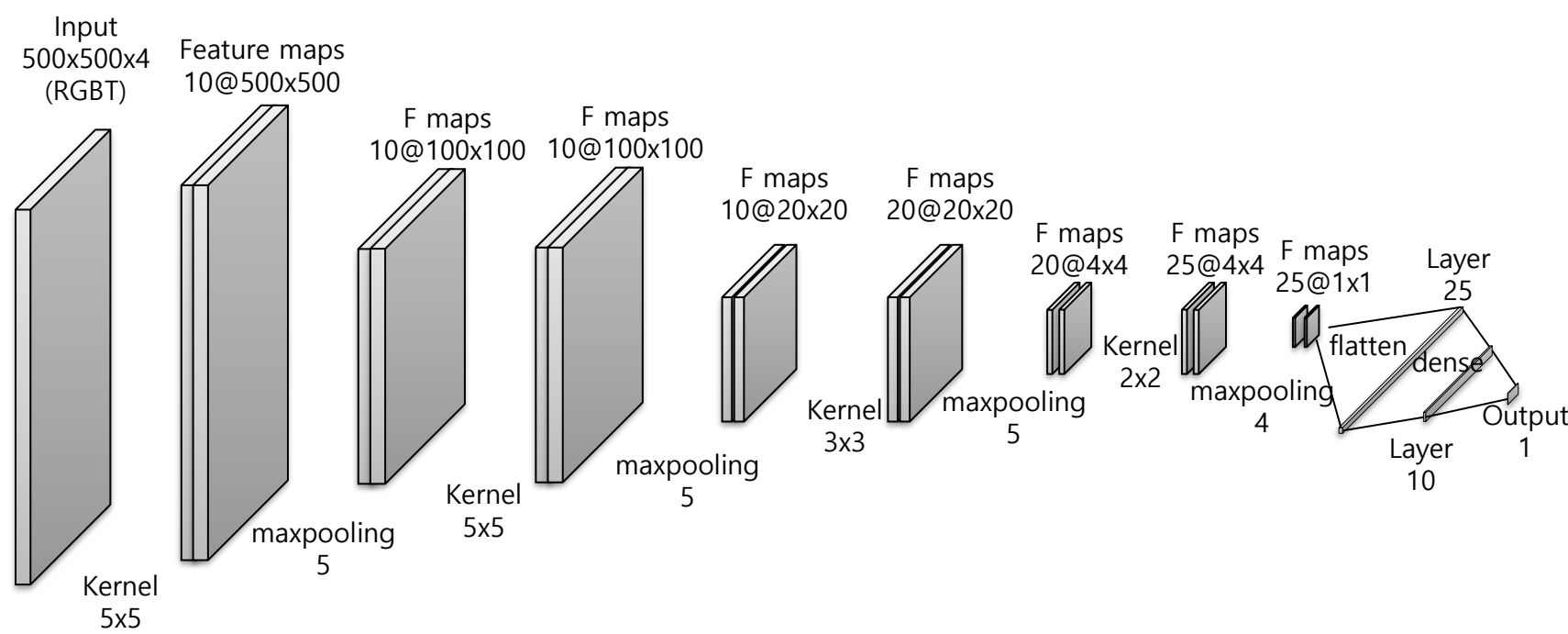


Pooling Layers

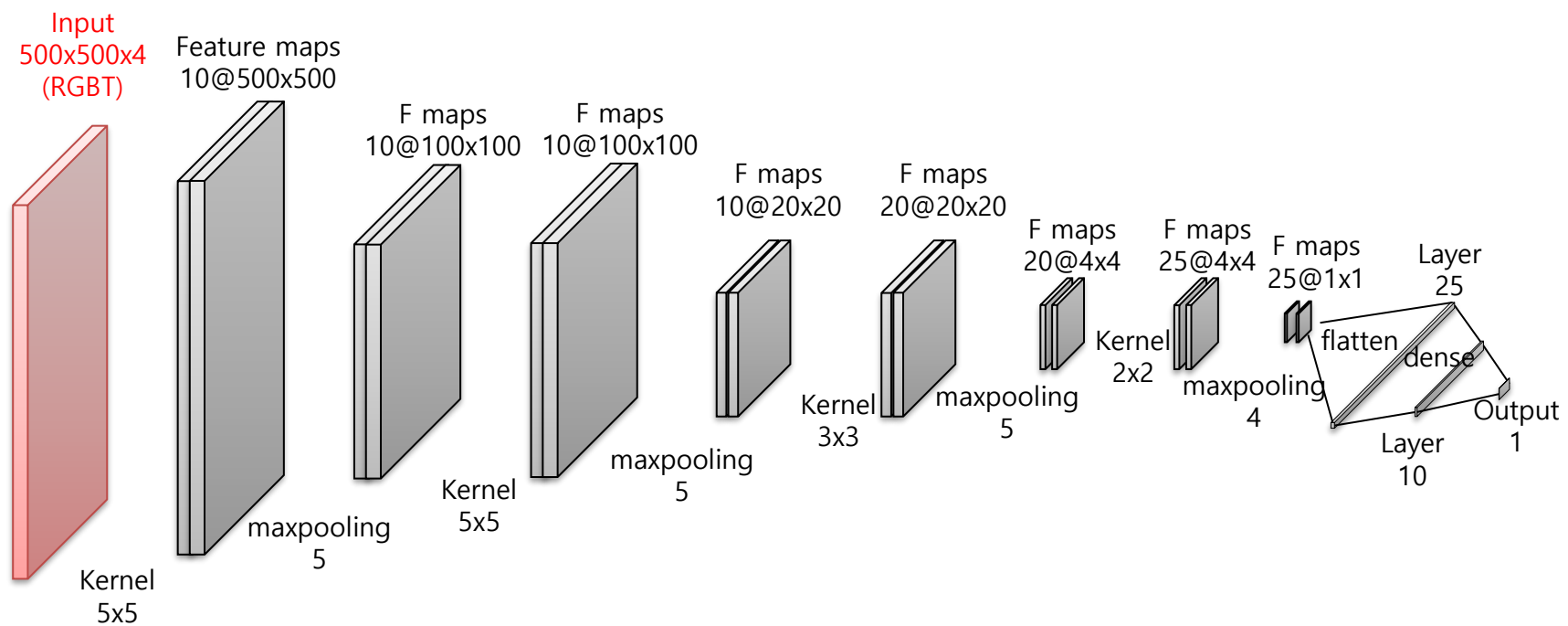


외벽 학습 CNN 모델

소스코드의 ConvNet 모델을 도식화 해서 표시하면 아래와 같이 표시 가능
Activation 함수를 추가로 표기할 수 도 있음

