

KOSA-Python과 Pytorch를 활용한 인공지능 딥러닝 입문과정

2021. 10. 18 ~ 21

정 준 수 Ph.D

과정 목표

다양한 딥러닝 Framework의 이해와 심층신경망(DNN)과 합성곱신경망(CNN) 구현 등 딥러닝의 원리를 이해하고 Pytorch의 다양한 라이브러리를 인공지능 모델 개발에 적용

1. 딥러닝 도구 및 모델의 종류 이해
2. Neural Network 작동 원리 및 딥러닝 모델 개요
3. Tensorflow, Pytorch framework을 이용한 Model Training pipeline
4. Vision tasks – Image Classification, Localization & Object detection
5. DNN, CNN, RNN, LSTM, Transformer 등을 이용한 다양한 응용 사례 실습

Deep learning is not like pure mathematics. It is a heavily experimental field, so it's important to be a strong practitioner, not just a theoretician.

딥 러닝은 순수한 수학과는 다릅니다. 매우 실험적인 분야이기 때문에 이론가가 아닌 실무자가 되는 것이 중요합니다.

<교안 및 실습 예제 프로그램>

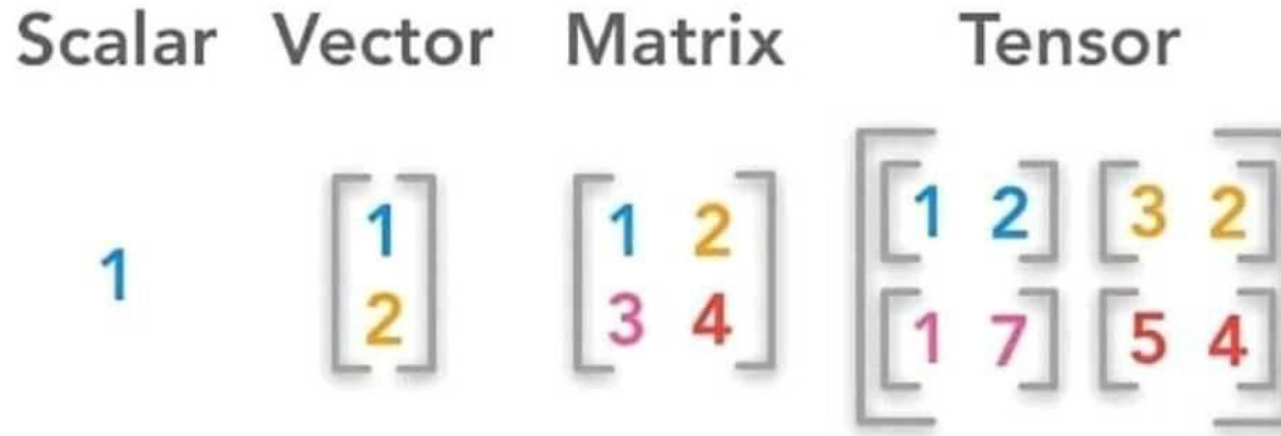
<https://github.com/JSJeong-me/KOSA-Pytorch>

<https://github.com/JSJeong-me/fastbook>

<https://github.com/deep-learning-with-pytorch/dlwpt-code>

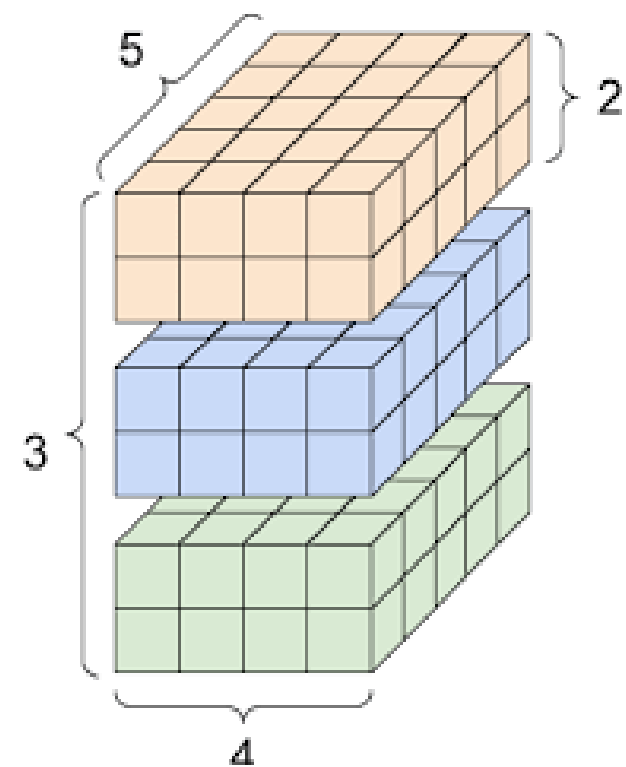
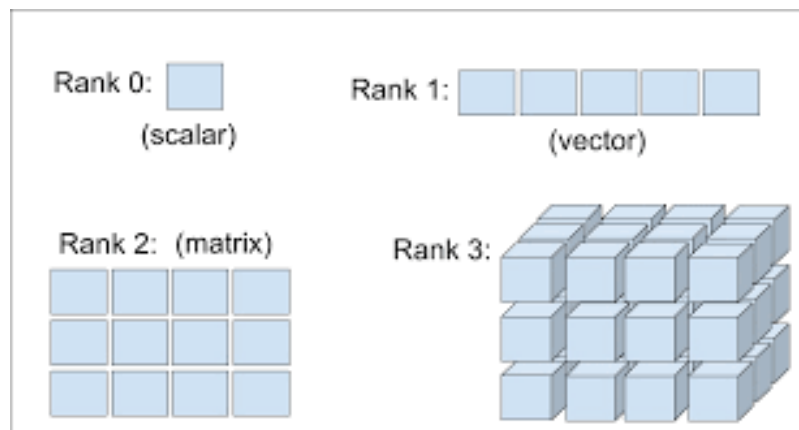
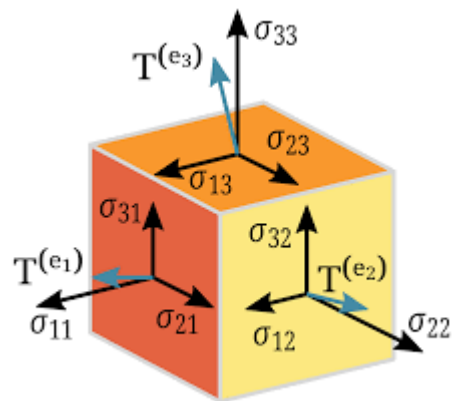
What is a Tensor?

In deep learning it is common to see a lot of discussion around tensors as the cornerstone data structure. Tensor even appears in name of Google's machine learning library: "TensorFlow". Tensors are a type of data structure used in linear algebra, and like vectors and matrices, you can calculate operations with tensors.

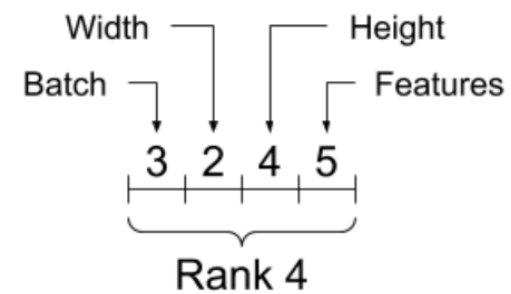


A tensor is a generalization of vectors and matrices and is easily understood as a multidimensional array. A vector is a one dimensional or first order tensor and a matrix is a two dimensional or second order tensor.

Tensor 축



일반적인 축 순서



딥러닝 Framework의 비교: TensorFlow vs Pytorch

TensorFlow:

<https://www.tensorflow.org/tutorials?hl=ko>

Pytorch:

<https://pytorch.org/tutorials/beginner/basics/intro.html>

Practical Deep Learning for Coders:

<https://course.fast.ai/>

TORCH.AUTOGRAD

`torch.autograd` is PyTorch's automatic differentiation engine that powers neural network training. In this section, you will get a conceptual understanding of how autograd helps a neural network train.

Training a NN happens in two steps:

Forward Propagation: In forward prop, the NN makes its best guess about the correct output. It runs the input data through each of its functions to make this guess.

Backward Propagation: In backprop, the NN adjusts its parameters proportionate to the error in its guess. It does this by traversing backwards from the output, collecting the derivatives of the error with respect to the parameters of the functions (gradients), and optimizing the parameters using gradient descent.



VS



```
In [4]: import numpy as np
from datetime import datetime
start = datetime.now()

np.random.seed(0)

N,D = 3,4

x = np.random.randn(N,D)
y = np.random.randn(N,D)
z = np.random.randn(N,D)

a = x * y
b = a * z
c = np.sum(b)
```

```
grad_c = 1.0
grad_b = grad_c * np.ones((N,D))
grad_a = grad_b.copy()
grad_z = grad_b.copy()
grad_y = grad_a * y
grad_x = grad_a * x
```

```
print(grad_x)
print(grad_y)
print(grad_z)
print(datetime.now()-start)
```

```
[[ 1.76405235  0.40015721  0.97873798  2.2408932 ]
 [ 1.86755799 -0.97727788  0.9608842  -0.15135721]
 [-0.10321885  0.4105985  0.14404357  1.45427351]]
[[ 0.76103773  0.12167502  0.44386323  0.33367433]
 [ 1.49407907 -0.20515826  0.3130677  -0.85409574]
 [-2.55298982  0.6536186  0.8644362  -0.74216502]]
[[ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]
 [ 1.  1.  1.  1.]]
0:00:00.003751
```

Gradient Calculation

```
In [6]: import torch
from torch.autograd import Variable
from datetime import datetime
start = datetime.now()

N,D = 3,4

x = Variable(torch.randn(N,D).cuda(), requires_grad=True)
y = Variable(torch.randn(N,D).cuda(), requires_grad=True)
z = Variable(torch.randn(N,D).cuda(), requires_grad=True)

a = x * y
b = a * z
c = torch.sum(b)

c.backward(torch.cuda.FloatTensor([1.0]))

print(x.grad)
print(y.grad)
print(z.grad)
print(datetime.now()-start)
```

```
Variable containing:
-0.6048  0.6640 -1.8035  1.0894
-0.0731 -0.0702 -0.0474 -1.7546
-0.3247  0.6293  2.5135 -0.5967
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]
```

```
Variable containing:
0.1152  1.1809  1.4522  1.9417
1.0845  0.1587 -1.6526  0.4031
1.9585 -0.4729 -1.4024 -0.7388
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]
```

```
Variable containing:
1  1  1  1
1  1  1  1
1  1  1  1
[torch.cuda.FloatTensor of size 3x4 (GPU 0)]
```

```
0:00:00.003434
```

Differentiation in Autograd

We create another tensor Q from a and b.

$$Q = 3a^3 - b^2$$

$$Q=3*a^{**3} - b^{**2}$$

Let's assume a and b to be parameters of an NN, and Q to be the error. In NN training, we want gradients of the error w.r.t. parameters, i.e.

