딥러닝 비전

Computer Vision and Pattern Recognition

Byeongjoon Noh

powernoh@sch.ac.kr

SCH SOON CHUN HYANG UNIVERSITY

Most of the slides are available on Senseable AI Lab homepage: https://sailab.space/courses/

Contents

1. 깊은 신경망 - Review

프로그래밍 실습: 다층 퍼셉트론으로 MNIST 인식

- 예제는 tensorflow를 활용 (torch로 구현 가능)

프로그램 7-2 다층 퍼셉트론으로 MNIST 인식하기(SGD 옵티마이저)

```
01  import numpy as np
02  import tensorflow as tf
03  import tensorflow.keras.datasets as ds
04
05  from tensorflow.keras.models import Sequential
06  from tensorflow.keras.layers import Dense
07  from tensorflow.keras.optimizers import SGD
08
09  (x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
10  x_train=x_train.reshape(60000,784)
11  x_test=x_test.reshape(10000,784)
12  x_train=x_train.astype(np.float32)/255.0
13  x_test=x_test.astype(np.float32)/255.0
14  y_train=tf.keras.utils.to_categorical(y_train,10)
15  y_test=tf.keras.utils.to_categorical(y_test,10)
16
```

1차원 구조로 변환

[0,1]로 정규화

데이터 준비

원핫 코드로 변환

프로그래밍 실습: 다층 퍼셉트론으로 MNIST 인식

- 예제는 tensorflow를 활용 (torch로 구현 가능)

프로그램 7-3 다층 퍼셉트론으로 MNIST 인식하기(Adam 옵티마이저)

```
    07행과 21을 제외한 다른 부분은 [프로그램 7-2]와 같음
07  from tensorflow.keras.optimizers import Adam
21  mlp.compile(loss='MSE',optimizer=Adam(learning_rate=0.001),metrics
    =['accuracy'])
```

은닉층 노드 개수 512, 출력층 노드 개수 10

은닉층 활성 함수 tanh, 출력층 활성 함수 softmax

```
17  mlp=Sequential()
18  mlp.add(Dense(units=512,activation='tanh',input_shape=(784,)))
19  mlp.add(Dense(units=10,activation='softmax'))
20
21  mlp.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),
    metrics=['accuracy'])
22  mlp.fit(x_train,y_train,batch_size=128,epochs=50,validation_
    data=(x_test,y_test),verbose=2) ①
23
24  res=mlp.evaluate(x_test,y_test,verbose=0)
25  print('정확률=',res[1]*100) ②
```

입력층 모양

모델 선택 (신경망 구조 설계)

MSE 손실 함수

SGD 옵티마이저

학습률

학습

예측(성능 측정)

훈련 집합

미니 배치 크기

세대 수

프로그래밍 실습: 다층 퍼셉트론으로 MNIST 인식

- 예제는 tensorflow를 활용 (torch로 구현 가능)

```
Epoch 1/50 ①
469/469 - 1s - loss: 0.0876 - accuracy: 0.2390 - val_loss: 0.0842 - val_accuracy:
0.3410 - 1s/epoch - 3ms/step
Epoch 2/50
469/469 - 1s - loss: 0.0805 - accuracy: 0.4025 - val_loss: 0.0764 - val_accuracy:
0.4418 - 1s/epoch - 2ms/step
…
Epoch 49/50
469/469 - 1s - loss: 0.0189 - accuracy: 0.8870 - val_loss: 0.0179 - val_accuracy:
0.8941 - 1s/epoch - 2ms/step
Epoch 50/50
469/469 - 1s - loss: 0.0187 - accuracy: 0.8874 - val_loss: 0.0178 - val_accuracy:
0.8946 - 1s/epoch - 2ms/step
정확률= 89.4599974155426 ②
```

프로그래밍 실습: 성능 시각화

- SGD와 Adam 성능을 그래프로 비교

프로그램 7-4 다층 퍼셉트론으로 MNIST 인식하기(SGD와 Adam의 성능 그래프 비교)

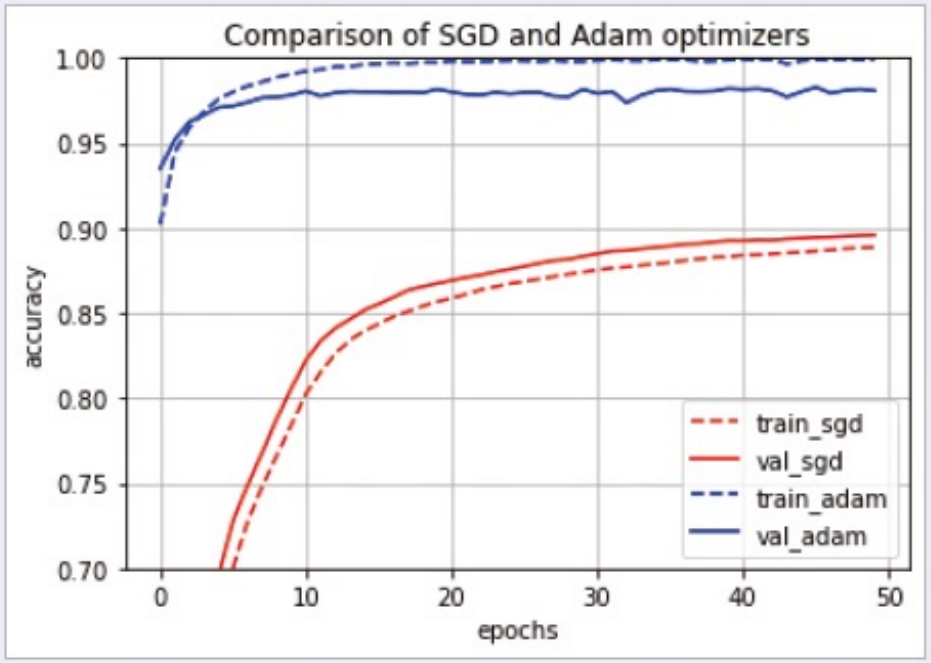
```
01  import numpy as np
02  import tensorflow as tf
03  import tensorflow.keras.datasets as ds
04
05  from tensorflow.keras.models import Sequential
06  from tensorflow.keras.layers import Dense
07  from tensorflow.keras.optimizers import SGD, Adam
08
09  (x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
10  x_train=x_train.reshape(60000,784)
11  x_test=x_test.reshape(10000,784)
12  x_train=x_train.astype(np.float32)/255.0
13  x_test=x_test.astype(np.float32)/255.0
14  y_train=tf.keras.utils.to_categorical(y_train,10)
15  y_test=tf.keras.utils.to_categorical(y_test,10)
16
17  mlp_sgd=Sequential()
18  mlp_sgd.add(Dense(units=512,activation='tanh',input_shape=(784,)))
19  mlp_sgd.add(Dense(units=10,activation='softmax'))
```

프로그래밍 실습: 성능 시각화

- SGD와 Adam 성능을 그래프로 비교

```
20
21  mlp_sgd.compile(loss='MSE',optimizer=SGD(learning_rate=0.01),metrics=['accuracy'])
22  hist_sgd=mlp_sgd.fit(x_train,y_train,batch_size=128,epochs=50,validation_
    data=(x_test,y_test),verbose=2)
23  print('SGD 정확률=',mlp_sgd.evaluate(x_test,y_test,verbose=0)[1]*100)
24
25  mlp_adam=Sequential()
26  mlp_adam.add(Dense(units=512,activation='tanh',input_shape=(784,)))
27  mlp_adam.add(Dense(units=10,activation='softmax'))
28
29  mlp_adam.compile(loss='MSE',optimizer=Adam(learning_rate=0.001),metrics
    =['accuracy'])
30  hist_adam=mlp_adam.fit(x_train,y_train,batch_size=128,epochs=50,validation_
    data=(x_test,y_test),verbose=2)
31  print('Adam 정확률=',mlp_adam.evaluate(x_test,y_test,verbose=0)[1]*100)
32
33  import matplotlib.pyplot as plt
34
35  plt.plot(hist_sgd.history['accuracy'],'r--')
36  plt.plot(hist_sgd.history['val_accuracy'],'r')
37  plt.plot(hist_adam.history['accuracy'],'b--')
38  plt.plot(hist_adam.history['val_accuracy'],'b')
39  plt.title('Comparison of SGD and Adam optimizers')
40  plt.ylim((0.7,1.0))
41  plt.xlabel('epochs')
42  plt.ylabel('accuracy')
43  plt.legend(['train_sgd','val_sgd','train_adam','val_adam'])
44  plt.grid()
45  plt.show()
```

```
SGD 정확률= 89.60000276565552
Adam 정확률= 98.07999730110168
```



(참고) 하이퍼파라미터 조정

파란 박스는 신경망 구조
빨간 박스는 학습에 관련된 하이퍼 매개변수

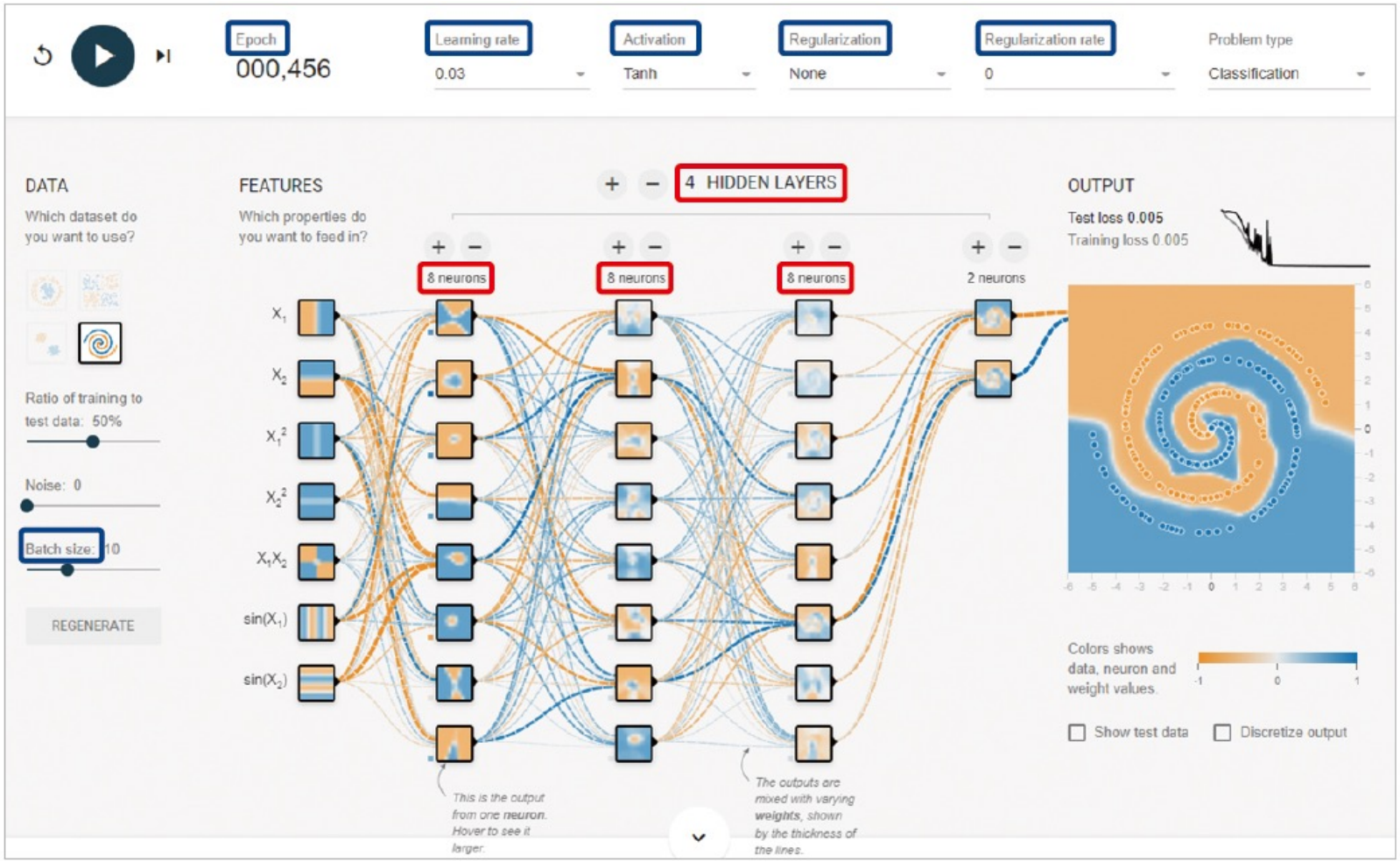


그림 7-22 하이퍼 매개변수 체험 사이트 https://playground.tensorflow.org/

프로그래밍 실습: CIFAR-10

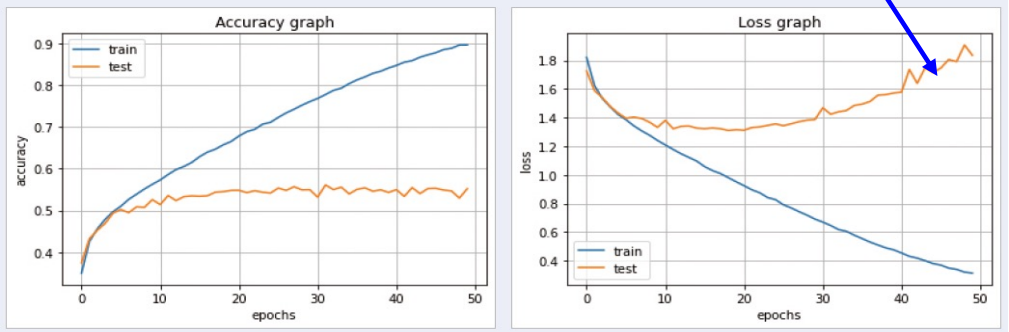
- 자연영상 인식

프로그램 7-6 깊은 다층 퍼셉트론으로 CIFAR-10 인식하기

```
    01~07행은 [프로그램 7-5]와 같음
08  … …
09  (x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
10  x_train=x_train.reshape(50000,3072)
11  x_test=x_test.reshape(10000,3072)
12  x_train=x_train.astype(np.float32)/255.0
13  x_test=x_test.astype(np.float32)/255.0
14  y_train=tf.keras.utils.to_categorical(y_train,10)
15  y_test=tf.keras.utils.to_categorical(y_test,10)
16
17  dmlp=Sequential()
18  dmlp.add(Dense(units=1024,activation='relu',input_shape=(3072,)))
19  dmlp.add(Dense(units=512,activation='relu'))
20  dmlp.add(Dense(units=512,activation='relu'))
21  dmlp.add(Dense(units=10,activation='softmax'))
    …
    23~47행은 [프로그램 7-5]와 같으며 27행의 dmlp.save('dmlp_trained.h5')는 삭제
```

Overfitting

```
정확률= 55.15999794006348
```



2. 딥러닝 비전 개요

컴퓨터 비전의 오랜 난제와 딥러닝의 혁신

- https://huggingface.co/facebook/detr-resnet-50-panoptic
- https://huggingface.co/spaces/akhaliq/CLIP_prefix_captioning

(a) 의미 분할 문제 (b) 영상 설명하기 문제

그림 7-1 딥러닝이 일으킨 혁신

패러다임의 변화

- 기계학습 이전의 패러다임
  - 규칙/통계 기반 접근 방법
  - 사람이 영상을 보고 규칙을 도출하고 프로그래밍
  - ➔ 분명한 한계

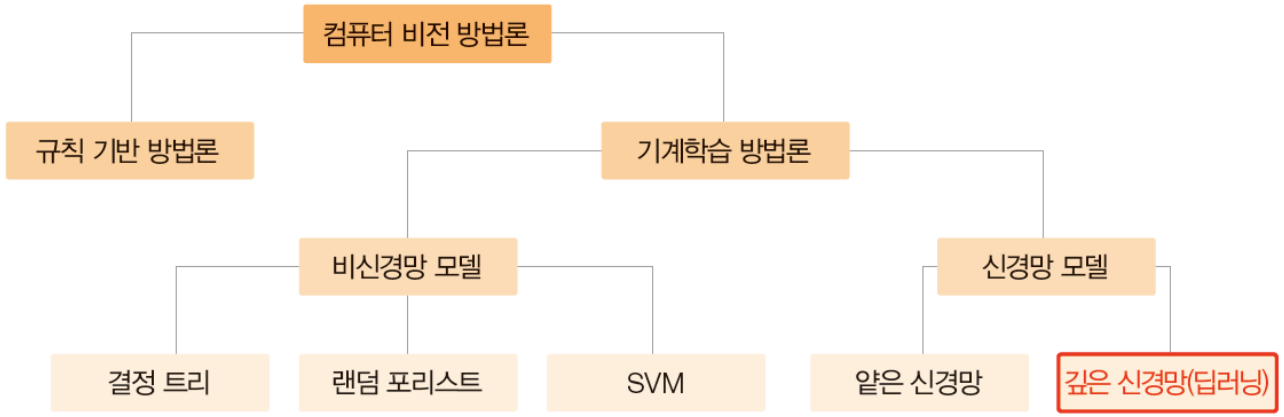
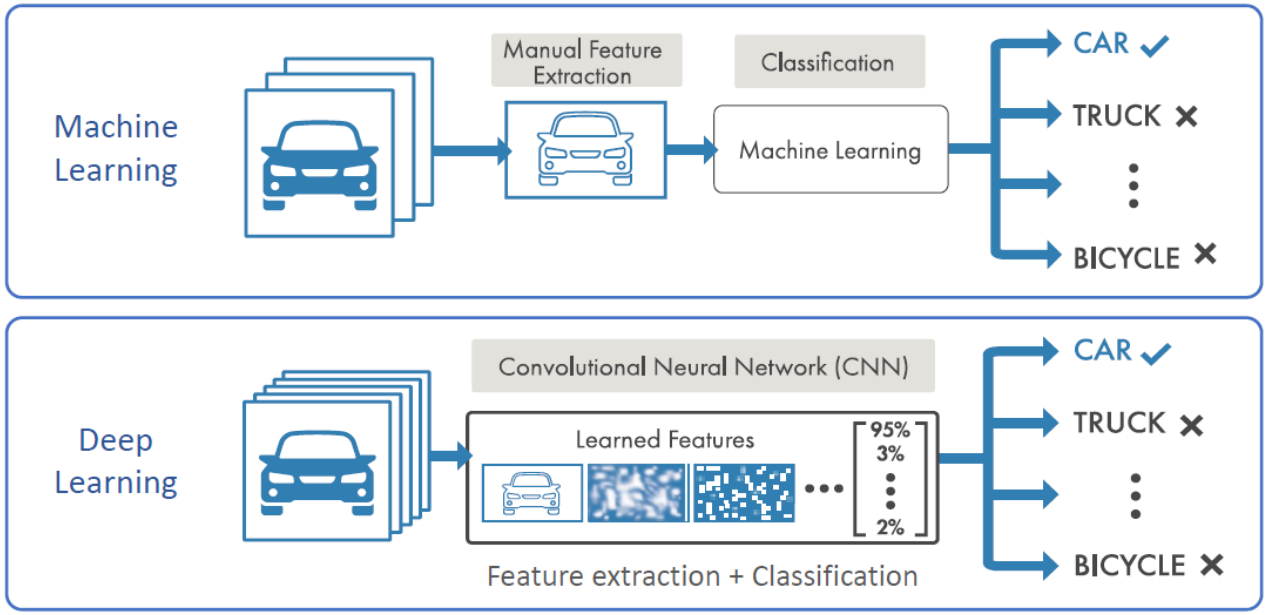


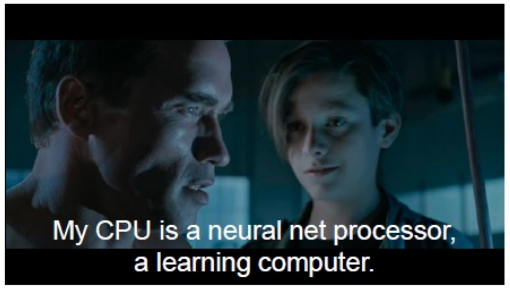
그림 7-2 컴퓨터 비전 방법론의 얼개

- 기계학습(딥러닝) 시대의 패러다임
  - 데이터 기반의 학습을 통해 문제를 해결
  - 기계학습: 주어진 데이터로부터 특징을 추출하고 모델의 입력으로 활용
    - 특징추출/특징차원축소 분야의 발전 이룩
  - 딥러닝: 특징추출과 분류(예측)을 한번에 하는 end-to-end 패러다임



딥러닝 시대

- Deep learning
  - 2000년대부터 사용되고 있는 심층 신경망의 또 다른 이름



Terminator 2 (1991)

AlexNet (2012)

Alpha Go (2016)

- 컴퓨터 비전 분야에서의 딥러닝

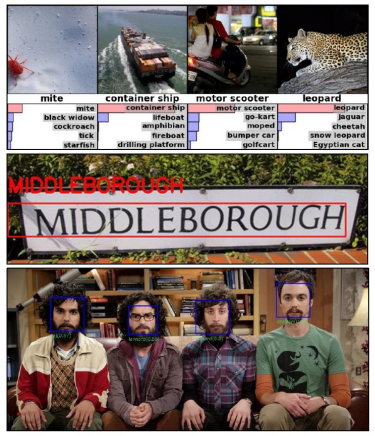
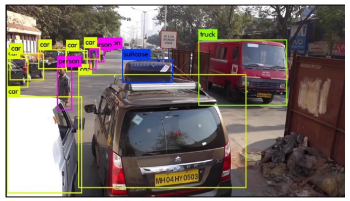
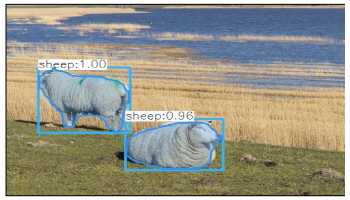


Image Recognition



Object Detection



Object Segmentation

Super-Resolution

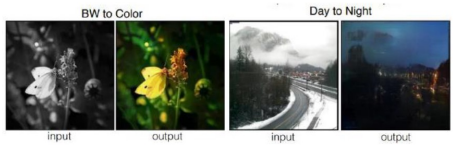
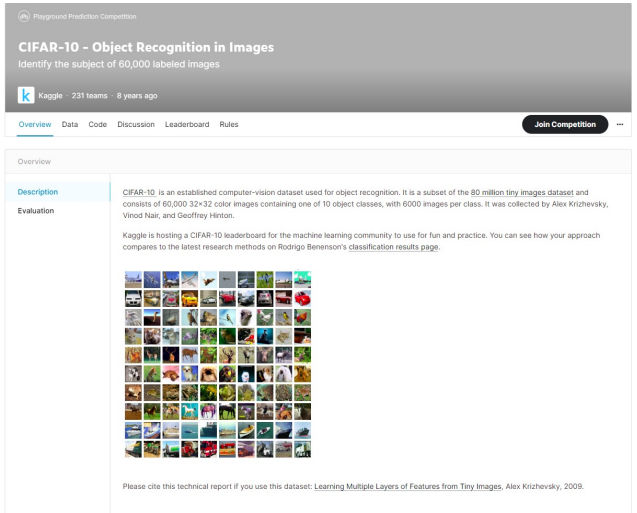


Image-to-Image Translation

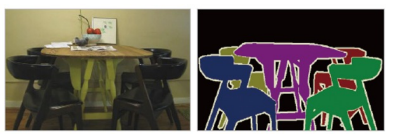
Image Inpainting

데이터셋과 방법론의 공진화

- 컴퓨터 비전은 AI 뿐만 아니라 데이터의 발전이 더 큰 혁신의 요인임
  - MNIST, CIFAR-10 등 꽤 난이도 있는 데이터셋 (현재는 비록 toy example 수준)
  - 도전적인 데이터셋 ➔ 대회 ➔ 경쟁 ➔ 아이디어 공유 ➔ 동반 성장



- 유명 데이터셋
  - ImageNet: 21,841 classes, 1,400만장
    - ILSVRC (대회): 1,000 classes 중 100만장 훈련, 5만장 검증 15만장 테스트
  - PASCAL VOC (Visual Object Classes): 20 classes, 50만장
  - COCO (Microsoft Common Object in Context): 80 classes, 33만장
  - Food-101: 101 classes, 10만장의 음식 사진
  - CheXpert: 65,240명 환자이 폐 X-ray 사진 약. 2만장



(a) Pascal VOC

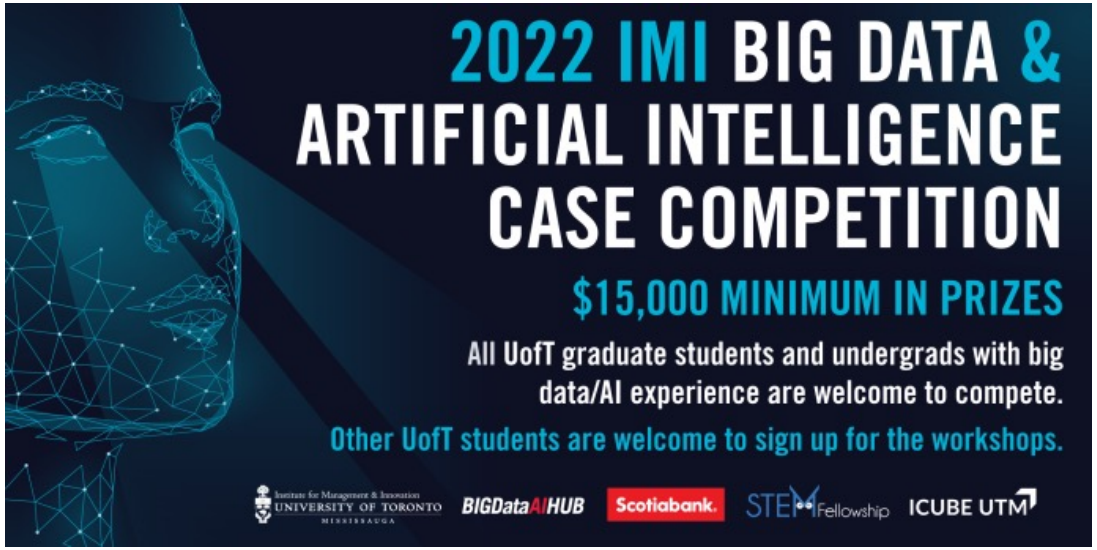
(b) ImageNet

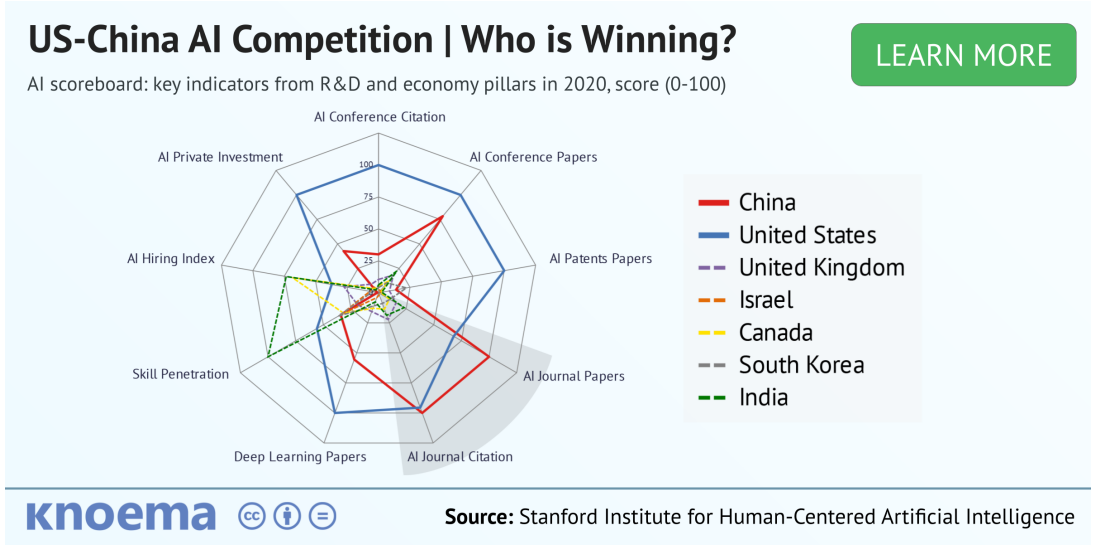
(c) COCO

(d) Open Images

데이터셋과 방법론의 공진화

- 합리적이고 공정한 성능 평가 척도 (metric) 필요
  - 세부 문제에 따라 다양한 기준 및 척도 적용
  - AI 대회는 척도와 척도를 계산하는 SW 모듈 제공 ➔ 공정성 유지
  - 훈련 집합 영상/GT(ground truth) 제공 ➔ 테스트 영상만 제공

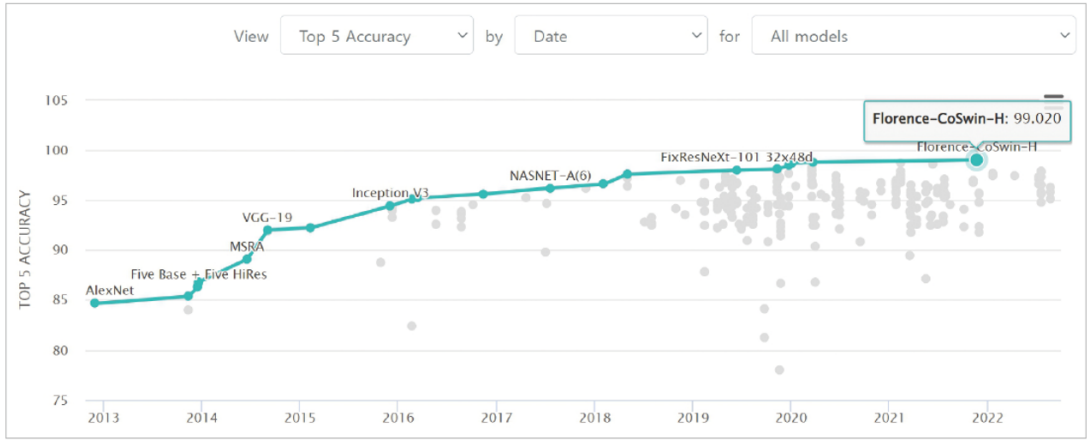




Computer vision tasks

- Image classification
  - 딥러닝에서 사례 분류 ➔ 사례 분류는 거의 다 풀림 ➔ 범주 분류에 초점
    - 사례 분류 (Instance classification): 모양과 텍스처가 고정된 특정 물체 분류 (표지판)
    - 범주 분류 (Categorical classification): 일반 물체 분류 (모양/자세 변화)
  - 초기 ILSVRC에서 고전 방법이 일부 우승(2회)

    ➔ 2012년부터 CNN기반의 신경망이 우승 (AlexNet) ➔ 패러다임 대 전환
  - Image classification performance (ImageNet dataset)



Computer vision tasks

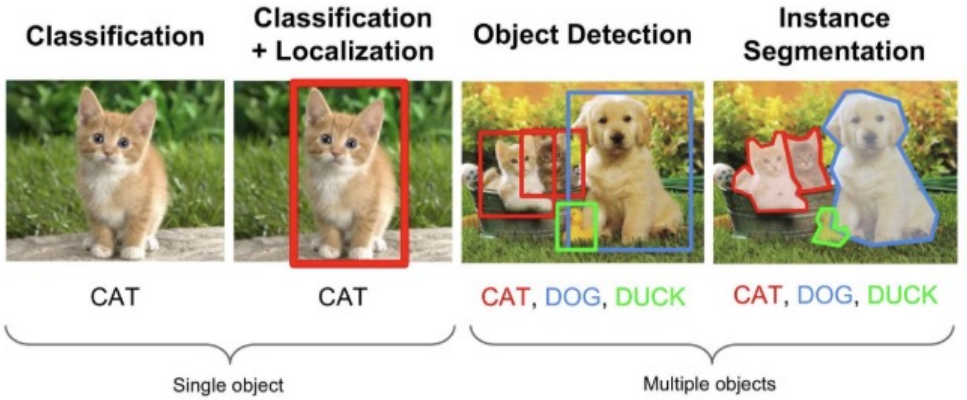
- Image classification ➔ Find-grained classification
  - 부류 내 변화가 크고, 부류 간 변화가 적어 아주 어려운 문제
  - 여러 데이터셋: Stanford dogs, Stanford cars, CUB-200, iNat, Oxford 102 flowers

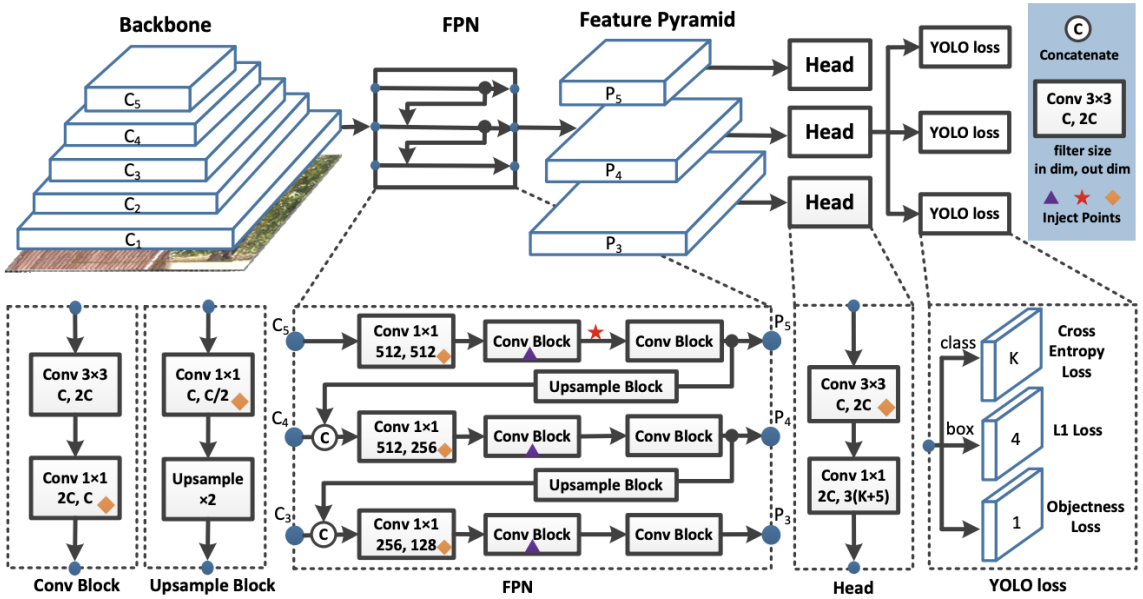
Laysan Albatross

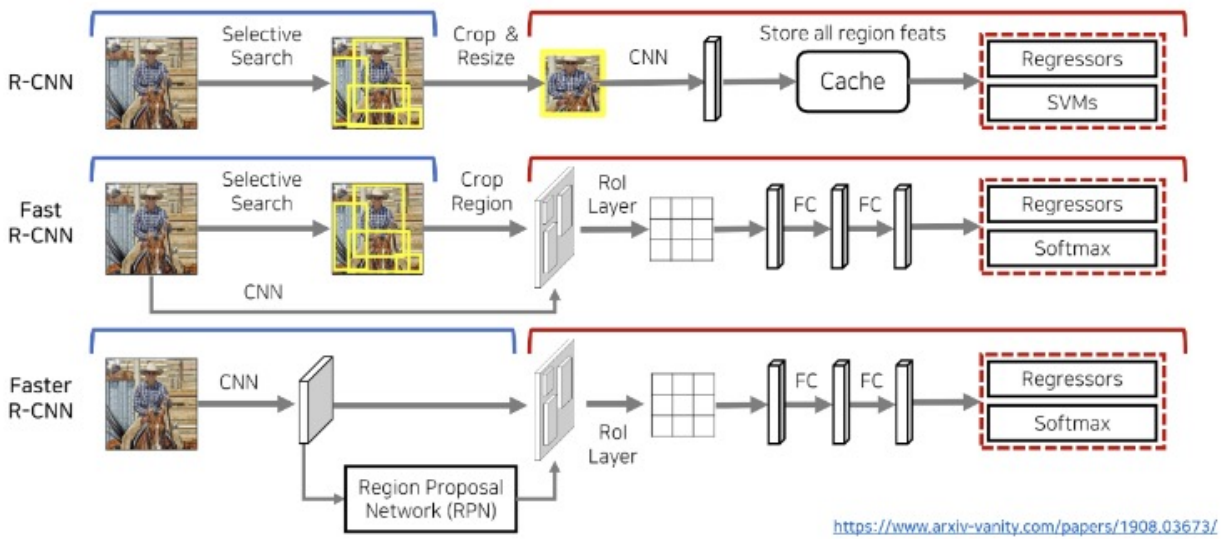
Rusty Blackbird Fish Crow Brewer Blackbird Shiny Cowbird

Computer vision tasks

- Object detection
  - Detecting and localizing object in image/video
  - 정확하고 효율적인 모델을 만들기 위한 노력
  - YOLO-series / R-CNN type
    - Yolo v1~v8, Fast/Faster/Mask R-CNN, etc.

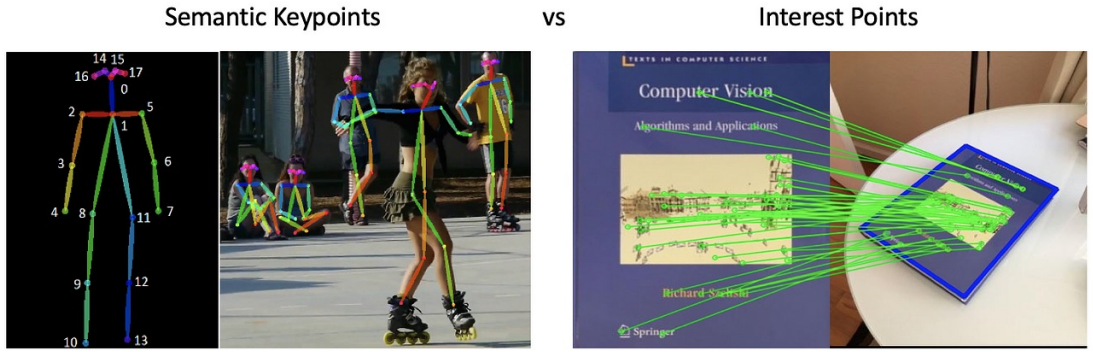


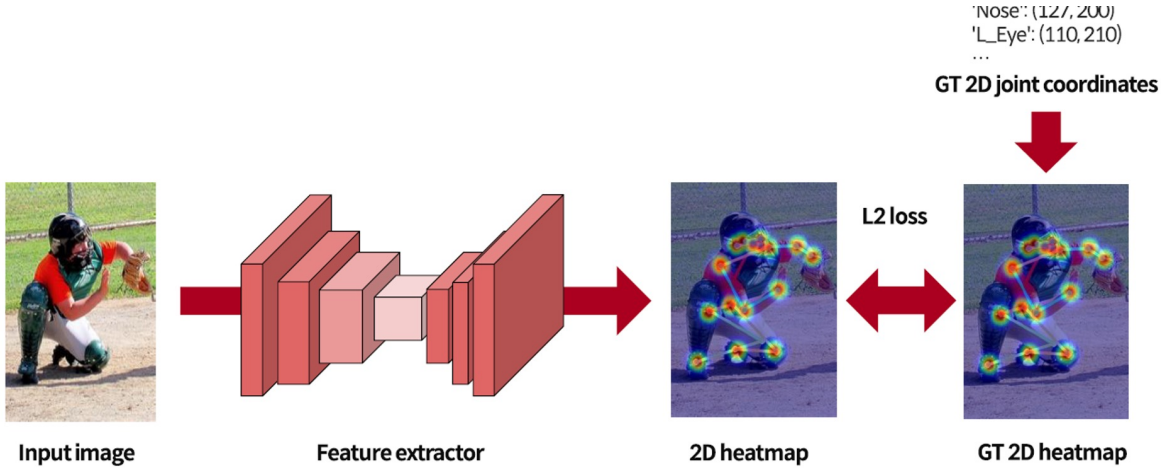


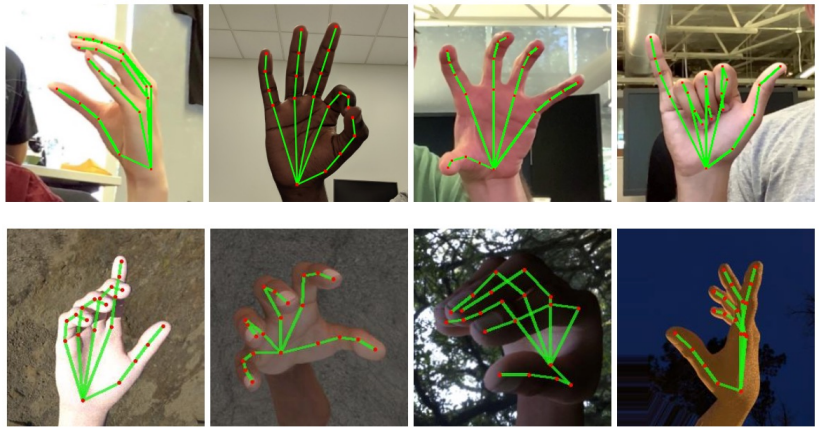


Computer vision tasks

- Pose estimation
  - https://www.youtube.com/watch?v=urvsIbiTmSg

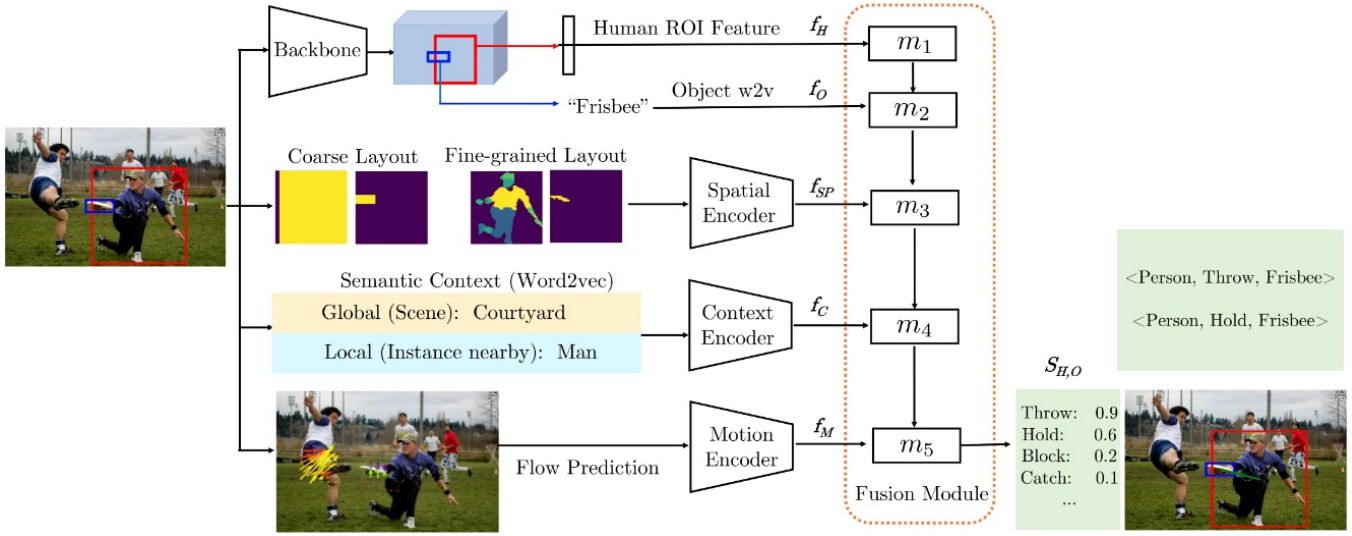


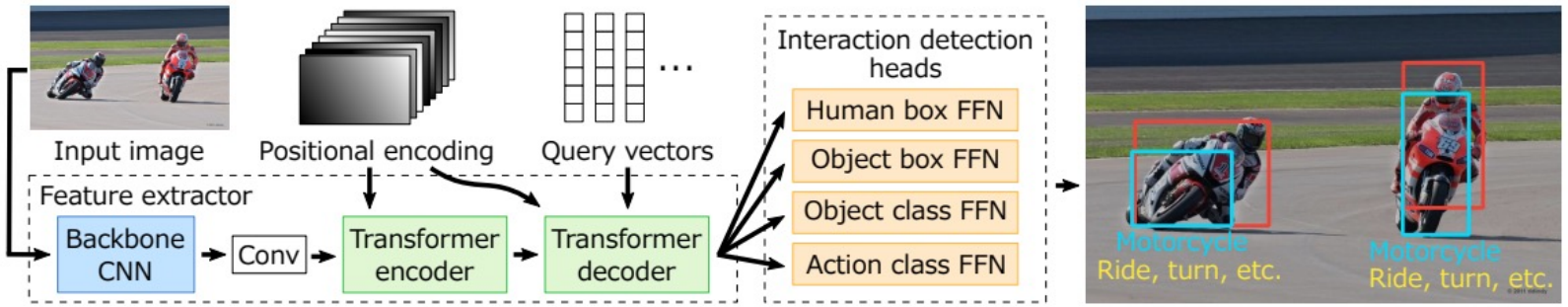




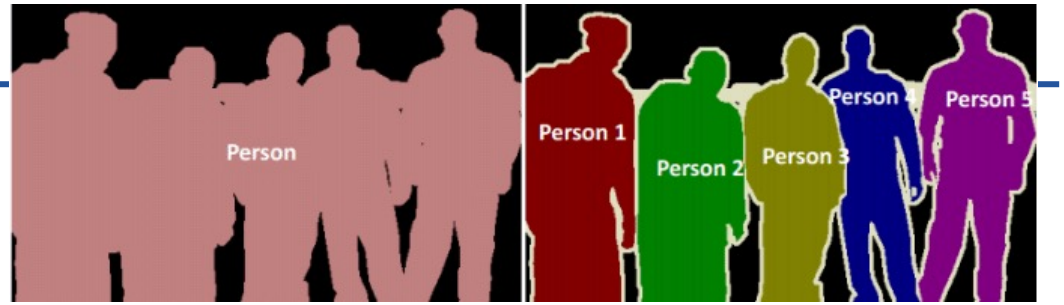
Computer vision tasks

- Human Object Interaction (HOI)
  - Recognizing the interactions between human and objects
  - Verb: holding, eating, taking, etc.
  - Object detection-based approach

