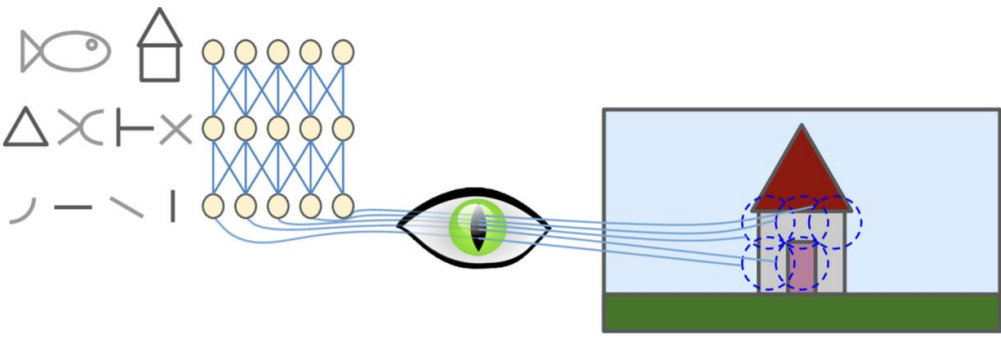
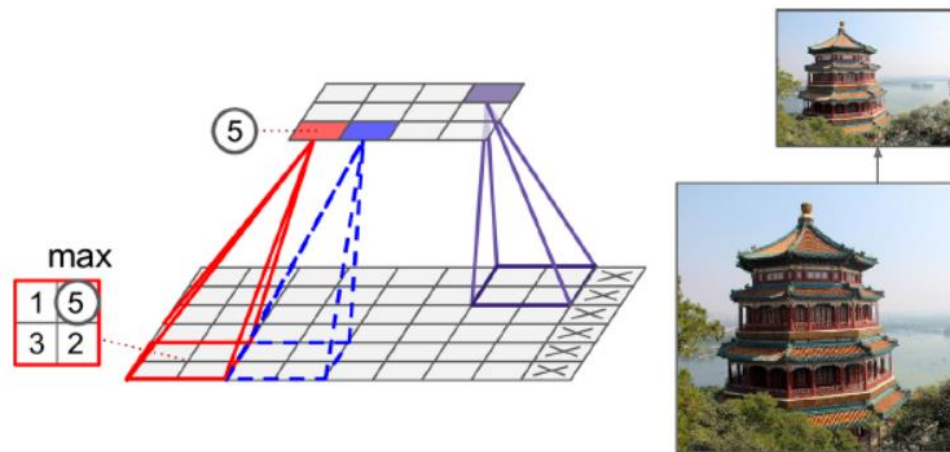


TAVE 서기

서기 내용			
서기 일자	21.11.06	서기	서가을
주제	합성곱 신경망을 사용한 컴퓨터 비전		
시간	20:30~22:30	장소	Zoom 미팅
스터디 인원	고성호, 권기호, 이아현, 서가을 : 시작		
			
	고성호, 권기호, 이아현, 서가을 : 종료		
			

	내용
배운 내용	<p>[목차]</p> <p>Chapter 14. 합성곱 신경망을 사용한 컴퓨터 비전</p> <p>14.1 시각 피질 구조</p> <p>14.2 합성곱 층</p> <p>14.3 풀링 층</p> <p>14.4 CNN 구조</p> <p>14.5 케라스를 사용해 ResNet-34 CNN 구현하기</p> <p>14.6 케라스에서 제공하는 사전훈련된 모델 사용하기</p> <p>14.7 사전훈련된 모델을 사용한 전이 학습</p> <p>14.8 분류와 위치 추정</p> <p>14.9 객체 탐지</p> <p>14.10 시맨틱 분할</p> <p>14.11 연습문제</p>
	<p>14.1. 시각 피질 구조</p>  <ul style="list-style-type: none"> • 눈의 한 쪽을 자극하면 특정 뉴런만 반응한다. • 이러한 연구를 바탕으로 신인식기, CNN, LeNet-5 구조 나옴. • 왜 이미지 인식엔 DNN 이 아닌 CNN 인가 • DNN: flatten 을 통해 받아 공간 위치 정보 훼손, 많은 뉴런 필요해 가중치 발생. • CNN: 공간 정보 훼손 없음, kernel size 만큼의 가중치만 학습 <p>14.2 합성곱 층</p> <ul style="list-style-type: none"> • 학교 시험 일정 때문에 11/8 (월) 발표 예정 <p>14.3 풀링 층</p>



