

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

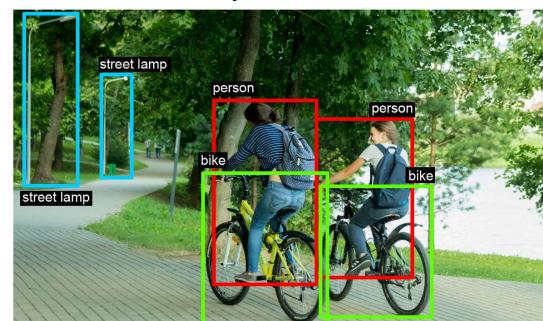


- | | |
|--|--|
| <input checked="" type="checkbox"/> Bike | <input checked="" type="checkbox"/> Tree |
| <input checked="" type="checkbox"/> Person | <input type="checkbox"/> Car |
| <input type="checkbox"/> Boat | <input type="checkbox"/> House |

Image segmentation

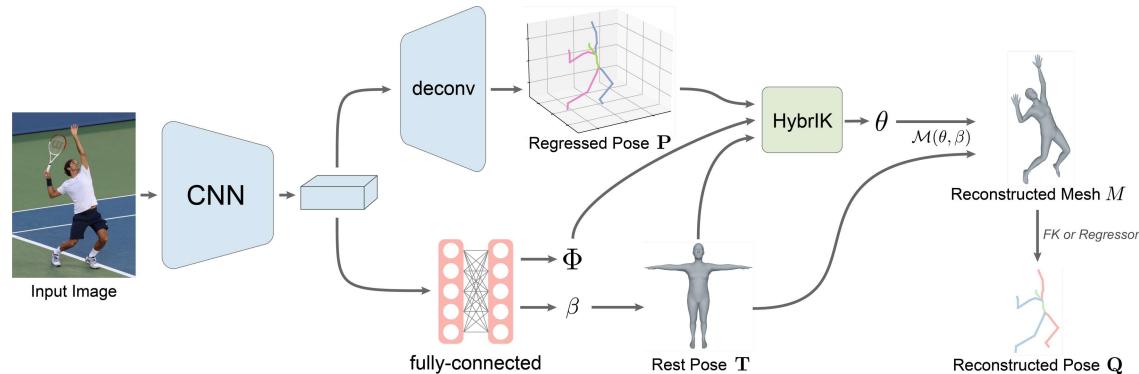


Object detection



기타 다양한 활용법

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

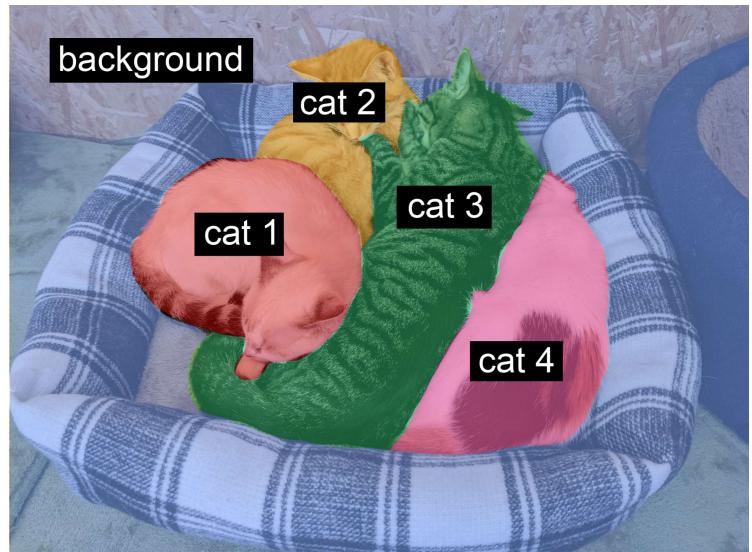


객체 탐지

- 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)

