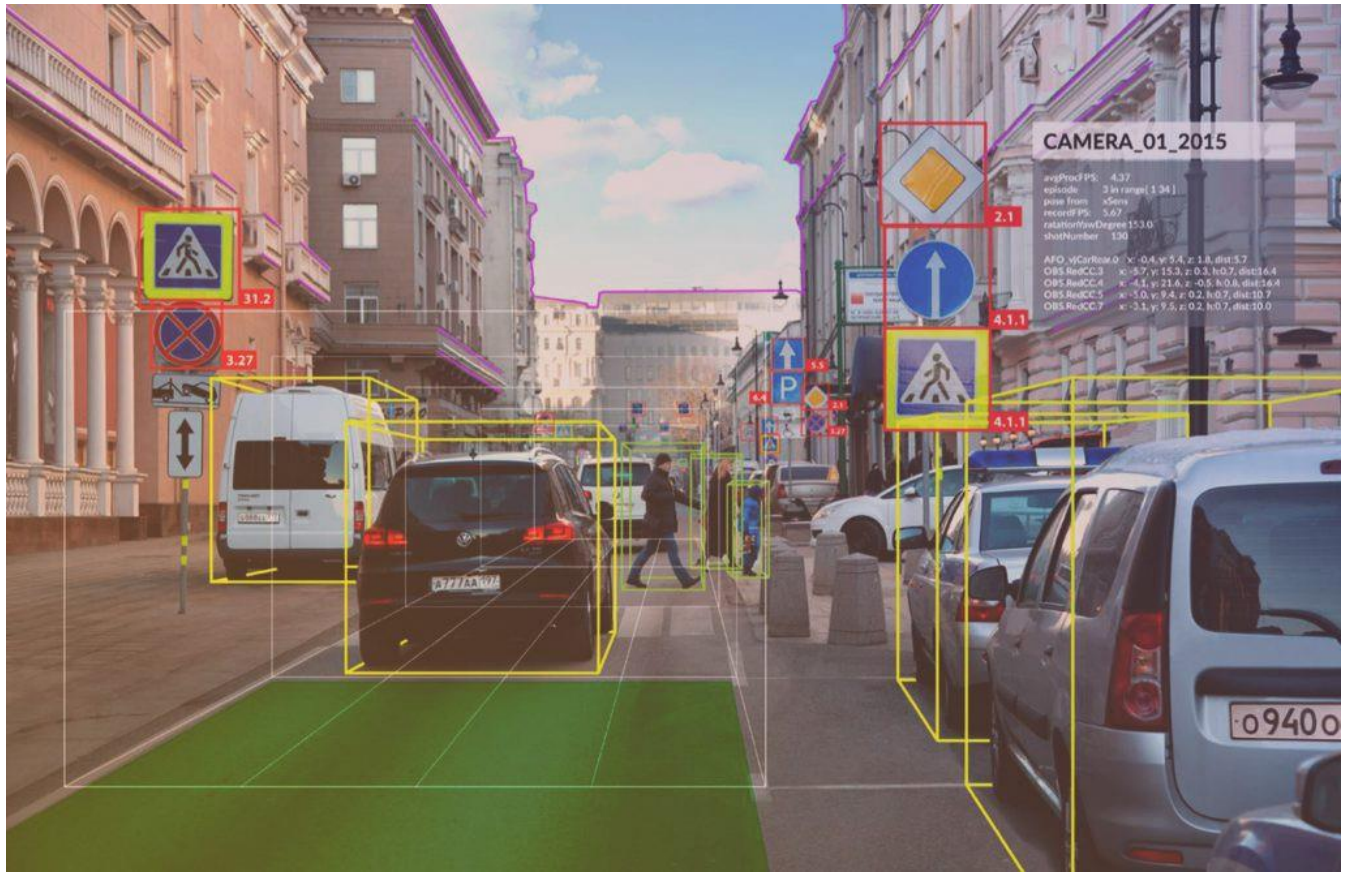
CNN을 사용한 컴퓨터 비전

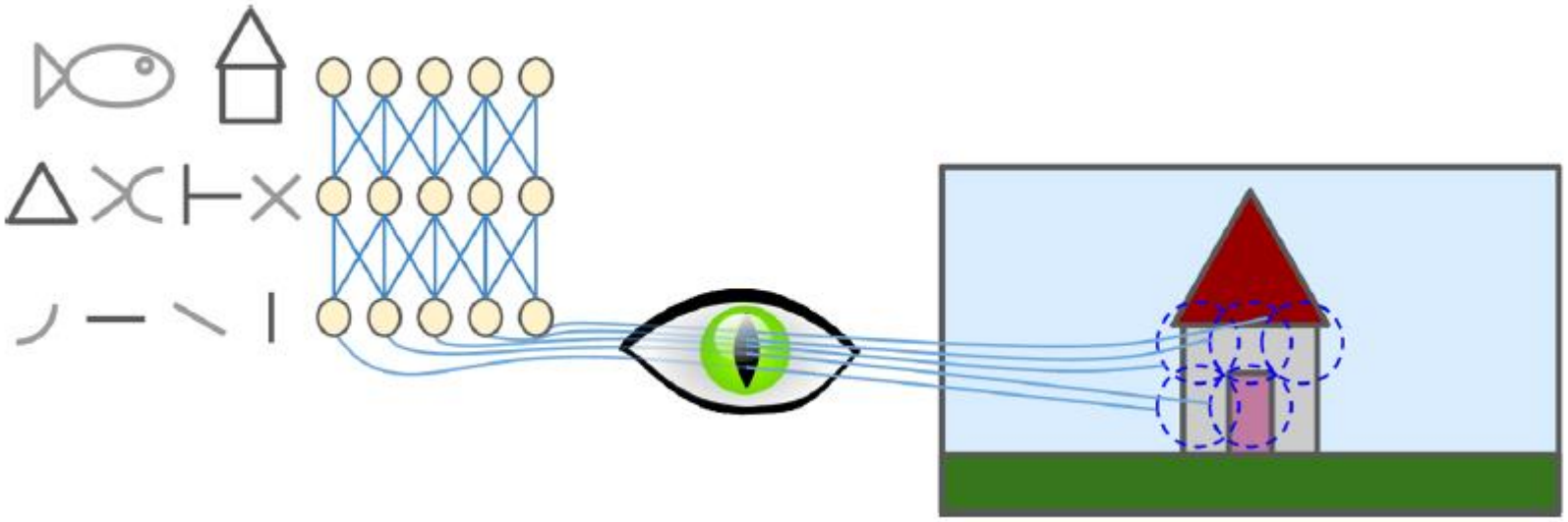


박 경 규

본 챕터 전체 내용의 출처는 도서 "핸즈온 머신러닝 2판(Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition)"입니다.

시각 피질(Visual Cortex) 구조

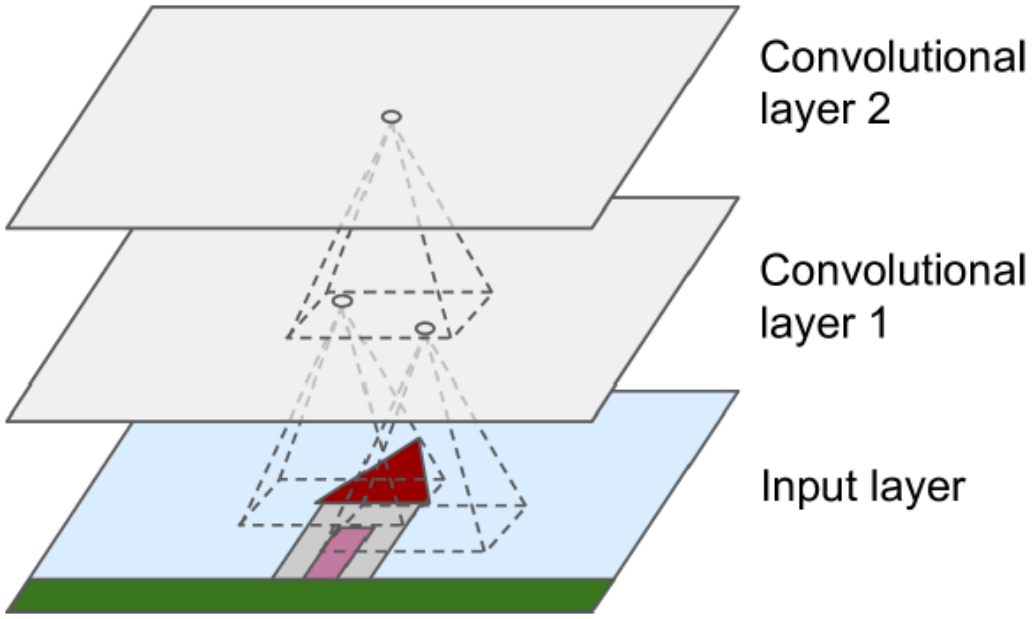
합성곱신경망은 대뇌의 시각피질 연구에서 시작되었습니다.
시각피질의 국부수용장(local receptive field)은 수용장이라 불리는 시야의 작은 영역에 있는 특정 패턴에 반응합니다.
시각신호가 연속적인 뇌 모듈을 통과하면서 뉴런이 더 큰 수용장에 있는 복잡한 패턴에 반응합니다.



- 일반적인 완전 연결층의 심층 신경망은 MNIST 같은 작은 이미지에서는 잘 작동하지만 큰 이미지에서는 아주 많은 파라미터가 만들어지기 때문에 문제가 됩니다. 예를 들어 100×100 이미지는 픽셀 10,000개로 이루어져 있습니다. 여기에 첫 번째 은닉층을 뉴런 1,000개로 만들어도 연결이 총 1천만개가 생깁니다. 이는 첫 번째 은닉층일 뿐입니다.
- CNN은 층을 부분적으로 연결하고 가중치를 공유하여 이 문제를 해결합니다.