- 최대 풀링 층은 계산량, 메모리 사용량, 파라미터 수 감소, 작은 변화에도 불변성 만듦.
- 평균 풀링 층보다 최대 풀링 층 더 성능 좋아 많이 사용
- 흔치 않지만 깊이 차원으로도 수행. 이를 통해 CNN 다양한 특성 학습

```
output = tf.nn.max_pool(images,
                          ksize=(1, 1, 1, 3),
                          strides=(1, 1, 1, 3),
                          padding="valid")
```

- 케라스는 깊이방향 풀링 층을 제공하지 않지만 텐서플로 저수준 딥러닝 API 를 사용할 수 있음

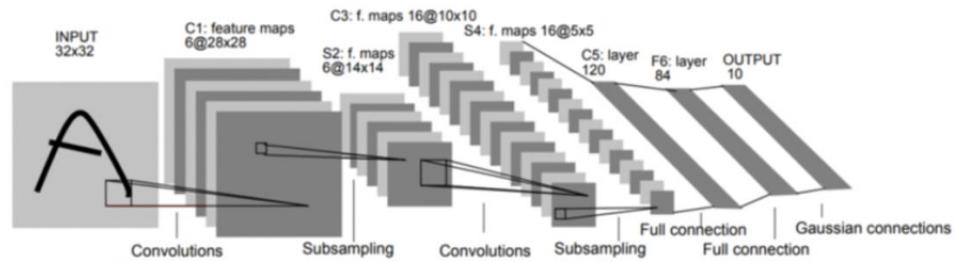
```
depth_pool = keras.layers.Lambda(lambda X: tf.nn.max_pool(
    X, ksize=(1, 1, 1, 3), strides=(1, 1, 1, 3), padding="
```

- 케라스 모델의 층으로 사용하고 싶으면 Lambda 층으로 감싸면 됨

14.4 CNN 구조

- 합성곱 층, 풀링 층을 반복해서 쌓는 것이 기본 구조
- 합성곱층에서는 큰 커널 대신 작은 커널 두 개를 사용하는 것이 더 좋음
- 단, 처음에는 큰 커널, 2 이상의 스트라이드 사용

14.4.1 LeNet-5



- MNIST 에 사용하는 CNN 구조
- MNIST 이미지는 제로패딩 -> 28X28 픽셀에서 32X32 픽셀이 됨.
- 네트워크 전 픽셀 키워놓고 네트워크하며 크기가 줄어듦

14.4.2 AlexNet

- LeNet 과 비슷하며 더 크고 깊음
- 처음으로 합성곱 층에 풀링 층을 쌓지 않고 합성곱 층끼리 쌓음.
- 과대적합을 줄이기 위한 방법
- 훈련하는 동안 드롭아웃 50% 적용
- 훈련 이미지를 랜덤으로 여러 간격으로 이동하거나 수평으로 뒤집고 조명을 바꿈
- 데이터증식: 인위적으로 훈련샘플을 만들어 크기를 늘림.
- 가장 강하게 활성화된 뉴런이 다른 특성 맵에 있는 같은 위치의 뉴런을 억제

14.4.3 GoogLeNet

- 인셉션 모델을 가지고 있어 이전 구조보다 효과적
- 처음에 입력 신호가 복사되어 네 개의 다른 층에 주입됨
- 모든 합성곱 층은 ReLU 활성화 함수 사용
- 두 번째 합성곱 층은 각기 다른 크기의 패턴을 잡음
- 모든 층의 출력 높이와 너비가 모두 입력과 같게 해 모든 출력을 깊이 연결 층에서 깊이 방향으로 연결 가능
- 텐서플로의 axis=3 매개변수로 tf.concat() 연산을 사용해 구현 가능
- 모든 합성곱 층은 ReLU 활성화 함수 사용