$$\frac{\partial Q}{\partial a} = 9a^2$$

$$\frac{\partial Q}{\partial b} = -2b$$

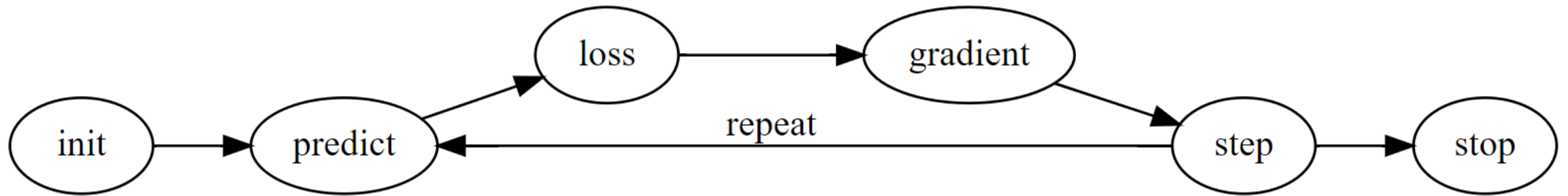
https://pytorch.org/tutorials/beginner/blitz/autograd_tutorial.html

Fast ai

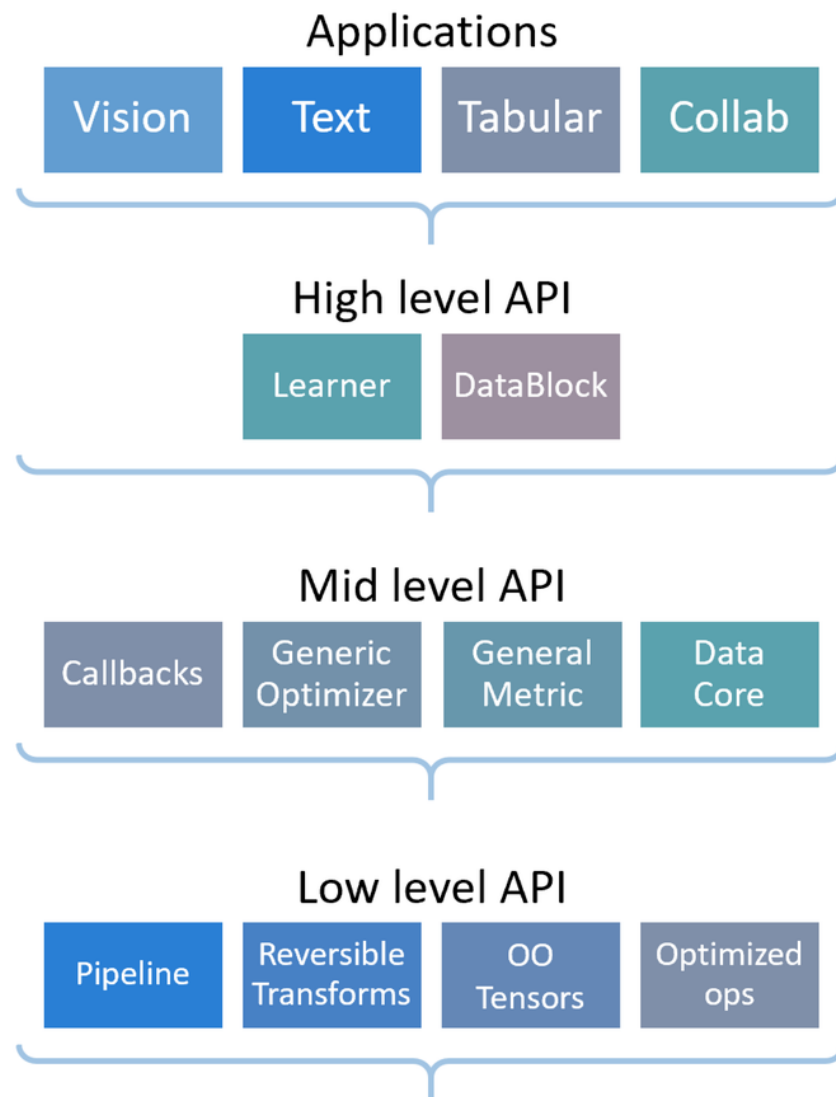
Everybody should be able to use deep learning to solve their problems with no more education than it takes to use a smart phone. Therefore, each year our main research goal is to be able to teach a wider range of deep learning applications, that run faster, and are more accurate, to people with less prerequisites.

누구나 스마트폰을 사용하는 것보다 더 많은 교육 없이도 딥 러닝을 사용하여 문제를 해결할 수 있어야합니다. 따라서 매년 우리의 주요 연구 목표는 전제 조건이 적은 사람들에게 더 빠르게 실행되고 더 정확한 더 넓은 범위의 딥 러닝 애플리케이션을 가르칠 수 있도록하는 것입니다.

Fast ai – Forward & Backward



Fast ai API의 구성



인공지능 기술의 각 기관별 분류와 기술 체계 – Computer Vision

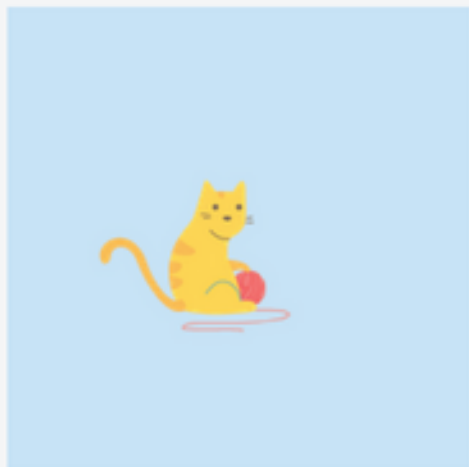
BCC	NIA - DNA 플러스 2020	NIA	인공지능협회	Aura	Deloitte	IntelliPaat	G2 Crowd	BoFAML	특허청	Waltz, D.	Tracica	정보통신 기술진흥원
전문가 시스템	컴퓨터 비전	사각지능	사각지능	사각지능	이미지 인식	컴퓨터 비전	사각지능	사각지능	사각 인식	전문가 시스템	이미지 인식	사각 이해
지능형 가상 비서	인간 언어 기술	언어/정각지능	언어 지능	언어 처리	자연어 처리	자연어 처리	자연어 처리	자연어 처리	언어 이해	자율로봇	자연어 처리	언어 이해
뉴얼 컴퓨팅	머신 러닝	음성지능	음성 지능	음성 지능	음성 인식	뉴얼 네트워크	음성지능	음성지능	학습과 추론	인지보조	음성 인식	상황 이해
지능형 임베디드 시스템	지식 표현	융합지능	시계열 데이터 처리/모델링	기계학습	기계학습	기계학습	기계학습	기계학습	상황 인식	사이버/알고리즘	응용프로그램	인지 인식 및 인지
자율화 로봇	자율 계획 및 스케줄링		종합 데이터 가공	전문가 시스템	입력/출력	입력/출력	전문가 시스템	규칙 기반 시스템	응용 분야	시스템 통합	인터페이스	학습 및 추론
	전문가 시스템		하드웨어/로봇틱스	로봇공학	전문가 시스템	인지 컴퓨팅	계획 및 최적화	계획, 스케줄링, 최적화			입력/출력	
	로봇공학		감각학습				로봇공학	로봇공학			인지 컴퓨팅	
			자율주행								기계 학습	
			일반지능									
			감성지능									
			실용 가능한 지능									

시각지능
 언어지능
 음성지능
 로봇공학
 전문가 시스템

*BoFAML: BofA Merrill Lynch Global Research

Computer Vision Problem Types

Classification



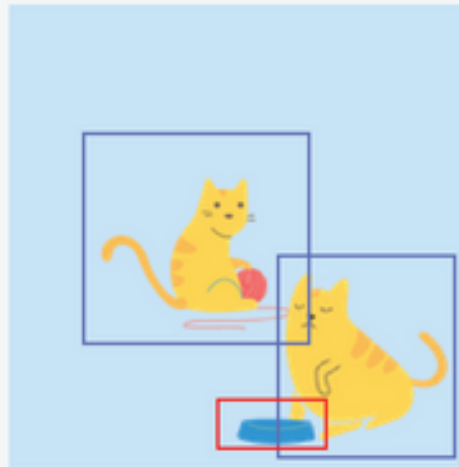
CAT

Classification
+ Localization



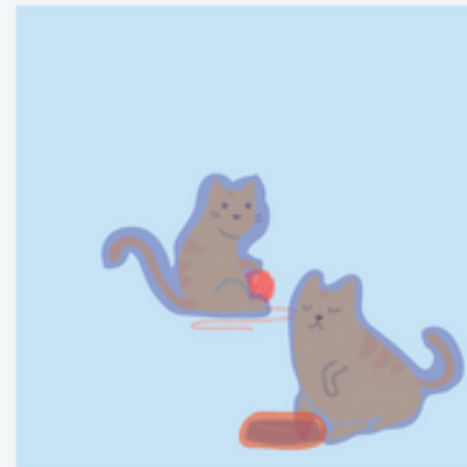
CAT

Object Detection



CAT, CAT, BOWL

Semantic
Segmentation



CAT, CAT, BOWL

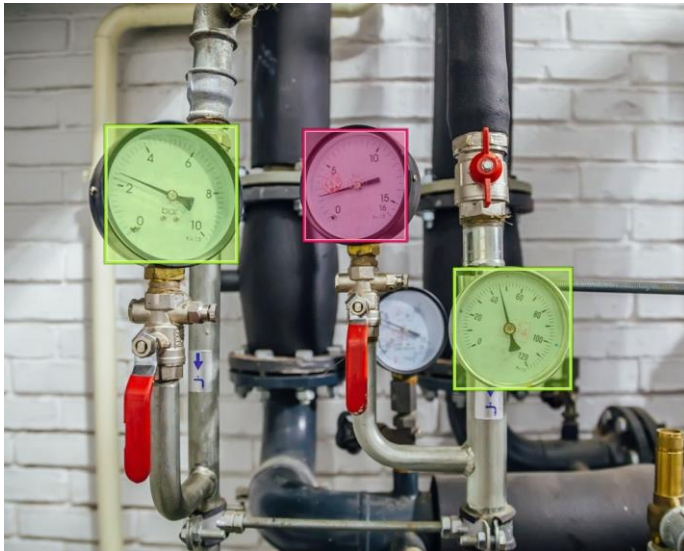
Single Object

Multiple Objects

roboflow

제조 현장에서의 컴퓨터 비전 적용 사례

Predictive maintenance



Defect detection



Assembling products and components

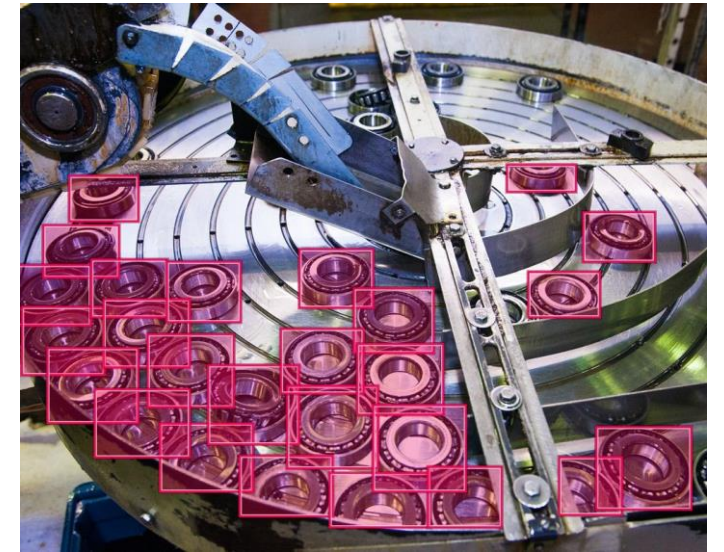
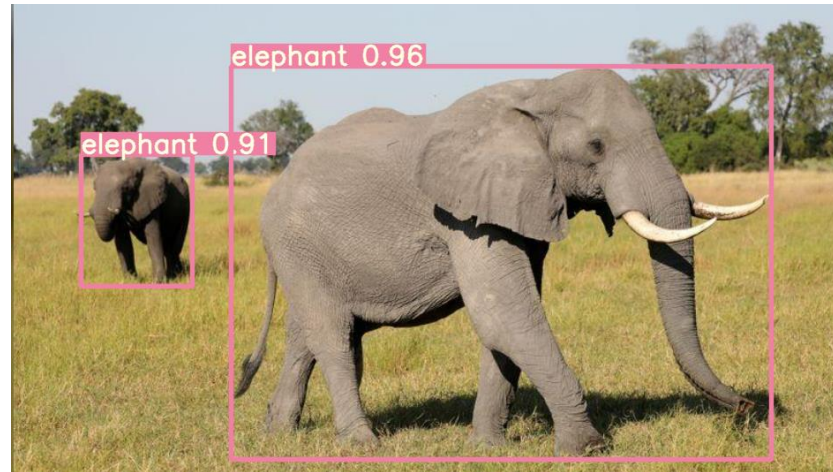