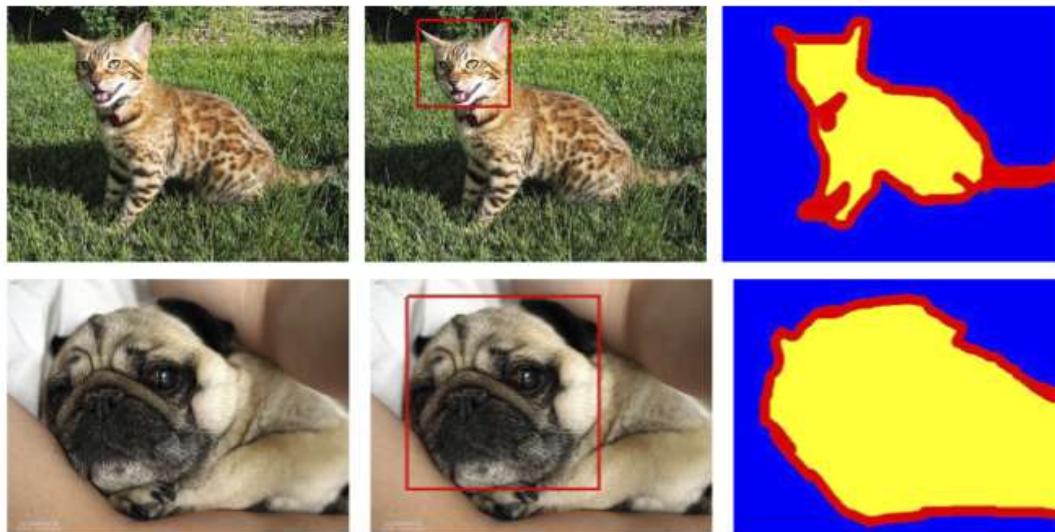


Oxford-IIIT 애완동물 데이터셋

- 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=img_size + (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

