

고급 컴퓨터 비전

주요 내용

- 합성곱 신경망의 주요 활용 분야(컴퓨터 비전)
 - 이미지 분류
 - 이미지 분할
 - 객체 탐지
- 합성곱 신경망 기본 아키텍처
 - 잔차 연결
 - 배치 정규화
 - 채널 분리 합성곱

컴퓨터 비전 주요 과제

Single-label multi-class classification



- Biking
- Running
- Swimming

Multi-label classification

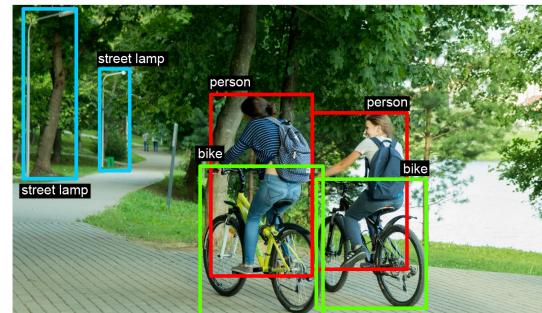


- Bike
- Person
- Tree
- Car
- Boat
- House

Image segmentation

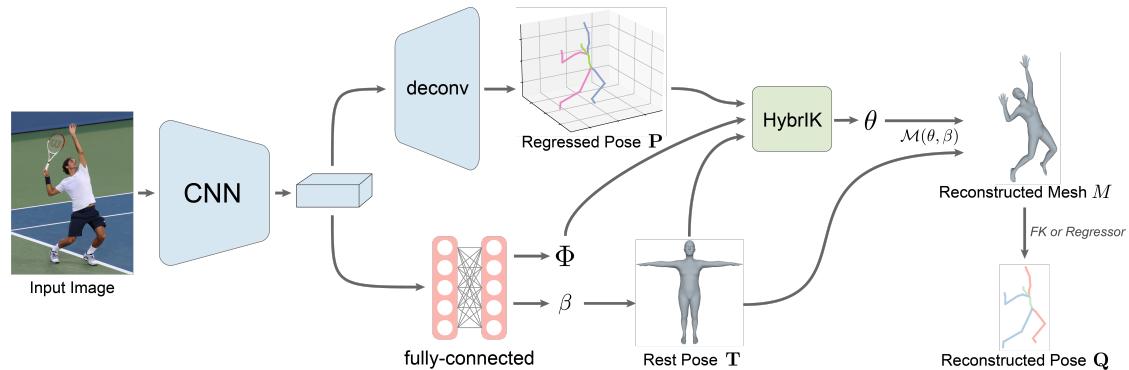


Object detection



기타 다양한 활용법

- 이미지 유사도 측정(image similarity scoring)
- 키포인트 탐지(keypoint detection)
- 자세 추정(pose estimation)
- 3D 메쉬 추정(3D mesh estimation)

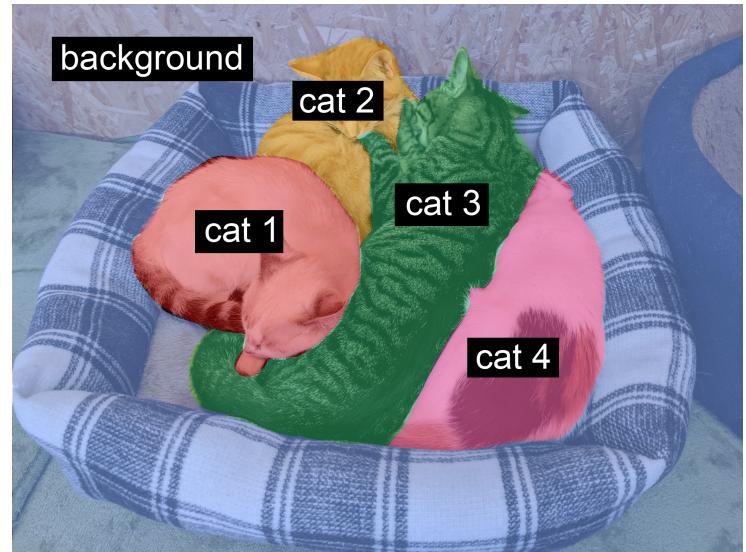


객체 탐지

- 작동원리 설명: 핸즈온 머신러닝 14장
- 기초 활용법: [RetinaNet 활용 객체 탐지](#)
- 주요 활용 예제: [YOLOv5](#)

이미지 분할

- 시맨틱 분할(semantic segmentation)
- 인스턴스 분할(instance segmentation)