합성곱 층

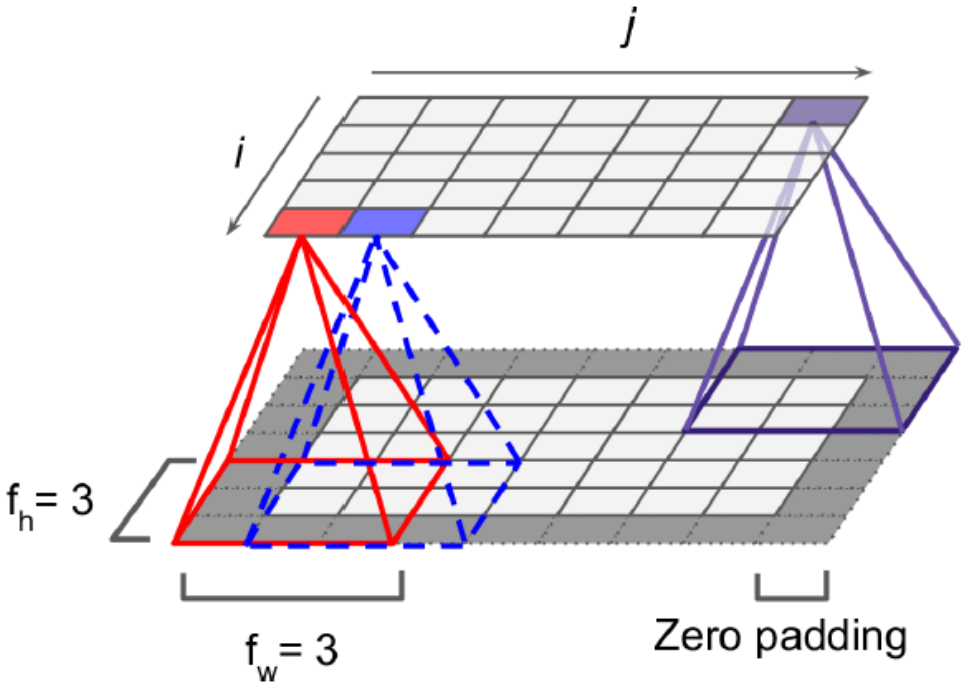
합성곱 층의 뉴런은 입력 이미지의 모든 픽셀이 아닌 합성곱 층 뉴런의 수용장 안에 있는 픽셀에만 연결됩니다. 이런 구조는 네트워크가 첫번째 은닉층에서는 작은 저수준 특성에 집중하고 그 다음 은닉층에서는 더 큰 수준 특성으로 조합해 나가도록 도와 줍니다.



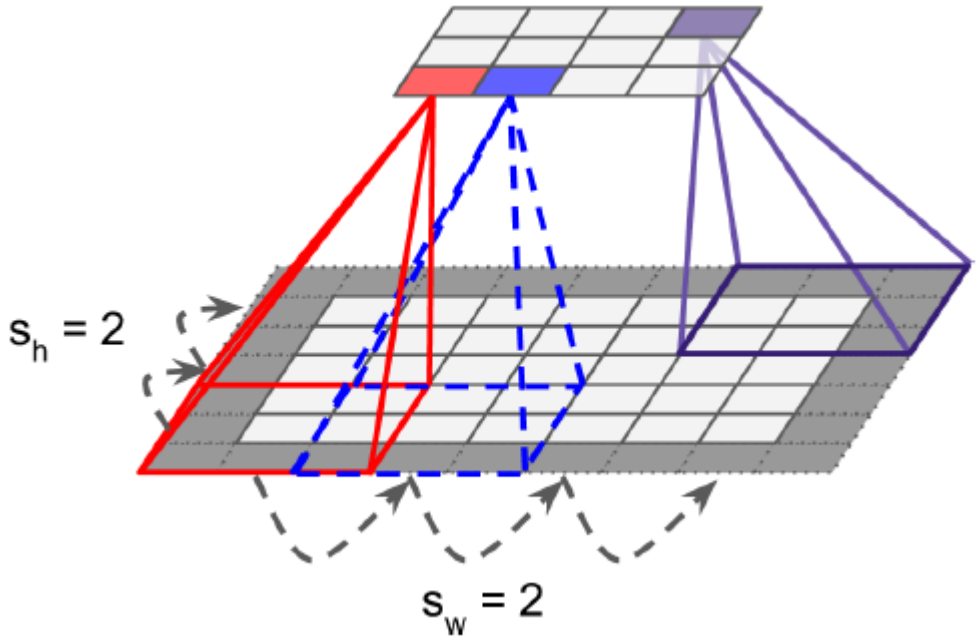
제로패딩, 스트라이드

높이와 너비를 이전 층과 같게 하기 위해 입력 이미지 주위에 0을 추가하는 것을 제로패딩이라고 합니다. 한 수용장과 다음 수용장 사이의 간격을 스트라이드(stride)라고 하며 입력층을 작은 층에 연결하면 모델의 계산복잡도가 크게 낮아집니다.

■ 제로패딩(zero padding)

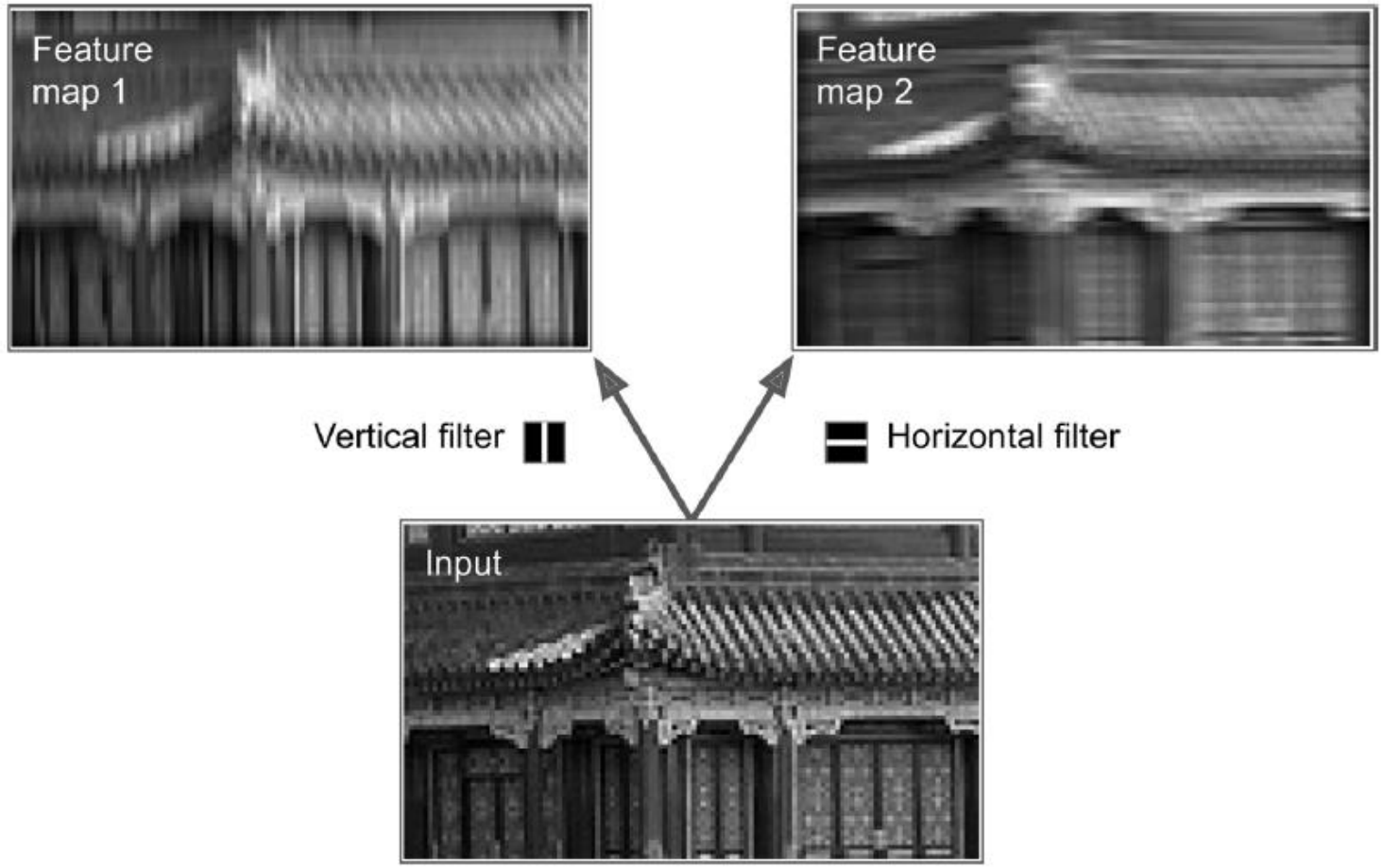


■ 스트라이드(stride)



필터

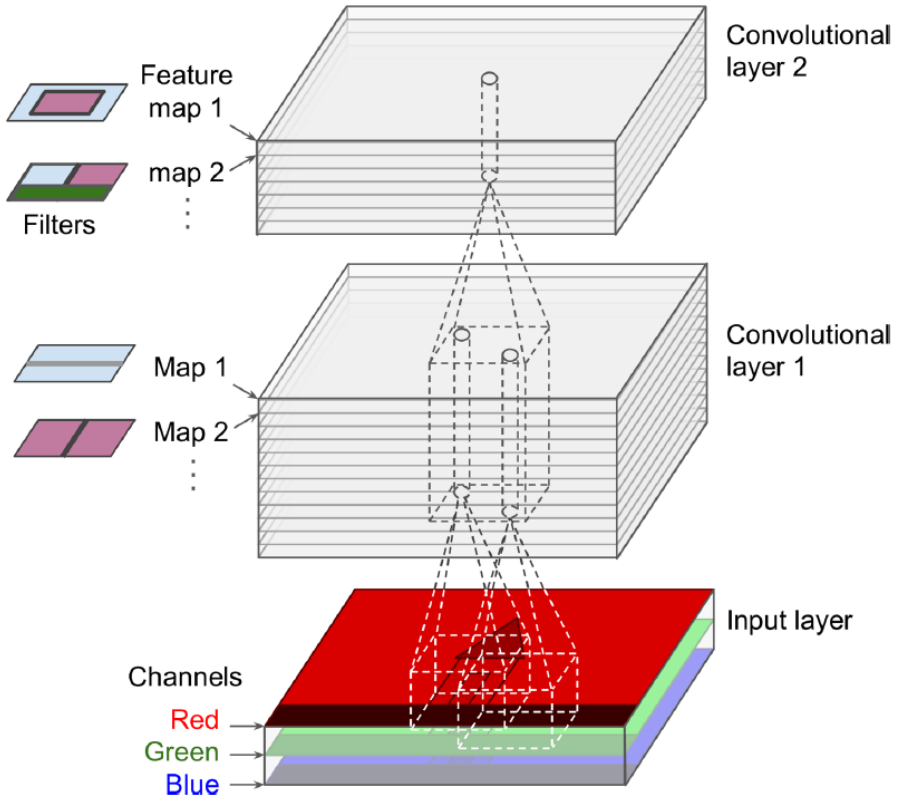
뉴런의 가중치는 수용장 크기의 작은 이미지로 표현될 수 있습니다. 수동으로 필터를 정의할 필요가 없으며 훈련하는 동안 합성곱 층이 자동으로 해당 문제에 가장 유용한 필터를 찾고 상위 층은 이들을 연결하여 더 복잡한 패턴을 학습합니다.



