Week1

CH14 - 합성곱 신경망을 사용한 컴퓨터 비전

이미지 인식 분야에 사용

- 객체 탐지
- 시멘틱 분할

14.1 시각 피질 구조

시각 피질이 **국부 수용장**을 가진다

→뉴런들이 시야의 일부 범위 안에 있는 시각 자극에만 반응한다

- 서로 겹칠 수 있어서 합치면 전체 시야
- 수평선의 이미지에만 반응
- 다른 각도의 선분에 반응(동일한 수용장을 가지고 있지만 다른 각도의 선분에 반응)
- 큰 수용장을 가져가 저수준 패턴이 조합된 더 복잡한 패턴에 반응

⇒ **고수준 뉴런이 이웃한 저수준 뉴런의 출력에 기반한다 ~ 합성곱 신경망(CNN)**

- LeNet-5

14.2 합성곱 층

입력층 → 합성곱 층 1 → 합성곱 층 2

- 입력 → 저수준 특성에 집중 → 고수준 특성으로 조합
- 계층적 구조

- 각 층이 2D로 표현(압력과 연결이 쉬움)
- **제로 패딩** (= 높이와 너비픞 이정 층과 같게 하기 위해 입력의 주위에 0을 추가하는 것)
- **스트라이드** (= 한 수용장과 다음 수용장 사이 간격)

  차원 축소를 통해 모델 복잡성 ↓

14.2.1 필터

뉴런의 가중치 = 수용장 크기의 작은 이미지

- 필터 (= 합성곱 커널)
  1. 가중치를 사용한 대표 수직선 → 입력 이미지 네트워크에 주입
     → 모든 뉴런에 같은 수직선 필터와 편향을 적용 → **수직 필터 이미지 ~ 특성 맵 1**
  2. 가중치를 사용한 대표 수평선 → 입력 이미지 네트워크에 주입
     → 모든 뉴런에 같은 수평선 필터와 편향을 적용 → **수평 필터 이미지 ~ 특성 맵 2**

✔️ **특성 맵**: 필터를 가장 크게 활성화시키는 이미지의 영역 강조 (자동으로 됨)

14.2.2 여러 가지 특성 맵 쌓기

여러 가지 필터를 가지고 하나의 필터마다 하나의 특성 맵 출력

→ 3D로 표현하는 것이 더 정확

- 각 특성 맵의 픽셀 = 하나의 뉴런
- 하나의 특성 맵 - 모든 뉴런이 같은 파라미터 공유(= 동일한 가중치 & 편향)
- 다른 특성 맵 - 다른 파라미터
- 한 뉴런의 수용장은 이전 층에 있는 모든 특성 맵에 걸쳐 확장

  = 합성곱 층이 입력에 여러 필터 적용 → 입력의 여러 특성 감지

입력 이미지

- 컬러 채널 - RGB 세 개의 채널
- 흑백 채널 - 하나의 채널
- 매우 많은 채널 - 가시광선 외 적외선과 같은 다른 빛의 파장(ex. 위성 이미지)

14.2.3 텐서플로 구현

입력 이미지: 3D [높이, 너비, 채널]

미니배치 크기: 4D [미니배치 크기, 높이, 너비, 채널]

```
# 샘플 이미지 로드
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

# 필터 2개 생성
filters = np.zeros(shape=(7, 7, channels, 2), dtype=np.float32)
filters[:, 3, :, 0] = 1  # 수직선
filters[3, :, :, 1] = 1  # 수평선

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

plt.imshow(outputs[0, :, :, 1], cmap="gray") # 첫 번째 이미지의 두 번째 특성맵 그리기
plt.show()
```

- padding의 옵션은 두 가지
  - VALID: 제로 패딩을 사용하지 않아 입력 이미지의 아래와 오른쪽 행과 열이 무시될 수 있음

    출력 뉴런의 크기 = 입력 뉴런의 수를 스트라이드로 나누고 나머지는 버림

- SAME: 제로 패딩 사용 (출력 뉴런의 크기 = 입력 뉴런의 수를 스트라이드로 나누어올림 한 것)

```
conv = keras.layers.Conv2D(filters=2, kernel_size=7, strides=1,
                           padding="SAME", activation="relu")
```