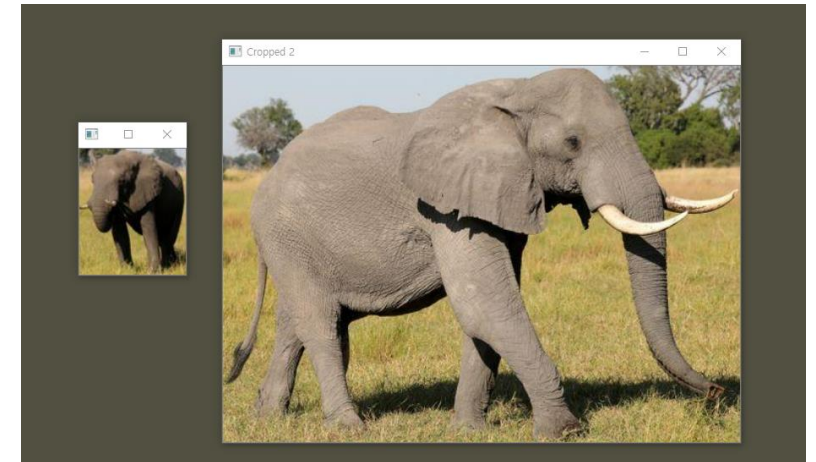
Image Classification & Object Detection



Source Image

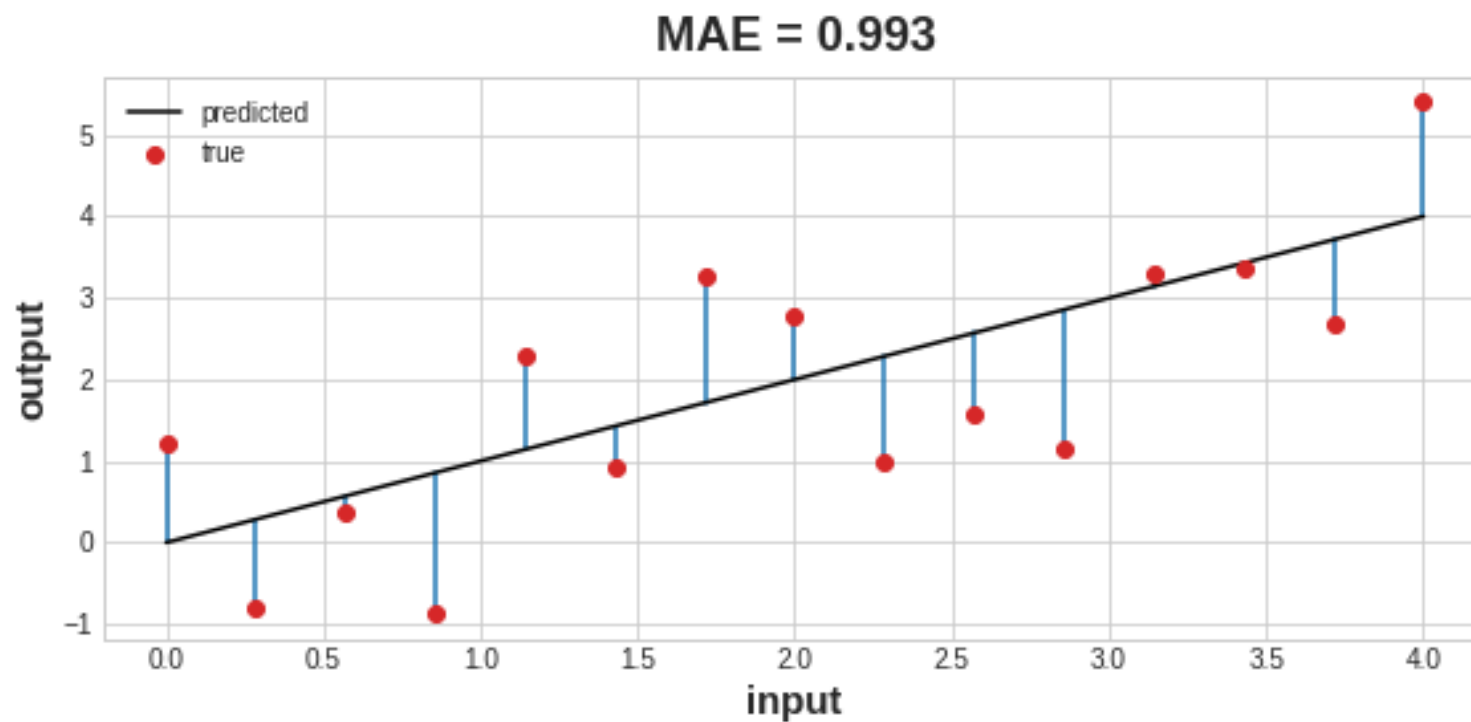


Object Detection

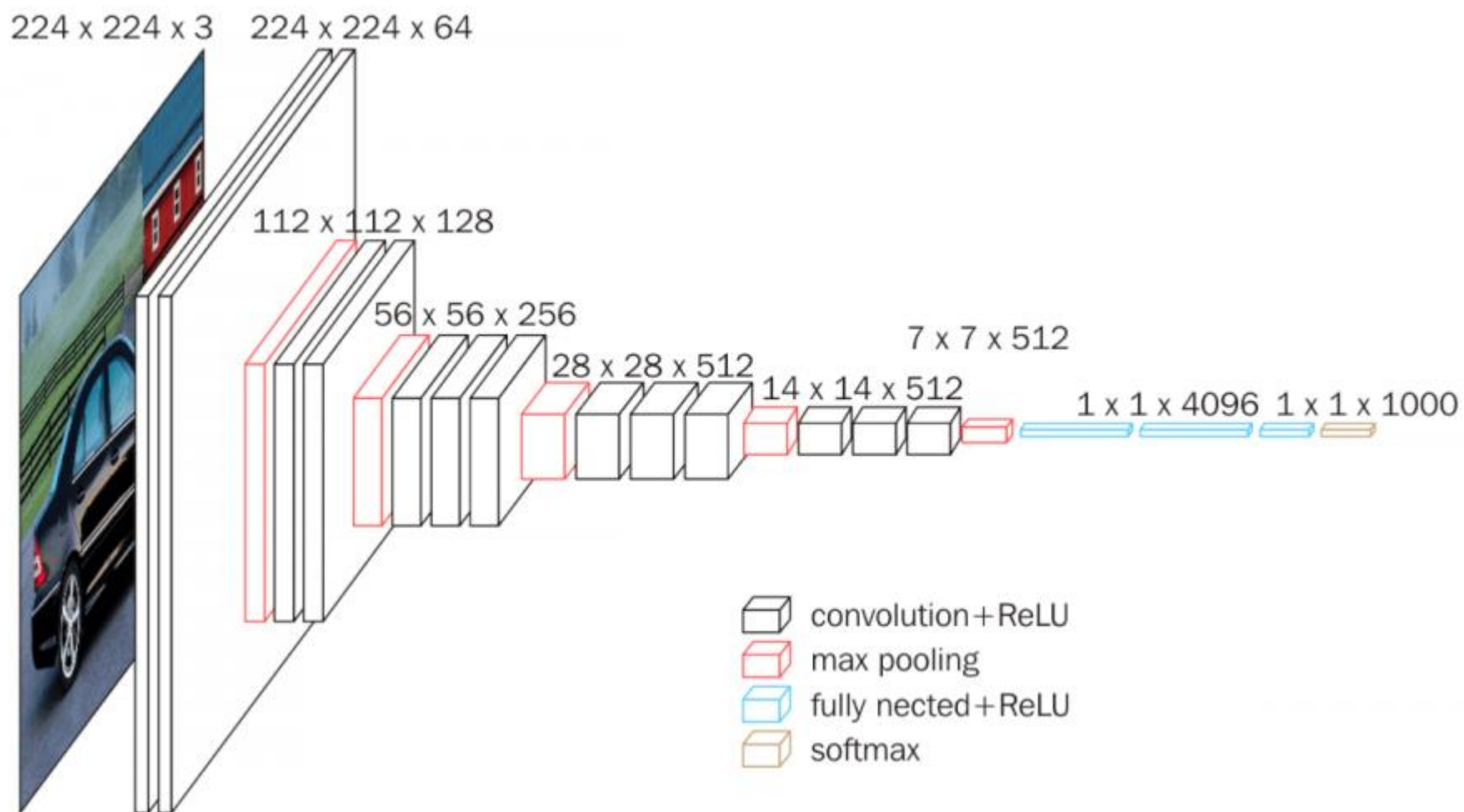


Cropped Image

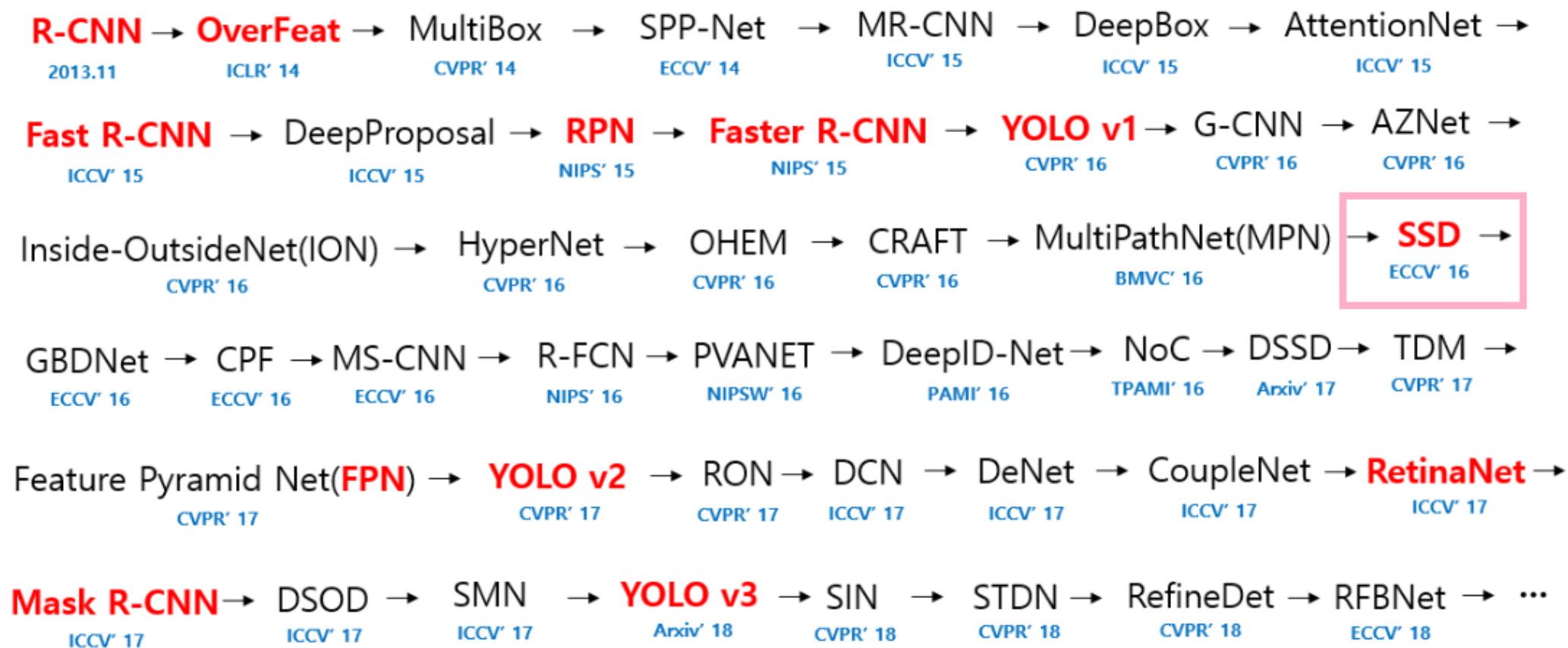
Loss & Metric



VGG16



Object Detection Models



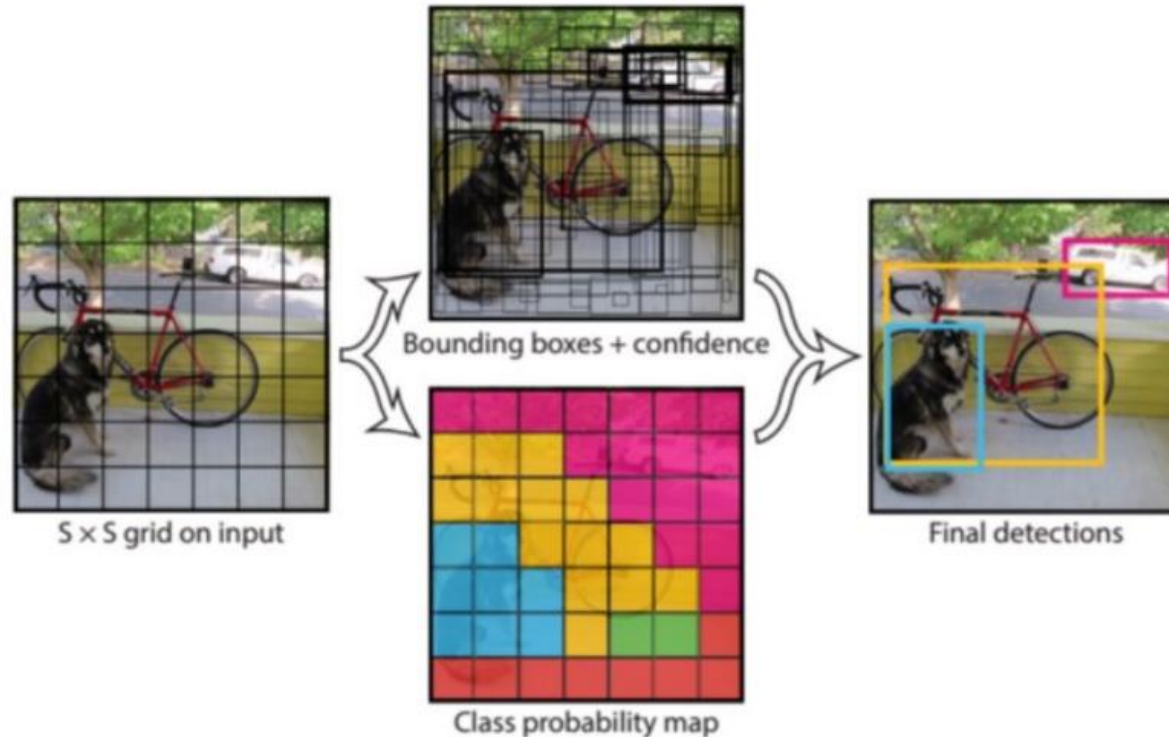
YOLO - 단일 단계 방식의 객체 탐지 알고리즘

- YOLO(You Only Look Once)는 대표적인 단일 단계 방식의 객체 탐지 알고리즘입니다. YOLO 알고리즘은 원본 이미지를 동일한 크기의 그리드로 나눕니다.
