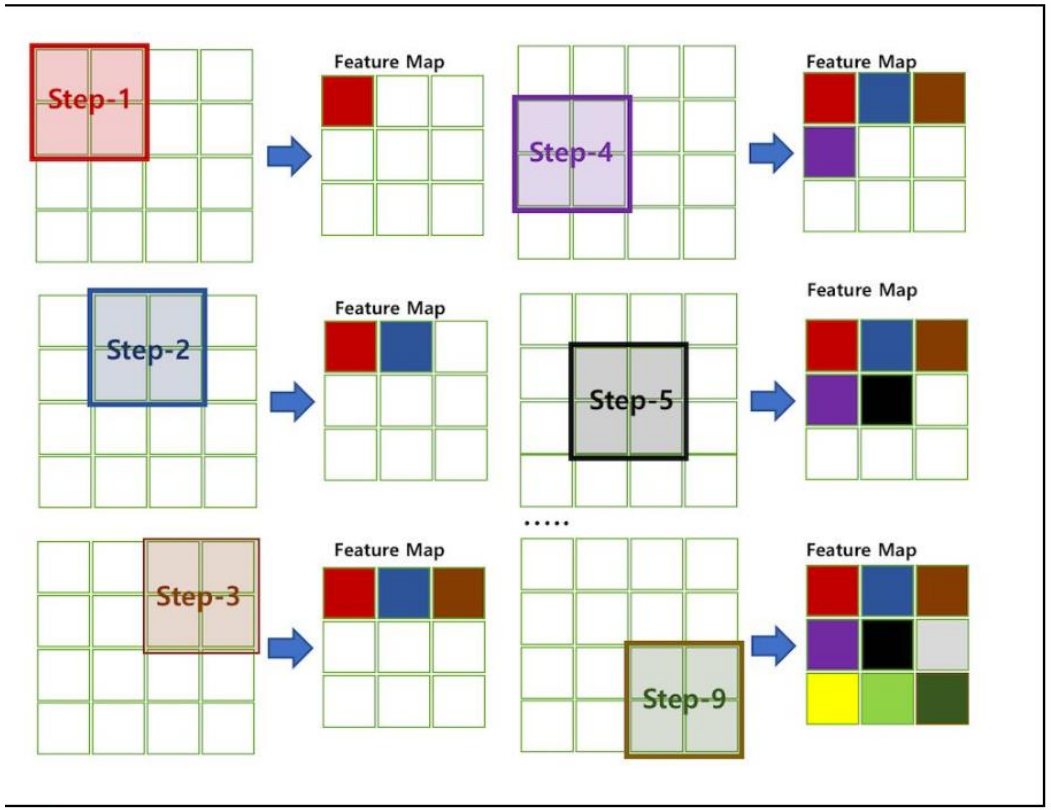
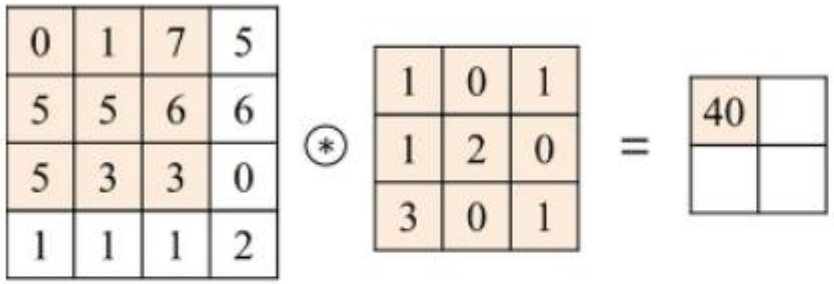
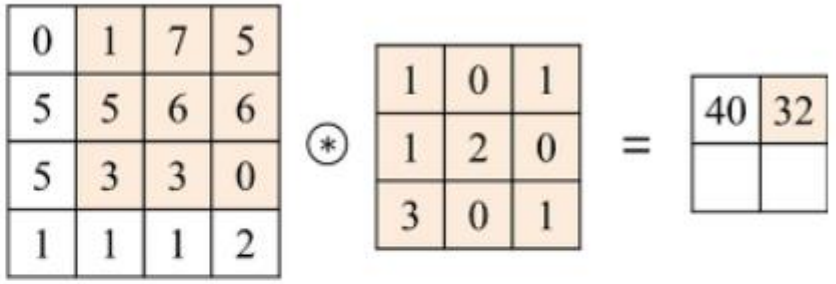
CNN(Convolutional Neural Network)