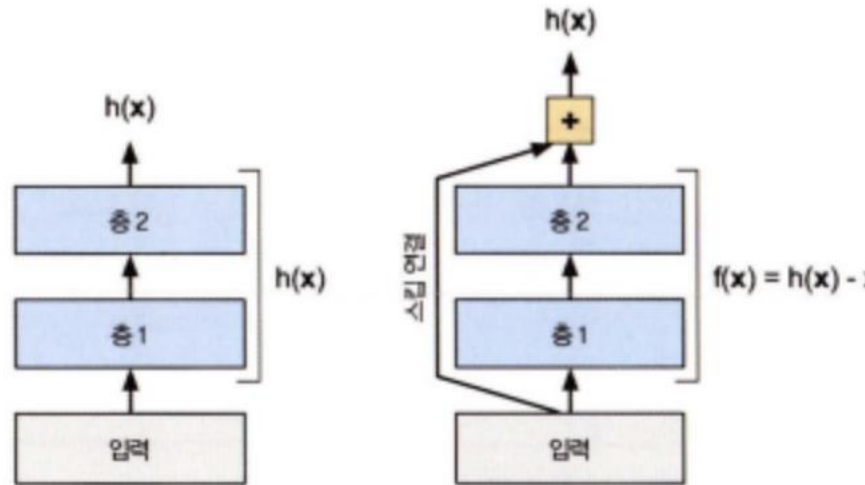
14.4.4 VGGNet

- 2 개의 합성곱 층과 풀링 층 뒤에 2 개의 합성곱 층과 풀링 층이 등장하는 방식
- VGGNet 의 종류에 따라 16 개 or 19 개의 합성곱 층 존재
- 밀집네트워크는 2 개의 은닉층과 출력층으로 구성

14.4.5 ResNet

- 더 적은 파라미터를 사용해 더 깊은 네트워크
- 스킵 연결: 깊은 네트워크 훈련,

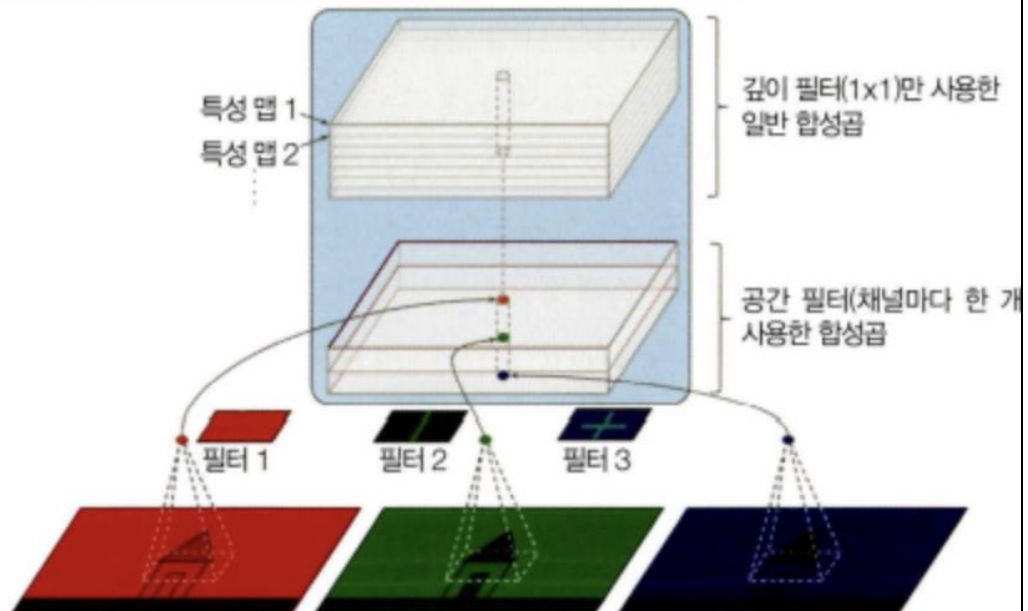
- 잔차 학습: 스킵 연결을 추가해 네트워크가 $f(x) = h(x) - x$



- 심층 잔차 네트워크: 스킵 연결을 가진 작은 신경망인 잔차유닛을 쌓은 것
- 신경망 훈련에서 목적 함수 $h(x)$ 모델링이 목표
- 목적함수가 항등함수에 가까우면 훈련 속도 상승