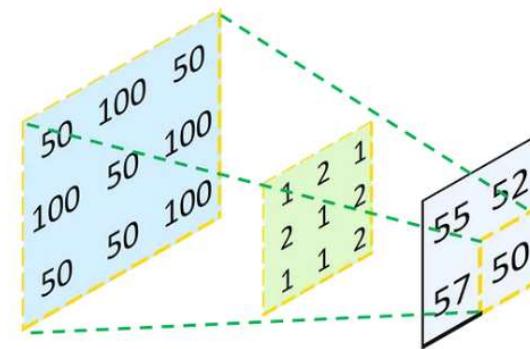
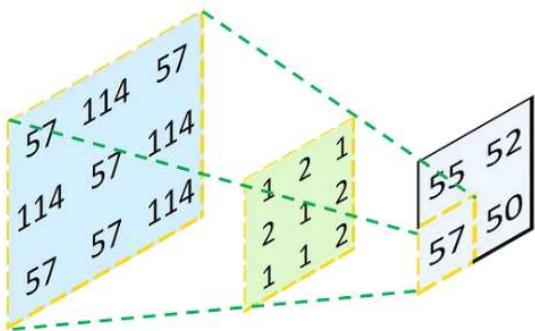
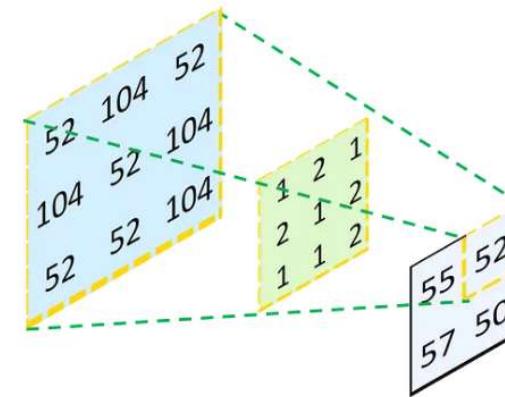
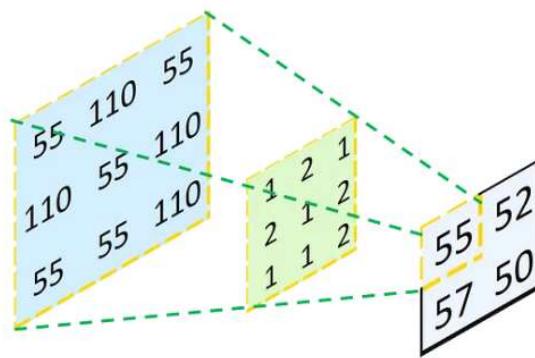
```
x = layers.Conv2DTranspose(256, 3, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(256, 3, activation="relu", padding="same",
strides=2)(x)
x = layers.Conv2DTranspose(128, 3, activation="relu", padding="same")(x)