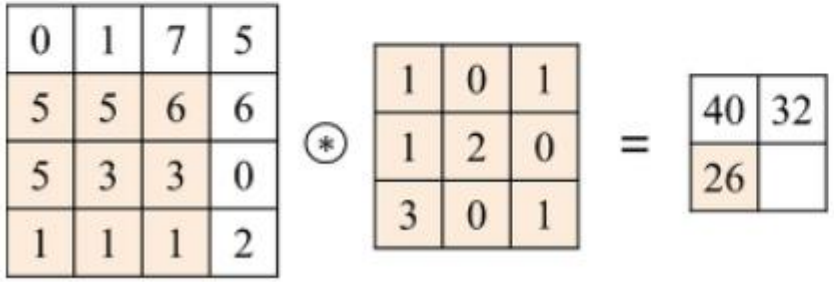


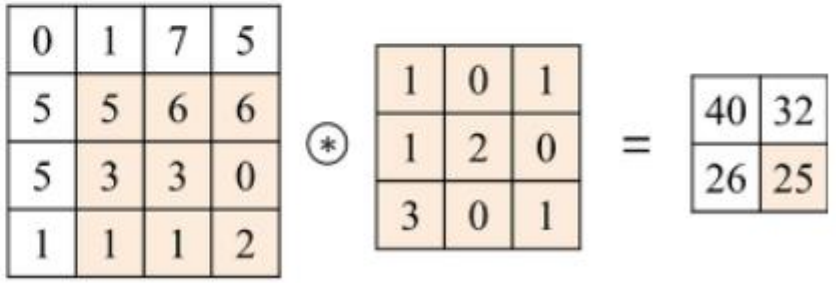
- 컨볼루션 연산
- 스트라이드
- 패딩

컨볼루션 연산

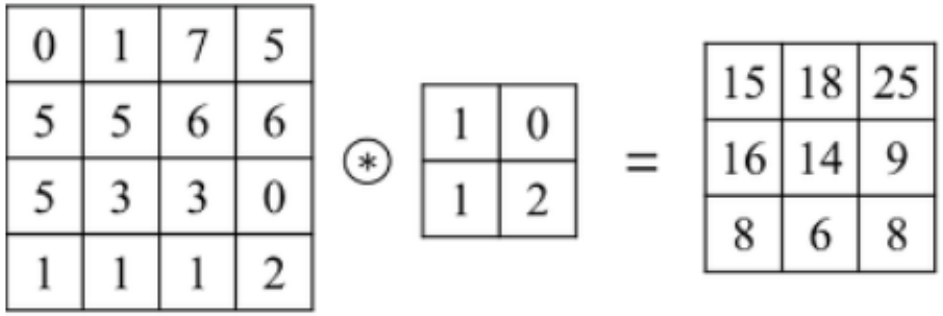

0 1 7 5
5 5 6 6
5 3 3 0
1 1 1 2
⊛
1 0 1
1 2 0
3 0 1
=
40


0 1 7 5
5 5 6 6
5 3 3 0
1 1 1 2
⊛
1 0 1
1 2 0