14.4.6 Xception

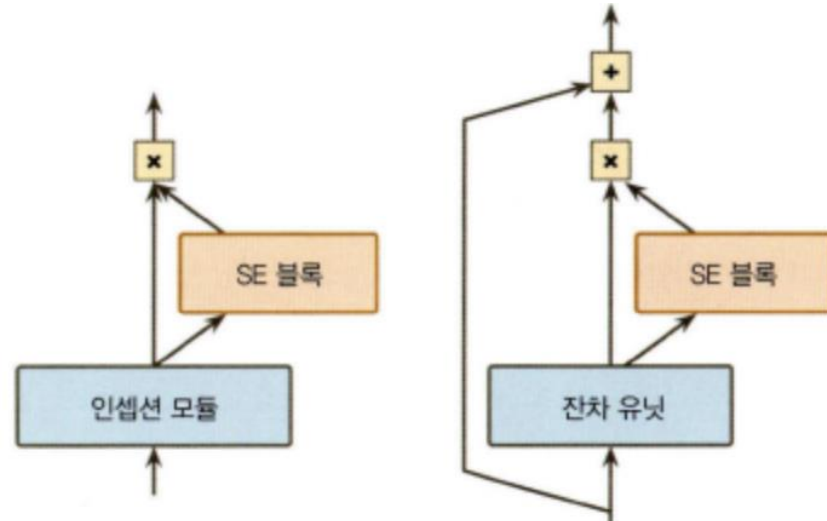
- 대규모 비전 문제에서 Inception-v3 보다 성능 좋음.
- 분리 합성곱 층은 두 부분으로 구성
- 첫 번째 부분: 하나의 공간 필터를 각 입력 특성 맵에 적용
- 채널 사이 패턴만 조사



- 분리 합성곱 층은 입력 채널마다 하나의 공간 필터만 가짐 -> 입력층과 같이 채널이 너무 적은 층 다음에 사용하는 것은 피하기

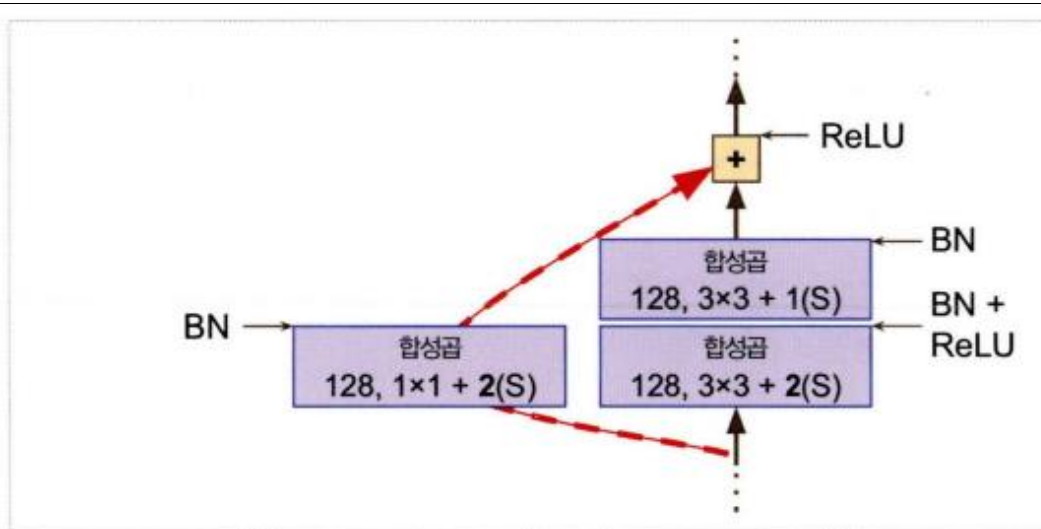
14.4.7 SENet

- ResNet 확장 버전
- 원래 구조에 있는 모든 유닛에 SE 블록이라는 작은 신경망 추가 -> 성능 향상



- 하나의 SE 블록은 3 개의 층으로 구성(전역 평균 풀링층, ReLU 활성화 함수를 사용하는 밀집 은닉층, 시그모이드 활성화 함수를 사용하는 집 출력층)
 1. 전역 평균 풀링 층이 각 특성 맵에 대한 평균 활성화 값 계산
 2. 다음 층에서 압축 발생. 이 저차원 벡터(임베딩)는 특성 응답의 분포 표현
 3. 출력층은 임베딩을 받아 특성 맵마다 0 과 1 사이의 하나의 숫자를 담은 보정된 벡터를 출력
 4. 특성 맵과 보정된 벡터를 곱해 관련 없는 특성값을 낮추고 관련 있는 특성값은 그대로 유