특성 맵 쌓기

합성곱 층은 여러 가지 필터를 가지고 필터마다 하나의 특성맵을 출력하므로 3D로 표현하며, 각 특성 맵의 픽셀은 하나의 뉴런에 해당하고 하나의 특성 맵 안에서 모든 뉴런이 같은 파라미터(가중치, 편향)을 공유하여 모델의 전체 파라미터 수를 급격하게 줄여 줍니다.

■ 합성곱 층



■ 합성곱 층에 있는 뉴런의 출력 계산

$$z_{i,j,k} = b_k + \sum_{u=0}^{f_h-1} \sum_{v=0}^{f_w-1} \sum_{k'=0}^{f_{n'}-1} x_{i',j',k'} \cdot w_{u,v,k',k} \quad \text{with} \begin{cases} i' = i \times s_h + u \\ j' = j \times s_w + v \end{cases}$$

$z_{i,j,k}$ 합성곱 층(/층)의 k 특성 맵에서 i 행, j 열에 위치한 뉴런의 출력

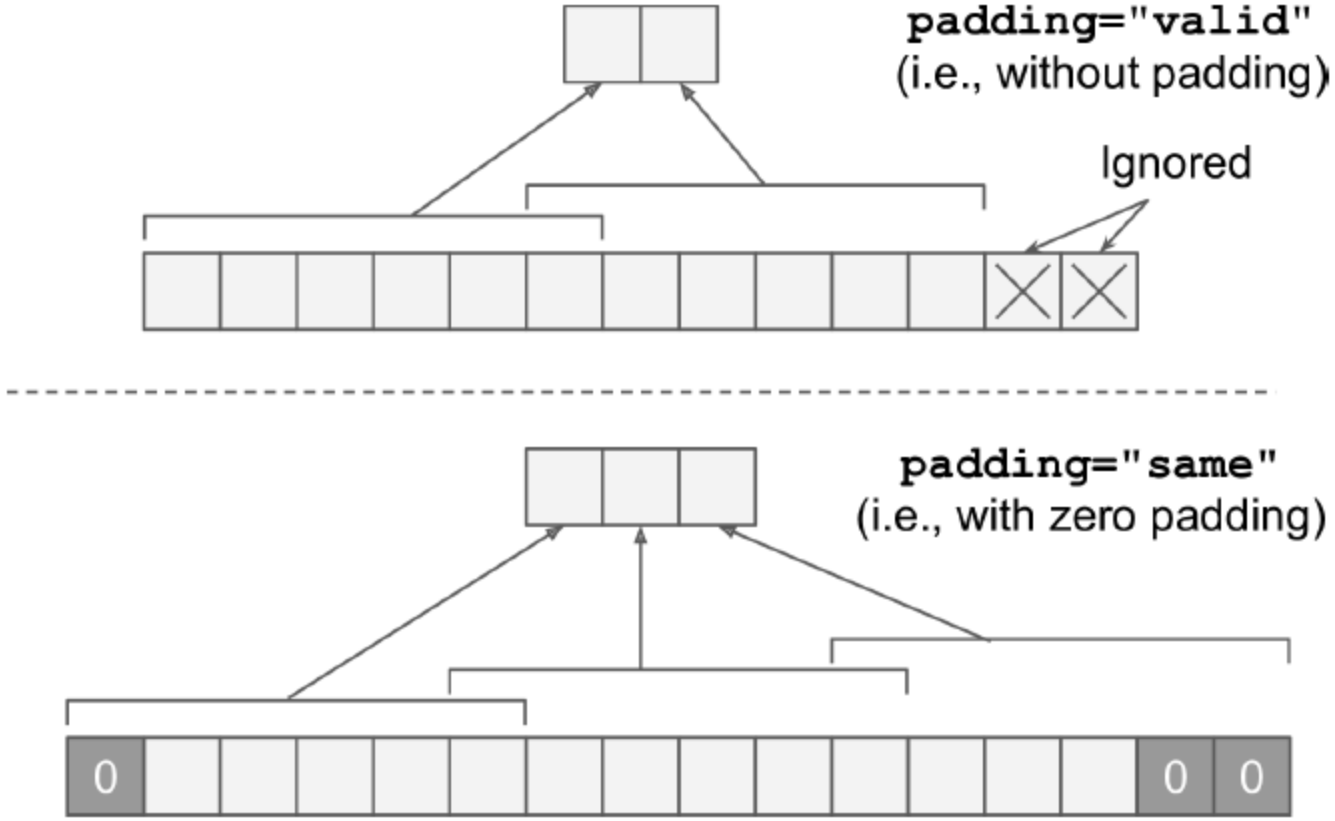
s_h s_w 수직과 수평 스트라이드

f_h f_w 수용장의 높이와 너비

합성곱 층 구현

```
from sklearn.datasets import load_sample_image

# Load sample images
china = load_sample_image("china.jpg") / 255
flower = load_sample_image("flower.jpg") / 255
images = np.array([china, flower])
batch_size, height, width, channels = images.shape

# Create 2 filters
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1  # vertical line
filters[3, :, :, 1] = 1  # horizontal line

outputs = tf.nn.conv2d(images, filters, strides=1, padding="SAME")

plt.imshow(outputs[0, :, :, 1], cmap="gray") # plot 1st image's 2nd feature map
plt.show()
```

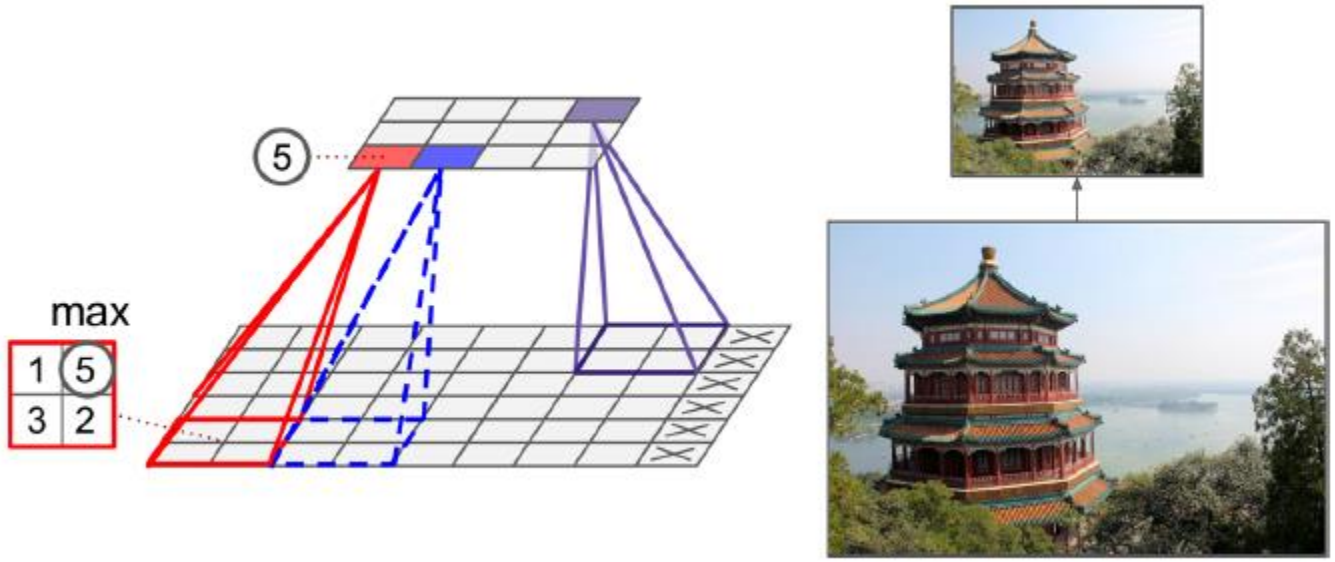
합성곱 층 구현




```
conv = keras.layers.Conv2D(filters=32, kernel_size=3, strides=1,
                           padding="same", activation="relu")
```

폴링 층

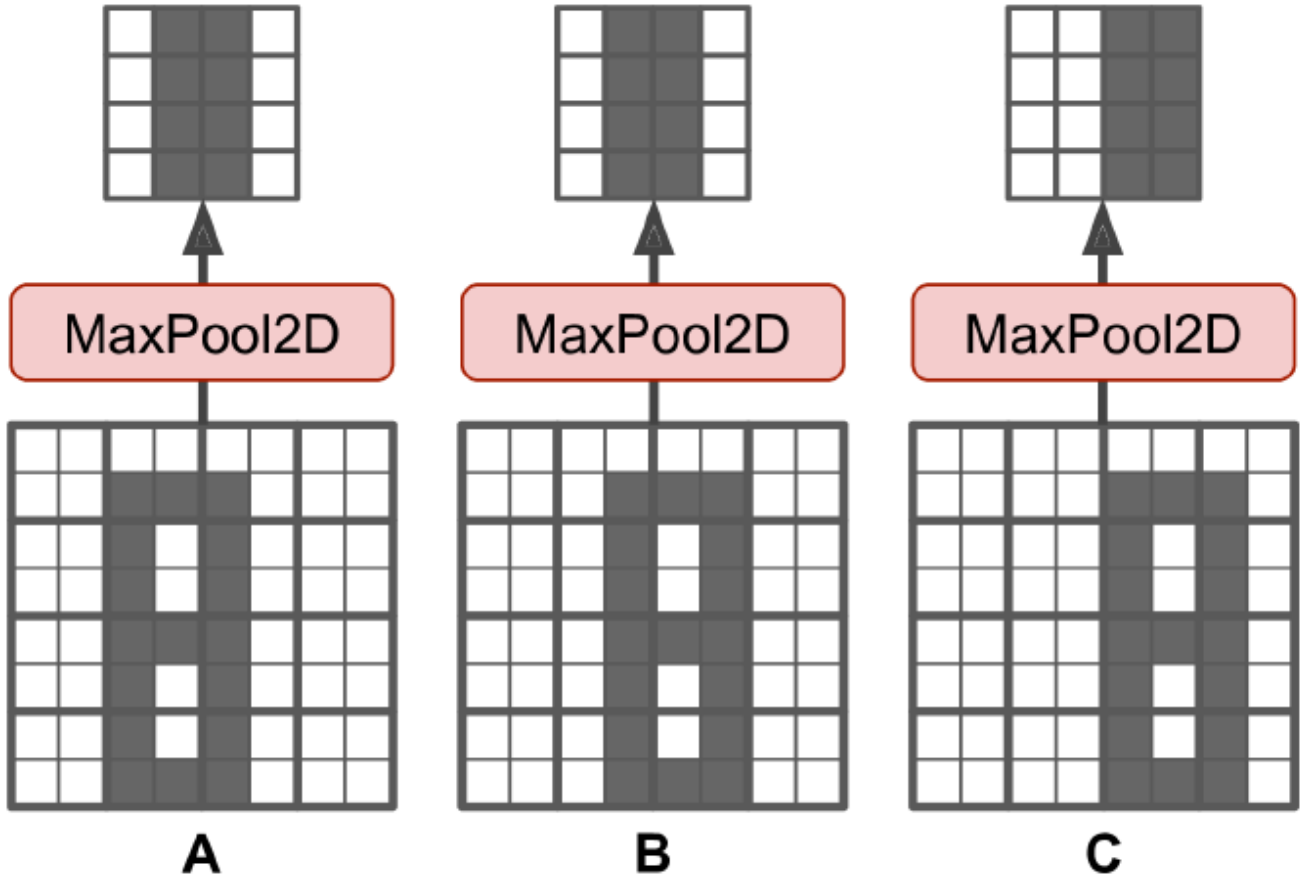
폴링 층은 계산량, 메모리 사용량, 파라미터 수를 줄이기 위해 입력 이미지의 sub sample(축소본)을 만드는 것입니다.