3 0 1
=
40 32

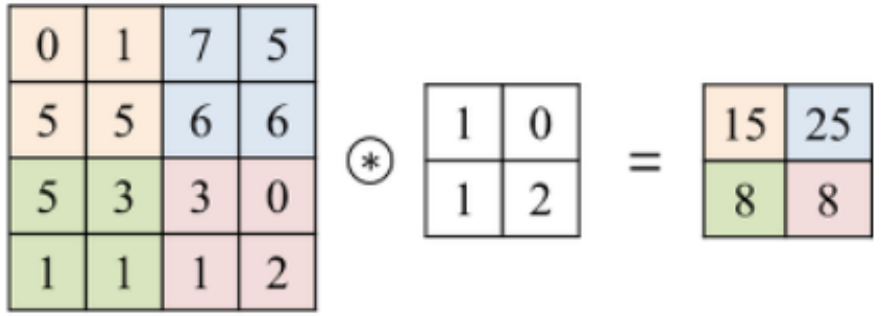

0 1 7 5
5 5 6 6
5 3 3 0
1 1 1 2
⊛
1 0 1
1 2 0
3 0 1
=
40 32
26


0 1 7 5
5 5 6 6
5 3 3 0
1 1 1 2
⊛
1 0 1
1 2 0
3 0 1
=
40 32
26 25

스트라이드

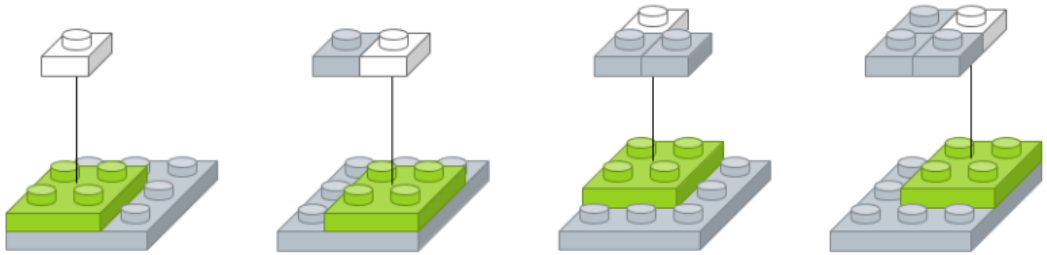


스트라이드 : 1