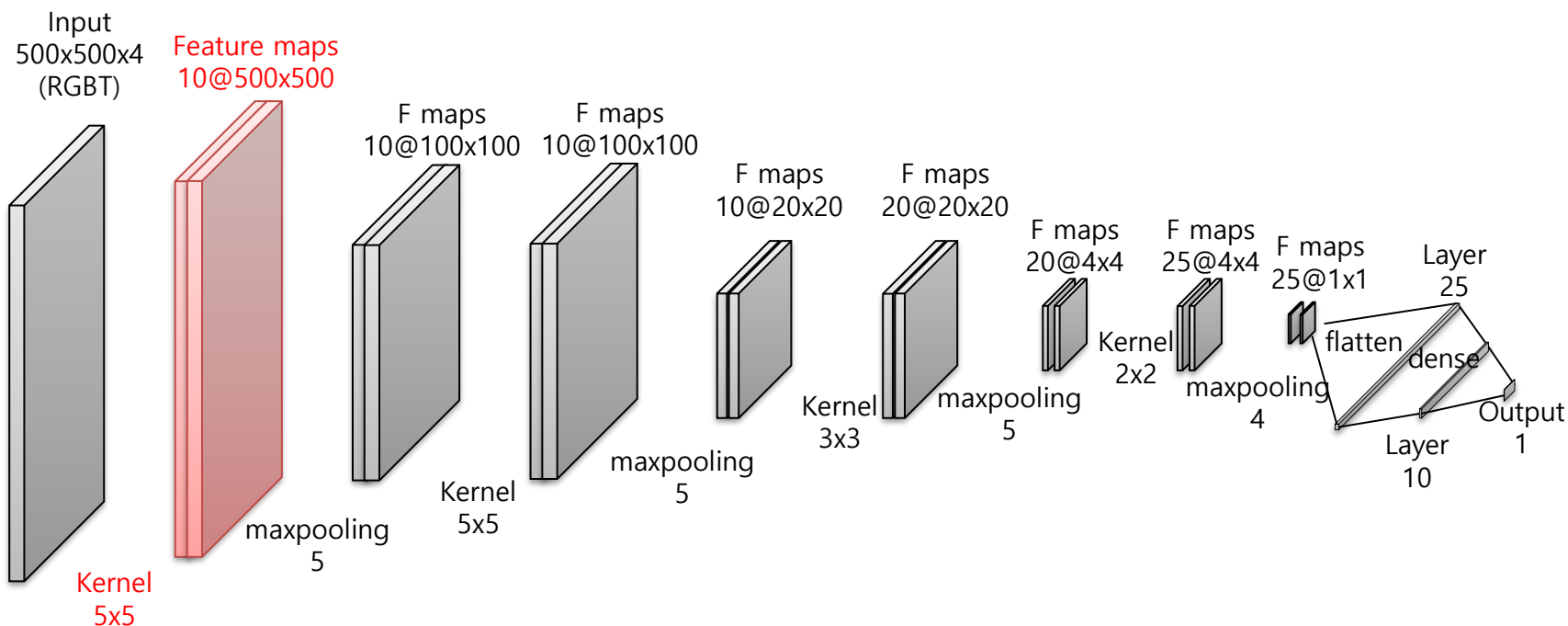


외벽 학습 CNN 모델 Source Code



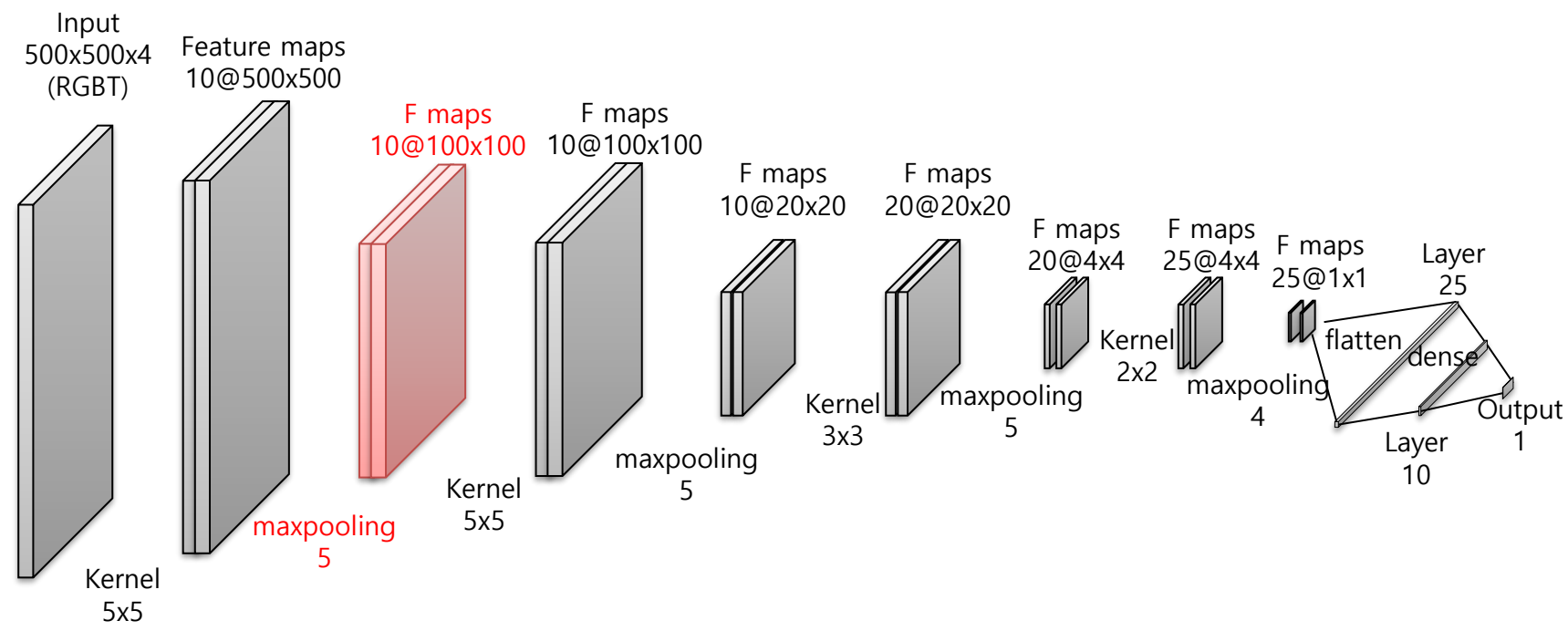
```
inputs = layers.Input(shape=(500, 500, 4))
```