x = layers.Conv2DTranspose(128, 3, activation="relu", padding="same",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

업샘플링 작동법



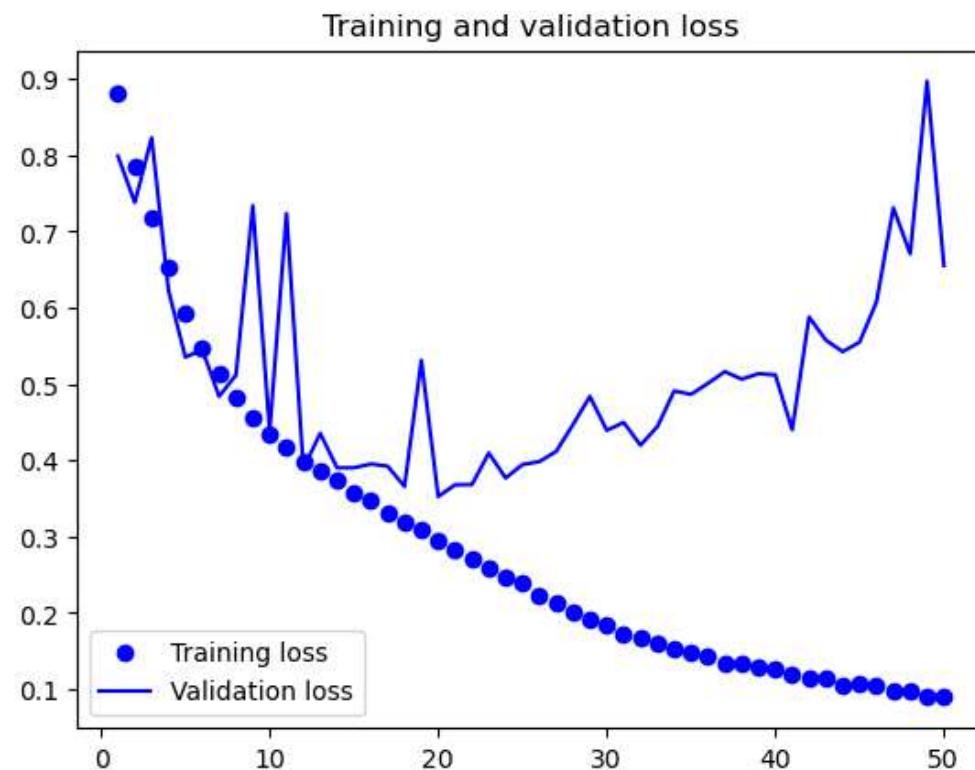
모델 컴파일과 훈련

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

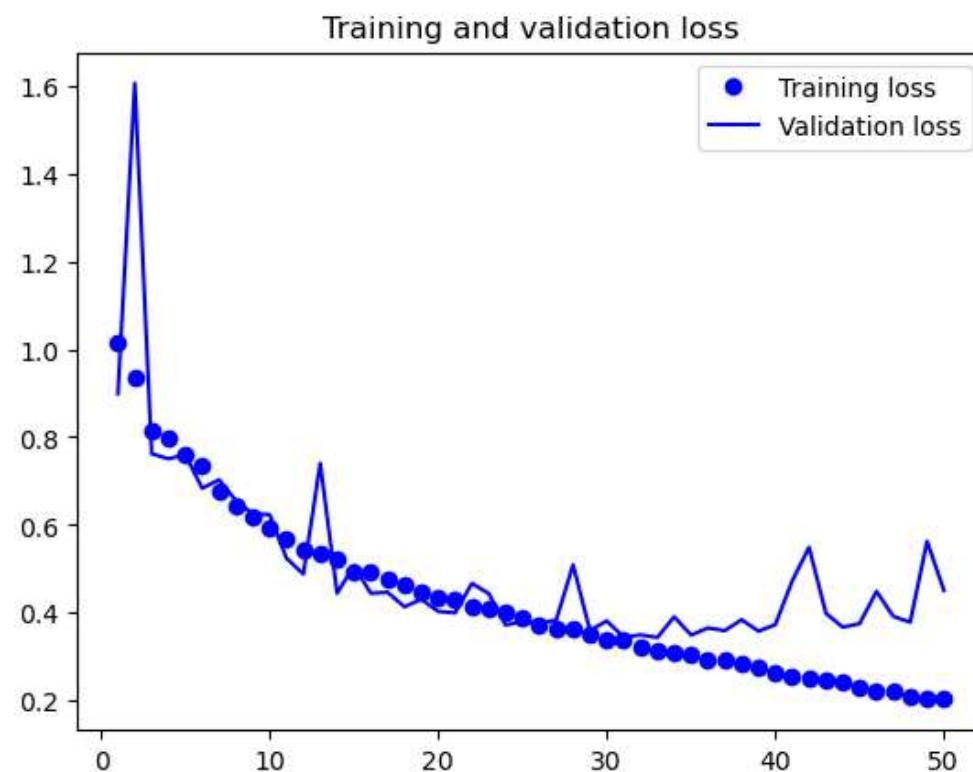
callbacks = [
    keras.callbacks.ModelCheckpoint("oxford_segmentation",
                                    save_best_only=True)
]

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))
```