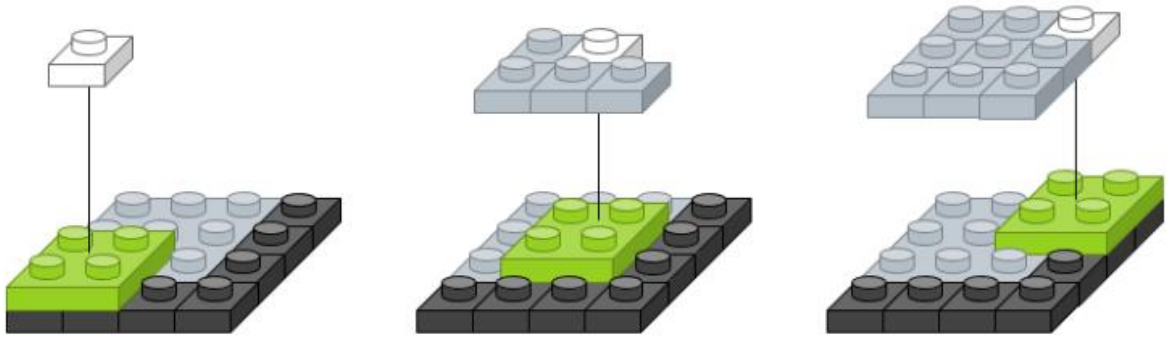


스트라이드 : 2

패딩

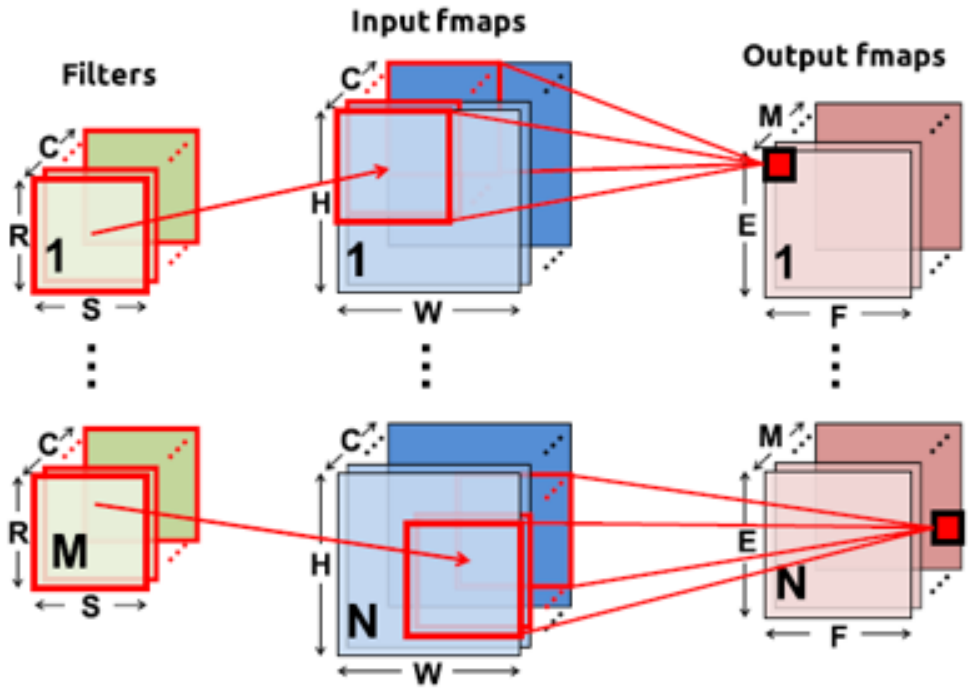


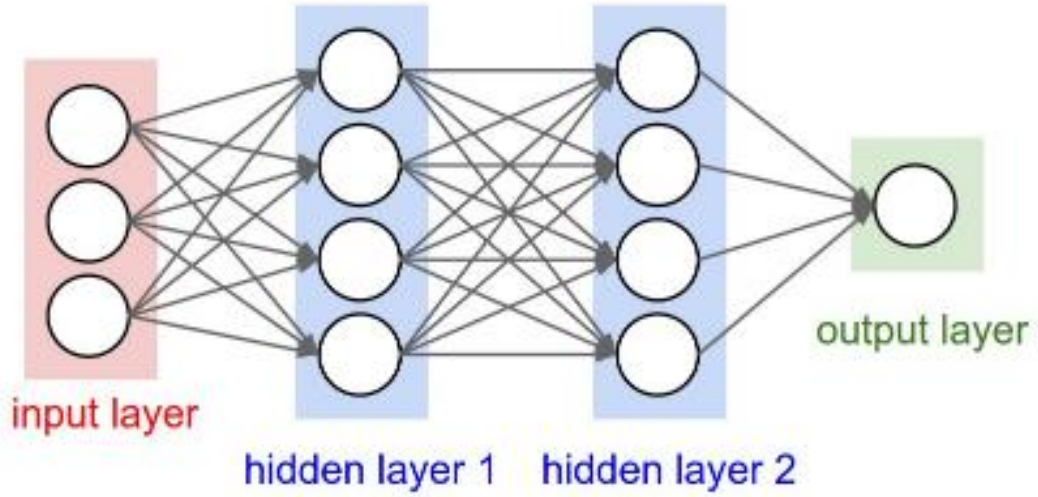
패딩을 사용하지 않는 경우
Feature map : 2x2



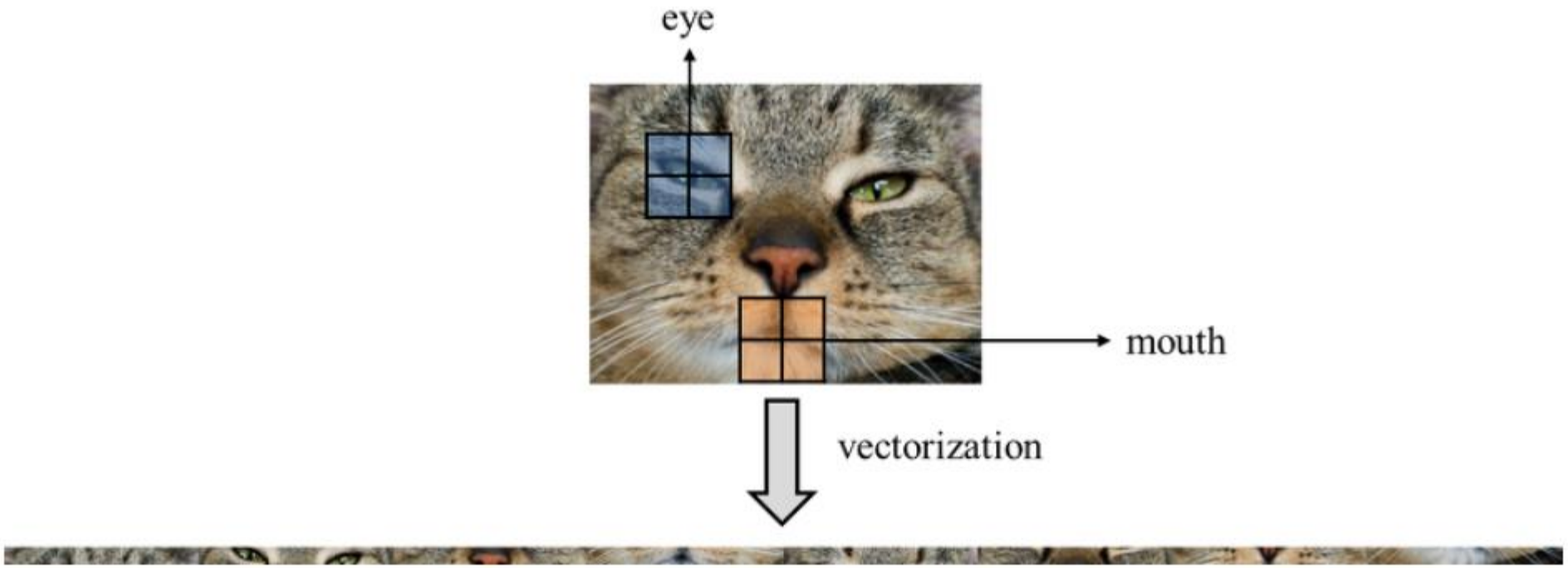
패딩을 사용한 경우
Feature map : 3x3

CNN(Convolutional Neural Network) vs Fully-Connected layer