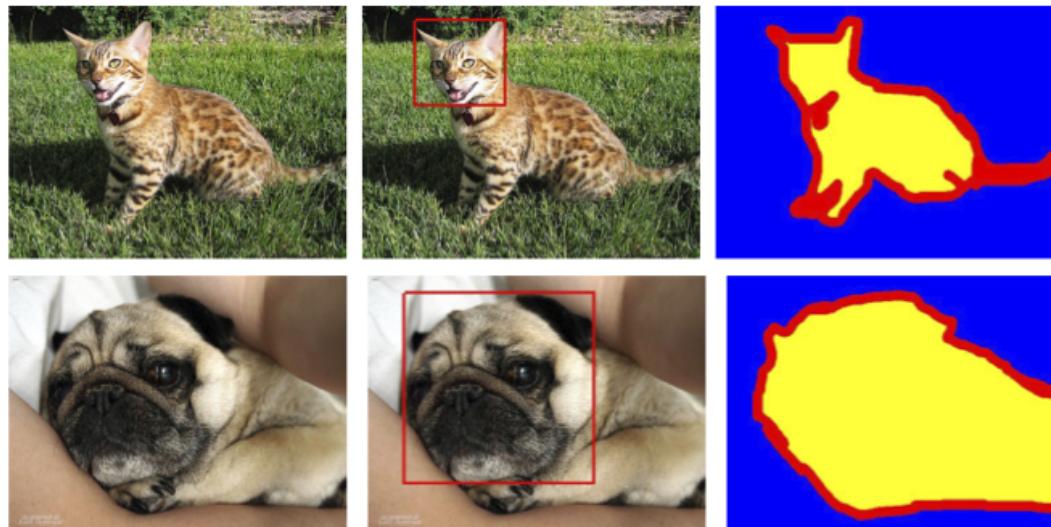


Oxford-IIIT 애완동물 데이터셋

- 데이터셋 크기: 7,390
- 클래스(범주) 개수: 37
- 클래스별 사진 수: 약 200 장
- 사진별 라벨: 종과 품종, 머리 표시 경계상자, 트라이맵 분할^{trimap segmentation} 마스크 등 4 종류로 구성

트라이맵 분할 마스크

- 원본 사진과 동일한 크기의 칼라 사진
- 1, 2, 3 셋 중에 하나의 값만 사용
 - 1: 동물의 몸에 해당하는 픽셀
 - 2: 배경에 해당하는 픽셀
 - 3: 동물과 배경을 구분하는 경계에 해당하는 픽셀



이미지 분할 모델 구성: 다운 샘플링

```
inputs = keras.Input(shape=(200, 200, 3,))  
x = layers.Rescaling(1./255)(inputs)  
  
x = layers.Conv2D(64, 3, strides=2, activation="relu", padding="same")(x)  
x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)  
x = layers.Conv2D(128, 3, strides=2, activation="relu", padding="same")(x)  
x = layers.Conv2D(128, 3, activation="relu", padding="same")(x)  
x = layers.Conv2D(256, 3, strides=2, padding="same", activation="relu")(x)  
x = layers.Conv2D(256, 3, activation="relu", padding="same")(x)
```