외벽 학습 CNN 모델 Source Code



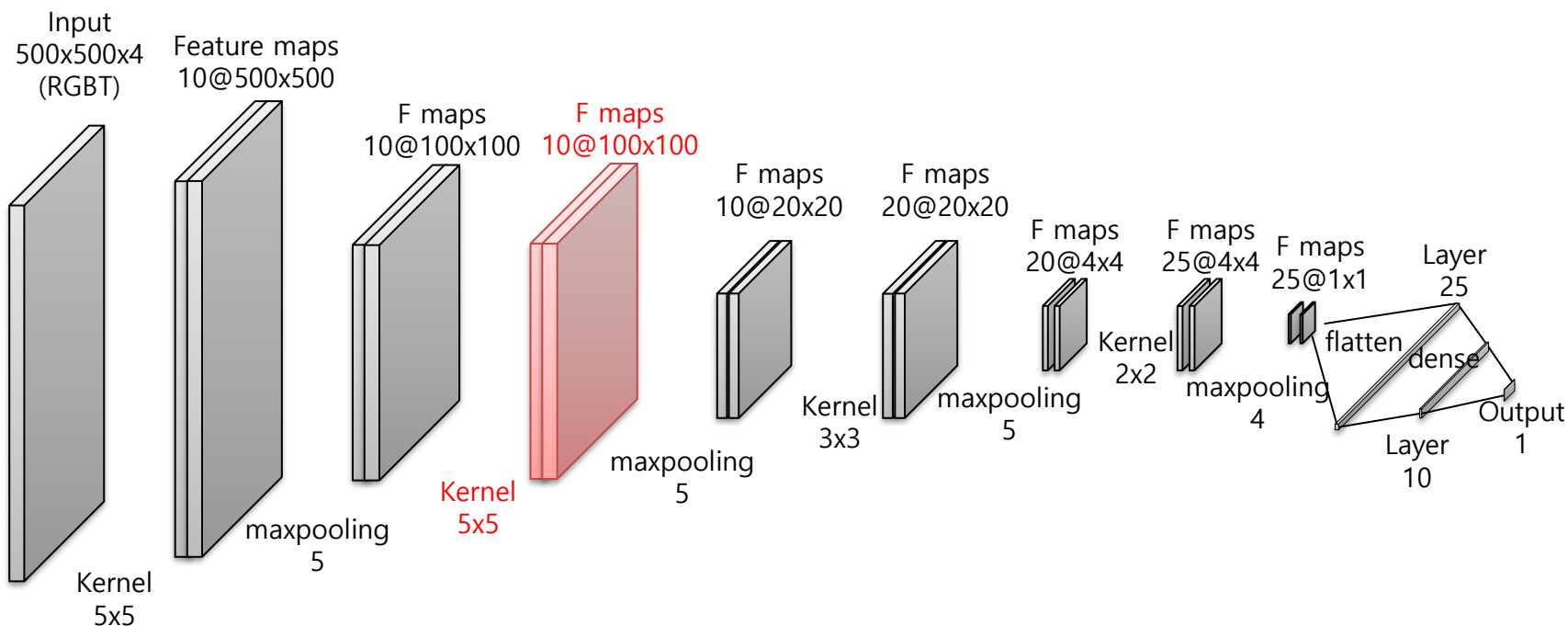
```
net = layers.Conv2D(10, kernel_size=5, padding='same')(inputs)
net = layers.LeakyReLU()(net)
```

외벽 학습 CNN 모델 Source Code



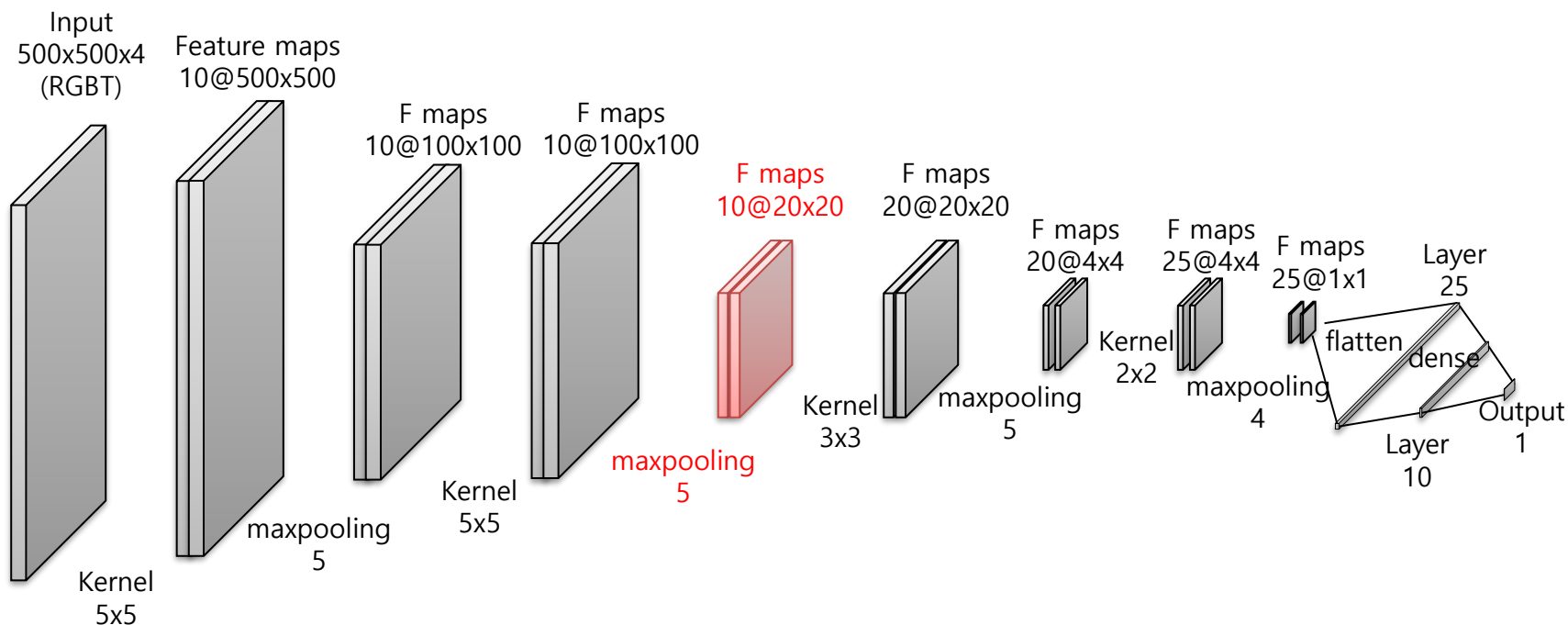
```
net = layers.MaxPool2D(pool_size=5)(net)
```

