

합성곱 신경망

주요 내용

- 합성곱 신경망
- 데이터 증식
- 모델 재활용: 전이학습

합성곱 신경망

- 2011년부터 2015년: 컴퓨터 비전 분야에서 딥러닝 기법이 획기적으로 발전하게 된 계기 마련함
- 사진 검색, 자율주행, 로봇공학, 의학 진단 프로그램, 얼굴 인식 등 컴퓨터 비전 분야에서 일상적으로 활용됨
- 컴퓨터 비전 분야 딥러닝 모델: CNN으로 불리는 **합성곱 신경망**이 대세

MNIST 데이터셋 분류 CNN 모델

```
# 입력층  
inputs = keras.Input(shape=(28, 28, 1))  
  
# 은닉층  
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)  
x = layers.MaxPooling2D(pool_size=2)(x)  
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)  
x = layers.MaxPooling2D(pool_size=2)(x)  
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)  
  
# 출력층으로 넘기기 전에 1차원 텐서로 변환  
x = layers.Flatten()(x)  
  
# 출력층  
outputs = layers.Dense(10, activation="softmax")(x)  
  
# 모델  
model = keras.Model(inputs=inputs, outputs=outputs)
```

```
>>> model.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 28, 28, 1)	0
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_4 (Conv2D)	(None, 11, 11, 64)	18,496
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_5 (Conv2D)	(None, 3, 3, 128)	73,856
flatten_1 (Flatten)	(None, 1152)	0
dense_1 (Dense)	(None, 10)	11,530

Total params: 104,202 (407.04 KB)

Trainable params: 104,202 (407.04 KB)

Non-trainable params: 0 (0.00 B)

MNIST 이미지 분류 훈련

훈련셋 준비