이미지 분할 모델 구성: 업 샘플링

```
x = layers.Conv2DTranspose(256, 3, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(256, 3, activation="relu", padding="same",
strides=2)(x)
x = layers.Conv2DTranspose(128, 3, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(128, 3, activation="relu", padding="same",
strides=2)(x)
x = layers.Conv2DTranspose(64, 3, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(64, 3, activation="relu", padding="same", strides=2)
(x)

outputs = layers.Conv2D(3, 3, activation="softmax", padding="same")(x)

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

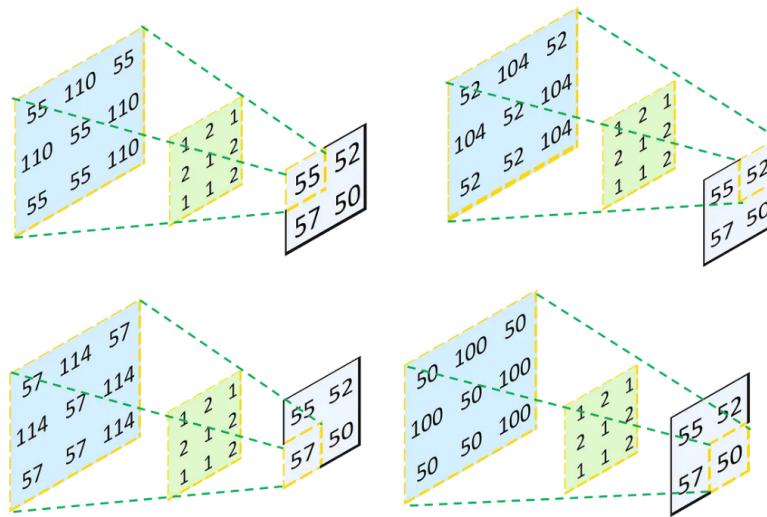
input_1 (InputLayer)	(None, 200, 200, 3)	0
rescaling (Rescaling)	(None, 200, 200, 3)	0
conv2d (Conv2D)	(None, 100, 100, 64)	1792
conv2d_1 (Conv2D)	(None, 100, 100, 64)	36928
conv2d_2 (Conv2D)	(None, 50, 50, 128)	73856
conv2d_3 (Conv2D)	(None, 50, 50, 128)	147584
conv2d_4 (Conv2D)	(None, 25, 25, 256)	295168
conv2d_5 (Conv2D)	(None, 25, 25, 256)	590080
conv2d_transpose (Conv2DTr)	(None, 25, 25, 256)	590080
conv2d_transpose_1 (Conv2DTr)	(None, 50, 50, 256)	590080
conv2d_transpose_2 (Conv2DTr)	(None, 50, 50, 128)	295040
conv2d_transpose_3 (Conv2DTr)	(None, 100, 100, 128)	147584
conv2d_transpose_4 (Conv2DTr)	(None, 100, 100, 64)	73792
conv2d_transpose_5 (Conv2DTr)	(None, 200, 200, 64)	36928
conv2d_6 (Conv2D)	(None, 200, 200, 3)	1731

출력값과 타깃

- 출력값 모양: `(200, 200, 3)`
- 출력층으로 사용된 `Conv2D` 층의 소프트맥스 활성화 함수: 0, 1, 2 중에 하나의 값을 예측해야 하기에 3 개의 출력맵으로 구성된 텐서에 대해 텐서의 마지막 차원, 즉 픽셀별로 작동.
- 모델의 손실함수: `sparse_categorical_crossentropy` 지정. 타깃 정수 0, 1, 2가 각각 `(1, 0, 0)`, `(0, 1, 0)`, `(0, 0, 1)`에 상응하도록 작동

업샘플링 작동법

- 검정 테두리로 표시된 2×2 모양의 텐서에 포함된 하나의 항목에 초록색 바탕의 3×3 모양의 필터를 적용하여 원편에 위치한 파랑색 바탕의 3×3 모양의 텐서 생성
- 이 과정을 2×2 텐서의 모든 항목에 대해 적용. 단, 생성되는 3×3 모양의 텐서는 지정된 보폭 크기만큼 오른쪽 또는 아래로 이동시키며 겹침.
- 중첩되는 영역에 포함된 항목은 해당 항목에 대해 계산된 모든 값들의 합으로 지정.



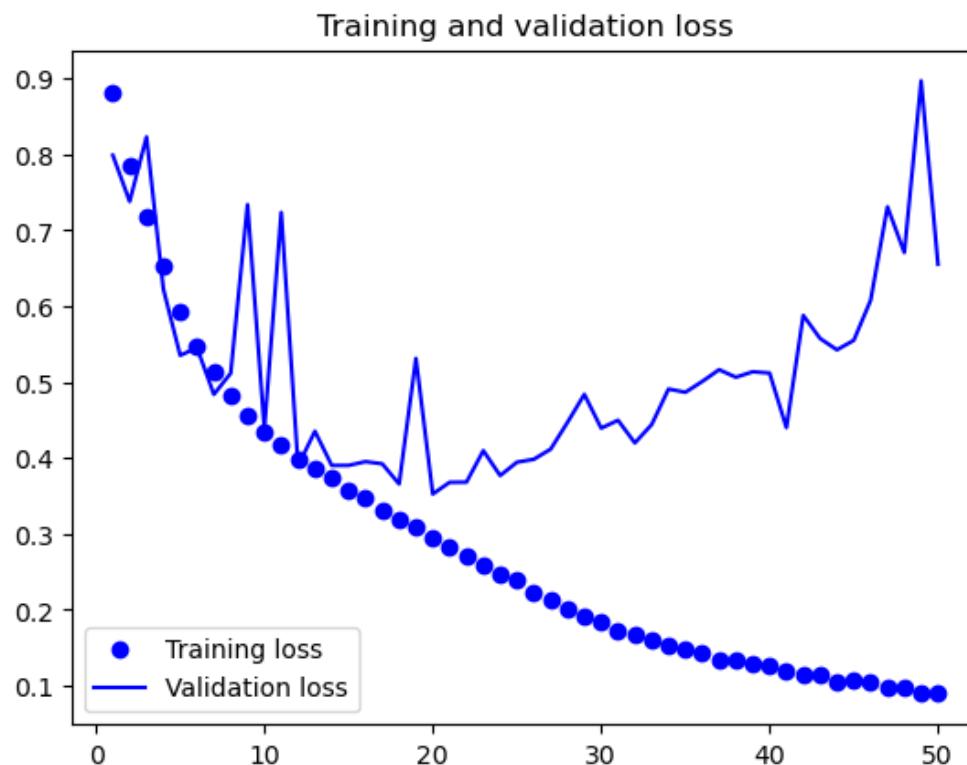
모델 컴파일과 훈련

```
model.compile(optimizer="rmsprop",
              loss="sparse_categorical_crossentropy")

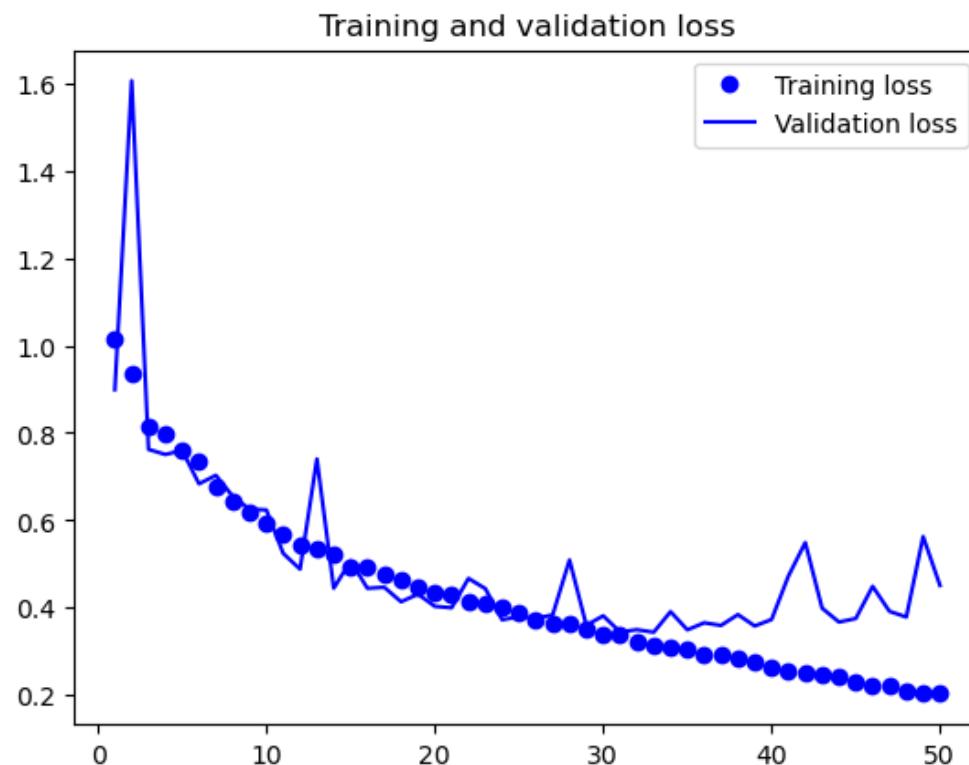
callbacks = [
    keras.callbacks.ModelCheckpoint("oxford_segmentation",
                                    save_best_only=True)
]

