

컴퓨터 비전 기초·합성 곱 신경망

주요 내용

- 합성곱 신경망
- 데이터 증식
- 모델 재활용: 전이학습

합성곱 신경망

- 2011년부터 2015년: 컴퓨터 비전 분야에서 딥러닝 기법이 획기적으로 발전
- 사진 검색, 자율주행, 로봇공학, 의학 진단 프로그램, 얼굴 인식 등 일상의 많은 영역에서 활용됨
- 컴퓨터 비전 분야 딥러닝 모델: **CNN** 또는 **convnet**으로 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: "model"
```

Layer (type)	Output Shape	Param #
<hr/>		
input_1 (InputLayer)	[None, 28, 28, 1]	0
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73856
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 10)	11530
<hr/>		
Total params: 104,202		
Trainable params: 104,202		
Non-trainable params: 0		

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"])

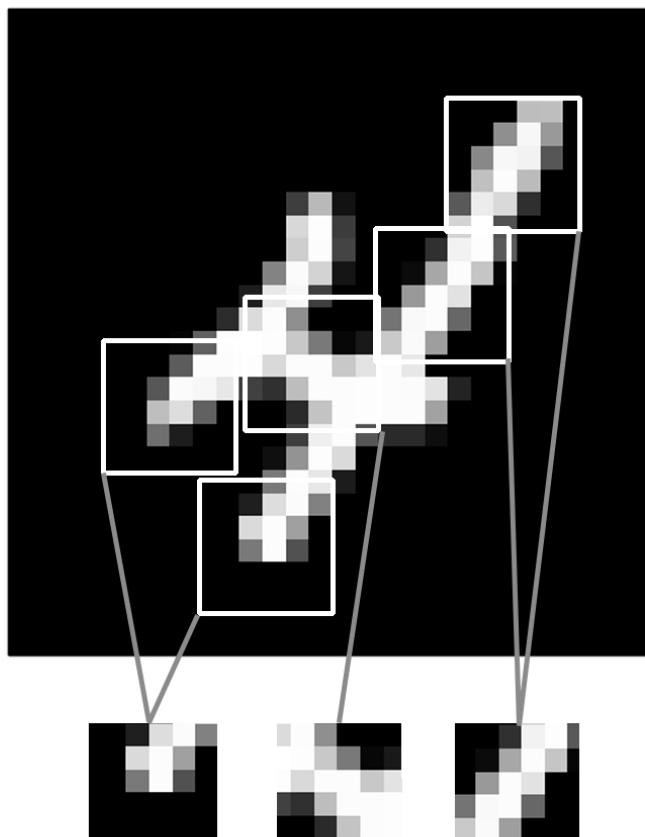
model.fit(train_images, train_labels, epochs=5, batch_size=64)
```

합성곱 연산

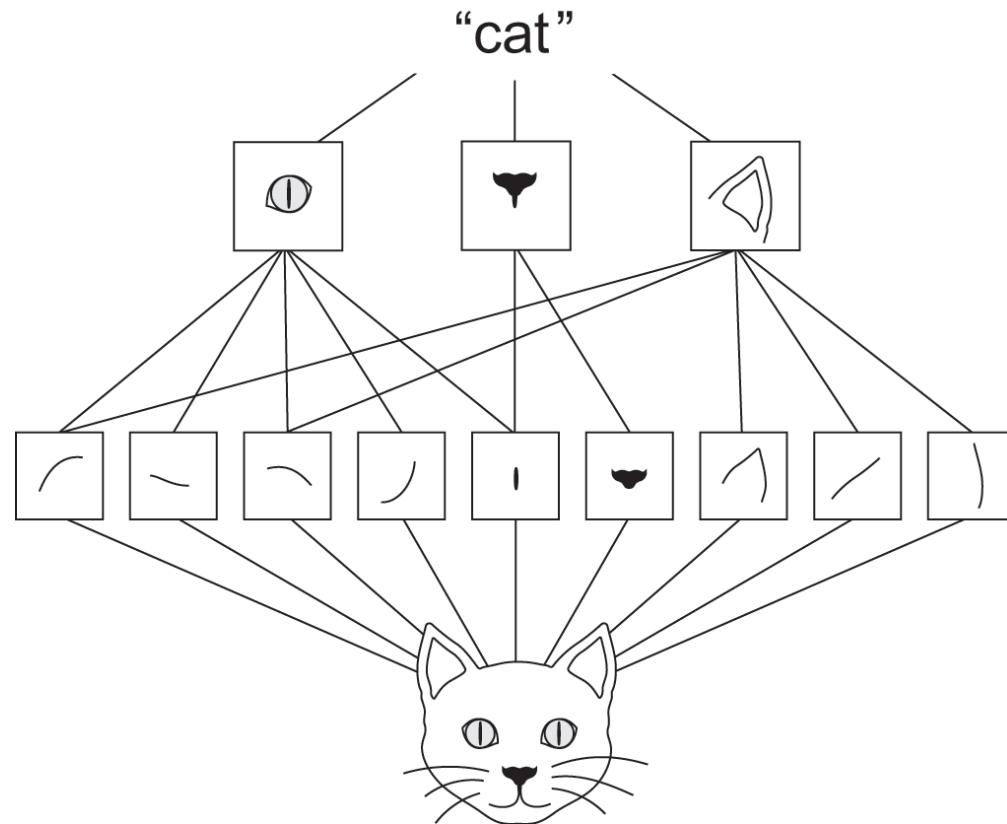
- Dense 층: 입력값의 전체 특성을 대상으로 한 번의 아핀 변환 적용
- Conv2D 층: `kernel_size`로 지정된 크기의 공간에 대해 여러 개의 아핀 변환 적용.

합성곱 층의 특징

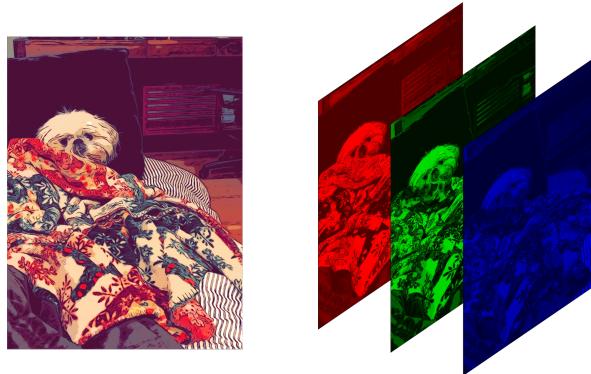
첫째, 위치와 무관하게 패턴을 찾아낸다. 즉, 서로 다른 위치에 있는 동일한 패턴은 동일한 방식으로 인식된다.