폴링층

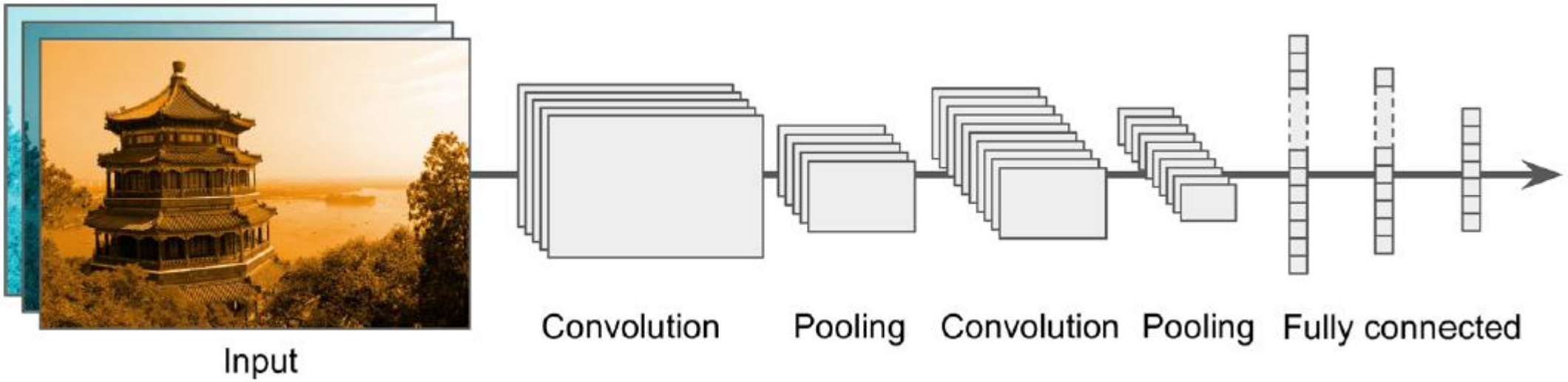
최대 폴링은 작은 변화에도 일정 수준의 불변성을 만들어 줍니다. 이미지 A, B, C가 2×2 커널, 스트라이드 2인 최대 폴링 층을 통과하면 A, B의 출력은 동일하고, 몇 개 층마다 폴링 층을 추가하면 전체적으로 일정 수준의 불변성을 얻을 수 있습니다.

```
max_pool = keras.layers.MaxPool2D(pool_size=2)
```



CNN 구조

**전형적인 구조는 합성곱 층을 몇 개 쌓고, 그 다음 폴링 층을 쌓고, 그 다음에 또 합성곱 층, 다시 폴링층을 쌓는 식입니다.
맨 위층에는 몇 개의 완전 연결 층으로 구성된 일반적인 피드포워드 신경망이 추가되고 마지막 층에서 예측을 출력합니다.**



네트워크를 통과할수록 이미지는 점점 작아지지만, 합성곱 층 때문에 점점 더 깊어지고 더 많은 특성맵을 가집니다.

CNN 구조

패션 MNIST 데이터셋 문제를 해결하기 위한 간단한 CNN입니다.

```
model = keras.models.Sequential([
    keras.layers.Conv2D(64, 7, activation="relu", padding="same",
                        input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(128, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.Conv2D(256, 3, activation="relu", padding="same"),
    keras.layers.MaxPooling2D(2),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(64, activation="relu"),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation="softmax")
])
```

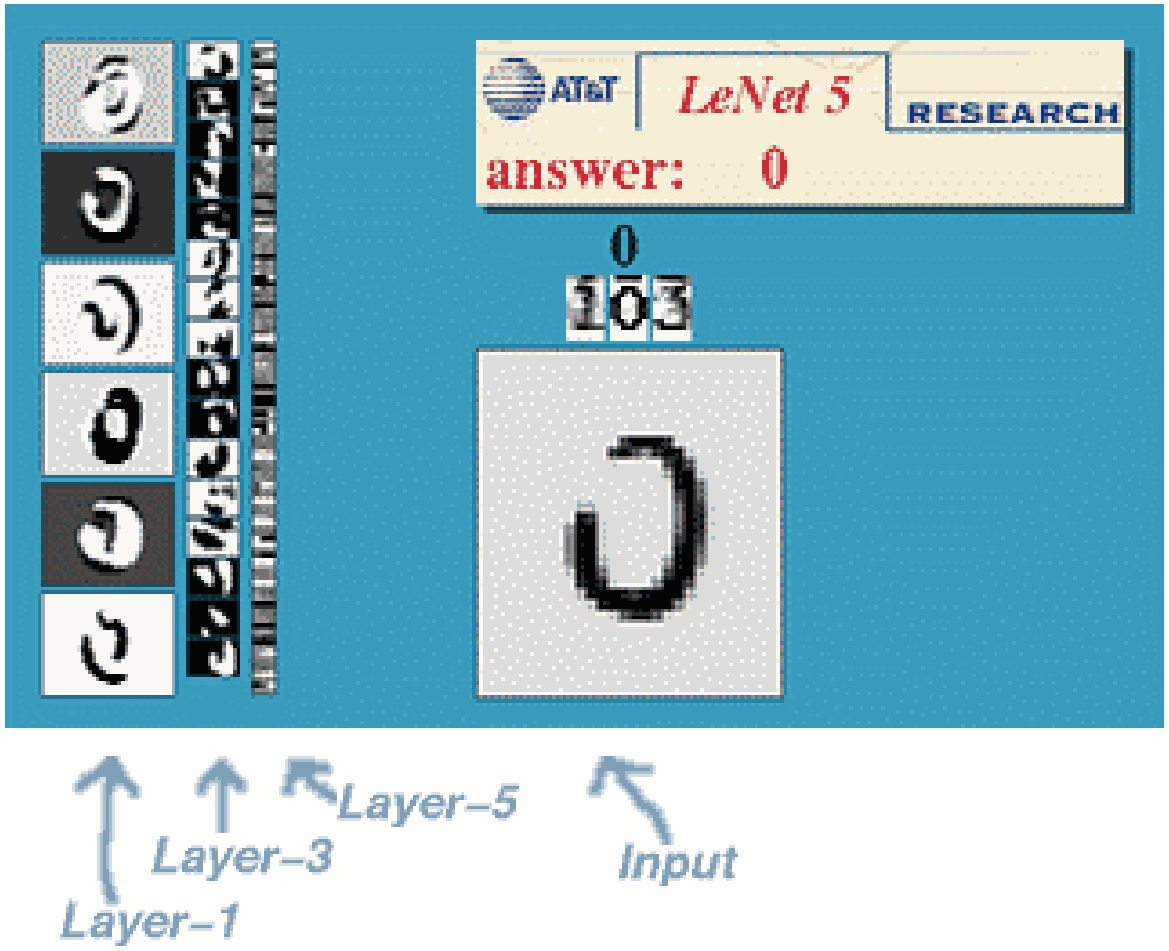
LeNet-5

널리 알려진 CNN 구조로 1998년 얀 르쿤이 만들었으며 손글씨 숫자 인식에 널리 사용 되었습니다.

■ LeNet-5 구조

Layer	Type	Maps	Size	Kernel size	Stride	Activation
Out	Fully connected	–	10	–	–	RBF
F6	Fully connected	–	84	–	–	tanh
C5	Convolution	120	1 × 1	5 × 5	1	tanh
S4	Avg pooling	16	5 × 5	2 × 2	2	tanh
C3	Convolution	16	10 × 10	5 × 5	1	tanh
S2	Avg pooling	6	14 × 14	2 × 2	2	tanh
C1	Convolution	6	28 × 28	5 × 5	1	tanh
In	Input	1	32 × 32	–	–	–

■ LeNet-5 데모



출처 : http://yann.lecun.com/exdb/lenet/index.html

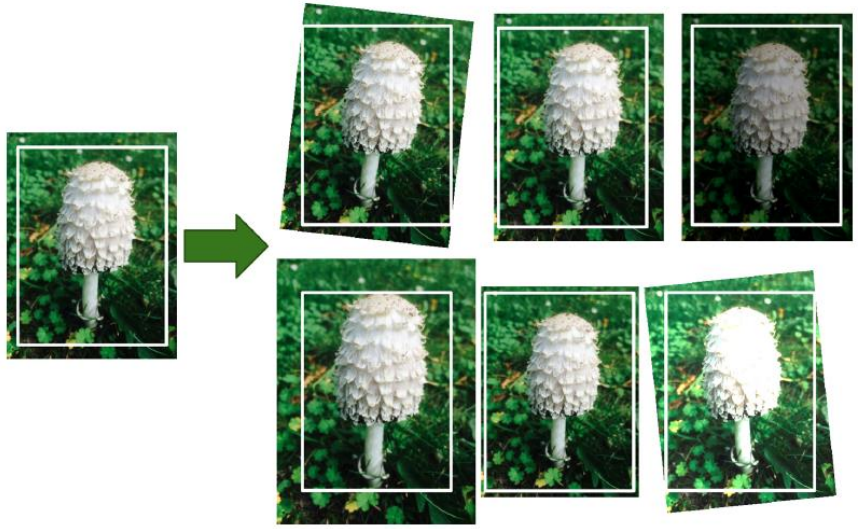
AlexNet

Alex Krizhevsky , Ilya Sutskever, Geoffrey Hinton이 만들었으며, 2012년 이미지넷 대회에서 큰 차이로 우승했습니다.
과대적합을 줄이기 위해 F9와 F10의 출력에 드롭아웃 50% 적용, 훈련 이미지를 데이터 증식으로 변경 하였습니다.