history = model.fit(train_input_imgs, train_targets,
                     epochs=50,
                     callbacks=callbacks,
#                      batch_size=64, # 고성능 GPU 활용
                     batch_size=16, # 저성능 GPU 활용
                     validation_data=(val_input_imgs, val_targets))
```

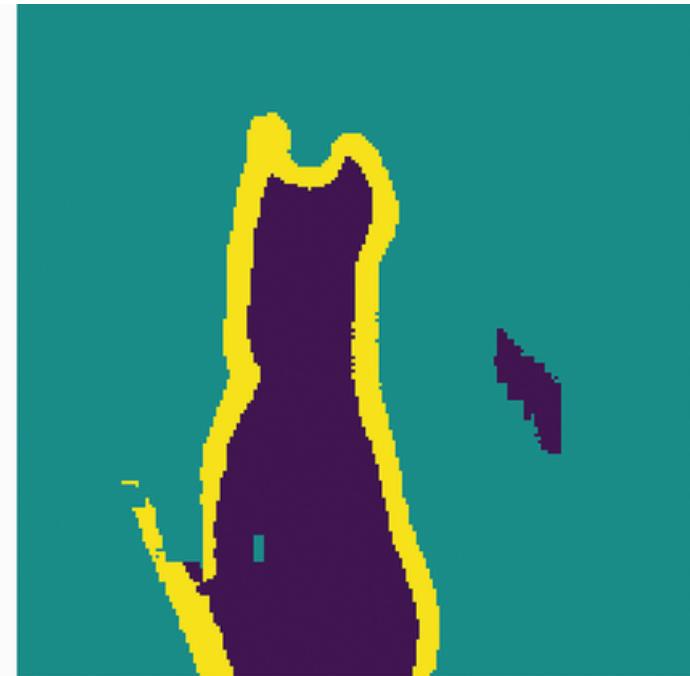
훈련 결과: batch_size = 16



훈련 결과: batch_size = 64



모델 예측

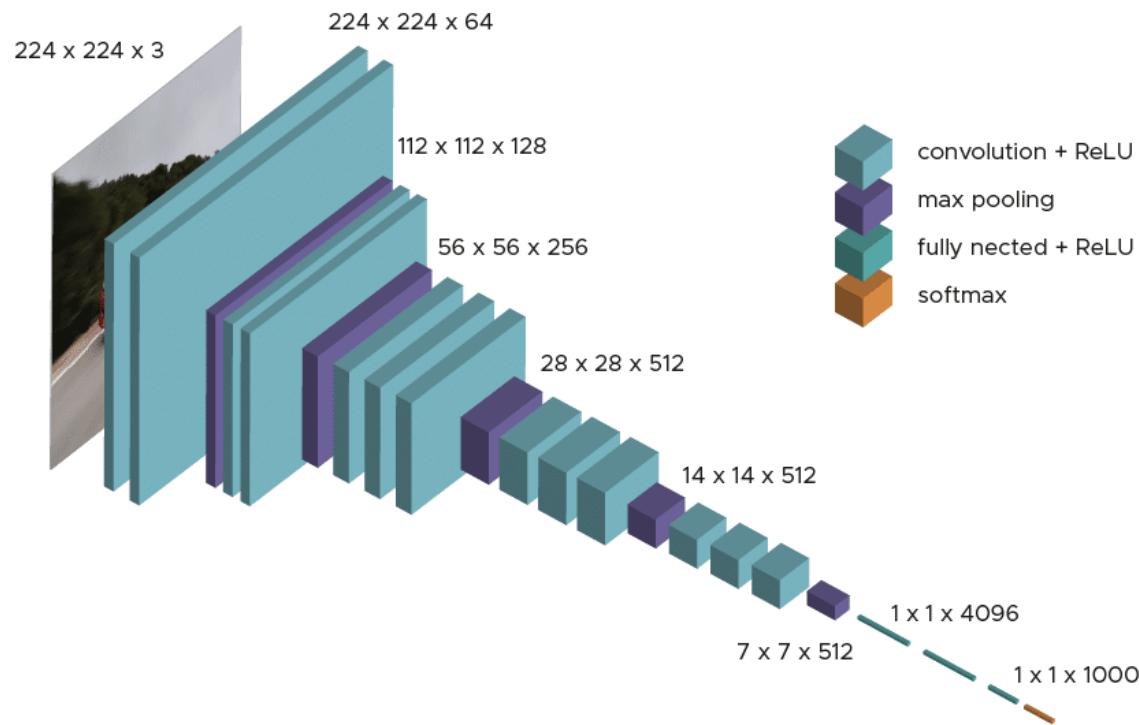


CNN 아키텍처 주요 유형

- 모델 아키텍처: 모델 설계 방식
- 심층 신경망 모델을 구성할 때 매우 중요
- 신경망 모델은 주어진 문제와 데이터셋에 따라 적절한 층을 적절하게 구성해야 함
- 적절한 모델 아키텍처를 사용할 수록 적은 양의 데이터로 보다 빠르게 좋은 성능의 모델을 얻을 가능성이 높아짐
- 하지만 좋은 모델 아키텍처와 관련되어 정해진 이론은 없음.
- 대신 많은 경험을 통한 직관이 모델 구성에 보다 중요한 역할 수행

신경망 모델의 아키텍처: 모듈(블록), 계층, 재활용

- 모듈(블록)을 재활용하여 쌓아 올린 계층 구조
- 대부분의 합성곱 신경망 모델은 **특성 피라미드** 형식의 계층적 구조 사용

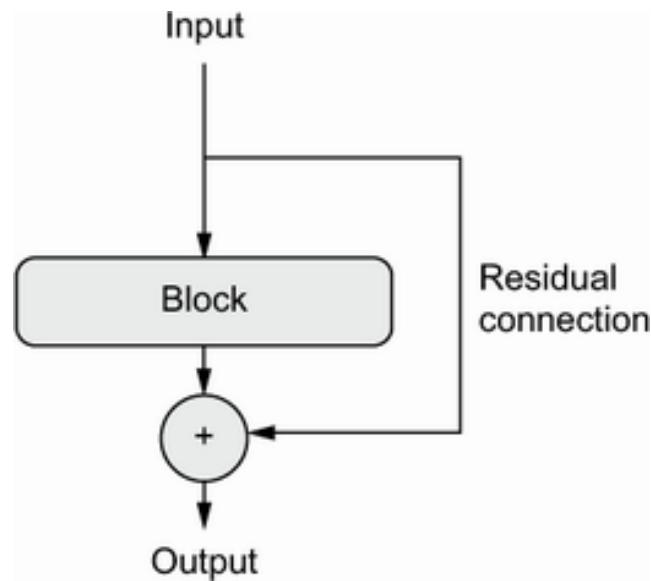


합성곱 신경망 주요 아키텍처 유형

- 블록을 연결하는 계층을 높게 쌓으면 그레이디언트 소실 문제 등으로 인해 모델의 훈련이 잘 이뤄지지 않음
- 이를 해결하는 다양한 해결책이 개발되었음.
- 여기서는 합성곱 신경망 구성에 사용되는 세 개의 주요 아키텍처 유형 소개
 - 잔차 연결 residual connections
 - 배치 정규화 batch normalization
 - 채널 분리 합성곱 depthwise separable convolutions

잔차 연결

- 그레이디언트 소실 문제: 층을 높이 쌓았을 때 역전파 과정에서 전달되어야 하는 손실값(오차)의 소실되는 문제
- 잔차 연결로 해결 가능
- 잔차 연결 아키텍처: 2015년 ILSVRC 이미지 분류 경진대회의 1등 모델에 사용됨



잔차 연결 핵심: 모양 맞추기

- 맥스풀링을 사용하지 않는 경우