14.2.4 메모리 요구 사항

많은 양의 RAM을 필요로 함

메모리 부족으로 훈련에 실패한다면,

- 미니배치 크기를 줄이거나
- 스트라이드를 이용해 차원을 줄이거나 몇 개의 층 제거
- 32비트 대신 16비트 부동소수 사용
- 여러 장치에 CNN 분산

14.3 풀링 층

✅ 계산량, 메모리 사용량, 파라미터 수를 줄이기 위해 입력 이미지의 부표본(=축소본)을 만드는 것

합성곱 층과 비슷하지만 풀링 뉴런은 가중치가 없음

→ 합산 함수를 사용해 입력 값을 더하는 것이 전부

최대 풀링 층: 각 수용장의 가장 큰 입력값이 전달되고 나머지는 버려짐

<장점>

- 이동에 대한 불변성
- 회전, 확대, 축소에 대한 불변성

→ 예측이 작은 부분에서 영향을 받지 않는 분류 작업에 유용

<단점>

- 파괴적
- 불변성이 필요하지 않은 경우(ex. 시맨틱 분할 → **등변성**이 목표)

14.3.1 텐서플로 구현

```
max_pool = keras.layers.MaxPool2D(pool_size=2) # valid padding 사용
```

평균 풀링 층 - `AvgPool2D` : 최댓값이 아닌 평균을 계산하는 것

→ 최댓값을 계산하는 것보다 정보 손실이 적음

→ 최대 풀링 층이 더 좋은 성능을 보임

의미 없는 것은 모두 제거하고 큰 특징만 유지 ~ 명확한 신호로 작업 가능

연산 비용이 더 적음 (더 뛰어난 이동 불변성)

깊이 방향 풀링 층: 회전에 상관없이 동일한 출력을 만드는 것

→ 두께, 밝기, 왜곡, 색상 등 어떤 것에 대해서고 불변성 학습 가능

→ 케라스에서 층을 제공 X

전역 평균 풀링 층 - `GlobalAvgPool2D`: 각 특성 맵의 평균을 계산

→ 각 샘플의 특성 맵마다 하나의 숫자를 출력

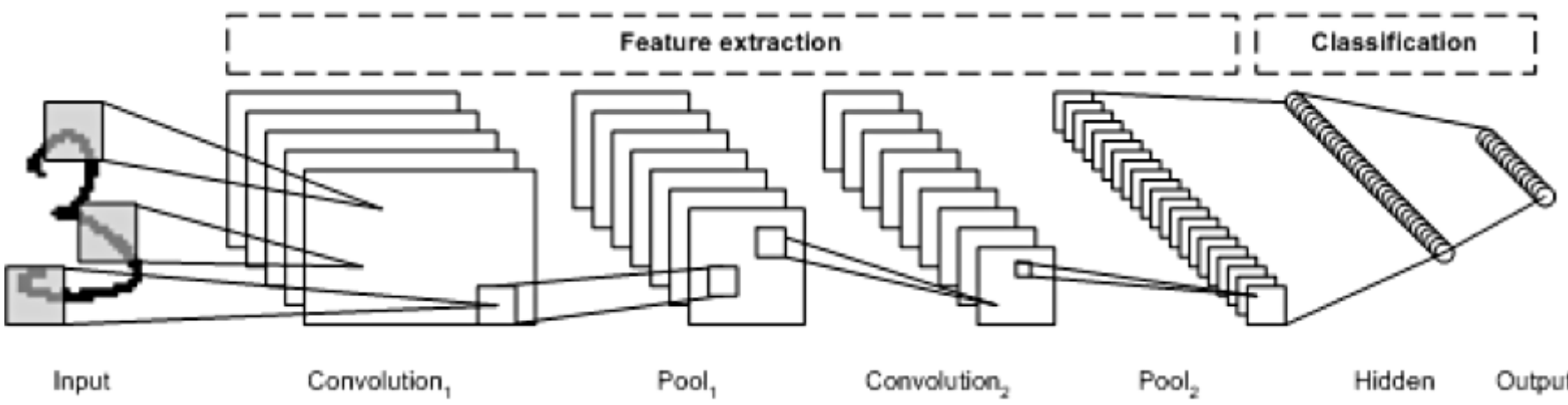
파괴적이나 출력층에 유용

```
>>> global_avg_pool = keras.layers.GlobalAvgPool2D()
>>> global_avg_pool(cropped_images)
<tf.Tensor: shape=(2, 3), dtype=float32, numpy=
array([[0.2788777 , 0.22507192, 0.20967275],
       [0.51288515, 0.45951638, 0.33423486]], dtype=float32)>

#위 코드와 동일한 결과 값

output_global_avg2 = keras.layers.Lambda(lambda X: tf.reduce_mean(X, axis=[1, 2]))
global_avg_pool(cropped_images)
```