```
from tensorflow.keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
train_images = train_images.reshape((60000, 28, 28, 1))
train_images = train_images.astype("float32") / 255
test_images = test_images.reshape((10000, 28, 28, 1))
test_images = test_images.astype("float32") / 255
```

모델 컴파일과 훈련

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy",
              metrics=["accuracy"])

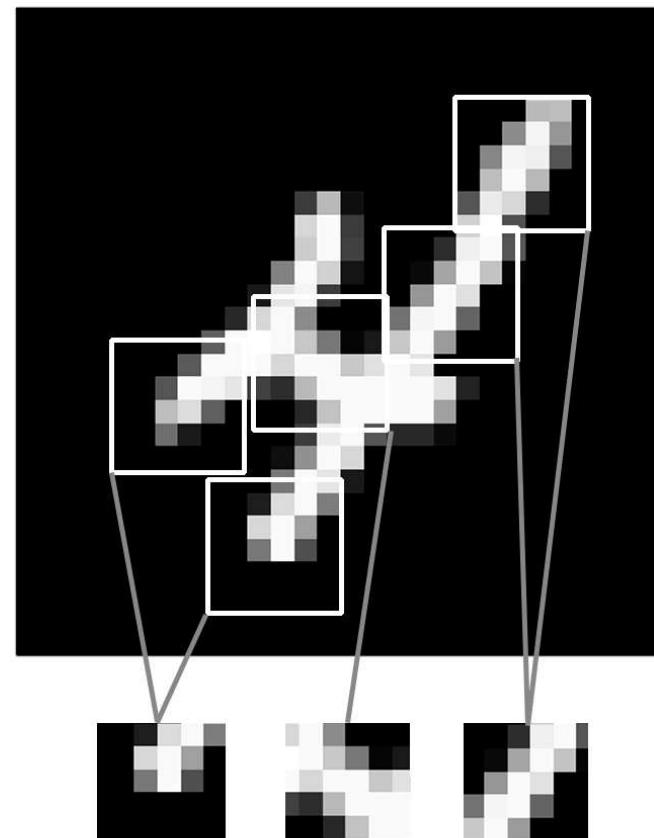
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

합성곱 연산

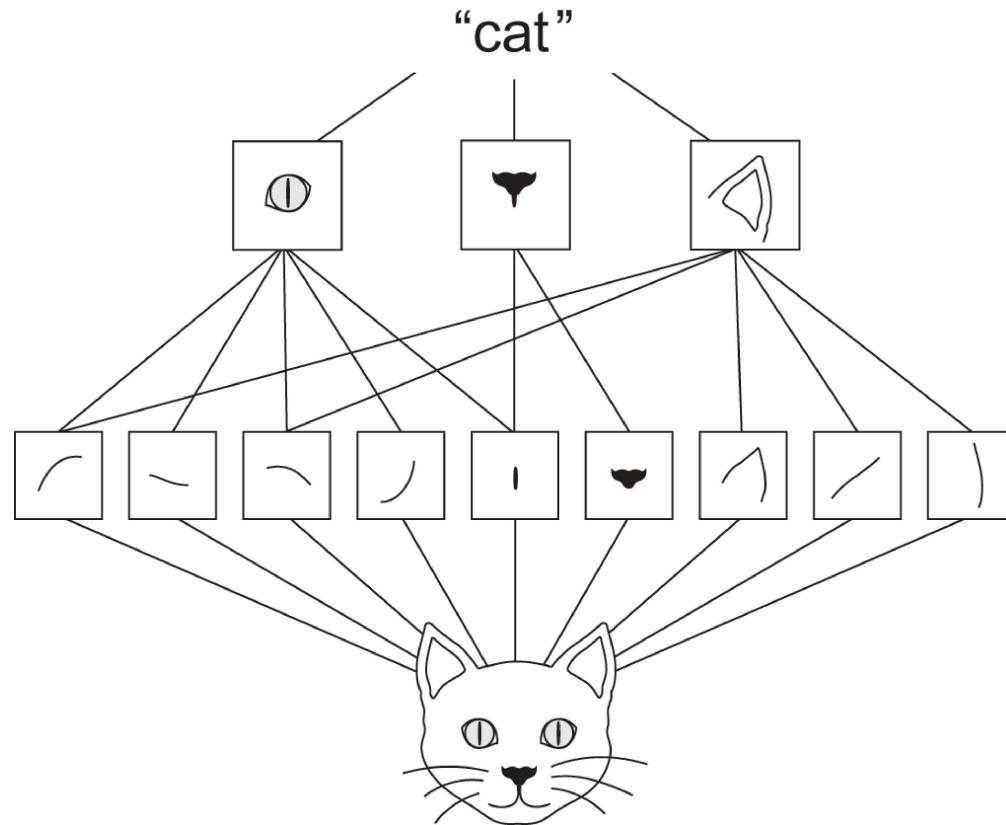
- `Dense` 층: 입력값의 전체 특성을 대상으로 한 번의 아핀 변환 적용
- `Conv2D` 층: `kernel_size`로 지정된 크기의 공간에 대해 여러 개의 아핀 변환 적용.

합성곱 층의 특징

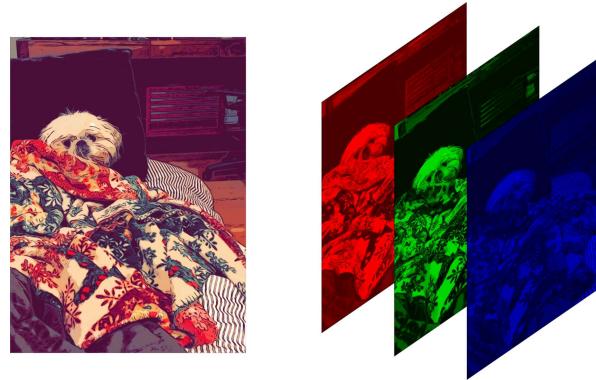
첫째, 위치와 무관한 패턴을 찾아낸다. 즉, 서로 다른 위치에 있는 동일한 패턴은 동일한 값으로 계산된다.



둘째, Con2D 층 여러 개를 연속적으로 통과시키면 보다 복잡한 패턴을 파악할 수 있다.



컬러 이미지와 채널

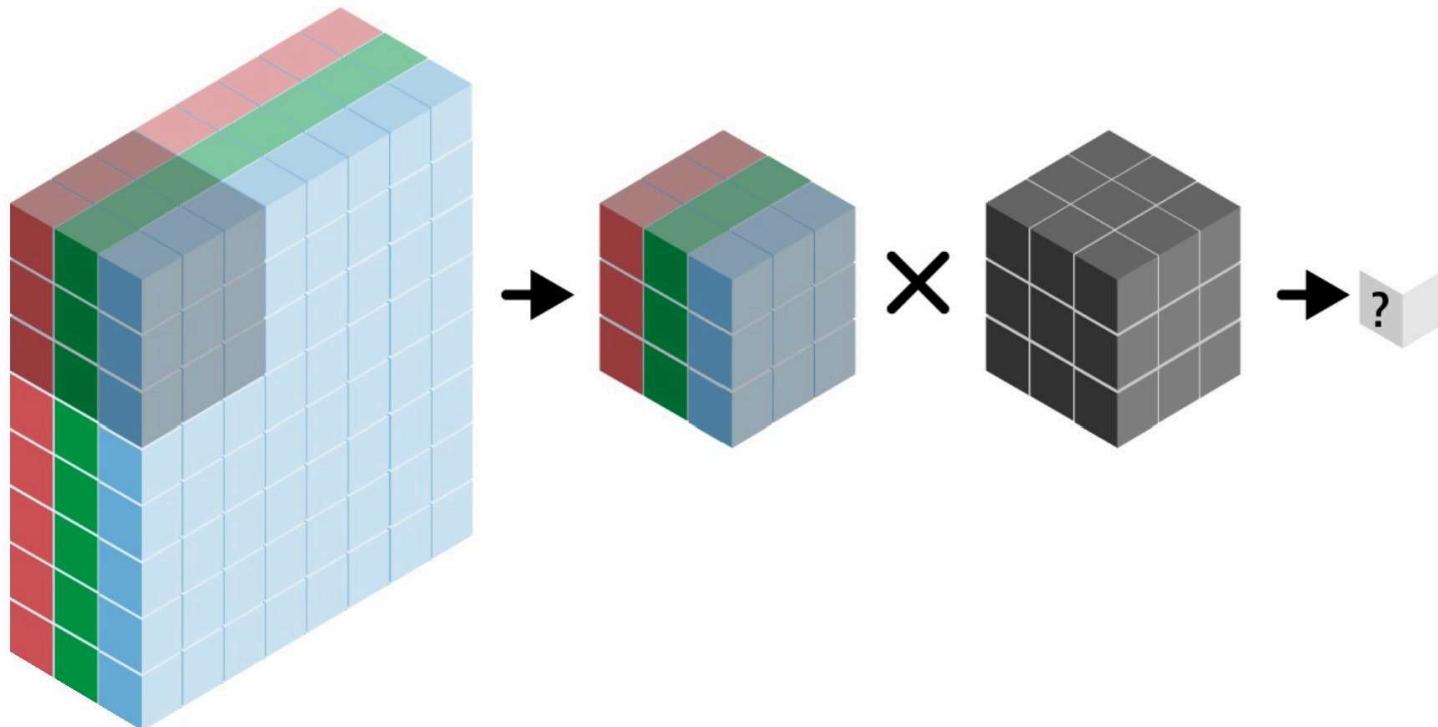


너비		
높이		
0	1	2
0	.392 .482 .576	
1	.478 .63 .169	.263 .376 .451
2	.580 .79 .263	.44 .306 .376 .561
0		.373 .60 .376 .443 .569 .674
1		
2		

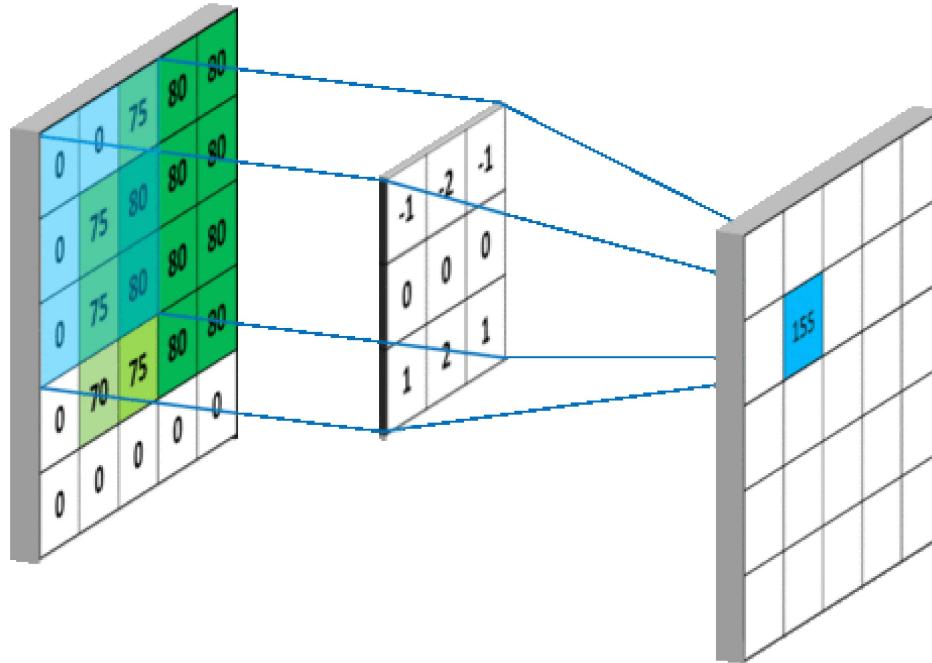
특성맵(채널), 필터, 출력맵

- **채널**_{channel}:
 - 특성맵 feature map이라고도 불림.
 - (높이, 너비) 모양의 2D 텐서.
 - 예제: MNIST 데이터셋에 포함된 (28, 28) 모양의 흑백 이미지 샘플이 (28, 28, 1) 모양의 채널(특성맵) 한 개로 구성된 3차원 텐서로 변형되어 입력 데이터로 사용됨.
 - 예제: 컬러사진의 경우 세 개의 채널(특성맵)로 구성됨.
 - 이미지에 포함된 채널(특성맵)의 수를 **깊이**라 부름.
- **필터**_{filter}: kernel_size를 이용한 3D 텐서.
 - 예제: kernel_size=3인 경우 필터는 (3, 3, 입력샘플의깊이) 모양의 3D 텐서.
 - 필터 수: filters 인자에 의해 결정됨.
- **출력맵**_{response map}: 입력 샘플을 대상으로 하나의 필터를 적용해서 생성된 하나의 특성맵(채널). 필터 수만큼의 출력맵 생성.

필터 적용



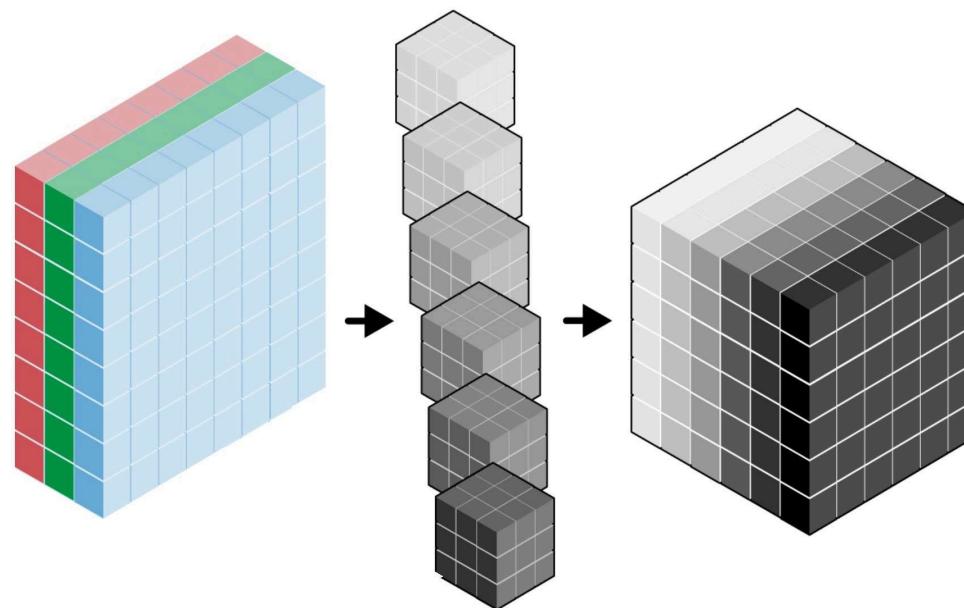
출력맵



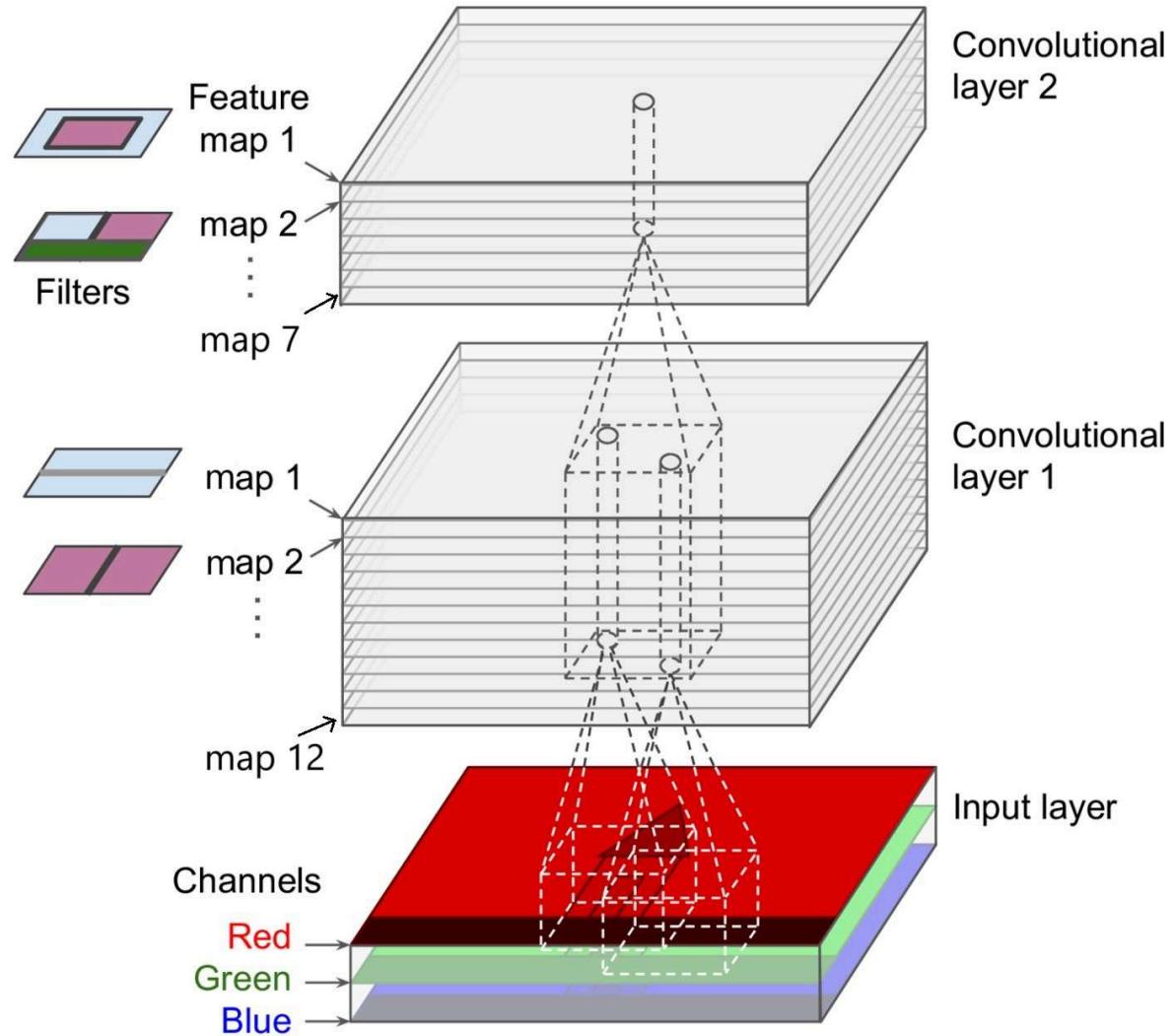
```
0 * -1 + 0 * -2 + 75 * -1 +
0 * 0 + 75 * 0 + 80 * 0 +
0 * 1 + 75 * 2 + 80 * 1 +
0
= 155
```

필터와 출력맵

- 입력 텐서: (10, 8, 3) 모양의 텐서
- 필터: (3, 3, 3) 모양의 텐서
 - 커널 크기(kernel_size): 3
 - 입력 텐서의 깊이: 3
- 출력 텐서: (8, 6, 6) 모양의 텐서
 - 필터 수가 6이기에 출력 텐서의 깊이가 6이 됨.



합성곱 층 연속 적용



합성곱 층 학습 파라미터

- Dense 층: 아핀 변환에 필요한 가중치와 편향을 학습시킴.
- Conv2D 층: 필터로 사용되는 텐서들을 학습시킴.
 - 학습되어야 하는 파라미터의 수는 커널 크기 `kernel_size`와 필터 수 `filters`에 의해 결정됨.

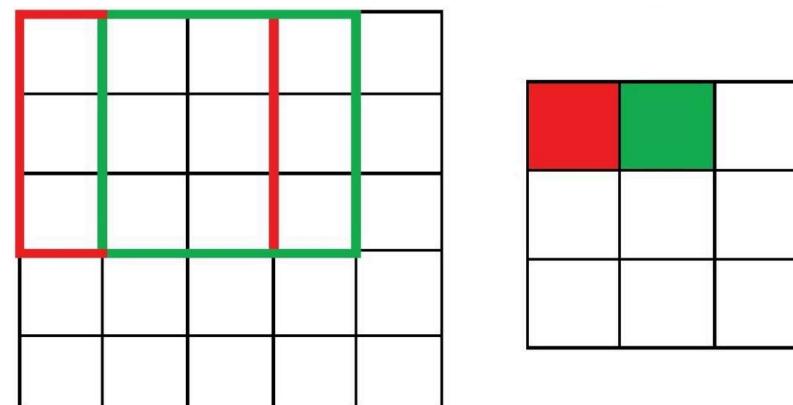
패딩과 보폭

- 필터를 적용하여 생성된 출력 특성맵의 모양이 입력 특성맵(채널)의 모양과 다를 수 있음.
- 출력 특성맵의 높이와 너비: 패딩의 사용 여부와 보폭의 크기에 의에 결정됨.