```
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Conv2D(32, 3, activation="relu")(inputs)
residual = x
x = layers.Conv2D(64, 3, activation="relu", padding="same")(x) # padding 사용
residual = layers.Conv2D(64, 1)(residual) # 필터 수 맞추기
x = layers.add([x, residual])
```

- 맥스풀링을 사용하는 경우

```
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Conv2D(32, 3, activation="relu")(inputs)
residual = x
x = layers.Conv2D(64, 3, activation="relu", padding="same")(x)
x = layers.MaxPooling2D(2, padding="same")(x)                      # 맥스풀링
residual = layers.Conv2D(64, 1, strides=2)(residual)                 # 보폭 사용
x = layers.add([x, residual])
```

```
inputs = keras.Input(shape=(32, 32, 3))
x = layers.Rescaling(1./255)(inputs)

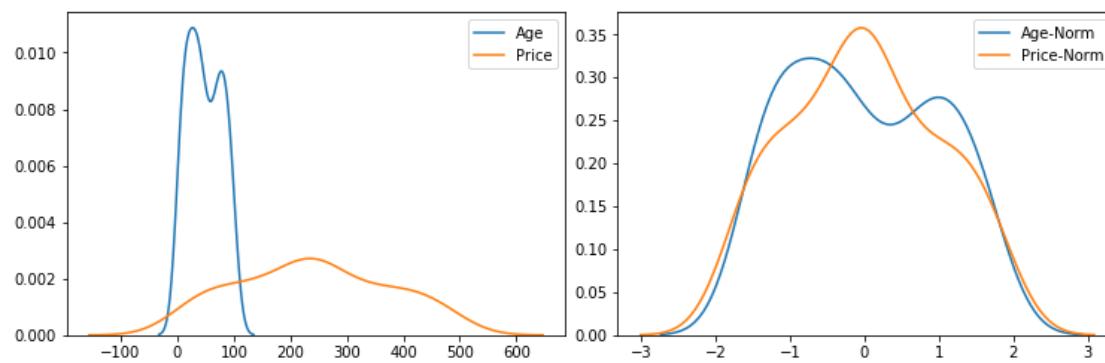
def residual_block(x, filters, pooling=False):
    residual = x
    x = layers.Conv2D(filters, 3, activation="relu", padding="same")(x)
    x = layers.Conv2D(filters, 3, activation="relu", padding="same")(x)
    if pooling:                      # 맥스풀링 사용하는 경우
        x = layers.MaxPooling2D(2, padding="same")(x)
        residual = layers.Conv2D(filters, 1, strides=2)(residual)
    elif filters != residual.shape[-1]: # 필터 수가 변하는 경우
        residual = layers.Conv2D(filters, 1)(residual)
    x = layers.add([x, residual])
    return x

x = residual_block(x, filters=32, pooling=True)
x = residual_block(x, filters=64, pooling=True)
x = residual_block(x, filters=128, pooling=False)
x = layers.GlobalAveragePooling2D()(x) # 채널 별로 하나의 값(채널 평균값) 선택