14.5. 케라스를 사용해 ResNet-34 CNN 구현하기



- 맵과 잔차유닛의 반비례
- 64 개의 특성 맵을 출력하는 3 개의 잔차유닛(RU),
- 128 개 맵의 4 개 잔차유닛,
- 256 개 맵의 6 개 잔차유닛,
- 512 개 맵의 3 개 잔차유닛을 포함
- 입출력 사이즈가 달라진다는 점을 해결하기 위해 128, 1x1, 2(s) 사용

```
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()]

    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)
```

14.6. 케라스에서 제공하는 사전훈련된 모델 사용하기

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

- 케라스에서 제공하는 사전훈련된 모델 사용하기

- `tf.image.resize()` : 가로세로 비율 유지 불가능
- `tf.image.crop_and_resize()` : 가로세로 비율 가능
- ResNet 모델은 caffe 스타일을 사용

```
inputs = keras.applications.resnet50.preprocess_input(images_resized * 255)
```

```
import numpy as np
from sklearn.datasets import load_sample_image

# Load sample images
china = load_sample_image("china.jpg") / 255
flower = load_sample_image("flower.jpg") / 255
images = np.array([china, flower])
```

- 이전에 /255 를 했으므로 *255 를 함.

•

14.7 사전훈련된 모델을 사용한 전이 학습

- 텐서플로는 데이터셋이 분리 X -> 직접 분리
- 전처리를 통해 사진 크기 통일(훈련, 검증, 테스트셋)
- 모델 생성
- 모델 컴파일, 훈련 (훈련 초기 사전훈련된 가중치 동결)
- 프리페치 적용을 통해 더 빠르게 할 수 있음.

```
for layer in base_model.layers:
    layer.trainable = False

optimizer = keras.optimizers.SGD(learning_rate=0.2, momentum=0.9, decay=0.01)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set,
                    steps_per_epoch=int(0.75 * dataset_size / batch_size),
                    validation_data=valid_set,
                    validation_steps=int(0.15 * dataset_size / batch_size),
                    epochs=5)


Epoch 1/5
86/86 [=====] - 570s 7s/step - loss: 1.4612 - accuracy: 0.7972 - val_loss: 1.2210 - val_accu
acy: 0.8088
Epoch 2/5
86/86 [=====] - 559s 7s/step - loss: 0.5396 - accuracy: 0.9019 - val_loss: 0.8222 - val_accu
acy: 0.8585
Epoch 3/5
86/86 [=====] - 559s 7s/step - loss: 0.2692 - accuracy: 0.9331 - val_loss: 0.7442 - val_accu
acy: 0.8640
Epoch 4/5
86/86 [=====] - 559s 7s/step - loss: 0.1337 - accuracy: 0.9640 - val_loss: 0.6899 - val_accu
acy: 0.8621
Epoch 5/5
86/86 [=====] - 560s 7s/step - loss: 0.0904 - accuracy: 0.9713 - val_loss: 0.7002 - val_accu
acy: 0.8621
```

- 정확도 80% -> 새로 추가한 최상위 층이 잘 훈련되었다는 뜻.
- 모든 층의 동결 해제 후 훈련 지속
- 층 동결/ 해제 시 반드시 모델 컴파일
- 사전훈련된 가중치의 훼손 방지 위해 작은 학습률 사용(0.2 - > 0.01)
- Test 셋에서 정확도 95%
- CPU 사용 시 매우 느리므로 GPU 사용 권장

14.8 분류와 위치 추정

- 바운딩박스를 만드는 것은 시간이 많이 듦
- Labeling 의 문제점

- 꽃 데이터셋은 꽃 주위에 바운딩 박스를 갖고 있지 않기 때문에 직접 만들어서 추가하는 작업 필요
- 바운딩 박스 평가 지표로는 MSE 보다 IoU 가 더 적합
- 일반적 특성 감지하는 심층 합성곱 신경망에 더 잘 작동