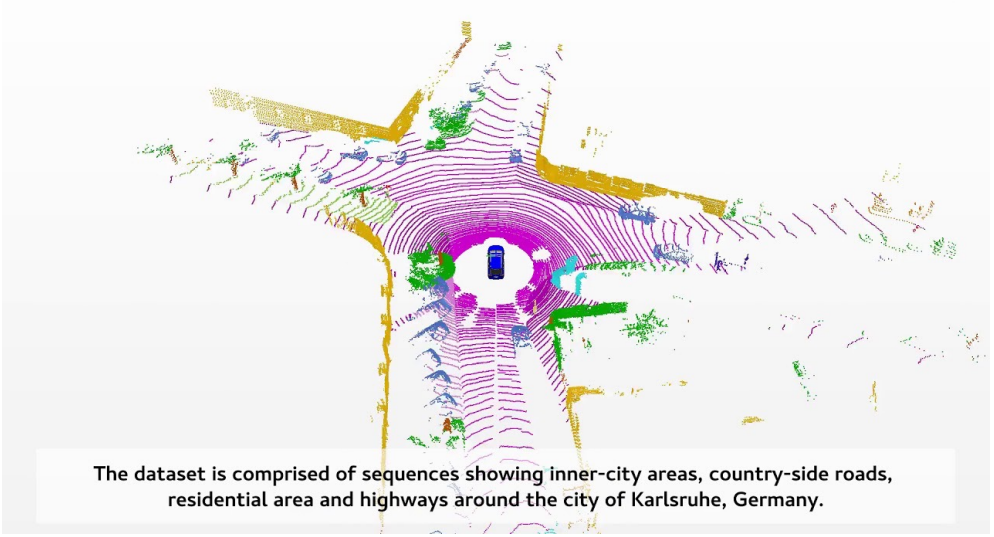




Computer vision tasks



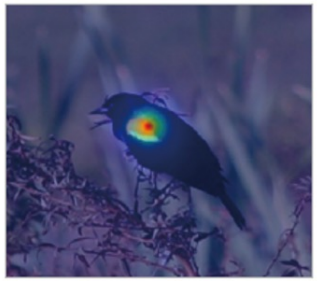
- Image segmentation
  - Semantic segmentation
    - Class가 같다면 구분하지 않고 같은 영역으로 취급
  - Instance segmentation
    - Class가 같더라도 다른 개체라면 이를 구분하여 인식
  - U-Net, Mask RCNN models, SemanticKITTI



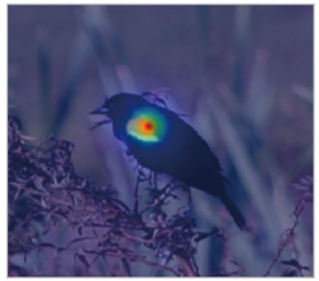
Computer vision tasks

- eXplainable AI (XAI)
  - 딥러닝의 한계: 분류 결과에 이유가 없다… (치명적)
  - “설명력”을 확보하기 위한 노력 ➔ CAM, GradCAM

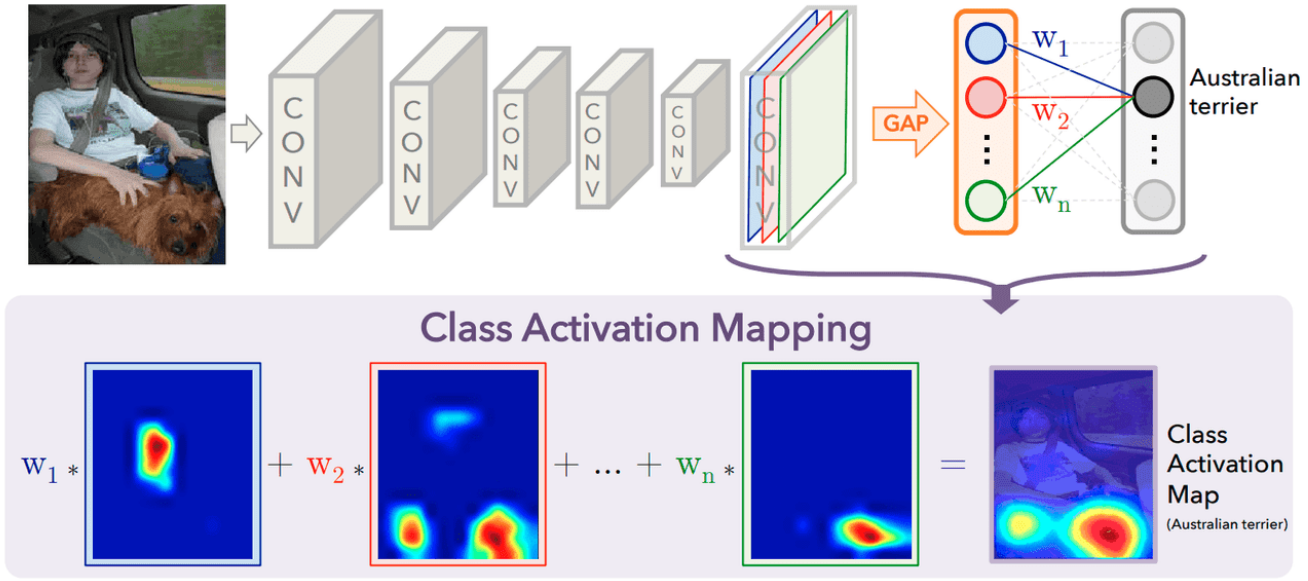
91.5% 확률로 붉은 날개 검은 새로 분류



하이라이트 영역을 보고 91.5% 확률로 붉은 날개 검은 새로 분류

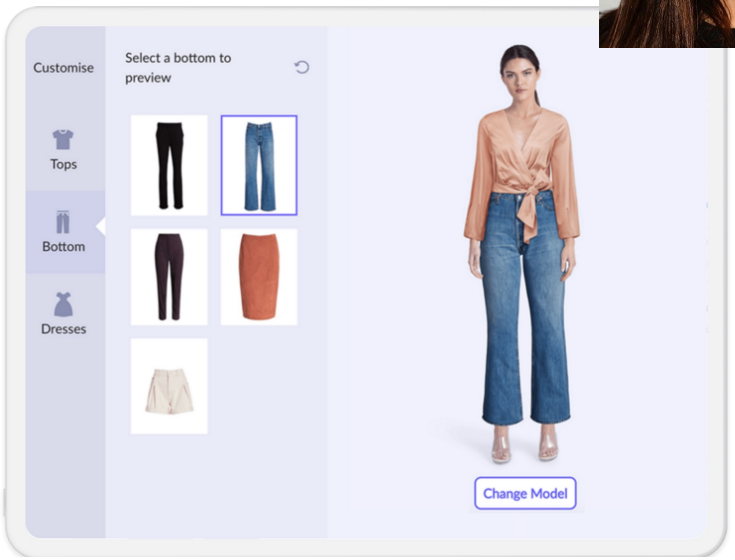


노란 띠와 붉은 반점을 보고 91.5% 확률로 붉은 날개 검은 새로 분류



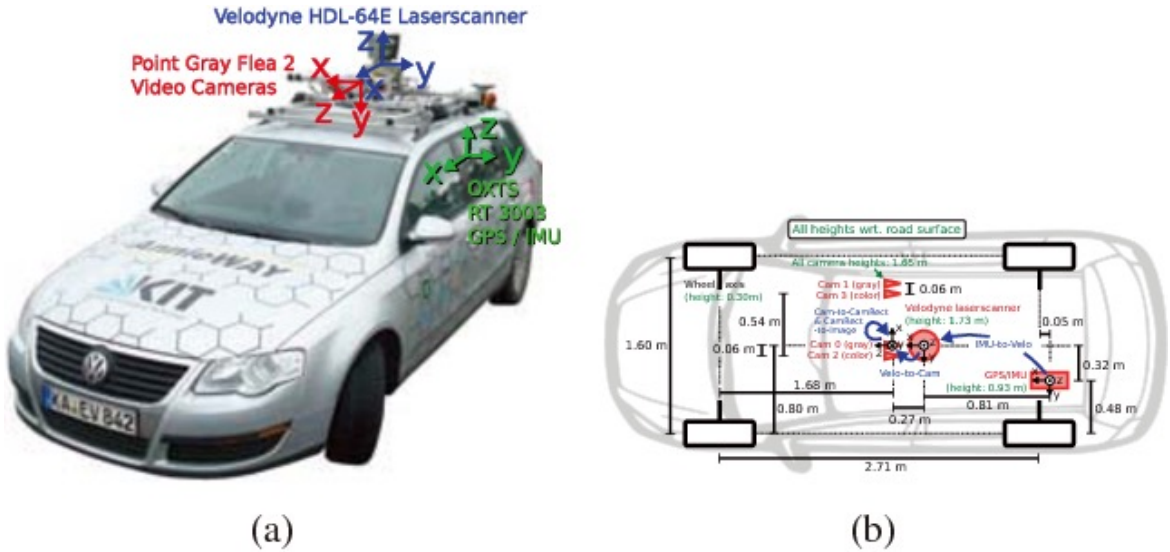
Computer vision tasks

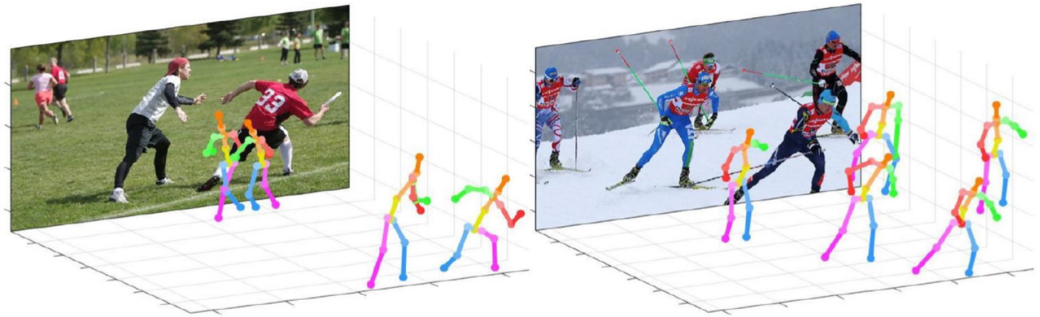
- Generative models
  - Generating new data samples that are similar to a tiven training dataset
  - GANs (Generative Adversarial Networks), VAE (Variational Autoencoder)

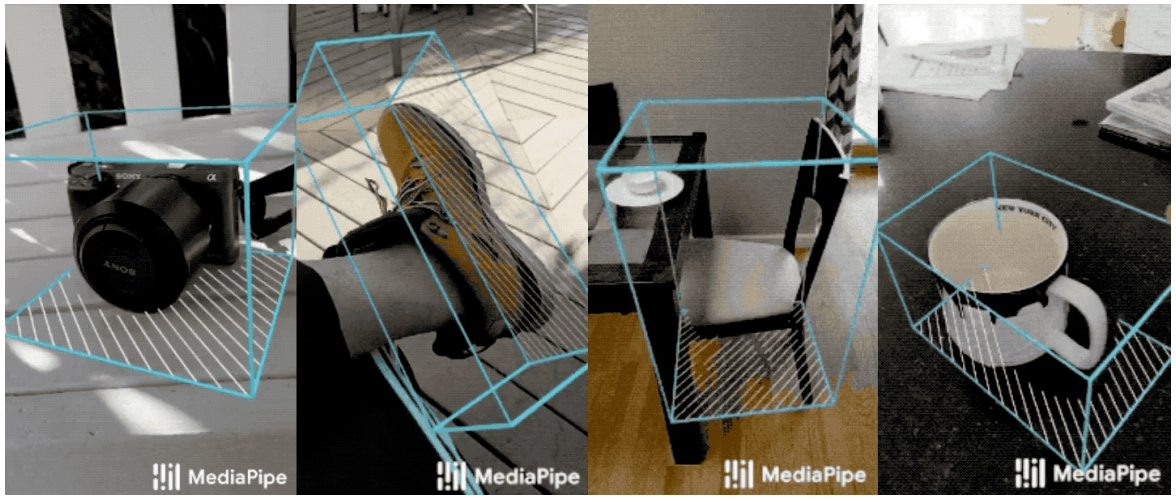


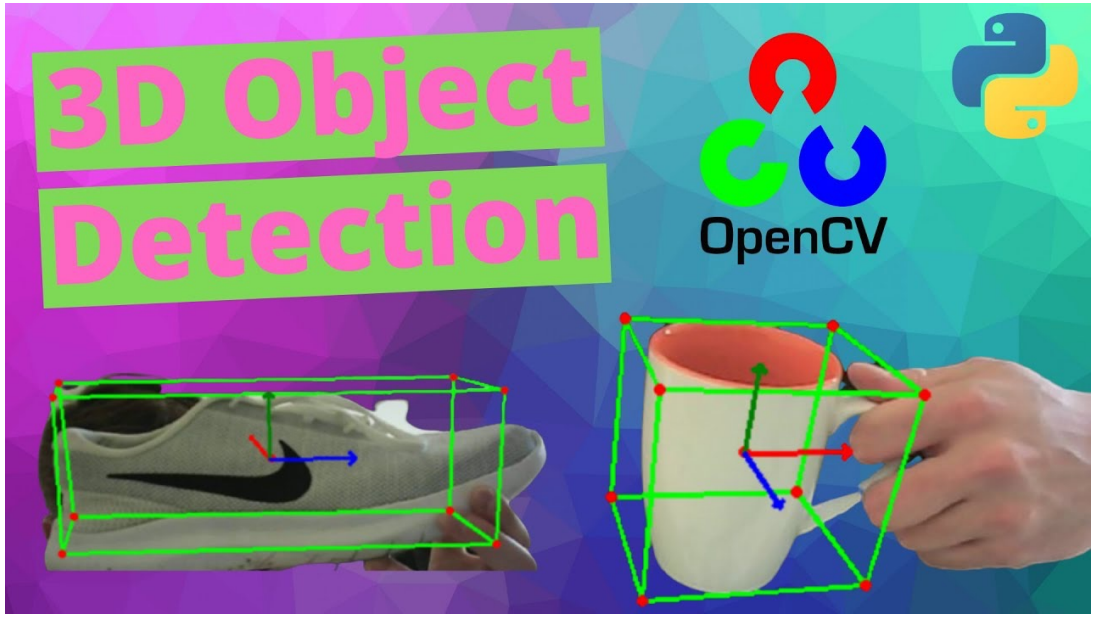
Computer vision tasks

- 3D recognition / reconstruction









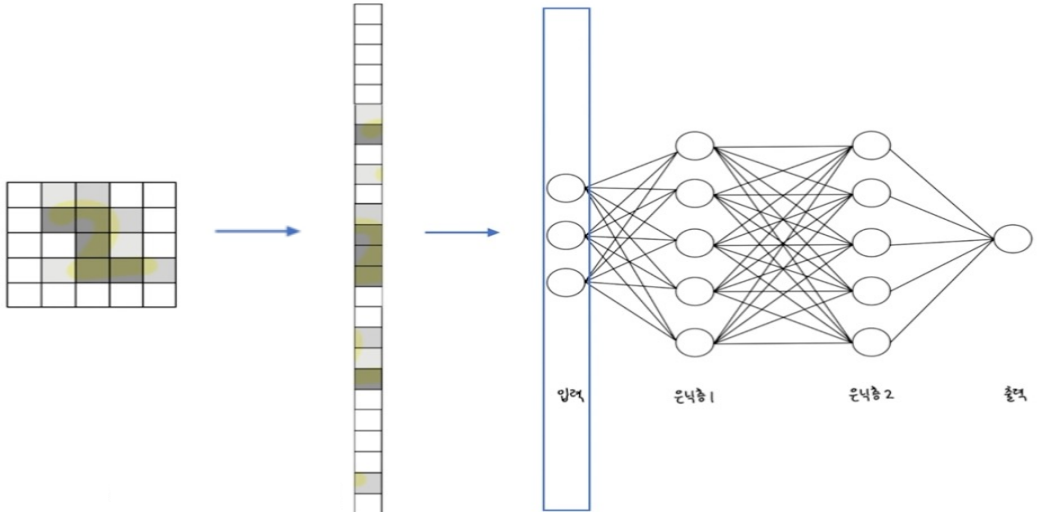
지금부터 살펴볼 내용

- Convolutional Neural Network
- Image dataset: PASCAL VOC, ImageNet, COCO
- Image classification models
  - AlexNet, VGGNet, ResNet, EfficientNet, ViT (Vision transformer)
- Evaluation metrics
  - IoU, Prevision, Recall, AP, mAP
- Object detection models
  - R-CNN models (Fast, Faster, Mask)
  - YOLOv1 ~ v8
    - The various deep learning techniques
- Tracking: SORT, DeepSORT

3. CNN 개요

CNN

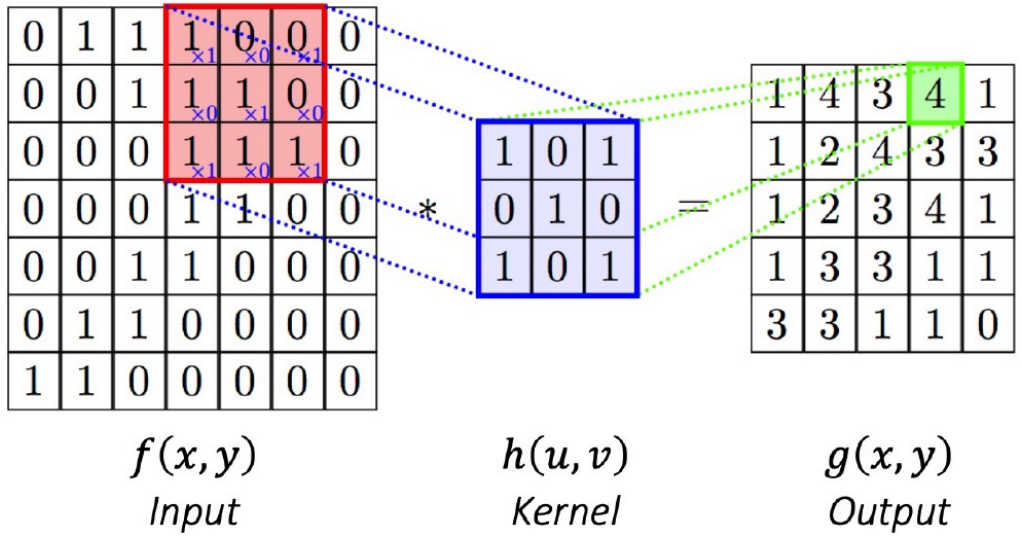
- 일반적인 신경망에 이미지 입력 시 지역적 위치 정보 소실



CNN

- Convolution operation

$$g = f * h; where\ g(x,y) = \sum_{u,v} f(x-u, y-v)h(u,v)$$



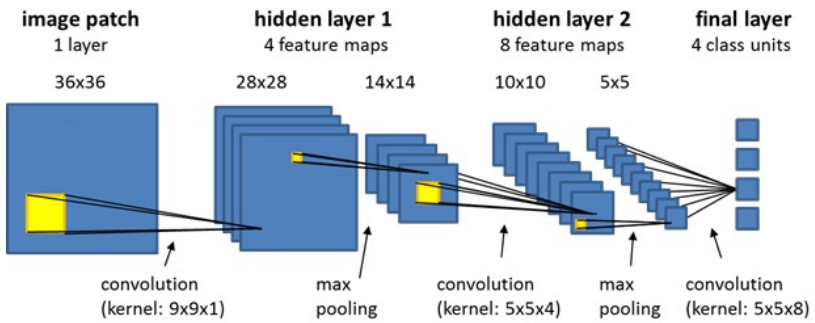
Smoothing

Sharpening

Gradient

CNN

- Basic architecture of a convolutional neural network (CNN)



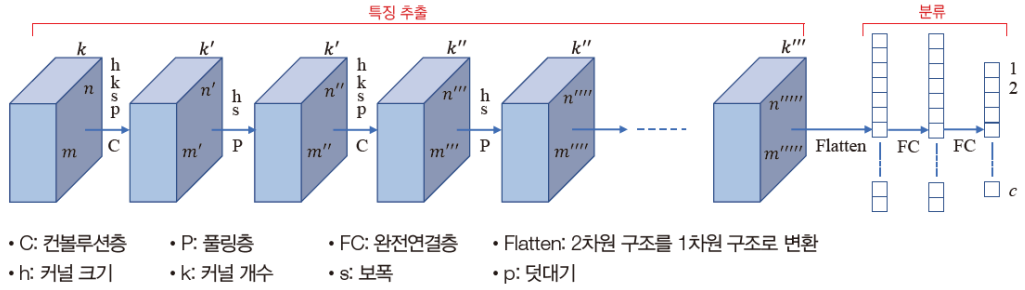
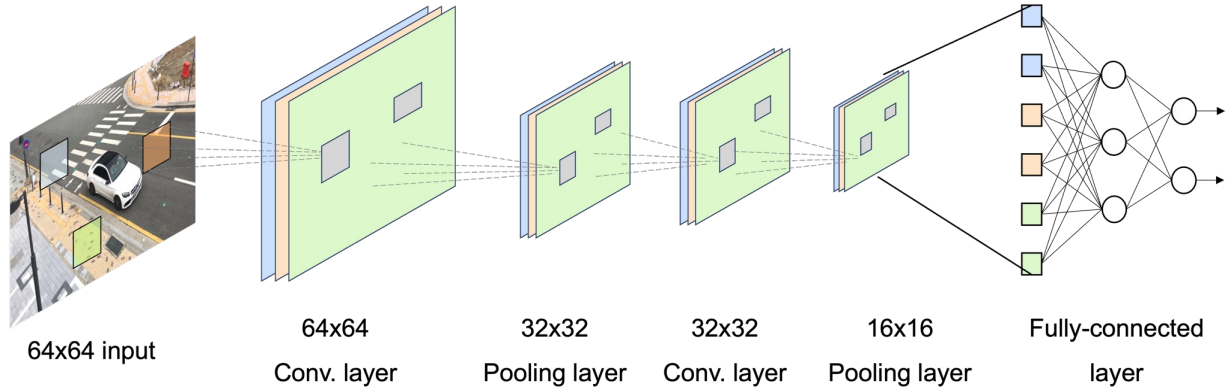


그림 8-10 컨볼루션층과 풀링층을 번갈아 쌓아 만드는 컨볼루션 신경망의 전형적인 구조



- CNN의 핵심 아이디어: "최적의 필터(kernel)을 학습으로 획득"
  - Deep neural network의 weight = CNN의 kernel

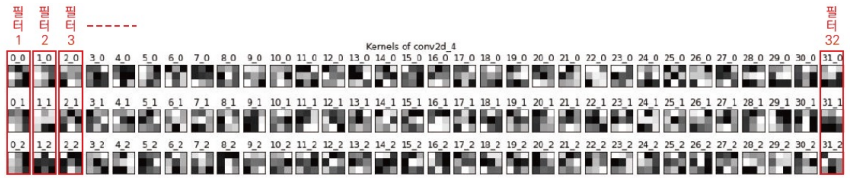
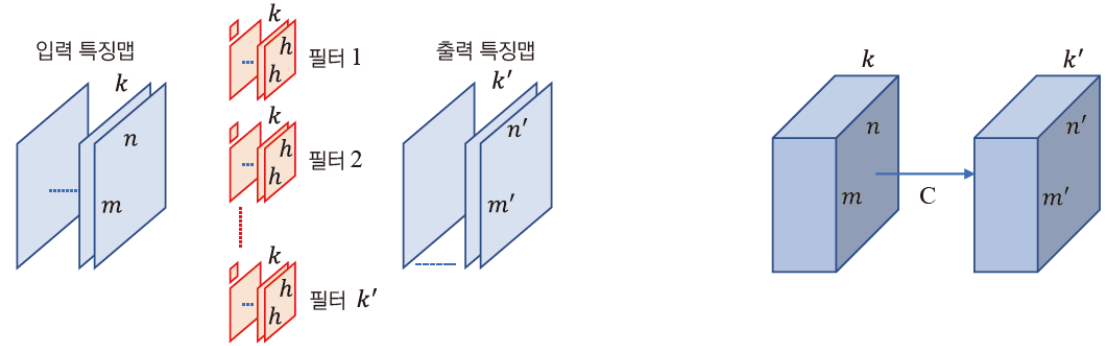


그림 8-14 CIFAR-10 데이터셋으로 학습한 컨볼루션 신경망의 최적 필터

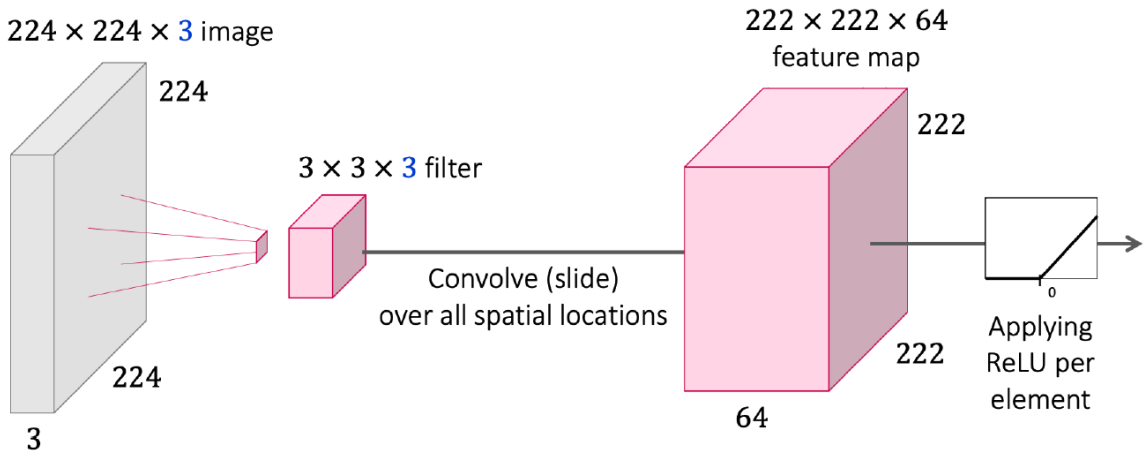
CNN

- Layer operation 1 – convolutional layers
  - 입력 특징맵(feature map)이 $m \times n \times k$ 크기의 tensor
    ➔ $h \times h \times k$ 크기의 필터(커널) tensor 필요
  - 출력 특징맵은 $m' \times n' \times k'$
    - ➔ padding과 stride로 계산할 수 있음



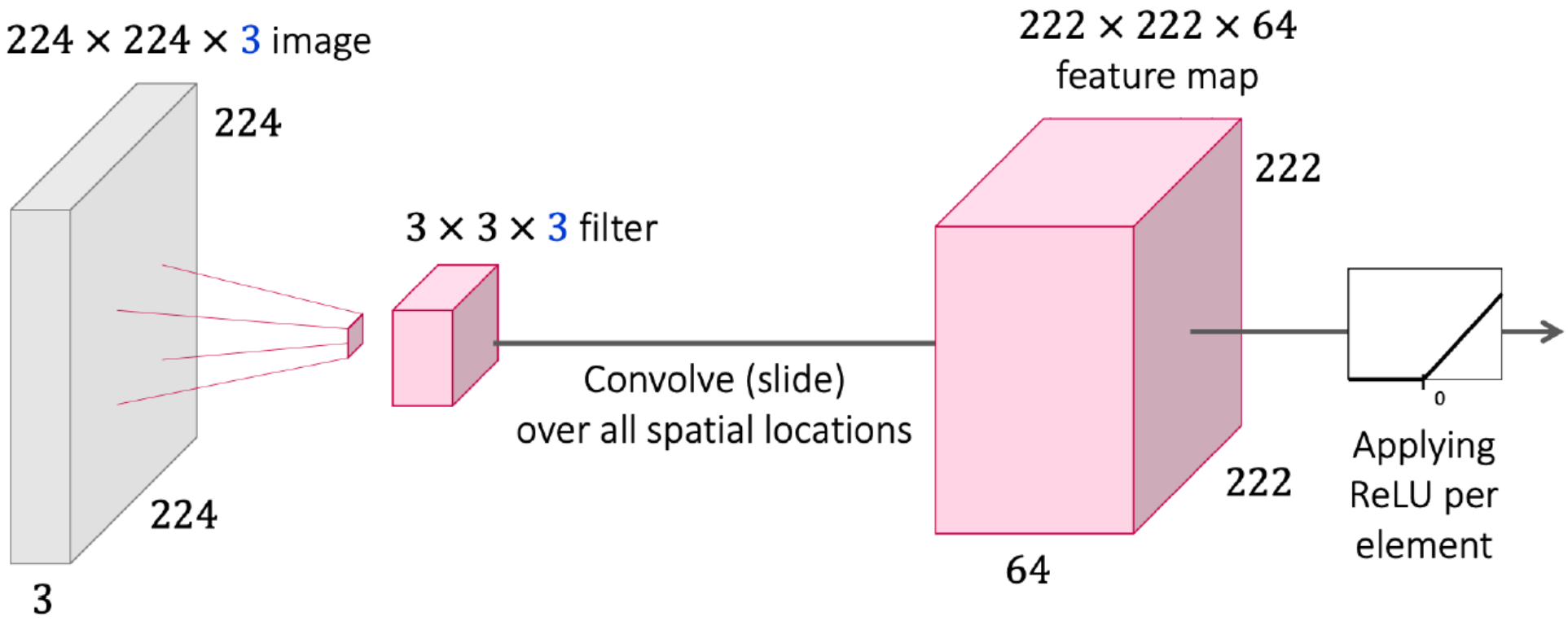
(a) 세부 내용 (b) 간결한 블록 표현

그림 8-6 컨볼루션층



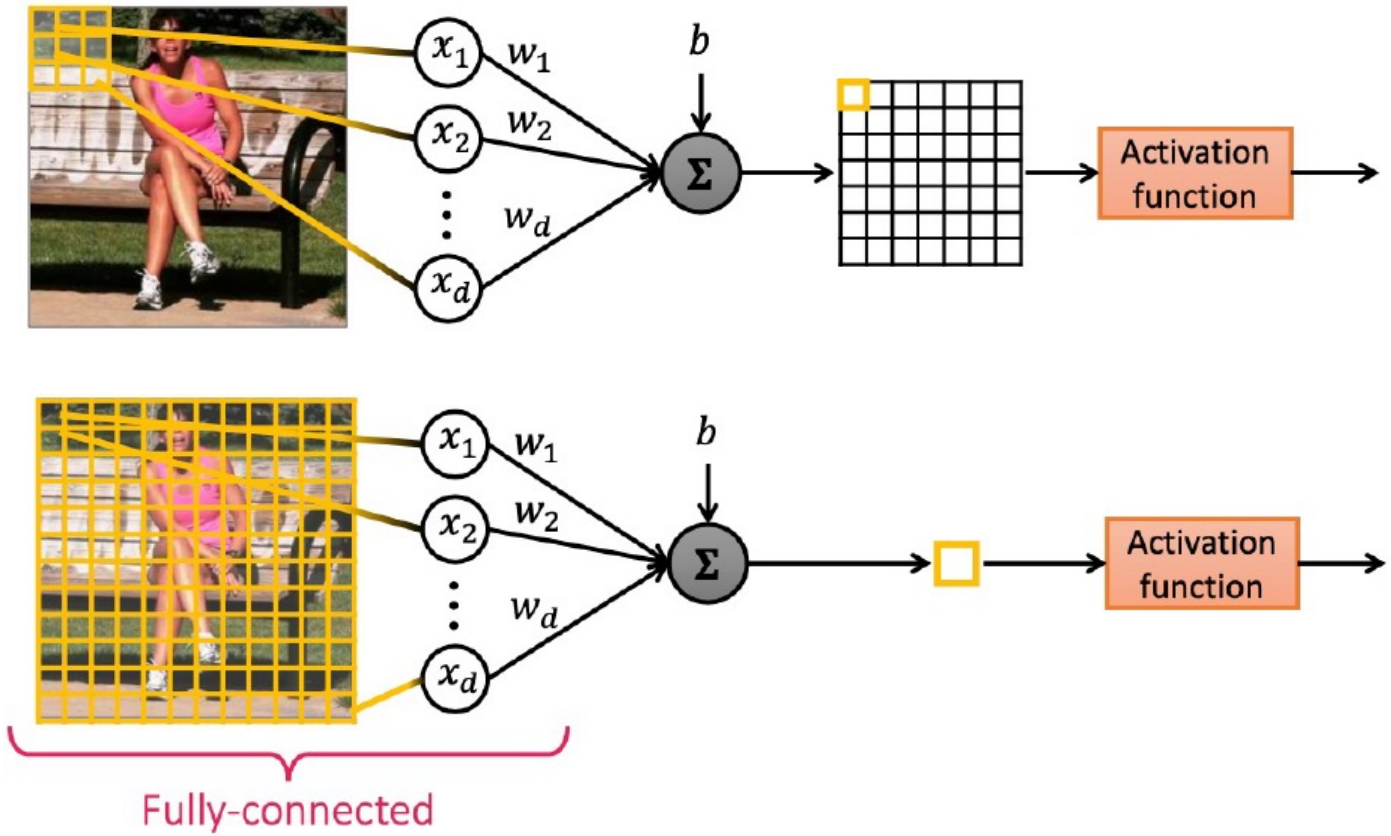
CNN

- Layer operation 1 – convolutional layers



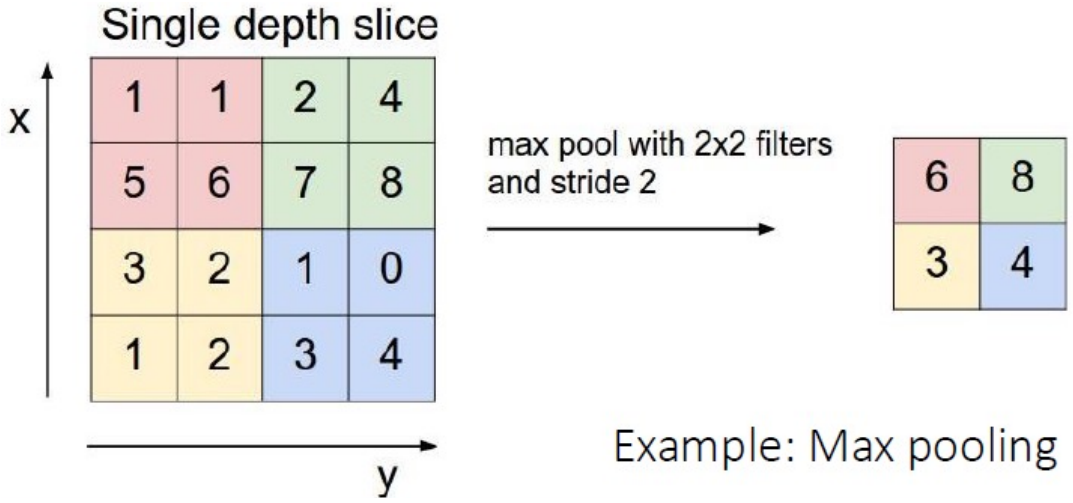
CNN

- Kernel size의 역할
  - 적당히 작은 크기의 kernel (3x3x1) vs input과 같은 크기의 kernel



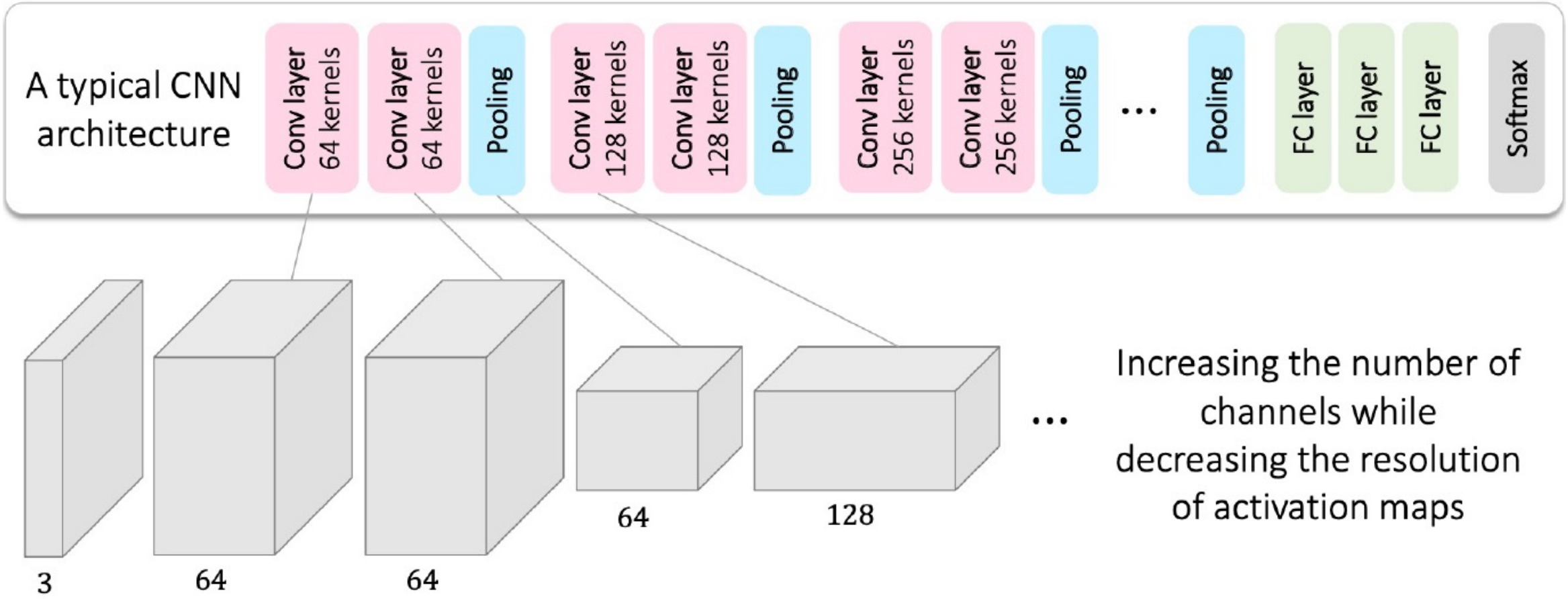
CNN

- Layer operation 2 – Pooling operations
  - pooling types
    - max pooling
    - average pooling
    - L2-norm pooling, etc.
  - 역할
    - 지역적 불변성 달성
    - 이미지 정보의 요약
    - 차원 축소



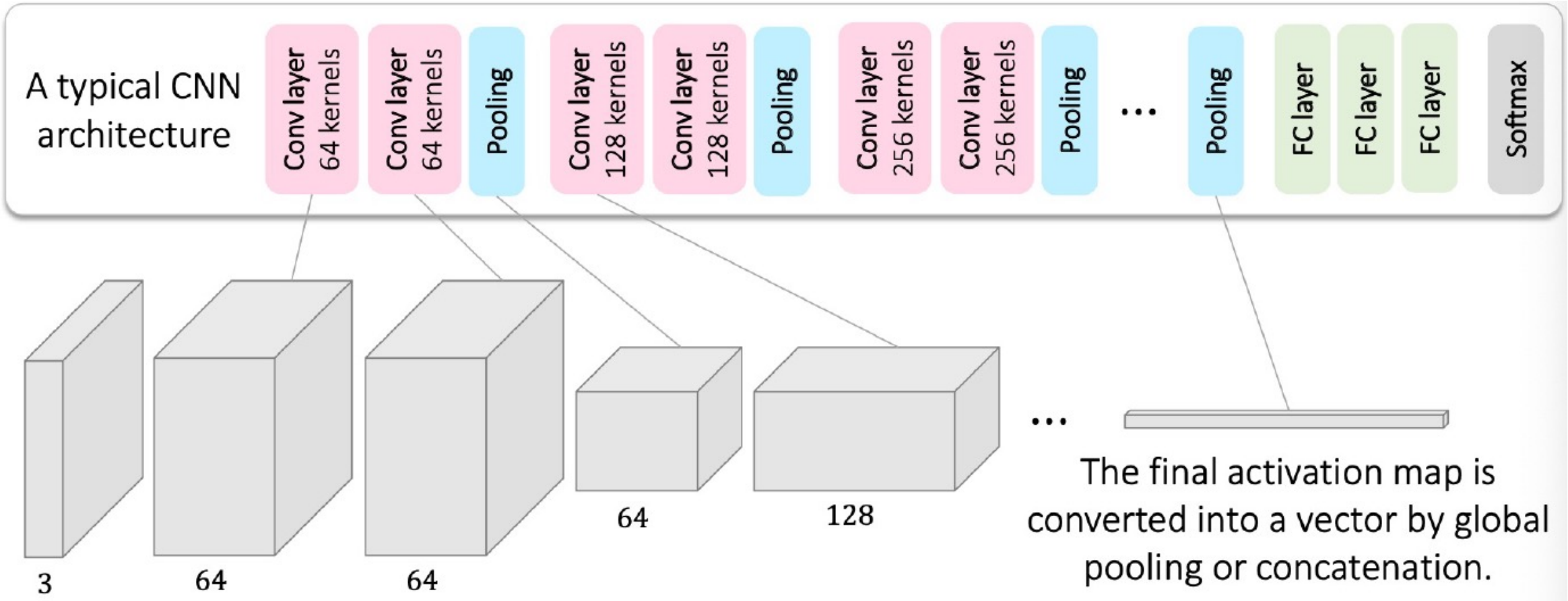
CNN

- Layer operations



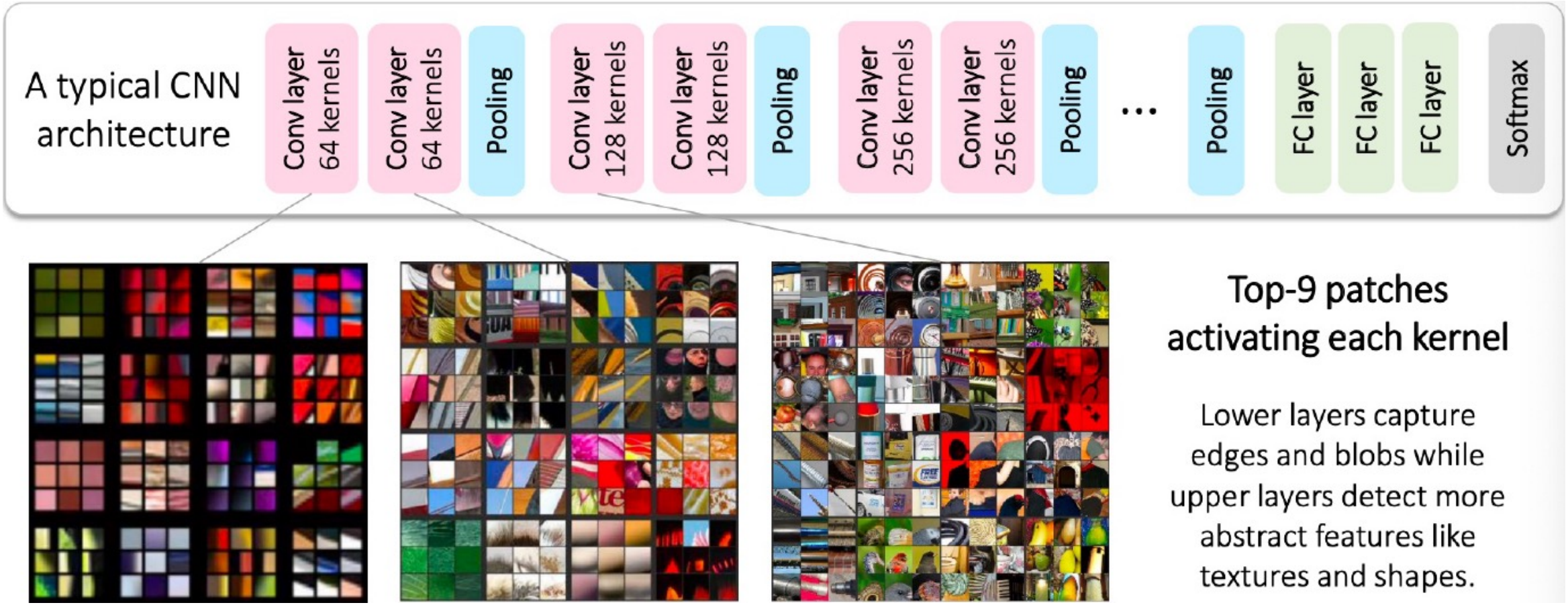
CNN

- Layer operations



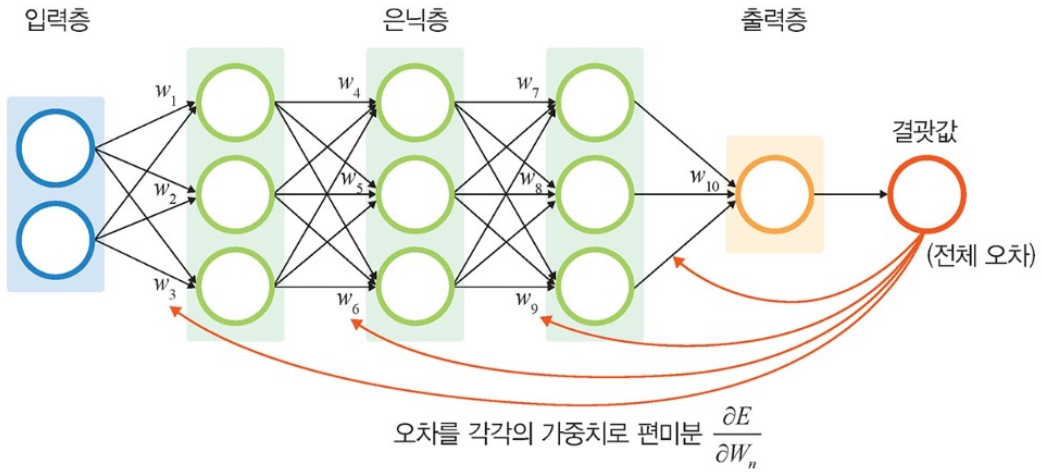
CNN

- Layer operations



(참고) feed-forward and back-propagation

- General DNN



- CNN

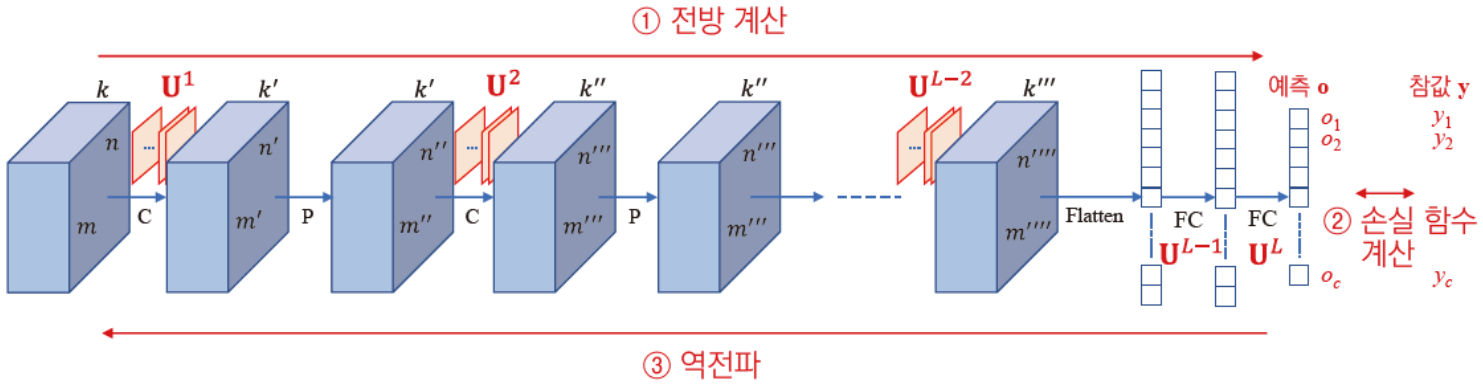


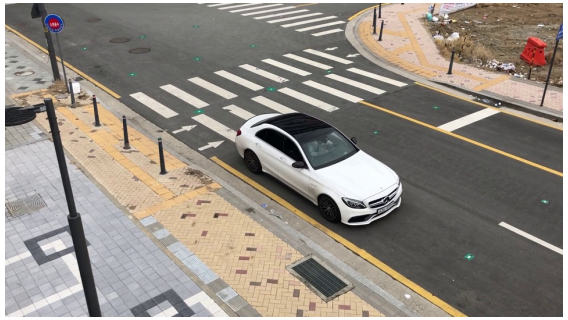
그림 8-13 컨볼루션 신경망을 학습하기 위한 역전파 알고리즘

4. Image Classification

Problems

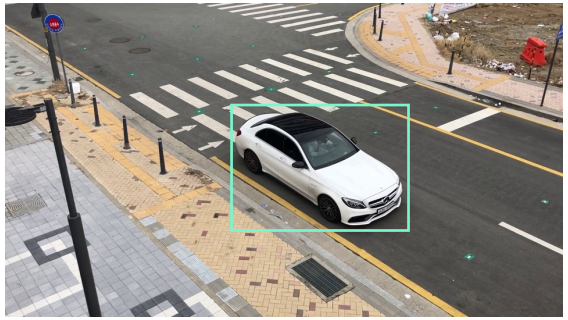
- 일반적인 신경망에 이미지 입력 시 지역적 위치 정보 소실

Classification



Car

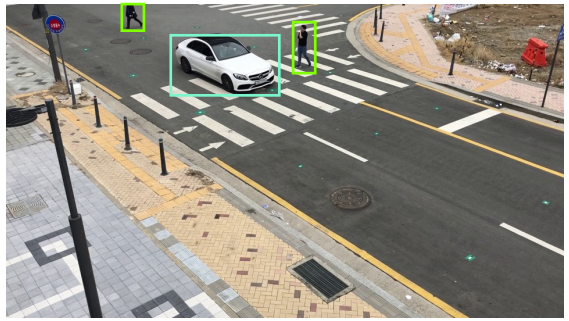
Classification + Localization



Car

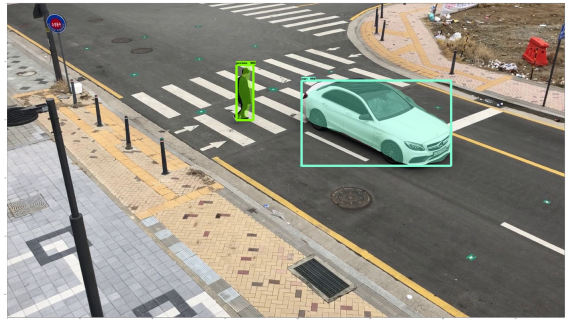
Single object

Object Detection



Car, Person

Instance Segmentation



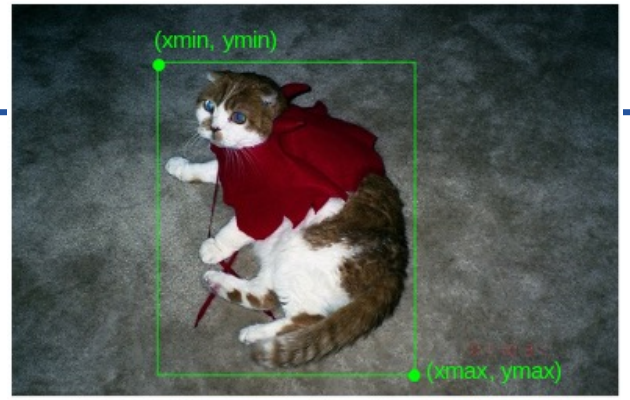
Car, Person

Multiple objects

Dataset

- ImageNet
  - Professor Fei-Fei Li, Stanford University
  - 2009년 공개
  - For image classification
    - 1 image – 1 label
    - WordNet 계층구조를 갖는 라벨 분류
  - ImageNet: 21,841 classes, 1,400만장
    - ILSVRC 2012
      - # of images: 1,400,000 images
      - # of classes: 1000 classes