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

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

정규화

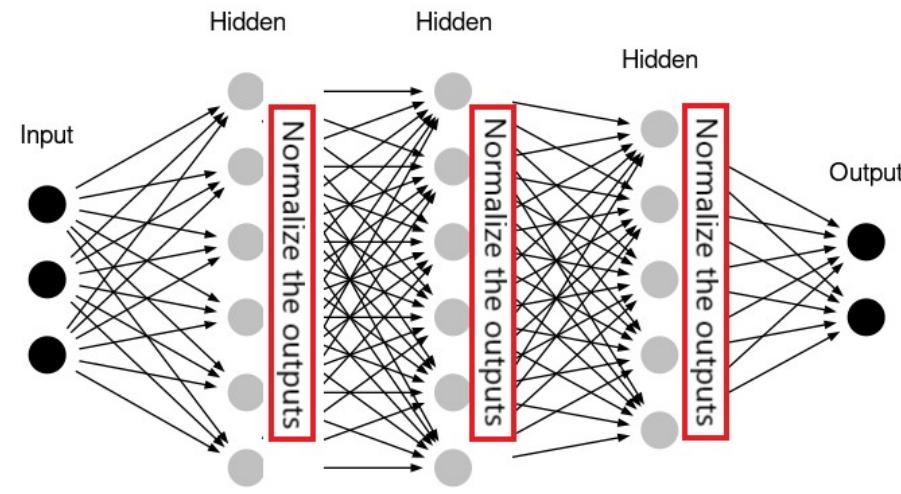


배치 정규화

- 배치 정규화: 2015년에 발표된 한 논문에서 소개됨
- 다음 층으로 넘겨주기 전에 정규화 진행

```
normalized_batch = (batch - np.mean(batch, axis=...)) / np.std(batch,  
axis=...)
```

- 케라스의 `layers.BatchNormalization` 층이 배치 정규화를 지원
- ResNet50, EfficientNet, Xception 모델 등은 배치 정규화 없이는 제대로 훈련되지 않음



배치 정규화 사용법

```
x = ...
x = layers.Conv2D(32, 3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = ...
```

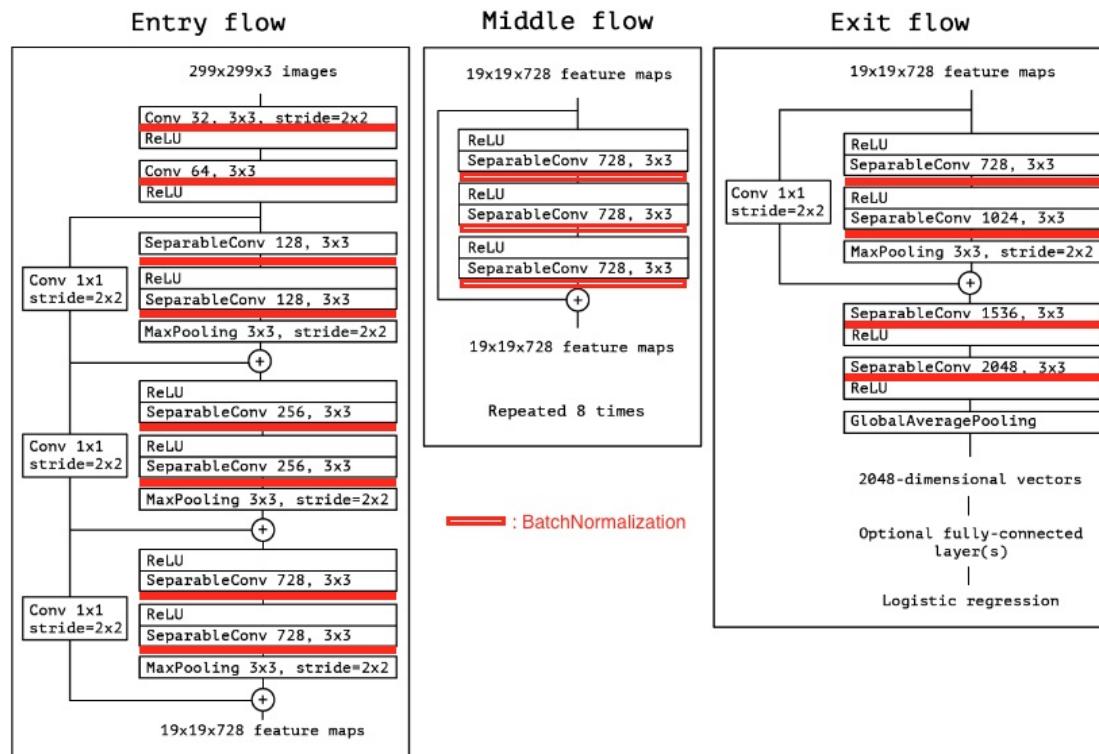
배치 정규화 작동 원리

- 정규화는 특성맵 단위로 이뤄지며 정규화에 사용되는 평균값과 분산은 배치 단위로 계산됨.
- `use_bias=False` 옵션: 배치 정규화에 의해 어차피 데이터의 평균값을 0으로 만들기에 굳이 편향 파라미터가 필요 없음
- 활성화 함수 사용 위치: 배치 정규화 이후 실행 권장.