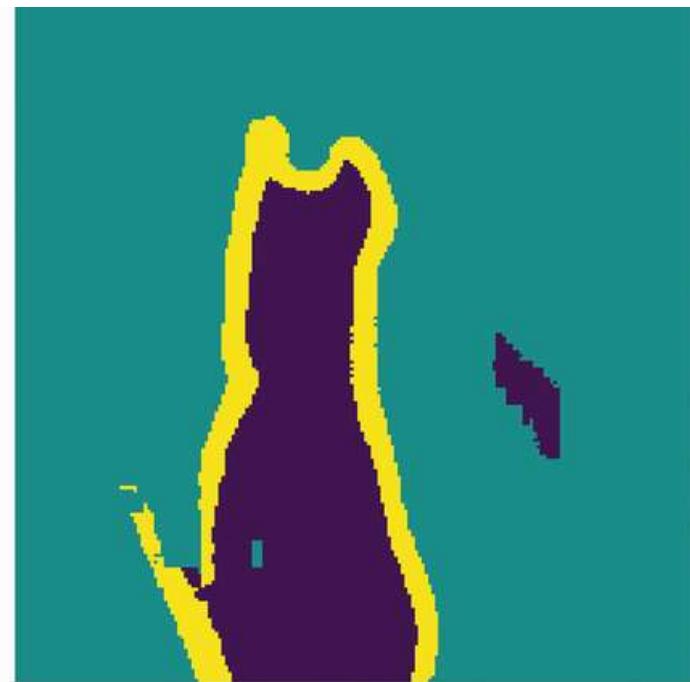
훈련 결과: batch_size = 16



훈련 결과: batch_size = 64



모델 예측

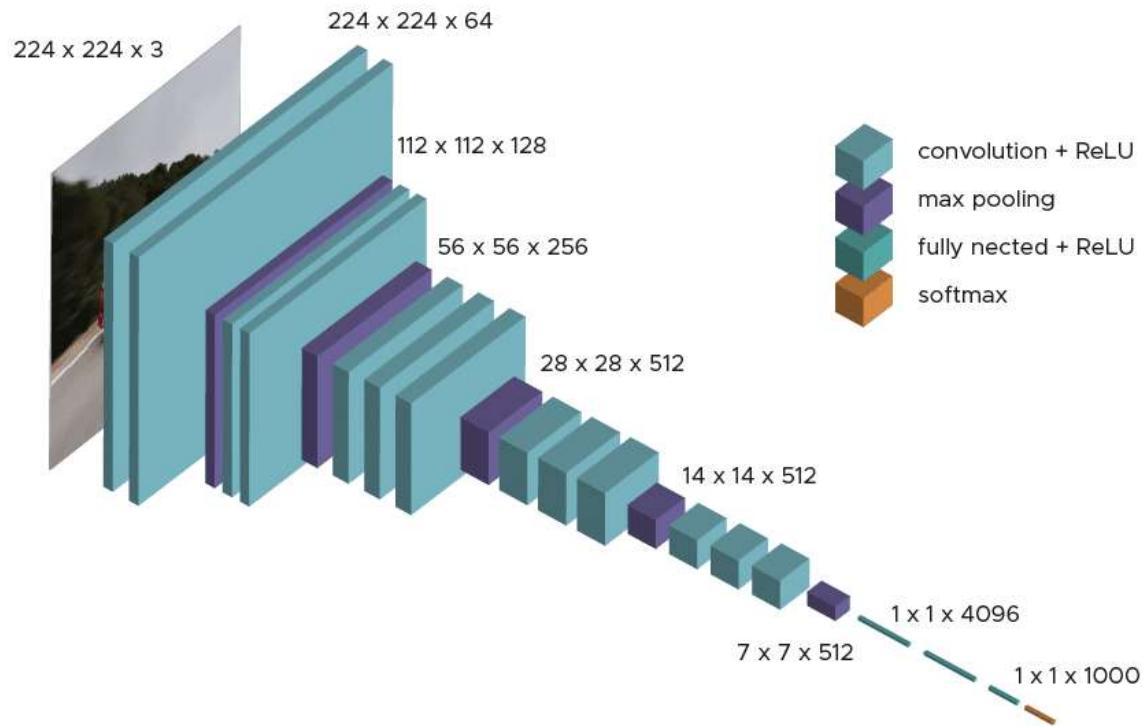


CNN 아키텍처 주요 유형

- 모델 아키텍처: 모델 설계방식 의미
- 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

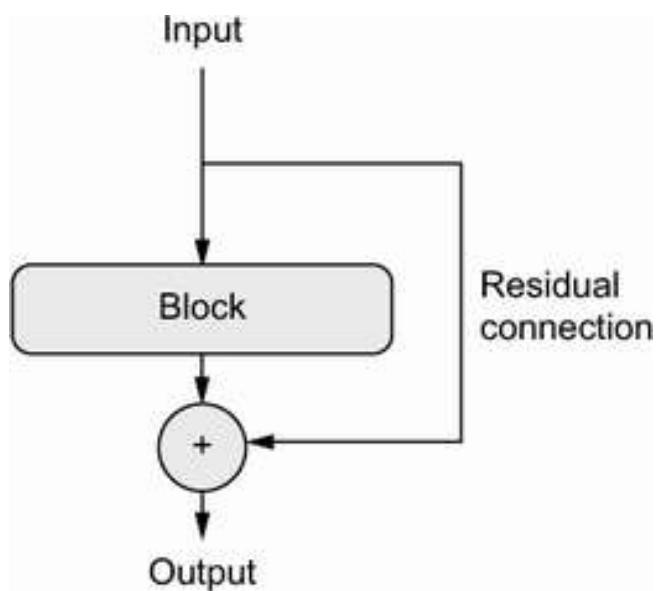


합성곱 신경망 주요 아키텍처 유형

- 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 경진대회에서 1등을 차지한 모델에서 사용



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

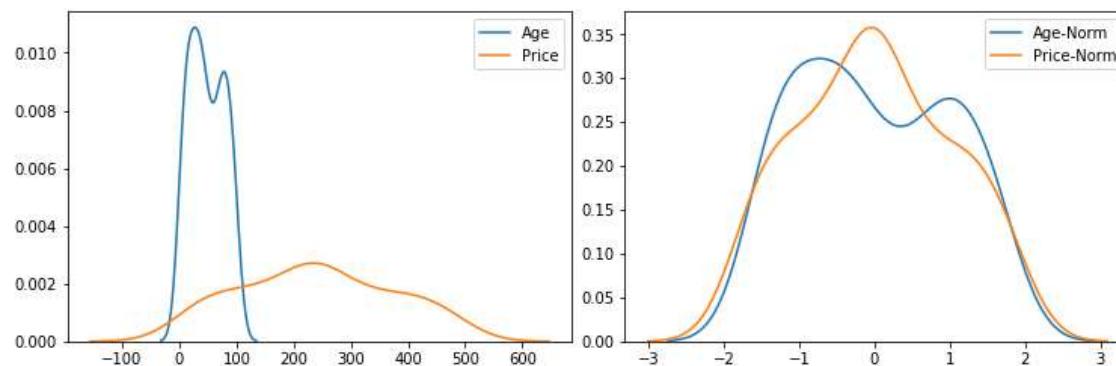
x = residual_block(x, filters=32, pooling=True)
x = residual_block(x, filters=64, pooling=True)
x = residual_block(x, filters=128, pooling=False)
x = layers.GlobalAveragePooling2D()(x) # 채널 별로 하나의 값(채널 평균값) 선택