- 각 그리드에 대해 그리드 중앙을 중심으로 미리 정의된 형태(predefined shape)으로 지정된 경계박스의 개수를 예측하고 이를 기반으로 신뢰도를 계산합니다.
- 이미지에 객체가 포함되어 있는지, 또는 배경만 단독으로 있는지에 대한 여부가 포함되겠죠. 높은 객체 신뢰도를 가진 위치를 선택해 객체 카테고리를 파악합니다.



YOLO - 객체 탐지 방식

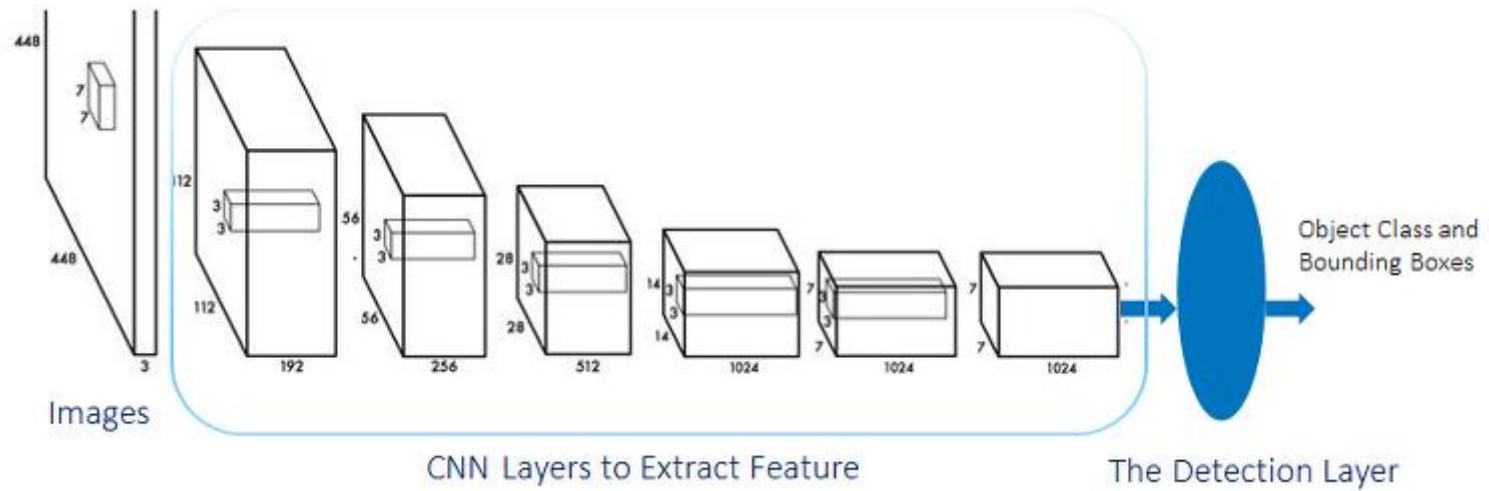
- 영역의 박스와 해당 박스의 분류를 동시에 복수개를 출력하는 네트워크
- 모두 98개($7 \times 7 \times 2$)가 제안되고, 중복되지 않고 확신도가 높은 것만 선택



모델 구조는 입력 이미지를 $S \times S$ 그리드로 나누고, 객체의 중심이 그리드 셀에 들어가면 해당 그리드 셀이 해당 객체를 감지함.



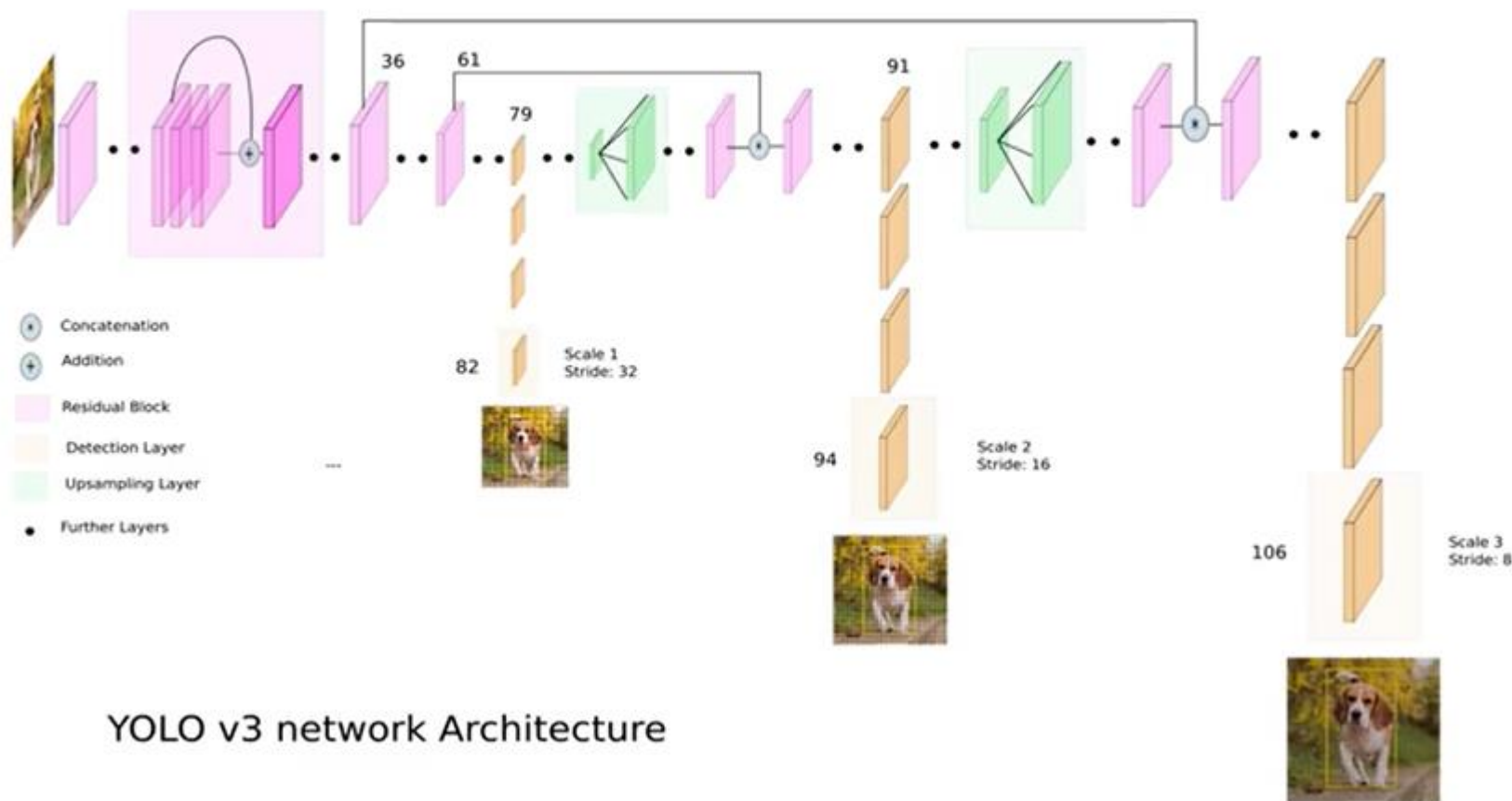
[그림 1] 앵커 박스(이미지 출처: https://www.flickr.com/photos/762_photo/16751321393/)