14.4 CNN 구조



MNIST 데이터셋 문제

```
# 92%의 성능

model = keras.models.Sequential([
    keras.layers.Conv2D(filters=64, kernel_size=7, activation='relu', padding="SAME", input_shape=[28, 28, 1]),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Conv2D(128, 3, activation='relu', padding="SAME"),
    keras.layers.Conv2D(128, 3, activation='relu', padding="SAME"),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Conv2D(256, 3, activation='relu', padding="SAME"),
    keras.layers.Conv2D(256, 3, activation='relu', padding="SAME"),
    keras.layers.MaxPooling2D(pool_size=2),
    keras.layers.Flatten(),      #1D 배열 기대 - 펼치기
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dropout(0.5),   #과대적합 감소
    keras.layers.Dense(units=64, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(units=10, activation='softmax'),
])
```

14.4.1 LeNet-5

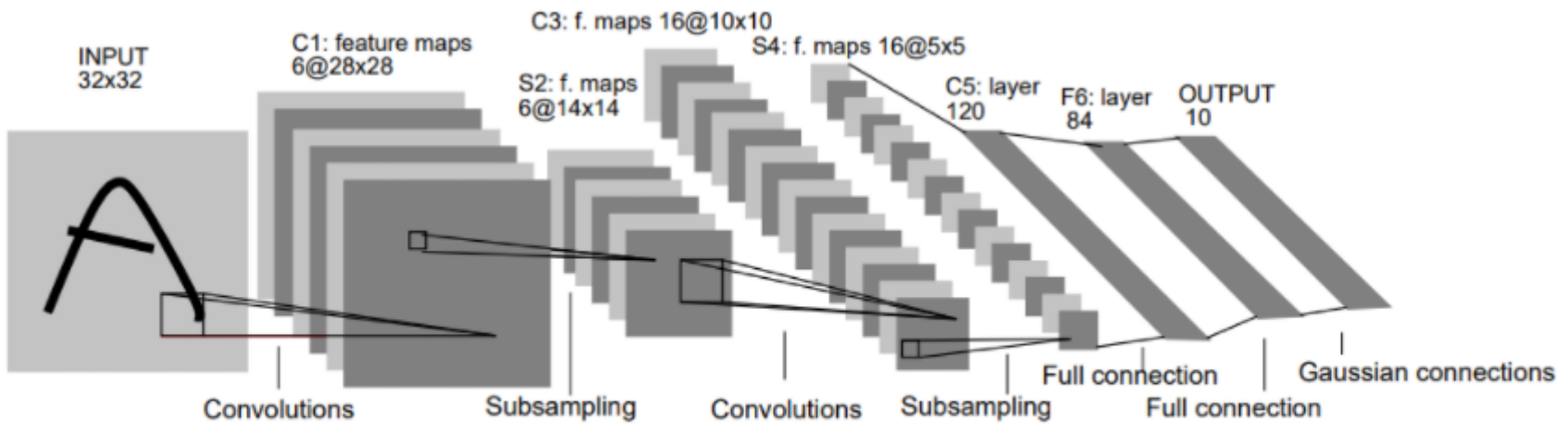
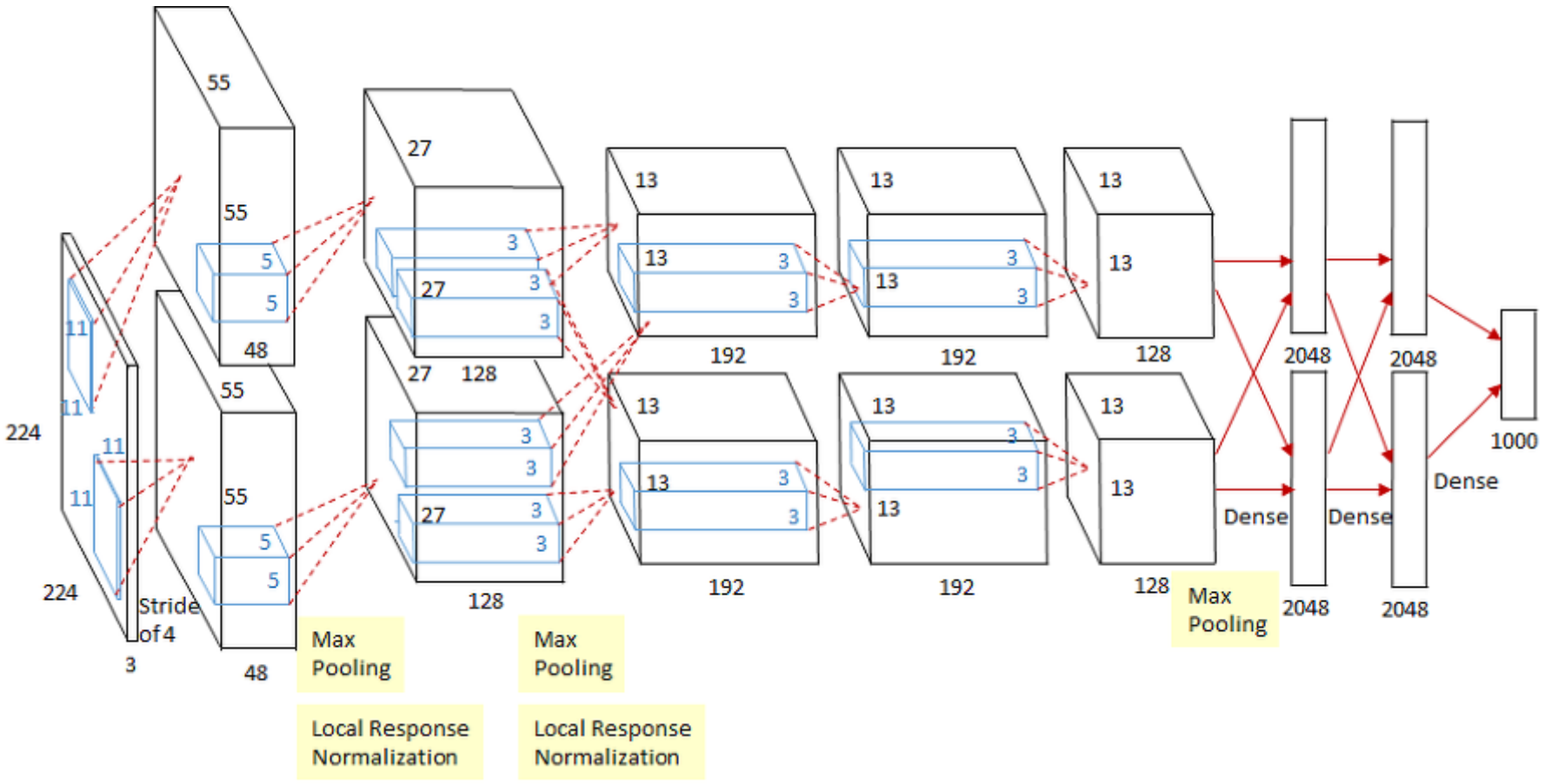