Dataset

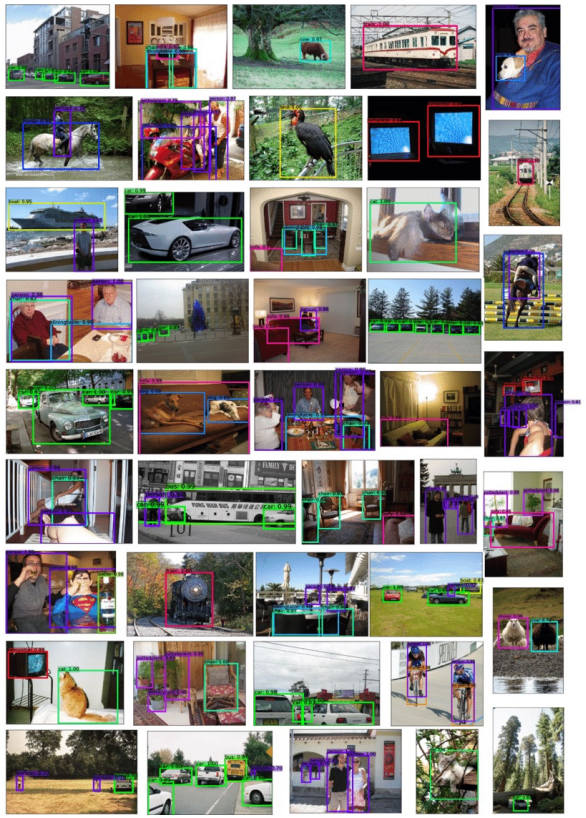


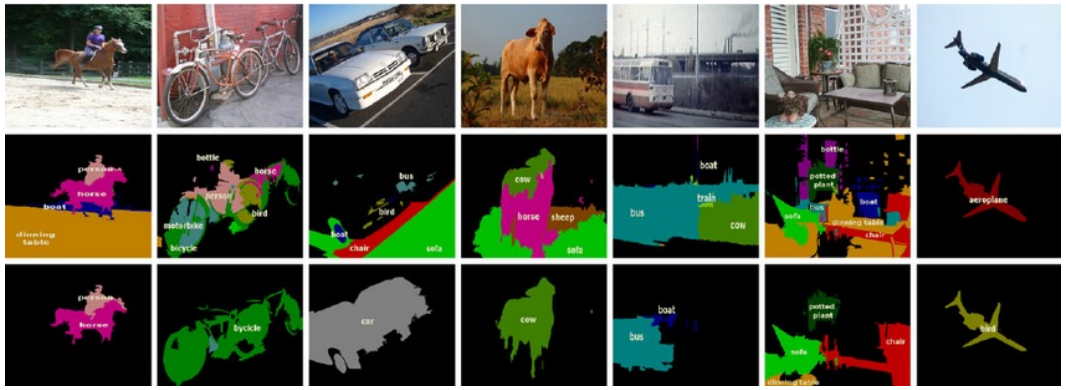
- PASCAL VOC (Version: PASCAL VOC 2007 / 2012)
  - For image classification and object detection
  - Format: XML format
  - Annotations
    - <bndbox>: bounding box coordinates (xmin, ymin, xmax, ymax)
    - <name>: object's class
    - <pose>: pose of the object (optional)
    - <truncated>: whether the object is truncated (0 or 1, optional)
    - <difficult>: flag indicating if the object is difficult to recognize (0 or 1, optiona

```
<annotation>
    <folder>VOC2007</folder>
    <filename>000001.jpg</filename>
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
        <flickrid>341012865</flickrid>
    </source>
    <owner>
        <flickrid>Fried Camels</flickrid>
        <name>Jinky the Fruit Bat</name>
    </owner>
    <size>
        <width>353</width>
        <height>500</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>dog</name>
        <pose>Left</pose>
        <truncated>1</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>48</xmin>
            <ymin>240</ymin>
            <xmax>195</xmax>
            <ymax>371</ymax>
        </bndbox>
    </object>
    <object>
        <name>person</name>
        <pose>Left</pose>
        <truncated>1</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>8</xmin>
            <ymin>12</ymin>
            <xmax>352</xmax>
            <ymax>498</ymax>
        </bndbox>
    </object>
</annotation>
```

Dataset

- PASCAL VOC
  - Object classes: 20 or 21 (depending on the version)
    - Including common objects; car, cat, dogs, and more
  - Segmentation
    - Only PASCAL VOC 2012
    - Each pixels in image is labeled with the class it belongs to
  - Dataset size
    - Object detection: 9,963 annotations
      - Train dataset: 5,011 annotations
    - Avg. # of objects in image: 2.4
    - Avg. # of classes in image: 1.4





Dataset

- COCO (Latest version: COCO2017, https://cocodataset.org/#home)
  - For object detection, segmentation, captioning
  - Format: json
  - Annotations
    - id / image_id: Identifier for annotation
    - annotations.bbox: bbox coordinates (xmin, ymin, width, height)
    - categories.name: object's class
  - Object classes: 80 (91 for segmentation)
    - Including common objects; car, cat, dogs, and more
  - Segmentation
    - Provide instance segmentation annotations
      - Refer annotation.segmentation

```
{
    "info": {...},
    "licenses": [...],
    "images": [
        {
            "id": ImageID,
            "width": ImageWidth,
            "height": ImageHeight,
            "file_name": "ImageFileName.jpg",
            "license": LicenseID,
            "flickr_url": "FlickrURL",
            "coco_url": "CocoURL",
            "date_captured": "DateCaptured"
        },
        ...
    ],
    "annotations": [
        {
            "id": AnnotationID,
            "image_id": ImageID,
            "category_id": CategoryID,
            "segmentation": [...],
            "area": ObjectArea,
            "bbox": [XMin, YMin, Width, Height],
            "iscrowd": IsCrowd,
            "attributes": {...},
        },
        ...
    ],
    "categories": [
        {
            "id": CategoryID,
            "name": "ObjectClassName",
            "supercategory": "SuperCategoryName"
        },
        ...
    ]
}
```

Dataset

- COCO
  - Dataset size
    - Training dataset: 118,000 images
    - Validation dataset: 5,000 images
    - Test dataset: 41,000 images
    - Avg. # of objects in image: 8
    - Avg. # of classes in image: 3.5

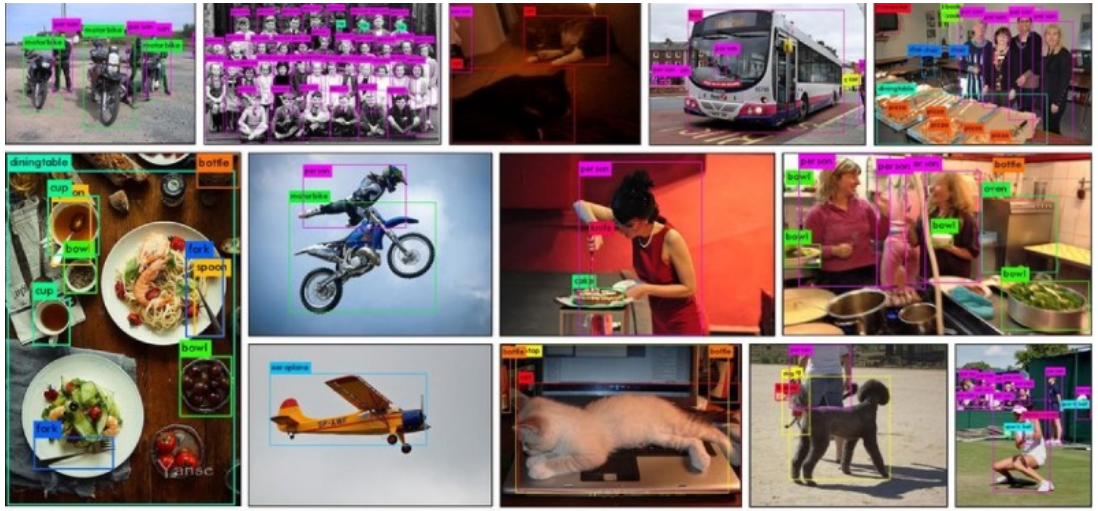
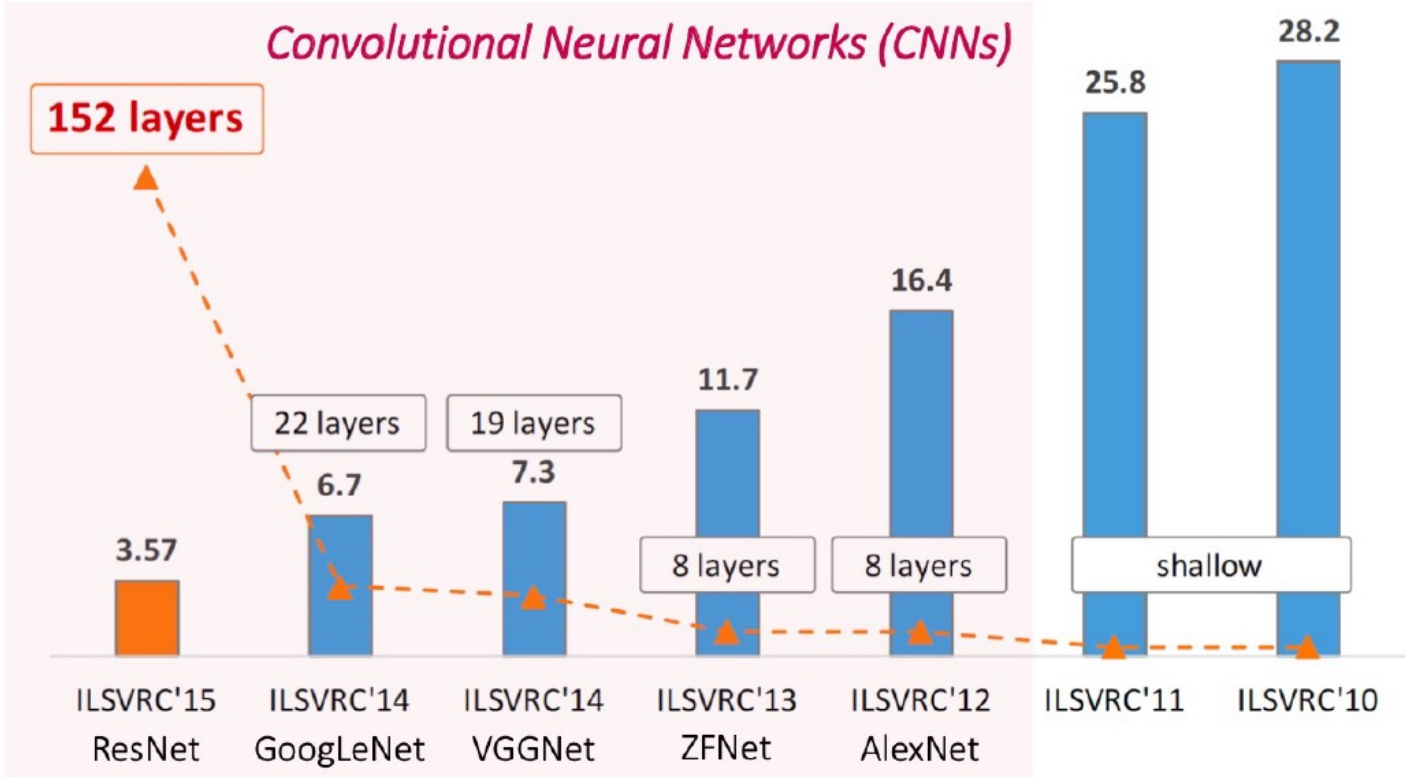


Image classification model

- ImageNet Large Scale Classification Challenge: top-5 error (%)
  - 2015 winner: ResNet (Residual Network)



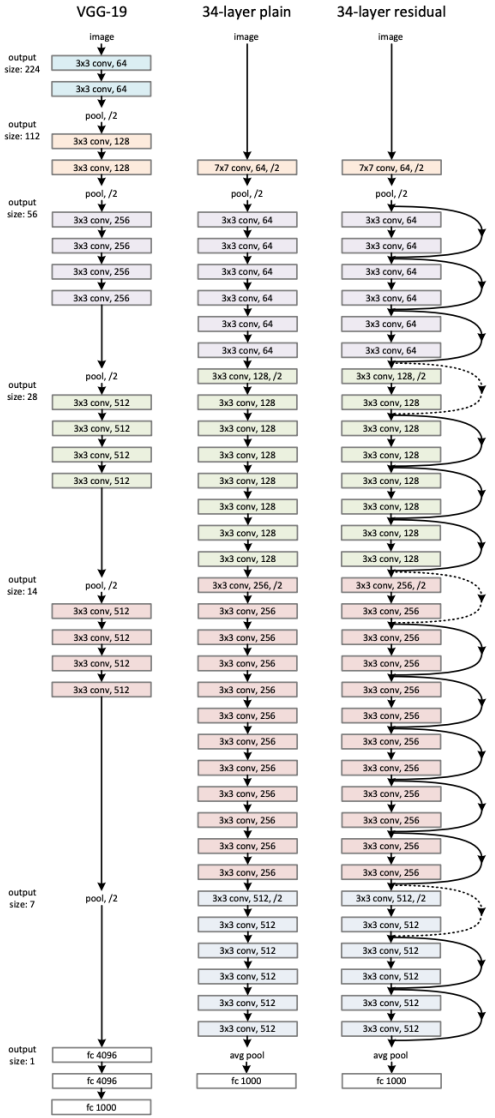
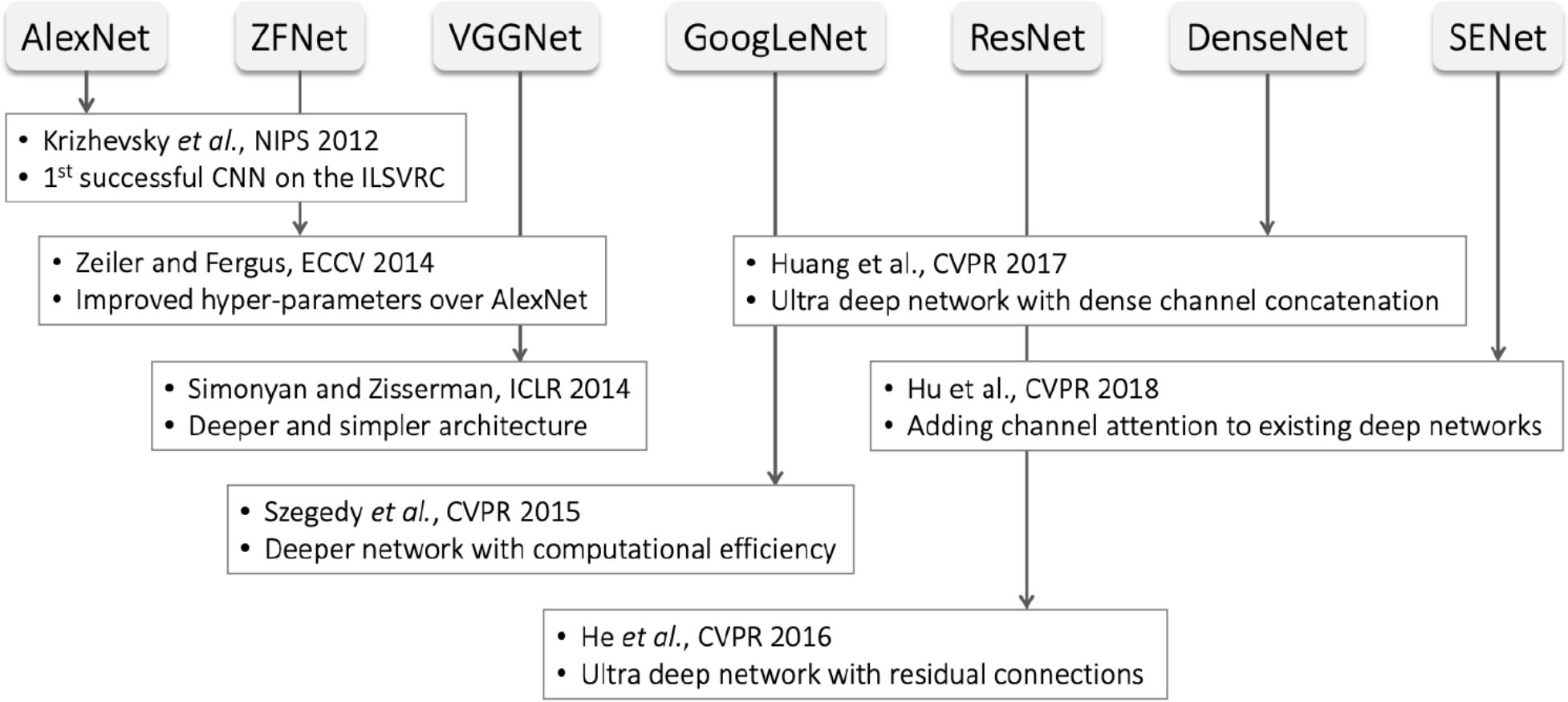


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

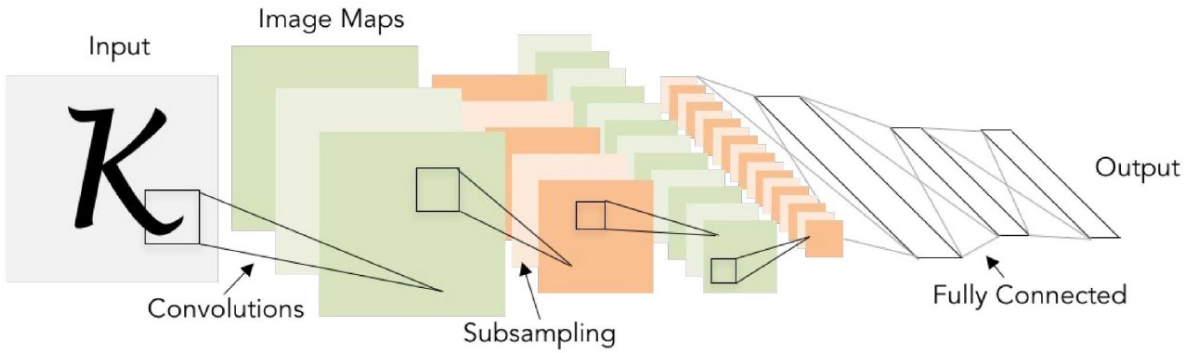
Image classification model

- History of the well-known CNN-based image classification model
  - + EfficientNet (ICML 2019)



LeNet-5

- The first CNN model for image classification
- LeNet-5: Gradient-based Learning Applied to Document Recognition (1988)
  - very simple CNN architecture by Yann Lecun
    - Yan Lecun: Facebook AI Research (FAIR), Geoffrey Hinton과 함께 2018년 Turing award 수상
  - 5x5 kernel with stride 1
  - 2x2 maxpooling with stride 2



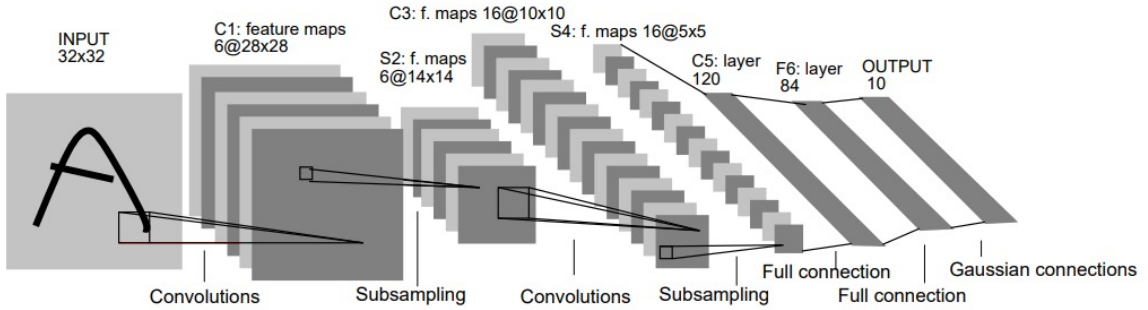


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

프로그래밍 실습: LeNet-5의 재현

- LeNet-5 직접 구현하고 MNIST 인식

프로그램 8-1 LeNet-5로 MNIST 인식하기

```
01  import numpy as np
02  import tensorflow as tf
03  import tensorflow.keras.datasets as ds
04
05  from tensorflow.keras.models import Sequential
06  from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dropout,Dense
07  from tensorflow.keras.optimizers import Adam
08
09  (x_train,y_train),(x_test,y_test)=ds.mnist.load_data()
10  x_train=x_train.reshape(60000,28,28,1)
11  x_test=x_test.reshape(10000,28,28,1)
12  x_train=x_train.astype(np.float32)/255.0
13  x_test=x_test.astype(np.float32)/255.0
14  y_train=tf.keras.utils.to_categorical(y_train,10)
15  y_test=tf.keras.utils.to_categorical(y_test,10)
16
```

데이터 준비

프로그래밍 실습: LeNet-5의 재현

- LeNet-5 직접 구현하고 MNIST 인식

```
17  cnn=Sequential()
18  cnn.add(Conv2D(6,(5,5),padding='same',activation='relu',input_
    shape=(28,28,1)))
19  cnn.add(MaxPooling2D(pool_size=(2,2),strides=2))
20  cnn.add(Conv2D(16,(5,5),padding='valid',activation='relu'))
21  cnn.add(MaxPooling2D(pool_size=(2,2),strides=2))
22  cnn.add(Conv2D(120,(5,5),padding='valid',activation='relu'))
23  cnn.add(Flatten())
24  cnn.add(Dense(units=84,activation='relu'))
25  cnn.add(Dense(units=10,activation='softmax'))
26
27  cnn.compile(loss='categorical_crossentropy',optimizer=Adam
    (learning_rate=0.001),metrics=['accuracy'])
28  cnn.fit(x_train,y_train,batch_size=128,epochs=30,validation_
    data=(x_test,y_test),verbose=2) ①
29
30  res=cnn.evaluate(x_test,y_test,verbose=0)
31  print('정확률=',res[1]*100) ②
```

컨볼루션층(5*5 커널을 6개 사용)

1차원 구조로 변환

모델 선택 (신경망 구조 설계)

학습

예측(성능 측정)

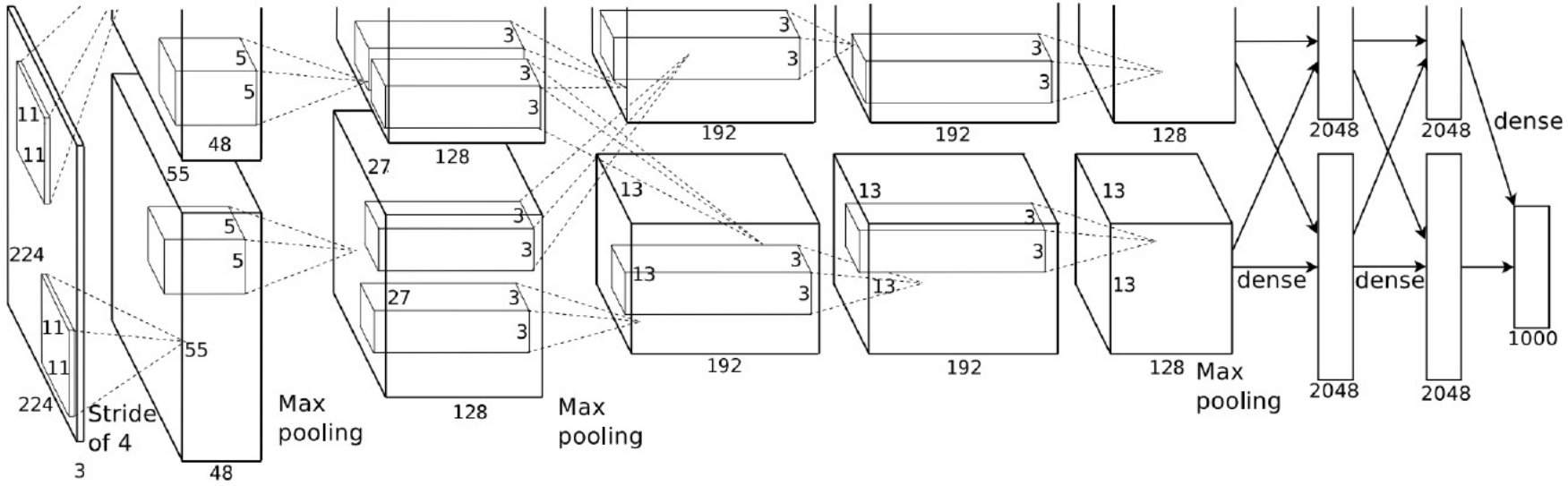
프로그래밍 실습: LeNet-5의 재현

- LeNet-5 직접 구현하고 MNIST 인식

```
Epoch 1/30 ①
469/469 - 3s - loss: 0.2555 - accuracy: 0.9212 - val_loss: 0.0794 - val_accuracy:
0.9742 - 3s/epoch - 6ms/step
Epoch 2/30
469/469 - 2s - loss: 0.0671 - accuracy: 0.9786 - val_loss: 0.0536 - val_accuracy:
0.9826 - 2s/epoch - 4ms/step
…
Epoch 29/30
469/469 - 2s - loss: 0.0037 - accuracy: 0.9988 - val_loss: 0.0506 - val_accuracy:
0.9902 - 2s/epoch - 5ms/step
Epoch 30/30
469/469 - 2s - loss: 0.0059 - accuracy: 0.9981 - val_loss: 0.0430 - val_accuracy:
0.9910 - 2s/epoch - 5ms/step
정확률= 99.09999966621399 ②
```

AlexNet

- Architecture (Similar with LeNet-5)
  - bigger (7 hidden layers, 650K neurons, 60 millions parameters)
  - trained with a larger amount of data (ImageNet)
  - conv-pool-LRN-conv-pool-LRN-conv-conv-conv-pool-FC-FC-FC
    - LRN: Local Response Normalization



AlexNet

- Strategy 1 - overlapped max pooling

4	3	5	5
4	1	4	9
3	2	0	1
5	2	4	6

Stride 2

4	9
5	6

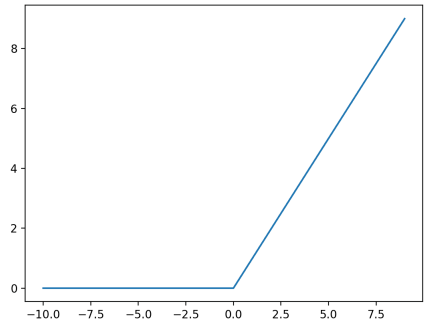
1	3	5	5
4	1	4	9
3	2	0	1
5	2	4	6

Stride 1

4	5	9
4	4	9
5	4	6

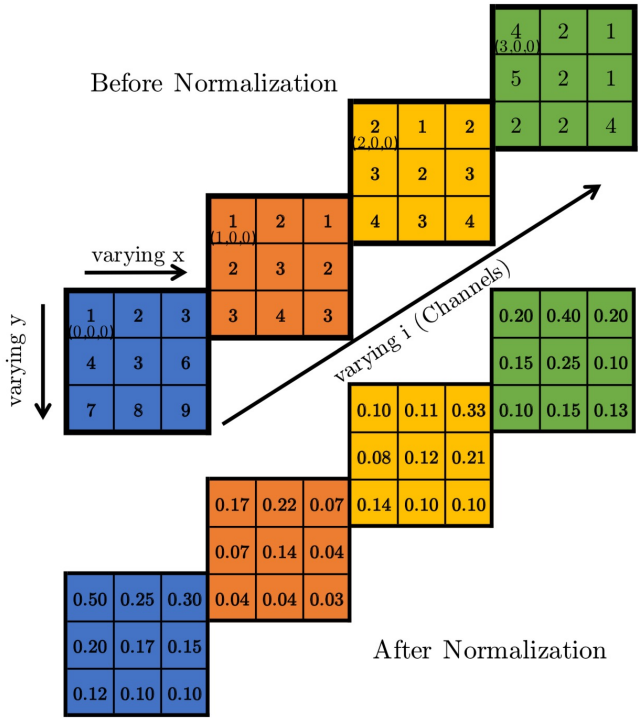
AlexNet

- Strategy 2 – Rectified Linear Unit (ReLU)



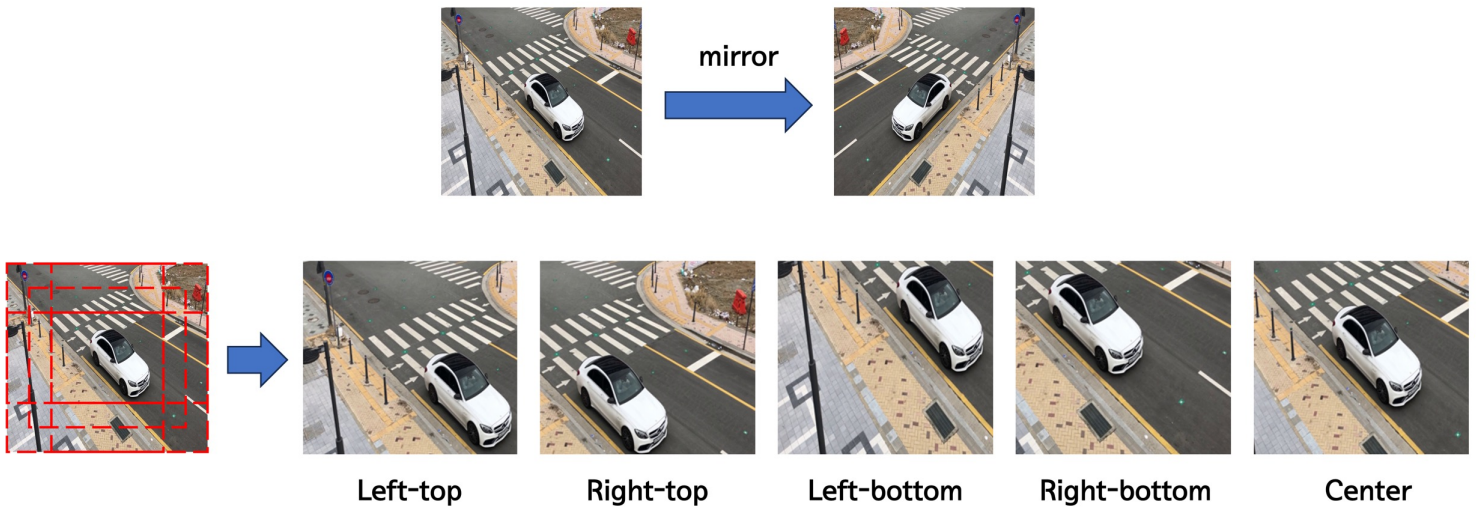
- Strategy 3 – Local Response Normalization (LRN) ➔ Kernel 정규화

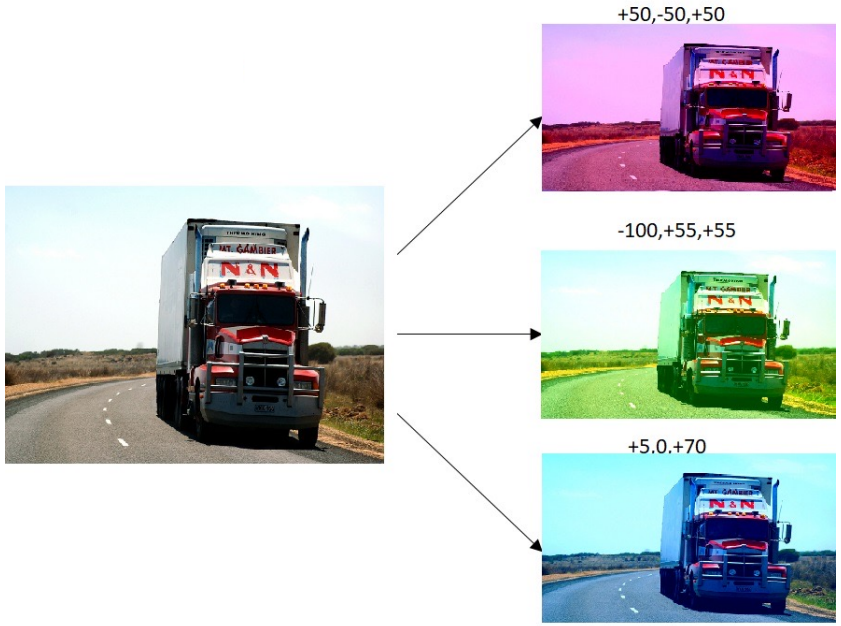
  - $b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max\left(0, i-\frac{n}{2}\right)}^{j=\min\left(N-1, i+\frac{n}{2}\right)} {a_{x,y}^{j}}^{2} \right)^{\beta}$
  - $b_{x,y}^{i}$ – regularized output for kernel i at position x, y
  - $a_{x,y}^{i}$ – source output of kernel i applied at position x, y
  - N – total number of kernels
  - n – size of normalization neighborhood
  - $\alpha, \beta, k, (n)$ - hyperparameters



AlexNet

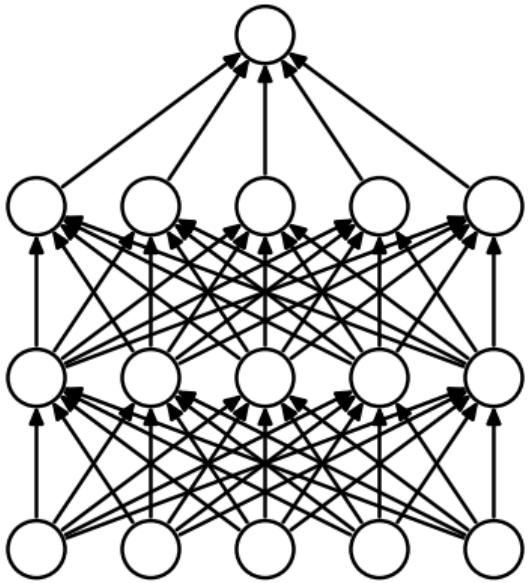
- Strategy 4 – Data augmentation
  - overfitting 예방 전략
    - PCA color augmentation
      - PCA를 활용하여 이미지의 특성은 유지하고 RGB color값을 이동 및 변경하는 기법



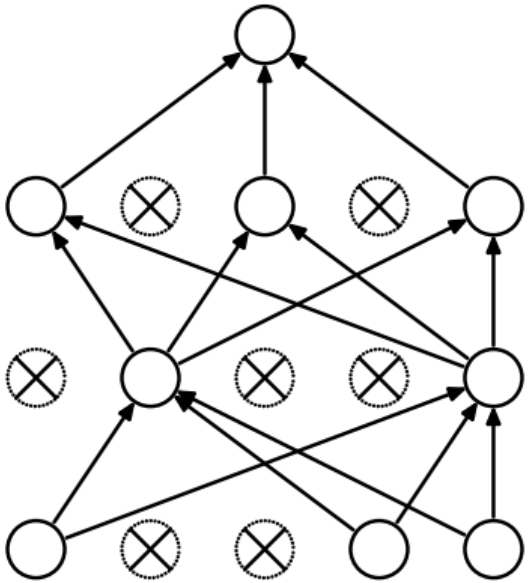


AlexNet

- Strategy 5 – Dropout



(a) Standard Neural Net

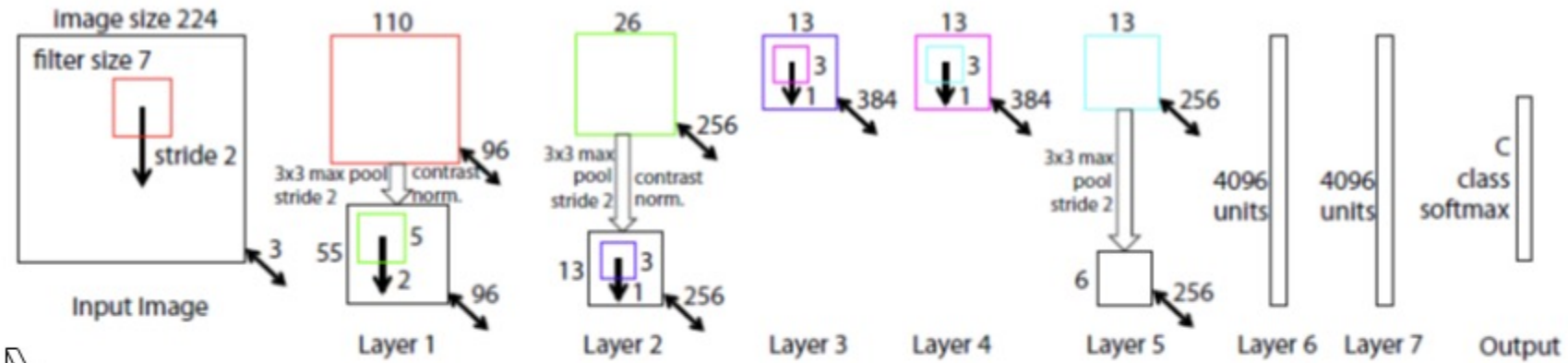


(b) After applying dropout.