모델 미세조정과 배치 정규화

- 배치 정규화 층이 포함된 모델을 미세조정할 때 배치 정규화 층도 함께 동결(freeze) 할 것 권장됨.
- 배치 정규화 층을 동결하면 재활용되는 모델이 훈련될 때 계산한 데이터셋의 평균값과 분산이 대신 활용됨.
- 미세조정의 경우 훈련셋의 데이터가 모델이 기존에 훈련했을 때의 훈련셋과 비슷하다고 전제함.

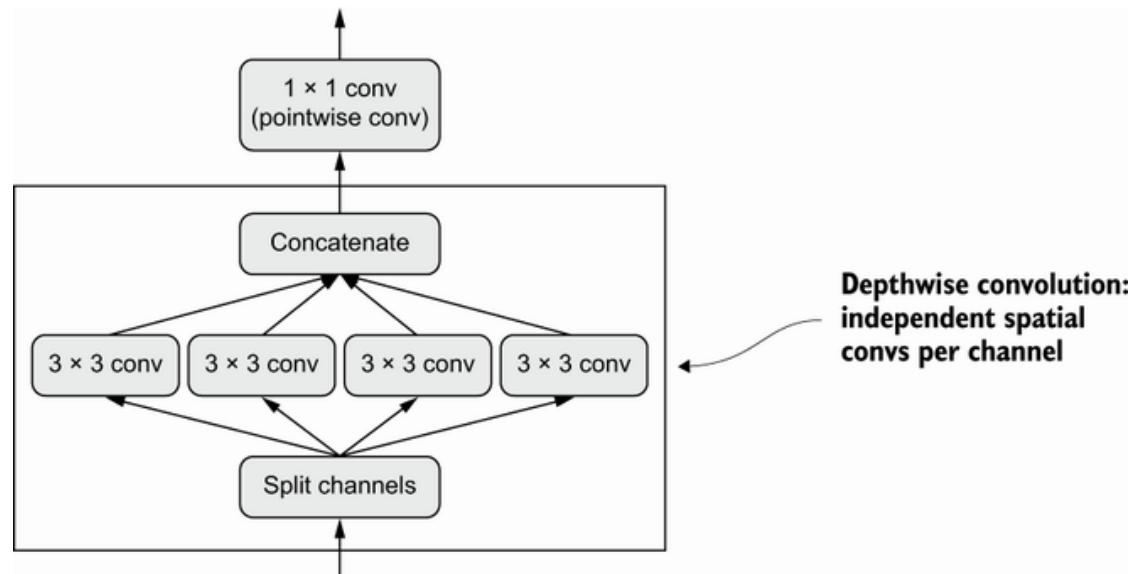
Xception 모델 (2017)



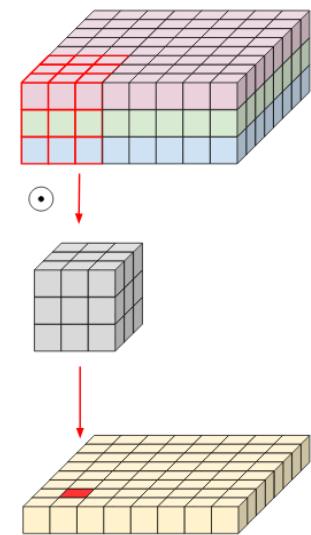
채널 분리 합성곱

- 케라스의 `SeparableConv2D` 층은 `Conv2D` 층보다 적은 수의 가중치 파라미터를 사용하지만 성능이 좀 더 좋은 모델 생성
- 채널 분리 합성곱_{depthwise separable convolution}라 불림
- 2017년 Xception 모델 논문에서 소개됨

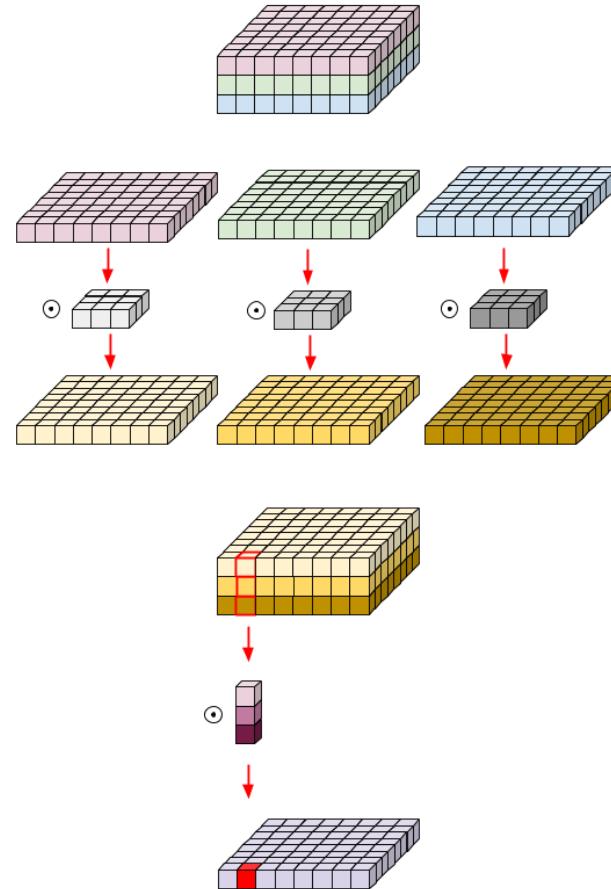
SeparableConv2D 작동법



Conv2D 작동 원리



SeparableConv2D 작동 원리



층의 파라미터 개수 비교

- 경우 1: 3x3 모양의 필터 64개를 3개의 채널을 갖는 입력 데이터에 사용할 경우
 - Conv2D 의 경우: $3 \times 3 \times 3 \times 64 = 1,728$
 - SeparableConv2D 의 경우: $3 \times 3 \times 3 + 3 \times 64 = 219$
- 경우 2: 3x3 모양의 필터 64개를 10개의 채널을 갖는 입력 데이터에 사용할 경우
 - Conv2D 의 경우: $3 \times 3 \times 10 \times 64 = 5,760$
 - SeparableConv2D 의 경우: $3 \times 3 \times 10 + 10 \times 64 = 730$