둘째, 패턴 공간의 계층 파악



컬러 이미지와 채널

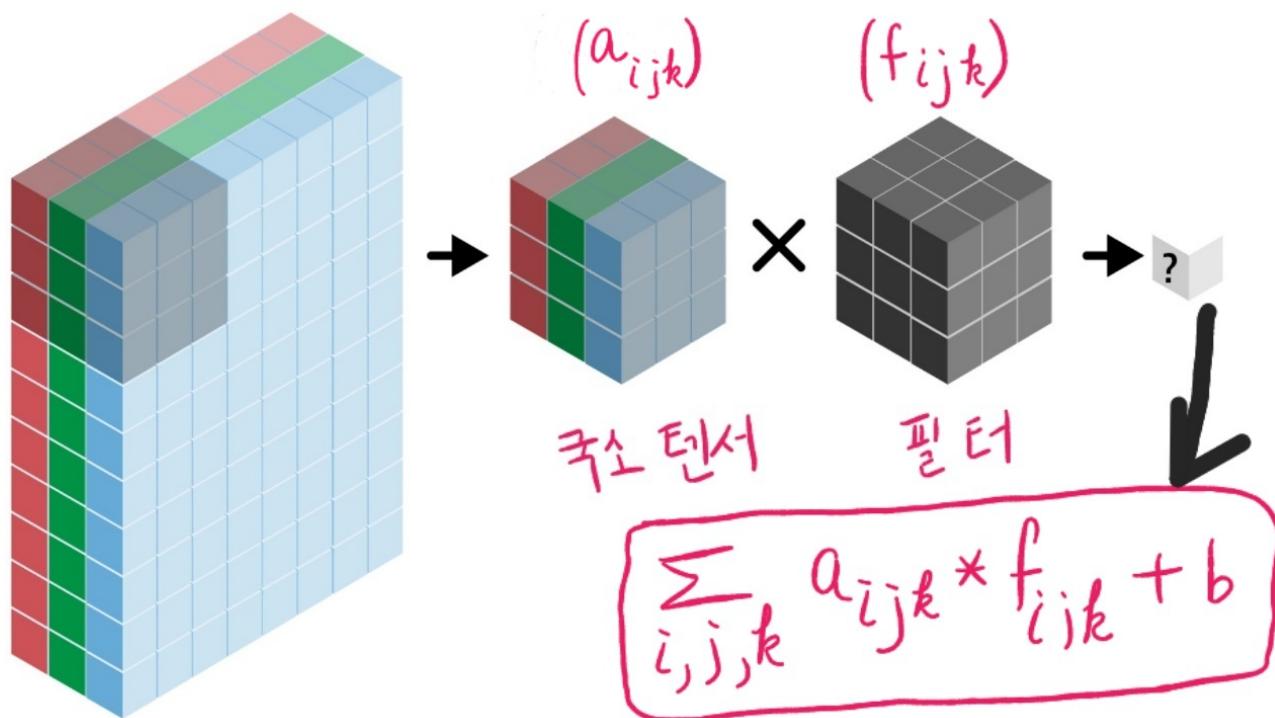


		나비	
		0 1 2	
	0	.392 .482 .576	
이	1	.478 .63 .169 .263 .376	
이	2	.580 .79 .263 .44 .306 .376 .451	
	0	.373 .60 .376 .443 .569 .674	
	1		
	2		
		채널	

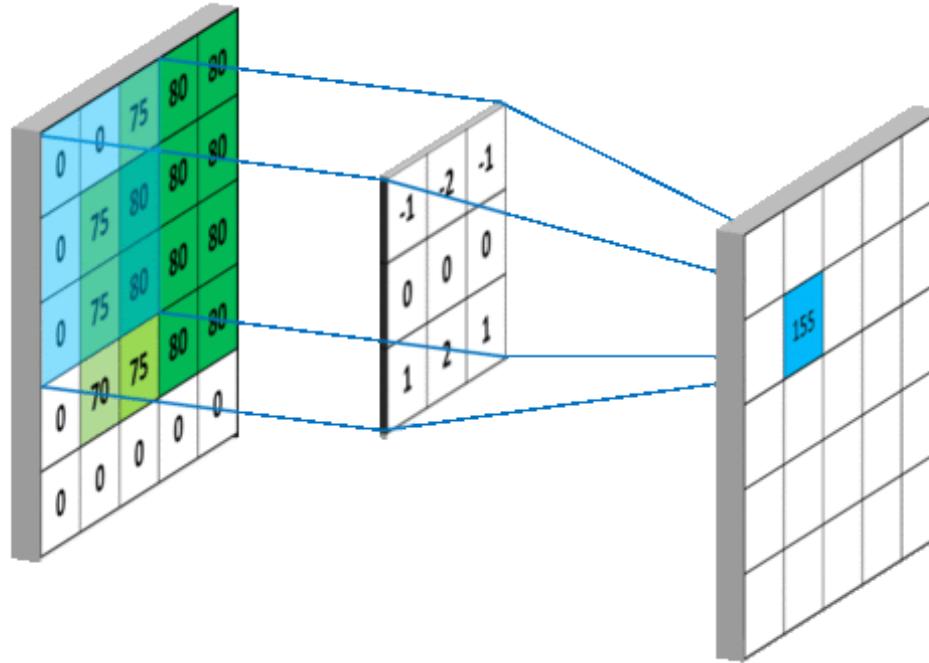
특성맵(채널), 필터, 출력맵

- **특성맵** feature map 또는 **채널** channel:
 - (높이, 너비) 모양의 2D 텐서.
 - 예제: 컬러사진의 경우 세 개의 채널(특성맵)로 구성됨. 채널 수를 깊이라 부름.
- **필터** filter: `kernel_size` 를 이용한 3D 텐서.
 - 예제: `kernel_size=3` 인 경우 필터는 $(3, 3, \text{입력샘플의깊이})$ 모양의 3D 텐서.
- **출력맵** response map: 입력 샘플을 대상으로 하나의 필터를 적용해서 생성된 하나의 특성 맵(채널).