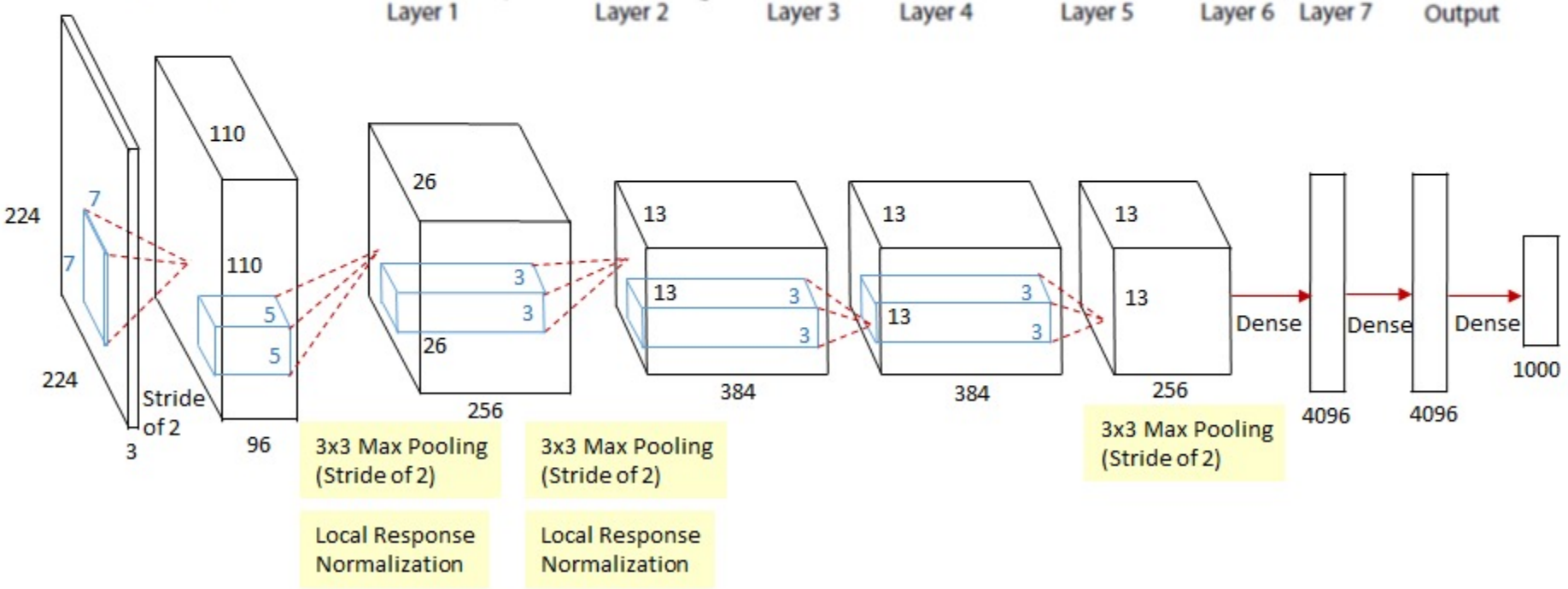
ZFNet

- Architecture

ZFNet



AlexNet



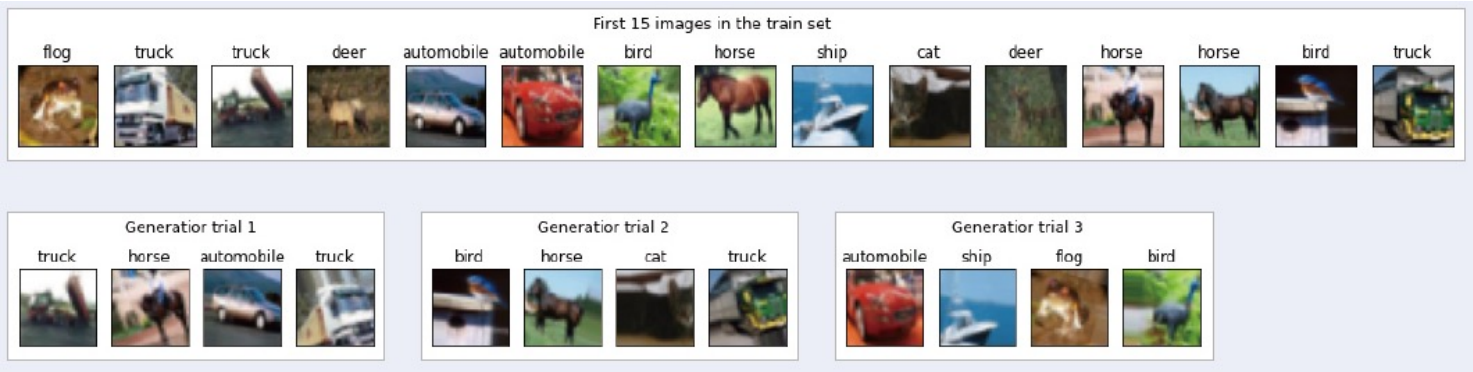
프로그래밍 실습: Data augmentation

프로그램 8-5 증강된 영상 확인하기

```
01  import tensorflow.keras.datasets as ds
02  from tensorflow.keras.preprocessing.image import ImageDataGenerator
03  import matplotlib.pyplot as plt
04
05  (x_train,y_train),(x_test,y_test)=ds.cifar10.load_data()
06  x_train=x_train.astype('float32'); x_train/=255
07  x_train=x_train[0:15,]; y_train=y_train[0:15,]   # 앞 15개에 대해서만 증대 적용
08  class_names=['airplane','automobile','bird','cat','deer','dog','flog','horse',
    'ship','truck']
09
10  plt.figure(figsize=(20,2))
11  plt.suptitle("First 15 images in the train set")
12  for i in range(15):
13      plt.subplot(1,15,i+1)
14      plt.imshow(x_train[i])
15      plt.xticks([]); plt.yticks([])
16      plt.title(class_names[int(y_train[i])])
17  plt.show()
18
```

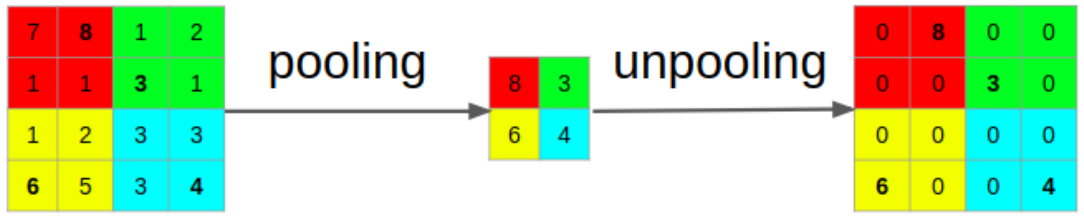
프로그래밍 실습: Data augmentation

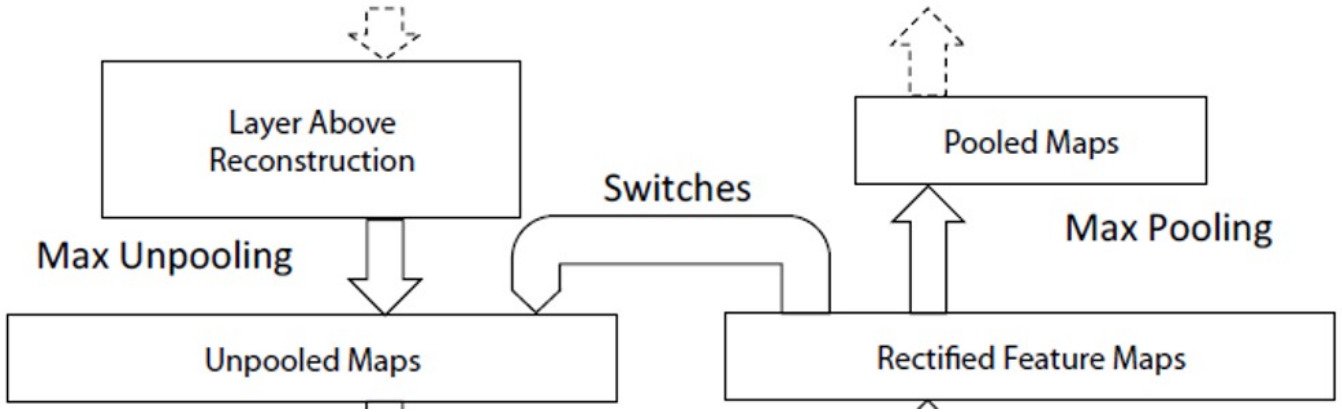
```
19  batch_siz=4                                    # 한 번에 생성하는 양(미니 배치)
20  generator=ImageDataGenerator(rotation_range=20.0,width_shift_range=0.2,
    height_shift_range=0.2,horizontal_flip=True)
21  gen=generator.flow(x_train,y_train,batch_size=batch_siz)
22
23  for a in range(3):
24      img,label=gen.next()                       # 미니 배치만큼 생성
25      plt.figure(figsize=(8,2.4))
26      plt.suptitle("Generatior trial «+str(a+1))
27      for i in range(batch_siz):
28          plt.subplot(1,batch_siz,i+1)
29          plt.imshow(img[i])
30          plt.xticks([]); plt.yticks([])
31          plt.title(class_names[int(label[i])])
32      plt.show()
```

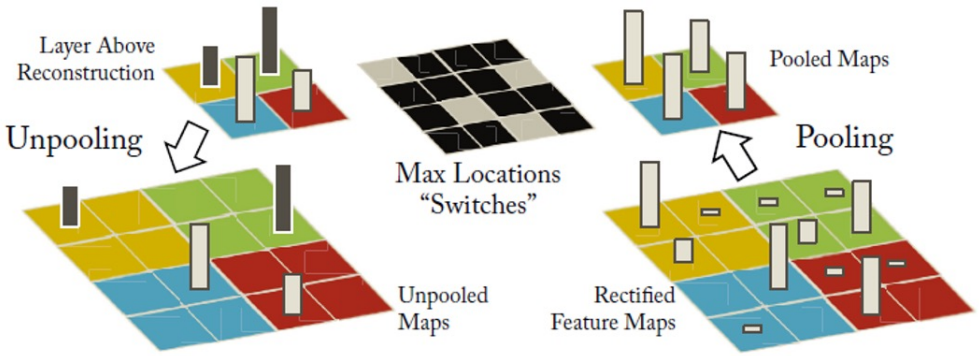


ZFNet

- Strategy 1 – Max unpooling
  - Max pooling 시 위치 정보를 기억
    - 역과정(unpooling) 수행 시 max값이 위치한 영역에 max값 할당, 나머지 영역은 0으로 할당



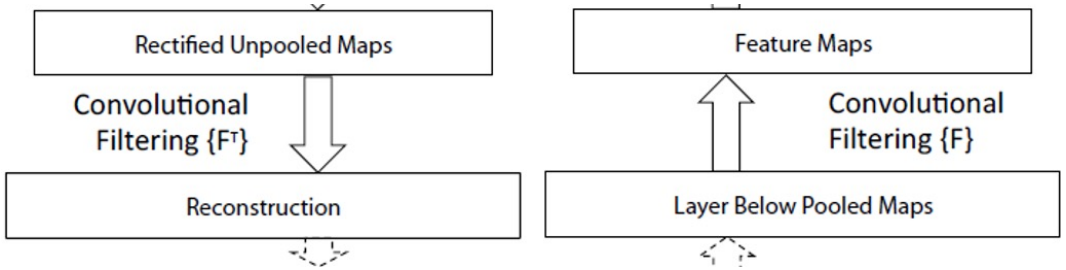


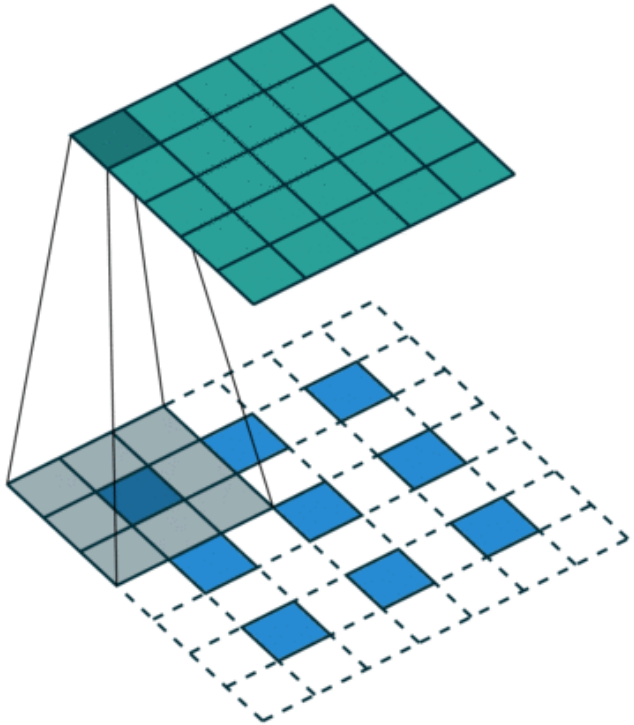


ZFNet

- Strategy 2 – ReLU recovering
  - Rectification
    - ReLU의 역과정 수행 ➔ 음의 값은 복원 불가... ➔ 거의 영향 없음을 밝힘

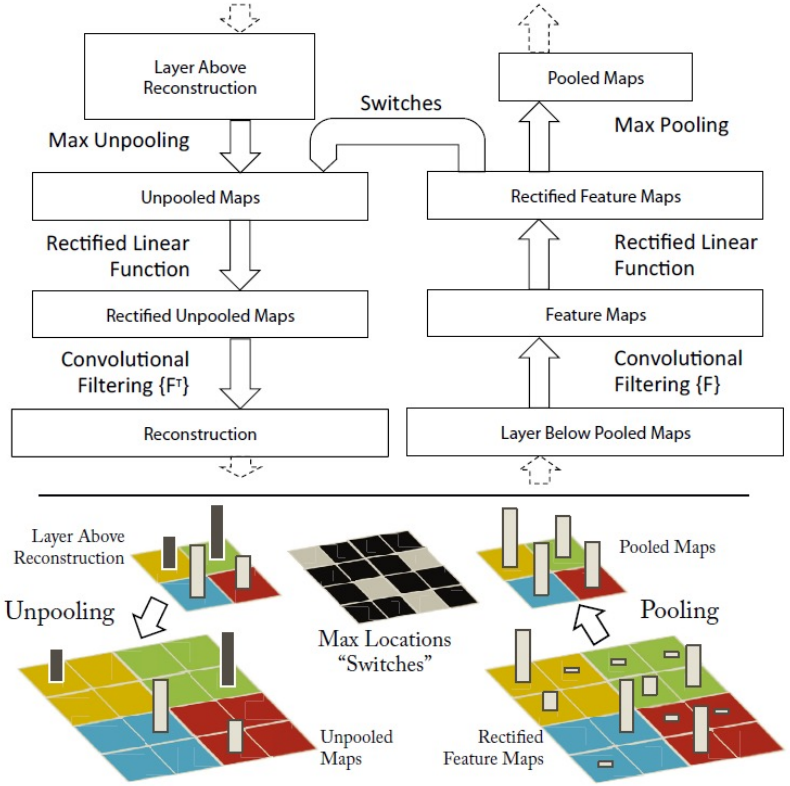
- Strategy 3 - Deconvolution

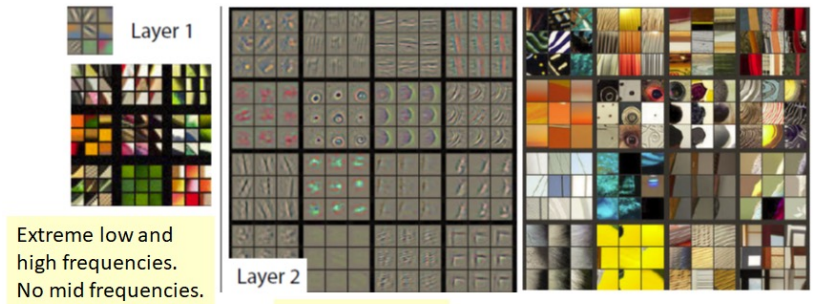


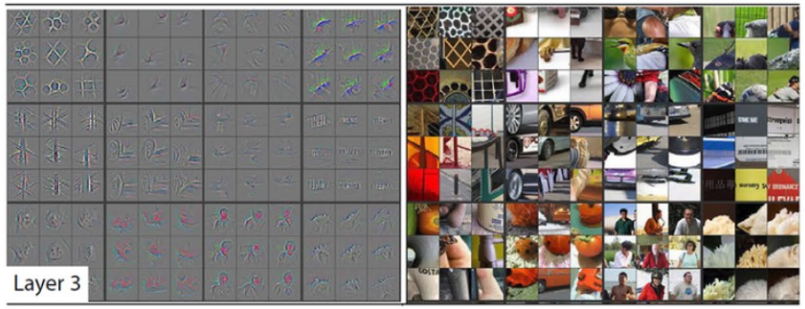


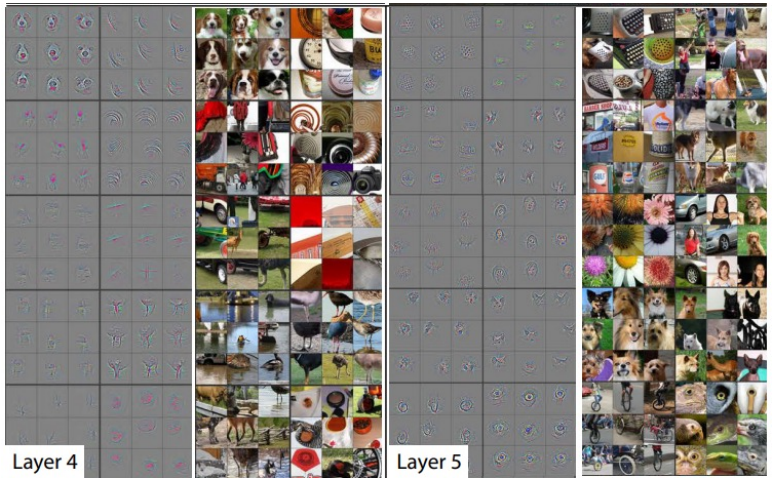
ZFNet

- Feature visualization
  - feature map의 시각화를 통해 deep learning의 black-box 구조를 살펴봄



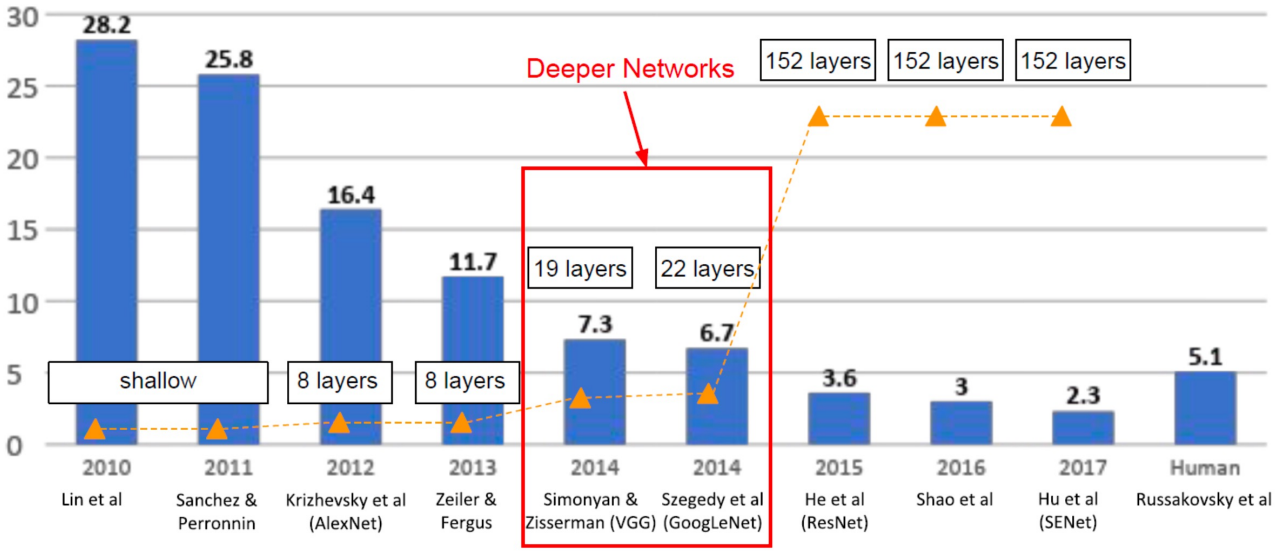


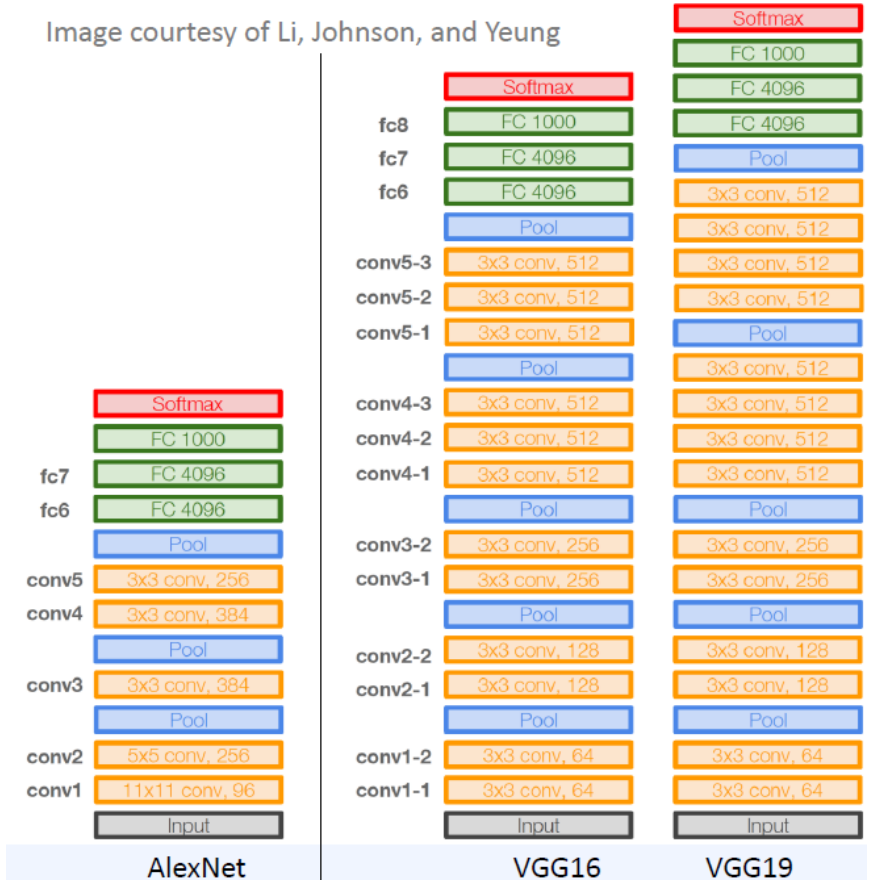




VGGNet

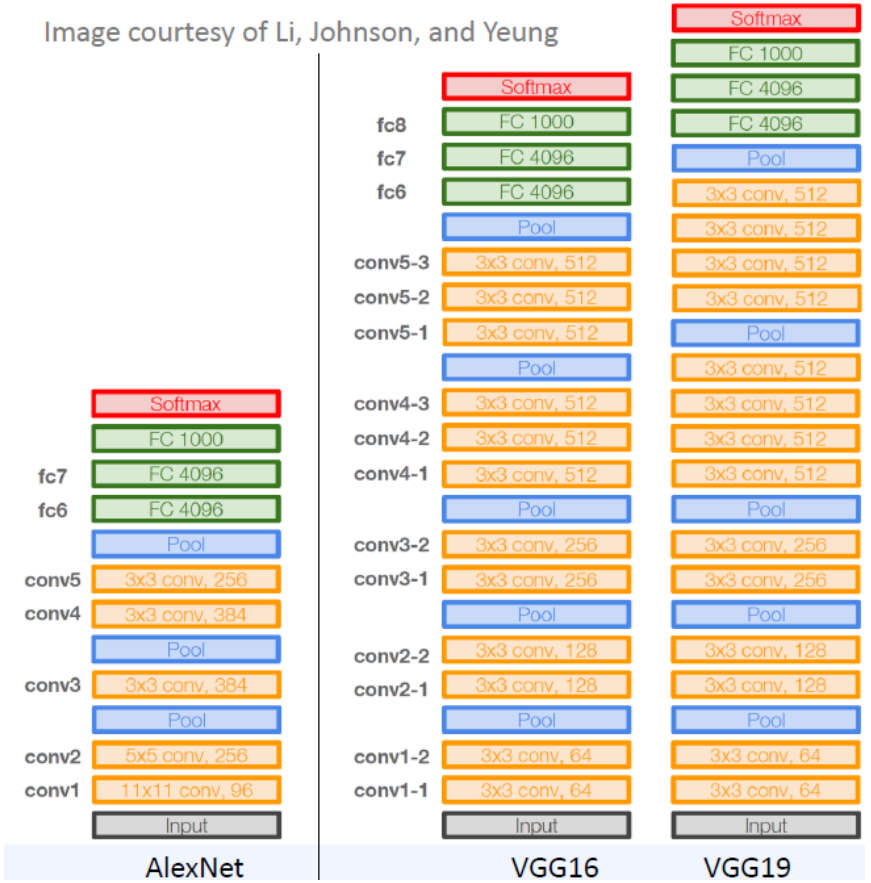
- Deeper networks learn more powerful features
  - AlexNet, ZFNet ➔ Layer 8개 이하
  - VGGNet ➔ 더 깊은 layers
    - 신경망이 깊어질 수록 나타나는 문제점
      - gradient vanishing





VGGNet

- Revolution of depth
  - Deeper architecture: 16 and 19 layers
  - Simpler architecture
    - No LRN
    - Only 3x3 conv filter, 2x2 max-pooling, a few FC layers
  - Better performance
    - Over AlexNet on ImageNet Challenge
  - Better generalization
    - Final features generalizing well without fine-tuning
  - Input: 224x224 RGB images



VGGNet

- Usage of VGGNet in library
  - Keras

```
from keras.applications import vgg16 model = vgg16.VGG16(weights ='imagenet', include_top = True)
model.summary()
```

  - Torch hub

```
import torch model = torch.hub.load('pytorch/vision:v0.10.0', 'vgg16', pretrained=True)
# pre-trained weight가 필요 없을때는 False
```

  - Torchvision

```
import torchvision model = torchvision.models.vgg16(pretrained=True)
```

ResNet

- Resolution of Depth
  - AlexNet (8 layers), VGGNet (19 layers) ➔ ResNet (maximum 152 layers)
  - Layer가 20개 이상일 때 degradation (gradient vanishing) 문제 발생 (성능이 떨어지기 시작)
- Two strategies for deeper network
  - residual learning
  - batch normalization

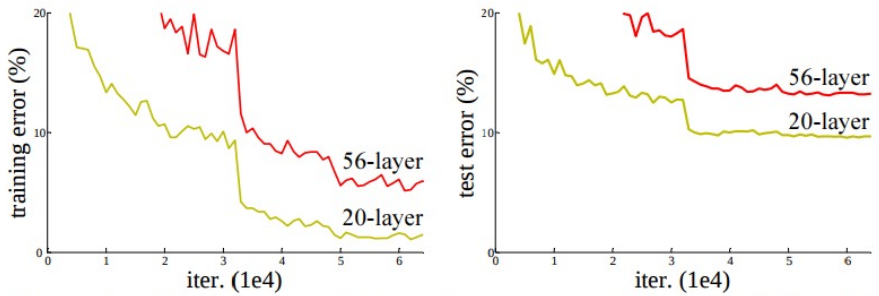
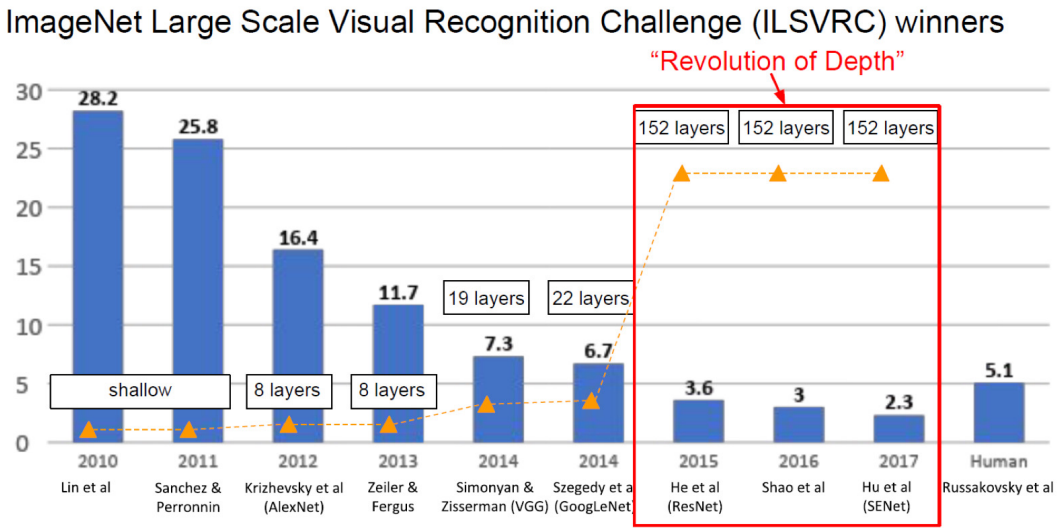
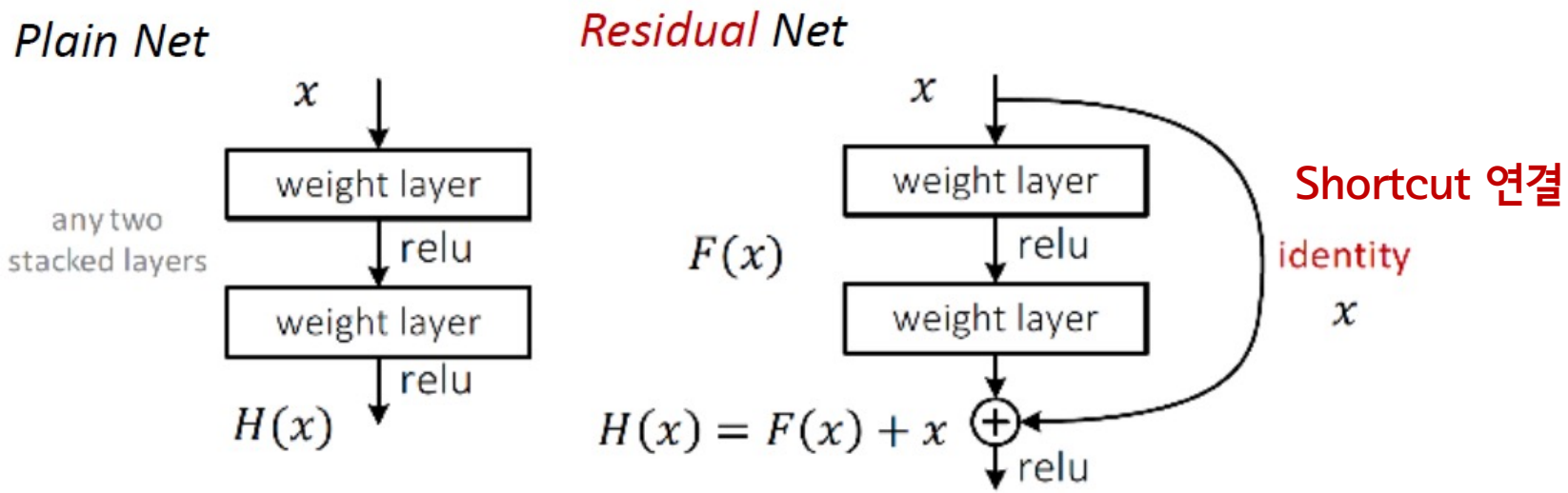


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.



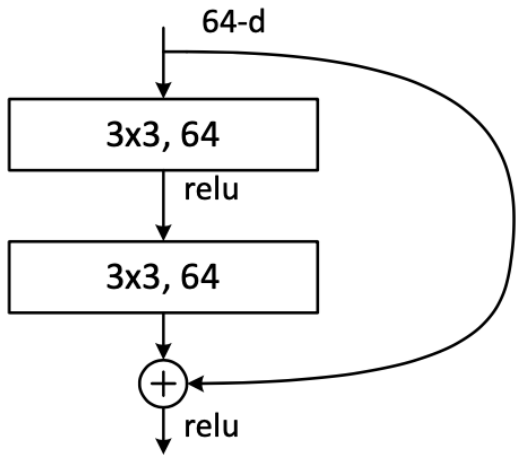
ResNet

- Techniques 1 – Residual learning (잔차학습)
  - 기본 아이디어
    - residual block (잔차 블록)을 통해 입력을 출력에 직접 연결하는 것
    - 네트워크가 학습하는 대상 (관점)
      - plane net: 입력데이터 ➔ 출력데이터
      - residual net: 입력데이터 ➔ 입력과 출력의 잔차 (residual)

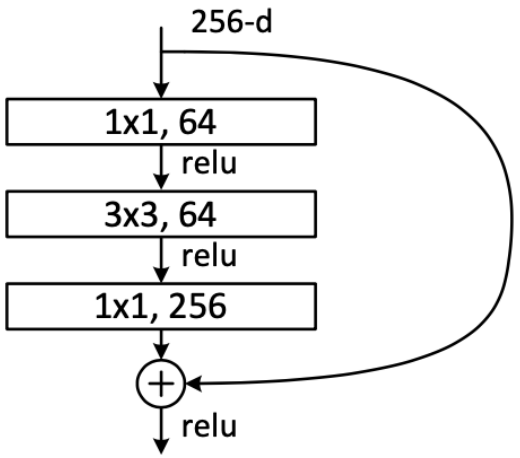


ResNet

- Techniques 1 – Residual learning (잔차학습)
  - 일반적인 잔차 블록: conv(3x3) + conv(3x3) + identity
  - Bottleneck 구조 (50 layers 이상일 때)의 잔차 블록: conv(1x1) + conv(3x3) + conv(1x1) + identity
    - 파라미터 수 감소 ➔ 연산량 감소
    - Layer 수 증가 (= activation function 증가) ➔ 더 많은 non-linearity 반영 (풍부한 특징 학습)



일반적인 residual block 구조



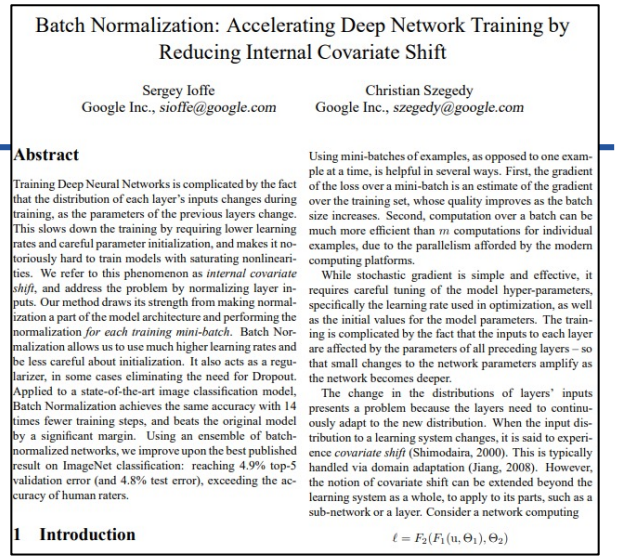
Bottleneck 구조

ResNet

- Techniques 2 – Batch normalization
  - Gradient vanishing / exploding issue
    - Gradient 기반 파라미터 학습

      ➔ parameter값의 작은 변화가 출력에 얼마나 미칠 것인가?
    - Gradient == 미분값 (변화량)

      ➔ 매우 커지거나 (exploding) 매우 작아지면 (vanishing) 모델의 성능 저하
  - Normalization ➔ Local optimum에 빠질 가능성을 낮춤



Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

Sergey Ioffe
Google Inc., sioffe@google.com

Christian Szegedy
Google Inc., szegedy@google.com

Abstract

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities. We refer to this phenomenon as *internal covariate shift*, and address the problem by normalizing layer inputs. Our method draws its strength from making normalization a part of the model architecture and performing the normalization *for each training mini-batch*. Batch Normalization allows us to use much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Applied to a state-of-the-art image classification model, Batch Normalization achieves the same accuracy with 14 times fewer training steps, and beats the original model by a significant margin. Using an ensemble of batch-normalized networks, we improve upon the best published result on ImageNet classification: reaching 4.9% top-5 validation error (and 4.8% test error), exceeding the accuracy of human raters.

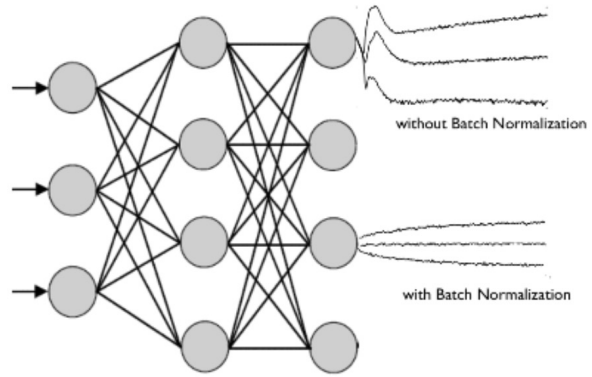
Using mini-batches of examples, as opposed to one example at a time, is helpful in several ways. First, the gradient of the loss over a mini-batch is an estimate of the gradient over the training set, whose quality improves as the batch size increases. Second, computation over a batch can be much more efficient than m computations for individual examples, due to the parallelism afforded by the modern computing platforms.

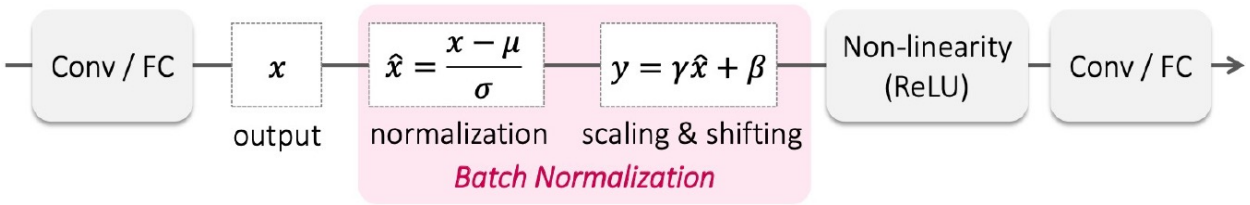
While stochastic gradient is simple and effective, it requires careful tuning of the model hyper-parameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters. The training is complicated by the fact that the inputs to each layer are affected by the parameters of all preceding layers – so that small changes to the network parameters amplify as the network becomes deeper.

The change in the distributions of layers' inputs presents a problem because the layers need to continuously adapt to the new distribution. When the input distribution to a learning system changes, it is said to experience *covariate shift* (Shimodaira, 2000). This is typically handled via domain adaptation (Jiang, 2008). However, the notion of covariate shift can be extended beyond the learning system as a whole, to apply to its parts, such as a sub-network or a layer. Consider a network computing

$$\ell = F_2(F_1(u, \Theta_1), \Theta_2)$$

1 Introduction





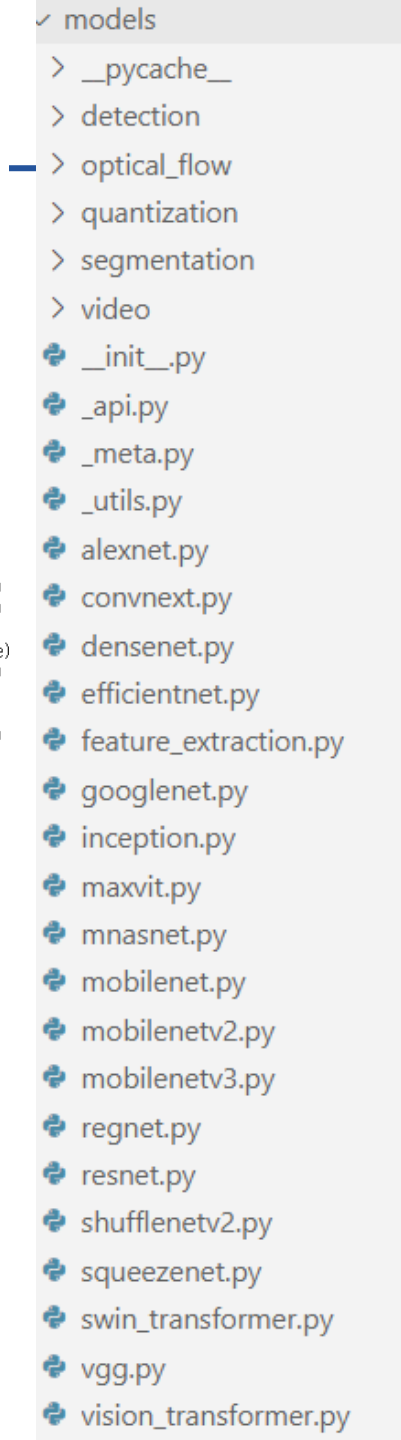
ResNet

- Usage of ResNet in library
  - Torchvision

```
from torchvision import models
import torch

resnet18_pretrained = models.resnet18(pretrained=True)
print(resnet18_pretrained)
```

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
```

- (참고) Torchvision은 컴퓨터비전에 필요한 다양한 모델 제공
  - [YOUR ENV.]/lib/python/site-packages/torchvision/models
  - ex) /home/user/anaconda3/envs/bj/lib/python3.8/site-packages/torchvision/models



프로그래밍 실습: Pre-trained ResNet으로 자연 영상 인식

- ImageNet dataset으로 pre-traine된 ResNet 활용

프로그램 8-6 ResNet50으로 자연 영상 인식하기

```
01  import cv2 as cv
02  import numpy as np
03  from tensorflow.keras.applications.resnet50 import ResNet50,preprocess_
    input,decode_predictions
04
05  model=ResNet50(weights='imagenet')
06
07  img=cv.imread('rabbit.jpg')
08  x=np.reshape(cv.resize(img,(224,224)),(1,224,224,3))
09  x=preprocess_input(x)
10
11  preds=model.predict(x)
12  top5=decode_predictions(preds,top=5)[0]
13  print('예측 결과:',top5)
14
15  for i in range(5):
16      cv.putText(img,top5[i][1]+':'+str(top5[i][2]),(10,20+i*20),cv.FONT_HERSHEY_
        SIMPLEX,0.5,(255,255,255),1)
17
18  cv.imshow('Recognition result',img)
19
20  cv.waitKey()
21  cv.destroyAllWindows()
```

```
예측 결과: [('n02325366', 'wood_rabbit', 0.74258107), ('n02326432', 'hare',
0.24038698), ('n02328150', 'Angora', 0.008831766), ('n01877812', 'wallaby',
0.0026911795), ('n02356798', 'fox_squirrel', 0.0012295933)]
```

프로그래밍 실습: Pre-trained DenseNet으로 견종 인식

- Stanford dogs 데이터셋 다운로드
  - 1) http://vision.stanford.edu/aditya86/ImageNetDogs/
    - image.tar, annotation.tar, list.tar 다운로드
  - 2) 소스코드가 있는 폴더에 datasets/Stanford_dogs 폴더 생성 후 다운로드 받은 파일들 저장
  - 3) 세 파일의 압축 해제 후 annotations, images, list 폴더 확인
    - images/images 폴더 내 n02085620-Chihuahua 등 120여개의 폴더가 있는지 확인

그림 8-25 Stanford dogs 데이터셋(왼쪽부터 Chihuahua, Japanese_spaniel, Maltese_dog, Pekinese, Shih-Tzu, Blenheim_spaniel)

프로그래밍 실습: Pre-trained DenseNet으로 견종 인식

프로그램 8-7 DenseNet121로 견종 인식하기

```
01  from tensorflow.keras.models import Sequential
02  from tensorflow.keras.layers import Flatten,Dense,Dropout,Rescaling
03  from tensorflow.keras.optimizers import Adam
04  from tensorflow.keras.applications.densenet import DenseNet121
05  from tensorflow.keras.utils import image_dataset_from_directory
06  import pathlib
07
08  data_path=pathlib.Path('datasets/stanford_dogs/images/images')
09
10  train_ds=image_dataset_from_directory(data_path,validation_split=0.2,subset
    ='training',seed=123,image_size=(224,224),batch_size=16) ①
11  test_ds=image_dataset_from_directory(data_path,validation_split=0.2,subset
    ='validation',seed=123,image_size=(224,224),batch_size=16)
12
13  base_model=DenseNet121(weights='imagenet',include_top=False,input_shape
    =(224,224,3))
14  cnn=Sequential()
15  cnn.add(Rescaling(1.0/255.0))
16  cnn.add(base_model)
17  cnn.add(Flatten())
18  cnn.add(Dense(1024,activation='relu'))
19  cnn.add(Dropout(0.75))
20  cnn.add(Dense(units=120,activation='softmax'))
```

폴더에서 데이터 읽기

분류 층을 배제

DenseNet121을 백본으로 사용

프로그래밍 실습: Pre-trained DenseNet으로 견종 인식

미세 조정 방식의 전이 학습:
학습률을 아주 작게 설정하여 특징 추출을 담당하는 층의 가중치를 유지

대신 세대 수를 늘려 오래 학습하는 전략

```
21
22  cnn.compile(loss='sparse_categorical_crossentropy',optimizer=Adam(learning_
    rate=0.000001),metrics=['accuracy'])
23  hist=cnn.fit(train_ds,epochs=200,validation_data=test_ds,verbose=2) ②
24
25  print('정확률=',cnn.evaluate(test_ds,verbose=0)[1]*100) ③
26
27  cnn.save('cnn_for_stanford_dogs.h5')        # 미세 조정된 모델을 파일에 저장
28
```

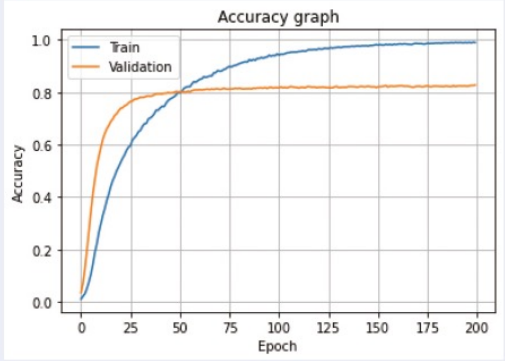
비전 에이전트에 쓰기 위해 학습된 모델을 저장함

프로그래밍 실습: Pre-trained DenseNet으로 견종 인식

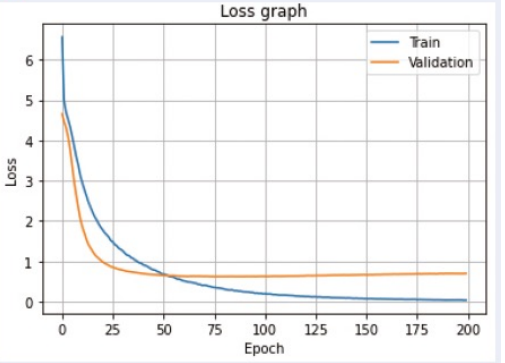
```
29  import pickle
30  f=open('dog_species_names.txt','wb')
31  pickle.dump(train_ds.class_names,f)
32  f.close()
33
34  import matplotlib.pyplot as plt
35
36  plt.plot(hist.history['accuracy']) ④
37  plt.plot(hist.history['val_accuracy'])
38  plt.title('Accuracy graph')
39  plt.ylabel('Accuracy')
40  plt.xlabel('Epoch')
41  plt.legend(['Train','Validation'])
42  plt.grid()
43  plt.show()
44
45  plt.plot(hist.history['loss']) ⑤
46  plt.plot(hist.history['val_loss'])
47  plt.title('Loss graph')
48  plt.ylabel('Loss')
49  plt.xlabel('Epoch')
50  plt.legend(['Train','Validation'])
51  plt.grid()
52  plt.show()
```

```
Found 20580 files belonging to 120 classes. ①
Using 16464 files for training.
Found 20580 files belonging to 120 classes.
Using 4116 files for validation.