채널 분리 합성곱 층의 약점

- 채널 분리 합성곱의 연산 CUDA에서 제대로 지원되지 않음
- GPU를 사용하더라도 기존 Conv2D 층만을 사용한 모델에 비해 학습 속도에 별 차이가 없음
- 하지만 적은 수의 파라미터를 사용하기에 일반화 성능이 보다 좋은 모델을 구현한다는 점이 매우 중요

CUDA와 cuDNN

- CUDA(Compute Unified Device Architecture)
 - CPU와 GPU를 동시에 활용하는 병렬 컴퓨팅을 지원하는 플랫폼
 - C, C++, Fortran 등의 저수준 언어 활용
- cuDNN(CUDA Deep Neural Network): CUDA를 활용하여 딥러닝 알고리즘의 실행을 지원하는 라이브러리
 - Conv2D 등 특정 딥러닝 알고리즘에 대해서만 최적화됨.

예제: 미니 Xception 모델

- 미니 Xception 모델을 직접 구현하여 강아지-고양이 이항분류 작업 실행
- 모델 구현에 사용되는 기법
 - 모듈을 쌓을 수록 필터 수는 증가시키고, 텐서 크기는 작게.
 - 층의 유닛은 적게, 모듈은 높게.
 - 잔차 연결을 활용
 - 모든 합성곱 층 이후에는 배치 정규화를 적용
 - 채널 분리 합성곱 신경망 활용
- 앞서 언급된 모든 기법은 컴퓨터 비전 프로젝트 일반에 적용될 수 있음.
- [DeepLabV3 모델](#): Xception 모델을 이용하는 2021년 기준 최고의 이미지 분할 모델

모델 구성

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

```
# 입력층
inputs = keras.Input(shape=(180, 180, 3))
x = data_augmentation(inputs)
x = layers.Rescaling(1./255)(x)

# 하나의 Conv2D 은닉층
x = layers.Conv2D(filters=32, kernel_size=5, use_bias=False)(x)
```

```
# SeparableConv2D, BatchNormalization, MaxPooling2D 층으로 구성된 모듈 쌓기
# 잔차 연결 활용
for size in [32, 64, 128, 256, 512]:    # 필터 수
    residual = x

    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)

    x = layers.BatchNormalization()(x)
    x = layers.Activation("relu")(x)
    x = layers.SeparableConv2D(size, 3, padding="same", use_bias=False)(x)

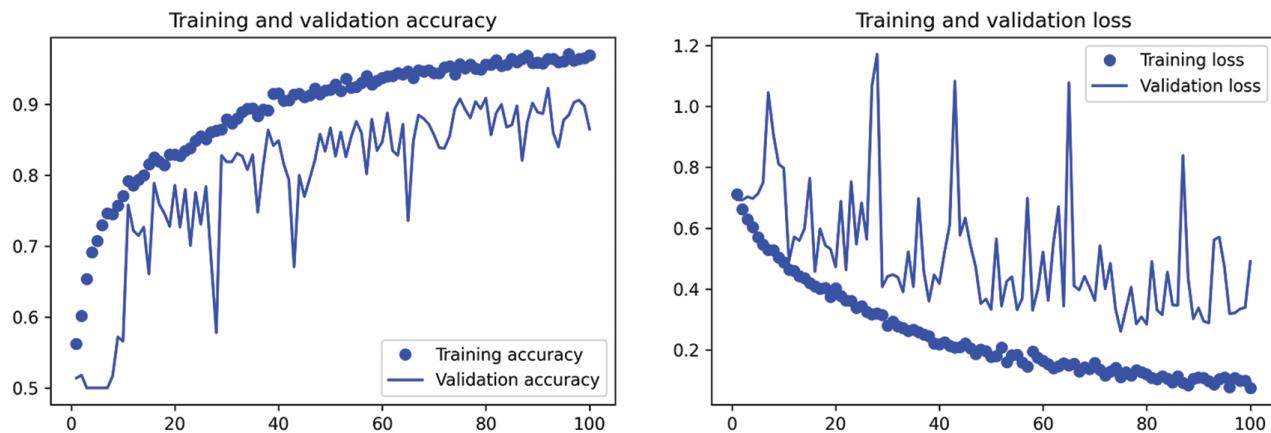
    x = layers.MaxPooling2D(3, strides=2, padding="same")(x)

    # 잔차 연결
    residual = layers.Conv2D(
        size, 1, strides=2, padding="same", use_bias=False)(residual)
    x = layers.add([x, residual])
```

```
# 마지막 은닉층은 GlobalAveragePooling2D과 Dropout
x = layers.GlobalAveragePooling2D()(x)      # flatten 역할 수행(채널 별 평균값으로 구성)
x = layers.Dropout(0.5)(x)

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

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



성능 높이기

- 하이퍼파라미터 조성: 그리드 탐색, 랜덤 탐색
- 양상블 학습