필터 적용



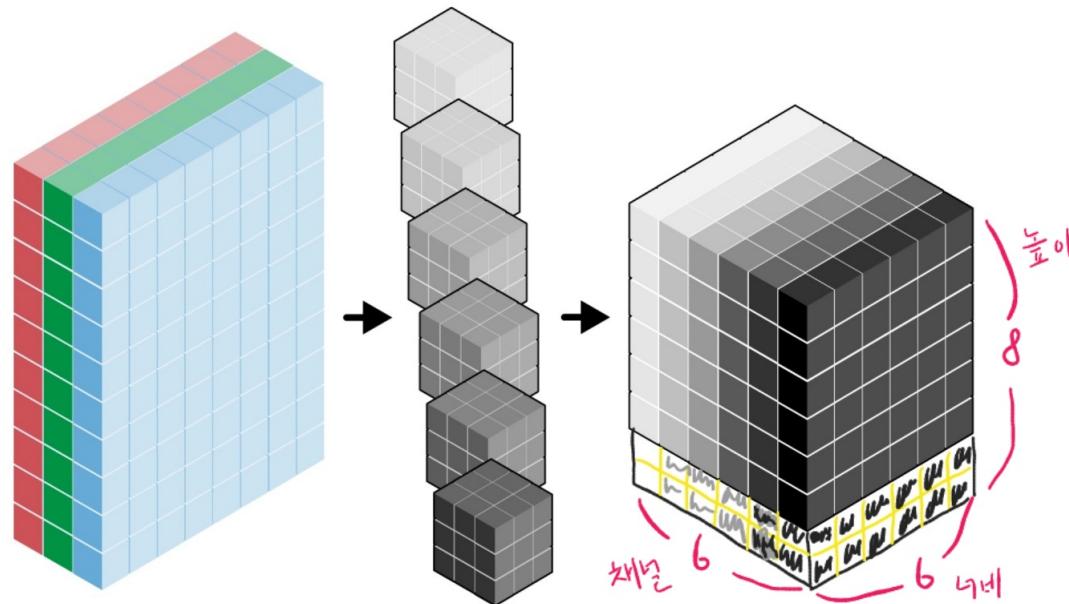
출력맵



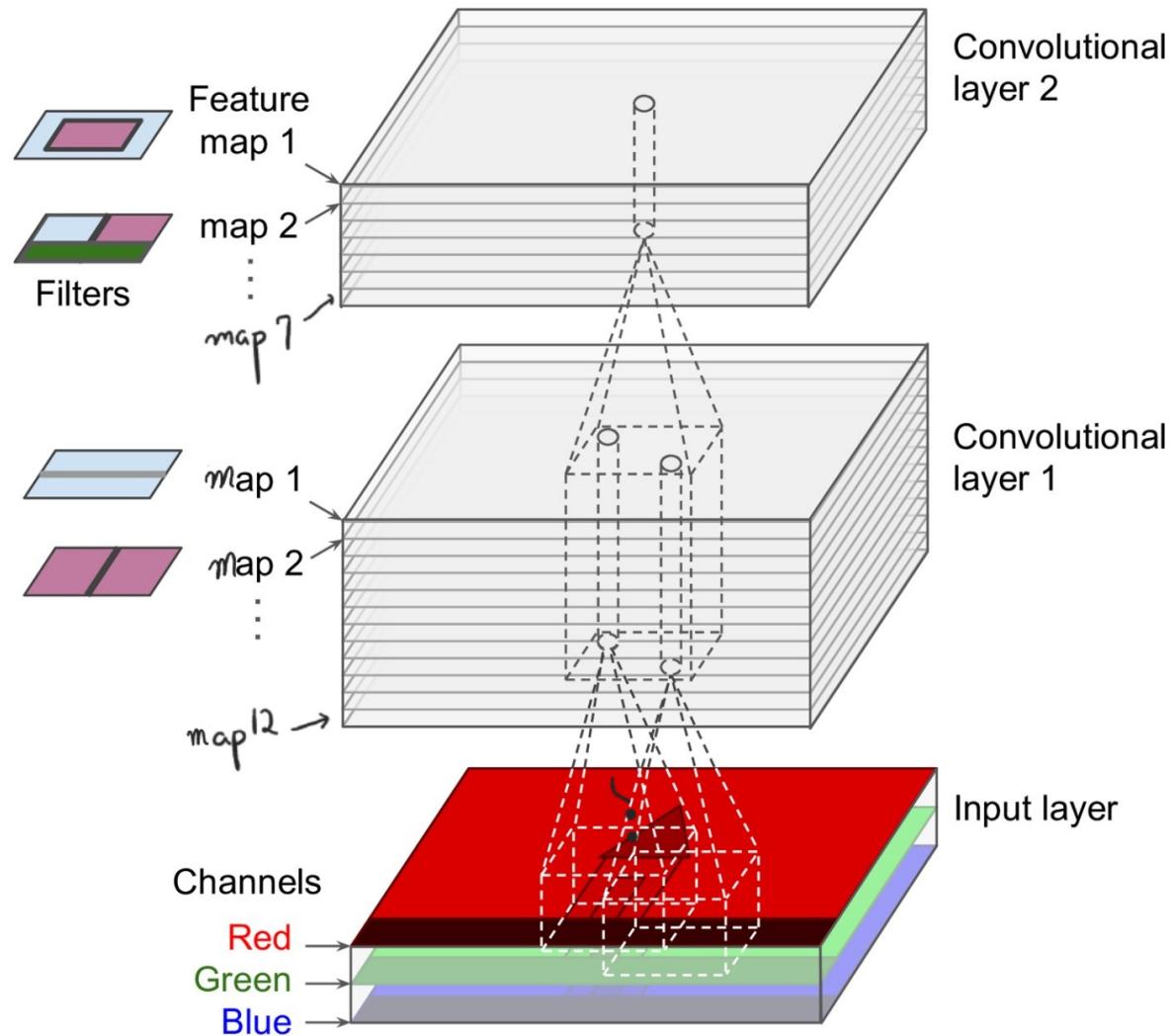
```
0 * -1 + 0 * -2 + 75 * -1 +
0 * 0 + 75 * 0 + 80 * 0 +
0 * 1 + 75 * 2 + 80 * 1 +
0
= 155
```