■ AlexNet 구조

Layer	Type	Maps	Size	Kernel size	Stride	Padding	Activation
Out	Fully connected	–	1,000	–	–	–	Softmax
F10	Fully connected	–	4,096	–	–	–	ReLU
F9	Fully connected	–	4,096	–	–	–	ReLU
S8	Max pooling	256	6 × 6	3 × 3	2	valid	–
C7	Convolution	256	13 × 13	3 × 3	1	same	ReLU
C6	Convolution	384	13 × 13	3 × 3	1	same	ReLU
C5	Convolution	384	13 × 13	3 × 3	1	same	ReLU
S4	Max pooling	256	13 × 13	3 × 3	2	valid	–
C3	Convolution	256	27 × 27	5 × 5	1	same	ReLU
S2	Max pooling	96	27 × 27	3 × 3	2	valid	–
C1	Convolution	96	55 × 55	11 × 11	4	valid	ReLU
In	Input	3 (RGB)	227 × 227	–	–	–	–

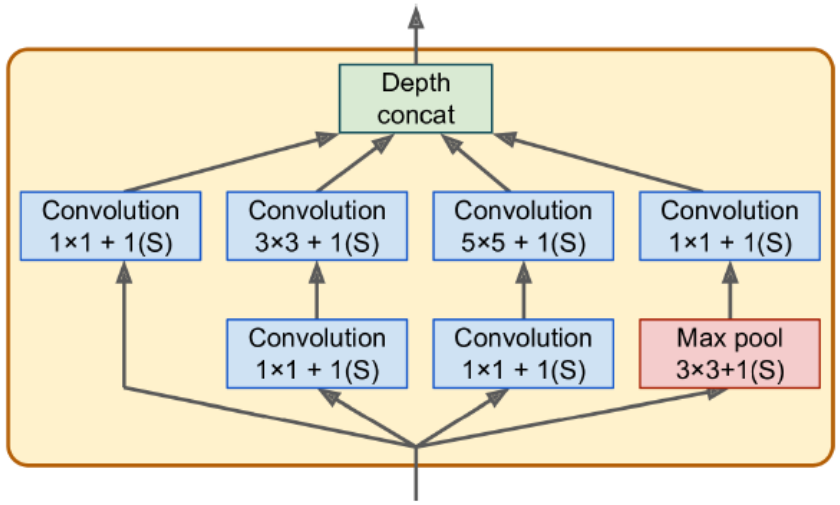
■ Data Augmentation



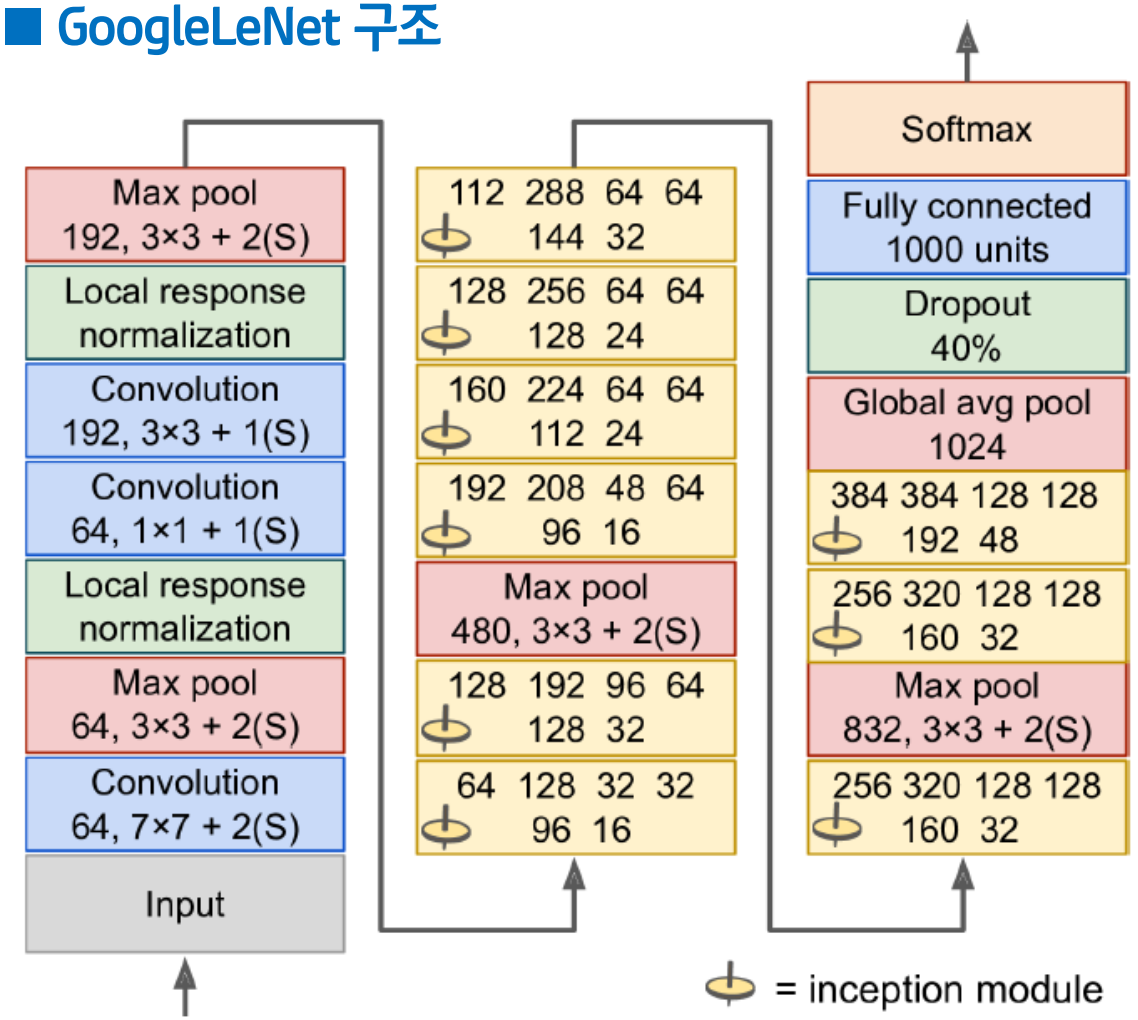
GoogleLeNet

Christian Szegedy 등이 개발하였으며, ILSVRC 2014 대회서 톱-5 에러율을 7% 이하로 낮추었습니다.
인셉션이라는 서브 네트워크를 가지고 있어서 이전 구조보다 훨씬 효과적을 파라미터를 사용합니다.

■ 인셉션 모듈



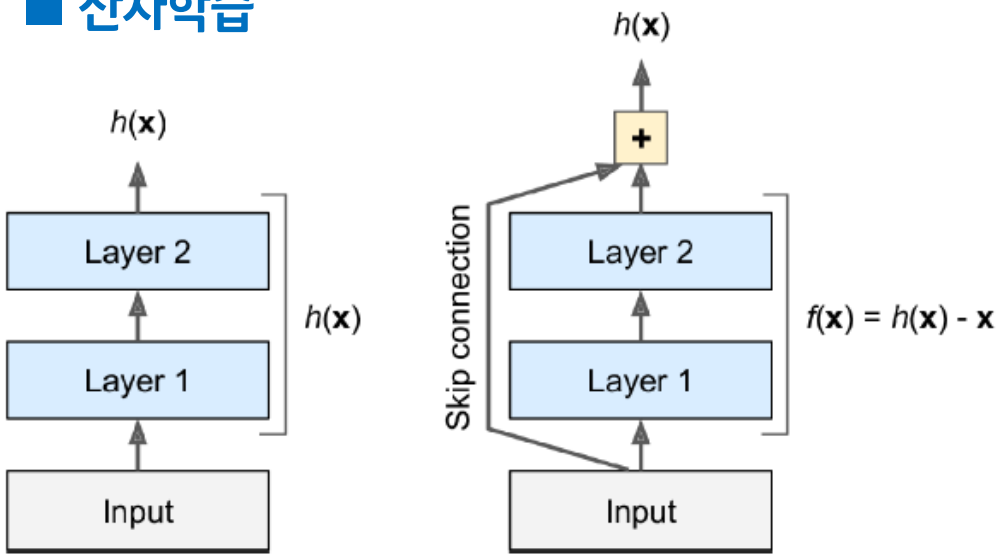
■ GoogLeNet 구조



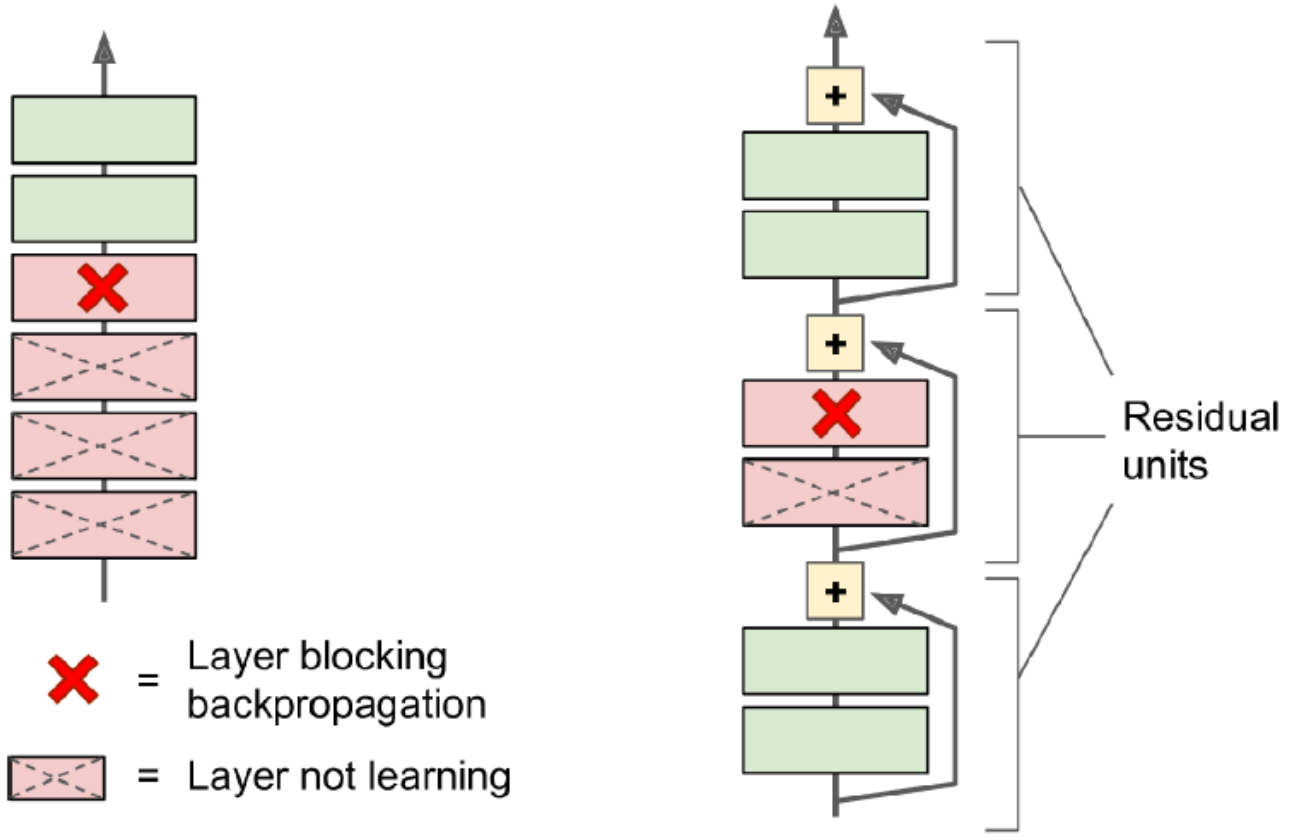
ResNet

Kaiming He 등은 ResNet을 사용해 ILSVRC 2015 대회에서 3.6%이라의 놀라운 톱-5 에러율을 기록했습니다. 152개 층으로 구성된 깊은 CNN을 사용하여 더 적은 파라미터를 사용해 더 깊은 네트워크로 모델을 구성하는 트렌드를 만들었습니다.