```
keras.layers.Conv2D(  
    filters,  
    kernel_size,  
    strides=(1, 1), # 보폭  
    padding="valid", # 패딩  
    ...  
)
```

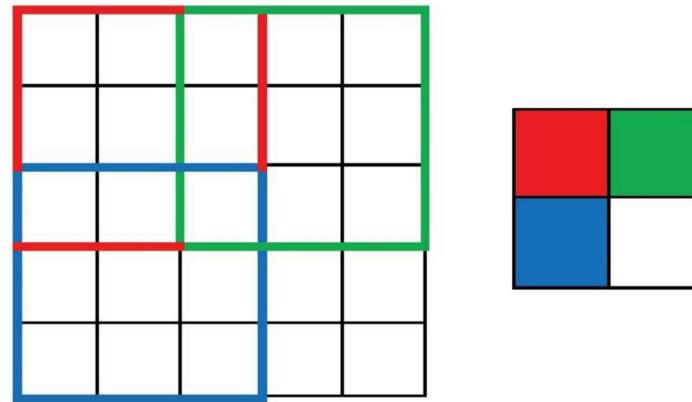
경우 1: 패딩 없음, 보폭은 1

- `strides` 와 `padding` 키워드 인자의 기본값 사용
- 필터가 1칸씩 슬라이딩
- 출력 특성맵의 깊이와 너비: 3×3
- 출력 특성맵의 깊이와 너비가 줄어듦.



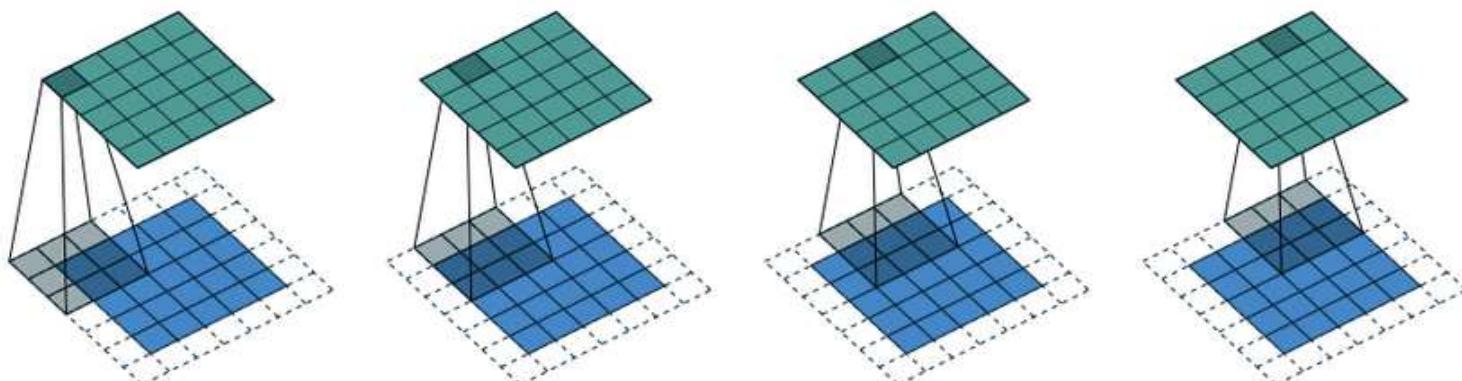
경우 2: 패딩 없음, 보폭은 2

- `strides=2` 또는 `strides=(2, 2)`
- 필터 2칸씩 건너 뛰며 슬라이딩
- 출력 특성맵의 깊이와 너비: 2×2
- 출력 특성맵의 깊이와 너비가 보폭의 반비례해서 줄어듦.



경우 3: 패딩 있음, 보폭은 1

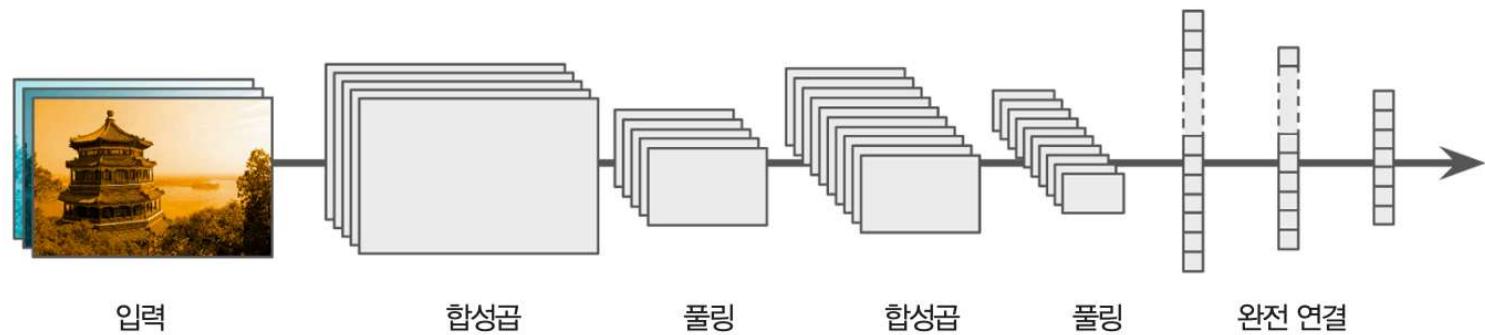
- padding="same"
- 입력 텐서의 테두리에 0으로 채워진 패딩 추가.
- 출력 특성맵의 깊이와 너비가 동일하게 유지됨.



경우 4: 패딩 있음, 보폭은 2 이상

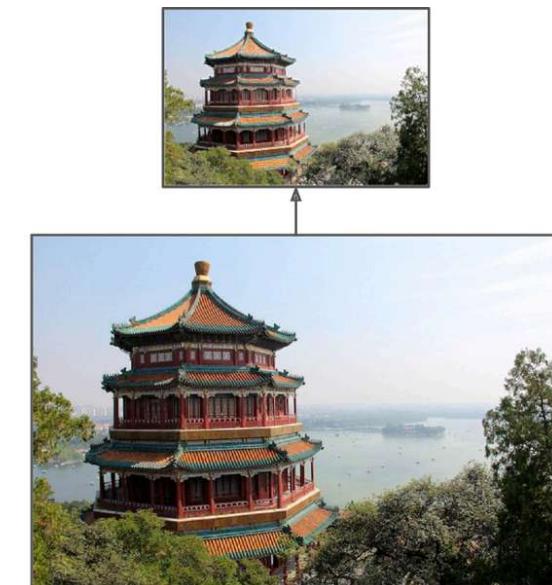
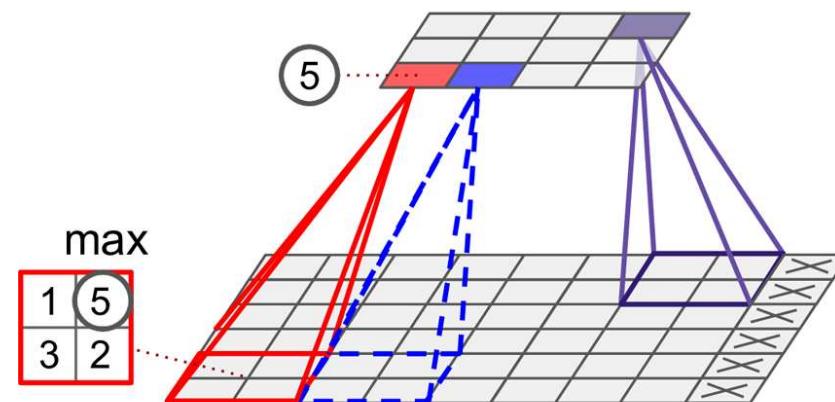
- 굳이 사용할 필요 없음.
- 이유: 보폭이 1보다 크기에 출력 특성맵의 깊이와 너비가 어차피 보폭에 반비례해서 줄어들기 때문임.

CNN 모델의 전형



맥스 풀링 기능