필터와 출력맵

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



합성곱 층 연속 적용

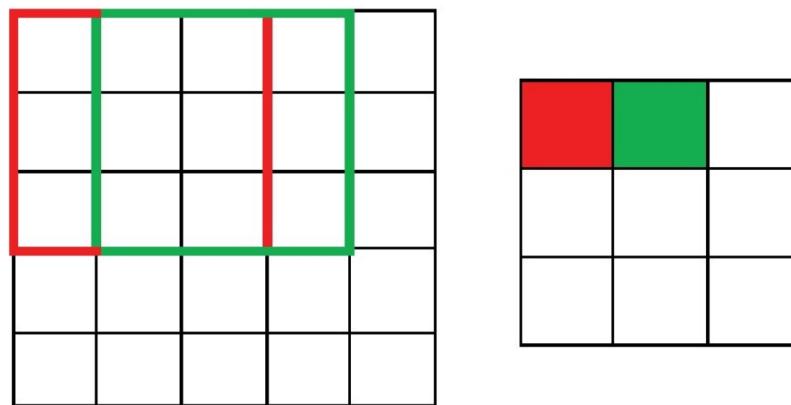


패딩과 보폭

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

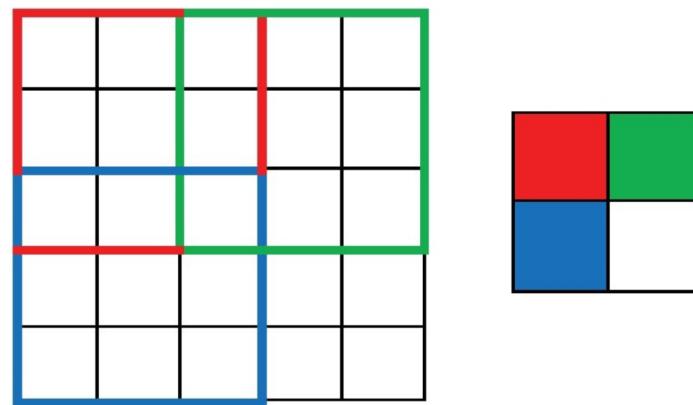
경우 1: 패딩 없음, 보폭은 1

- 필터가 1칸씩 슬라이딩
- 출력 특성맵의 깊이와 너비: 3×3
- 출력 특성맵의 깊이와 너비가 줄어듦.



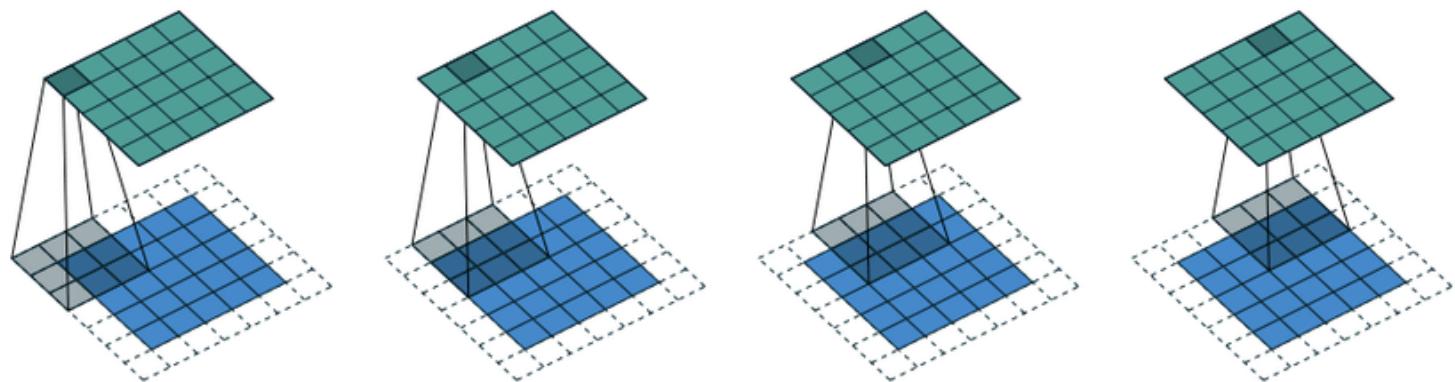
경우 2: 패딩 없음, 보폭은 2

- 필터 2칸씩 건너 뛰며 슬라이딩
- 출력 특성맵의 깊이와 너비: 2×2
- 출력 특성맵의 깊이와 너비가 보폭의 반비례해서 줄어듦.