Epoch 1/200 ②
1029/1029 - 181s - loss: 6.5628 - accuracy: 0.0111 - val_loss: 4.6525 - val_accuracy:
0.0355 - 181s/epoch - 176ms/step
Epoch 2/200
1029/1029 - 172s - loss: 4.9830 - accuracy: 0.0214 - val_loss: 4.4681 - val_accuracy:
0.0717 - 172s/epoch - 168ms/step
Epoch 3/200
1029/1029 - 170s - loss: 4.6948 - accuracy: 0.0315 - val_loss: 4.3172 - val_accuracy:
0.1273 - 170s/epoch - 166ms/step
...
Epoch 199/200
1029/1029 - 167s - loss: 0.0360 - accuracy: 0.9907 - val_loss: 0.6933 - val_accuracy:
0.8258 - 167s/epoch - 163ms/step
Epoch 200/200
1029/1029 - 168s - loss: 0.0378 - accuracy: 0.9897 - val_loss: 0.6995 - val_accuracy:
0.8273 - 168s/epoch - 163ms/step

정확률= 82.72594809532166 ③
```

④



⑤



EfficientNet

- By Google AI research team
- 효율성과 정확도의 균형을 맞추는 것을 목표로 함 ➔ 기존 모델들보다 훨씬 적은 계산비용으로 높은 정확도 달성
- 핵심 특징: 균형 잡힌 compound scaling

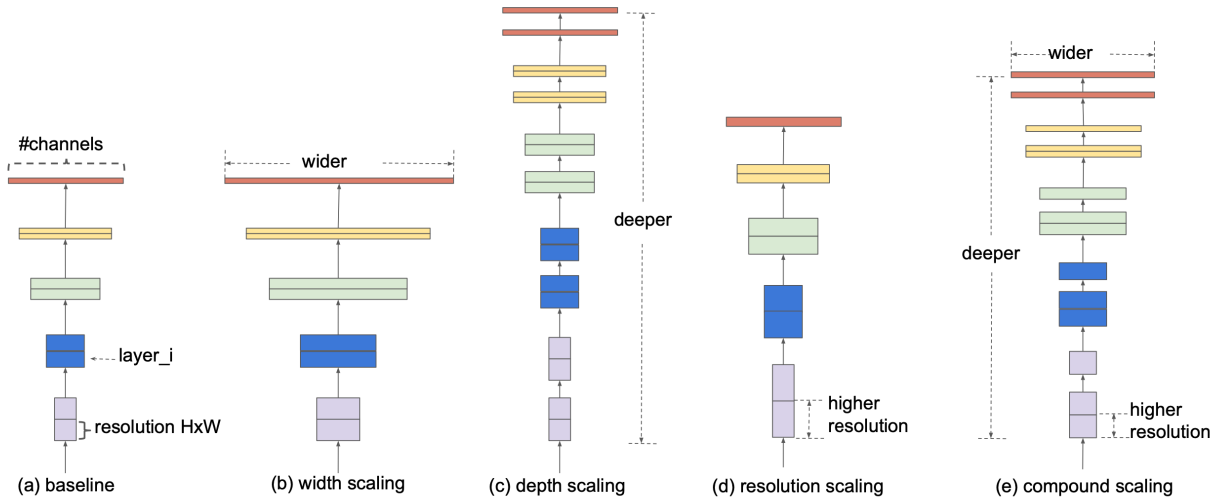


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks

Mingxing Tan [1] **Quoc V. Le** [1]

Abstract

Convolutional Neural Networks (ConvNets) are commonly developed at a fixed resource budget, and then scaled up for better accuracy if more resources are available. In this paper, we systematically study model scaling and identify that carefully balancing network depth, width, and resolution can lead to better performance. Based on this observation, we propose a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective *compound coefficient*. We demonstrate the effectiveness of this method on scaling up MobileNets and ResNet.

To go even further, we use neural architecture search to design a new baseline network and scale it up to obtain a family of models, called *EfficientNets*, which achieve much better accuracy and efficiency than previous ConvNets. In particular, our EfficientNet-B7 achieves state-of-the-art 84.3% top-1 accuracy

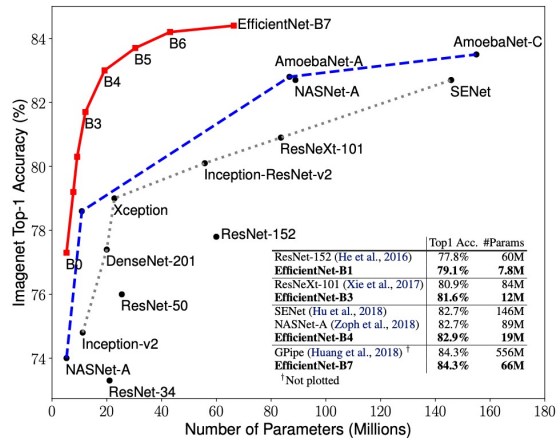


Figure 1. **Model Size vs. ImageNet Accuracy.** All numbers are for single-crop, single-model. Our EfficientNets significantly outperform other ConvNets. In particular, EfficientNet-B7 achieves new state-of-the-art 84.3% top-1 accuracy but being 8.4x smaller and 6.1x faster than GPipe. EfficientNet-B1 is 7.6x smaller and 5.7x faster than ResNet-152. Details are in Table 2 and 4.

EfficientNet

- Compound scaling
  - 모델의 깊이(depth), 너비(width), 입력 이미지의 해상도(resolution)을 동시에 조정하는 방식
- EfficientNet: 모델의 성능을 효율적으로 향상시키기 위해 세가지 차원의 균형을 최적화
  - 원리
    - 네트워크의 깊이 증가 ➔ 모델이 더 복잡한 특징과 패턴을 학습할 수 있음
    - 네트워크의 너비 (== 각 층의 채널 수) 증가 ➔ 모델이 더 많은 특징을 동시에 학습할 수 있음
    - 입력 해상도 증가 ➔ 모델이 더 세밀한 패턴과 텍스처를 학습할 수 있음
- EfficientNet-B0 (baseline) ~ B1~B7
  - compound scaling 비율에 따라 모델 확장

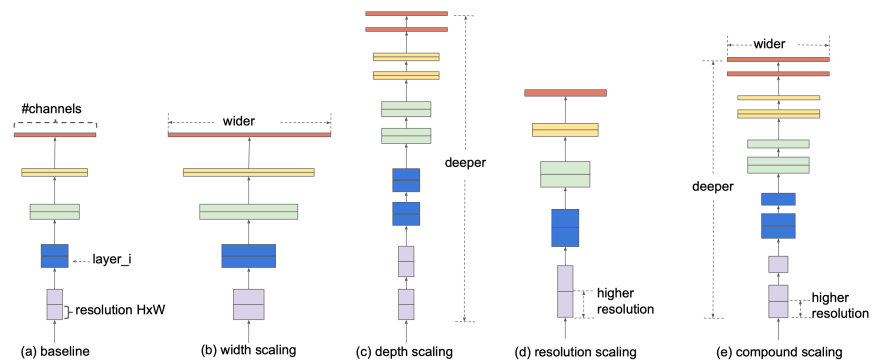
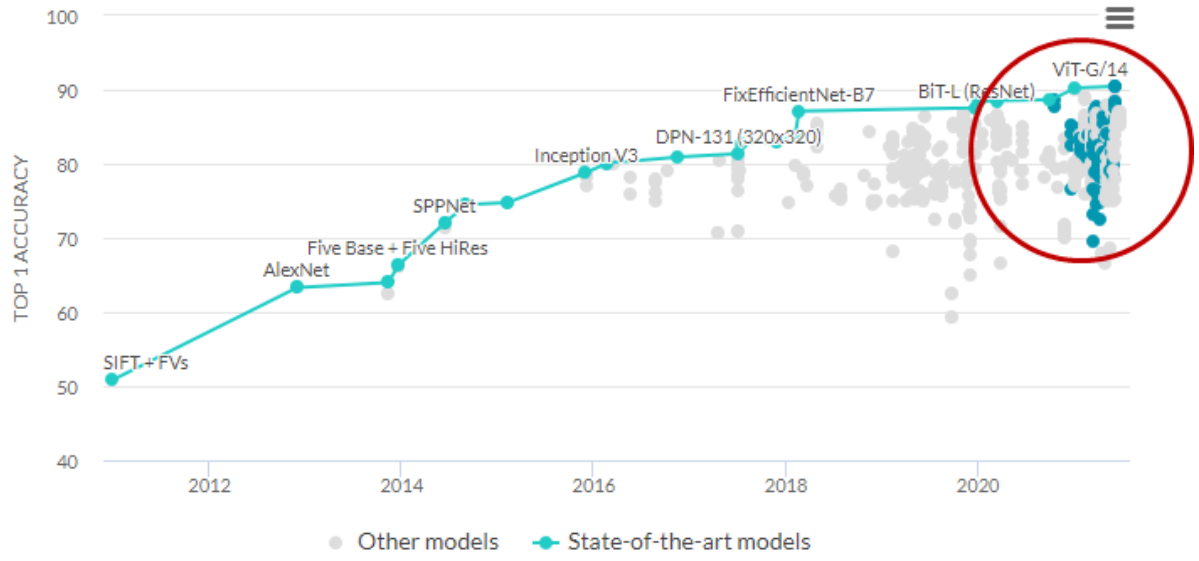


Figure 2. **Model Scaling.** (a) is a baseline network example; (b)-(d) are conventional scaling that only increases one dimension of network width, depth, or resolution. (e) is our proposed compound scaling method that uniformly scales all three dimensions with a fixed ratio.

Vision Transformer (ViT)

- An Image is Worth 16x16 Words: Transformer for Image Recognition at Scale
- Transformer 구조에서 착안
  - Transformer: 높은 계산 효율성과 확장성, self-attention



AN IMAGE IS WORTH 16X16 WORDS:
TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy*,†, Lucas Beyer*, Alexander Kolesnikov*, Dirk Weissenborn*, Xiaohua Zhai*, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby*,†

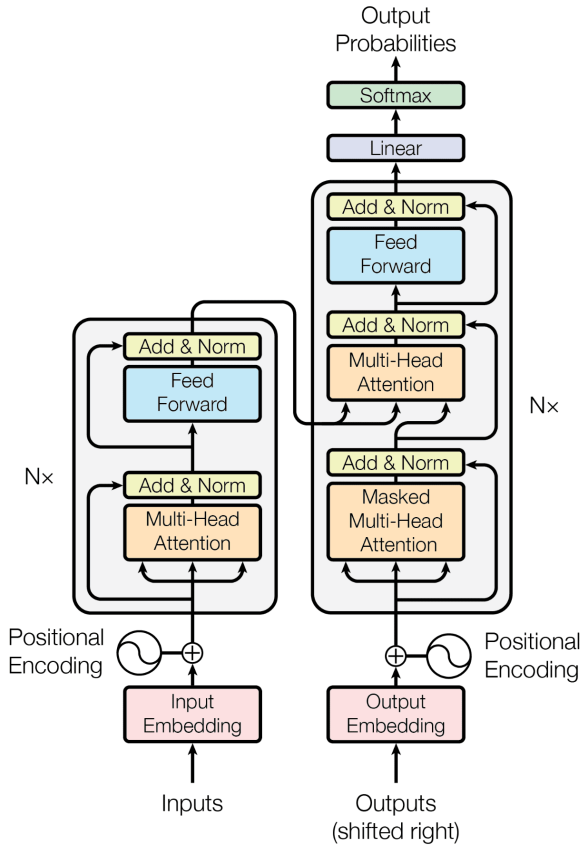
*equal technical contribution, †equal advising

Google Research, Brain Team

{adosovitskiy, neilhoulsby}@google.com

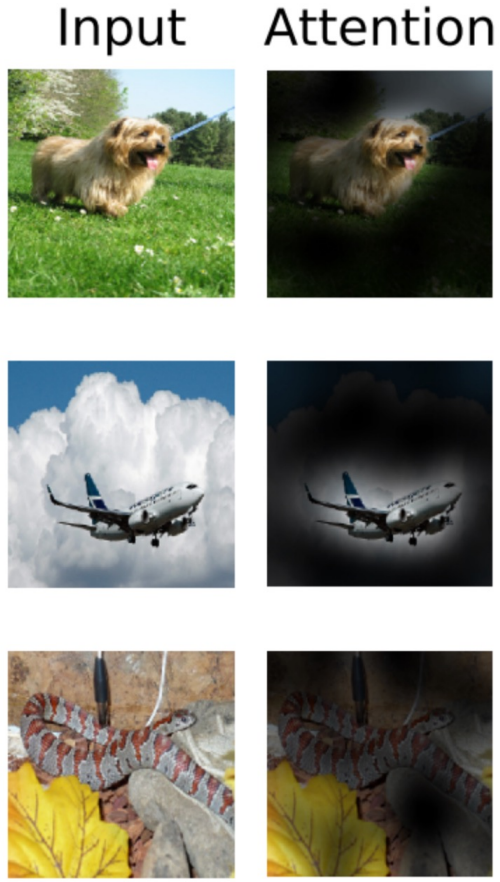
ABSTRACT

While the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, or used to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.[1]



Vision Transformer (ViT)

- ViT 동작 방식



5. Object Detection

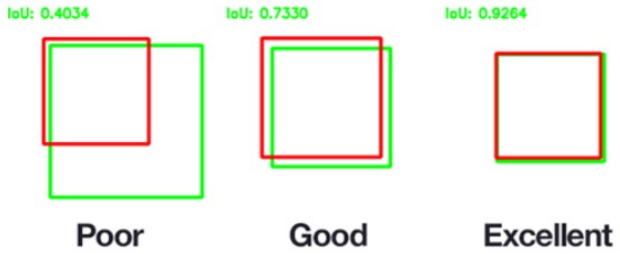
Evaluation metrics

- 객체 탐지 결과 평가 지표
  - 정확한 경계 상자를 찾았는가?
  - 영상 내 모든 객체를 찾았는가?
  - 각 객체마다 정확한 클래스를 찾았는가?
  - …

Evaluation metrics

- Intersection over Union (IoU)
  - 실제 경계 (ground truth)와 모델이 예측한 bounding box가 얼마나 겹치는지를 나타내는 지표
    - 클수록 잘 예측
  - 객체의 위치를 성공적으로 잘 포착했는가?
  - 일반적으로 threshold로 활용

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



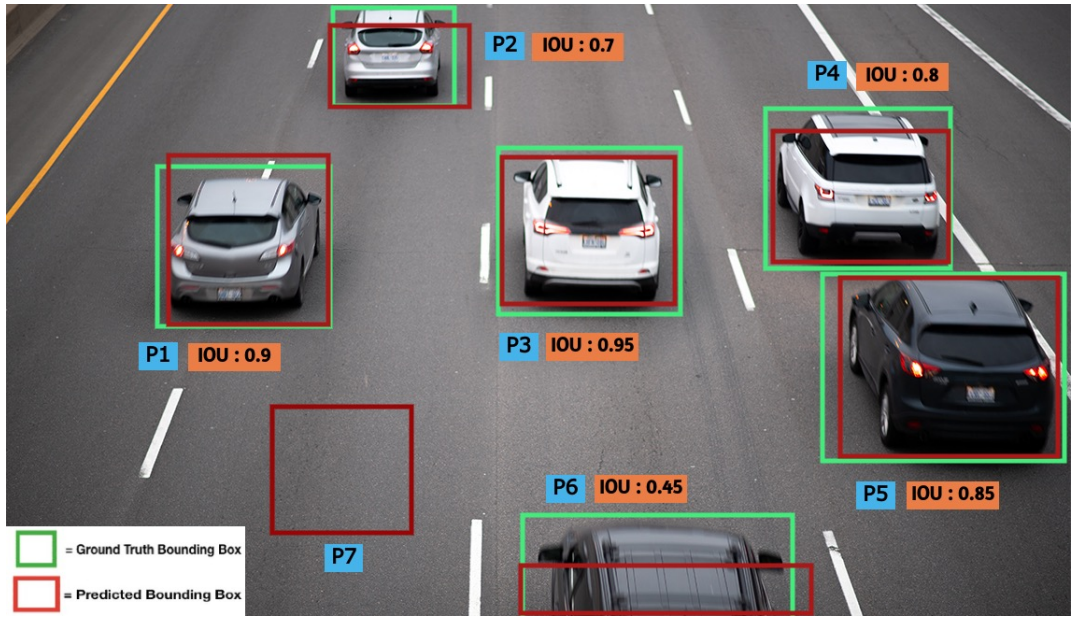
Evaluation metrics

- Precision and recall ➔ 해당 클래스를 옳게 분류하였는가?
  - Precision = TP / (TP + FP) ➔ 모델이 얼마나 정확하냐?
    - 참이라고 예측한 것들 중에서 정답을 맞춘 비율
    - ➔ ex) 강아지라고 예측한 것들 중에서 실제 강아지의 비율
    - TP: 예측된 bounding box가 실제 객체를 포함하고 IoU가 0.8 이상일 경우
    - FP: 예측된 bounding box가 실제 객체를 포함하지 않거나 IoU가 0.8 미만일 경우
  - Recall = TP / (TP + FN) ➔ 물체를 안 빼먹고 잘 검출하냐?
    - 실제 정답 중 얼마나 답을 맞추었는지에 대한 비율
    - ➔ ex) 실제 강아지인 것들 중에 강아지로 예측한 비율
    - FN: 실제 객체가 있는데 모델이 그것을 탐지하지 못한 경우

		Predicted	
		Negative (N) -	Positive (P) +
Actual	Negative -	True Negative (TN)	**False Positive (FP) Type I Error**
	Positive +	**False Negative (FN) Type II Error**	True Positive (TP)

Evaluation metrics

- Precision and recall
  - single class object

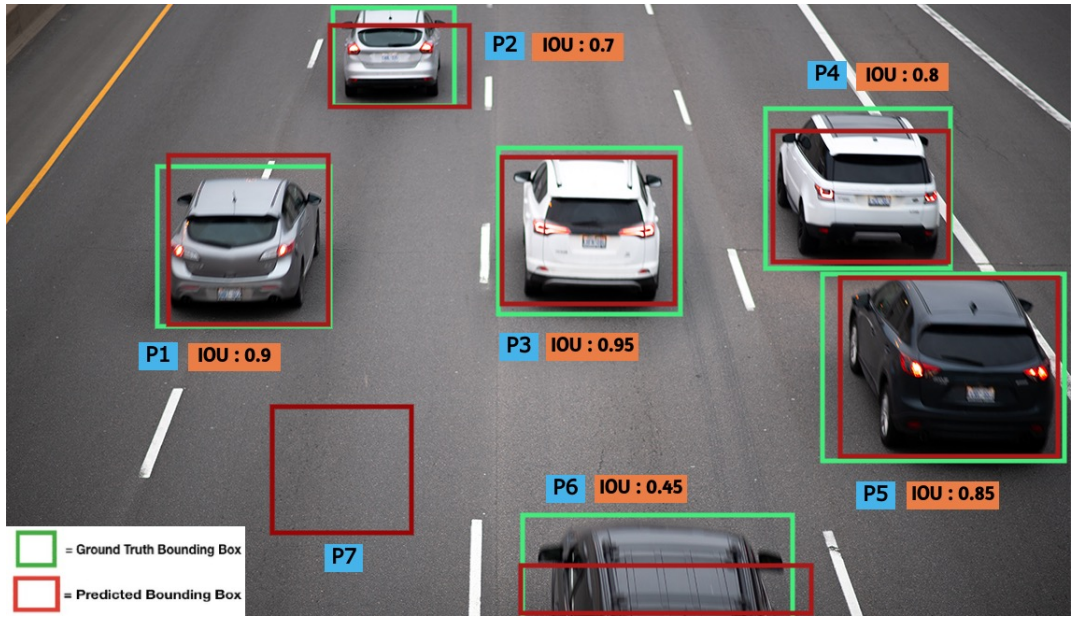


- 검출되어야 하는 물체: 6개 (TP+FN)
- 모델이 검출한 물체: 7개 (TP+FP)
- IoU threshold: 0.8 (사용자가 설정)
  - 모델이 잘 검출한 물체: 4개 (TP)
    - P1, P3, P4, P5

- ➔ Precision = 4/7 = 0.57
- ➔ Recall = 4/6 = 0.67

Evaluation metrics

- Precision and recall
  - single class object

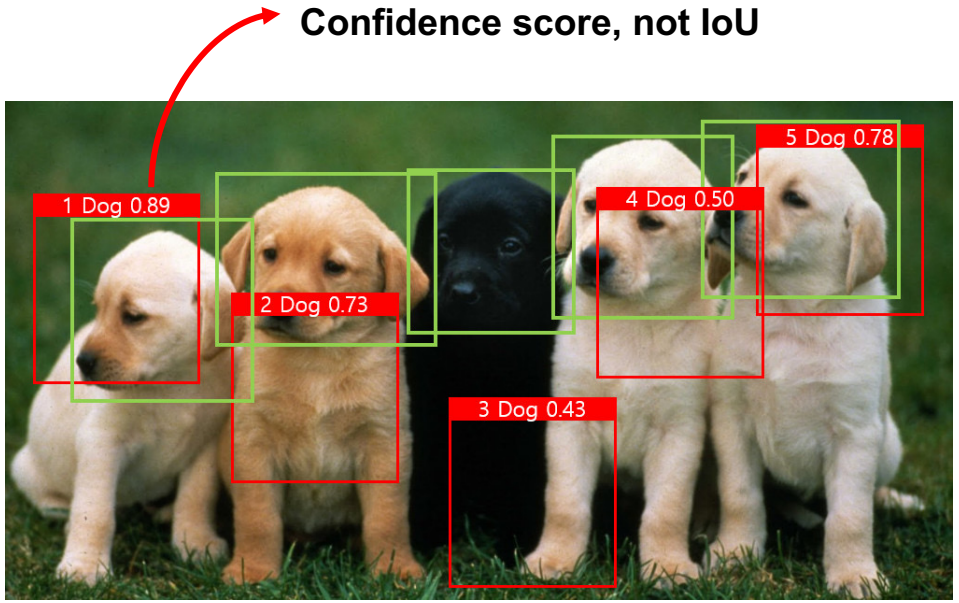


- 검출되어야 하는 물체: 6개 (TP+FN)
- 모델이 검출한 물체: 7개 (TP+FP)
- IoU threshold: 0.5 (사용자가 설정)
  - 모델이 잘 검출한 물체: 5개 (TP)
    - P1, P2, P3, P4, P5

- ➔ Precision = 5/7 = 0.71
- ➔ Recall = 5/6 = 0.83

Evaluation metrics

- Precision and recall
  - single class object

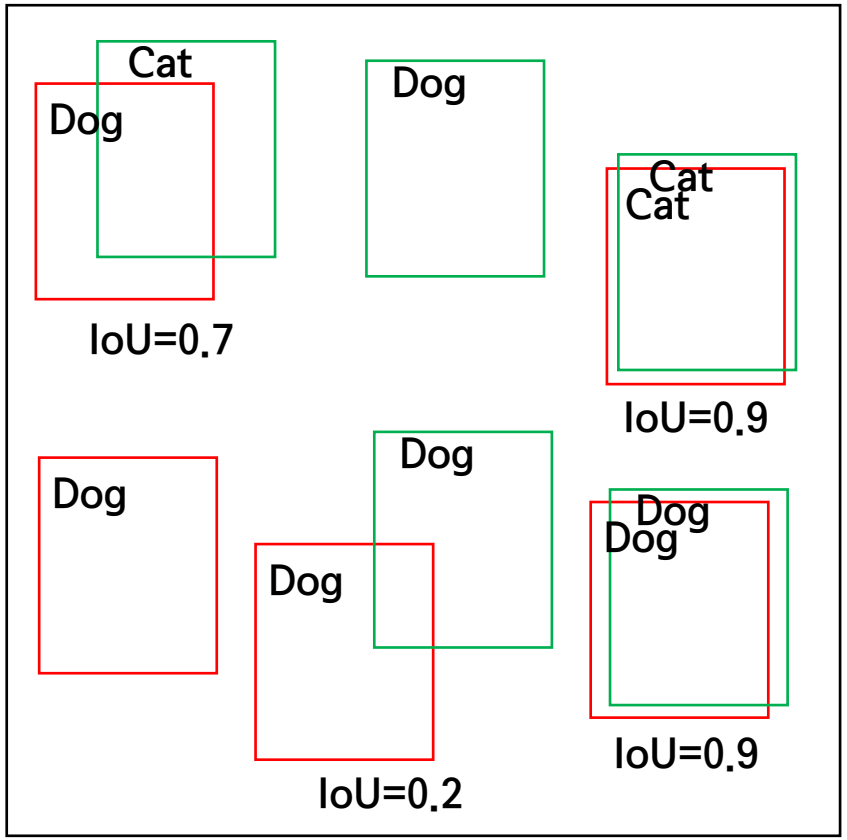


- 검출되어야 하는 물체: 5개 (TP+FN)
- 모델이 검출한 물체: 5개 (TP+FP)
- IoU threshold: 0.7 (사용자가 설정)
  - 모델이 잘 검출한 물체: 3개 (TP)
    - object 1, 2, 5

- ➔ Precision = 3/5 = 0.60
- ➔ Recall = 3/5 = 0.60

Evaluation metrics

- Precision and recall
  - multi class object



- 검출되어야 하는 cat: 2 (TP+FN)
- 검출되어야 하는 dog: 3 (TP+FN)
- 모델이 검출한 cat: 1 (TP+FP)
- 모델이 검출한 dog: 4 (TP+FP)
- 잘 검출된 cat: 1 (TP)
- 잘 검출된 dog: 1 (TP)

Precision(cat) = 1 / 1 = 1.00

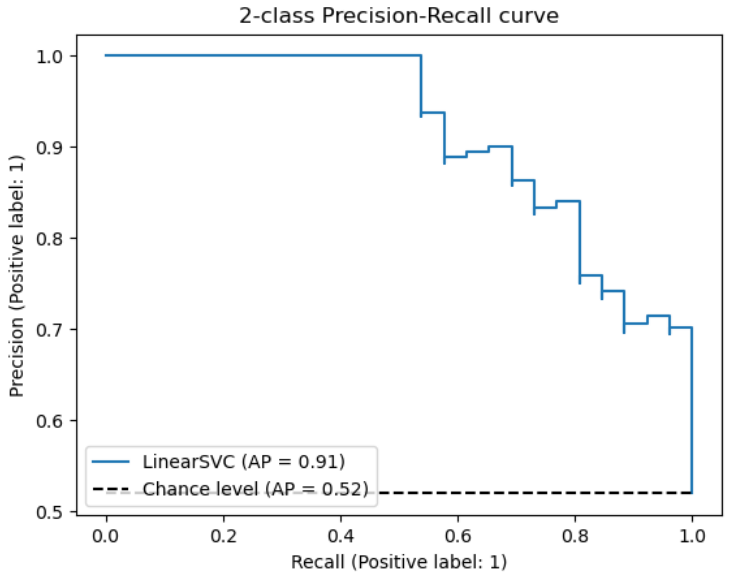
Recall(cat) = 1 / 2 = 0.50

Precision(dog) = 1 / 4 = 0.25

Recall(dog) = 1 / 3 = 0.33

Evaluation metrics

- AP (Average Precision)
  - 특정 클래스에 대한 Precision-Recall 곡선의 아래 면적
  - 면적의 크기가 1에 가까울 수록 모델의 성능이 뛰어남

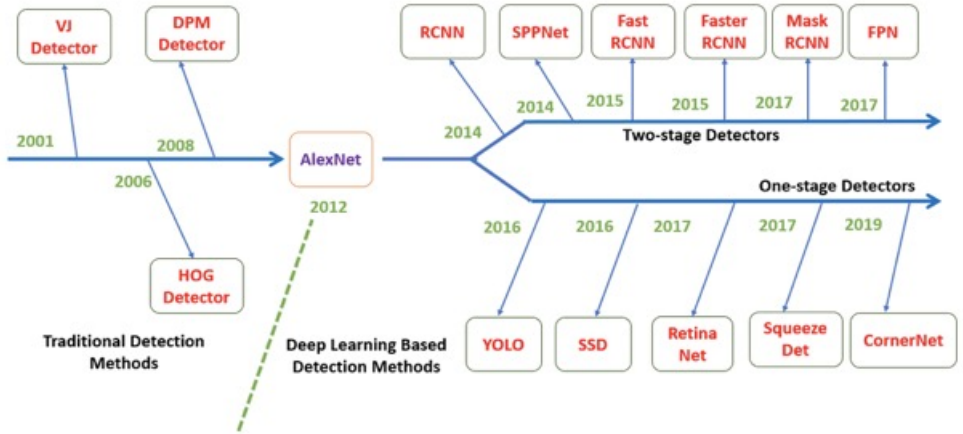


- mAP (mean Average Precision)
  - 여러 클래스에 대한 AP값의 평균

- AP@.50, AP@.75, ...
  - AP@.50 = IoU threshold가 0.5일 때의 AP
  - AP@.75 = IoU threshold가 0.75일 때의 AP

History of object detection

- Two-stage detector
  - Regional Convolutional Neural Network (R-CNN)
  - …
- One-stage detector
  - Single Shot Detector (SSD)
  - RetinaNet
  - You Only Look Once (YOLO)
  - DETR (Detection Transformer)
  - …



R-CNN (2013.11) → OverFeat (ICLR' 14) → MultiBox (CVPR' 14) → SPP-Net (ECCV' 14) → MR-CNN (ICCV' 15) → DeepBox (ICCV' 15) → AttentionNet (ICCV' 15)

Fast R-CNN (ICCV' 15) → DeepProposal (ICCV' 15) → Faster R-CNN (NIPS' 15) → OHEM (CVPR' 16) → YOLO v1 (CVPR' 16) → G-CNN (CVPR' 16) → AZNet (CVPR' 16)

Inside-OutsideNet(ION) (CVPR' 16) → HyperNet (CVPR' 16) → CRAFT (CVPR' 16) → MultiPathNet(MPN) (BMVC' 16) → SSD (ECCV' 16) → GBDNet (ECCV' 16)

CPF (ECCV' 16) → MS-CNN (ECCV' 16) → R-FCN (NIPS' 16) → PVANET (NIPSW' 16) → DeepID-Net (PAMI' 16) → NoC (TPAMI' 16) → DSSD (arXiv' 17) → TDM (CVPR' 17) → YOLO v2 (CVPR' 17)

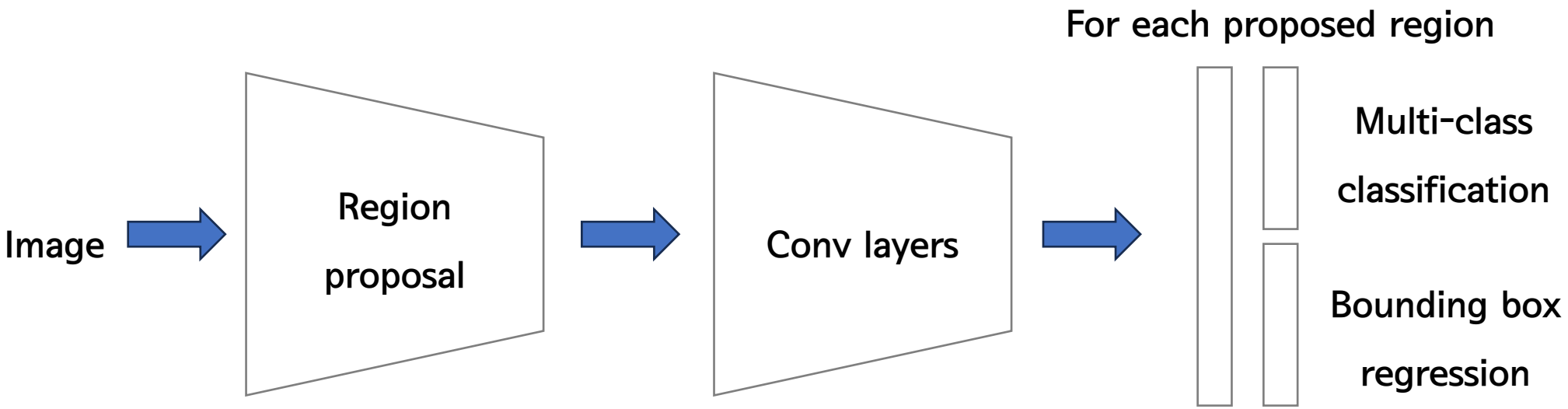
Feature Pyramid Net(FPN) (CVPR' 17) → RON (CVPR' 17) → DCN (ICCV' 17) → DeNet (ICCV' 17) → CoupleNet (ICCV' 17) → RetinaNet (ICCV' 17) → DSOD (ICCV' 17)

Mask R-CNN (ICCV' 17) → SMN (ICCV' 17) → YOLO v3 (arXiv' 18) → SIN (CVPR' 18) → STDN (CVPR' 18) → RefineDet (CVPR' 18) → MLKP (CVPR' 18) → Relation-Net (CVPR' 18)

Cascade R-CNN (CVPR' 18) → RFBNet (ECCV' 18) → CornerNet (ECCV' 18) → PFPNet (ECCV' 18) → Pelee (NIPS' 18) → HKRM (NIPS' 18) → R-DAD (AAAI' 19) → M2Det (AAAI' 19) …

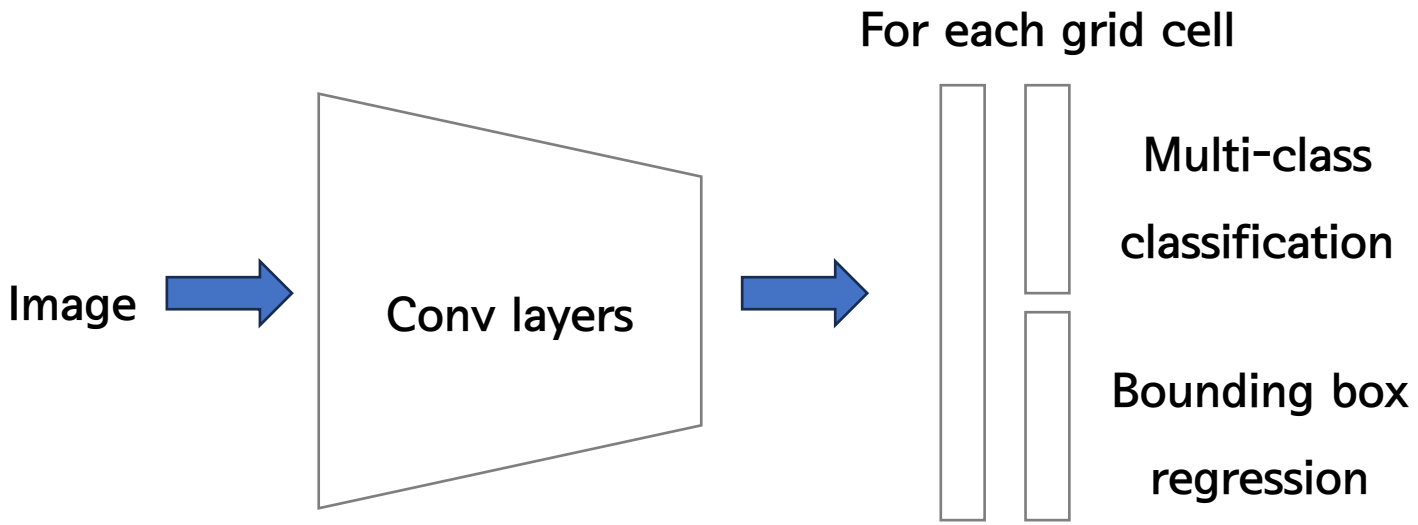
History of object detection

- Two-stage detector
  - region proposal stage
    - 후보 영역 (region proposal) 식별 ➔ 이미지 전체에서 객체가 존재할 가능성이 있는 영역을 찾음
  - classification and bounding box regression stage
    - 각 후보 영역에 대해 CNN을 사용하여 객체의 클래스를 분류하고, 객체의 위치를 나타내는 좌표를 예측



History of object detection

- One-stage detector
  - direct detection
    - 후보 영역 제안 단계 없이 바로 객체의 위치와 클래스를 동시에 예측
    - 특정한 경계 박스 안에서 객체가 있을 확률과 객체의 클래스를 예측, 동시에 바운딩 박스 좌표 조정



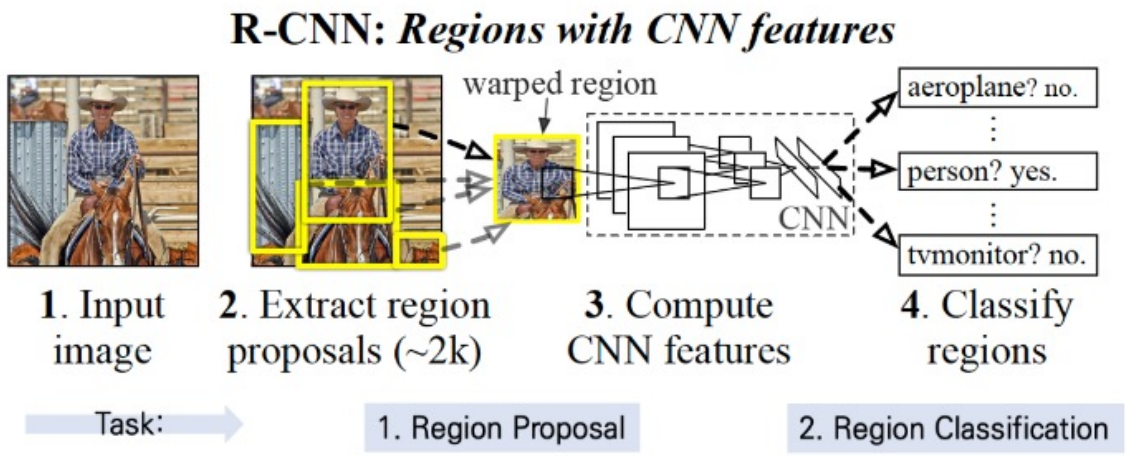
Two-stage detector

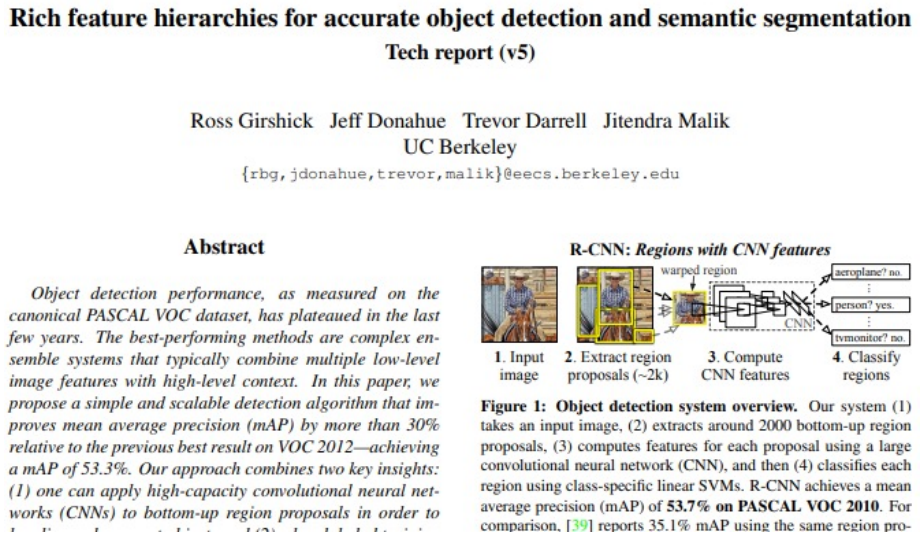
- Regional Convolutional Neural Network (R-CNN)
- Fast R-CNN
- Faster R-CNN
- Mask R-CNN

Regional Convolutional Neural Network

- CNN을 object detection 분야에 최초 적용
- 초기에는 end-to-end 형식은 아님
- 설정한 region을 CNN의 feature로 활용

- region proposal ➔ warping ➔ feature extraction
  ➔ image classification and bbox regression




Rich feature hierarchies for accurate object detection and semantic segmentation
Tech report (v5)

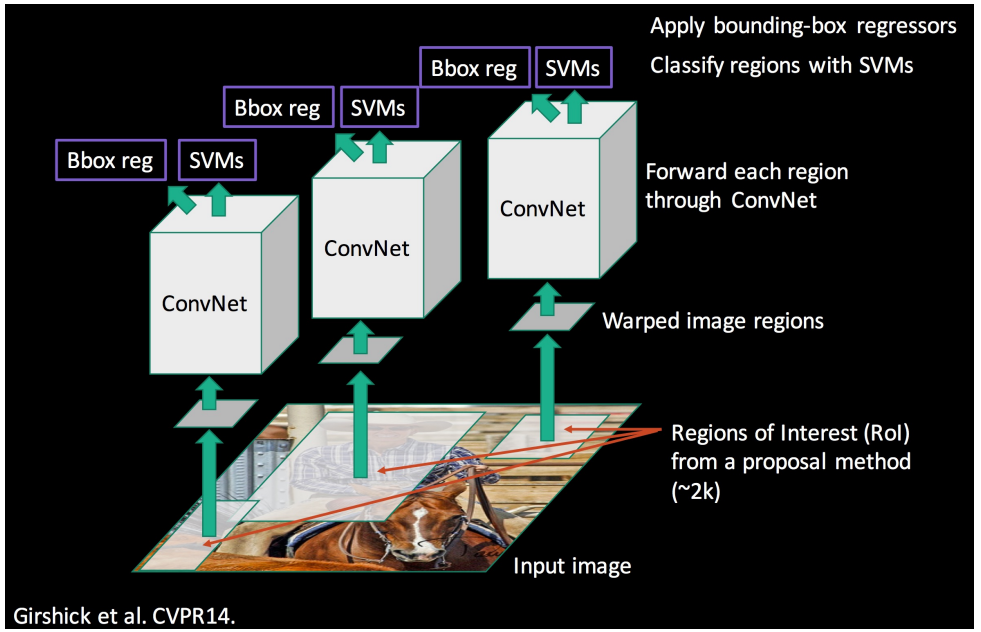
Ross Girshick Jeff Donahue Trevor Darrell Jitendra Malik
UC Berkeley
{rbg,jdonahue,trevor,malik}@eecs.berkeley.edu

Abstract

Object detection performance, as measured on the canonical PASCAL VOC dataset, has plateaued in the last few years. The best-performing methods are complex ensemble systems that typically combine multiple low-level image features with high-level context. In this paper, we propose a simple and scalable detection algorithm that improves mean average precision (mAP) by more than 30% relative to the previous best result on VOC 2012—achieving a mAP of 53.3%. Our approach combines two key insights: (1) one can apply high-capacity convolutional neural networks (CNNs) to bottom-up region proposals in order to

R-CNN: *Regions with CNN features*
warped region
aeroplane? no.
person? yes.
tvmonitor? no.
CNN
1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

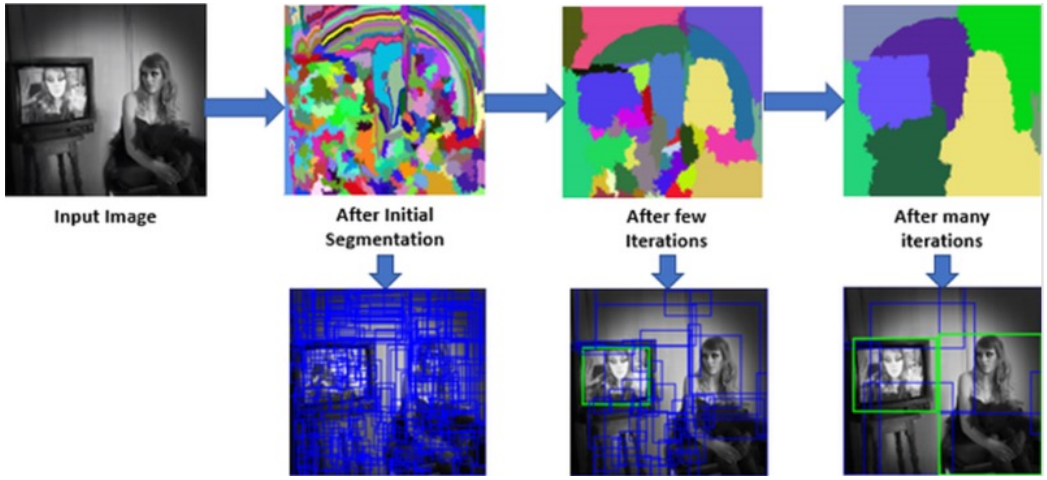
Figure 1: Object detection system overview. Our system (1) takes an input image, (2) extracts around 2000 bottom-up region proposals, (3) computes features for each proposal using a large convolutional neural network (CNN), and then (4) classifies each region using class-specific linear SVMs. R-CNN achieves a mean average precision (mAP) of **53.7% on PASCAL VOC 2010**. For comparison, [39] reports 35.1% mAP using the same region pro-

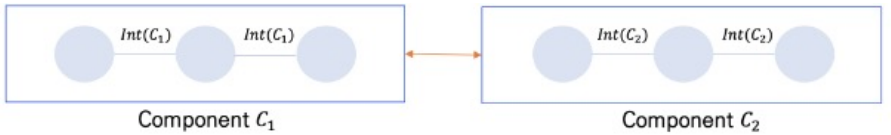


Girshick et al. CVPR14.

Regional Convolutional Neural Network

- Algorithm
  - Step 1: image input
  - Step 2: region proposal
    - selective search algorithm
      - 객체 주변의 color, texture 차이 등을 파악하고, merge하면서 영역화
      - 2,000여개의 ROI (region of interests) 추출
      - 비교적 random한 bbox 생성





$$D(C_1, C_2) = \begin{cases} true \;\; if\; Dif\,(C_1, C_2) > MInt(C_1, C_2) \\ false\,(\rightarrow \text{영역합침}) \quad otherwise \end{cases}$$

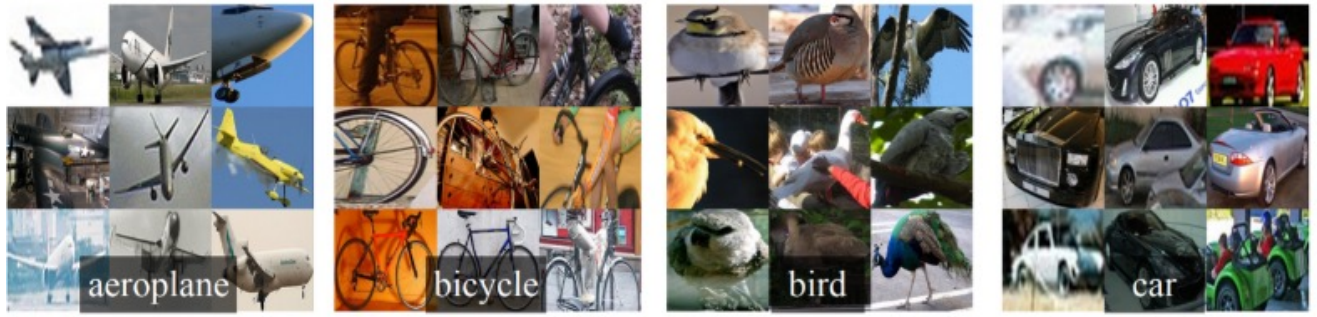
$$MInt(C_1, C_2) = \min\{\,\mathrm{Int}(C_1) + \tau(C_1)\,, \mathrm{Int}(C_2) + \tau(C_2)\,\} \quad \rightarrow \quad \tau(C_2) = \frac{k}{|C|}$$

$$\mathrm{Int}(C_1) = \max_{e \in MST(C,E)} w(e)$$

$$Dif(C_1, C_2) = \min_{v_i \in C_1, v_j \in C_2, (v_i, v_j) \in E} w((v_i, v_j))$$

Regional Convolutional Neural Network

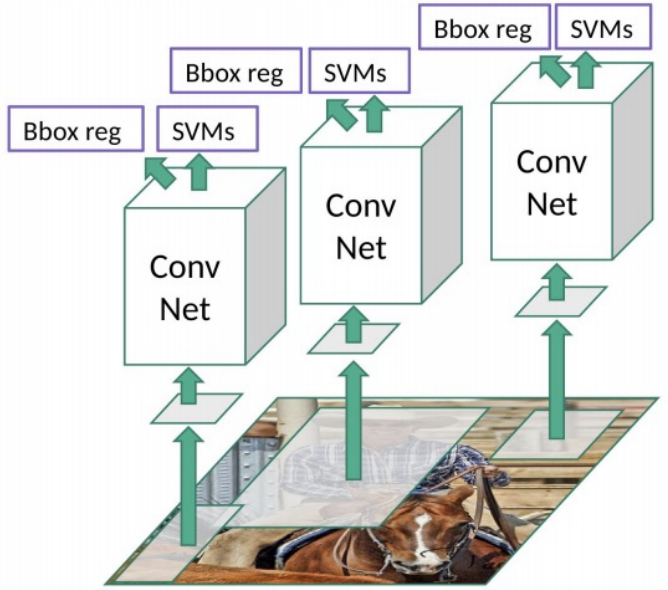
- Algorithm
  - Step 3: Warping
    - warping: 데이터를 고정된 크기로 바꾸는 작업
    - 영상 크기를 227x227로 통일
    - 각기 다른 크기로 제안된 bbox를 모두 정사각형 형태로 변형 ➔ 정보 손실 발생

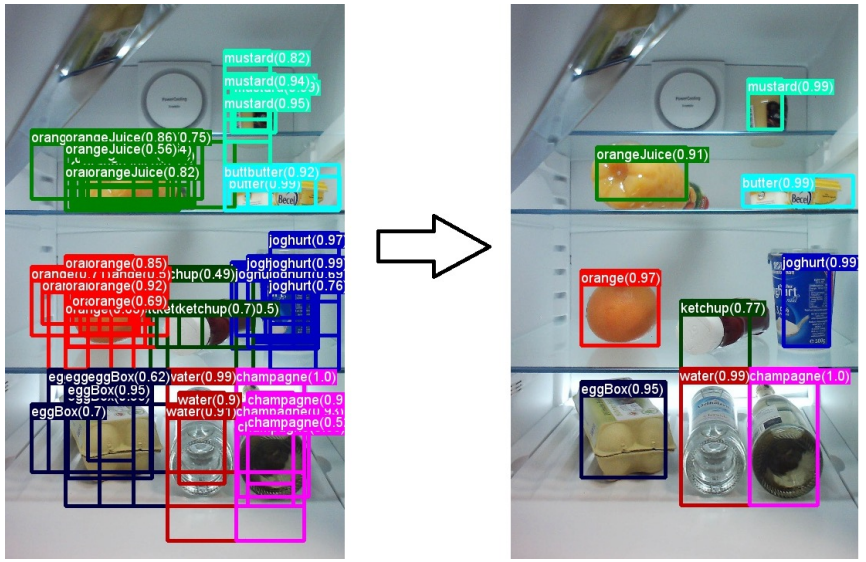


  - Step 4: CNN feature extraction
    - 2,000개의 이미지 x 227 x 227를 CNN의 입력으로 하여 분류 시작
      - CNN부분은 AlexNet과 똑같음

Regional Convolutional Neural Network