외벽 학습 CNN 모델 Source Code



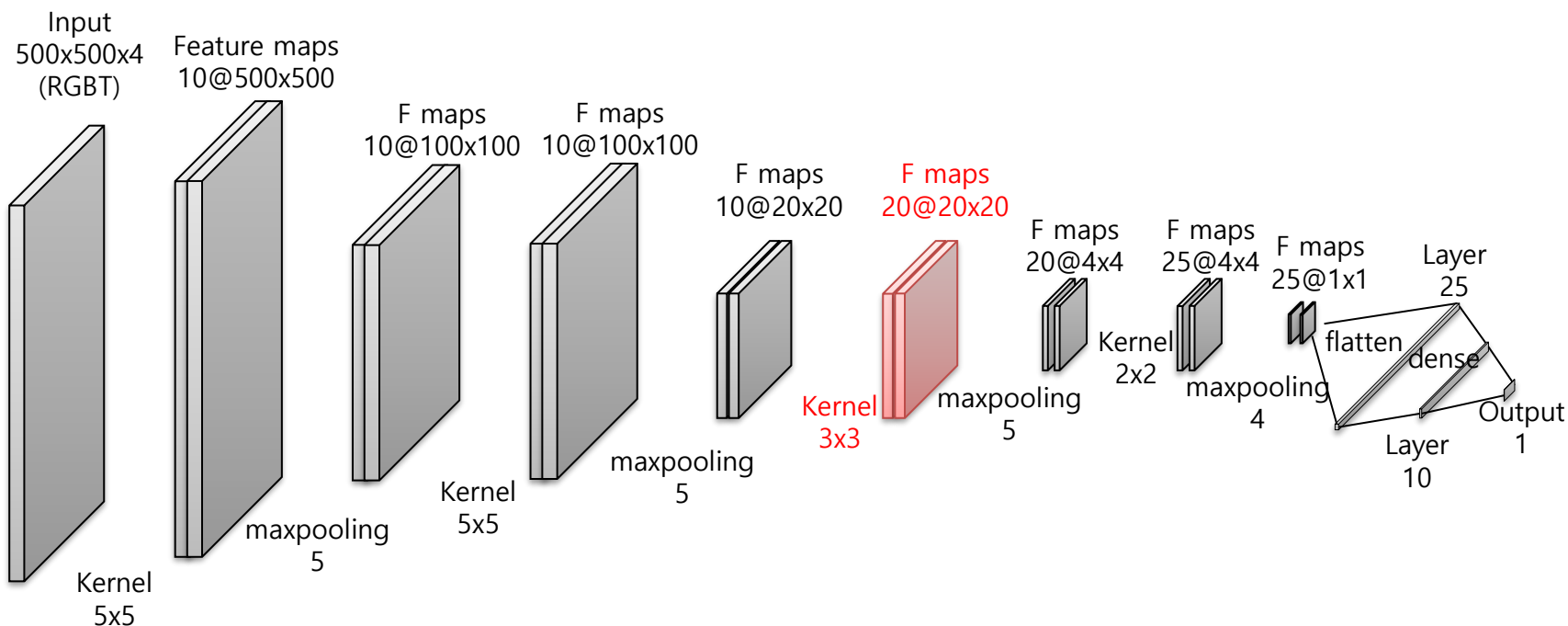
```
net = layers.Conv2D(10, kernel_size=5, padding='same')(net)
net = layers.LeakyReLU()(net)
```