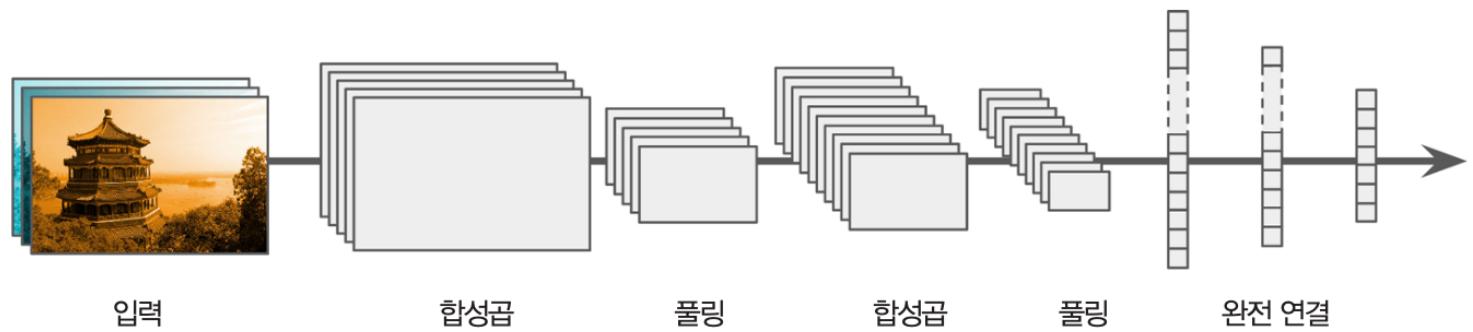


경우 3: 패딩 있음, 보폭은 1

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

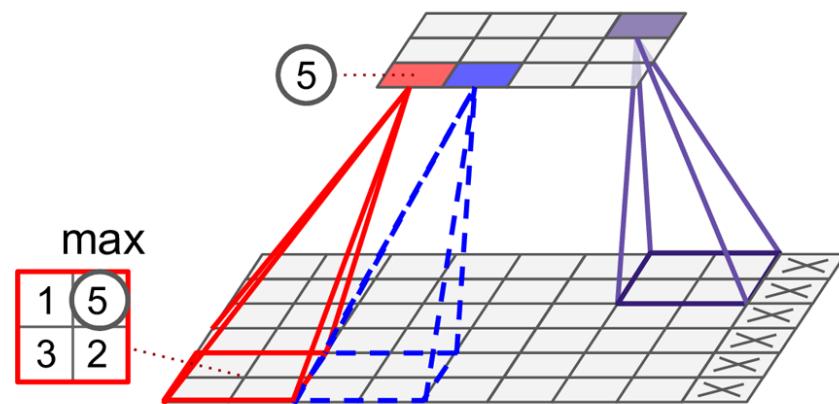


CNN 모델의 전형



맥스 풀링 기능

`layers.MaxPooling2D(pool_size=2)(x)`



맥스 풀링 사용 이유

첫째, 모델이 학습 파라미터(가중치와 편향) 수를 줄인다.

```
>>> model_no_max_pool.summary()
```

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[None, 28, 28, 1]	0
conv2d_3 (Conv2D)	(None, 26, 26, 32)	320
conv2d_4 (Conv2D)	(None, 24, 24, 64)	18496
conv2d_5 (Conv2D)	(None, 22, 22, 128)	73856
flatten_1 (Flatten)	(None, 61952)	0
dense_1 (Dense)	(None, 10)	619530
<hr/>		
Total params: 712,202		
Trainable params: 712,202		
Non-trainable params: 0		

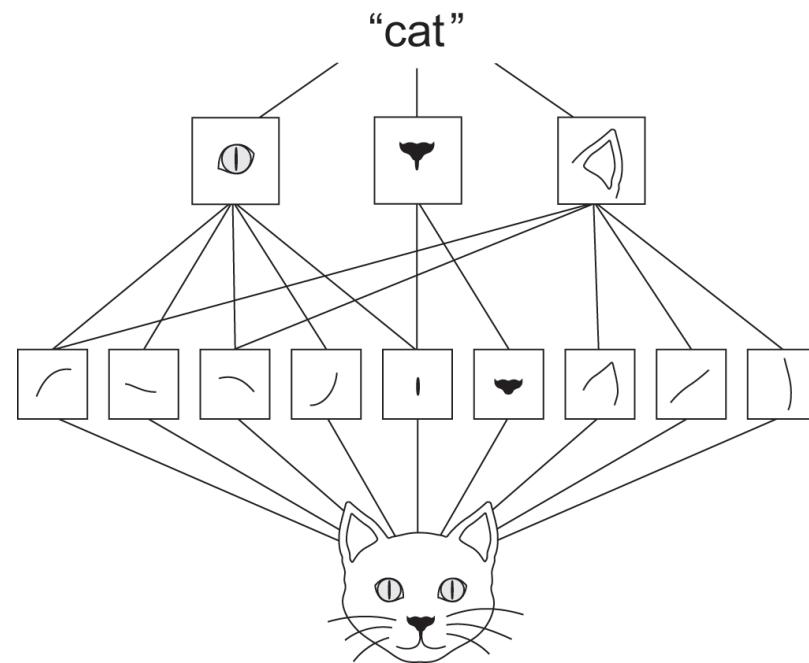
맥스 풀링 사용 모델의 경우는 다음과 같다.

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

```
Model: "model"
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[None, 28, 28, 1]	0
conv2d (Conv2D)	(None, 26, 26, 32)	320
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73856
flatten (Flatten)	(None, 1152)	0
dense (Dense)	(None, 10)	11530
<hr/>		
Total params: 104,202		
Trainable params: 104,202		
Non-trainable params: 0		

둘째, 상위 합성곱 층으로 이동 수록 입력 데이터의 보다 넓은 영역에 대한 정보를 획득한다.



합성곱 신경망 실전 활용 예제

작은 데이터셋과 딥러닝 모델

- 이미지 분류 모델 훈련: 데이터셋의 크기가 일반적으로 작음
- 모델은 개와 고양이 사진을 대상으로 하는 이진 분류 합성곱 신경망 모델 훈련
- 실전 상황을 재현: 5천 개의 이미지로 이루어진 작은 데이터셋

데이터 다운로드

- 캐글(Kaggle) 계정 필요
- 캐글의 "Account" 페이지의 계정 설정 창: "API" 항목에서 "Create New API Token"을 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

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

모델 지정

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

모델 컴파일

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

데이터 불러오기와 전처리

사진들을 무작위로 섞어 크기가 32인 배치들로 구성된 훈련셋, 검증셋, 테스트셋을 지정

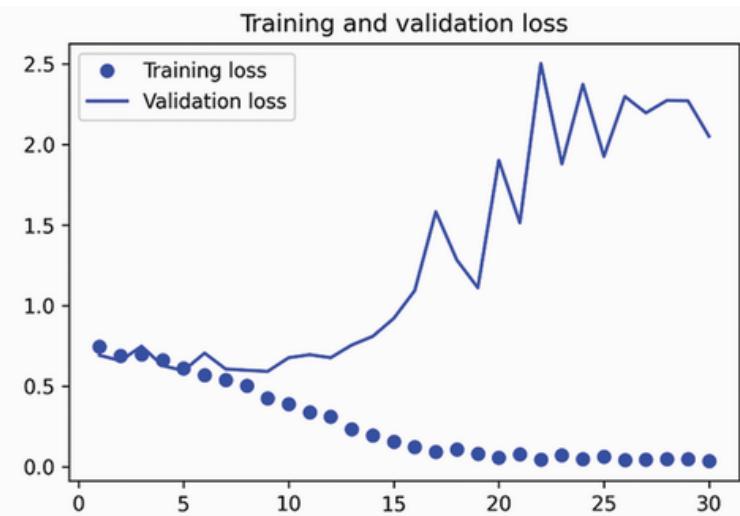
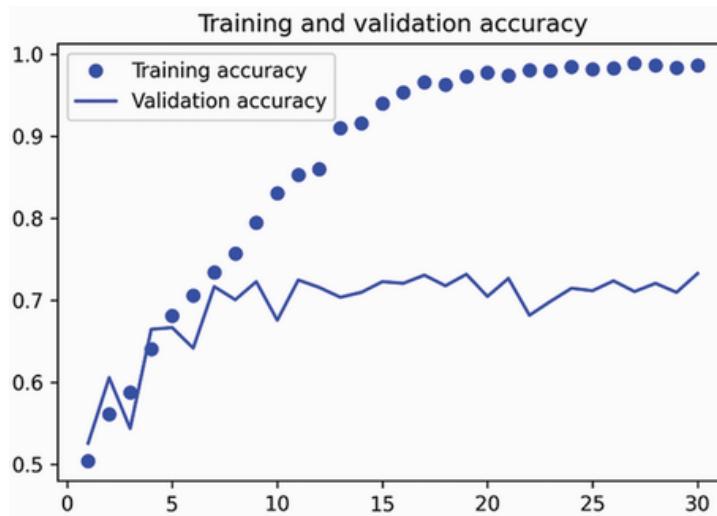
```
from tensorflow.keras.utils import image_dataset_from_directory  
  