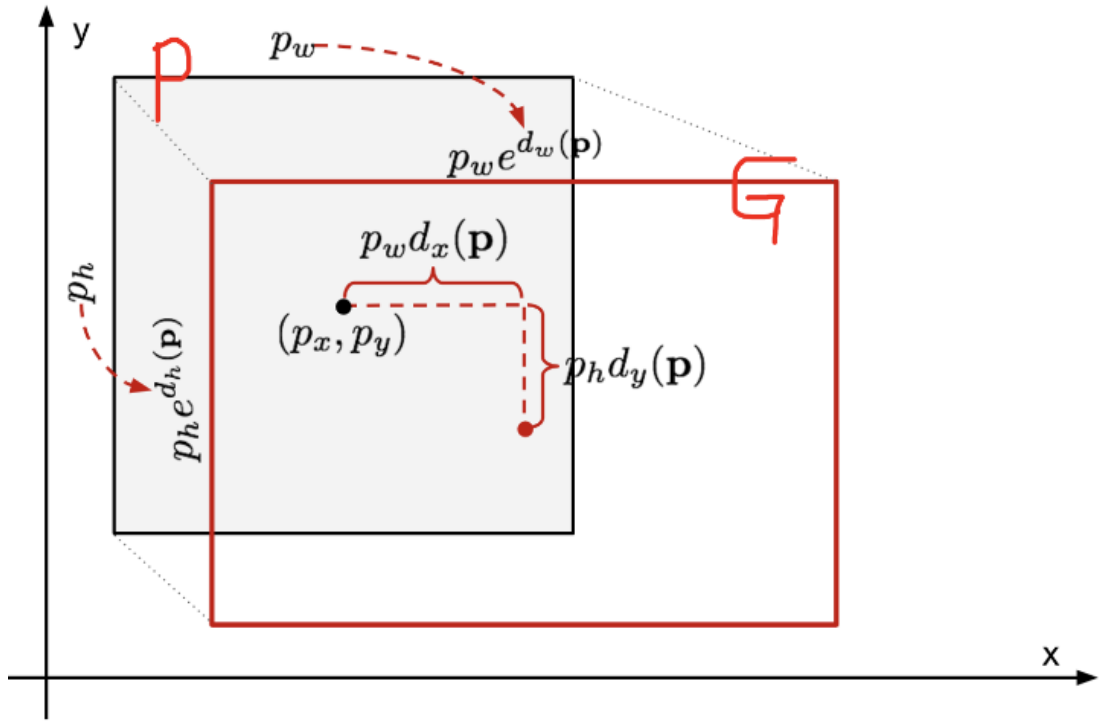
- Algorithm
  - Step 5: Image classification with SVM
    - CNN을 통해 추출된 벡터를 각 class 별로 SVM 모델 학습
    - Output: 2,000개의 bbox의 class 확률값
  - Step 6: Non-maximum suppression (NMS)
    - 주변에서 가장 확률값이 큰 bbox만 인정





Regional Convolutional Neural Network

- Algorithm
  - Step 7: Bounding Box Regression (BBR)
    - Selective search에서 나타난 bbox 위치 (P)와 ground truth bbox (G)간의 차이를 조정하는 linear regression
    - ➔ bbox 위치를 교정 ➔ 모델 성능 극대화



Regional Convolutional Neural Network

- Limitations
  - 이미지의 크기를 강제로 변형함 (227x227)
  - AlexNet 구조를 그대로 활용한 특징추출 및 분류기
  - selective search의 활용
    - region proposal 2,000개가 모두 input ➔ 시간 너무 오래 걸림
  - SVM, selective search ➔ GPU 연산에 최적화 시킬 수 없음
  - end-to-end training X
    - SVM, bbox regressor 학습 ➔ CNN (AlexNet)으로 backpropagration 불가능

Fast R-CNN

- Spatial pyramid pooling layer (SPP layer, SPPNet)
  - 이미지의 사이즈나 스케일에 상관없이 고정된 길이의 벡터 생성
  - ➔ Selective search를 통해서 찾은 RoI를 "RoI pooling layer"를 통해 고정된 사이즈의 feature를 추출

Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun

Abstract—Existing deep convolutional neural networks (CNNs) require a fixed-size (*e.g.*, 224×224) input image. This requirement is "artificial" and may reduce the recognition accuracy for the images or sub-images of an arbitrary size/scale. In this work, we equip the networks with another pooling strategy, "spatial pyramid pooling", to eliminate the above requirement. The new network structure, called SPP-net, can generate a fixed-length representation regardless of image size/scale. Pyramid pooling is also robust to object deformations. With these advantages, SPP-net should in general improve all CNN-based image classification methods. On the ImageNet 2012 dataset, we demonstrate that SPP-net boosts the accuracy of a variety of CNN architectures despite their different designs. On the Pascal VOC 2007 and Caltech101 datasets, SPP-net achieves state-of-the-art classification results using a single full-image representation and no fine-tuning.

The power of SPP-net is also significant in object detection. Using SPP-net, we compute the feature maps from the entire image only once, and then pool features in arbitrary regions (sub-images) to generate fixed-length representations for training the detectors. This method avoids repeatedly computing the convolutional features. In processing test images, our method is 24-102× faster than the R-CNN method, while achieving better or comparable accuracy on Pascal VOC 2007.

In ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014, our methods rank #2 in object detection and #3 in image classification among all 38 teams. This manuscript also introduces the improvement made for this competition.

Index Terms—Convolutional Neural Networks, Spatial Pyramid Pooling, Image Classification, Object Detection

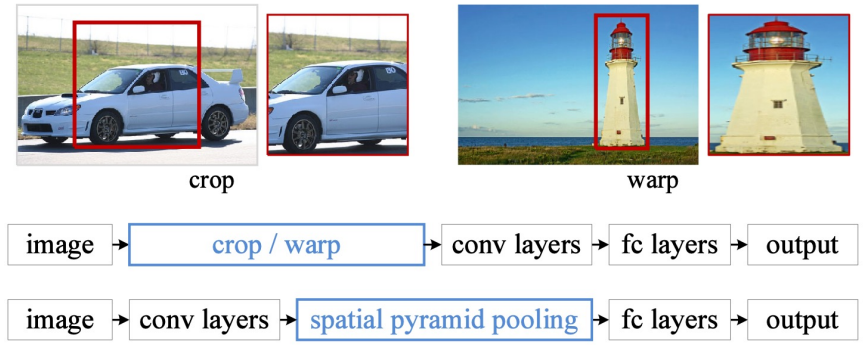
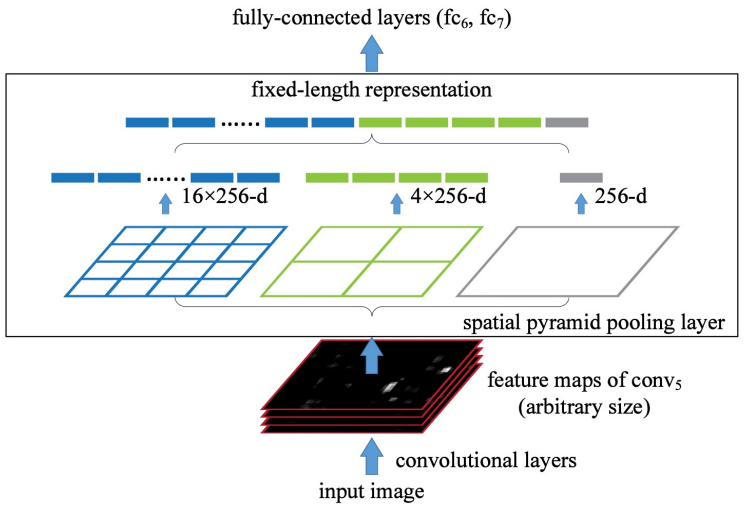


Figure 1: Top: cropping or warping to fit a fixed size. Middle: a conventional CNN. Bottom: our spatial pyramid pooling network structure.



Fast R-CNN

Ross Girshick
Microsoft Research
rbg@microsoft.com

Abstract

This paper proposes a Fast Region-based Convolutional Network method (Fast R-CNN) *for object detection. Fast R-CNN builds on previous work to efficiently classify object proposals using deep convolutional networks. Compared to previous work, Fast R-CNN employs several innovations to improve training and testing speed while also increasing detection accuracy. Fast R-CNN trains the very deep VGG16 network 9× faster than R-CNN, is 213× faster at test-time, and achieves a higher mAP on PASCAL VOC 2012. Compared to SPPnet, Fast R-CNN trains VGG16 3× faster, tests 10× faster, and is more accurate. Fast R-CNN is implemented in Python and C++ (using Caffe) and is available under the open-source MIT License at https://github.com/rbgirshick/fast-rcnn.*

while achieving top accuracy on PASCAL VOC 2012 [7] with a mAP of 66% (vs. 62% for R-CNN).[1]

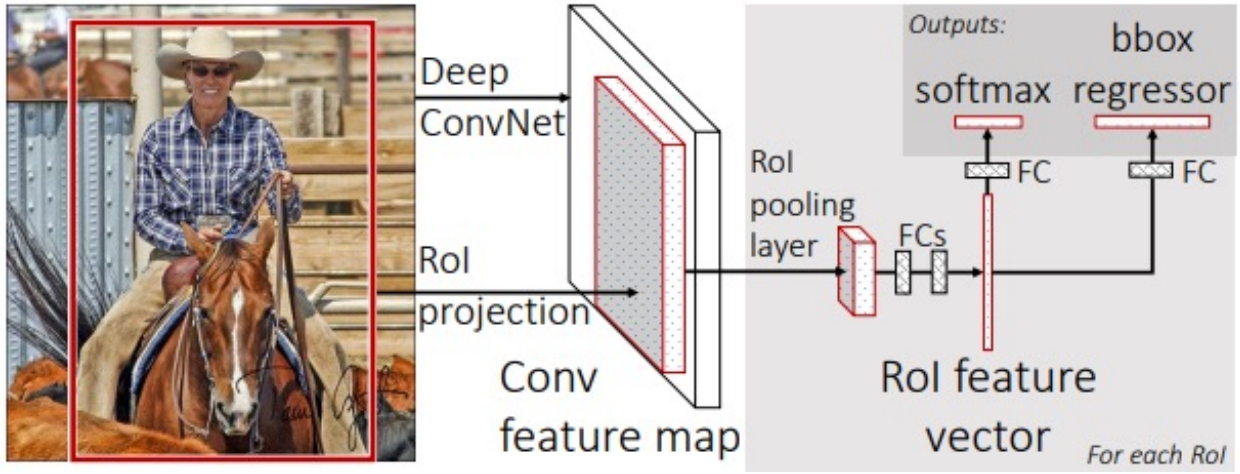
1.1. R-CNN and SPPnet

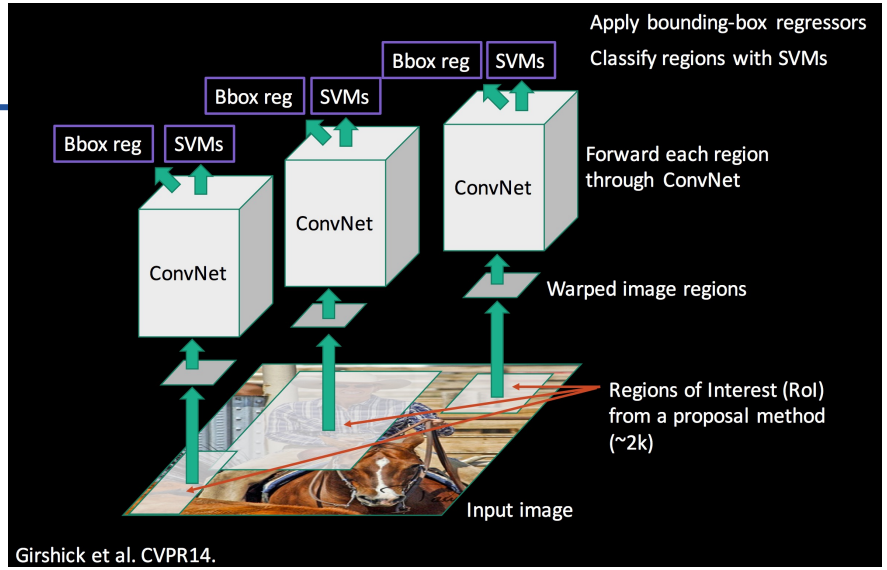
The Region-based Convolutional Network method (R-CNN) [9] achieves excellent object detection accuracy by using a deep ConvNet to classify object proposals. R-CNN, however, has notable drawbacks:

1. **Training is a multi-stage pipeline.** R-CNN first fine-tunes a ConvNet on object proposals using log loss. Then, it fits SVMs to ConvNet features. These SVMs act as object detectors, replacing the softmax classifier learnt by fine-tuning. In the third training stage, bounding-box regressors are learned.
2. **Training is expensive in space and time.** For SVM

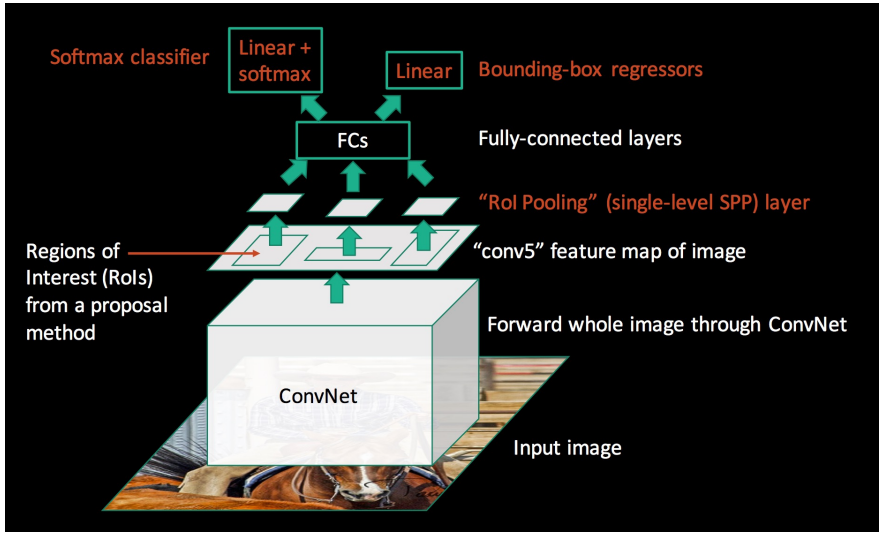
Fast R-CNN

- Region proposal
  - Selective search를 통해 RoI 추출 (R-CNN과 동일)
- 전체 이미지를 CNN에 통과시켜 feature map 추출 + RoI pooling layer
  ➔ feature map 크기 고정
- object classification: softmax
- bbox regression: R-CNN과 동일





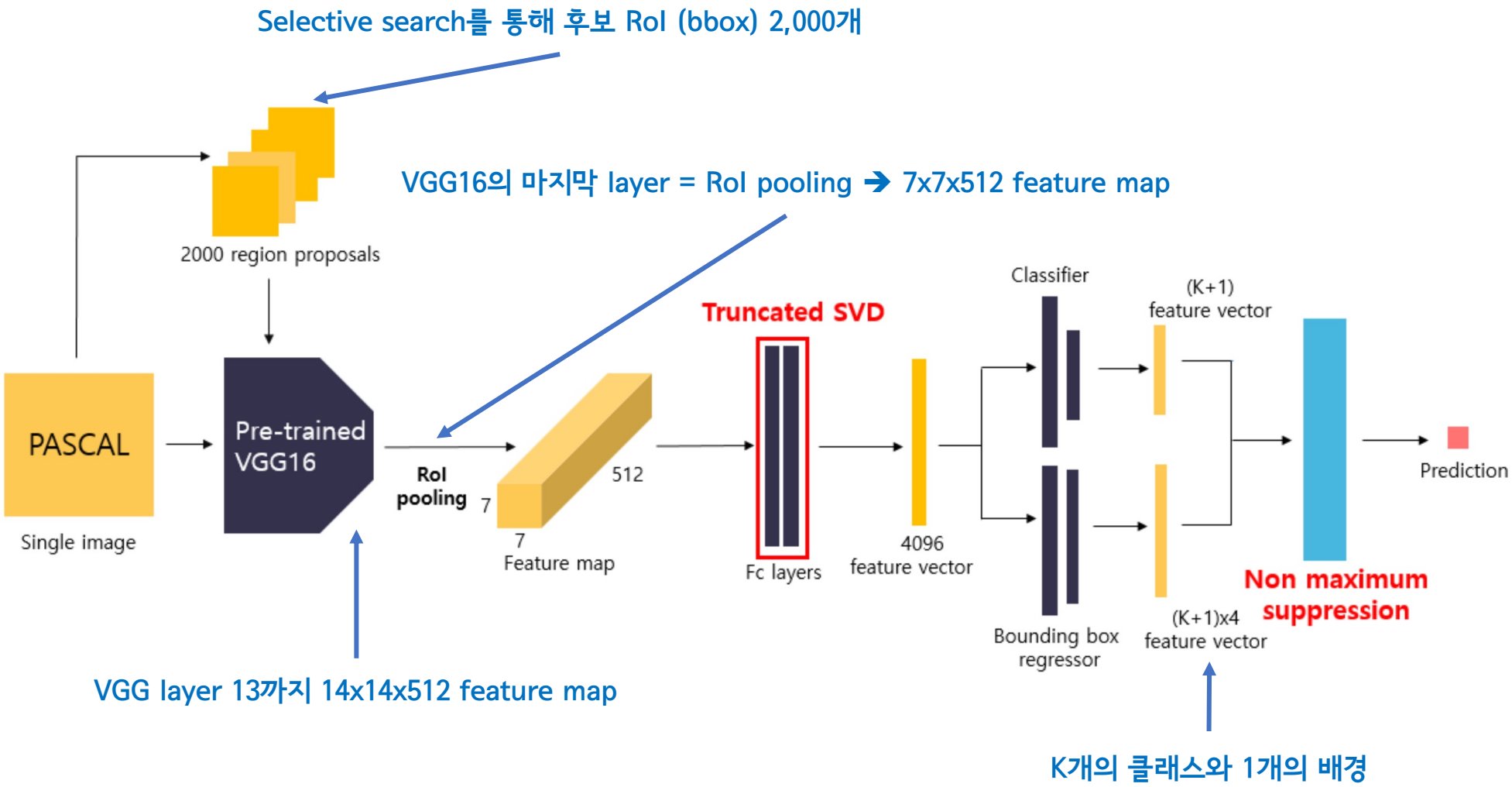
R-CNN



Fast R-CNN

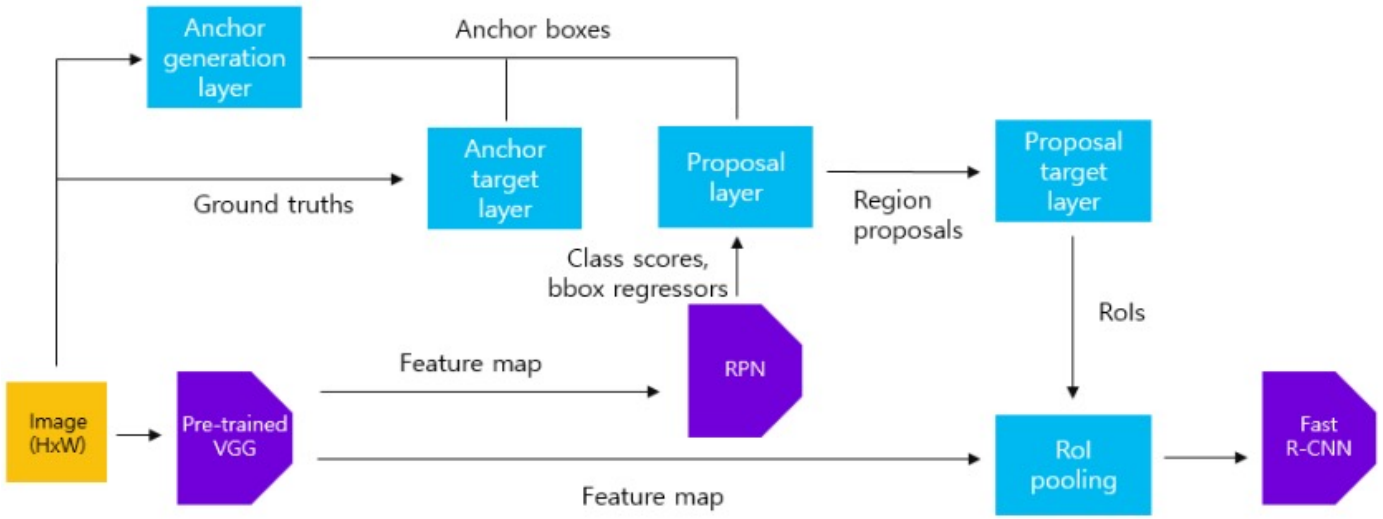
Fast R-CNN

- Main branches



Faster R-CNN

- Region proposal
  - Region Proposal Network (RPN)을 통해 RoI 계산
    - anchor box의 개념을 도입
  - GPU를 통한 RoI 계산가능 ➔ 계산 효율성 증가
  - End-to-end 완성
    - Fast R-CNN + RPN의 결합

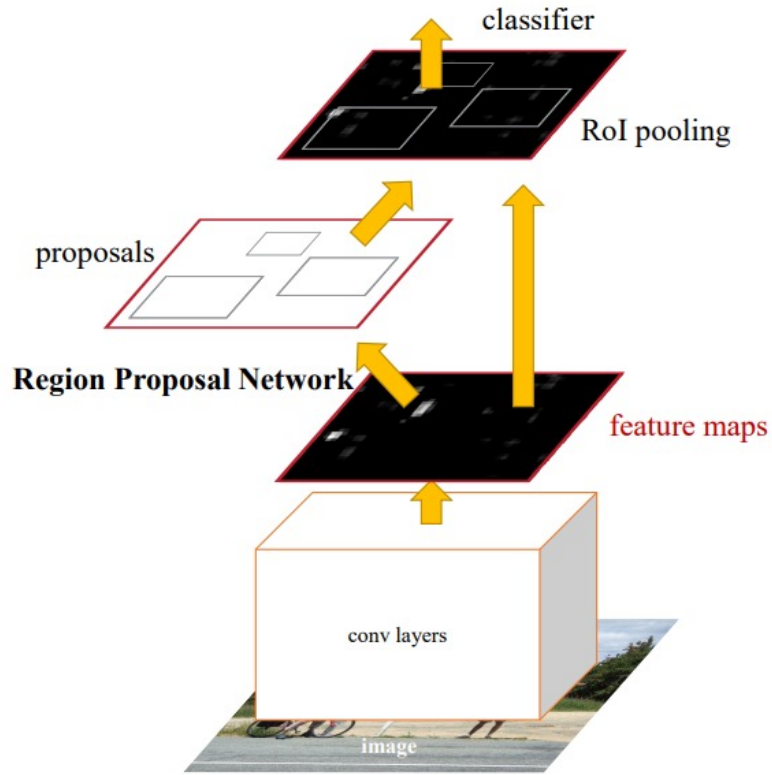


Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun

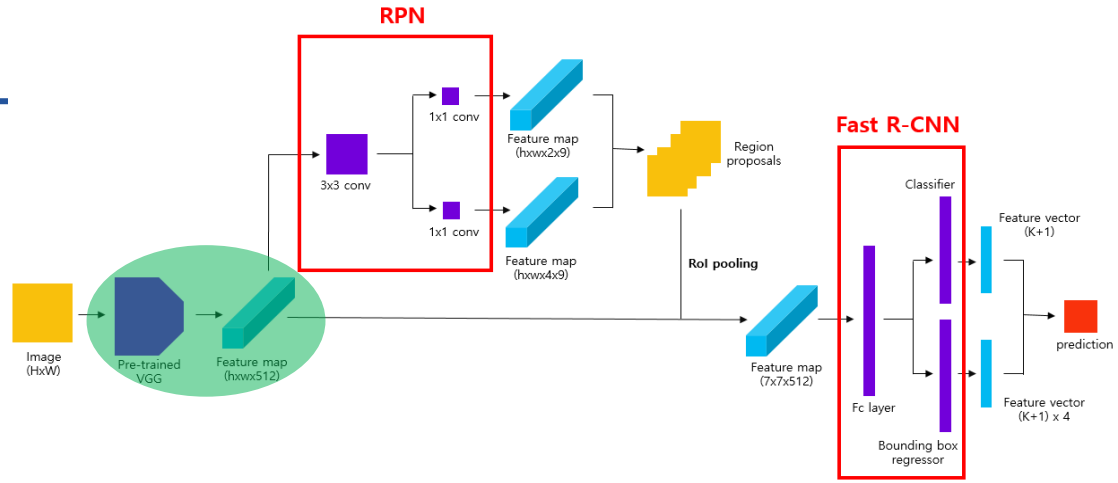
Abstract—State-of-the-art object detection networks depend on region proposal algorithms to hypothesize object locations. Advances like SPPnet [1] and Fast R-CNN [2] have reduced the running time of these detection networks, exposing region proposal computation as a bottleneck. In this work, we introduce a *Region Proposal Network* (RPN) that shares full-image convolutional features with the detection network, thus enabling nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. We further merge RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with "attention" mechanisms, the RPN component tells the unified network where to look. For the very deep VGG-16 model [3], our detection system has a frame rate of 5fps (*including all steps*) on a GPU, while achieving state-of-the-art object detection accuracy on PASCAL VOC 2007, 2012, and MS COCO datasets with only 300 proposals per image. In ILSVRC and COCO 2015 competitions, Faster R-CNN and RPN are the foundations of the 1st-place winning entries in several tracks. Code has been made publicly available.

Index Terms—Object Detection, Region Proposal, Convolutional Neural Network.

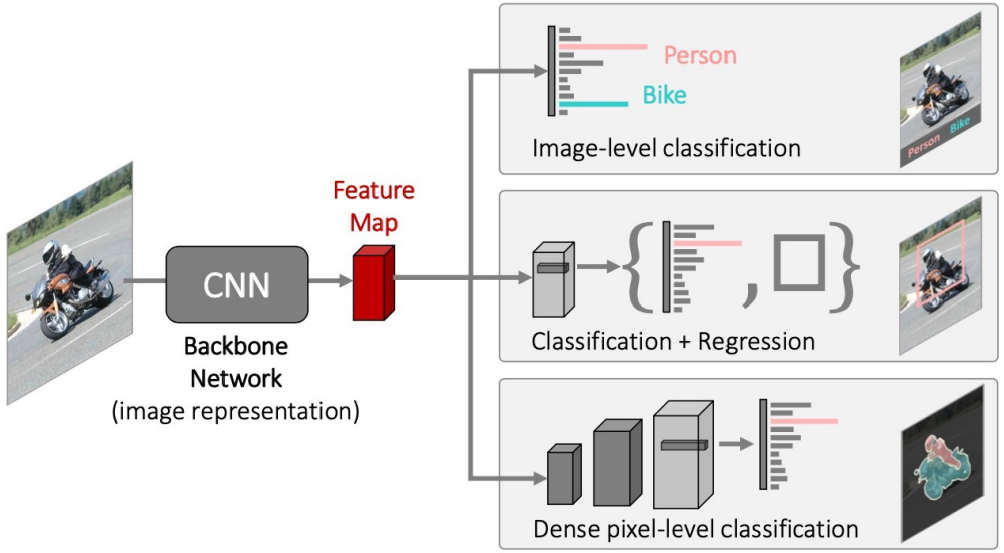


Faster R-CNN

- Network design – feature map 추출
  - pre-trained VGG16을 이용한 feature map 추출



- Backbone의 개념 등장
  - 입력 이미지 ➔ feature map을 추출하기 위한 네트워크 구조
  - 입력 이미지(데이터)의 공간적인 정보를 추출하기 위해 conv-pooling-activation의 블럭의 반복
  - object detection에서의 backbone
    - == image classification 모델
    - AlexNet, VGGNet, ResNet, DarkNet, etc.



Faster R-CNN

- Network design – RPN (sliding window + anchor box)
  - RPN의 목적: region proposal 추출
  - VGG를 통과한 feature map을 입력으로 받음
  - sliding window (3x3)의 중심점을 기준으로 k 개의 anchor box를 이동시키면서 객체 후보 영역 특징 추출
    - anchor box: object 크기와 비율이 어떻게 될지 모르니 크기, 비율 별 box를 사전에 정의함
  - ex) feature map크기 = $W \times H$

    ➔ anchor box 수 = $W \times H \times k$
  - Faster R-CNN에서 $k = 9$

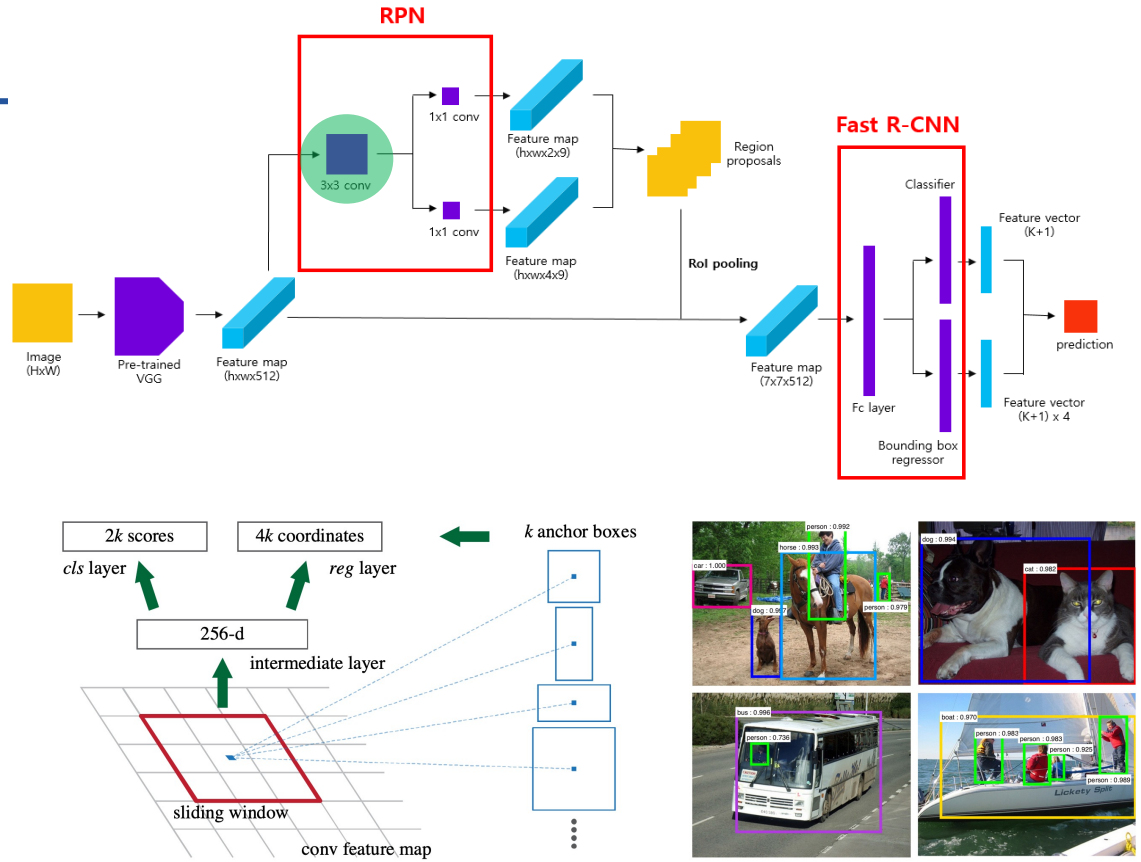
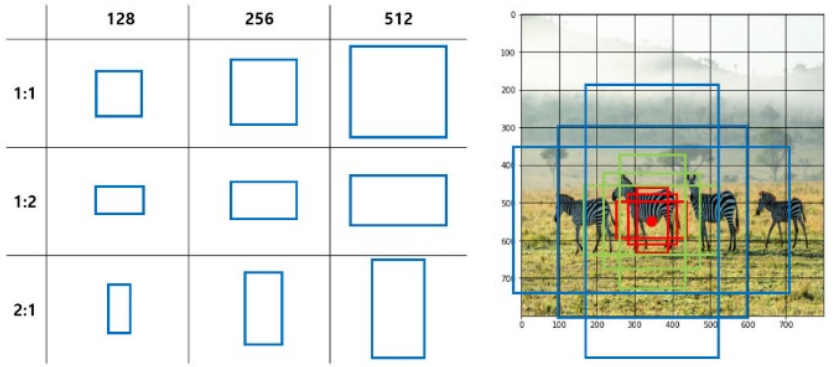
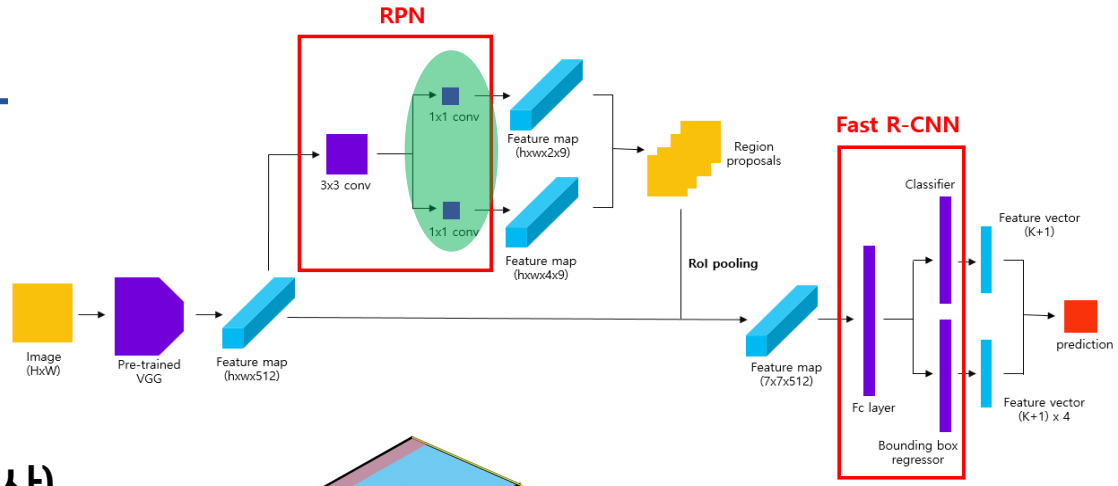


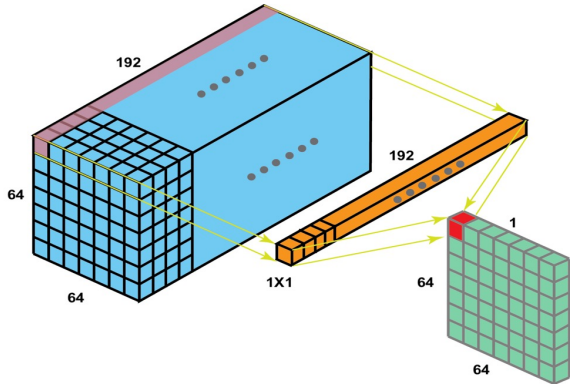
Figure 3: **Left**: Region Proposal Network (RPN). **Right**: Example detections using RPN proposals on PASCAL VOC 2007 test. Our method detects objects in a wide range of scales and aspect ratios.

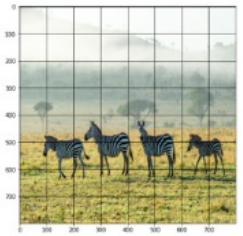


Faster R-CNN

- Architecture - RPN (1x1 conv)
  - 1x1 conv의 의미: feature map 크기 변경 없이 채널 수 조정 (reduction layer 라고도 함)
  - class scoring (class 분류가 아니라 객체 포함 여부만 조사)
    - 1x1 conv 후 channel 수가 $2\times k(=9)$가 되도록 조정
      - 2 = object or not
      - 9 = anchor box 수
  - bbox regressor
    - 1x1 conv 후 channel 수가 $4\times k(=9)$가 되도록 조정
      - 4 = bbox (x1, y1, x2, y2)
      - 9 = anchor box 수





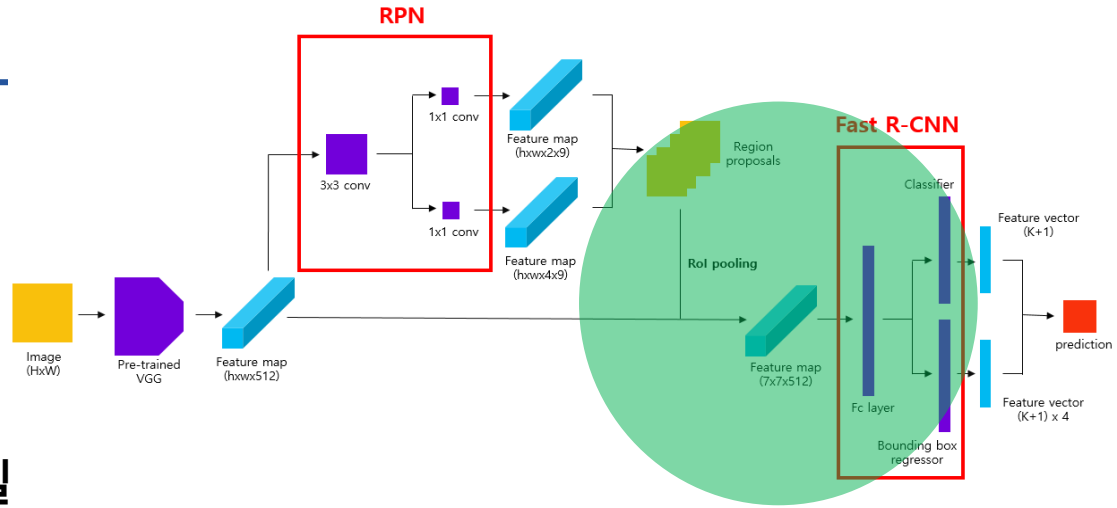


	object O	object X
Anchor type1		
...	...	...
Anchor type 9		

	t_x	t_y	t_w	t_h
Anchor type 1				
...	...	...	...	...
Anchor type 9				

Faster R-CNN

- Network design
  - RoI pooling: Fast R-CNN과 동일
    - 출력 feature map 크기 = 7x7x512
  - object classification / bbox regression: R-CNN과 동일
- Loss function
  - classification과 bbox regression 수행 결과를 종합



$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i L_{reg}(t_i, t_i^*)$$

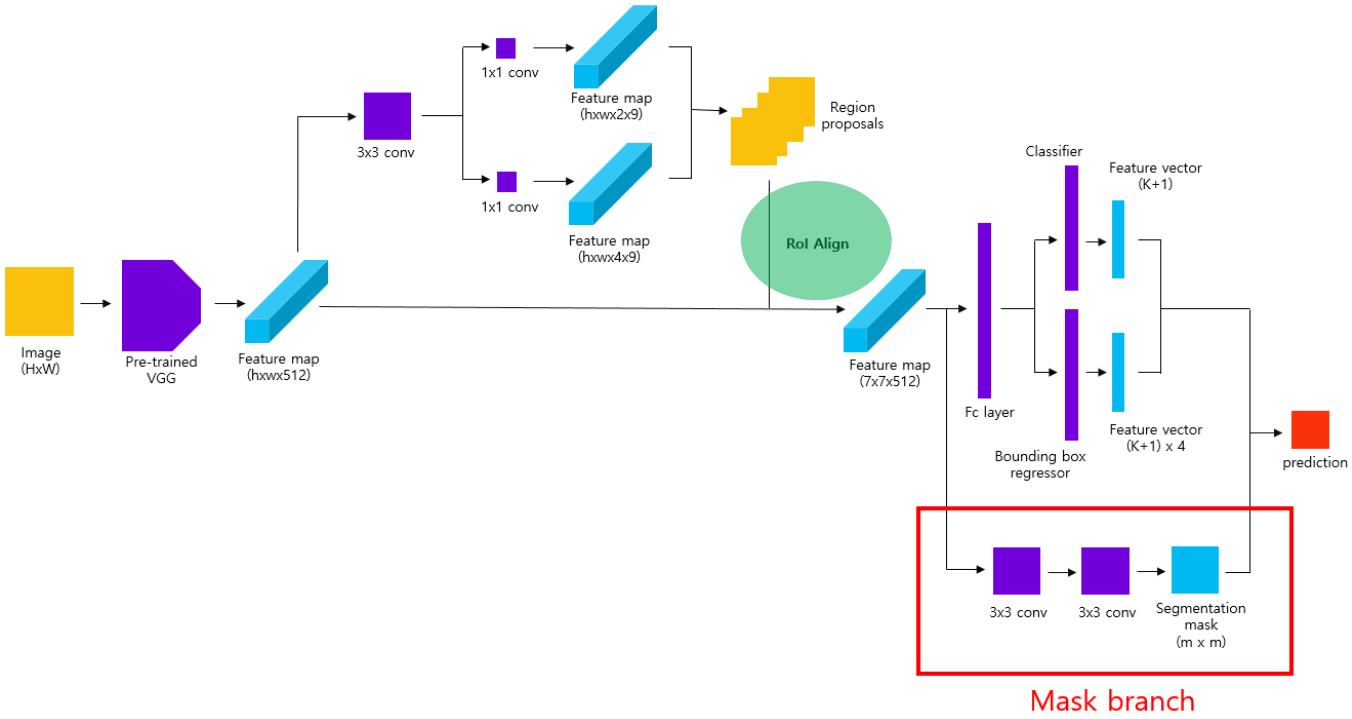
- i: anchor box
- p_i, t_i: classification, bbox에 대한 ground truth

L_{cls}: cross entropy

$$L_{reg}: smooth_{L1}(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases}$$

Mask R-CNN

- Network design
  - Faster R-CNN의 RoI pooling의 feature map에 segmentation mask 예측을 위한 mask branch를 추가



Mask R-CNN

Kaiming He Georgia Gkioxari Piotr Dollár Ross Girshick

Facebook AI Research (FAIR)

Abstract

We present a conceptually simple, flexible, and general framework for object instance segmentation. Our approach efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance. The method, called Mask R-CNN, extends Faster R-CNN by adding a branch for predicting an object mask in parallel *with the existing branch for bounding box recognition. Mask R-CNN is simple to train and adds only a small overhead to Faster R-CNN, running at 5 fps. Moreover,*

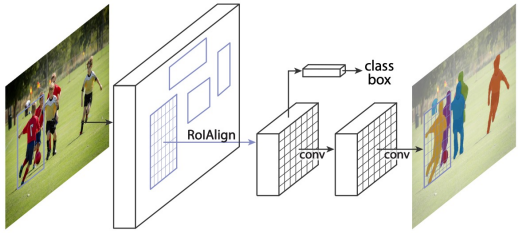


Figure 1. The **Mask R-CNN** framework for instance segmentation.

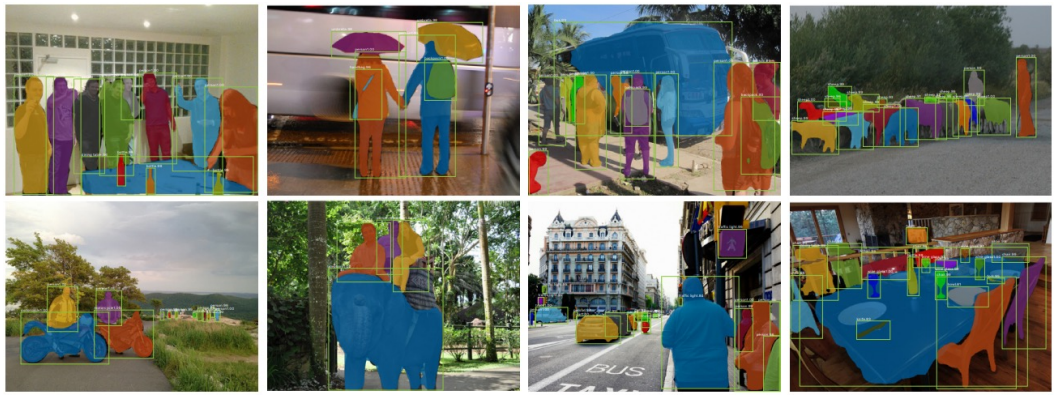


Figure 2. **Mask R-CNN** results on the COCO test set. These results are based on ResNet-101 [19], achieving a *mask* AP of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.

Mask R-CNN

- Network design
  - RoI Align
    - 사실 object detection에서는 정확한 위치 정보에 대한 계산이 필요가 없었음
    - RoI가 소수점 좌표일 때 반올림하여 pooling
    - RoI pooling: 입력 이미지의 크기와 상관없이 고정된 크기의 feature map 획득 가능
      - ➔ RoI pooling으로 얻은 feature와 RoI가 어긋나는 misalignment가 있다고 주장
    - mask branch에서 영역 내 점의 값을 계산하기 위해 bilinear interpolation (쌍선형 보간법) 활용

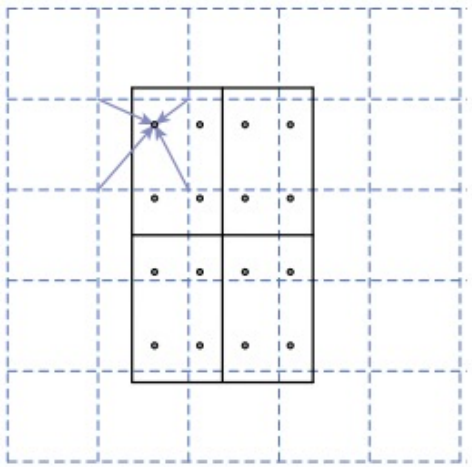
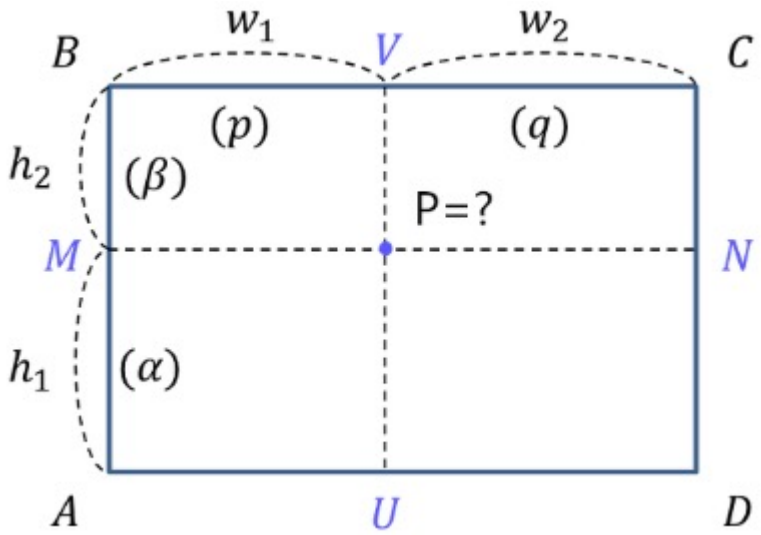
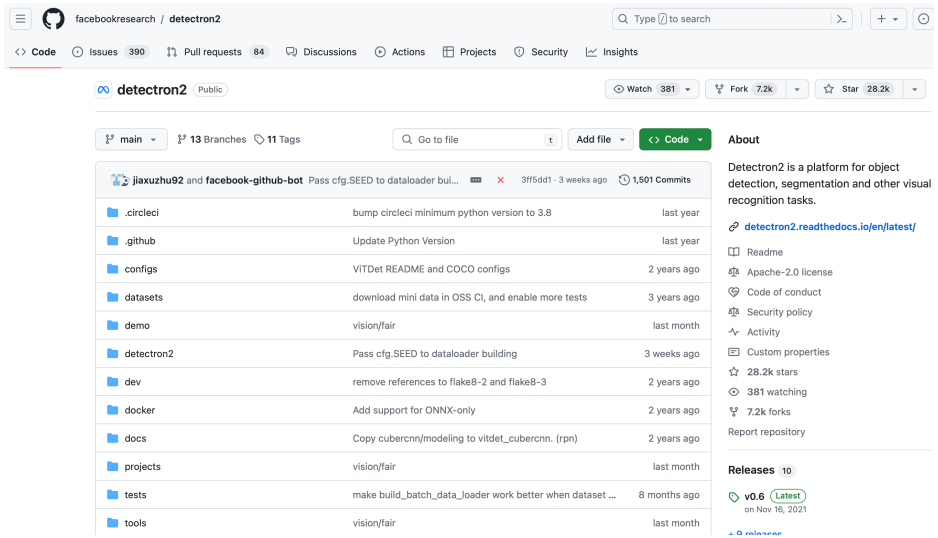


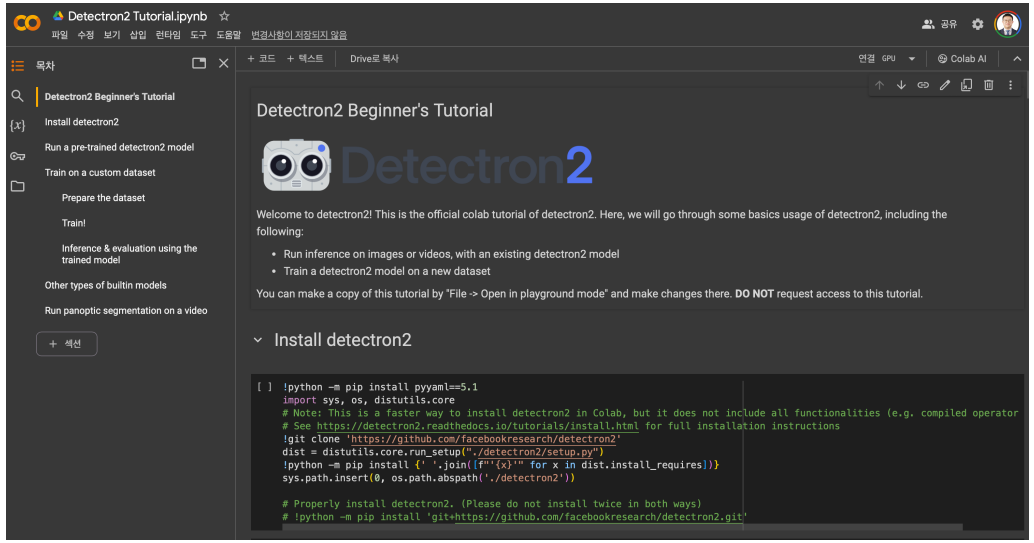
Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an RoI (with 2×2 bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the RoI, its bins, or the sampling points.



프로그래밍 실습: Mask R-CNN

- Github:
  - R-CNN: https://github.com/rbgirshick/rcnn
  - Fast R-CNN https://github.com/rbgirshick/fast-rcnn
  - Faster R-CNN https://github.com/jwyang/faster-rcnn.pytorch
  - Mask R-CNN https://github.com/matterport/Mask_RCNN
  - **Detectron2: https://github.com/facebookresearch/detectron2**





One-stage detector

- You Only Look Once: Unified, Real-time Object detection (2016 CVPR) ← YOLOv1
- YOLO9000: Better, Faster, Stronger (2016 CVPR) ← YOLOv2
- YOLOv3: An Incremental Improvement (2018)
- YOLOv4: Optimal speed and accuracy of object detection (2020)
- YOLOv5 (2020) ➔ 공식 paper 없음. by Ultralytics
- End-to-End Object Detection with Transformer (2020 ECCV) ← DETR
- YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors (2022)

One-stage detector

- YOLO v1, v2, v3
  - v3: https://github.com/ultralytics/yolov3/tree/master
- YOLO v4 (v4->v7) https://github.com/WongKinYiu/PyTorch_YOLOv4.git
- YOLO v5 (v3->v5->v8) https://github.com/ultralytics/yolov5
- YOLO v6 https://github.com/meituan/YOLOv6