■ 잔차학습

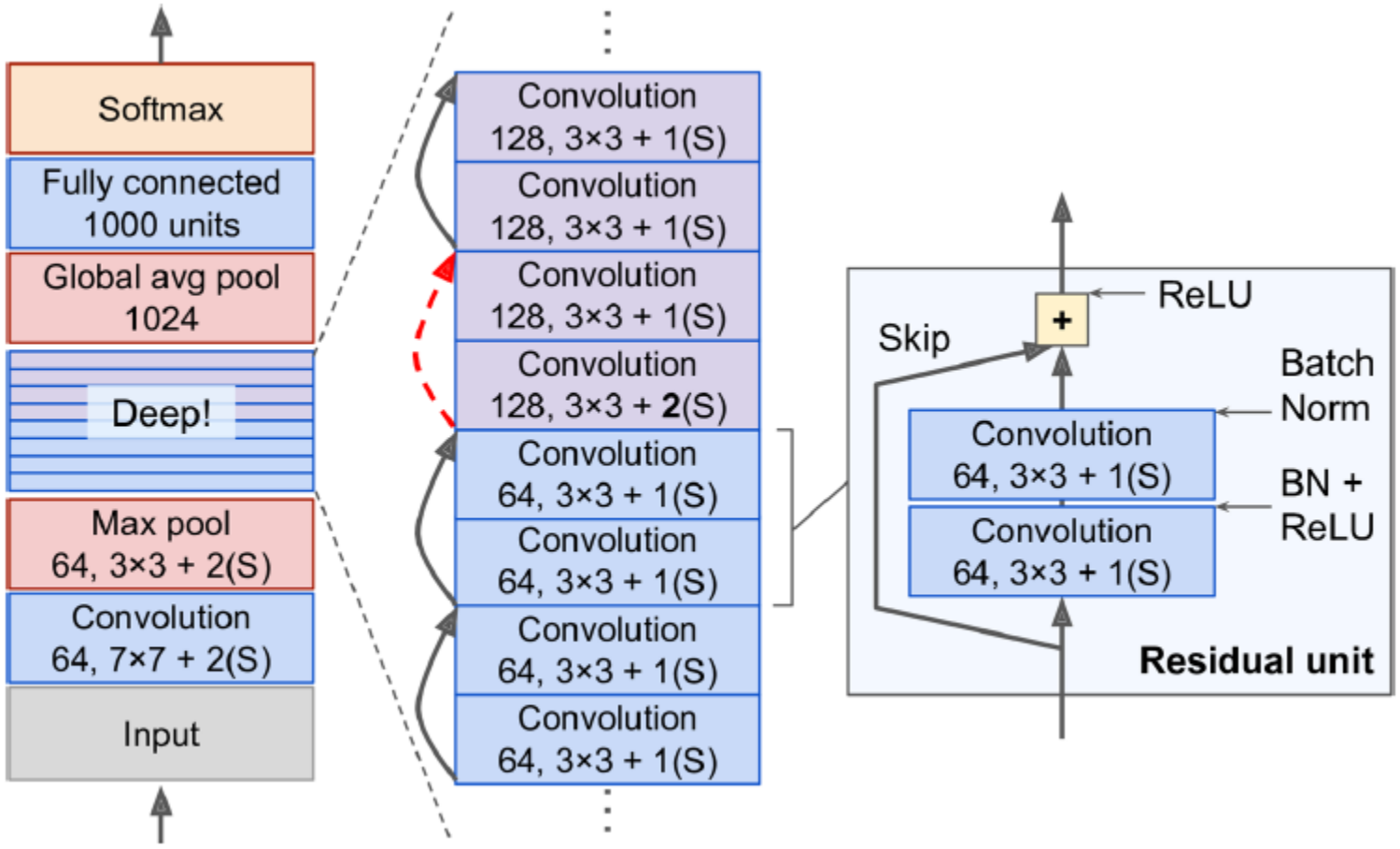


■ 일반적인 심층 신경망과 심층 잔차 네트워크



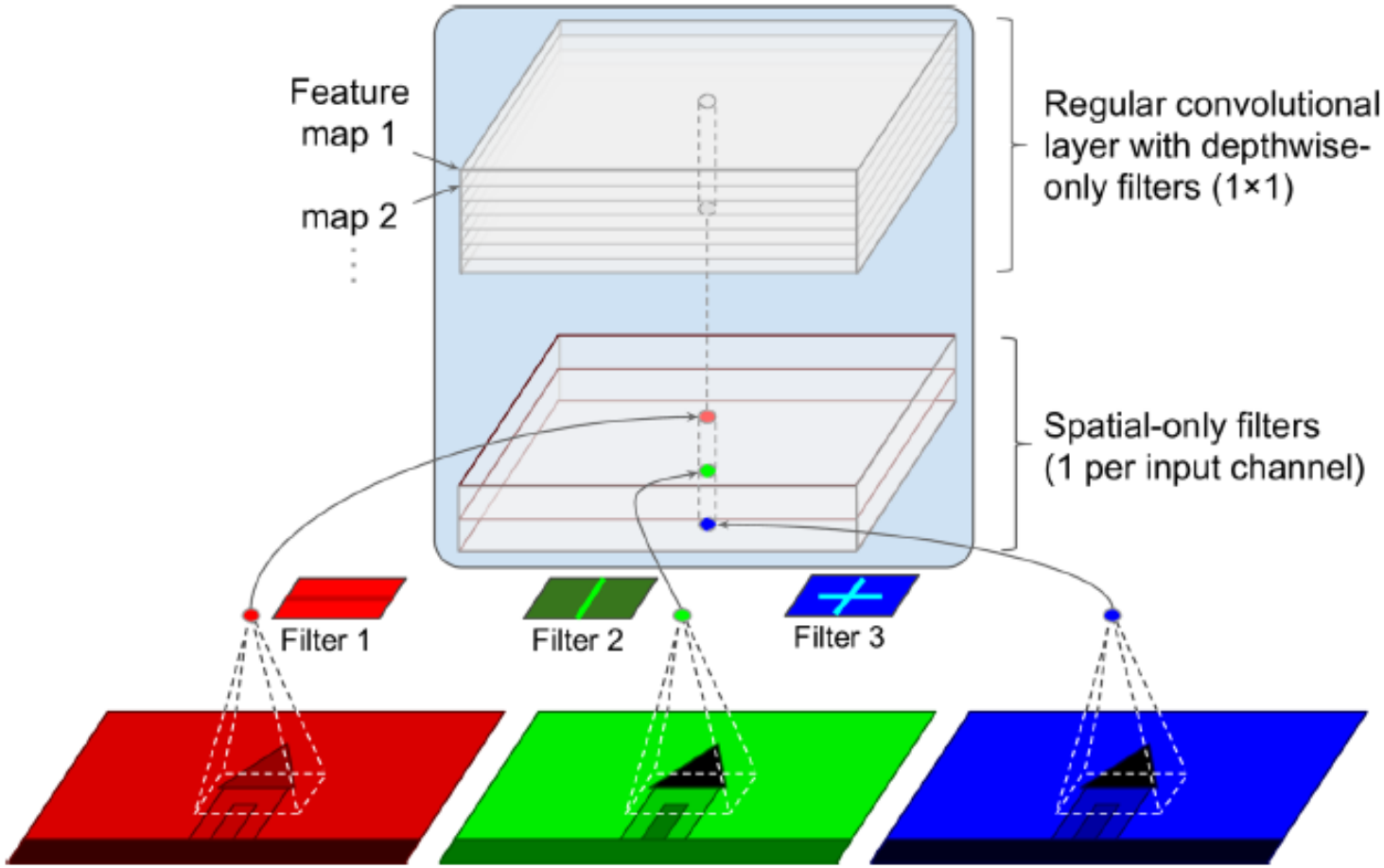
ResNet

■ ResNet 구조



Xception

■ 깊이별 분리 합성곱 층



ResNet-34 CNN 구현

```
class ResidualUnit(keras.layers.Layer):
    def __init__(self, filters, strides=1, activation="relu", **kwargs):
        super().__init__(**kwargs)
        self.activation = keras.activations.get(activation)
        self.main_layers = [
            keras.layers.Conv2D(filters, 3, strides=strides,
                                padding="same", use_bias=False),
            keras.layers.BatchNormalization(),
            self.activation,
            keras.layers.Conv2D(filters, 3, strides=1,
                                padding="same", use_bias=False),
            keras.layers.BatchNormalization()]
        self.skip_layers = []
        if strides > 1:
            self.skip_layers = [
                keras.layers.Conv2D(filters, 1, strides=strides,
                                    padding="same", use_bias=False),
                keras.layers.BatchNormalization()]

    def call(self, inputs):
        Z = inputs
        for layer in self.main_layers:
            Z = layer(Z)
        skip_Z = inputs
        for layer in self.skip_layers:
            skip_Z = layer(skip_Z)
        return self.activation(Z + skip_Z)
```

ResNet-34 CNN 구현

```
model = keras.models.Sequential()
model.add(keras.layers.Conv2D(64, 7, strides=2, input_shape=[224, 224, 3],
                              padding="same", use_bias=False))
model.add(keras.layers.BatchNormalization())
model.add(keras.layers.Activation("relu"))
model.add(keras.layers.MaxPool2D(pool_size=3, strides=2, padding="same"))
prev_filters = 64
for filters in [64] * 3 + [128] * 4 + [256] * 6 + [512] * 3:
    strides = 1 if filters == prev_filters else 2
    model.add(ResidualUnit(filters, strides=strides))
    prev_filters = filters
model.add(keras.layers.GlobalAvgPool2D())
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(10, activation="softmax"))
```