new_base_dir = pathlib.Path("cats_vs_dogs_small")  
  
train_dataset = image_dataset_from_directory(  
    new_base_dir / "train",  
    image_size=(180, 180),  
    batch_size=32)  
  
validation_dataset = image_dataset_from_directory(  
    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)
```

모델 훈련

- 크기가 32인 배치 단위로 이미 묶여 있음
- `fit()` 메서드: 배치 크기(`batch_size`)는 지정 불필요
- 텐서플로우가 배치 단위의 입력 처리 가능

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

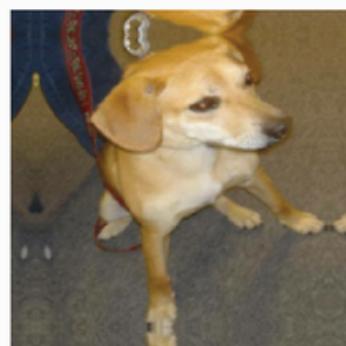
과대 적합 빠르게 발생



데이터 증식

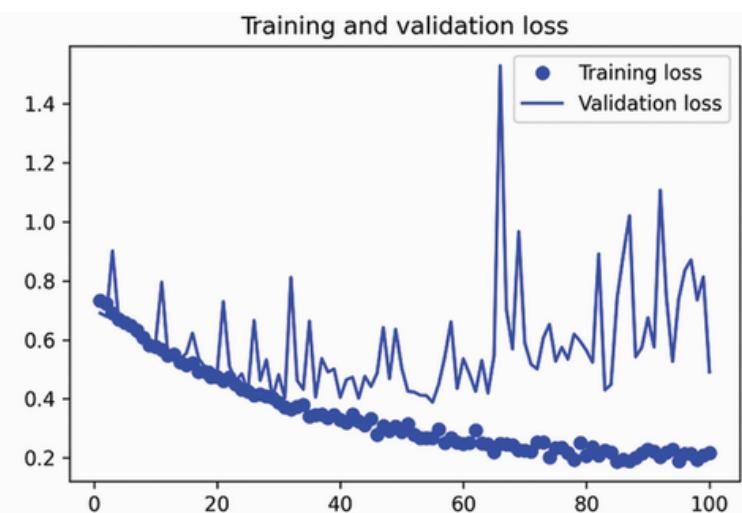
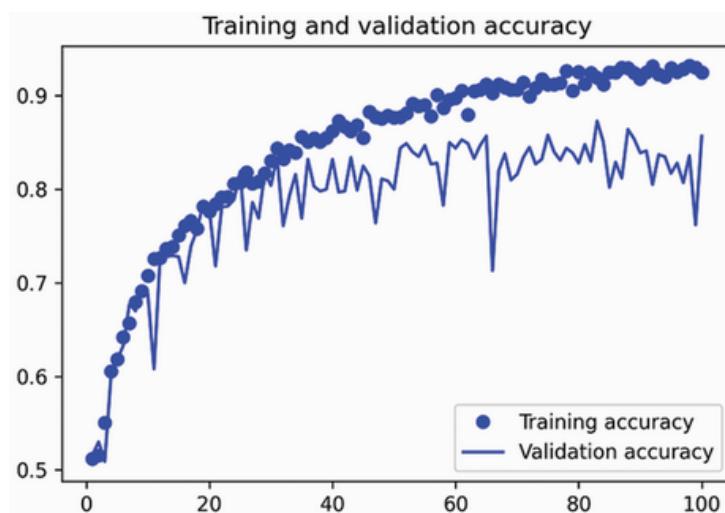
- `RandomFlip()`: 사진을 50%의 확률로 지정된 방향으로 반전.
- `RandomRotation()`: 사진을 지정된 범위 안에서 임의로 좌우로 회전
- `RandomZoom()`: 사진을 지정된 범위 안에서 임의로 확대 및 축소