```
keras.layers.MaxPooling2D(  
    ...  
    pool_size=(2, 2), # 풀링 크기  
    ...  
    strides=None,  
    ...  
    padding="valid",  
    ...  
)
```



맥스 풀링 사용 이유

첫째, 모델이 학습 파라미터(가중치와 편향) 수를 줄인다.

- 맥스 풀링 사용하는 경우: 104,202개
- 그렇지 않은 경우: 712,202개

```
inputs = keras.Input(shape=(28, 28, 1))
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(inputs)
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)
x = layers.Flatten()(x)
outputs = layers.Dense(10, activation="softmax")(x)
model_no_max_pool = keras.Model(inputs=inputs, outputs=outputs)
```

```
>>> model.summary()
```

Model: "functional_2"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 28, 28, 1)	0
conv2d_6 (Conv2D)	(None, 26, 26, 32)	320
conv2d_7 (Conv2D)	(None, 24, 24, 64)	18,496
conv2d_8 (Conv2D)	(None, 22, 22, 128)	73,856
flatten_2 (Flatten)	(None, 61952)	0
dense_2 (Dense)	(None, 10)	619,530

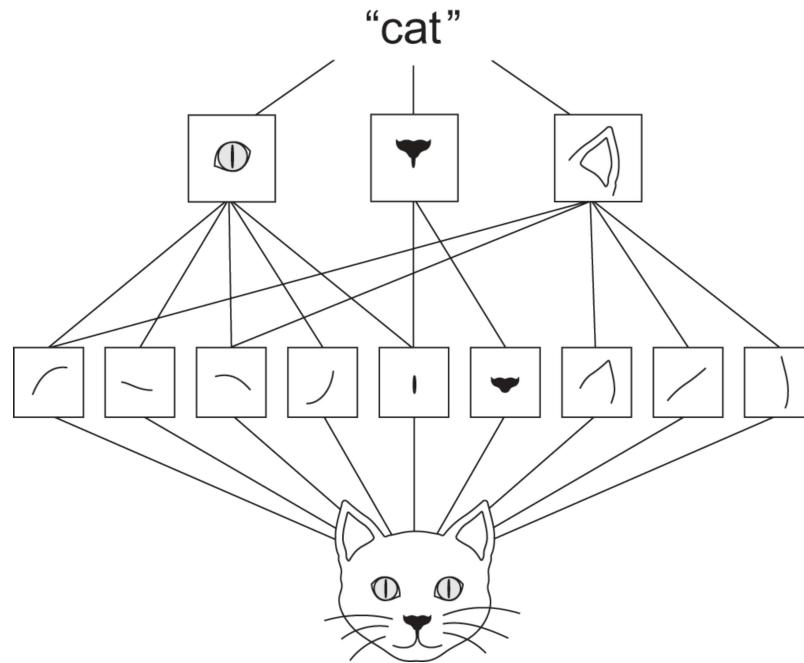
Total params: 712,202 (2.72 MB)

Trainable params: 712,202 (2.72 MB)

Non-trainable params: 0 (0.00 B)

둘째, 상위 합성곱 층으로 이동할 수록 입력 데이터의 보다 넓은 영역에 대한 함축된 정보를 얻을 수 있다.

- 또한 이 과정을 통해 이미지에서의 위치와 무관한 특정 패턴을 모델이 보다 용이하게 학습할 수 있음.



합성곱 신경망 실전 활용 예제

작은 데이터셋과 딥러닝 모델

- 이미지 분류 모델 훈련: 많은 양의 데이터를 모으기 힘들어서 데이터셋의 크기가 작은 경우가 많이 발생.
- 데이터셋의 크기가 몇 백개에서 몇 만개 정도가 일반적임.
- 실전 상황 재현: 5천 개의 이미지로 이루어진 작은 데이터셋을 효율적으로 활용하는 기법 소개
- 개와 고양이 사진을 대상으로 하는 이진 분류 합성곱 신경망 모델 훈련

데이터 다운로드

- 캐글(Kaggle) 계정 필요
- 캐글의 "Account" 페이지의 계정 설정 창: "API" 항목에서 "Create New Token"을 눌러 토큰 생성 후 다운로드
- [캐글: Dogs vs. Cats](#): "I Understand and Accept" 버튼 클릭

- 총 25,000장의 강아지와 고양이 사진
- 570MB 정도 용량
- 강아지 사진 고양이 사진: 각각 12,500 장씩 포함
- 사진들의 크기가 다음과 같이 일정하지 않음



훈련셋, 검증셋, 테스트셋 준비

5,000 장의 사진만 사용해서 합성곱 신경망 모델 훈련

- 훈련셋: 강아지와 고양이 각각 1,000 장
- 검증셋: 강아지와 고양이 각각 500 장
- 테스트셋: 강아지와 고양이 각각 1,000 장

- 5,000 장의 사진이 아래 구조의 하위 디렉토리에 저장되어 있다고 가정
- 5천장 선택과 저장 과정은 ([구글 코랩](#)) 합성곱 신경망 참고

```
cats_vs_dogs_small/
...train/
.....cat/
.....dog/
...validation/
.....cat/
.....dog/
...test/
.....cat/
.....dog/
```

모델 지정

```
# 입력층  
inputs = keras.Input(shape=(180, 180, 3)) # 컬러사진  
  
# 은닉층  
x = layers.Rescaling(1./255)(inputs)  
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)  
x = layers.MaxPooling2D(pool_size=2)(x)  
x = layers.Conv2D(filters=64, kernel_size=3, activation="relu")(x)  
x = layers.MaxPooling2D(pool_size=2)(x)  
x = layers.Conv2D(filters=128, kernel_size=3, activation="relu")(x)  
x = layers.MaxPooling2D(pool_size=2)(x)  
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)  
x = layers.MaxPooling2D(pool_size=2)(x)  
x = layers.Conv2D(filters=256, kernel_size=3, activation="relu")(x)  
x = layers.Flatten()(x)  
  
# 출력층  
outputs = layers.Dense(1, activation="sigmoid")(x)  
model = keras.Model(inputs=inputs, outputs=outputs)
```

- 입력층: 입력 샘플의 모양을 `(180, 180, 3)`으로 지정. 사진의 크기가 제각각이기에 먼저 지정된 크기의 텐서로 변환을 해주는 전처리 과정이 필요함.
- `Rescaling(1./255)` 층: 0에서 255 사이의 값을 0에서 1 사이의 값으로 변환하는 용도로 사용
- `Conv2D` 층에서 지정되는 필터의 수는 32에서 256으로 점차 키워짐. 맥스풀링 층에 의해 사진의 크기가 줄어드는 대신 필터 수를 늘려 출력맵의 개수를 키워주면서 모델의 정보 저장 능력을 유지시키기 위함임.
- `MaxPooling2D` 층을 `Conv2D` 층과 함께 사용.
- `Flatten` 층에 최종적으로 `7 x 7` 크기의 않은 특성맵으로 구성된 입력값이 전달됨.
- 출력층: 이항 분류 모델이기에 한 개의 유닛과 시그모이드 활성화 함수 사용하는 `Dense` 층으로 지정.

모델 컴파일

```
model.compile(loss="binary_crossentropy",
               optimizer="rmsprop",
               metrics=["accuracy"])
```

데이터 불러오기와 전처리

사진들을 무작위로 섞어 (180, 180, 3) 모양의 이미지가 32개씩 묶인 배치들로 구성된 훈련셋, 검증셋, 테스트셋 지정

```
from tensorflow.keras.utils import image_dataset_from_directory

new_base_dir = pathlib.Path("cats_vs_dogs_small")

train_dataset = image_dataset_from_directory(
    new_base_dir / "train",
    image_size=(180, 180),
    batch_size=32)

validation_dataset = image_dataset_from_directory(
    new_base_dir / "validation",
    image_size=(180, 180),
    batch_size=32)

test_dataset = image_dataset_from_directory(
    new_base_dir / "test",
    image_size=(180, 180),
    batch_size=32)
```

예제: 훈련셋 배치 모양 확인

`train_dataset`에 포함된 각각의 배치는 `(32, 180, 180, 3)` 모양의 4차원 입력 텐서와 `(32,)` 모양의 1차원 타깃으로 구성된다.