[그림 2] YOLO 네트워크 아키텍처

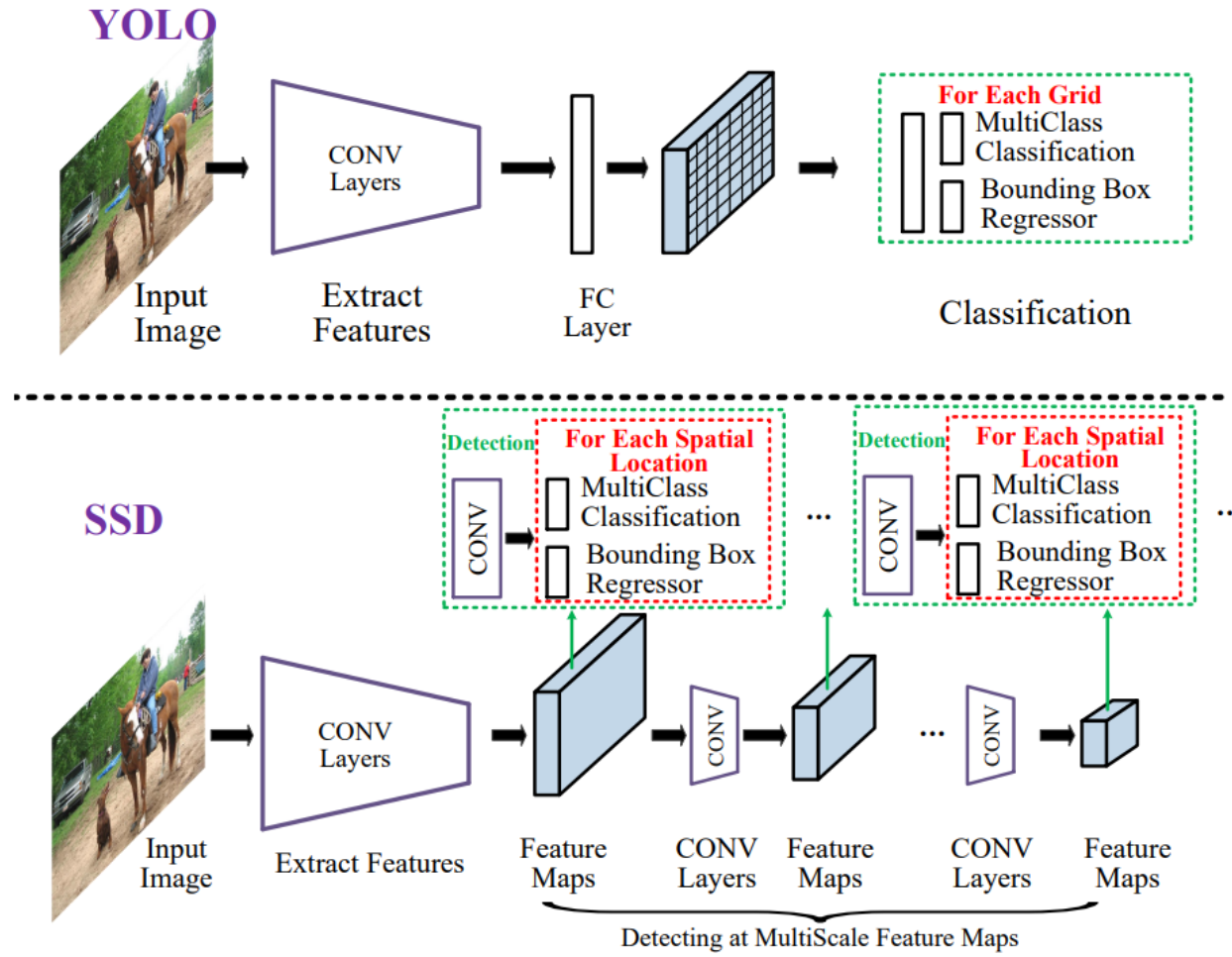
YOLO3 – Architecture

YOLO3



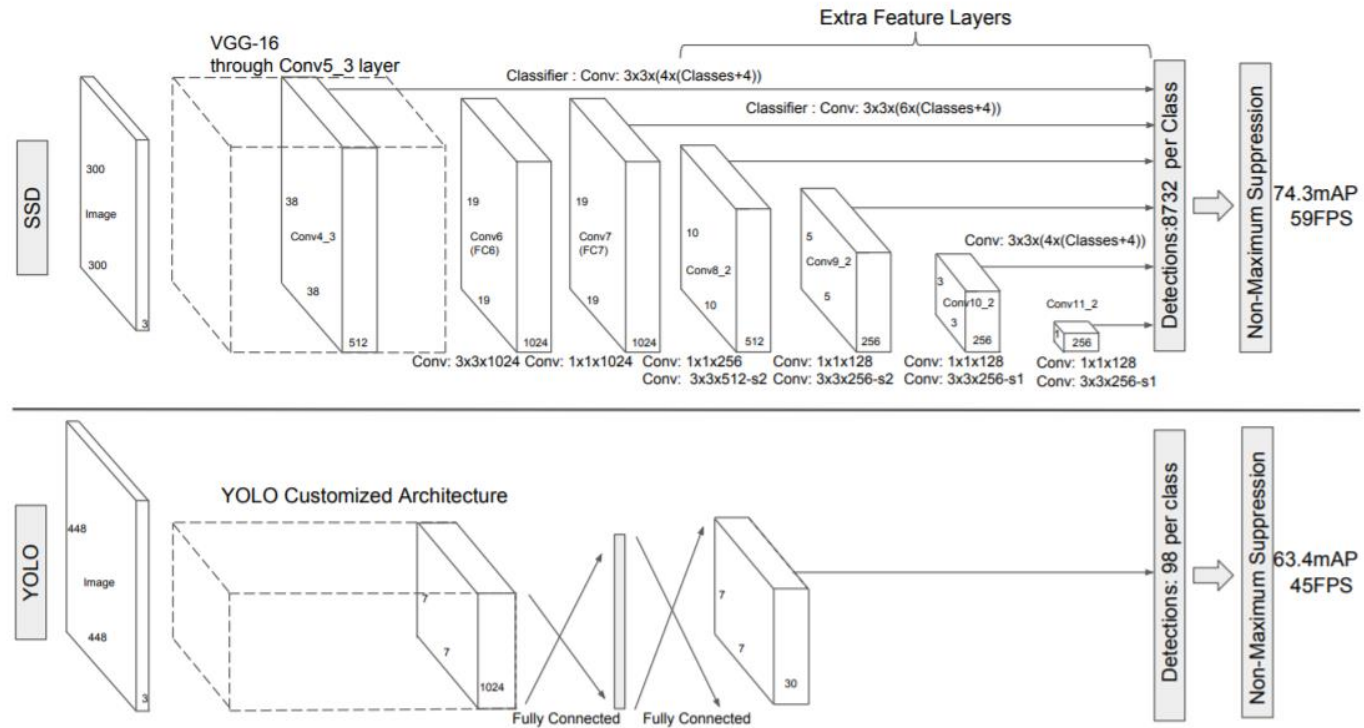
YOLO v3 network Architecture

YOLO와 SSD 모델의 차이 비교



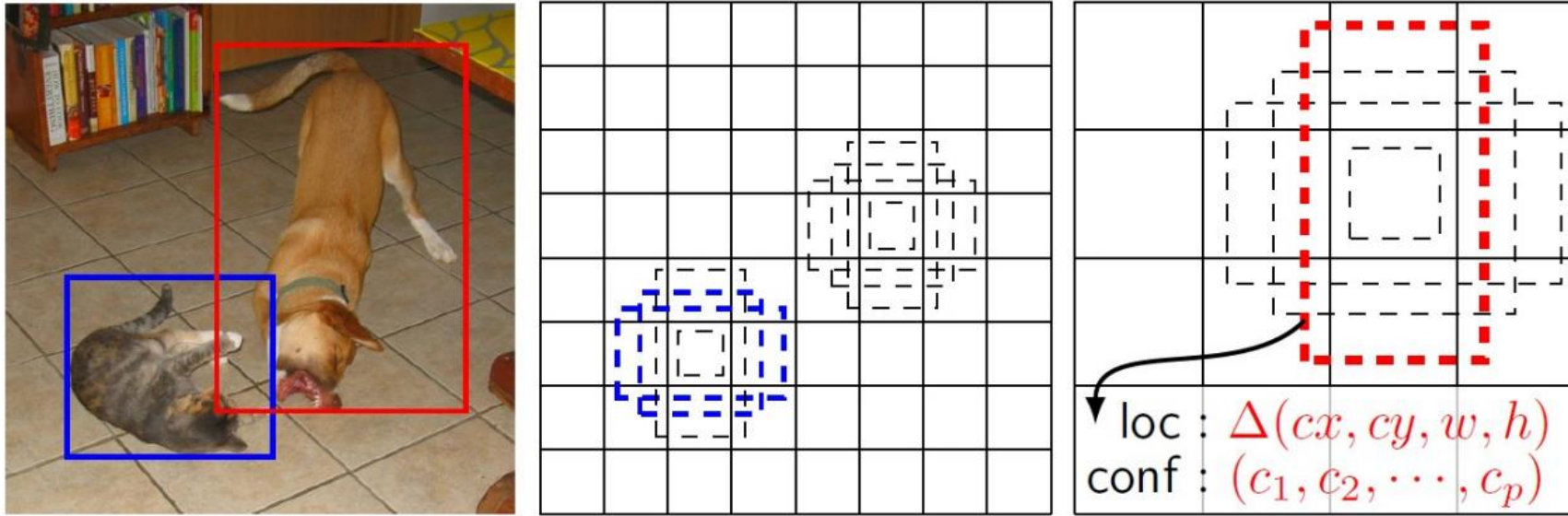
경계 상자 가설 (1 단계)에 대한 픽셀 또는 특징을 리샘플링하지 않고 접근 방식만큼 정확한 최초의 딥 네트워크 기반 객체 감지함.

Multifl Feature Map의 활용



6개의 기본 컨볼루션 네트워크의 끝에서 피처를 가져오며, 이러한 Layer의 크기는 점진적으로 줄어들고 여러 스케일에서 탐지하여 예측가능.