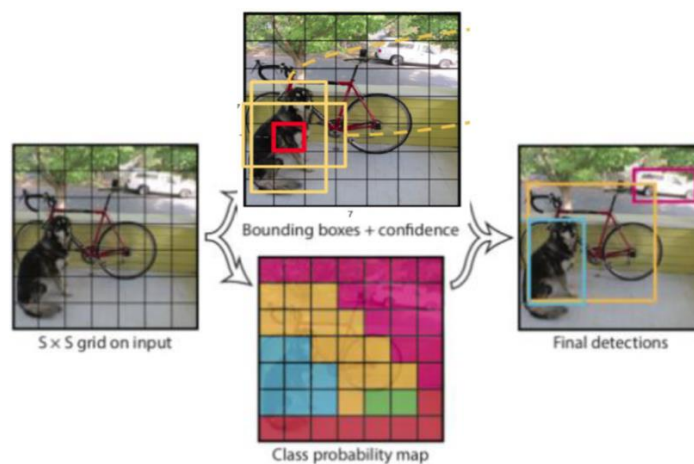
$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


14.9 객체 탐지

14.9.1 완전 합성곱 신경망

- 전통적 방식: CNN 무한반복: 여러 바운딩박스 만들고 가장 IoU 가 높은 박스만 살리는 방식으로 제거할 바운딩 박스 없어질 때까지 하는 과정을 무한 반복
- 개선한 것이 완전 합성곱 신경망 (FCN): 여러 필터 씌워 1X1 특성맵 출력해 한번의 합성곱 -> 시간 절약

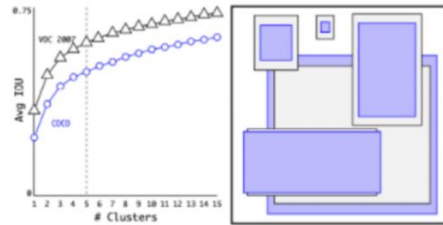
14.9.2 YOLO



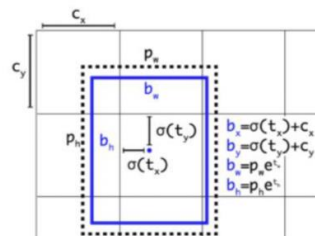
- 기존 객체 탐지 모델 vs YOLO

- 기존 객체 탐지 모델: Regional proposal → Classification
- YOLO: 한 번에 객체 탐지 (바운딩 박스, Classification 동시에)
- Kernel 돌리지 않고도 그림을 한번에 인식

■ Anchor Box 사용



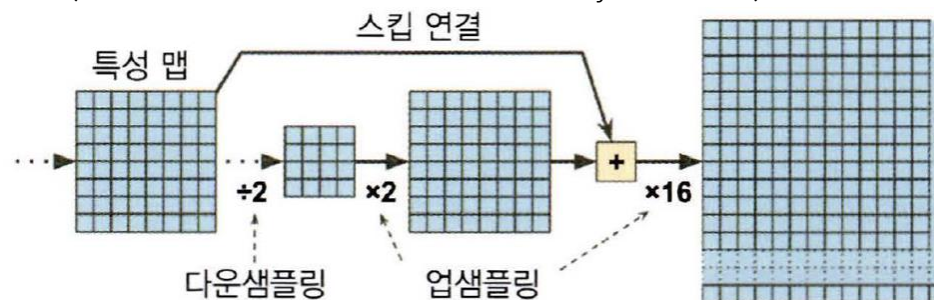
■ Bounding Box Regression



- YOLOv3
- 1. Anchor Box 사용: 사전에 크기와 비율 정해진 박스 사용, 학습 통해 박스 조정
- 2. Bounding Box Regression: 박스의 상대 좌표 이용(학습에 유리)
- 위치 학습에 더 뛰어남.

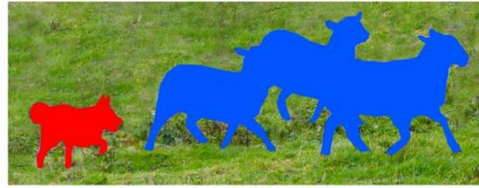
14.10 시맨틱 분할

- 픽셀 단위가 아닌 픽셀이 속한 클래스 단위로 분류(같은 클래스의 픽셀은 분류 X)
- 일반적인 CNN 을 통과할 때 점진적으로 위치정보를 잃음(1 이상의 스트라이드를 사용하는 layer 때문에)

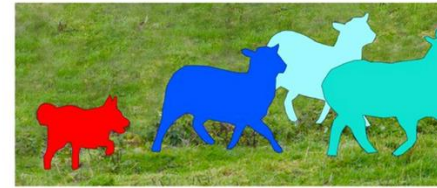


- 사전훈련된 CNN → FCN 하면 전체 스트라이드 32(input 이미지보다 32 배 작은 맵) → 업샘플링 통해 해상도 32 배 -

> skip architecture 추가해 super-resolution (초해상도)