- YOLO v7 (2022) https://github.com/WongKinYiu/yolov7
- YOLO v8 (2023) https://github.com/ultralytics/ultralytics

History of major object detector

- Rich feature hierarchies for accurate object detection and semantic segmentation (2014 CVPR) ← R-CNN
- Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition (2014 ECCV) ← SPPNet
- Fast R-CNN (2015 ICCV)
- Faster R-CNN (2015 NIPS)
- You Only Look Once: Unified, Real-time Object detection (2016 CVPR) ← YOLOv1
- YOLO9000: Better, Faster, Stronger (2016 CVPR) ← YOLOv2
- Mask R-CNN (2017 ICCV)
- YOLOv3: An Incremental Improvement (2018)
- Path Aggregation Network for Instance Segmentation (2018 CVPR) ← PANet
- YOLOv4: Optimal speed and accuracy of object detection (2020)
- A New Backbone that can Enhance Learning Capability of CNN (2019) ← CSPNet
- YOLOv5 (2020) ➔ 공식 paper 없음. by Ultralytics
- End-to-End Object Detection with Transformer (2020 ECCV) ← DETR
- YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors (2022)

YOLOv1

- You Only Look Once: Unified, Real-time Object Detection
  - 전체 이미지를 보는 횟수 = 1회
  - Unified: one-stage
    - classification과 localization을 한 번에 처리
  - Real-time: 속도 개선 (45 fps)
    - Faster R-CNN: 5~7 fps

You Only Look Once:
Unified, Real-Time Object Detection

Joseph Redmon*, Santosh Divvala*†, Ross Girshick¶, Ali Farhadi*†
University of Washington*, Allen Institute for AI†, Facebook AI Research¶
http://pjreddie.com/yolo/

Abstract

We present YOLO, a new approach to object detection. Prior work on object detection repurposes classifiers to perform detection. Instead, we frame object detection as a regression problem to spatially separated bounding boxes and associated class probabilities. A single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation. Since the whole detection pipeline is a single network, it can be optimized end-to-end directly on detection performance.

Our unified architecture is extremely fast. Our base YOLO model processes images in real-time at 45 frames per second. A smaller version of the network, Fast YOLO, processes an astounding 155 frames per second while still achieving double the mAP of other real-time detectors. Compared to state-of-the-art detection systems, YOLO makes more localization errors but is less likely to predict false positives on background. Finally, YOLO learns very general representations of objects. It outperforms other detection methods, including DPM and R-CNN, when generalizing from natural images to other domains like artwork.

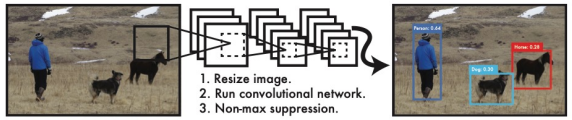
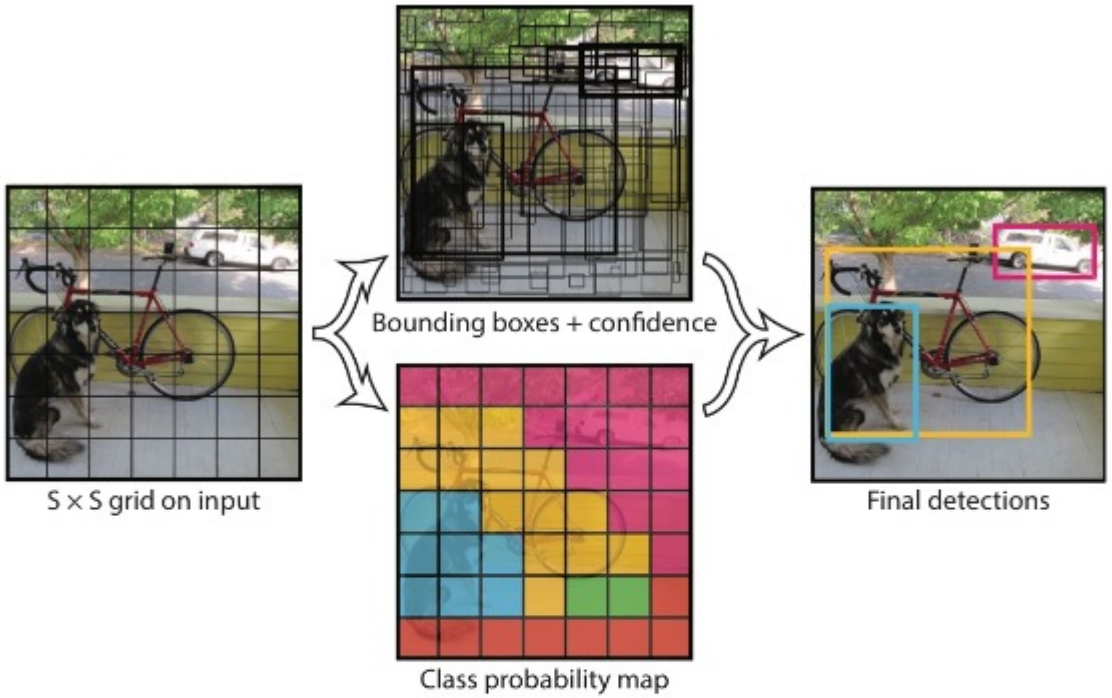


Figure 1: The YOLO Detection System. Processing images with YOLO is simple and straightforward. Our system (1) resizes the input image to 448 × 448, (2) runs a single convolutional network on the image, and (3) thresholds the resulting detections by the model's confidence.

methods to first generate potential bounding boxes in an image and then run a classifier on these proposed boxes. After classification, post-processing is used to refine the bounding boxes, eliminate duplicate detections, and rescore the boxes based on other objects in the scene [13]. These complex pipelines are slow and hard to optimize because each individual component must be trained separately.

We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities. Using our system, you only

YOLOv1

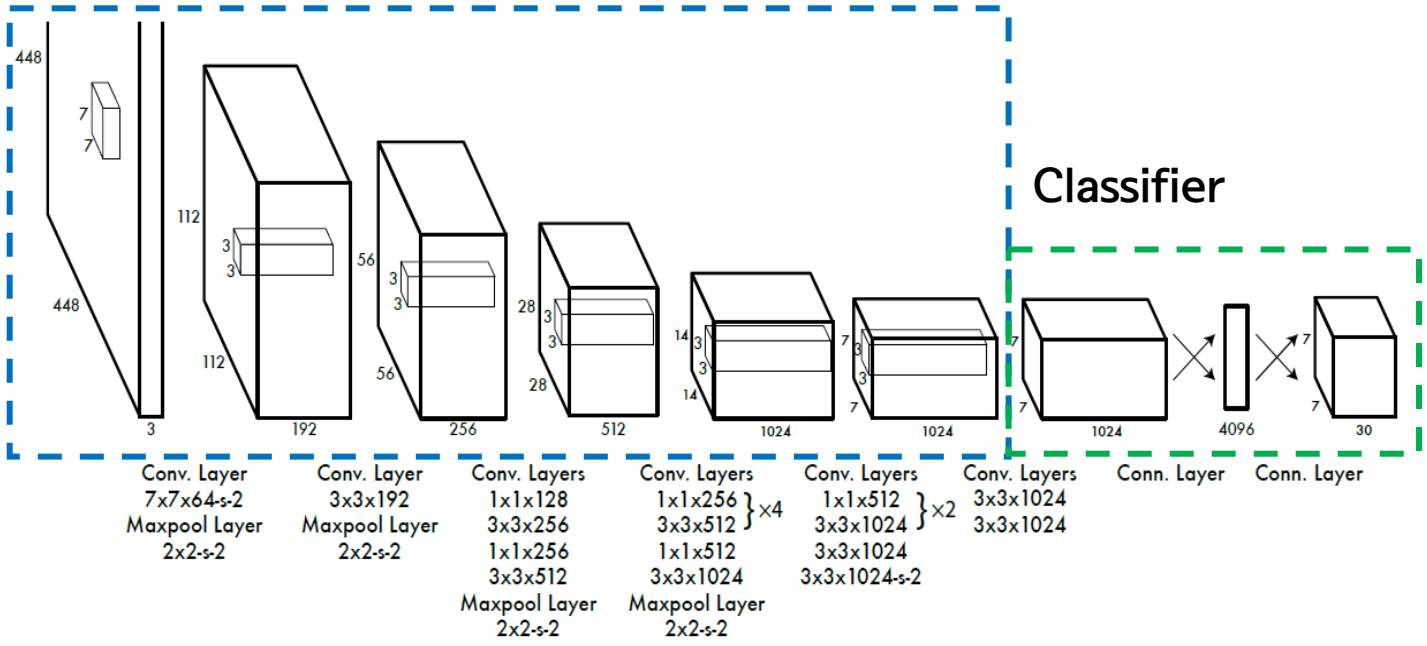


- Region proposal, feature extraction, classification, bbox regression ➔ 하나의 network에서 처리
  - 이미지 전체에서 얻은 feature map 활용 ➔ 모든 클래스에 대한 확률 계산 및 bbox 예측

YOLOv1

- Network design
  - Modified GoogleNet
    - 24개의 convolution layers, 2개의 fully-connected layers

Pre-trained

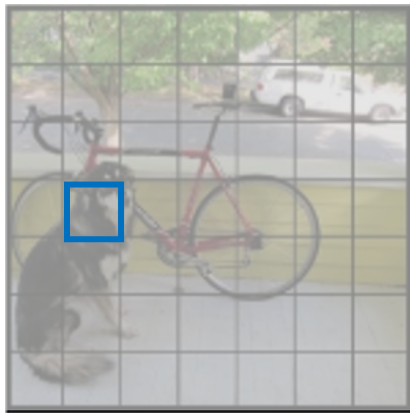


- input tensor: 448x448x3
- output tensor $= S \times S \times (B * 5 + C) = 7 \times 7 \times 30$
  - S (=7)
  - B (=2): number of boxes
  - C (=20): class probabilities,
  - 5: bounding box center inside cell (x, y), bounding box width and height (w, h), bounding box confidence

YOLOv1

- Output tensor 계산
  - ex) S = 7, B = 20, C = 20

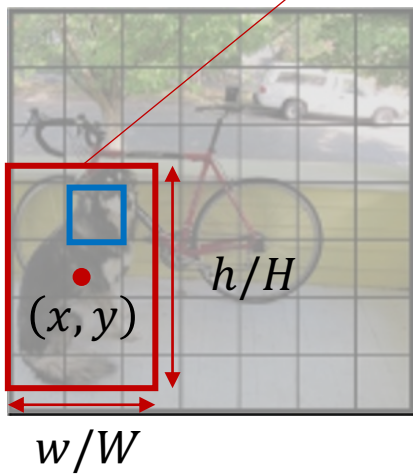
Resized image를 7x7로 분할 (S=7)



Grid cell 마다 box 2개씩 예측 (B=2)

YOLOv1

- Output tensor 계산
  - ex) S = 7, B = 20, C = 20



bbox #1

$$\begin{pmatrix} x \\ y \\ w \\ h \\ p_c \end{pmatrix}$$

bbox의 좌표 (x, y, w, h)

input image size로 normalized (0~1)

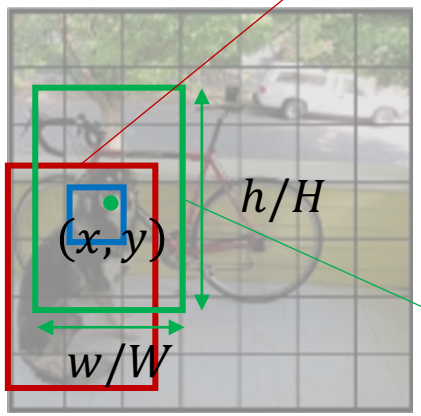
confidence score $= \Pr(\text{Object}) * \text{IoU}_{\text{pred}}^{\text{truth}}$

$\Pr(\text{Object})$: 물체가 bbox 내에 있으면 1, 아니면 0

Grid cell 마다 box 2개씩 예측 (B=2)

YOLOv1

- Output tensor 계산
  - ex) S = 7, B = 20, C = 20



bbox #1

bbox #2

x
y
w
h
p_c
x
y
w
h
p_c

bbox의 좌표 (x, y, w, h)
input image size로 normalized (0~1)

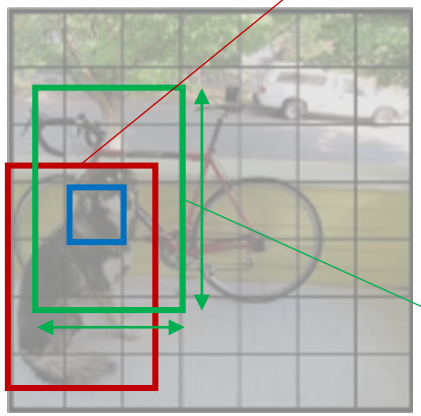
confidence score $= \Pr(\text{Object}) * \text{IoU}_{\text{pred}}^{\text{truth}}$

$\Pr(\text{Object})$: 물체가 bbox 내에 있으면 1, 아니면 0

Grid cell 마다 box 2개씩 예측 (B=2)

YOLOv1

- Output tensor 계산
  - ex) S = 7, B = 20, C = 20



bbox #1

bbox #2

Grid cell 마다 box 2개씩 예측 (B=2)

x
y
w
h
p_c
x
y
w
h
p_c
c_1
c_2
$\vdots$
c_{20}

bbox의 좌표 (x, y, w, h)

input image size로 normalized (0~1)

confidence score $= \Pr(\text{Object}) * \text{IoU}_{\text{pred}}^{\text{truth}}$

$\Pr(\text{Object})$: 물체가 bbox 내에 있으면 1, 아니면 0

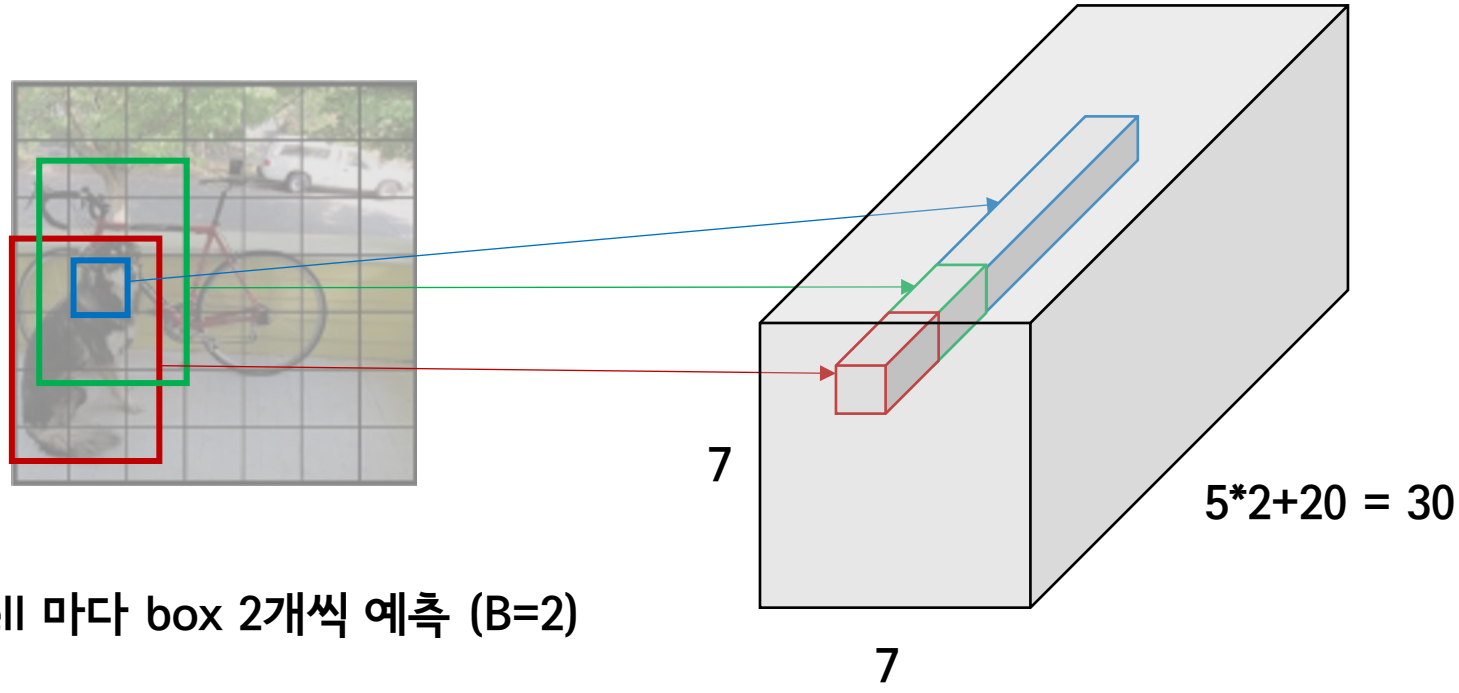
$\Pr(\text{Class}_i|\text{Object})$:

물체가 bbox 내에 있을 때, Grid cell에 있는 object가

i번째 class에 속할 확률

YOLOv1

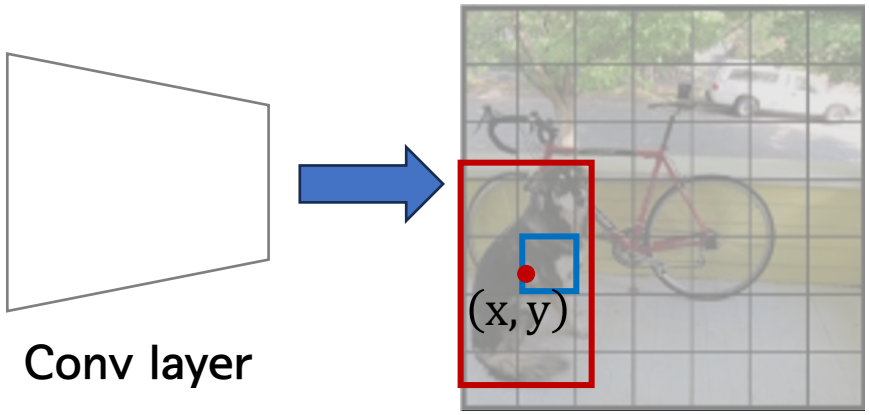
- Output tensor 계산
  - ex) S = 7, B = 20, C = 20



Grid cell 마다 box 2개씩 예측 (B=2)

YOLOv1

- Training process
  - 특정 object에 responsible한 cell i는 ground truth box의 중심이 위치하는 cell로 할당



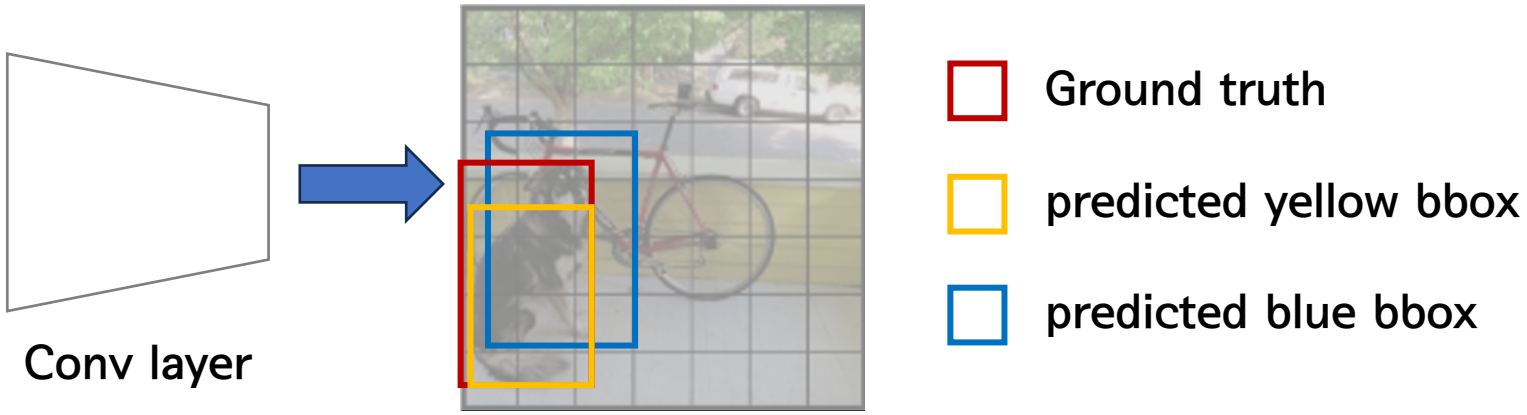
Ground truth

Ground truth box내에 존재하는 object에 responsible한 cell

  - GT box의 중심이 cell 30에 위치

YOLOv1

- Training process
  - 학습단계에서는 IoU_{pred}^{truth}가 가장 높은 bbox 1개만 사용
  - ➔ $\mathbb{I}_{i,j}^{obj}$: cell i에서 responsible한 j번째 bbox를 표시하여 loss function에 반영



  - GT box의 중심이 cell 30에 위치
  - 노란색 bbox만 학습에 사용 ➔ $\mathbb{I}_{30,yellow}^{obj} = 1 \ \left(\because IoU_{yellow}^{truth} > IoU_{blue}^{truth}\right)$

YOLOv1

- Loss function

Localization loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Confidence loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

YOLOv1

- Loss function

Localization loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Confidence loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Localization loss

- λ_{coord}: grid cell의 가중치 (=5)
  - bbject가 존재하지 않은 cell이 더 많음

    ➔ confidence score가 0에 수렴하는 것을 방지
- S^2: grid cell의 수 (=7x7=49)
- B: grid별 bbox의 수
- $\mathbb{1}_{i,j}^{\text{obj}}$: i번째 cell의 j번째 Bbox가 객체를 예측 ➔ 1, else 0

YOLOv1

- Loss function

Localization loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Confidence loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification loss

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Confidence loss

- λ_{noobj}: 객체를 포함하지 않는 grid cell의 영향력을 줄이기 위한 가중치 (0.5)
- $1_{i,j}^{\text{noobj}}$: i번째 grid cell의 j번째 Bbox가 객체를 예측 ➔ 0, else 1
- C_i: 객체가 포함되어 있을 경우 1, else 0
- $\hat{C}_i$: 예측한 Bbox의 confidence score

YOLOv1

- Loss function

Localization loss

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

Confidence loss

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

Classification loss

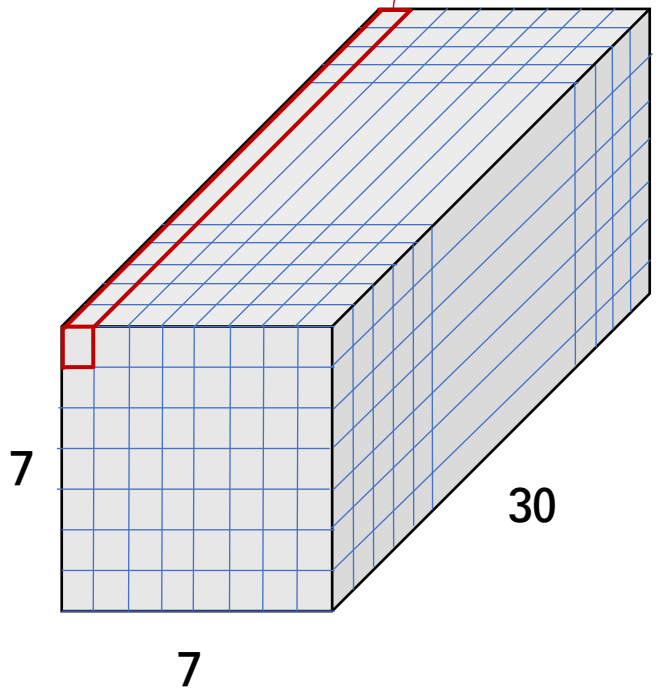
$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

Classification loss

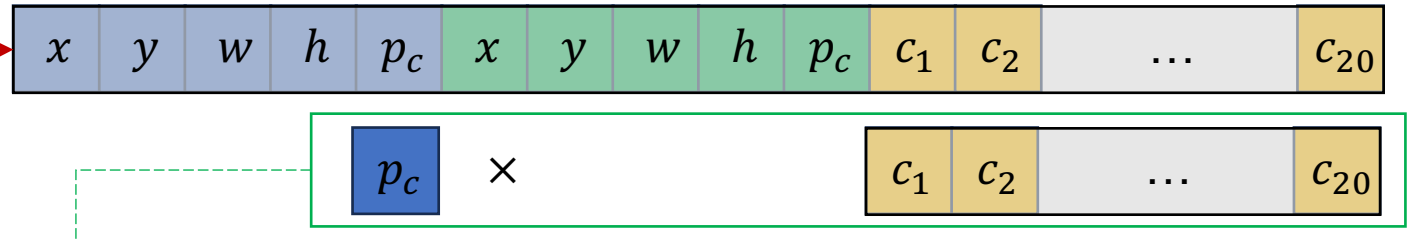
- (실제 class prob)와 (예측 class prob)의 weighted SSE
- 해당 cell에 값이 있을 때만 연산에 참여

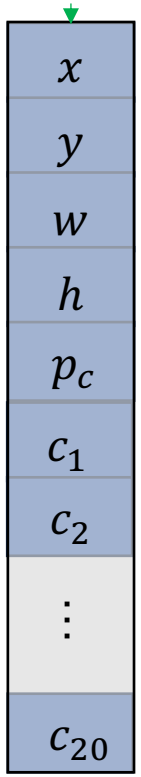
YOLOv1

- Inference stage



Output tensor





Class-specific confidence score

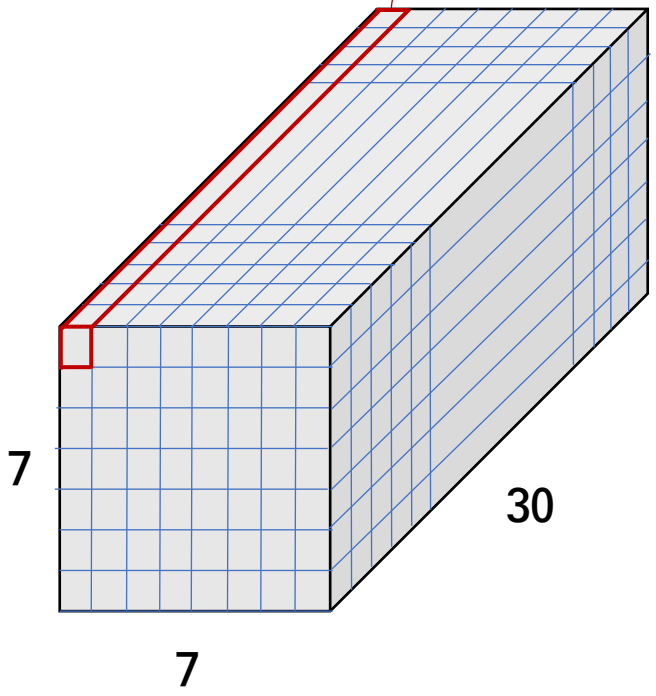
$$Pr(Class_i \mid Object) * p_c$$

$$= Pr(Class_i \mid Object) * Pr(Object) * IoU_{pred}^{truth}$$

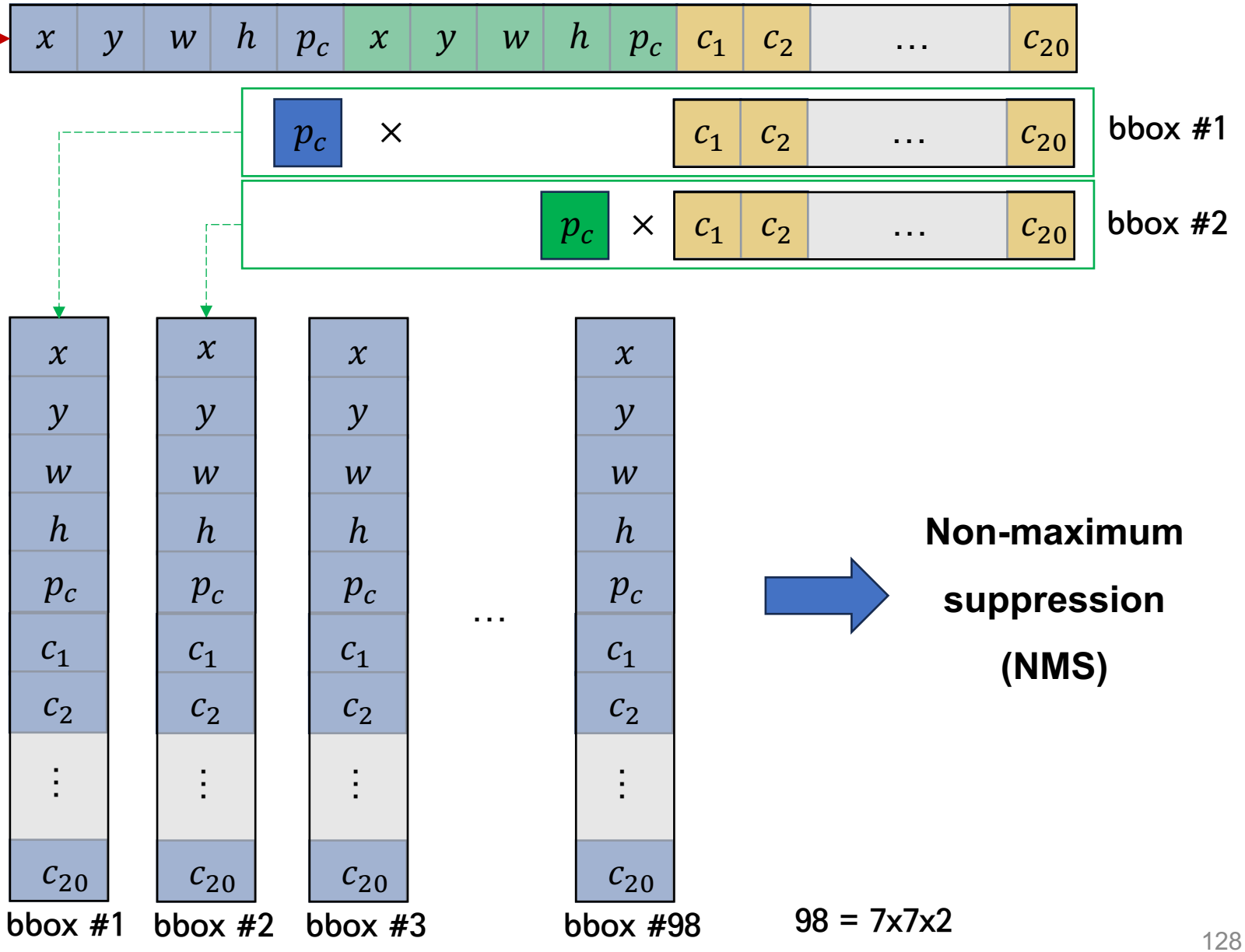
$$= Pr(Class_i) * IoU_{pred}^{truth}$$

YOLOv1

- Inference stage

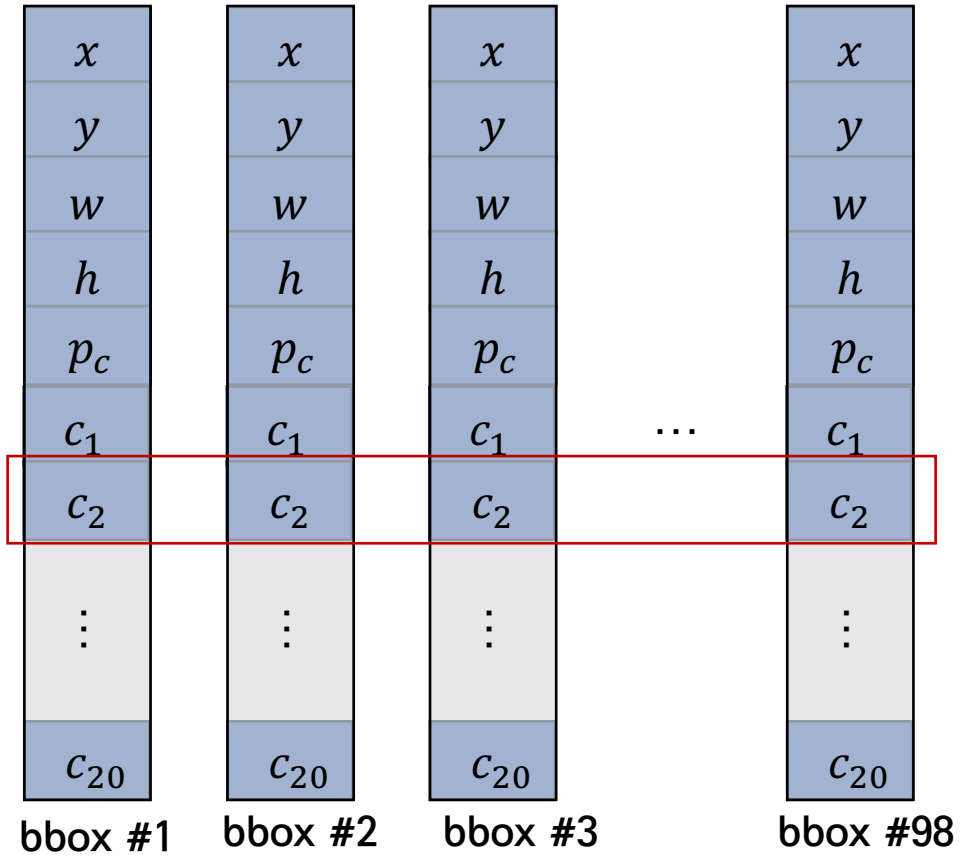


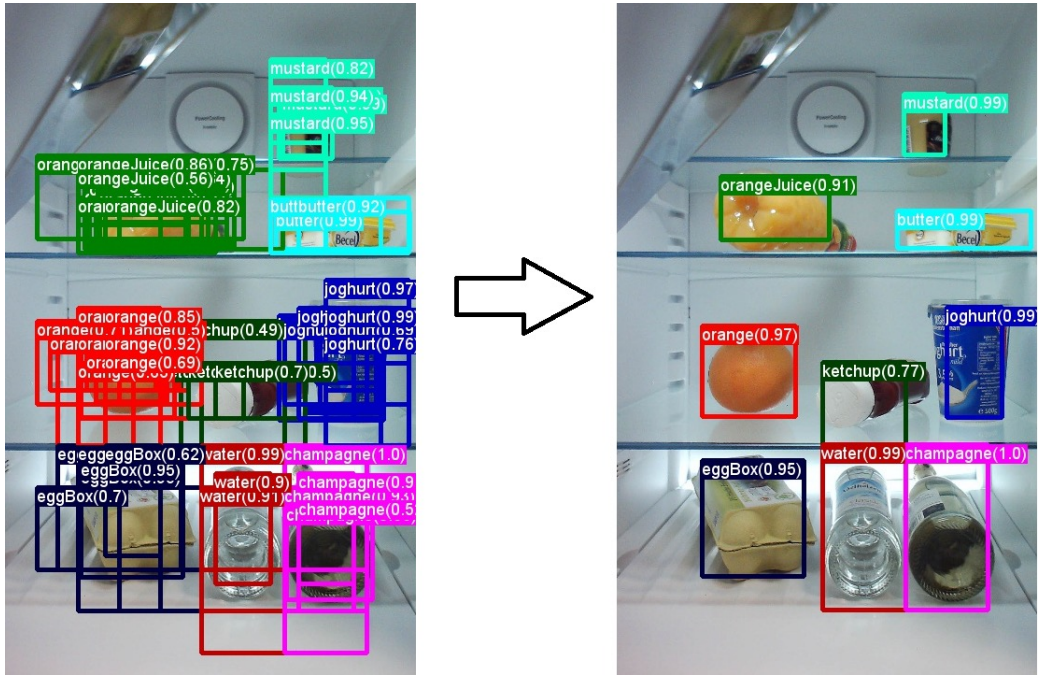
Output tensor



YOLOv1

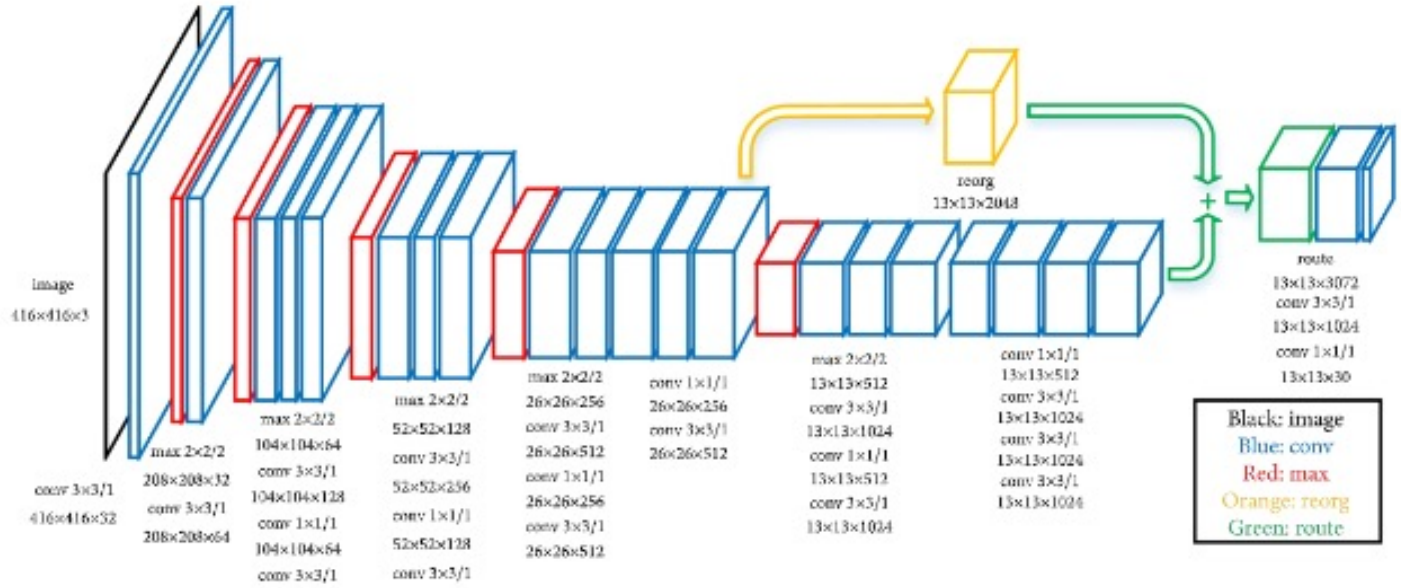
- Inference stage
  - NMS: 각 object에 대한 여러 bbox 중 가장 예측력이 좋은 bbox만 남김





YOLOv2

- YOLO9000: Better, Faster, Stronger
- Better
  - batch normalization (BN)
  - input size: 448x448로 통일
    - YOLOv1 ➔ training 224x224 / inference 448x448

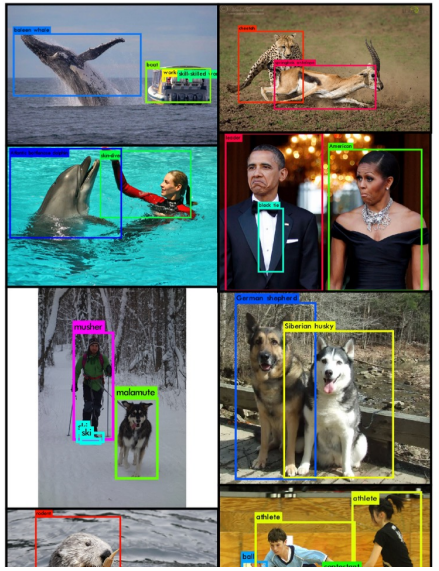


YOLO9000:
Better, Faster, Stronger

Joseph Redmon*†, Ali Farhadi*†
University of Washington*, Allen Institute for AI†
http://pjreddie.com/yolo9000/

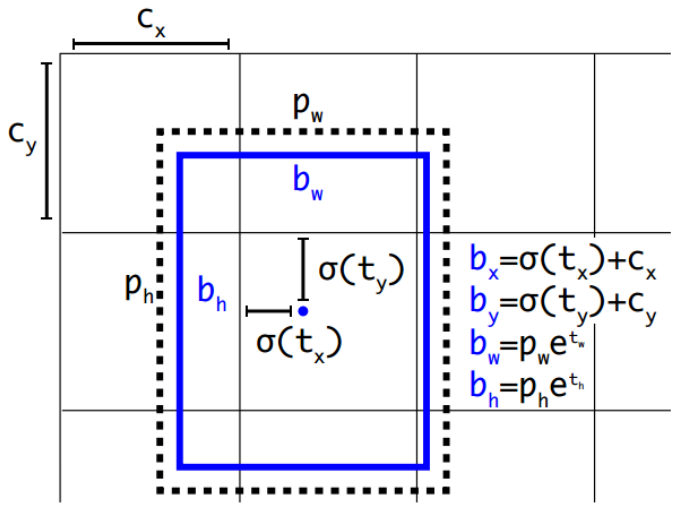
Abstract

We introduce YOLO9000, a state-of-the-art, real-time object detection system that can detect over 9000 object categories. First we propose various improvements to the YOLO detection method, both novel and drawn from prior work. The improved model, YOLOv2, is state-of-the-art on standard detection tasks like PASCAL *VOC and COCO. Using a novel, multi-scale training method the same YOLOv2 model can run at varying sizes, offering an easy tradeoff between speed and accuracy. At 67 FPS, YOLOv2 gets 76.8 mAP on VOC 2007. At 40 FPS, YOLOv2 gets 78.6 mAP, outperforming state-of-the-art methods like Faster R-CNN with ResNet and SSD while still running significantly faster. Finally we propose a method to jointly train on object detection and classification. Using this method we train YOLO9000 simultaneously on the COCO detection dataset and the ImageNet classification dataset. Our joint training allows YOLO9000 to predict detections for object classes that don't have labelled detection data. We validate our approach on the ImageNet detection task. YOLO9000 gets 19.7 mAP on the ImageNet detection validation set despite only having detection data for 44 of the 200 classes. On the 156 classes not in COCO, YOLO9000 gets 16.0 mAP. But YOLO can detect more than just 200 classes; it predicts detections for more than 9000 different object categories. And it still runs in real-time.*



YOLOv2

- YOLO9000: Better, Faster, Stronger
- Better
  - anchor box offset 추정
    - 각 anchor box에 대해서 4개의 offset 값을 예측
    - p_w, p_h: anchor box size
    - t_x, t_y, t_w, t_h: 모델의 예측 offset 값
    - b_x, b_y, b_w, b_h: 예측된 bbox
      - 중심위치 b_x, b_y: grid cell 내의 상대적 위치
        - sigmoid function을 통해 0~1사이 값으로 조정
      - 너비 b_w, 높이 b_h: anchor box 차원에 대한 스케일 조정값
        - 지수 함수를 통해 계산



YOLOv2

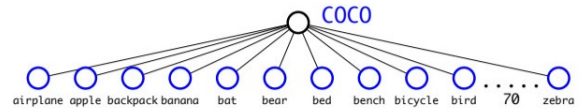
- Faster
  - Backbone: Darknet-19

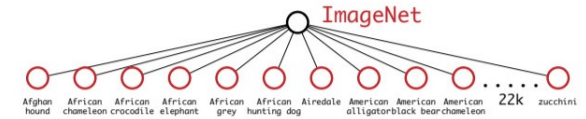
Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

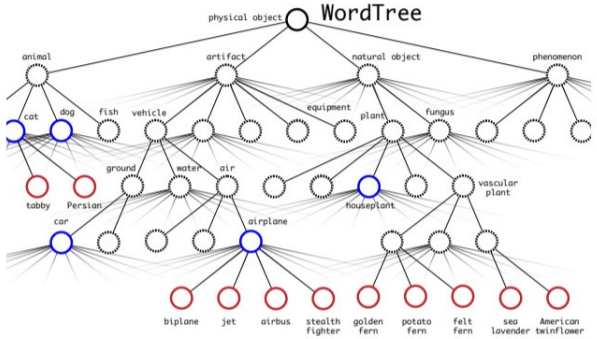
- Stronger
  - Hierarchical classification
    - ImageNet label: WordNet 구조에 따라 정의됨
      - WordNet은 graph 구조

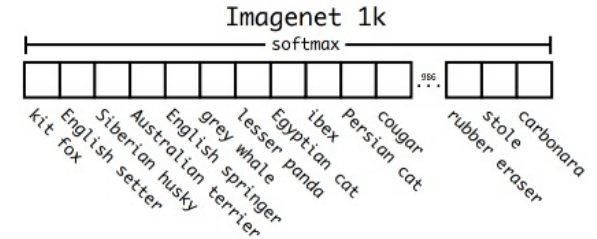
        ➔ 문제: 개 / 요크셔 테리어 가 다른 class로 분류
      - Tree 구조로 전환

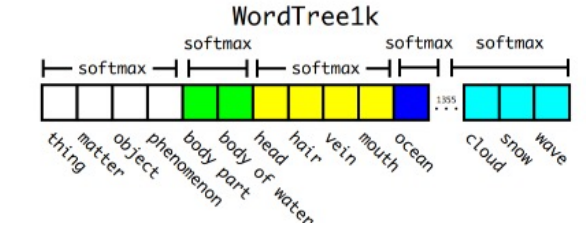
$$Pr(Yorkshireterrier) = Pr(Yorkshierterrier|terrier) * Pr(terrier|huntingdog) * \ldots * Pr(mammal|animal) * Pr(animal|physicalobject)$$





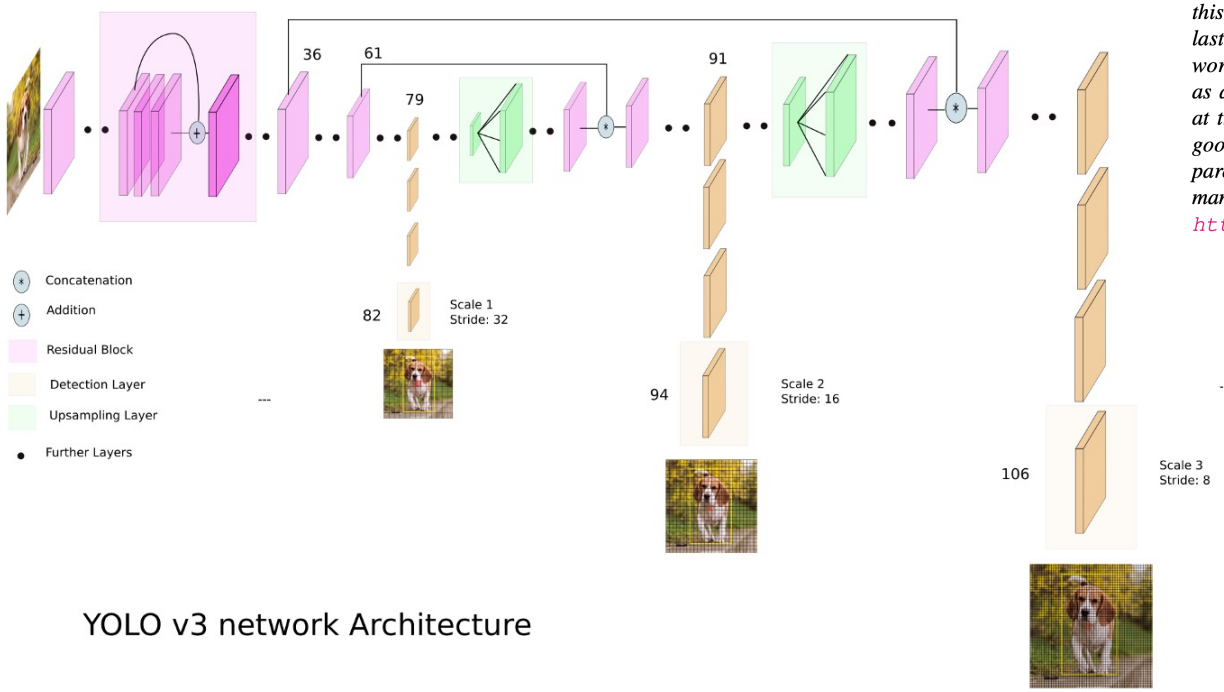






YOLOv3

- YOLOv3: An Incremental Improvement
- Network design



YOLO v3 network Architecture

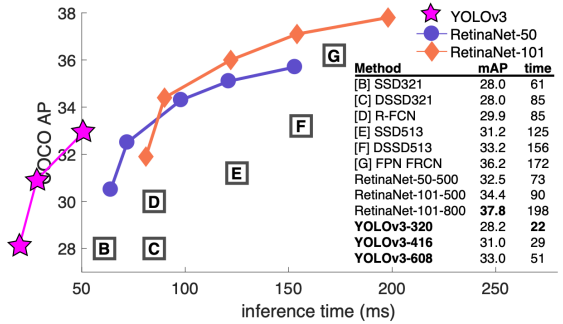
- Keyword: Darknet-53, Feature Pyramid Network (FPN)

YOLOv3: An Incremental Improvement

Joseph Redmon Ali Farhadi
University of Washington

Abstract