```
>>> for data_batch, labels_batch in train_dataset:  
...     print("data batch shape:", data_batch.shape)  
...     print("labels batch shape:", labels_batch.shape)  
...     break  
  
data batch shape: (32, 180, 180, 3)  
labels batch shape: (32, )
```

모델 훈련

- 크기가 32인 배치 단위로 이미 묶여 있음
- `fit()` 메서드: 배치 크기(`batch_size`)는 지정 불필요
- `train_dataset` 과 `validation_dataset` 이 모두 입력 데이터셋과 타깃 데이터셋의 튜플로 구성되어 있음에 주의할 것.
- 콜백 활용

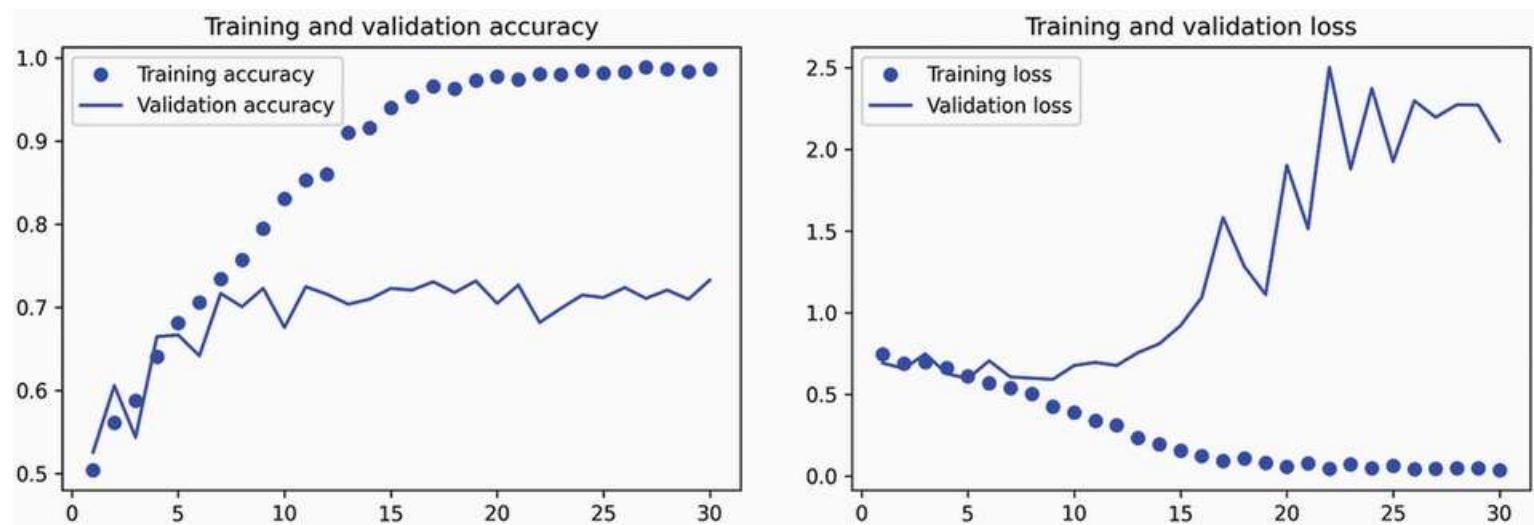
```
history = model.fit(  
    train_dataset,  
    epochs=30,  
    validation_data=validation_dataset,  
    callbacks=callbacks)
```

ModelCheckpoint 콜백 활용

```
callbacks = [  
    keras.callbacks.ModelCheckpoint(  
        filepath="convnet_from_scratch",  
        save_best_only=True,  
        monitor="val_loss")  
]
```

빠른 과대 적합 발생

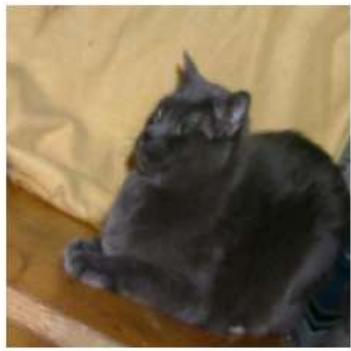
- 훈련셋의 크기가 2000 정도로 매우 작기에 과대 적합이 빠르게 발생함.



데이터 증식

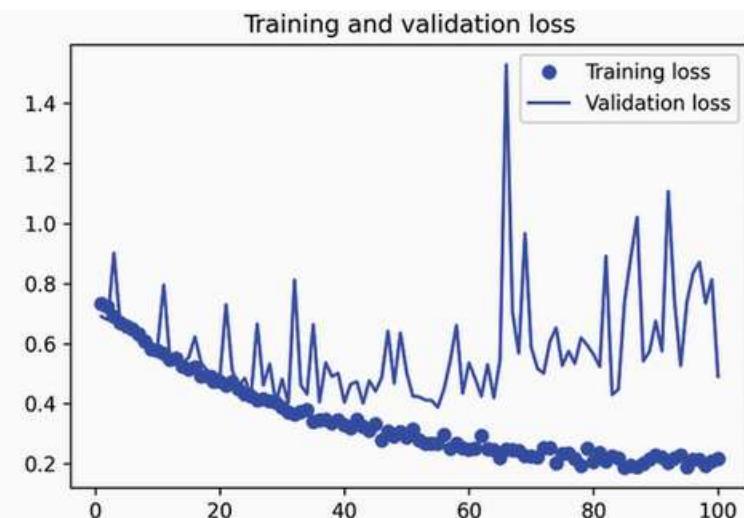
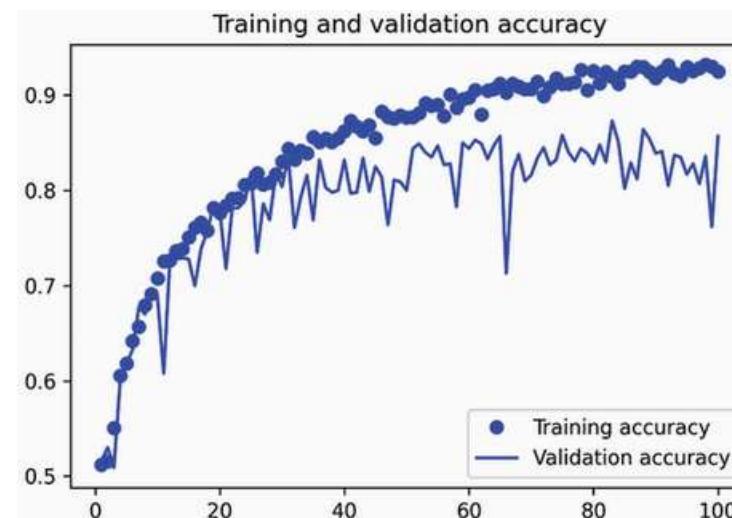
- `RandomFlip()`: 사진을 50%의 확률로 지정된 방향으로 반전.
- `RandomRotation()`: 사진을 지정된 범위 안에서 임의로 좌우로 회전
- `RandomZoom()`: 사진을 지정된 범위 안에서 임의로 확대 및 축소

```
data_augmentation = keras.Sequential(  
    [layers.RandomFlip("horizontal"),  
     layers.RandomRotation(0.1),  
     layers.RandomZoom(0.2)]  
)
```



```
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)
x = layers.Conv2D(filters=32, kernel_size=3, activation="relu")(x)
x = layers.MaxPooling2D(pool_size=2)(x)
...
x = layers.Flatten()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

과대 적합이 보다 늦게 발생



모델 재활용

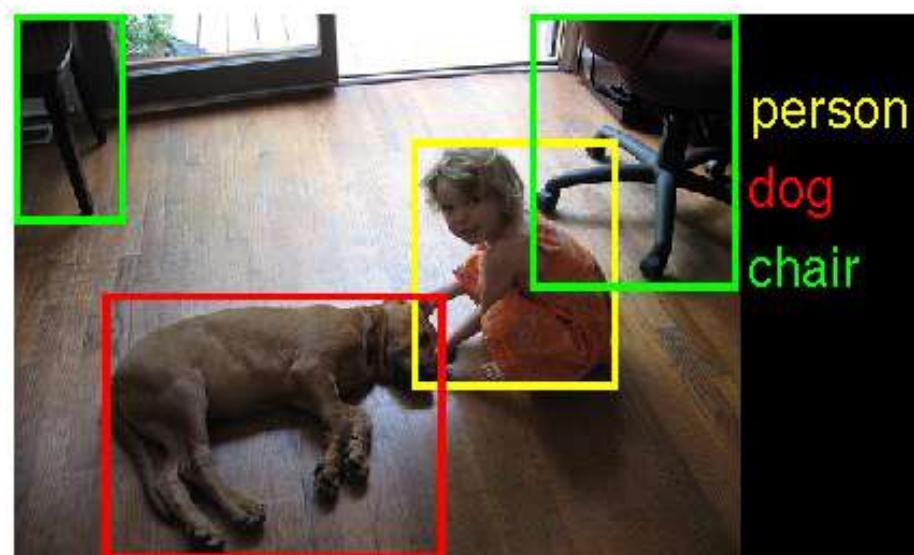
- 전이 학습 transfer learning
- 모델 미세조정 model fine tuning