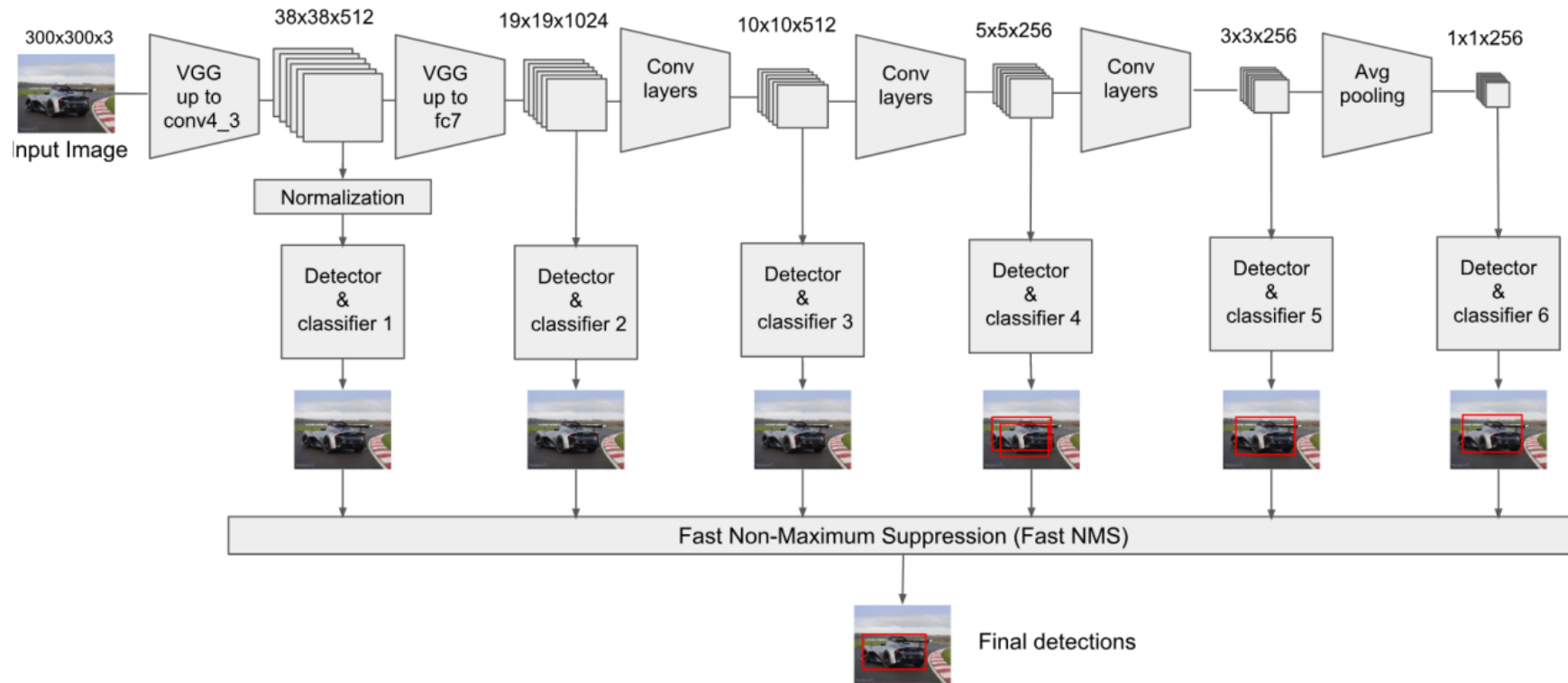
다양한 배율 Convolution Layer에서 종횡 비율에 따른 선택



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

다른 레이어의 피쳐 맵을 사용하여 스케일 차이를 처리하며, 특정 기능 맵 위치는 이미지의 특정 영역과 개체의 특정 배율에 반응하는 법을 학습함.

Anchor Box에 대한 배율 및 종횡비에 따른 선택



상대적으로 큰 물체는 종단 Layer에서 Object를 선택함.

YOLO – Anchor Boxes

- 미리 정의된 형태를 가진 경계박스 수를 '앵커 박스(Anchor Boxes)'라고 하는데요.
앵커 박스는 k -평균 알고리즘에 의한 데이터로부터 생성되며, 데이터 세트의 객체 크기와 형태에 대한 사전 정보를 확보합니다.
- 각각의 앵커는 각기 다른 크기와 형태의 객체를 탐지하도록 설계되어 있습니다.
앞장의 [그림 2]를 보면 한 장소에 3개의 앵커가 있는데요. 이 중 붉은색 앵커 박스는 가운데에 있는 사람을 탐지합니다.
- 이 알고리즘은 앵커 박스와 유사한 크기의 개체를 탐지한다는 뜻인데요.
최종 예측은 앵커의 위치나 크기와는 차이가 있습니다. 이미지의 피쳐(Feature) 맵에서 확보한 최적화된 오프셋이 앵커 위치나 크기에 추가됩니다.

앵커 박스?

이미지에서 다양한 물체를 예측하고 현지화하기 위해 EfficientDet 및 YOLO 모델과 같은 대부분의 최신 물체 감지 모델은 이전과 같이 앵커 상자로 시작하여 거기에서 조정됩니다.

최근 물체 감지 모델은 일반적으로 다음 순서로 경계 상자(Bounding Box)를 사용합니다.

이미지 주위에 수천 개의 후보 앵커 박스 형성

- 각 앵커 박스에 대해 해당 박스에서 일부 오프셋을 후보 상자로 예측
- Ground Truth 예를 기반으로 손실 함수 계산
- 주어진 오프셋 상자가 실제 물체와 겹칠 확률 계산
- 해당 확률이 0.5보다 크면 예측을 손실 함수로 인수합니다.
- 예측된 박스를 보상하고 페널티를 줌으로써 실제 객체 만 현지화 하는 쪽으로 모델을 천천히 이끌어 냄.


그렇기 때문에 모델을 약간만 훈련했을 때 모든 곳에 예측 상자가 표시되는 것을 볼 수 있습니다.

Anchor sorting and filtering



https://github.com/matterport/Mask_RCNN

학습방법 - 매칭 전략

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


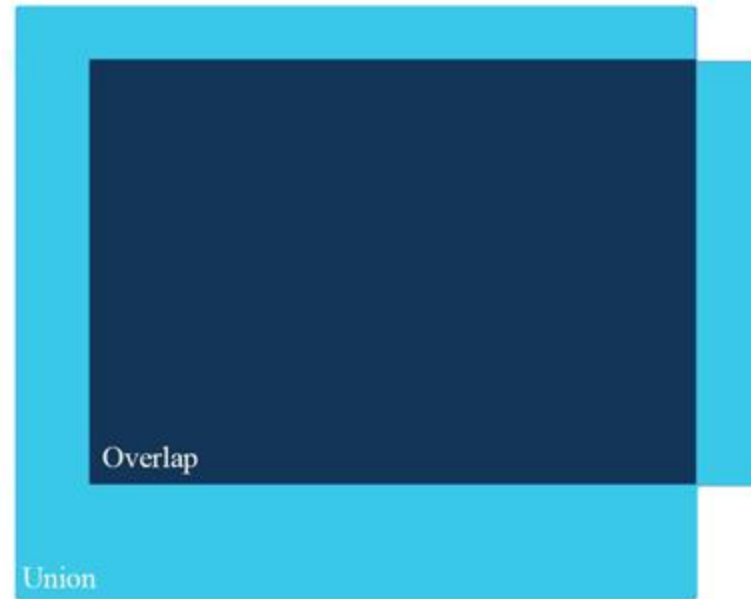
The diagram illustrates the Jaccard Overlap (IoU) concept. It shows two overlapping rectangles: a white one on the left and a blue one on the right. The intersection of these two rectangles is shaded in a darker blue. Below this, a single large blue shape represents the union of the two rectangles, which is the total area covered by both rectangles combined.

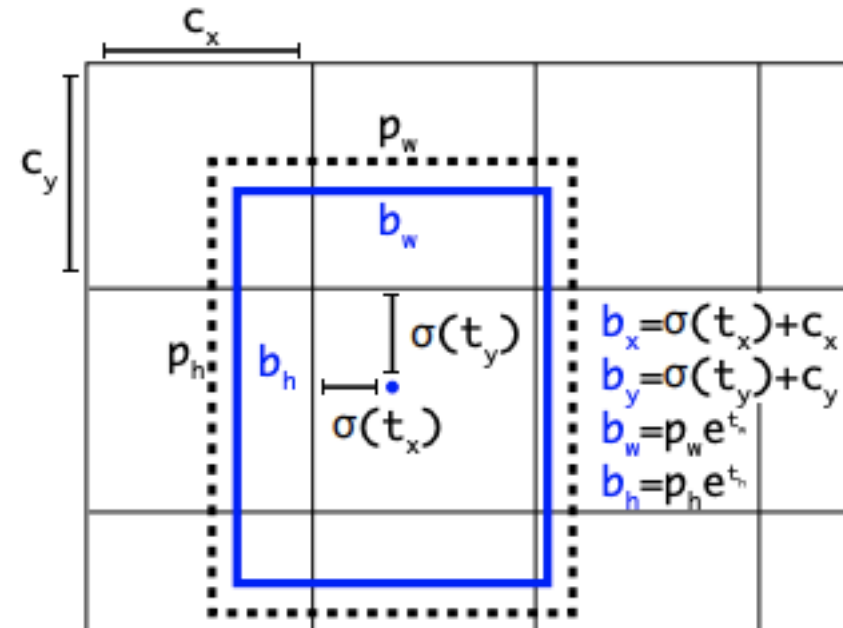
모든 Ground Truth 상자에는 최소한 1 개의 대응을 시키며, jaccard 겹침이 가장 큰 Default Box에 각 원본 영상과 일치시킴.

그런 다음 jaccard 오버랩이 임계 값 (0.5)보다 높은 모든 Ground Truth에 Default Box에 일치시킴.



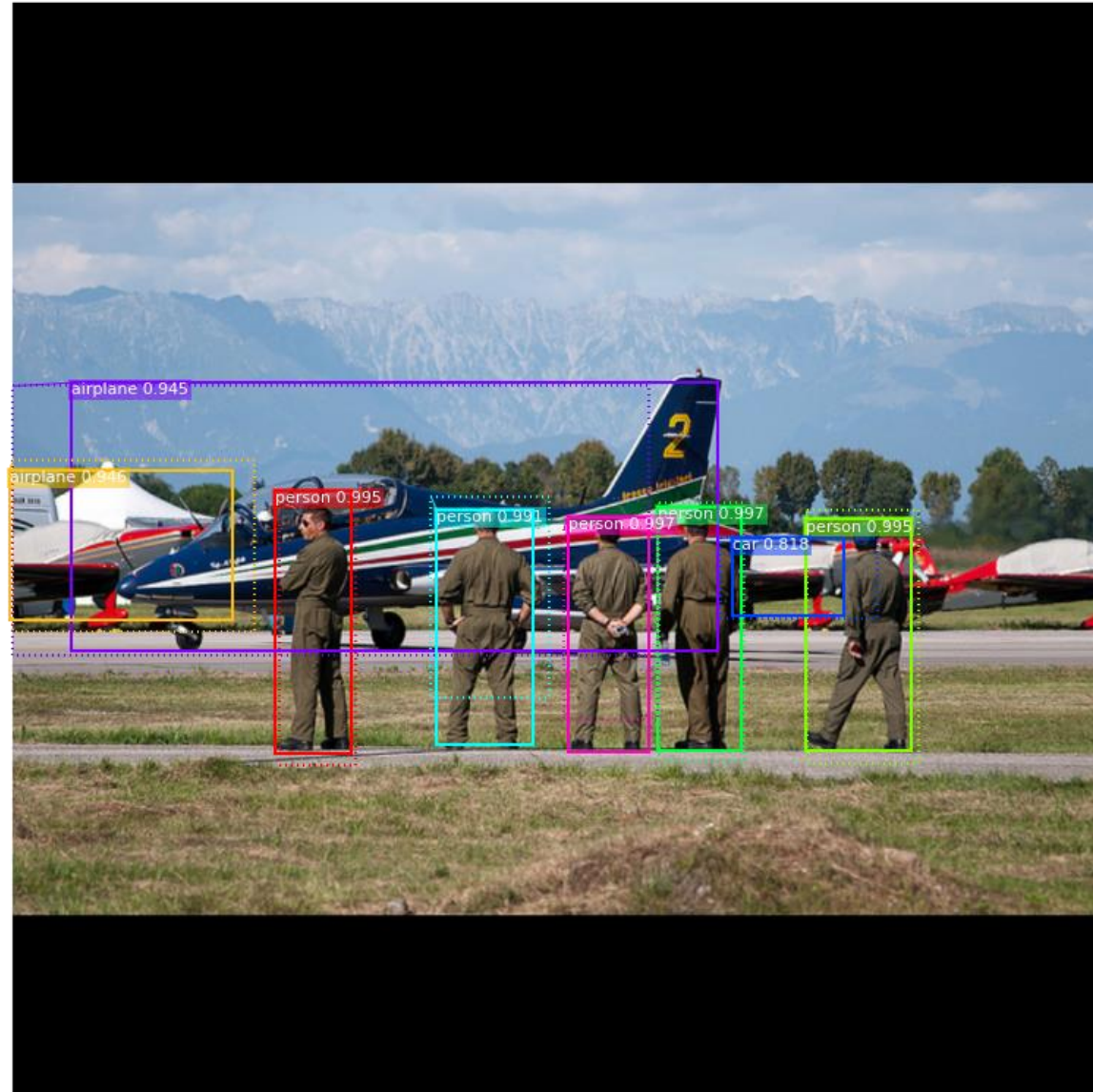
$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$