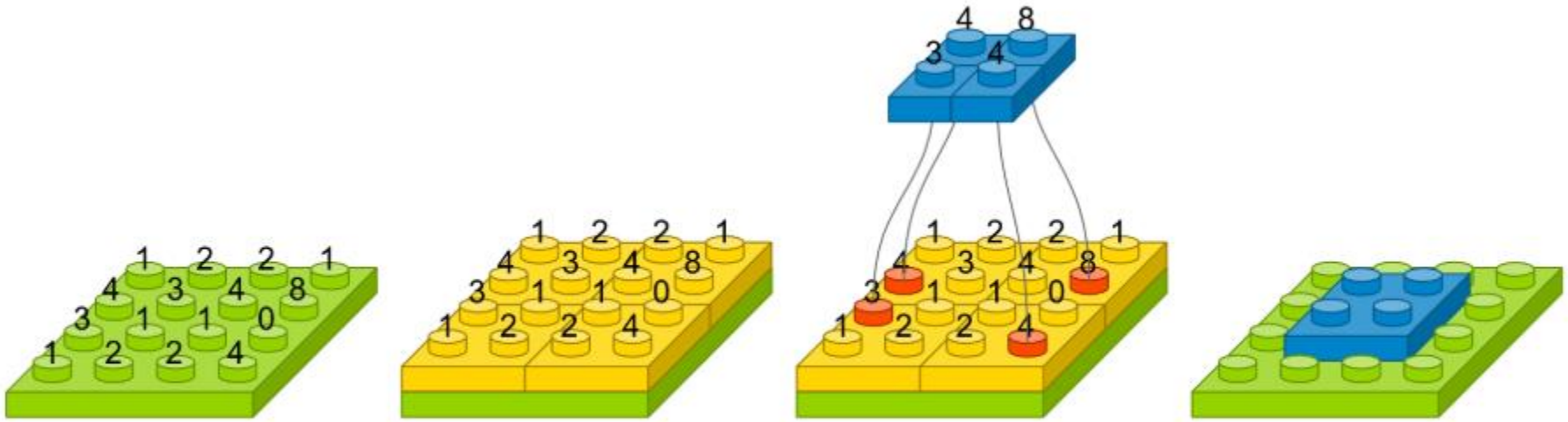




각 레이어의 입출력 데이터의 형상 유지
이미지의 공간 정보를 유지하면서 인접 이미지와의 특징을 효과적으로 인식



추출한 이미지의 특징을 모으고 강화하는 Pooling 레이어

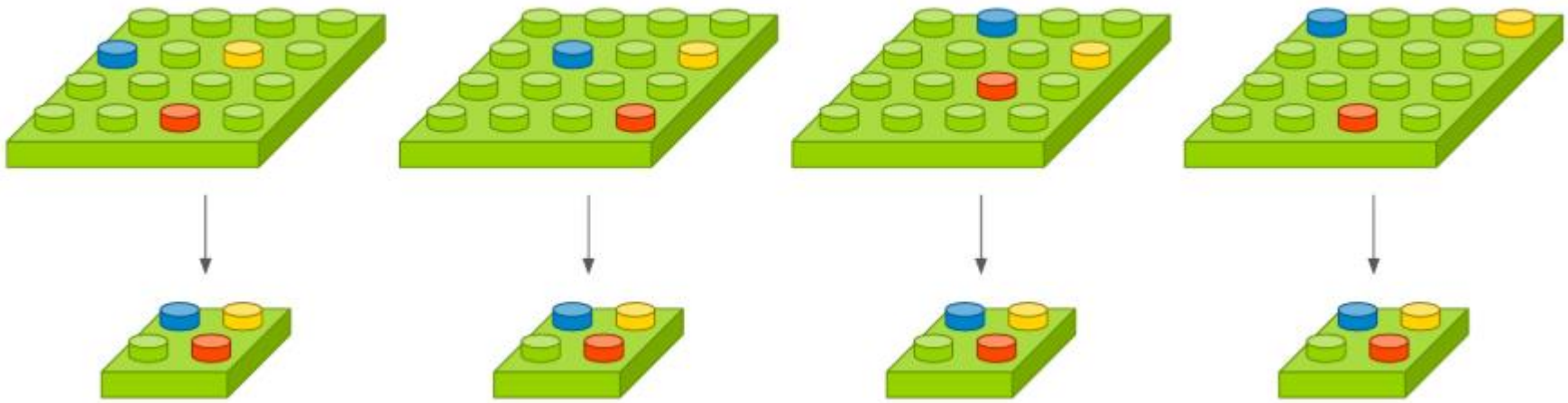


input image 크기 4x4

Pooling 크기 2x2
(stride 크기 2)