VGG16 모델

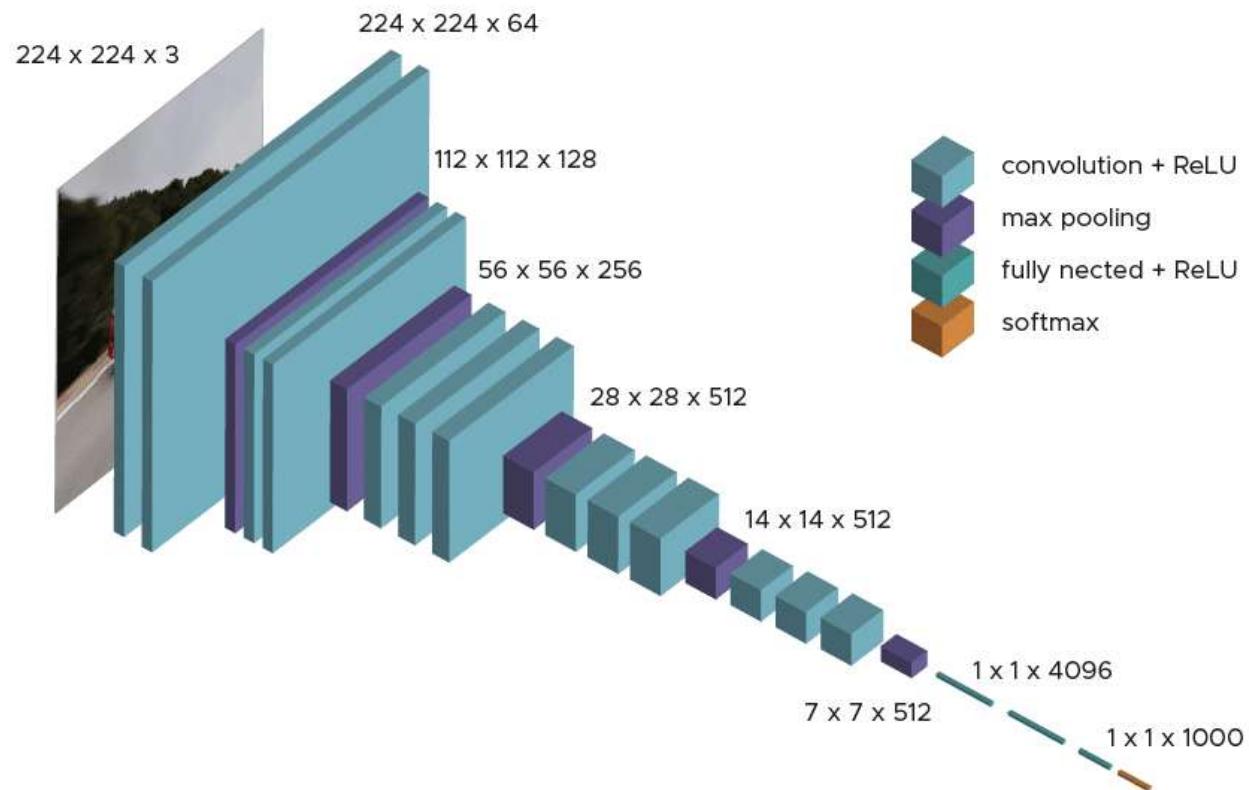
- VGG16 모델: ILSVRC 2014

경진대회에 참여해서 2등을 차지한 모델

- 데이터셋: 120만 장의 이미지
- 1,000개의 클래스로 분류



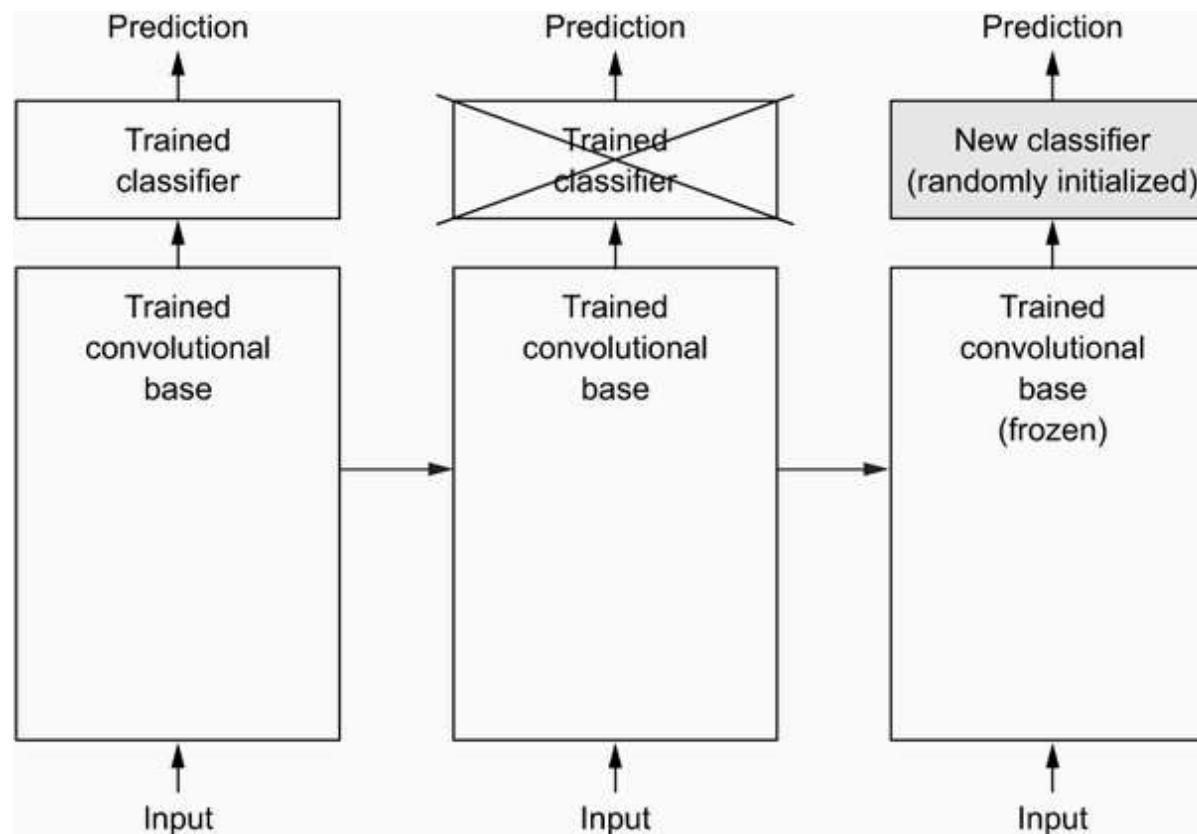
VGG16 모델 구조



유명 합성곱 신경망 모델

- VGG16
- Xception
- ResNet
- MobileNet
- EfficientNet
- DenseNet
- ...