We present some updates to YOLO! We made a bunch of little design changes to make it better. We also trained this new network that's pretty swell. It's a little bigger than last time but more accurate. It's still fast though, don't worry. At 320 × 320 YOLOv3 runs in 22 ms at 28.2 mAP, as accurate as SSD but three times faster. When we look at the old .5 IOU mAP detection metric YOLOv3 is quite good. It achieves 57.9 AP_{50} in 51 ms on a Titan X, compared to 57.5 AP_{50} in 198 ms by RetinaNet, similar performance but 3.8× faster. As always, all the code is online at https://pjreddie.com/yolo/.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 1. We adapt this figure from the Focal Loss paper [9].

YOLOv3

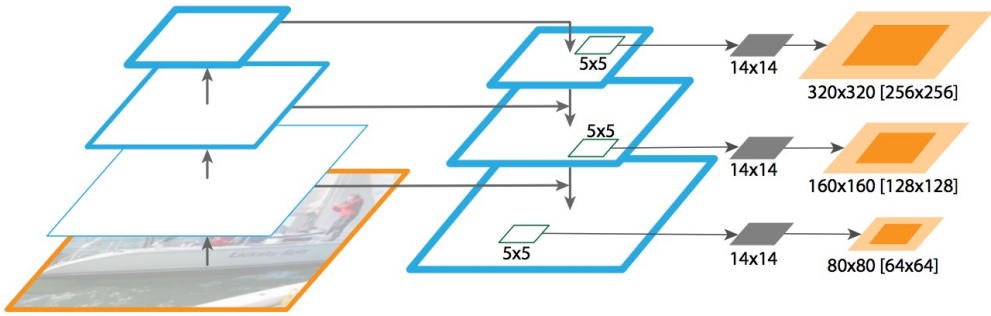
- Backbone: Darknet-53
  - ResNet-101보다 약 1.5배 빠름
  - Darknet에 residual 개념 추가

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Type	Filters	Size/Stride	Output
Convolutional	32	3 × 3	224 × 224
Maxpool		2 × 2/2	112 × 112
Convolutional	64	3 × 3	112 × 112
Maxpool		2 × 2/2	56 × 56
Convolutional	128	3 × 3	56 × 56
Convolutional	64	1 × 1	56 × 56
Convolutional	128	3 × 3	56 × 56
Maxpool		2 × 2/2	28 × 28
Convolutional	256	3 × 3	28 × 28
Convolutional	128	1 × 1	28 × 28
Convolutional	256	3 × 3	28 × 28
Maxpool		2 × 2/2	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Convolutional	256	1 × 1	14 × 14
Convolutional	512	3 × 3	14 × 14
Maxpool		2 × 2/2	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	512	1 × 1	7 × 7
Convolutional	1024	3 × 3	7 × 7
Convolutional	1000	1 × 1	7 × 7
Avgpool		Global	1000
Softmax			

Table 1. **Darknet-53.**

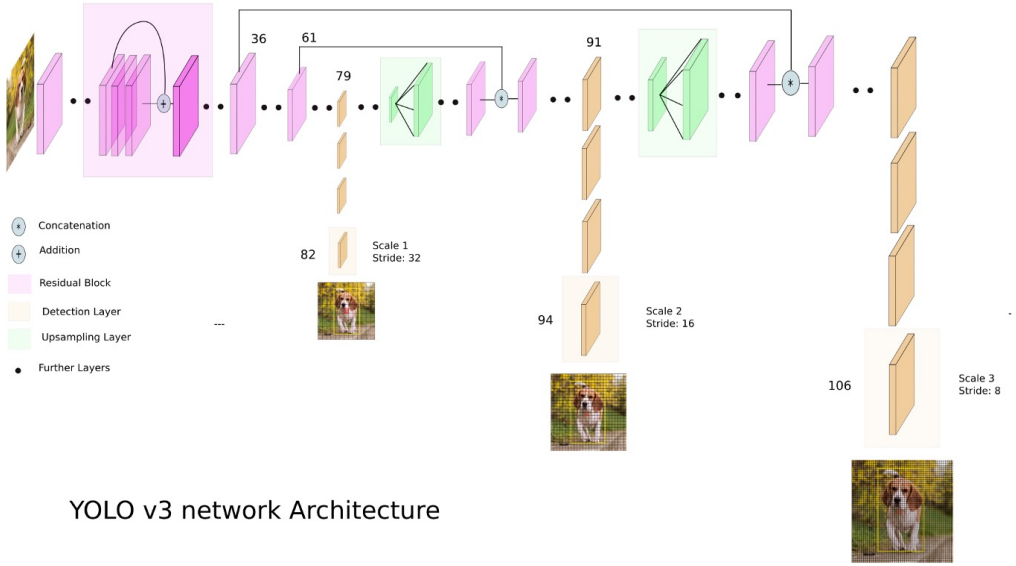
YOLOv3



- Feature Pyramid Network (FPN)
  - CNN-block을 통해 feature map 생성 후 down sampling
  - down sampling된 feature map과 up sampling된 feature map의 결합
  - ➔ 다양한 스케일의 feature map 구성 가능

- 멀티 스케일 출력
  - 네트워크의 다양한 깊이에서 세 가지 다른 크기의 특징 맵 출력
  - 3개의 스케일에 대한 결과 제공

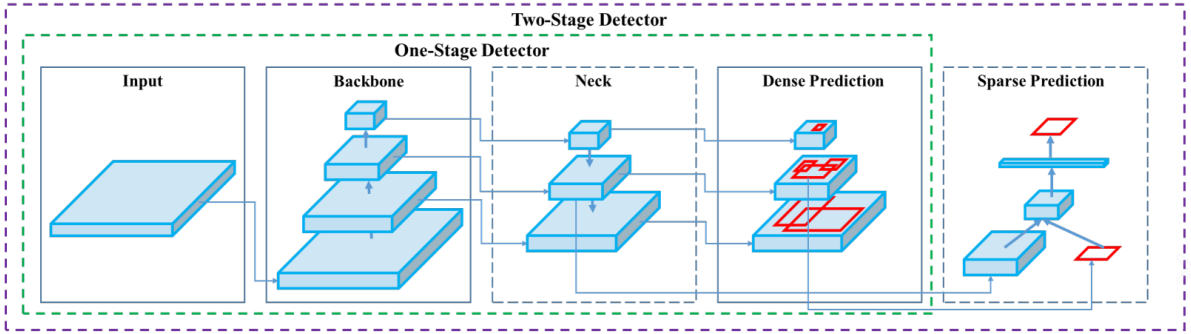
- 코드 설명: https://github.com/ultralytics/yolov3/tree/master



YOLO v3 network Architecture

(참고) Object detection model의 구성

- Backbone: 모델의 기본이 되는 CNN blocks
  - 입력 이미지로부터 고수준의 feature map 추출
  - 이미지의 복잡한 패턴과 구조 이해를 목적으로 함



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

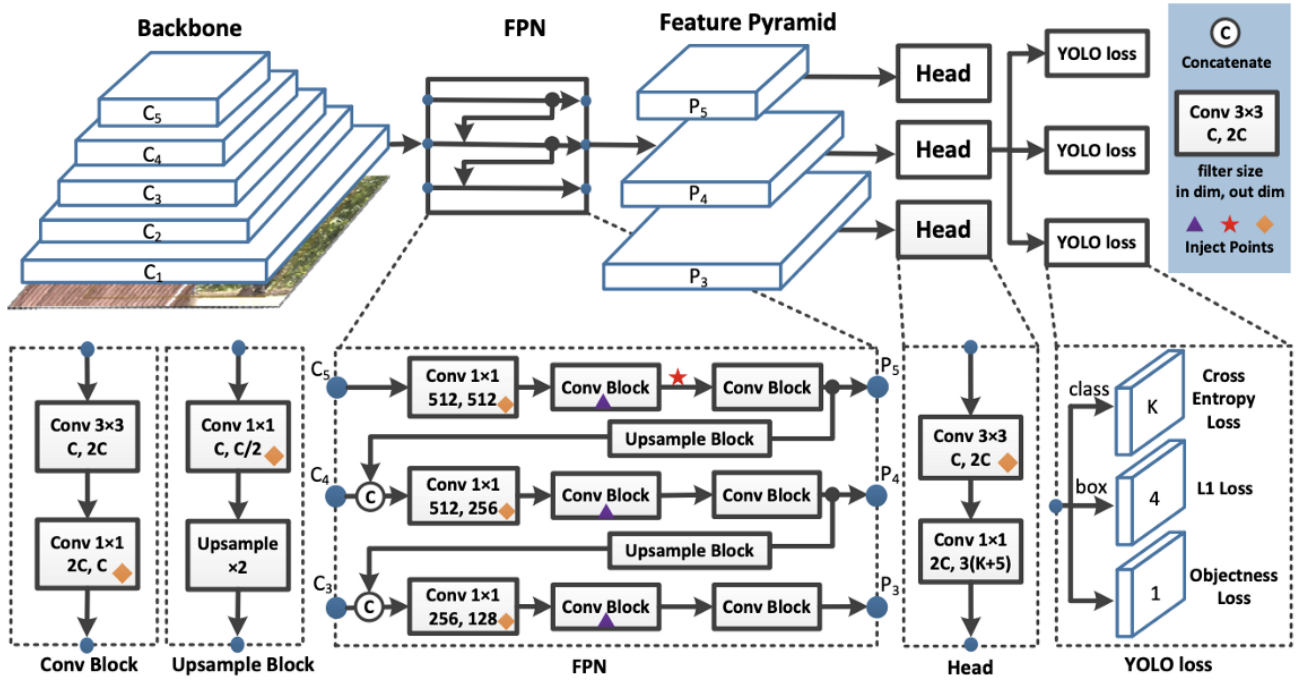
Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

- Neck: backbone에서 추출된 feature map을 추가로 가공
  - 객체 탐지를 위한 더욱 풍부한 특징 정보 생성
  - feature map의 다양한 스케일을 통합

- Head: 실제 객체 탐지를 수행하는 부분
  - fully connected layers, bbox regressor, classification 부분을 포함

YOLOv4

- YOLOv4: Optimal Speed and Accuracy of Object Detection
- Network design



- Keyword: Cross Stage Partial DenseNet (CSPNet), SPP, PAN

YOLOv4: Optimal Speed and Accuracy of Object Detection

Alexey Bochkovskiy*
alexeyab84@gmail.com

Chien-Yao Wang*
Institute of Information Science
Academia Sinica, Taiwan
kinyiu@iis.sinica.edu.tw

Hong-Yuan Mark Liao
Institute of Information Science
Academia Sinica, Taiwan
liao@iis.sinica.edu.tw

Abstract

There are a huge number of features which are said to improve Convolutional Neural Network (CNN) accuracy. Practical testing of combinations of such features on large datasets, and theoretical justification of the result, is required. Some features operate on certain models exclusively and for certain problems exclusively, or only for small-scale datasets; while some features, such as batch-normalization and residual-connections, are applicable to the majority of models, tasks, and datasets. We assume that such universal features include Weighted-Residual-Connections (WRC), Cross-Stage-Partial-connections (CSP), Cross mini-Batch Normalization (CmBN), Self-adversarial-training (SAT) and Mish-activation. We use new features: WRC, CSP, CmBN, SAT, Mish activation, Mosaic data augmentation, CmBN, DropBlock regularization, and CIoU loss, and combine some of them to achieve state-of-the-art results: 43.5% AP (65.7% AP_{50}) for the MS COCO dataset at a real-time speed of ~65 FPS on Tesla V100. Source code is at https://github.com/AlexeyAB/darknet.

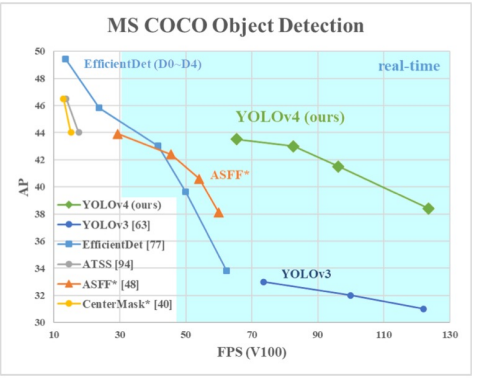


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.

YOLOv4

- Backbone
  - Cross Stage Partial DenseNet
  - CSPNet (CSPDarkNet-53)
    - DensNet 기반의 layer의 feature map을 분할한 후 Cross-Stage Hierarchy 방법을 통해 결합하여 연산량을 획기적으로 낮춘 모델

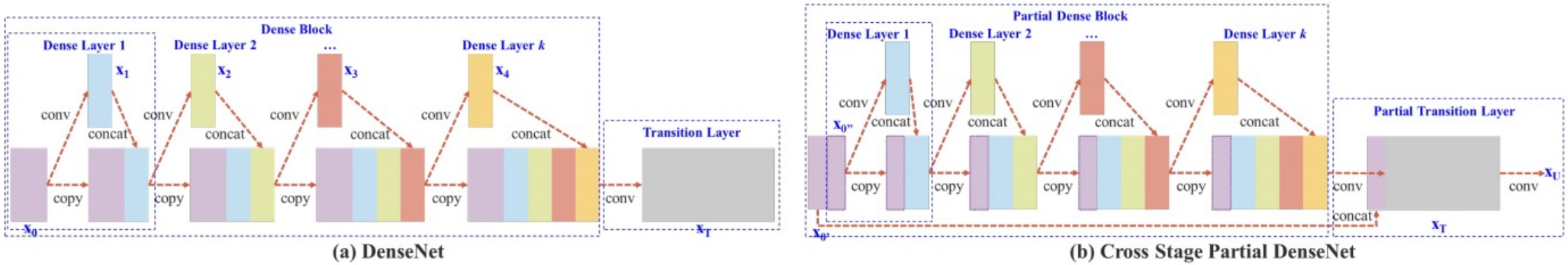
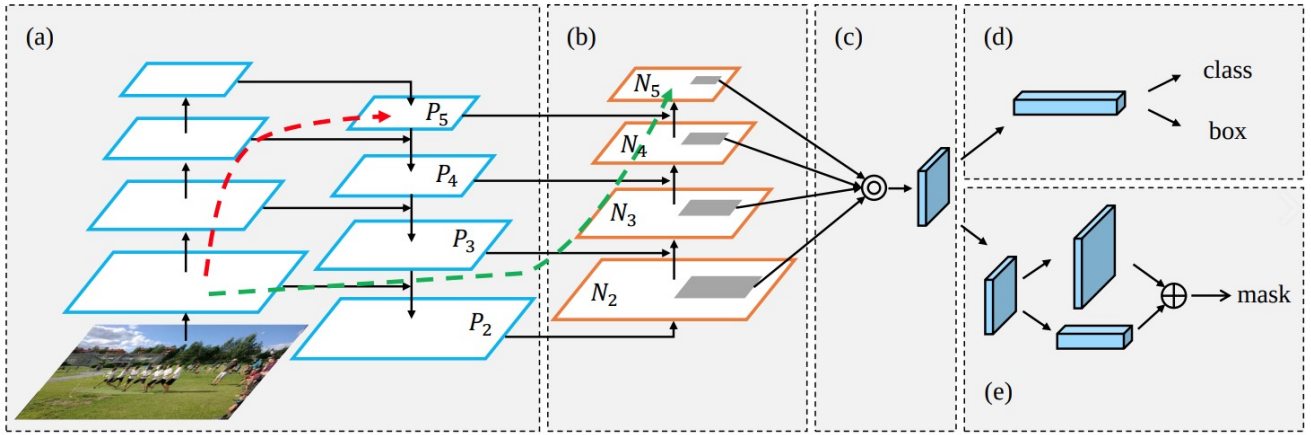


Figure 2: Illustrations of (a) DenseNet and (b) our proposed Cross Stage Partial DenseNet (CSPDenseNet). CSPNet separates feature map of the base layer into two part, one part will go through a dense block and a transition layer; the other one part is then combined with transmitted feature map to the next stage.

YOLOv4

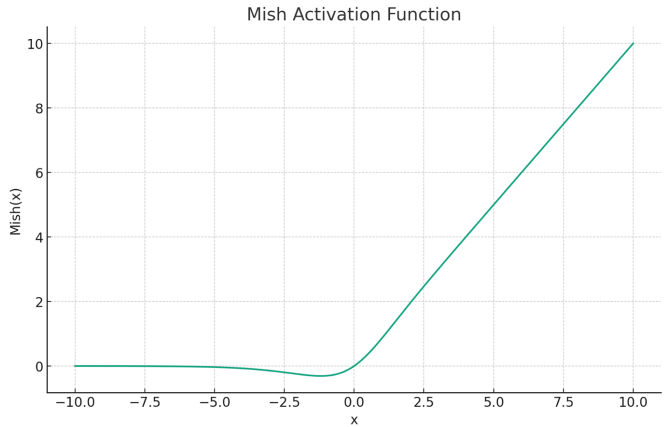
- Neck
  - SPP 구조 (Additional block)
    - feature 길이 고정 (Fast R-CNN 참조)
  - Path Aggregation Network (PAN)
    - FPN의 변형 (Yolo v3)
    - bottom-up path augmentation을 통해
      low-level feature의 정보를 high-level feature에 효과적으로 전달 ➔ localization 성능 향상



- Head
  - Yolo v3와 같음

YOLOv4

- **Bag-of-Specials (BoS):** 모델 성능 개선을 위해 layer에 추가적으로 다양한 layer/network의 추가/조작을 통해 성능을 개선하는 방법
  - CSPNet
  - Mish Activation Function



- **Bag-of-Freebies (BoF):** 모델의 구조나 hyperparameter에 대한 변경 없이 모델의 성능을 개선하는 방법
  - Augmentation, Regularization, Optimization
  - Mosaic / MixUp data augmentation
  - DropBlock Regularization

DropBlock: A regularization method for convolutional networks

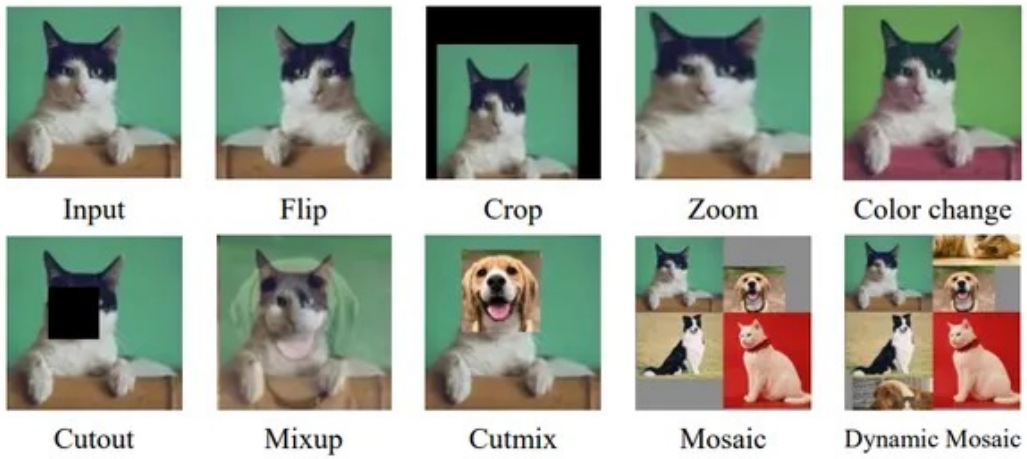
Golnaz Ghiasi
Google Brain

Tsung-Yi Lin
Google Brain

Quoc V. Le
Google Brain

Abstract

Deep neural networks often work well when they are over-parameterized and trained with a massive amount of noise and regularization, such as weight decay and dropout. Although dropout is widely used as a regularization technique for fully connected layers, it is often less effective for convolutional layers. This lack of success of dropout for convolutional layers is perhaps due to the fact that activation units in convolutional layers are spatially correlated so information can still flow through convolutional networks despite dropout. Thus a structured form of dropout is needed to regularize convolutional networks. In this paper, we introduce DropBlock, a form of structured dropout, where units in a contiguous region of a feature map are dropped together. We found that applying DropbBlock in skip connections in addition to the convolution layers increases the accuracy. Also, gradually increasing number of dropped units during training leads to better accuracy and more robust to hyperparameter choices. Extensive experiments show that DropBlock works better than dropout in regularizing convolutional networks. On ImageNet classification, ResNet-50 architecture with DropBlock achieves 78.13% accuracy, which is more than 1.6% improvement on the baseline. On COCO detection, DropBlock improves Average Precision of RetinaNet from 36.8% to 38.4%.



YOLOv7

- YOLOv5
  - 코드 설명: https://github.com/ultralytics/yolov5
  - ultralytics: yolov8 개발팀
- YOLOv7
  - YOLOv5 코드를 기반으로 만들어짐

YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors

Chien-Yao Wang[1], Alexey Bochkovskiy, and Hong-Yuan Mark Liao[1]
[1]Institute of Information Science, Academia Sinica, Taiwan
kinyiu@iis.sinica.edu.tw, alexeyab84@gmail.com, and liao@iis.sinica.edu.tw

Abstract

YOLOv7 surpasses all known object detectors in both speed and accuracy in the range from 5 FPS to 160 FPS and has the highest accuracy 56.8% AP among all known real-time object detectors with 30 FPS or higher on GPU V100. YOLOv7-E6 object detector (56 FPS V100, 55.9% AP) outperforms both transformer-based detector SWIN-L Cascade-Mask R-CNN (9.2 FPS A100, 53.9% AP) by 509% in speed and 2% in accuracy, and convolutional-based detector ConvNeXt-XL Cascade-Mask R-CNN (8.6 FPS A100, 55.2% AP) by 551% in speed and 0.7% AP in accuracy, as well as YOLOv7 outperforms: YOLOR, YOLOX, Scaled-YOLOv4, YOLOv5, DETR, Deformable DETR, DINO-5scale-R50, ViT-Adapter-B and many other object detectors in speed and accuracy. Moreover, we train YOLOv7 only on MS COCO dataset from scratch without using any other datasets or pre-trained weights. Source code is released in https://github.com/WongKinYiu/yolov7.

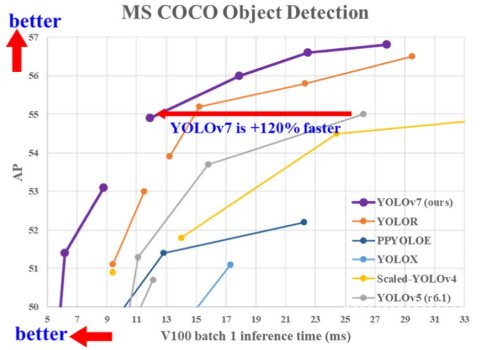
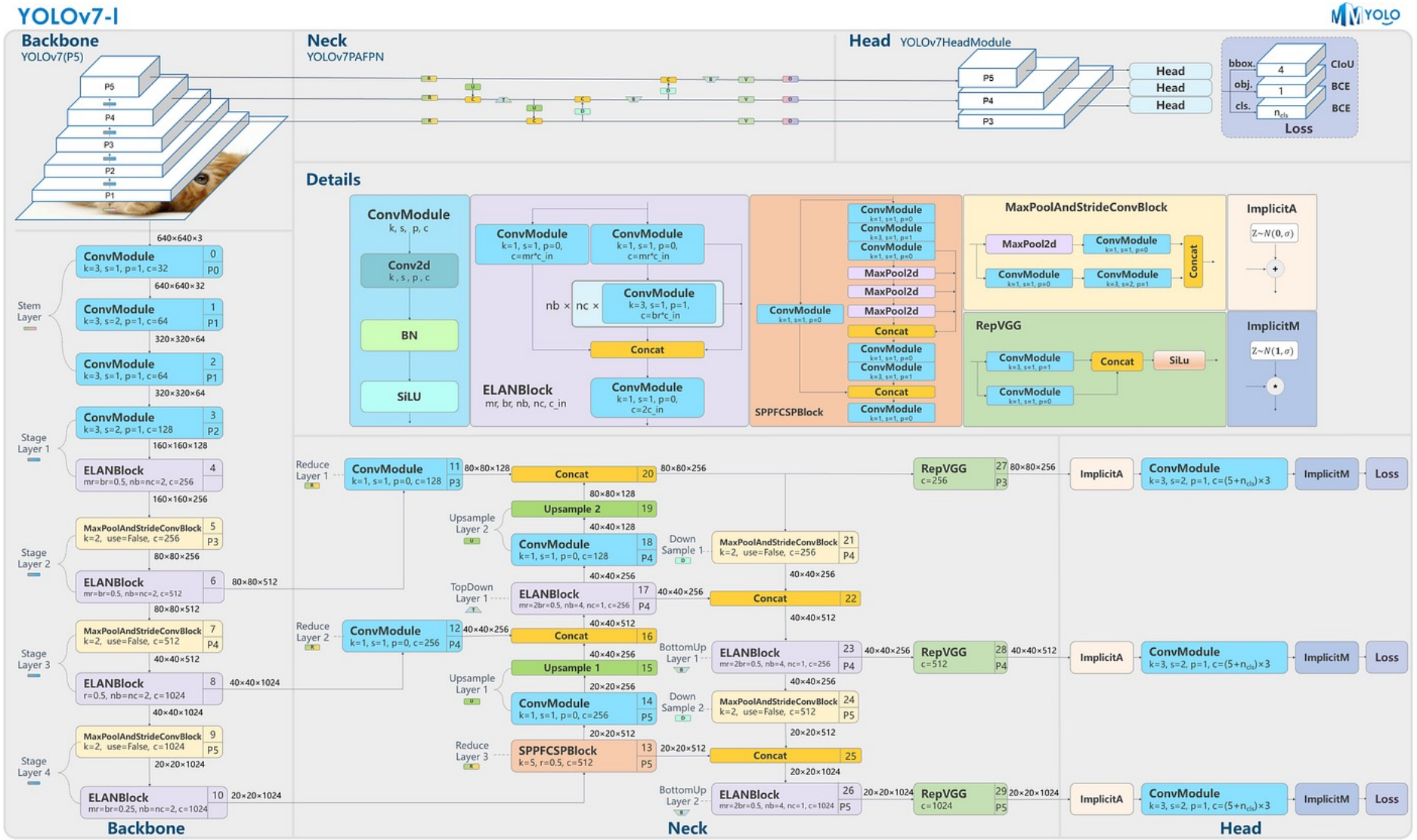


Figure 1: Comparison with other real-time object detectors, our proposed methods achieve state-of-the-arts performance.

YOLOv7

- Network design



YOLOv7

- Keyword
  - Extended efficient layer aggregation networks (E-ELAN)
  - Model scaling for concatenation-based model
  - Planned re-parameterized convolution (RepConvN)
  - Coarse for auxiliary and fine for lead loss

YOLOv7

- Extended efficient layer aggregation networks (E-ELAN)
  - network의 효율성과 효과를 극대화하기 위하여 model architecture 개선
    - 파라미터 수, 계산량, 계산 밀도 등을 종합적으로 고려

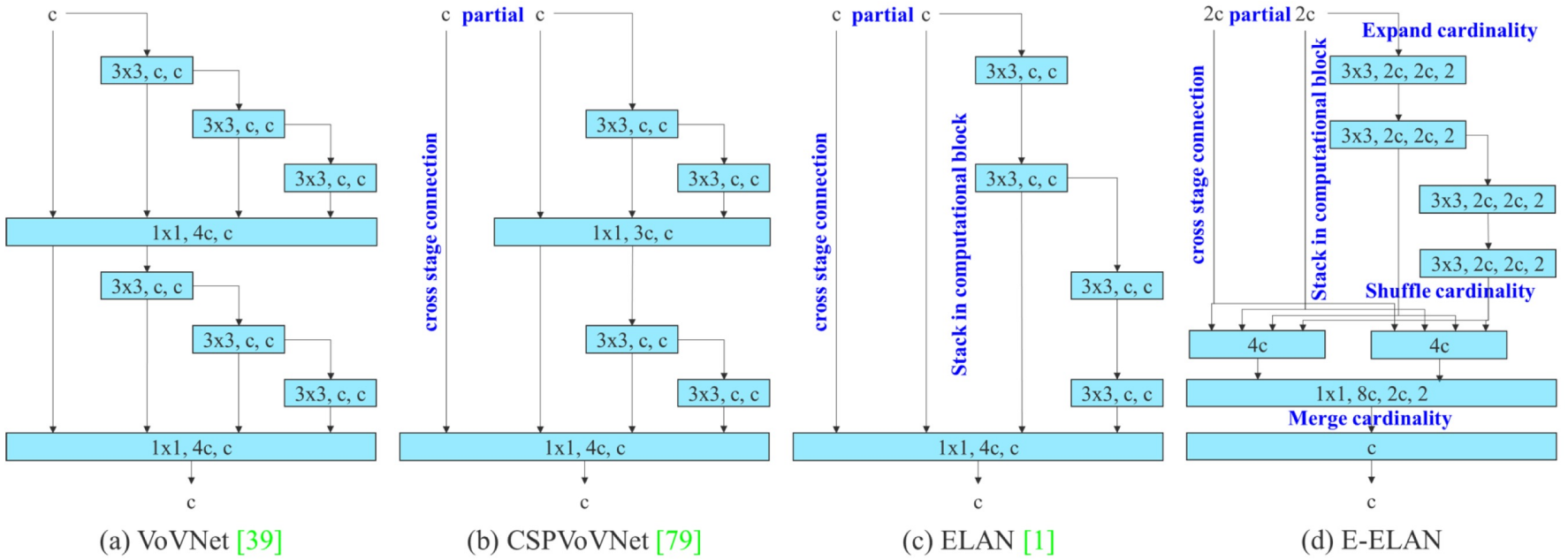
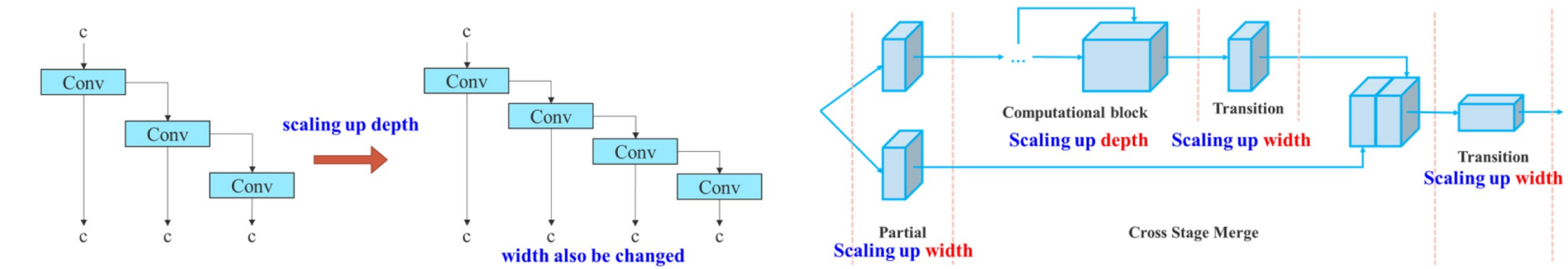


Figure 2: Extended efficient layer aggregation networks. The proposed extended ELAN (E-ELAN) does not change the gradient transmission path of the original architecture at all, but use group convolution to increase the cardinality of the added features, and combine the features of different groups in a shuffle and merge cardinality manner. This way of operation can enhance the features learned by different feature maps and improve the use of parameters and calculations.

YOLOv7

- Model scaling for concatenation-based model
  - 모델 스케일링의 목적: 모델의 일부 속성을 조정하고 inference 속도 향상
    - EfficientNet ➔ 모델의 너비, 깊이, 해상도 등을 고려한 스케일링 수행
  - 문제점: scaling 시 출력 크기가 바뀜 ➔ 출력 부분의 scale을 유지하면서 풍부한 특징 학습 전략
    - ex) depth 증가 시 width 크기 변화



(a) concatenation-based model (b) scaled-up concatenation-based model (c) compound scaling up depth and width for concatenation-based model

Figure 3: Model scaling for concatenation-based models. From (a) to (b), we observe that when depth scaling is performed on concatenation-based models, the output width of a computational block also increases. This phenomenon will cause the input width of the subsequent transmission layer to increase. Therefore, we propose (c), that is, when performing model scaling on concatenation-based models, only the depth in a computational block needs to be scaled, and the remaining of transmission layer is performed with corresponding width scaling.

YOLOv7

- Planned re-parameterized convolution (RepConvN): 재매개변수화 계획
  - inference cost를 늘리지 않고 정확도를 향상시키는 방법

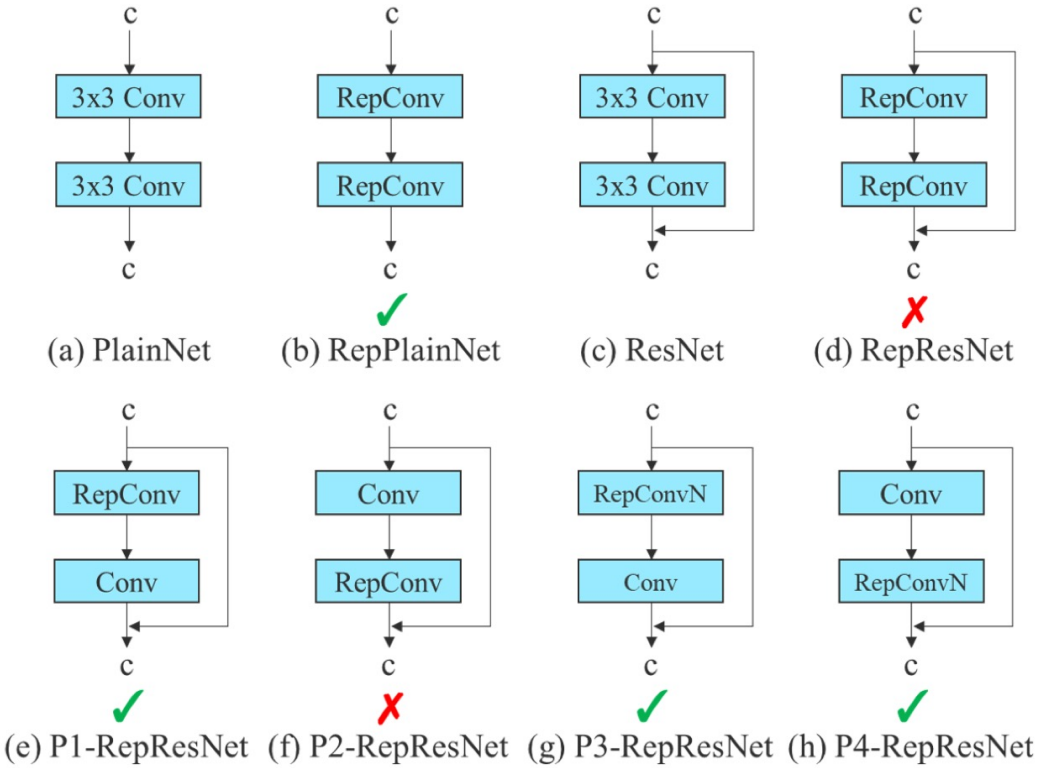


Figure 4: Planned re-parameterized model. In the proposed planned re-parameterized model, we found that a layer with residual or concatenation connections, its RepConv should not have identity connection. Under these circumstances, it can be replaced by RepConvN that contains no identity connections.

YOLOv7

- Coarse for auxiliary and fine for lead loss
  - 다양한 해상도의 손실함수를 사용하여 모델을 학습시키는 방법
  - “coarse for auxiliary”
    - 보조적은 목적을 위해 낮은 해상도의 데이터를 학습하여 학습을 가속화
  - “fine for lead loss”
    - 고해상도의 데이터를 사용하여 정밀한 객체 탐지 능력을 향상 시키는 전략

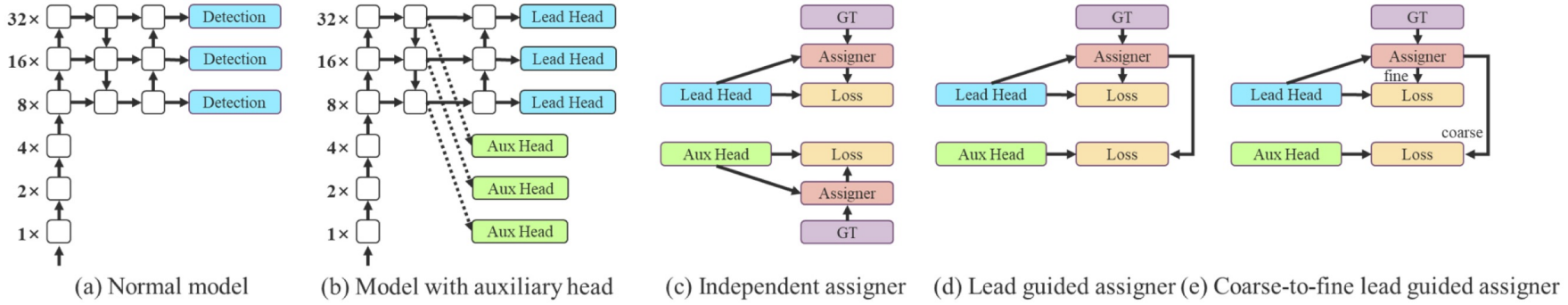


Figure 5: Coarse for auxiliary and fine for lead head label assigner. Compare with normal model (a), the schema in (b) has auxiliary head. Different from the usual independent label assigner (c), we propose (d) lead head guided label assigner and (e) coarse-to-fine lead head guided label assigner. The proposed label assigner is optimized by lead head prediction and the ground truth to get the labels of training lead head and auxiliary head at the same time. The detailed coarse-to-fine implementation method and constraint design details will be elaborated in Apendix.

YOLOv7

- **코드 설명 및 실습:** https://github.com/WongKinYiu/yolov7

End of slide