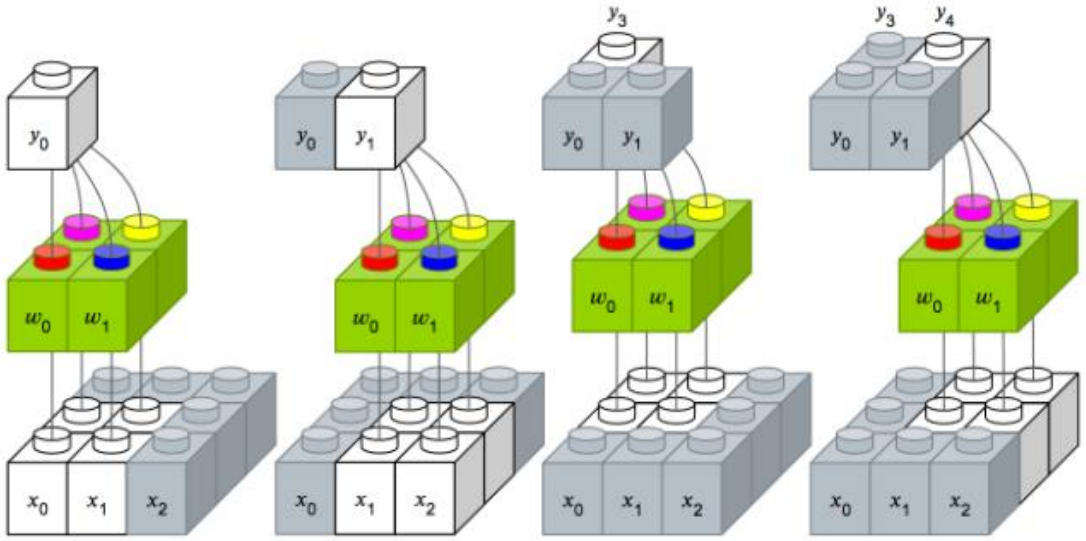
Feature map 크기 2x2

추출한 이미지의 특징을 모으고 강화하는 Pooling 레이어

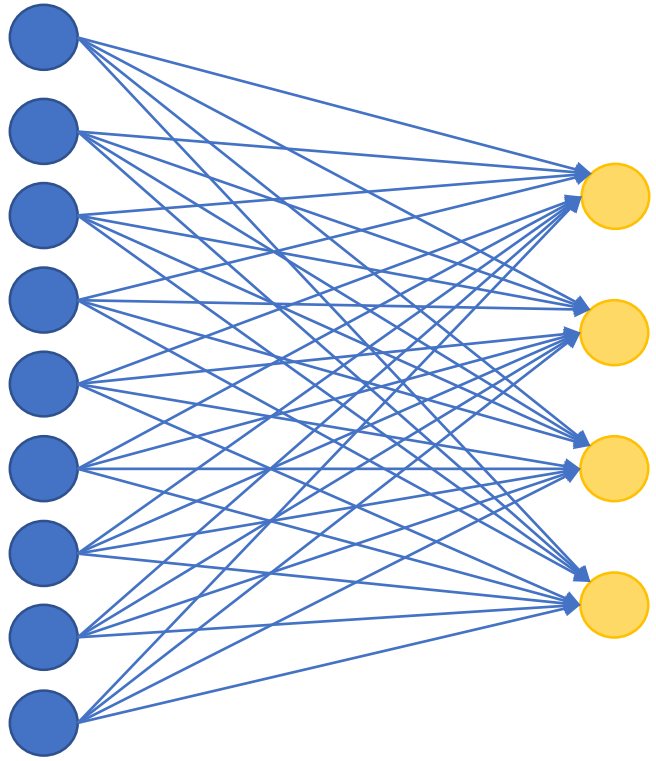


Input의 변화에 덜 민감하다

필터를 공유 파라미터로 사용하기 때문에, 일반 인공신경망과 비교하여 학습 파라미터가 매우 적음

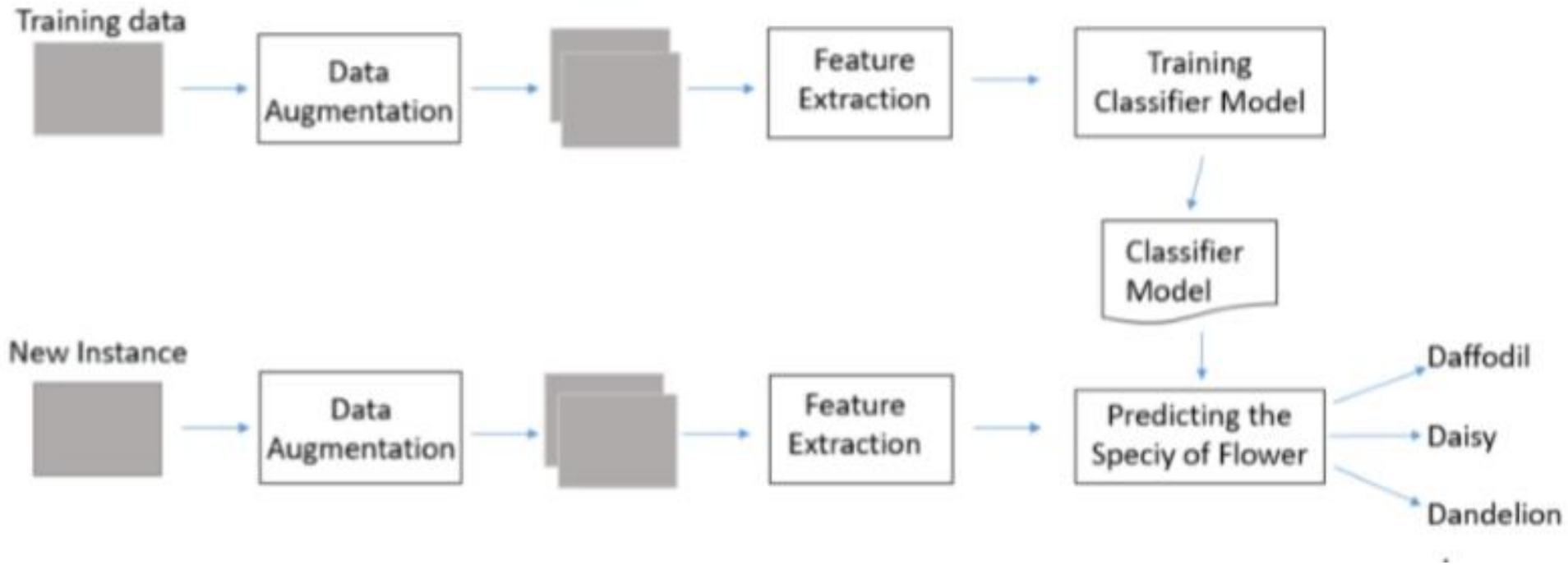


Input image 크기 3x3
Filter 크기 2x2 (stride=1)
Feature map 크기 2x2



Input 크기 9
Weight 개수 36
Output 크기 4

Team Project



Data Augmentation

```
train_datagen = ImageDataGenerator(rescale=1./255,
                                   vertical_flip=True,      #수직 대칭 이미지를 50% 확률로 만들어 추가합니다.
                                   width_shift_range=0.1,   #전체 크기의 10% 범위에서 좌우로 이동합니다.