전이 학습



VGG16 모델을 이용한 전이 학습

```
conv_base = keras.applications.vgg16.VGG16(  
    weights="imagenet",  
    include_top=False,  
    input_shape=(180, 180, 3))
```

특성 추출

- 전이 학습 모델을 이용한 데이터 변환
- 두 가지 방식 소개

1) 단순 특성 추출

```
def get_features_and_labels(dataset):
    all_features = []
    all_labels = []

    # 배치 단위로 VGG16 모델 적용
    for images, labels in dataset:
        preprocessed_images = keras.applications.vgg16.preprocess_input(images)
        features = conv_base.predict(preprocessed_images)
        all_features.append(features)
        all_labels.append(labels)

    # 생성된 배치를 하나의 텐서로 묶어서 반환
    return np.concatenate(all_features), np.concatenate(all_labels)
```

훈련셋, 검증셋, 테스트셋 변환

```
train_features, train_labels = get_features_and_labels(train_dataset)
val_features, val_labels = get_features_and_labels(validation_dataset)
test_features, test_labels = get_features_and_labels(test_dataset)
```

- 간단한 분류 모델을 구성
- 변환된 데이터셋을 훈련 데이터셋으로 사용

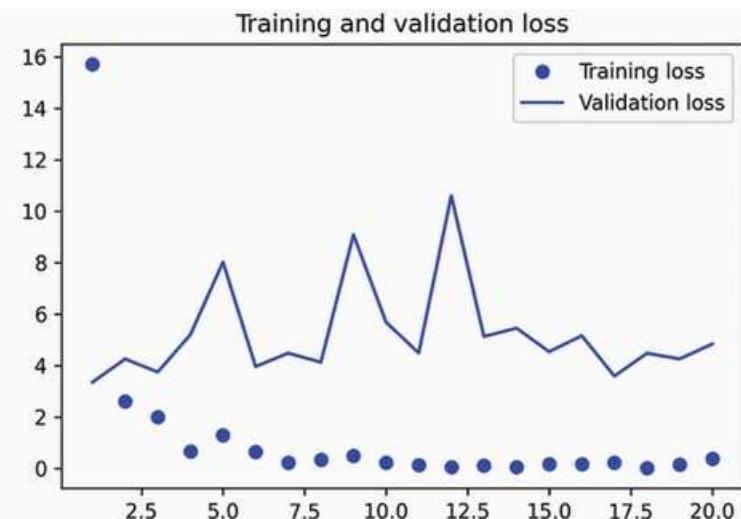
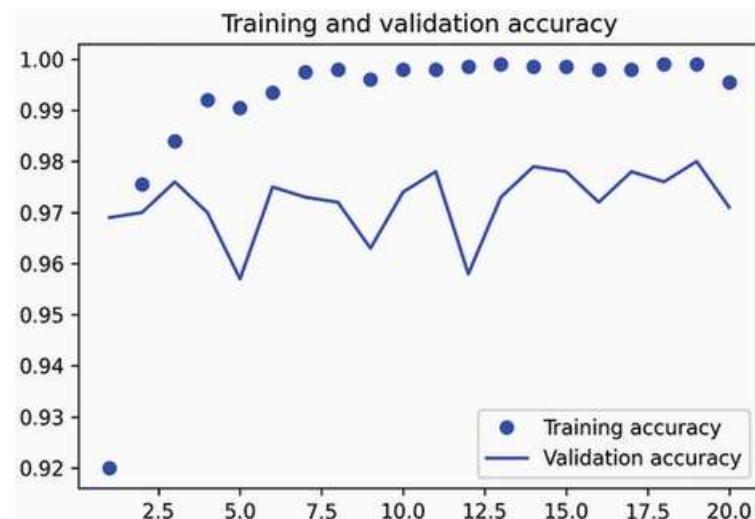
```
# 입력 층
inputs = keras.Input(shape=(5, 5, 512))

# 은닉 층
x = layers.Flatten()(inputs)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)

# 출력 층
outputs = layers.Dense(1, activation="sigmoid")(x)

# 모델
model = keras.Model(inputs, outputs)
```

- 검증셋 정확도: 97% 정도까지 향상
- 과대적합이 매우 빠르게 발생
- 훈련셋이 너무 작기 때문



2) 데이터 증식과 특성 추출

- 데이터 증식 기법을 활용
- VGG16 합성곱 기저(베이스)를 구성요소로 사용하는 모델을 직접 정의
- **동결(freezing)** 기법 적용
- 입력 데이터의 모양도 미리 지정하지 않음

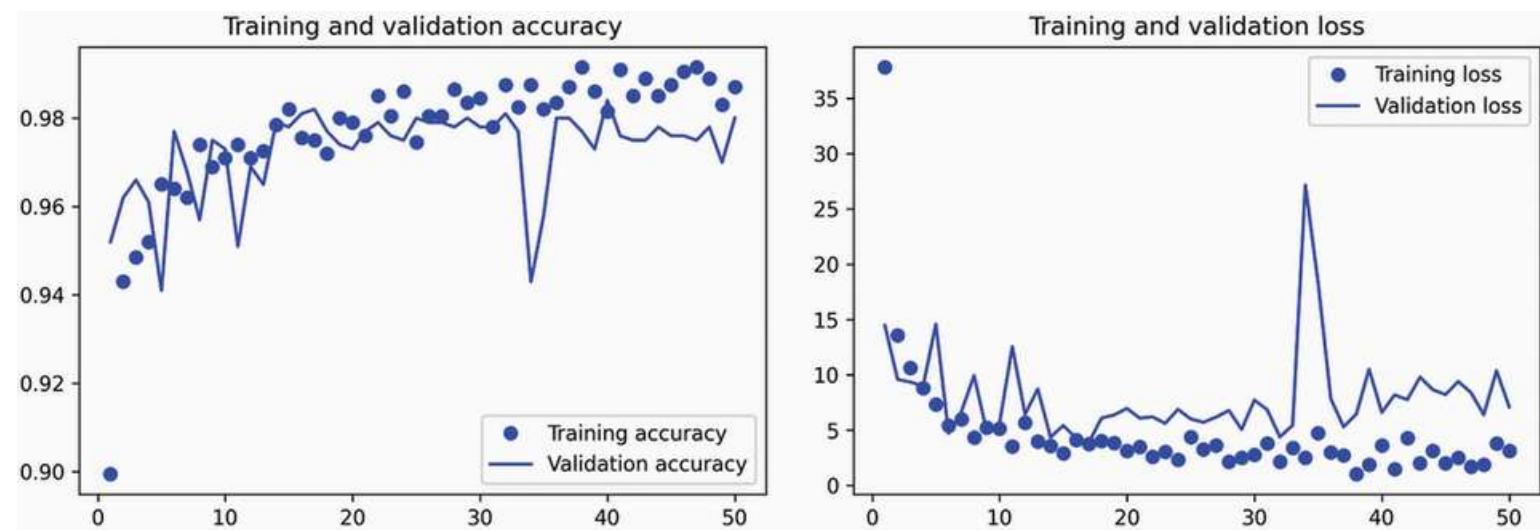
```
conv_base = keras.applications.vgg16.VGG16(  
    weights="imagenet",  
    include_top=False)  
  
# 기저 동결  
conv_base.trainable = False
```

```
# 모델 구성
inputs = keras.Input(shape=(180, 180, 3))

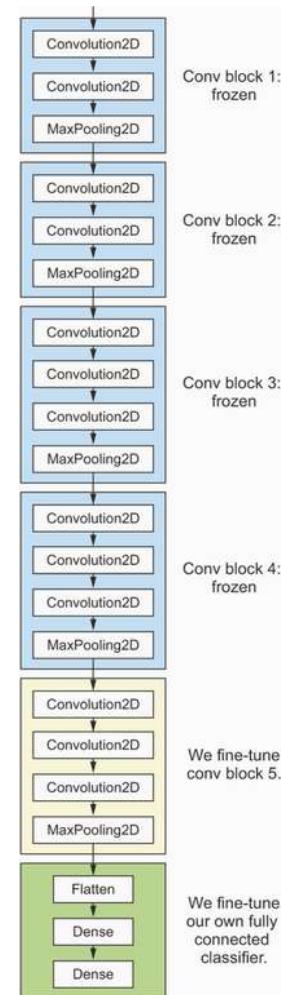
x = data_augmentation(inputs) # 데이터 증식
x = keras.applications.vgg16.preprocess_input(x) # VGG16용 전처리
x = conv_base(x) # VGG16 베이스
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)

outputs = layers.Dense(1, activation="sigmoid")(x) # 출력층

model = keras.Model(inputs, outputs)
```



모델 미세 조정



```
conv_base.trainable = True
for layer in conv_base.layers[:-4]:
    layer.trainable = False
```