Semantic Segmentation



Instance Segmentation

- 인스턴스 분할: 같은 중에서도 다른 객체 구별함. Ex. Mask R-CNN

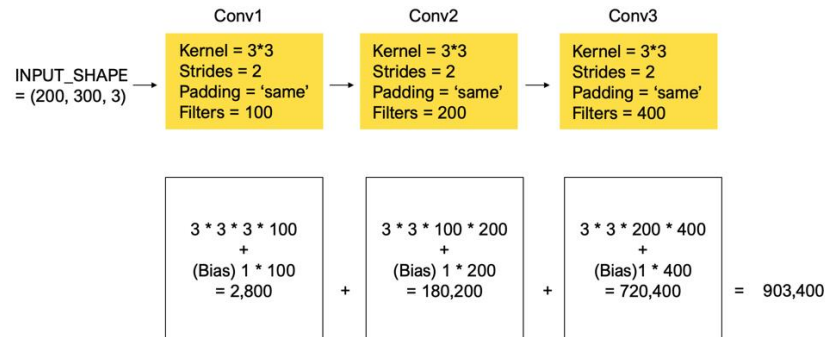
14.11 연습문제

- 1. 이미지 분류에서 완전 연결 DNN 보다 CNN 이 나은 점?
- 기본 구조가 시각 피질 구조이므로 필터를 이용해 이미지 분류에 적합하고 과대 적합 가능성 적음
- 2. 3X3 커널, 스트라이드 2, "same" 패딩으로 된 합성곱 층 세 개로 구성된 CNN 이 있다. 가장 아래 층은 특성 맵 100 개 출력, 중간 층은 200 개, 가장 위 층은 400 개 출력한다. 입력 이미지는 200X300 픽셀의 RGB 이미지이다.

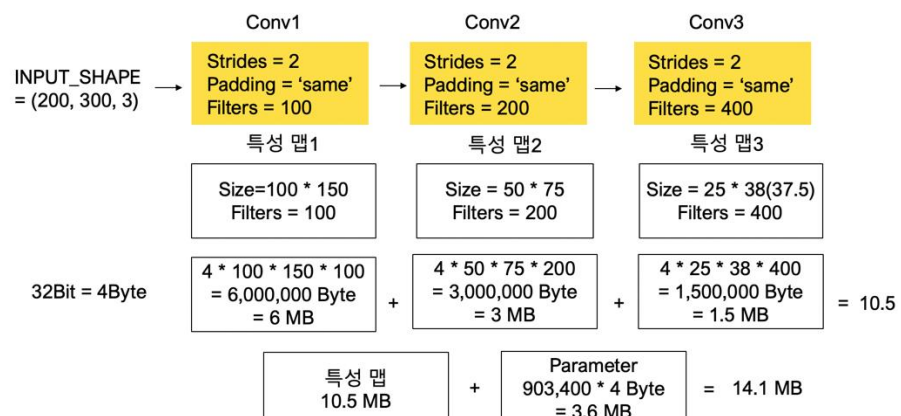
2-1 이 CNN 의 전체 파라미터 수는?

2-2 32 비트 부동소수를 사용한다면 네트워크가 하나의 샘플을 예측하기 위해 적어도 얼마의 RAM 이 필요한가?

2-3 50 개의 이미지를 미니배치로 훈련시킬 때는 얼마가 필요한가?



- 2-1: 커널 크기, 깊이(R, G, B), 필터 곱함 + 가중치 두 번째부터는 이전 필터가 특성맵이 됨.