사전훈련 모델 사용

일반적을 GoogleLeNet이나 ResNet 같은 표준 모델을 직접 구현할 필요가 없습니다.
keras.applications 패키지에 준비되어 있는 사전 훈련된 모델을 코드로 불러올 수 있습니다.

```
model = keras.applications.resnet50.ResNet50(weights="imagenet")

images_resized = tf.image.resize(images, [224, 224])

inputs = keras.applications.resnet50.preprocess_input(images_resized * 255)

Y_proba = model.predict(inputs)

top_K = keras.applications.resnet50.decode_predictions(Y_proba, top=3)
for image_index in range(len(images)):
    print("Image #{}".format(image_index))
    for class_id, name, y_proba in top_K[image_index]:
        print("  {} - {:12s} {:.2f}%".format(class_id, name, y_proba * 100))
    print()
```

출처 : ooo

사전훈련된 모델을 사용한 전이학습

충분하지 않은 훈련 데이터로 이미지 분류기를 훈련하려면 사전훈련 된 모델의 하위층을 사용하는 것이 좋습니다.
사전 훈련된 Xception 모델을 사용해 꽃 이미지를 분류하는 모델을 훈련해보겠습니다.

```
import tensorflow_datasets as tfds

dataset, info = tfds.load("tf_flowers", as_supervised=True, with_info=True)
dataset_size = info.splits["train"].num_examples # 3670
class_names = info.features["label"].names # ["dandelion", "daisy", ...]
n_classes = info.features["label"].num_classes # 5

test_split, valid_split, train_split = tfds.Split.TRAIN.subsplit([10, 15, 75])

test_set = tfds.load("tf_flowers", split=test_split, as_supervised=True)
valid_set = tfds.load("tf_flowers", split=valid_split, as_supervised=True)
train_set = tfds.load("tf_flowers", split=train_split, as_supervised=True)
```

사전훈련된 모델을 사용한 전이학습

충분하지 않은 훈련 데이터로 이미지 분류기를 훈련하려면 사전훈련 된 모델의 하위층을 사용하는 것이 좋습니다.
사전 훈련된 Xception 모델을 사용해 꽃 이미지를 분류하는 모델을 훈련해보겠습니다.

```
def preprocess(image, label):
    resized_image = tf.image.resize(image, [224, 224])
    final_image = keras.applications.xception.preprocess_input(resized_image)
    return final_image, label

batch_size = 32
train_set = train_set.shuffle(1000)
train_set = train_set.map(preprocess).batch(batch_size).prefetch(1)
valid_set = valid_set.map(preprocess).batch(batch_size).prefetch(1)
test_set = test_set.map(preprocess).batch(batch_size).prefetch(1)

base_model = keras.applications.xception.Xception(weights="imagenet",
                                                  include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
output = keras.layers.Dense(n_classes, activation="softmax")(avg)
model = keras.Model(inputs=base_model.input, outputs=output)
```

사전훈련된 모델을 사용한 전이학습

충분하지 않은 훈련 데이터로 이미지 분류기를 훈련하려면 사전훈련 된 모델의 하위층을 사용하는 것이 좋습니다.
사전 훈련된 Xception 모델을 사용해 꽃 이미지를 분류하는 모델을 훈련해보겠습니다.

```
for layer in base_model.layers:
    layer.trainable = False


optimizer = keras.optimizers.SGD(lr=0.2, momentum=0.9, decay=0.01)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set, epochs=5, validation_data=valid_set)

for layer in base_model.layers:
    layer.trainable = True

optimizer = keras.optimizers.SGD(lr=0.01, momentum=0.9, decay=0.001)
model.compile(...)
history = model.fit(...)
```

분류와 위치 추정(Classification and Localization)

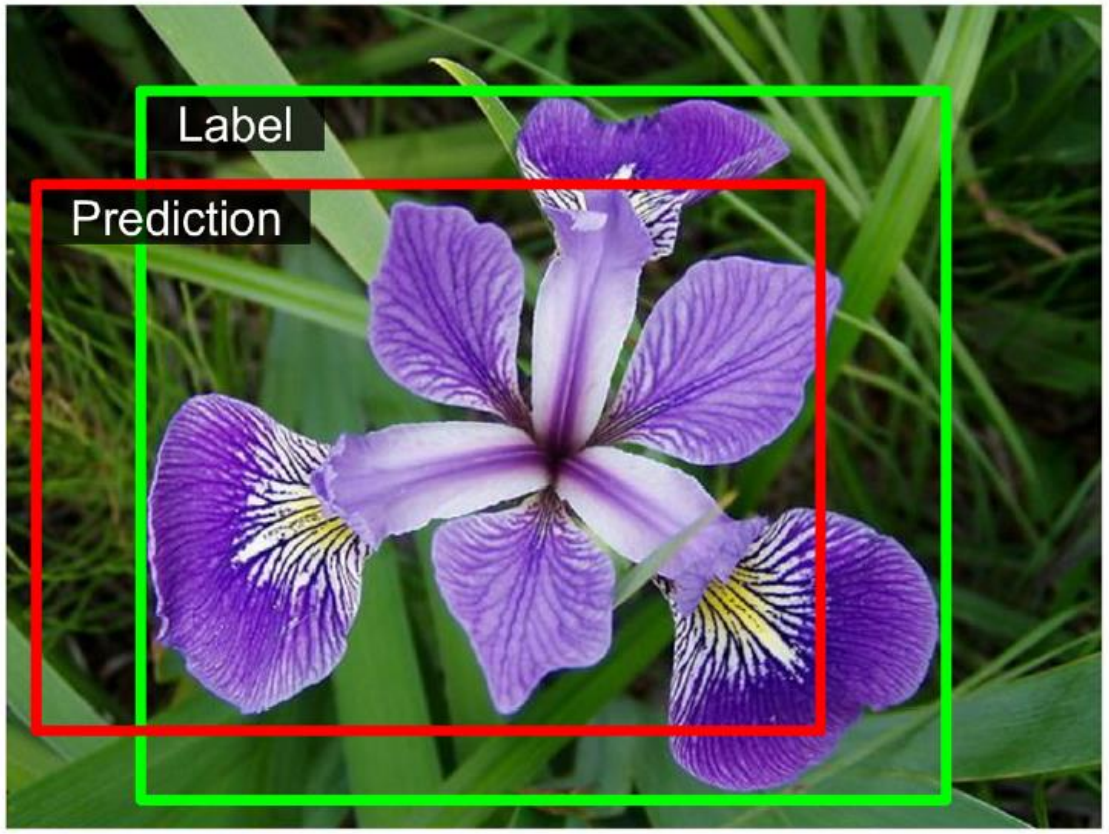
사진에서 물체의 위치를 추정하는 것은 회귀 작업으로 나타낼 수 있습니다. 물체 주위의 바운딩 박스를 예측하는 일반적인 방법은 물체 중심의 수평, 수직 좌표와 높이, 너비를 예측하는 것입니다.

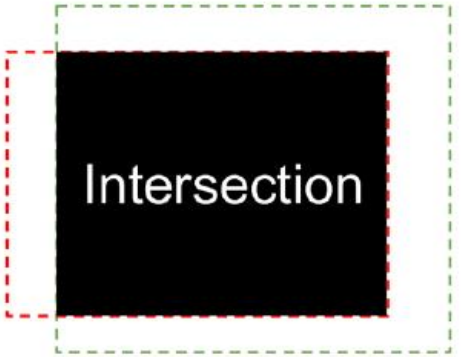
```
base_model = keras.applications.xception.Xception(weights="imagenet",
                                                  include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
class_output = keras.layers.Dense(n_classes, activation="softmax")(avg)
loc_output = keras.layers.Dense(4)(avg)
model = keras.Model(inputs=base_model.input,
                    outputs=[class_output, loc_output])
model.compile(loss=["sparse_categorical_crossentropy", "mse"],
              loss_weights=[0.8, 0.2], # depends on what you care most about
              optimizer=optimizer, metrics=["accuracy"])
```

분류와 위치 추정(Classification and Localization)

모델이 바운딩 박스를 얼마나 잘 예측하는지 평가하는 데 널리 사용되는 지표는 IoU(Intersection over Union) 입니다.