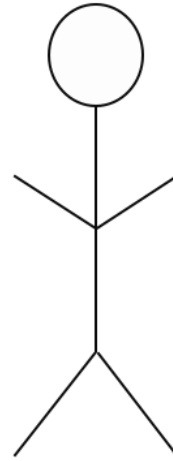
A picture of bounding box prediction based on an anchor box

Bounding Box Refinement(Detection after NMS)



Y2

Object Class



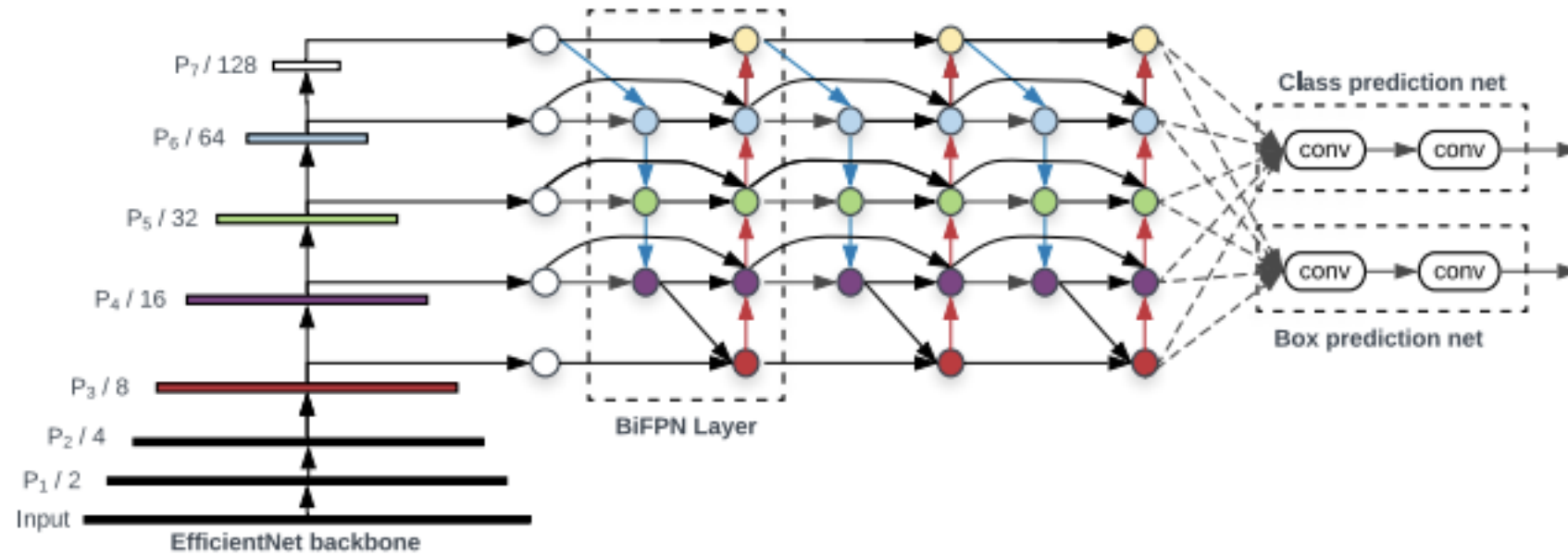
Y1

X1

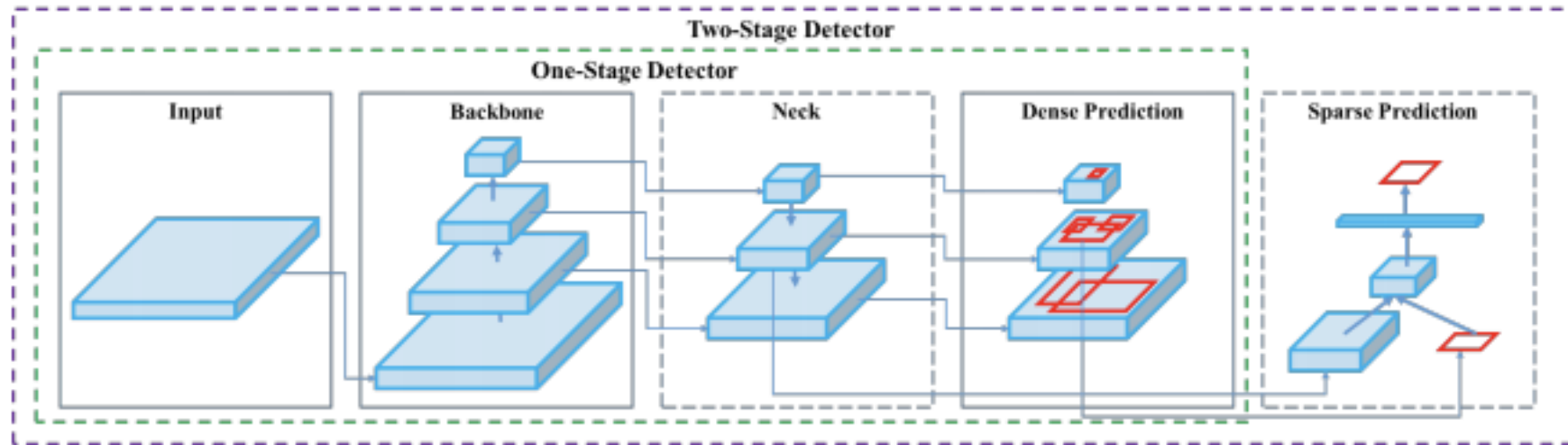
X2



The anatomy of an object detector ([citation](#))



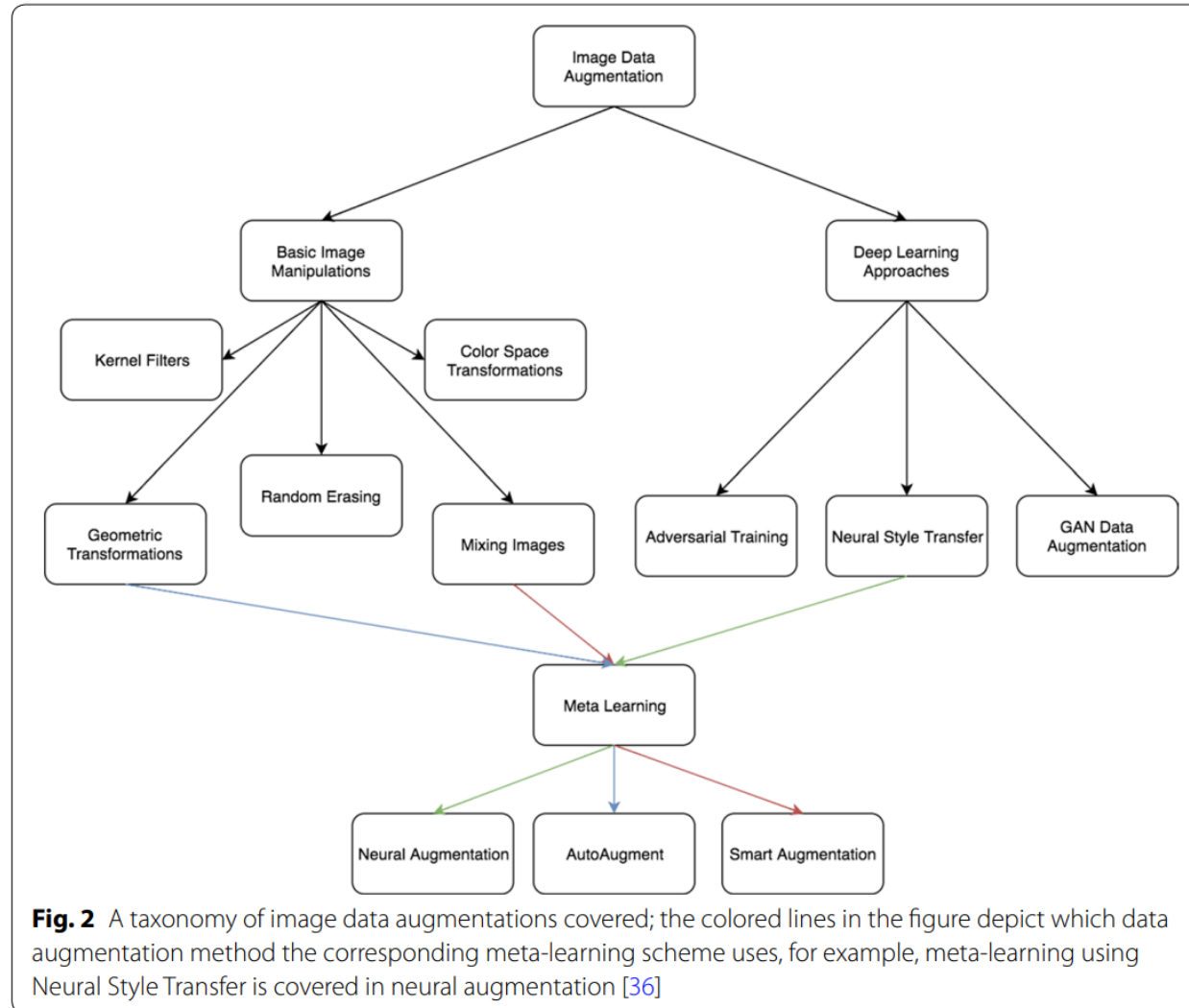
Another picture of the object detection process ([citation](#) from YOLOv4)



```
# parameters nc: 80
# number of classes depth_multiple: 0.33
# model depth multiple width_multiple: 0.50 #
layer channel multiple
# anchors
anchors:
  - [116,90, 156,198, 373,326] # P5/32
  - [30,61, 62,45, 59,119] # P4/16
  - [10,13, 16,30, 33,23] # P3/8
# YOLOv5 backbone
```

The anchors in the YOLOv5 config file are now auto learned based on training data.