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

LeNet-5 구조

14.4.2 AlexNet

처음으로 합성곱 층 위에 풀링 층을 쌓지 않고 바로 합성곱 층끼리 쌓음



과대적합을 줄이기 위해

- 첫 번째 훈련 동안 F9, F10에 드롭아웃 50% 적용
- 데이터 증식 수행

  훈련 이미지를 랜덤하게 여러 간격으로 이동하거나 수평으로 뒤집고 조명을 바꾸는 식

  진짜 같은 훈련 샘플을 인공적으로 생성

경쟁적 정규화 단계 사용: Local Response Normalization

가장 강하게 활성화 된 뉴런이 다른 특성 맵에 있는 같은 위치의 뉴런 억제

- 특성 맵을 각기 다른 것과 구분
- 더 넓은 시각에서 특징 탐색

→ **일반화 성능 향상**

14.4.3 GoogLeNet

인셉션 모듈 - 효과적으로 파라미터 사용

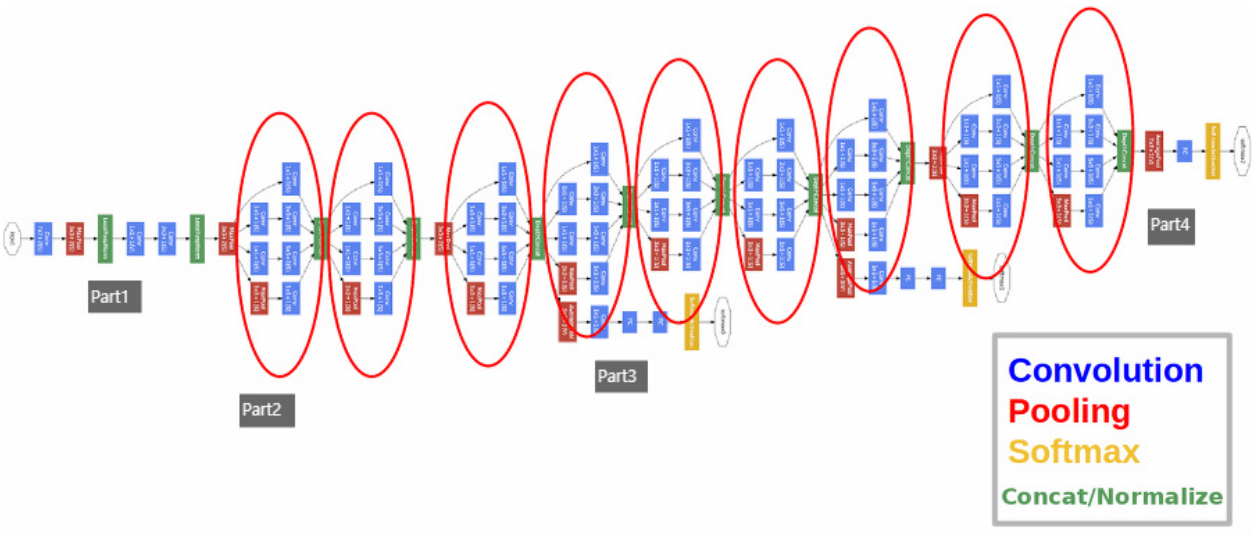
인셉션 모듈이 1 x 1 커널 합성곱 층을 가지는 이유

- 깊이 차원을 따라 놓인 패턴을 잡을 수 있음
- 병목 층의 역학 담당(입력보다 더 적은 특성 맵 출력 - 차원 축소 의미)