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

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

배치 정규화

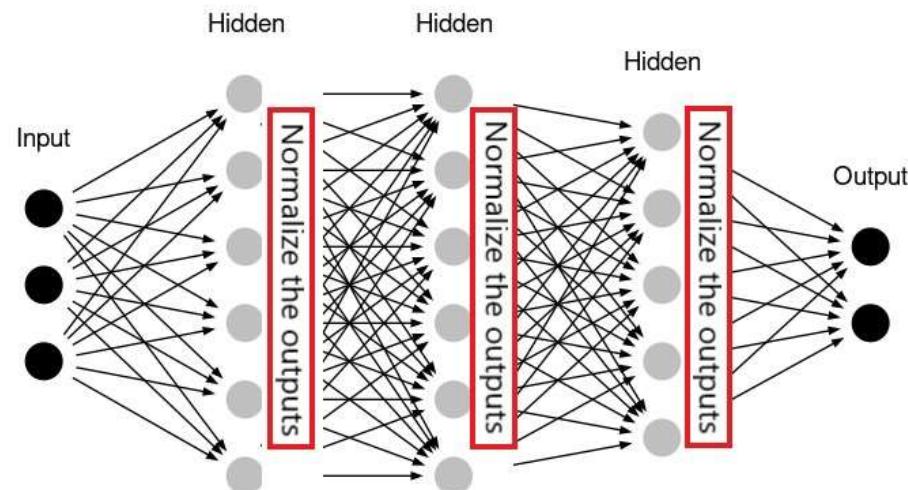
- 정규화



- 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

- BatchNormalization 층은 보통 활성화 함수를 적용하기 전에 사용함.

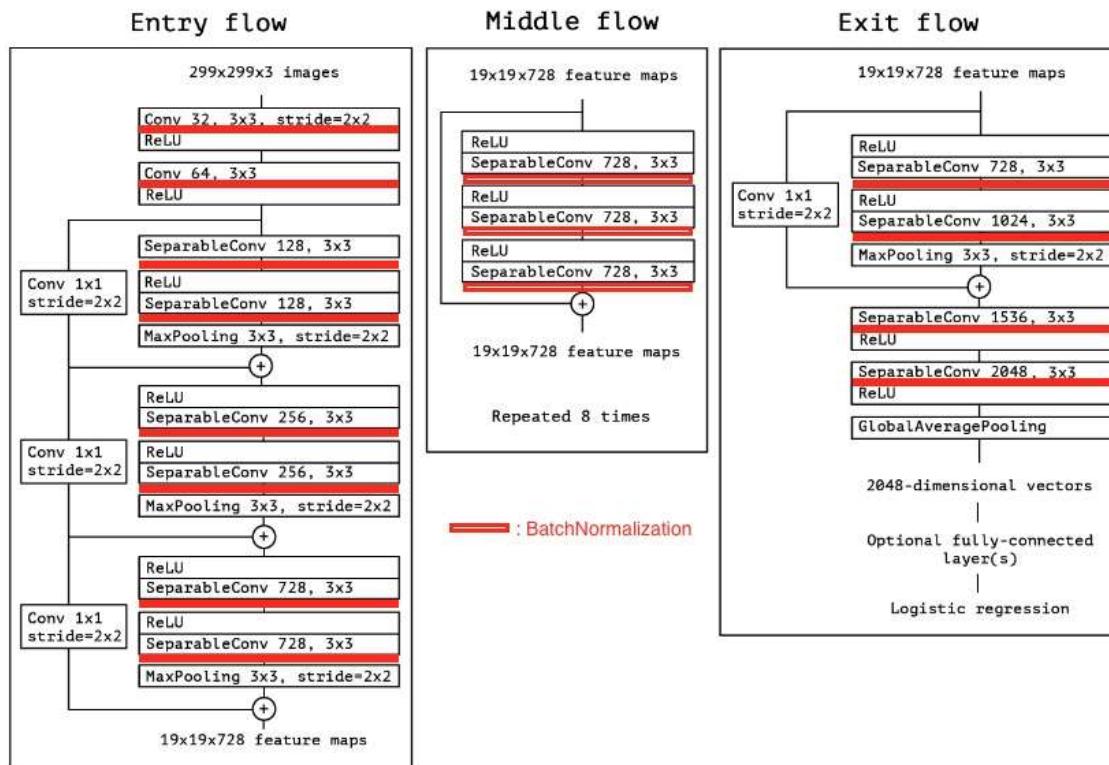
```
x = ...
x = layers.Conv2D(32, 3, use_bias=False)(x)
x = layers.BatchNormalization()(x)
x = layers.Activation("relu")(x)
x = ...
```

- 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 이후에 활성화 함수를 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 평균 값과 분산이 대신 활용됨.
- 미세조정의 경우 훈련셋의 데이터가 모델이 기준에 훈련했을 때의 훈련셋과 비슷 하다고 전제하기 때문임.

Xception 모델 (2017)



채널 분리 합성곱

계속 이어짐...