외벽 학습 CNN 모델 Source Code



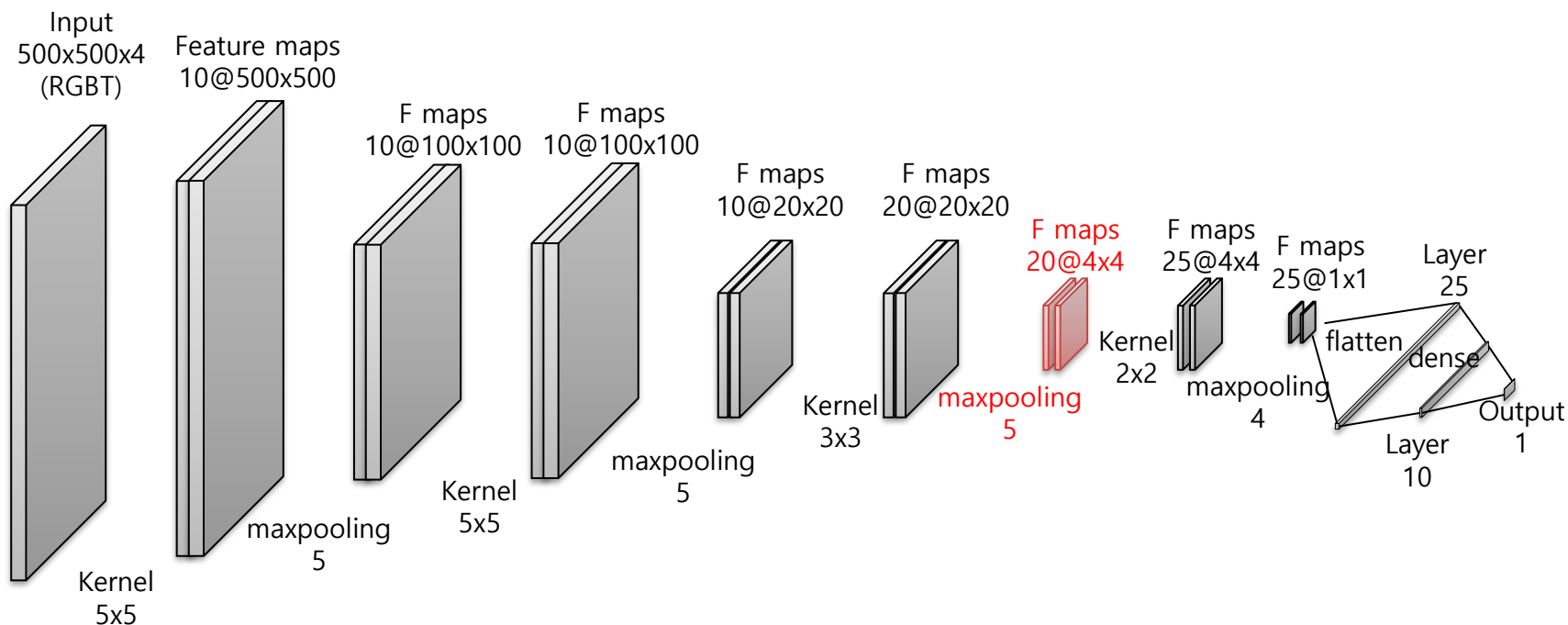
```
net = layers.MaxPool2D(pool_size=5)(net)
```


외벽 학습 CNN 모델 Source Code



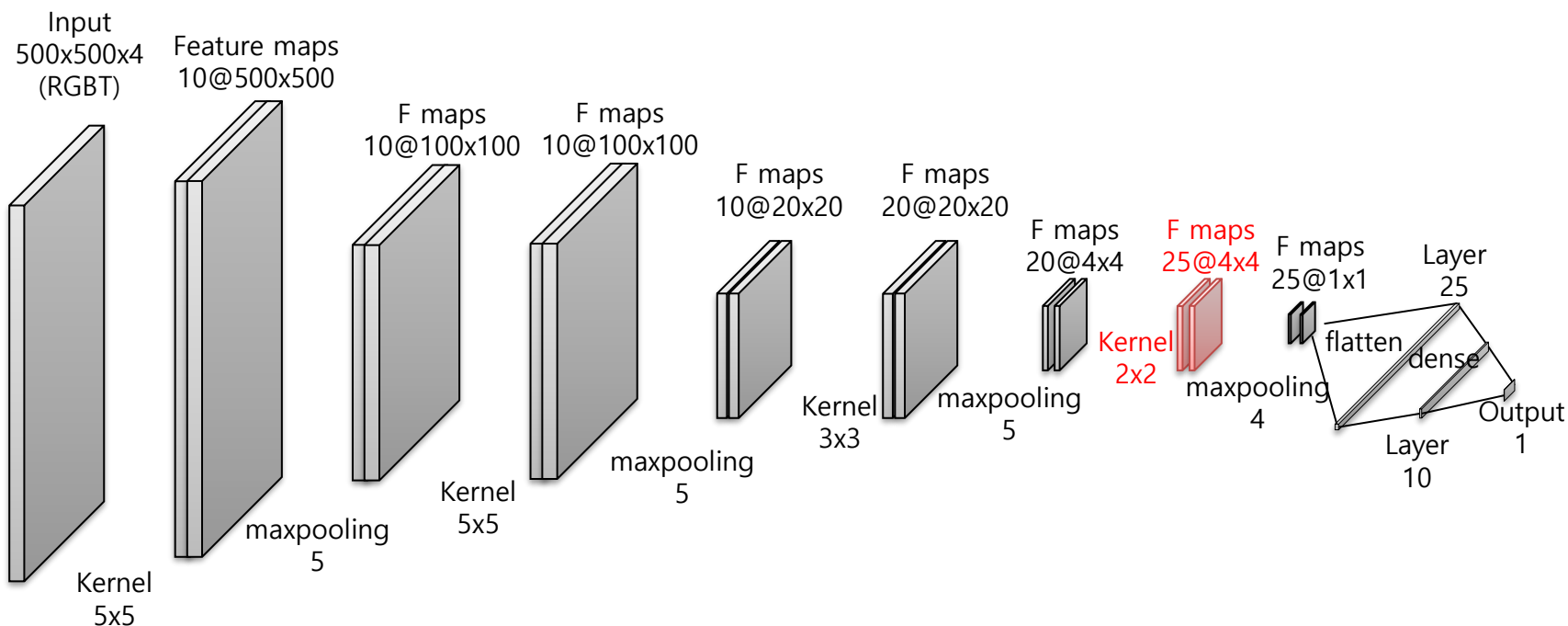
```
net = layers.Conv2D(20, kernel_size=3, padding='same')(net)
net = layers.LeakyReLU()(net)
```