                                   height_shift_range=0.1, #마찬가지로 위, 아래로 이동합니다.
                                   #rotation_range=5,
                                   #shear_range=0.7,
                                   #zoom_range=[0.9, 2.2],
                                   #vertical_flip=True,
                                   fill_mode='nearest')
```

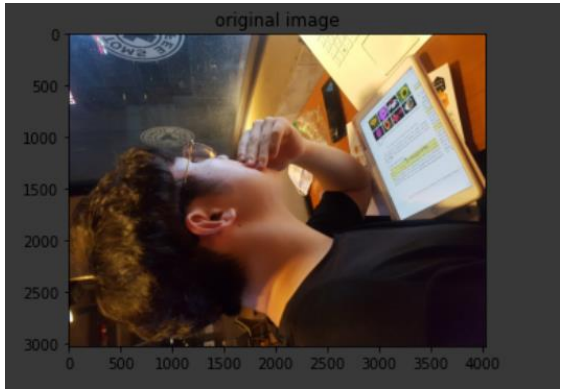
Fig. 3: Samples of vertically flipped images

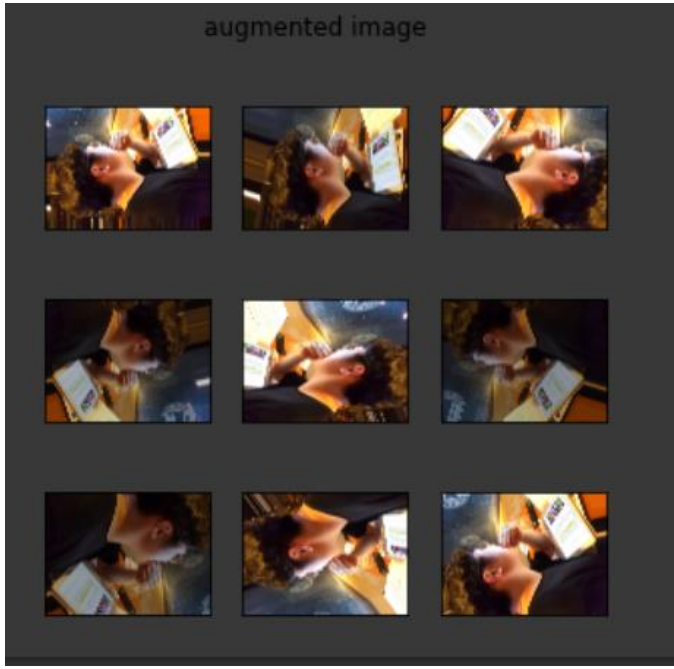
technique. We have doubled the number of images in both datasets by vertically flipping all the images.