Augmentation



Attention is all you needs

Attention Is All You Need

Ashish Vaswani^{*}
Google Brain
avaswani@google.com

Llion Jones^{*}
Google Research
llion@google.com

Noam Shazeer^{*}
Google Brain
noam@google.com

Aidan N. Gomez[†]
University of Toronto
aidan@cs.toronto.edu

Niki Parmar^{*}
Google Research
nikip@google.com

Lukasz Kaiser^{*}
Google Brain
lukasz.kaiser@google.com

Jakob Uszkoreit^{*}
Google Research
usz@google.com

Illia Polosukhin[‡]
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

1 Introduction

Recurrent neural networks, long short-term memory [13] and gated recurrent [7] neural networks in particular, have been firmly established as state of the art approaches in sequence modeling and

^{*}Equal contribution. Listing order is random. Jakob proposed replacing RNNs with self-attention and started the effort to evaluate this idea. Ashish, with Illia, designed and implemented the first Transformer models and has been crucially involved in every aspect of this work. Noam proposed scaled dot-product attention, multi-head attention and the parameter-free position representation and became the other person involved in nearly every detail. Niki designed, implemented, tuned and evaluated countless model variants in our original codebase and tensor2tensor. Llion also experimented with novel model variants, was responsible for our initial codebase, and efficient inference and visualizations. Lukasz and Aidan spent countless long days designing various parts of and implementing tensor2tensor, replacing our earlier codebase, greatly improving results and massively accelerating our research.

[†]Work performed while at Google Brain.

[‡]Work performed while at Google Research.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

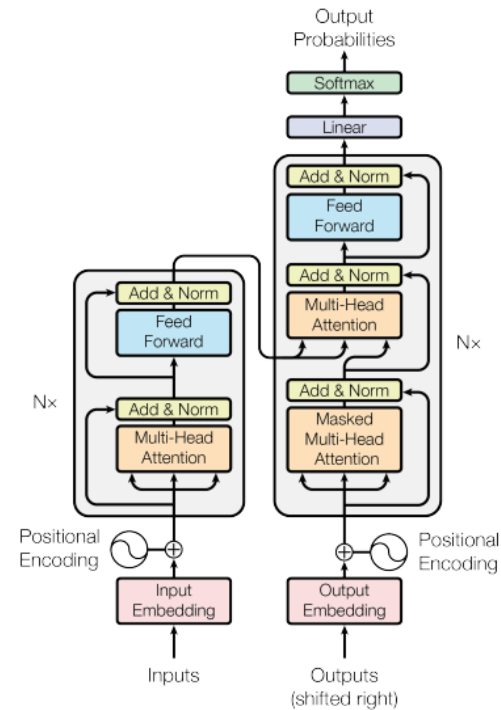
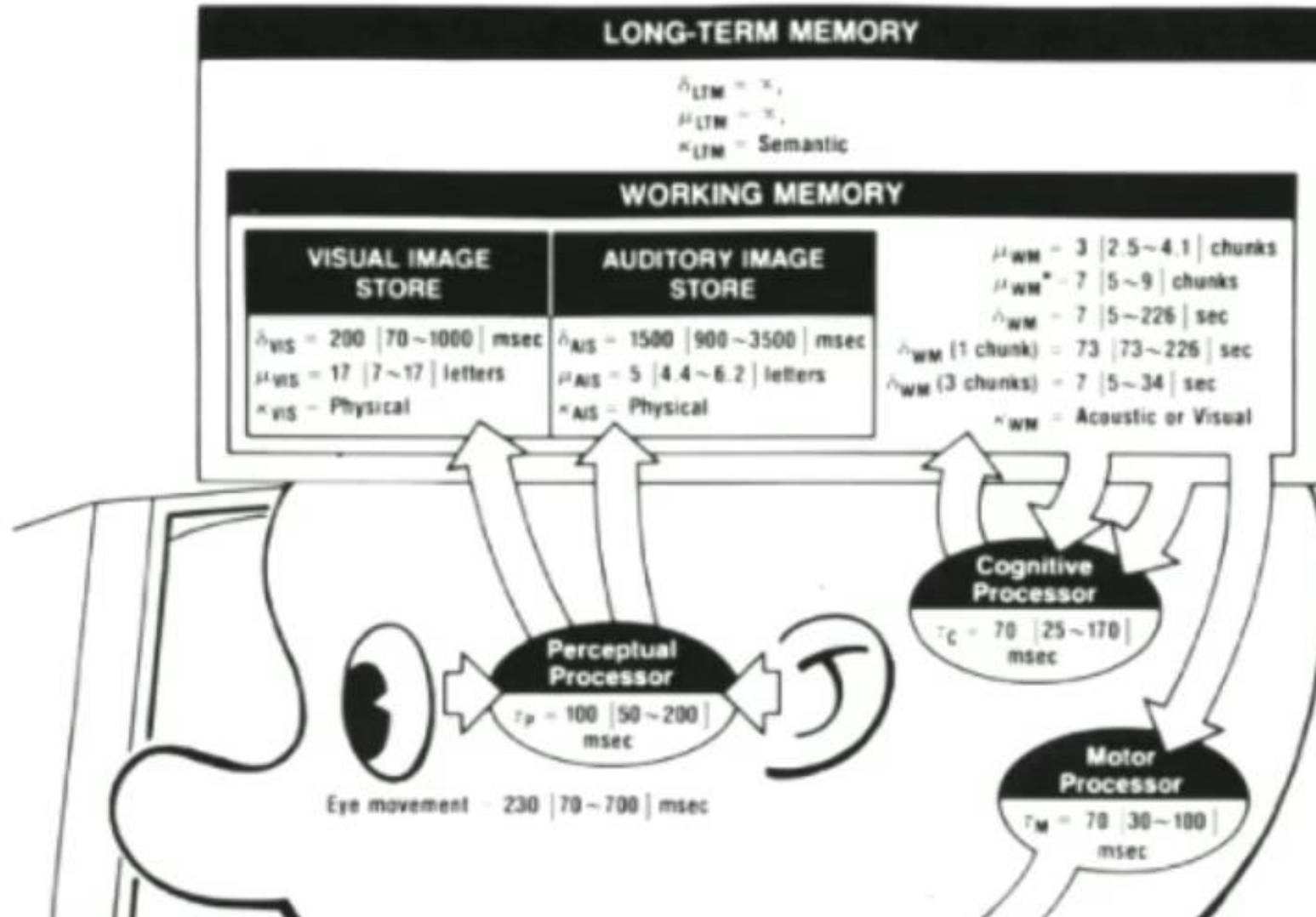


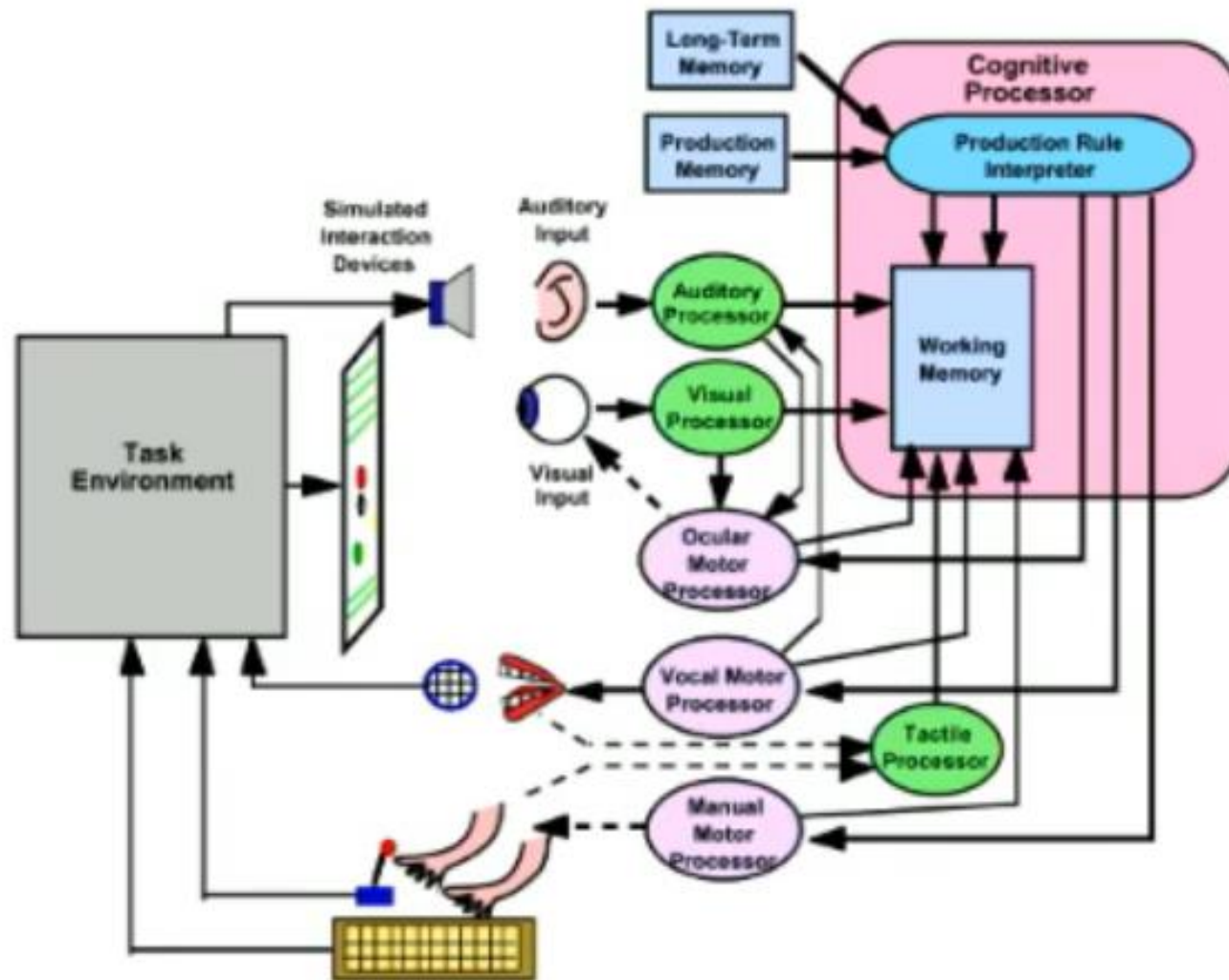
Figure 1: The Transformer - model architecture.

https://keras.io/examples/vision/image_classification_with_vision_transformer/

Artificial intelligence started as a field whose goal was to replicate human level intelligence in a machine.

인공 지능은 인간 수준의 지능을 기계에 복제하는 것을 목표로 시작되었습니다.





The EPIC architecture
for cognition
and performance
with application to
human-computer interaction

강사 소개

정 준 수 / Ph.D (heinem@naver.com)

- 前) 삼성전자 연구원
- 前) 삼성의료원 (삼성생명과학연구소)
- 前) 삼성SDS (정보기술연구소)
- 現) (사)한국인공지능협회, AI, 머신러닝 강의
- 現) 한국소프트웨어산업협회, AI, 머신러닝 강의
- 現) 서울디지털재단, AI 자문위원
- 現) 한성대학교 교수(겸)
- 전문분야: 시각 모델링, 머신러닝(ML), RPA
- <https://github.com/JSJeong-me/>