외벽 학습 CNN 모델 Source Code



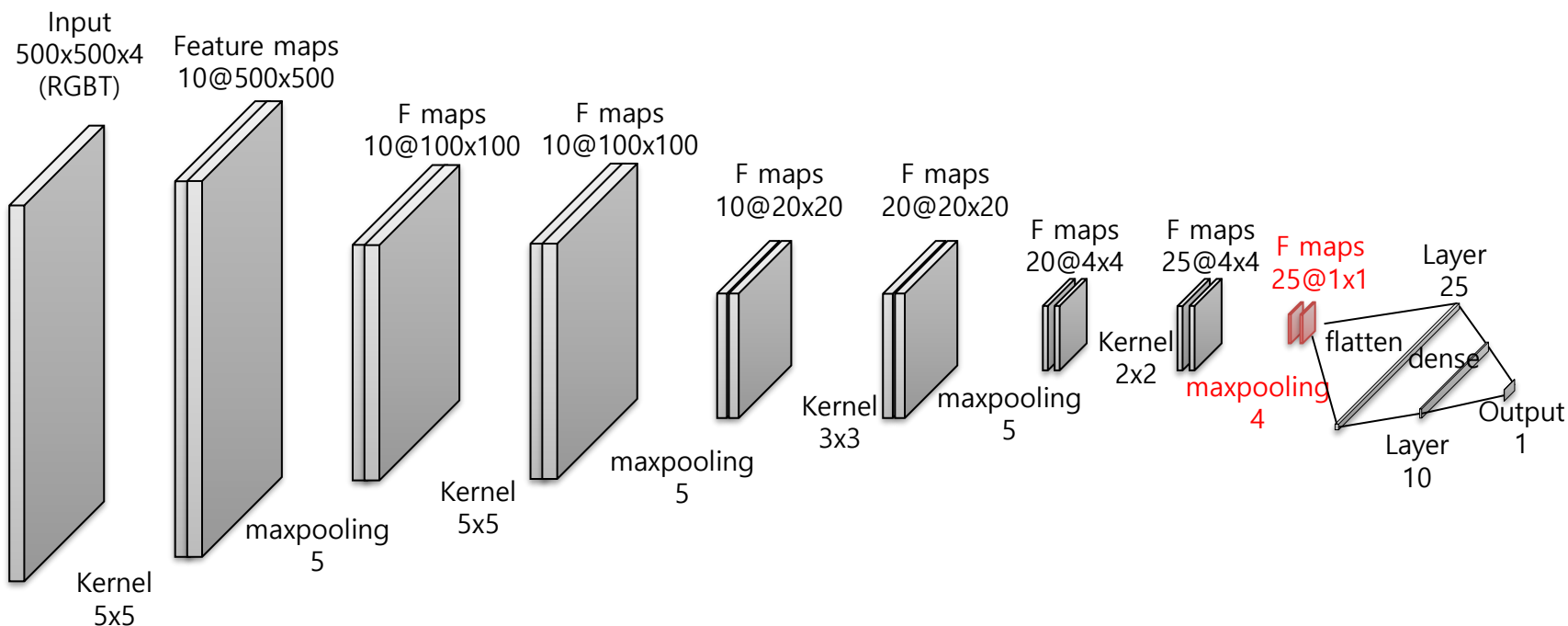
```
net = layers.MaxPool2D(pool_size=5)(net)
```

외벽 학습 CNN 모델 Source Code



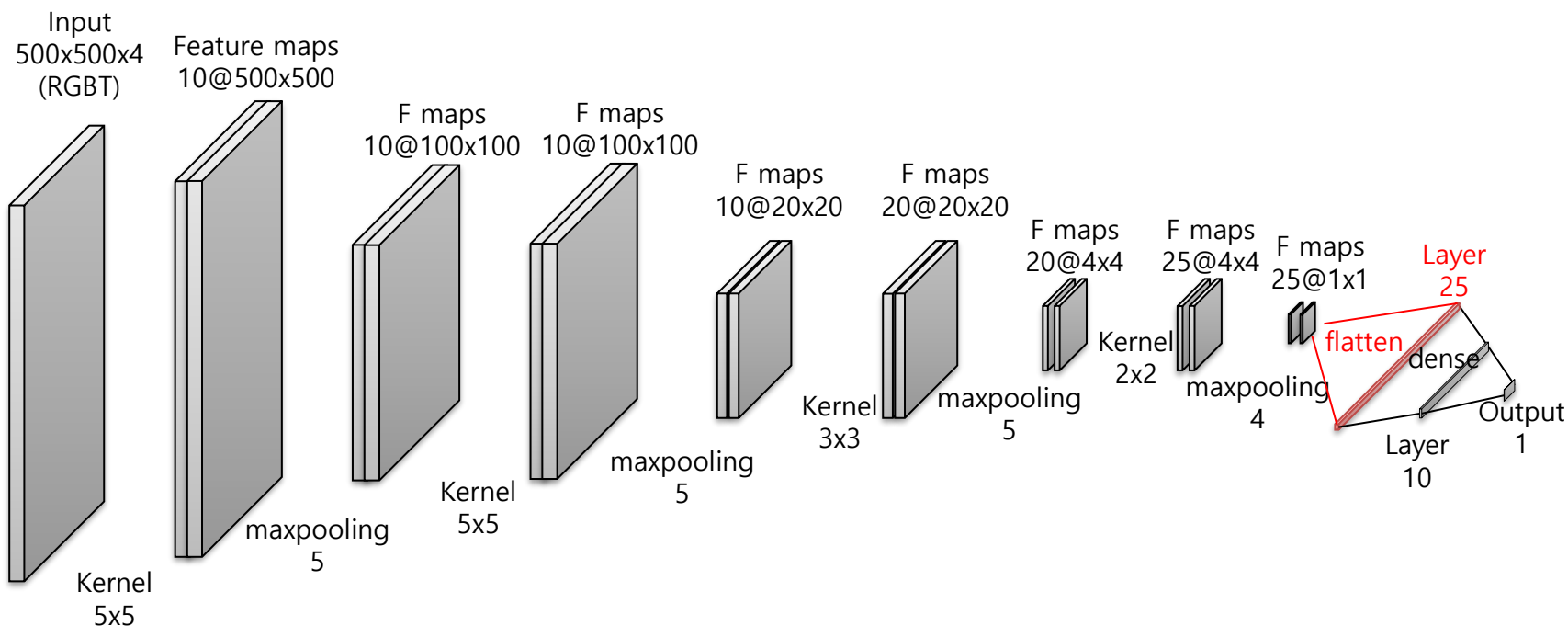
```
net = layers.Conv2D(25, kernel_size=2, padding='same')(net)
net = layers.LeakyReLU()(net)
```