```
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

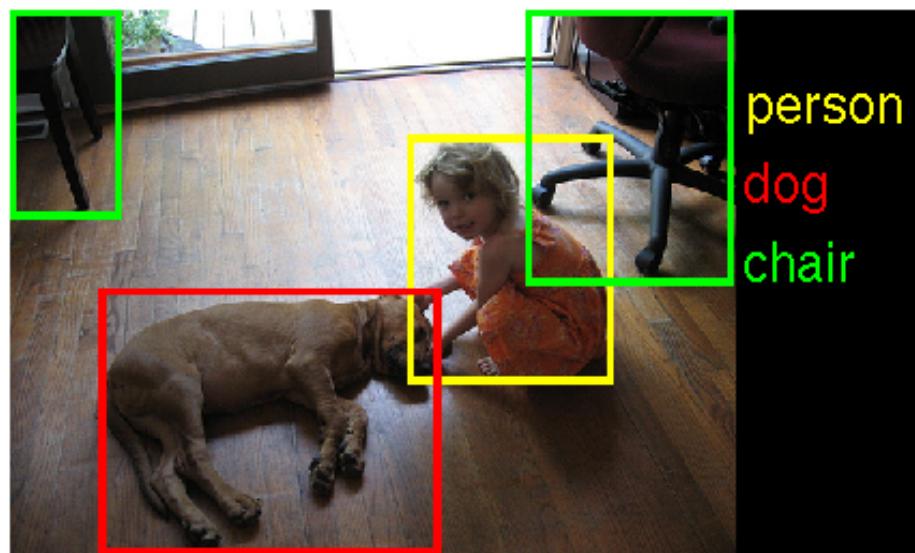


모델 재활용

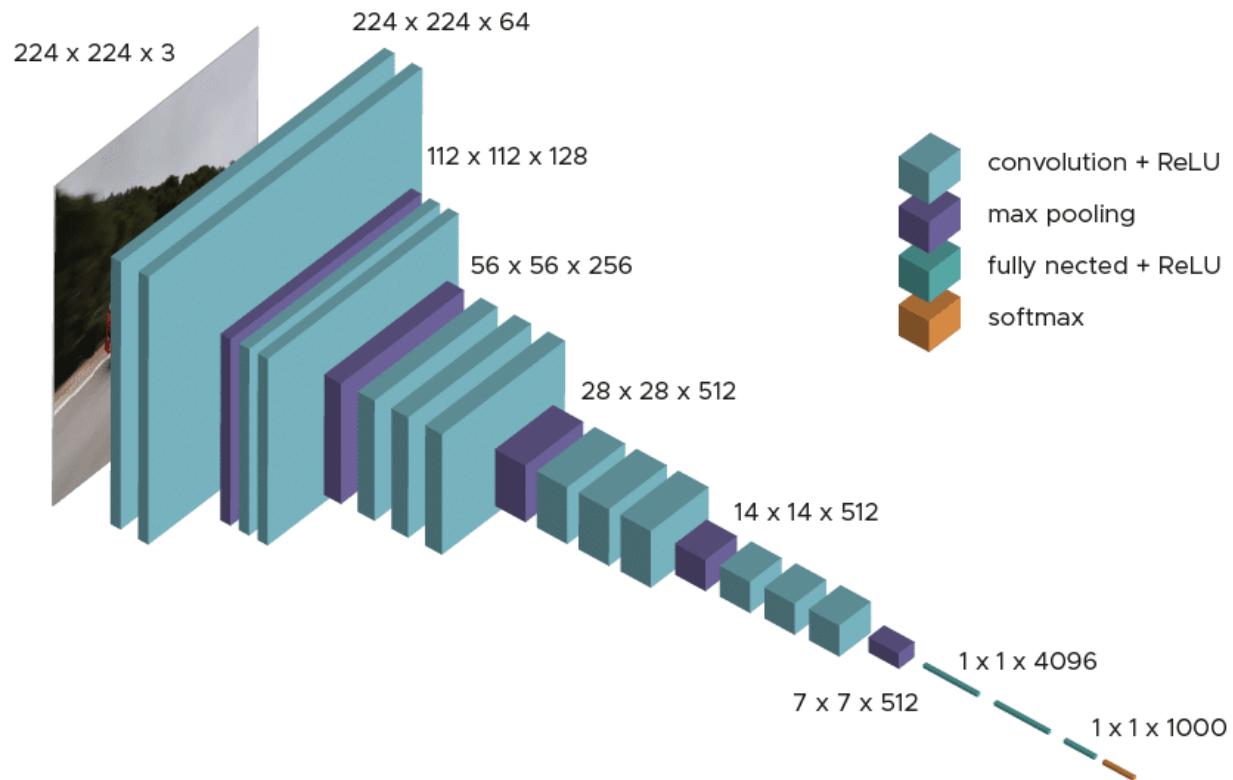
- 전이 학습 transfer learning
- 모델 미세조정 model fine tuning

VGG16 모델

- VGG16 모델: [ILSVRC 2014](#) 경진대회에 참여해서 2등을 차지한 모델
- 데이터셋: 120만 장의 이미지
- 1,000개의 클래스로 분류