  연산 비용, 파라미터 개수 ↓ 훈련 속도, 성능 ↑
- 합성곱 층의 쌍이 한 개의 강력한 합성곱 층처럼 작동

  더 복잡한 패턴 감지 가능

  두 개의 층을 가진 신경망으로 이미지를 훑는 것



Inception-v1

14.4.4 VGGNet

단순, 고전적 구조

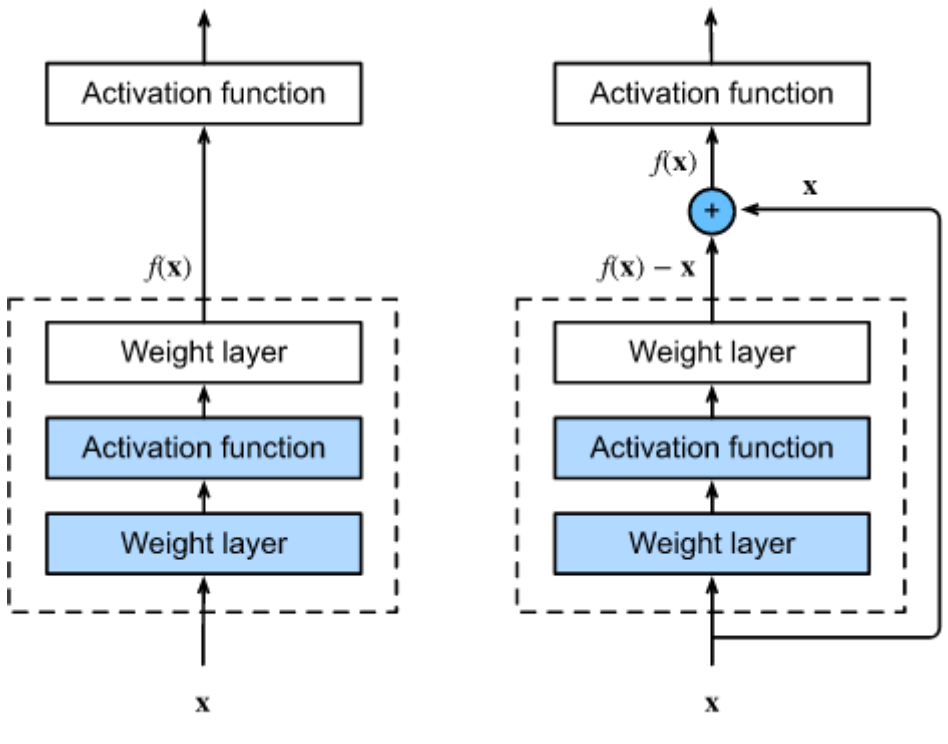
합성곱 층 → 풀링 층 → 합성곱 층 → 풀링 층

14.4.5 ReNet

잔차 네트워크

극도로 깊은 CNN 구조 - 더 적은 파라미터로 더 깊은 네트워크 모델 구성을 트렌드로...

스킵 연결(= 숏컷 연결): 어떤 층에 주입되는 신호가 상위 층의 출력에도 더해지는 것



왼쪽) h(x)를 모델링 // 오른쪽) 스킵 연결 추가로 f(x) = h(x) - x 학습

잔차 유닛: 스킵 연결을 가진 작은 신경망

단순한 구조(드롭 아웃층 제외)

GoogLeNet과 똑같이 시작해 똑같이 종료함 (다만, 단순한 잔차 유닛을 중간 중간 깊게 쌓음)

- ReNet-34
- ReNet-152

14.4.6 Xception

Inception-v4 GoogLeNet 과 ReNet의 아이디어를 합친 것

인셉션 모듈은 **깊이별 분리 합성곱 층**으로 대체

공간상의 패턴: ex. 타원 형태

채널 사이의 패턴: ex. 눈+코+입 = 얼굴

보통 합성곱 층은 공간 패턴과 채널 사이의 패턴을 동시에 잡기 위해 필터 사용

→ **분리 합성곱 층**은 공간 패턴과 채널 사이의 패턴을 분리하여 모델링 할 수 있다고 가정

14.4.7 SENet

- 인셉션 네트워크 확장 버전: SE - Inception
- ResNet 확장한 버전: SE - ResNet

원래 구조에 있는 모든 유닛에 **SE 블록** 추가

- 깊이 차원에서 분석
- 특성 맵을 보정(입, 코 관찰 → 눈 기대)
- 3개 층으로 구성(전역 평균 풀링 층 - ReLU를 사용하는 밀집 은닉층 - 시그모이드를 사용하는 밀집 출력층)

14.5 케라스를 사용해 ReNet-34 CNN 구현하기

```
#특성맵의 크기와 깊이가 바뀔 때 스킵 연결 함수 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()]

#두 출력을 더하여 활성화 함수 적용

    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)
```

```
#모델 생성

model = keras.models.Sequential()
model.add(keras.layers.Conv2D(64, kernel_size=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"))
```

14.6 케라스에서 제공하는 사전훈련된 모델 사용하기

`keras.applications` 에서 제공하는 사전훈련된 모델을 사용한다

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

#이미지 크기 변경
images_resized = tf.image.resize(images, [224, 224])
.
.
.
#모델 전처리
inputs = keras.applications.resnet50.preprocess_input(images_resized * 255)

#예측
Y_proba = model.predict(inputs)

#결과 출력
>>> 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()