외벽 학습 CNN 모델 Source Code



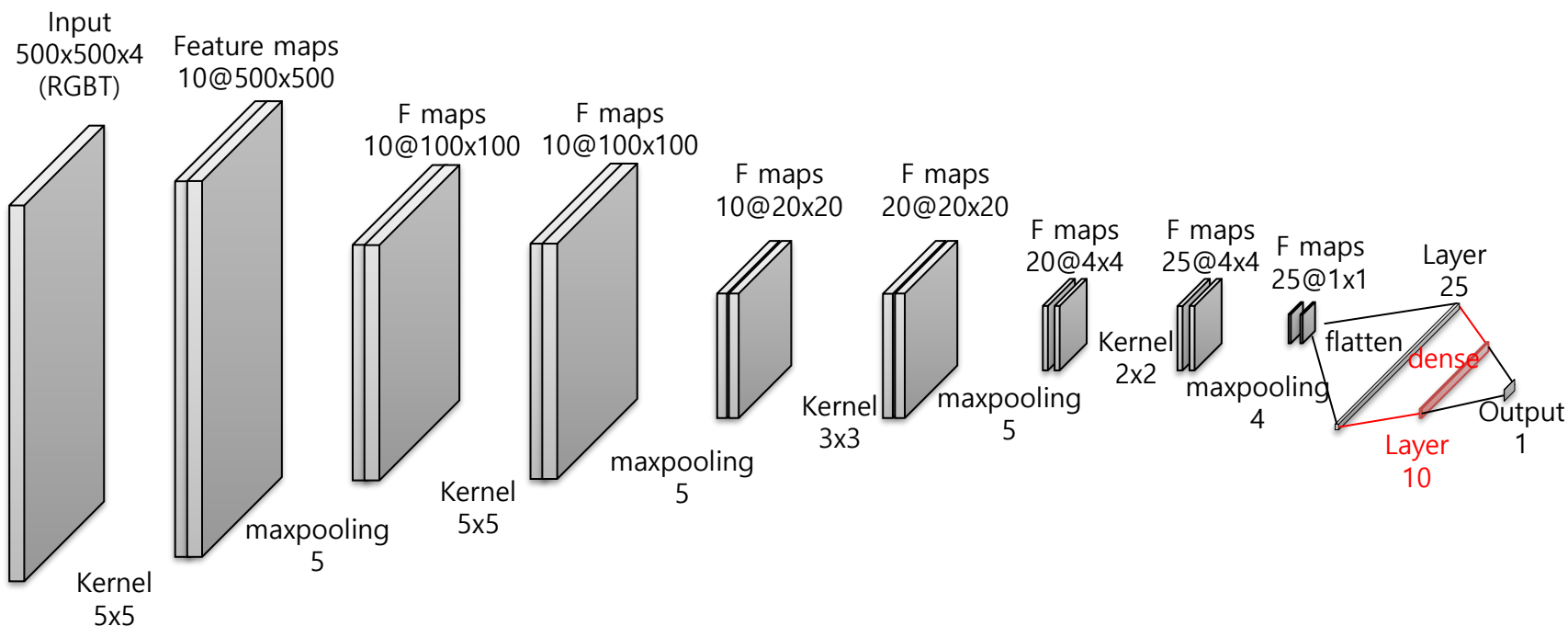
```
net = layers.MaxPool2D(pool_size=4)(net)
```

외벽 학습 CNN 모델 Source Code



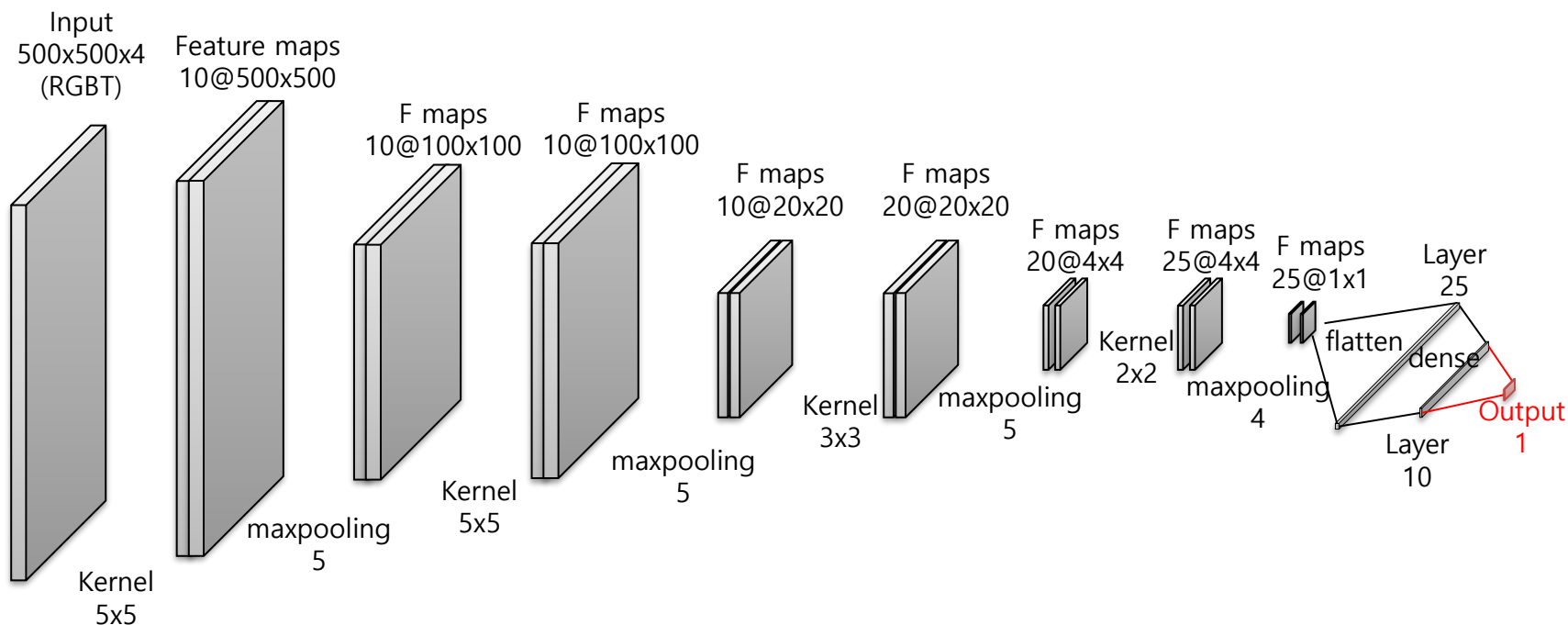
```
net = layers.Flatten()(net)
```

외벽 학습 CNN 모델 Source Code



```
net = layers.Dense(10, activation='relu')(net)
net = layers.Dropout(0.5)(net)
```