Data Augmentation

```python
from tensorflow.keras.preprocessing.image import load_img, img_to_array, ImageDataGenerator
import matplotlib.pyplot as plt
import numpy as np


train_datagen = ImageDataGenerator(horizontal_flip = True,
                                   vertical_flip = True,
                                   shear_range = 0.5,
                                   brightness_range = [0.5, 1.5],
                                   zoom_range = 0.2,
                                   width_shift_range = 0.1,
                                   height_shift_range = 0.1,
                                   rotation_range = 30,
                                   fill_mode = 'nearest'
                                  )
```





Modeling

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(500,350,3),activation='relu'))
model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(32, (3, 3),activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(64, (3, 3),activation='relu'))
model.add(MaxPooling2D(pool_size=2))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(4))
model.add(Activation('softmax'))

#모델 컴파일
model.compile(loss='categorical_crossentropy', optimizer=optimizers.Adam(learning_rate=0.0002), metrics=['accuracy'])

#모델 실행
history = model.fit_generator(
       train_generator,
       steps_per_epoch=30,
       epochs=100,
       validation_data=test_generator,
       validation_steps=10)
```

Epoch : 전체 train 데이터를 이용하여 한 바퀴 돌며 학습하는 것을 1회 epoch
Step : Weight와 Bias를 1회 업데이트 하는 것을 1 Step
Batch Size : 전체 train 데이터 중 batch size만큼 모아서 업데이트 함

Step = (train_data_number)/(batch size) *(epoch_number)

Result

```
30/30 [==============================] - 6s 189ms/step - loss: 0.2531 - accuracy: 0.9000 - val_loss: 0.4105 - val_accuracy: 0.9000
Epoch 87/100
30/30 [==============================] - 6s 190ms/step - loss: 0.3212 - accuracy: 0.8867 - val_loss: 0.2133 - val_accuracy: 0.9000
Epoch 88/100
30/30 [==============================] - 6s 188ms/step - loss: 0.2533 - accuracy: 0.9200 - val_loss: 0.3340 - val_accuracy: 0.8600
Epoch 89/100
30/30 [==============================] - 6s 193ms/step - loss: 0.3427 - accuracy: 0.8800 - val_loss: 0.2534 - val_accuracy: 0.9000
Epoch 90/100
30/30 [==============================] - 6s 192ms/step - loss: 0.2474 - accuracy: 0.8800 - val_loss: 0.2267 - val_accuracy: 0.9200
Epoch 91/100
30/30 [==============================] - 6s 190ms/step - loss: 0.3824 - accuracy: 0.8600 - val_loss: 0.1066 - val_accuracy: 0.9600
Epoch 92/100
30/30 [==============================] - 6s 190ms/step - loss: 0.2631 - accuracy: 0.9333 - val_loss: 0.2452 - val_accuracy: 0.9400
Epoch 93/100
30/30 [==============================] - 6s 192ms/step - loss: 0.2971 - accuracy: 0.8800 - val_loss: 0.2678 - val_accuracy: 0.8800
Epoch 94/100
30/30 [==============================] - 6s 193ms/step - loss: 0.3426 - accuracy: 0.9000 - val_loss: 0.5770 - val_accuracy: 0.8200
Epoch 95/100
30/30 [==============================] - 6s 192ms/step - loss: 0.2512 - accuracy: 0.9067 - val_loss: 0.2567 - val_accuracy: 0.8800
Epoch 96/100
30/30 [==============================] - 6s 191ms/step - loss: 0.3137 - accuracy: 0.8867 - val_loss: 0.0980 - val_accuracy: 0.9800
Epoch 97/100
30/30 [==============================] - 6s 190ms/step - loss: 0.2930 - accuracy: 0.9067 - val_loss: 0.1889 - val_accuracy: 0.9400
Epoch 98/100
30/30 [==============================] - 6s 190ms/step - loss: 0.3856 - accuracy: 0.8667 - val_loss: 0.3186 - val_accuracy: 0.9000
Epoch 99/100
30/30 [==============================] - 6s 194ms/step - loss: 0.2682 - accuracy: 0.9267 - val_loss: 0.2750 - val_accuracy: 0.9200
Epoch 100/100
30/30 [==============================] - 6s 191ms/step - loss: 0.2114 - accuracy: 0.9067 - val_loss: 0.3013 - val_accuracy: 0.9000
```