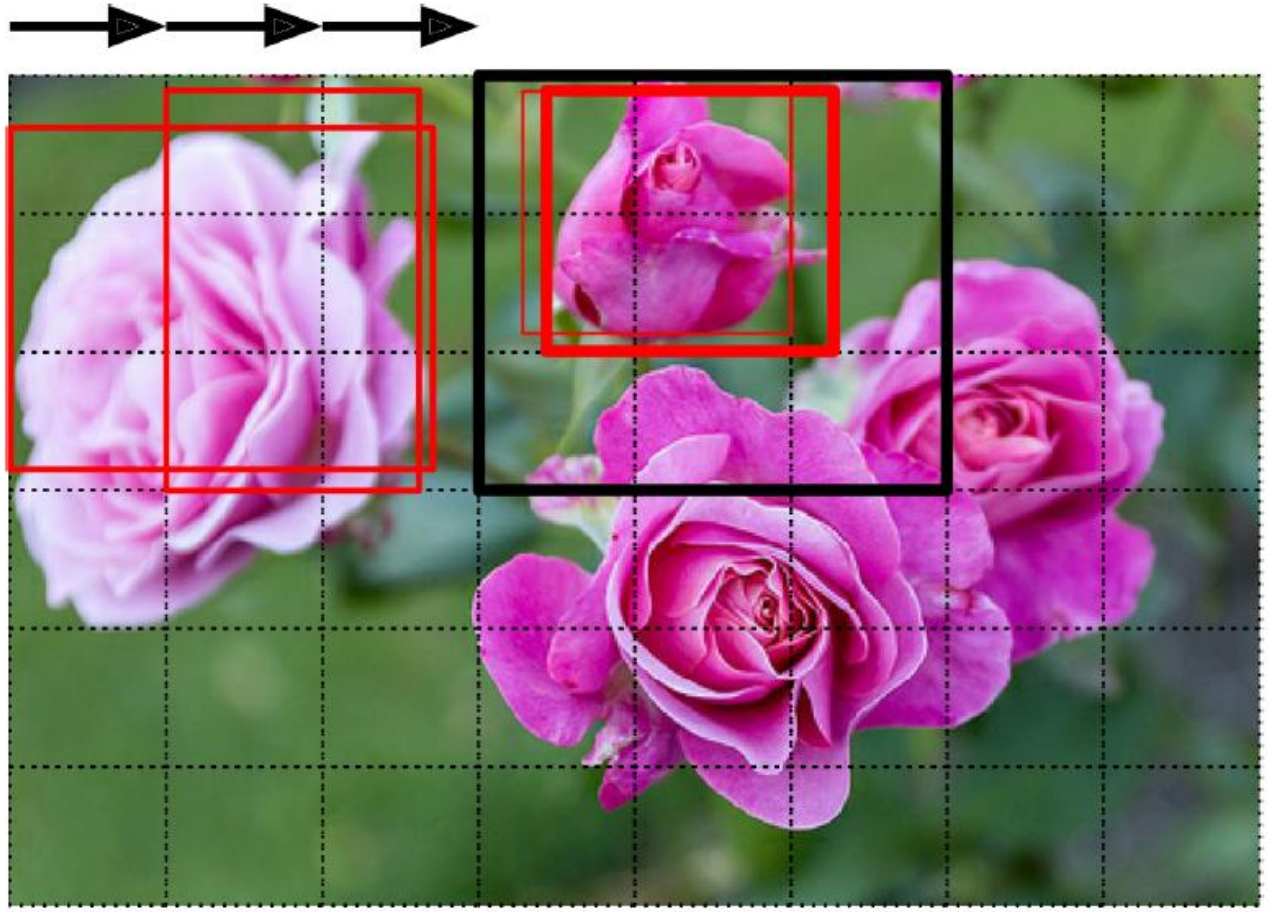
■ 바운딩 박스를 위한 IoU 지표





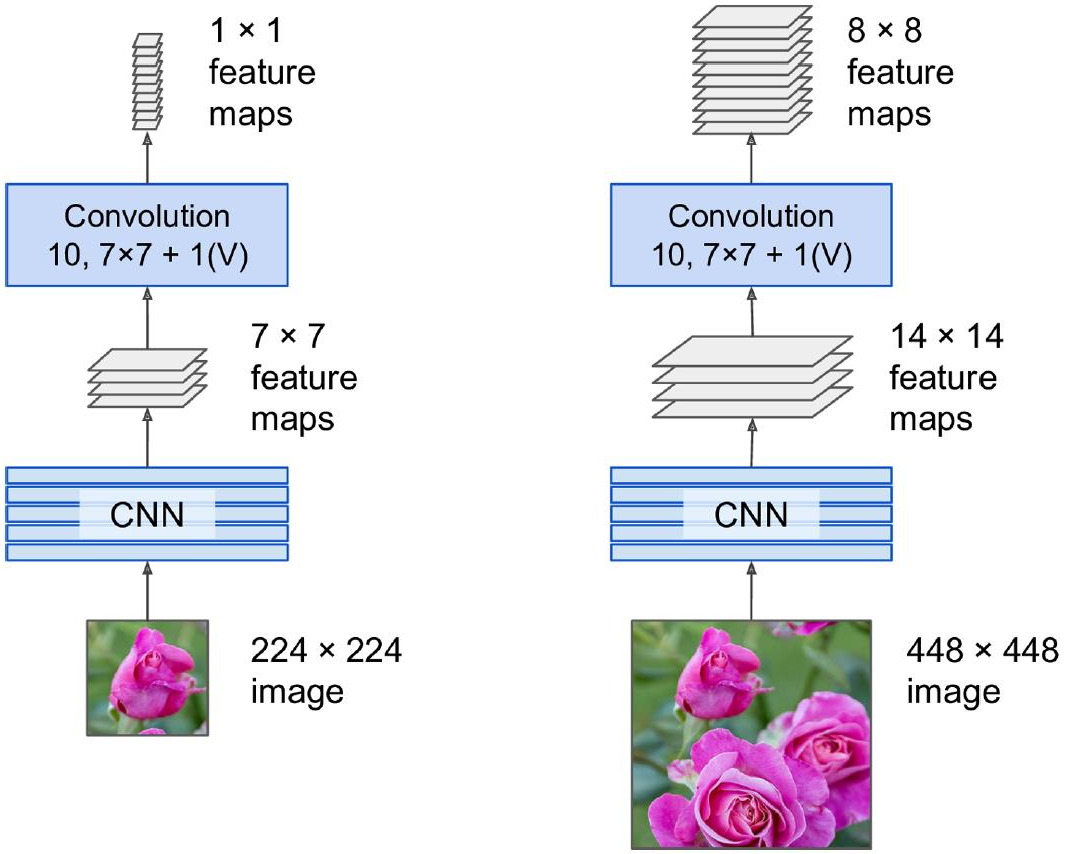
객체 탐지(Object Detection)

하나의 이미지에서 여러 물체를 분류하고 위치를 추정하는 작업을 객체 탐지라고 합니다.

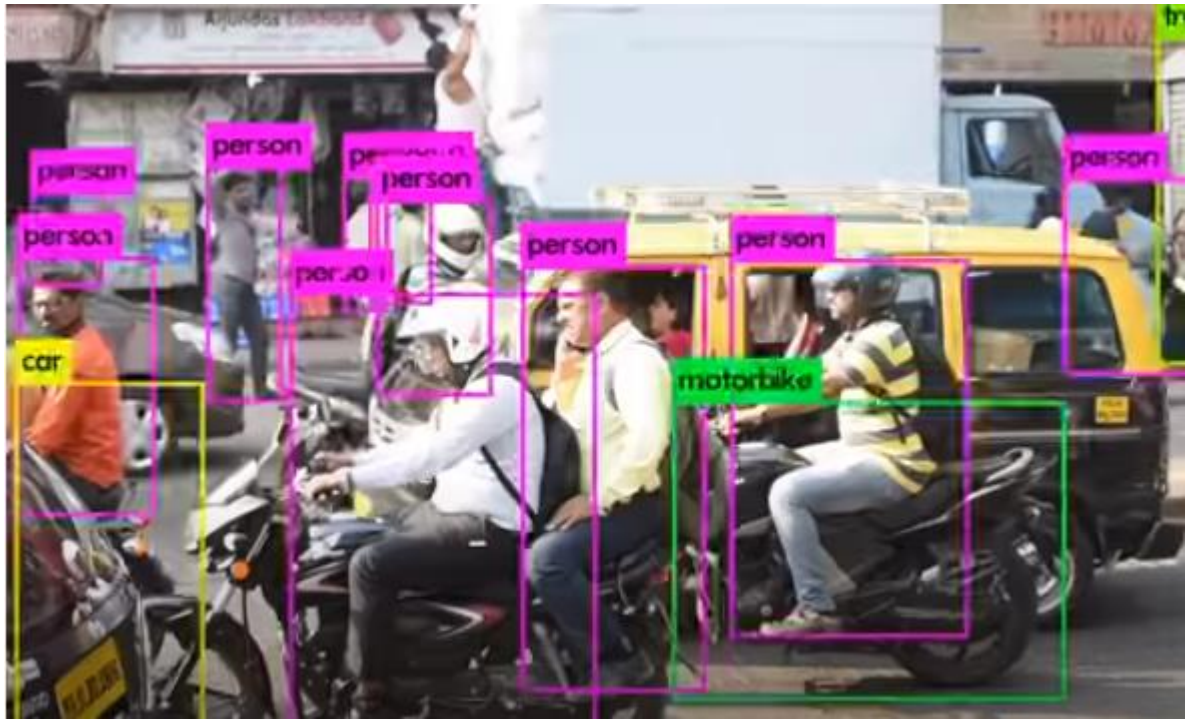


객체 탐지(Object Detection)

■ 완전 합성곱 신경망



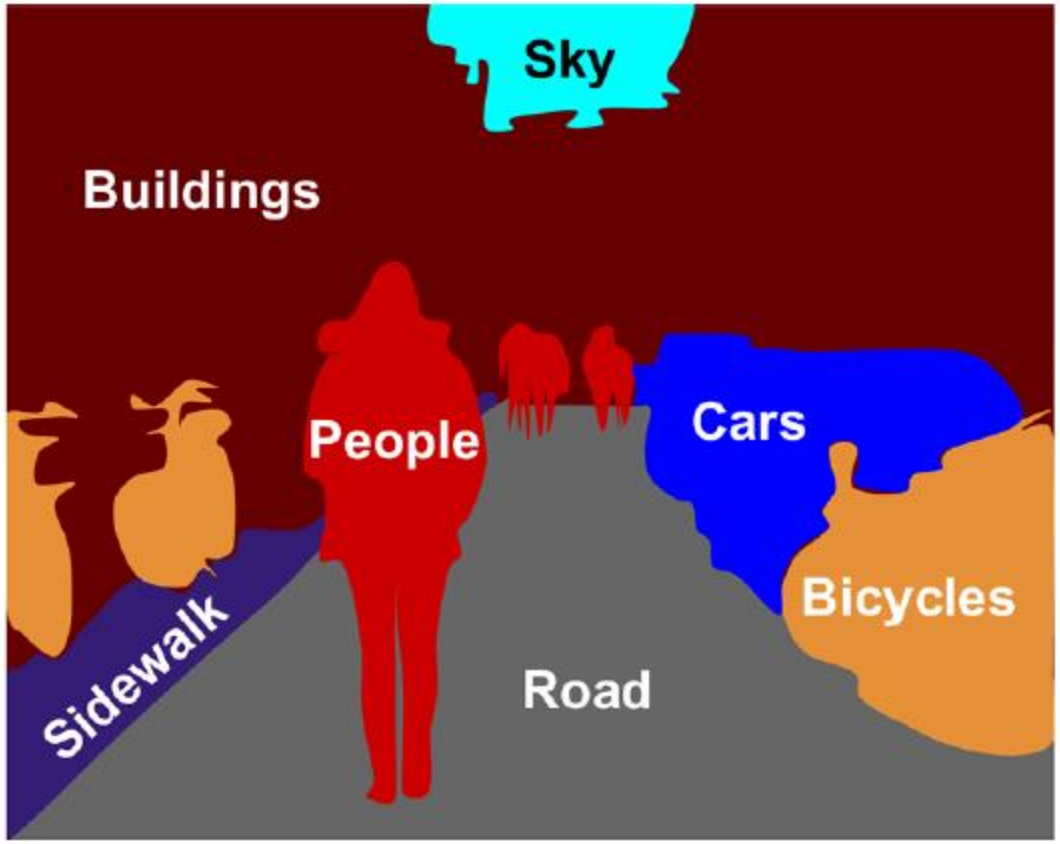
■ YOLO(You Only Look Once)



출처 : https://www.youtube.com/watch?v=MPU2HistivI

시맨틱 분할(Semantic Segmentation)

시맨틱 분할에서 각 픽셀은 픽셀이 속한 객체의 클래스로 구분됩니다. 클래스가 같은 물체는 구별되지 않습니다.



Hands-on

Thank you

박경규, 010-7240-1128
kgpark88@gmail.com