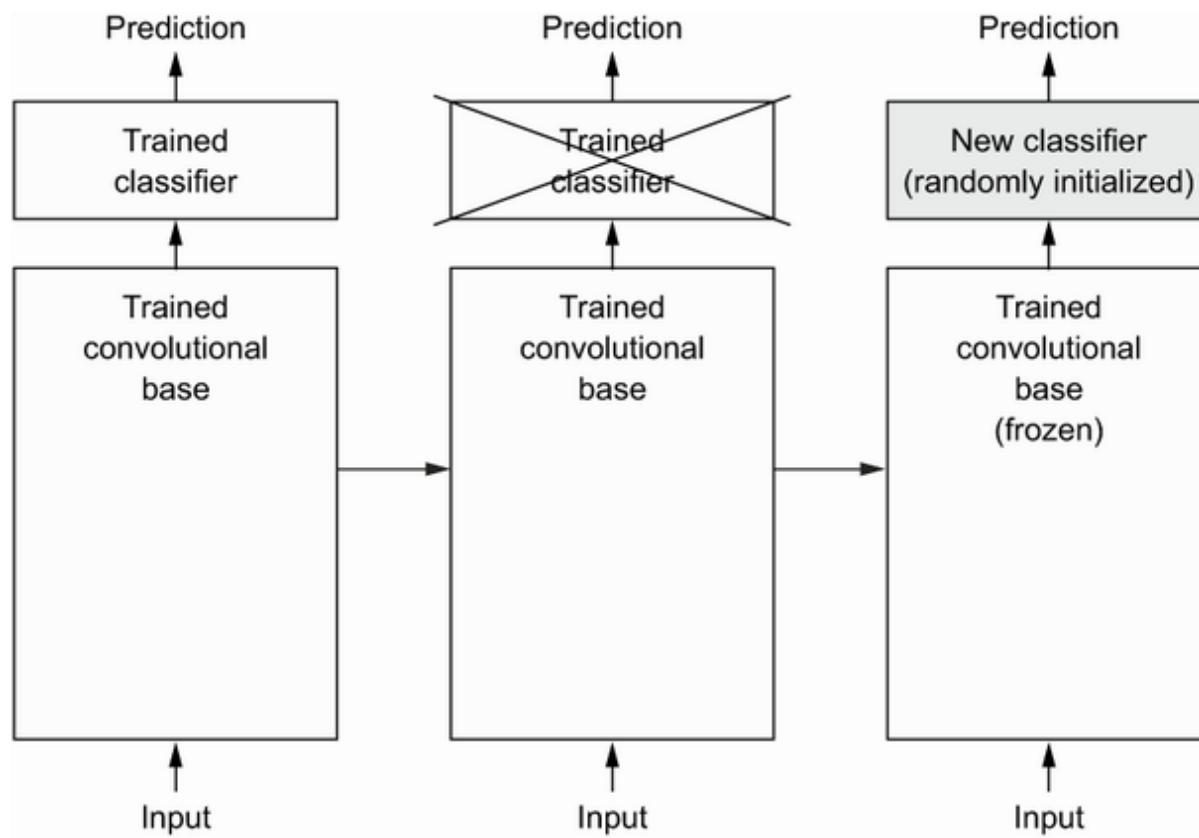
VGG16 모델 구조



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)
```

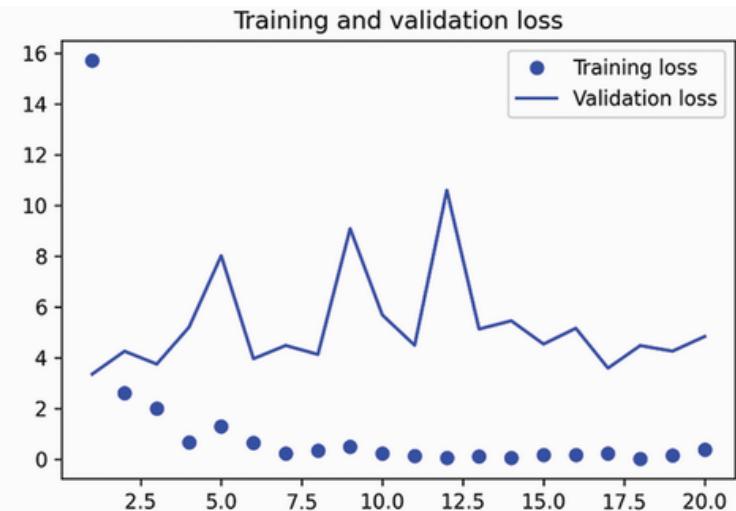
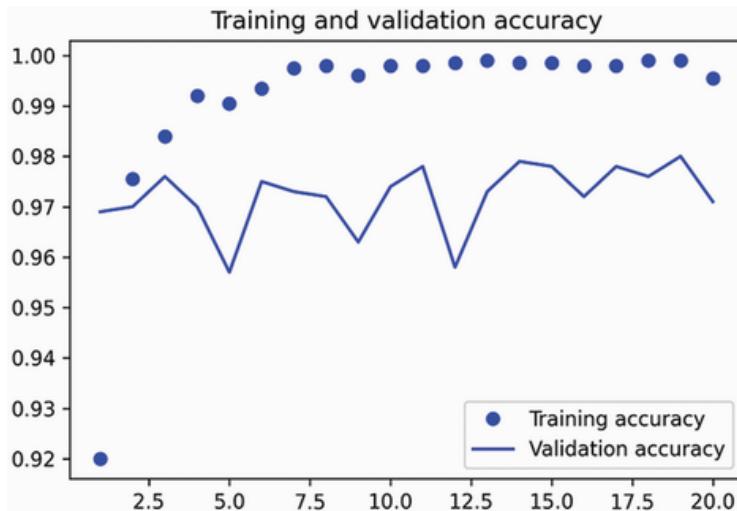
훈련셋, 검증셋, 테스트셋 변환

```
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

```
# 입력층  
inputs = keras.Input(shape=(5, 5, 512))  
  
# 은닉층  
x = layers.Flatten()(inputs)  
x = layers.Dense(256)(x)  
x = layers.Dropout(0.5)(x)  
  
# 출력층  
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

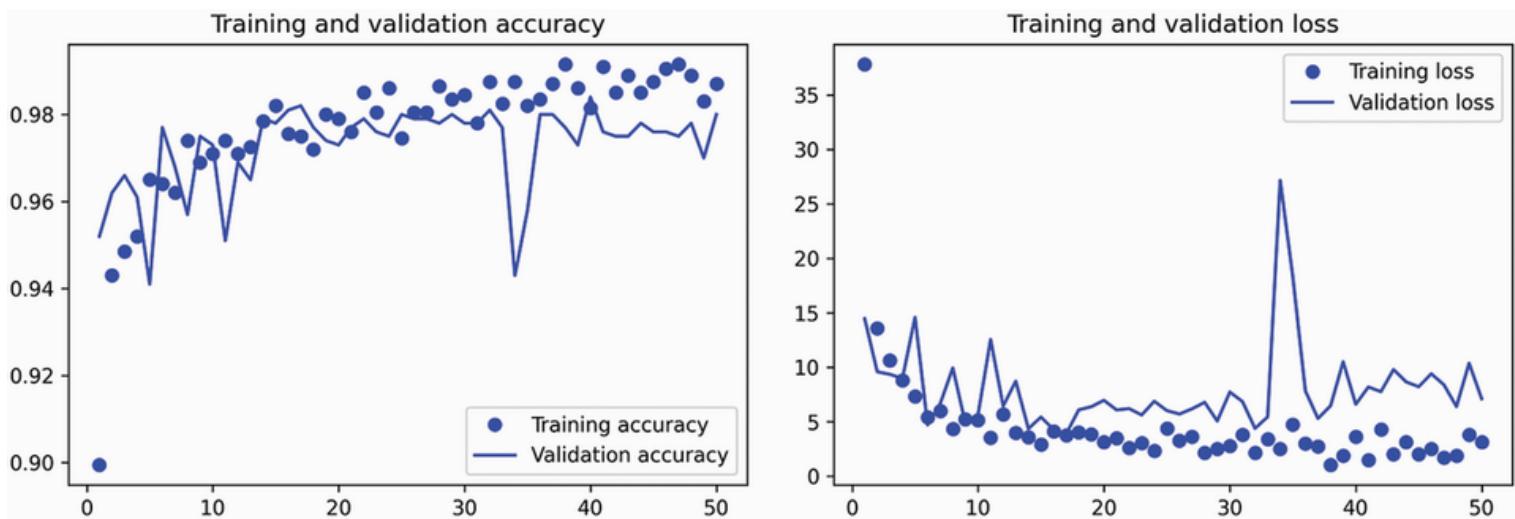
```
conv_base = keras.applications.vgg16.VGG16(  
    weights="imagenet",  
    include_top=False)  
  
# 기저 동결  
conv_base.trainable = False
```

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

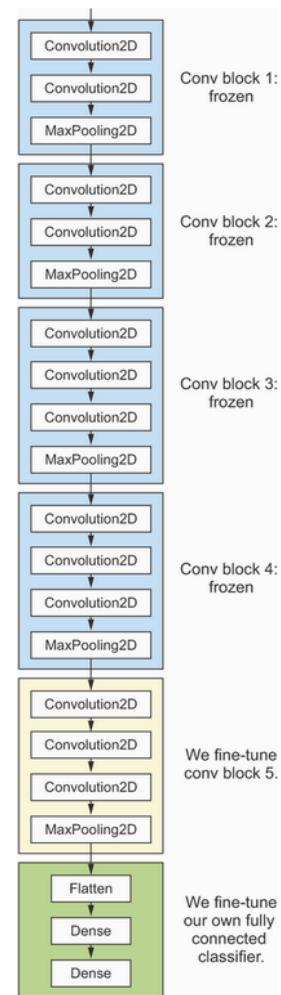
x = data_augmentation(inputs) # 데이터 증식
x = keras.applications.vgg16.preprocess_input(x) # VGG16용 전처리
x = conv_base(x) # VGG16 베이스
x = layers.Flatten()(x)
x = layers.Dense(256)(x)
x = layers.Dropout(0.5)(x)

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
```