외벽 학습 CNN 모델 Source Code



```
net = layers.Dense(1, activation='sigmoid')(net)
```

외벽 학습 CNN 모델

학습결과를 predict 함수를 이용하여 확인 할 수 있음

result_training - NumPy object ...

| | 0 | 1 |
|----|---|-------------|
| 0 | 1 | 0.994002 |
| 1 | 0 | 0.00129718 |
| 2 | 0 | 0.00539643 |
| 3 | 0 | 1.82699e-07 |
| 4 | 0 | 0.0019688 |
| 5 | 1 | 0.992645 |
| 6 | 0 | 3.53e-06 |
| 7 | 0 | 0.0108379 |
| 8 | 0 | 3.11721e-07 |
| 9 | 1 | 0.999981 |
| 10 | 1 | 0.990927 |
| 11 | 0 | 5.51026e-06 |
| 12 | 0 | 3.6702e-05 |
| 13 | 0 | 2.56262e-09 |

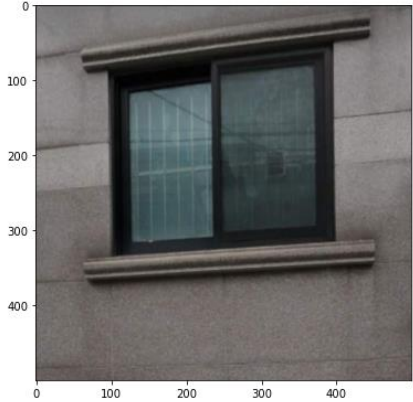
Format

Resize

☒ Background color

Save and Close

Close



실제 : 대리석 // 예측 : 대리석

다음

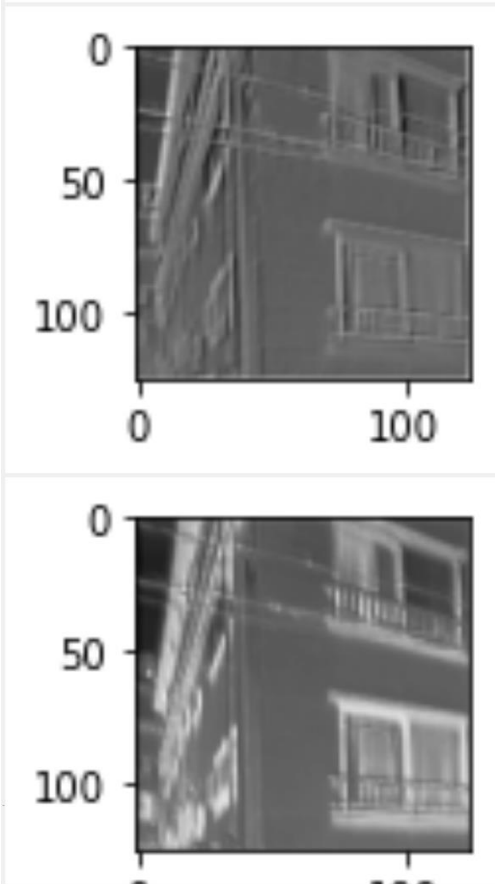


실제 : 벽돌 // 예측 : 벽돌

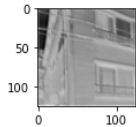
외벽 학습 CNN 모델

각 단계별 데이터 변화를 모델을 잘라서 확인할 수 있음

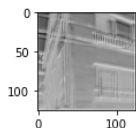
Conv 1



Conv 3



다음:
7번째 레이어 : conv2d_12 (1번째 데이터)



flatten

```
[[ -30.289343  -29.561031  -33.822357  -25.433182   11.5601635 -29.130713  
  -19.346176  -24.334515   -4.015135   41.16101  -12.693679  -21.961527  
  -24.564245   -6.401454   44.127457  -28.485262  -27.805992  -33.388123  
  -10.581995   31.267431  -28.679811  -20.988543  -20.961002  -15.3293085  
    2.9477847]]
```

Flatten Layer 분석

25차원으로 데이터 축소 효과 유사한 벡터를 군집화 하면 이미지 차이 확인 가능

클러스터 1



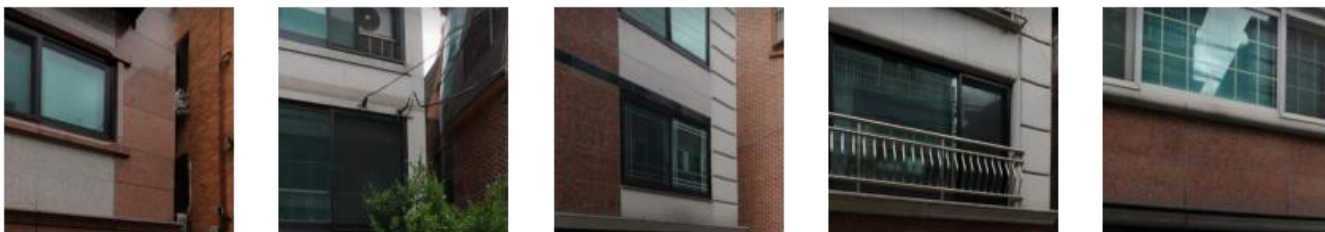
클러스터 2



클러스터 3



클러스터 4



학습과정에 Validationt Set 지정하기

```
hist = model.fit(training_x, training_y, epochs=100, validation_data=(testing_x, testing_y))
```

학습과정에 저장하기

Callback 함수 이용

```
from keras.callbacks import ModelCheckpoint
```

```
filename = 'gdrive/My Drive/Colab Notebooks/models/save-{epoch:02d}-{val_loss:.4f}.h5'
```

```
checkpoint = ModelCheckpoint(filename,          # file명을 지정합니다
                             monitor='val_loss', # val_loss 값이 개선되었을때 호출됩니다
                             verbose=1,         # 로그를 출력합니다
                             save_best_only=True, # 가장 best 값만 저장합니다
                             mode='auto'        # auto는 알아서 best를 찾습니다. min/max
                             )
```

```
hist = model.fit(training_x, training_y, epochs=100, validation_data=(testing_x, testing_y), callbacks=[checkpoint])
```