Image #0
  n03877845 - palace       43.39%
  n02825657 - bell_cote    43.07%
  n03781244 - monastery    11.70%

Image #1
  n04522168 - vase         53.96%
  n07930864 - cup          9.52%
  n11939491 - daisy        4.97%

#성능이 꽤 괜찮음
```

14.7 사전훈련된 모델을 사용한 전이 학습

충분하지 않은 훈련 데이터로 이미지 분류기를 훈련하고 싶다

→ 사전훈련된 모델의 하위층 사용

14.8 분류와 위치 추정

물체의 위치를 추정 - 회귀 작업

바운딩 박스를 예측 = 물체 중심의 수평/수직 좌표, 높이, 너비 예측

레이블을 만드는 것은 어려움

- 오픈 소스 이미지 레이블 도구 검색 & 사용
- 크라우드소싱 플랫폼 고려

→ 모델이 바운딩 박스를 얼마나 잘 예측해 내는지의 지표: **IoU**

14.9 객체 탐지

객체 탐지: 하나의 이미지에서 여러 물체를 분류하고 위치를 추정하는 작업

- 슬라이딩 방식 : 쉽지만 중복이 일어남

  불필요한 바운딩 박스 삭제 필요 → NMS(non-max suppression)

  완전 합성곱 신경망 사용 !

14.9.1 완전 합성곱 신경망

완전 합성곱 신경망(FCN) : 빠르게 이미지에 슬라이딩시킬 수 있는 방법

✔️ CNN 맨 위 밀집 층을 합성곱 층으로 바꿀 수 있다

밀집 층의 유닛 수 = 합성곱 층의 필터 수

밀집 층의 입력 특성 맵 크기 = 합성곱 층의 필터 크기

‼ 특정 입력 크기 이미지 처리 → 어떤 크기의 이미지도 처리

→ FCN은 전체 이미지를 딱 한번 처리한다. ~ **YOLO**

가장 최신 YOLO : YOLOv3

- 격자 셀마다 5개의 바운딩 박스 출력
- 격자 셀에 대한 상대 좌표 예측
- 신경망 훈련 전, 앵커 박스(= 사전 바운딩 박스)라고 부르는 5개의 대표 바운딩 박스 크기 찾기
- 네트워크가 다른 이미지를 사용하여 훈련

<mAP>

mAP(= mean average precision) : 객체 탐지에서 사용하는 평가 지표

재현율 ↑ 정밀도 ↑ 영역 포함 - 특히, 재현율 값이 낮을 때

- 최소 00% 재현율에서의 최대 정밀도 - 평균 → 평균 정밀도(AP)

  각 클래스의 AP를 계산 - 평균 AP 계산 → mAP

객체 시스템에서는,

정확한 클래스 탐지 - 잘못된 위치 → 올바른 예측 X

- IOU 임계점 정의
- mAP@IOU 임계점

14.10 시맨틱 분할

시맨틱 분할 : 각 픽셀은 속한 객체에서 클래스로 분류

- 클래스가 같은 물체는 구별 X
- 점진적으로 위치 정보 소실 (1 이상의 스트라이드를 사용하는 층 때문)
  1. CNN을 FCN으로 변환
  2. CNN이 입력 이미지에 적용하는 전체 스트라이드 계산(ex. 32) (1보다 큰 스트라이드를 모두 더하기)
  3. 32배만큼 작은 특성 맵 출력

→ 해상도를 32배로 늘리기 = **업샘플링 층** 하나 추가

🔥 업샘플링 방법

- 전치 합성곱 층 사용
  1. 이미지에 빈 행과 열을 삽입
  2. 일반적인 합성곱 수행 (부분 스트라이드를 사용하는 일반 합성곱)

~~↓ 떨어지는 정확도 개선~~

- 아래쪽 층부터 스킵 연결 추가
  1. 2배로 출력 이미지 업샘플링 → 해상도를 2배로 키우기
  2. 이 결과를 16배로 늘려 업샘플링

  → 풀링층에서 잃은 일부 공간 정보 복원

  → 해상도를 증가시는 방법 : **초해상도**

인스턴스 분할: 동일한 클래스 물체를 개별적으로 구분하여 표시

14.11 연습문제