- 2-2:

층에서 필요한 RAM

1 Sample : 10.5MB



50 Sample : 525MB

입력 이미지를 위한 RAM

INPUT = (200, 300, 3)



$50 * 4(\text{Byte}) * 200 * 300 * 3$
= 36MB

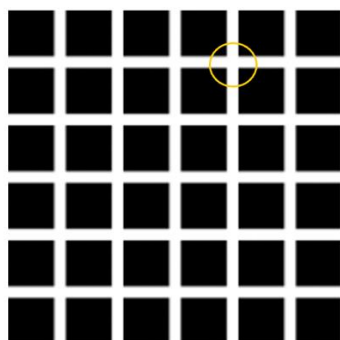
모델 파라미터를 위한 RAM

$903,400 * 4 \text{ Byte}$
= 3.6 MB

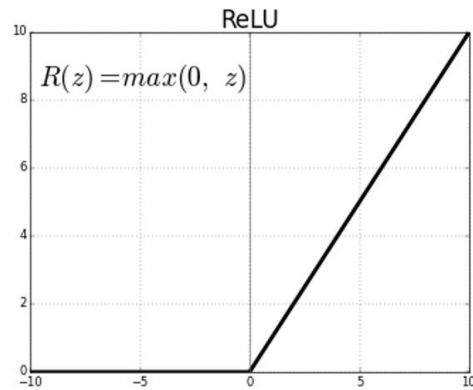
$$525 + 36 + 3.6 = 564.6\text{MB}$$

- 2-3: $5 * 10.5 = 525$ 에 입력이미지, 모델 파라미터 위한 RAM 더해 564.6 MB
- 3. 어떤 CNN 을 훈련시킬 때 GPU 에서 메모리 부족이 발생했다면 해결 어떻게?
미니배치 감소, 스트라이드 증가(출력사이즈 감소), 층 제거 (파라미터 감소), 32 비트에서 16 비트로, 분산 CNN, RAM 확장
- 4. 같은 크기의 스트라이드의 합성곱 층 대신 최대 풀링층을 추가하는 이유는?
합성곱층: 파라미터 수 많음
최대풀링층: 파라미터 없음
Max Pooling 을 사용하면 파라미터가 0 이 되므로 램을 아낄 수 있음.
- 5. LRN 층을 추가해야 하는 이유?

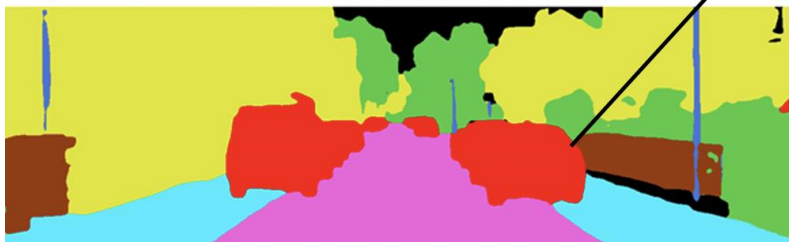
헤르만 격자



검은 사각형 때문에 이웃뉴런이 억제되는 현상 때문에



- 하지만 LRN 보다 Batch Normalization 을 사용하는 추세
- 7. 완전 합성곱 신경망은 무엇이고 밀집층을 어떻게 합성곱 층으로 바꿀 수 있는가?
FCN: CNN 에 있는 밀집층을 합성곱이나 풀링층으로 대체해 합성곱과 풀링층으로만 구성된 신경망으로 객체탐지나 시맨틱 분할에 이용.
커널의 크기, 스트라이드, 패딩으로 밀집층과 같은 역할의 CNN 층 생성 가능
- Padding: Same
- Kernel Size = Fully Connected Input Size
- Filter_Size = Neuron_Size
- 시맨틱 분할에서의 기술적 어려움?



- 각 물체를 큰 하나의 픽셀 덩어리로 인식하기 때문에 이미지의 위치 정보를 손실.
- FCN, 업샘플링 통해 복구
- 9. 자신만의 CNN 을 만들고 MNIST 데이터셋에서 가능한 최대 정확도를 달성해

	<div style="display: flex; justify-content: space-around; align-items: center;"> <div style="text-align: center;"> <p>CNN Kernel size = 3, padding = same</p> <p>MAX_POOLING Pool size = 3</p> </div> <div style="text-align: center;"> </div> <div style="text-align: center;"> <p>DENSE activation = <u>relu</u> activation = <u>softmax</u></p> </div> <div style="text-align: center;"> </div> </div> <p style="text-align: center; margin-top: 20px;">보세요.</p>
과제할 당	<p>Chapter 15</p> <p>15.1 고성호</p> <p>15.2 안세윤</p> <p>15.3 허주희</p> <p>15.4 박제윤</p> <p>15.5 이아현</p>
특이사 항	스터디 발표 및 질의응답 시간 이후 다음 시간에 볼 포럼을 정하고 다음 일정을 안내했습니다.
비고	없음