

# Capstone Design

## Team F

2016310237 김동우

2016314577 김동한

2016311033 김영현

2017312329 최형규



# 주제 선정 배경



머신 러닝 기술이 학습을 통해 낙서를 인식할 수 있을까요?

여러분의 그림으로 머신 러닝의 학습을 도와주세요. Google은 머신 러닝 연구를 위해  
세계 최대의 낙서 데이터 세트를 오픈소스로 공유합니다

시작하기

# 주제 선정 배경

그림 1/6

다음을 그리세요

피자

20초 이내

알겠어요!

# 주제 선정 배경

피자 그리기

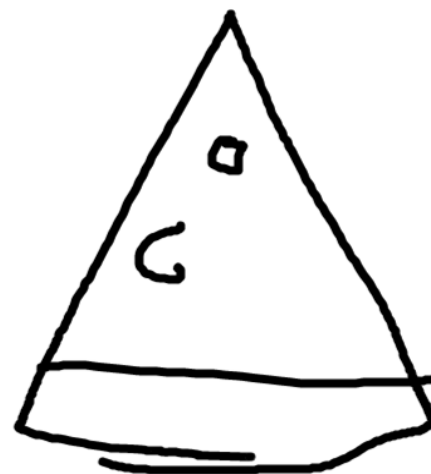
00:15



원지 알 것 같아요. 줄

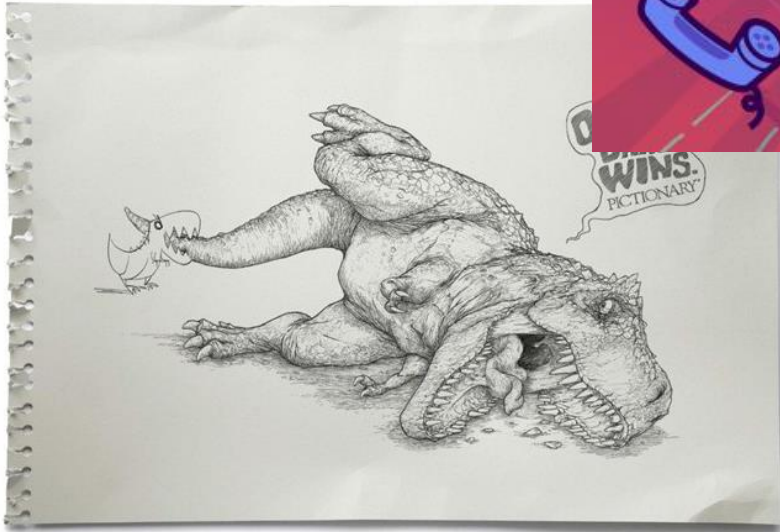
피자 그리기

00:09



이제 알겠어요. 피자 맞죠?

# 주제 관련 사전조사



# 주제 관련 사전조사



# 프로젝트 개요

---

## GAME MODE 1

1. 키워드 주어짐
2. 시간 제한 이내에 키워드에 맞는 그림 스케치
3. AI가 각 그림을 점수매겨 포인트를 줌

## GAME MODE 2

1. 키워드 주어짐
2. 첫번째 사람이 시간 제한 이내에 키워드에 맞는 그림 스케치
3. AI가 이에 대한 키워드 다음 사람에게 전달
4. 2~3 반복
5. 종료 후 변하는 과정 출력



# 프로젝트 개요

---

## GAME MODE 4

1. 키워드 주어짐
2. 첫번째 사람이 시간 제한 이  
내에 키워드에 맞는 그림 스  
케치
3. AI가 이에 대한 키워드 다음  
사람에게 전달
4. 2~3 반복
5. 종료 후 변하는 과정 출력

# 프로젝트 개요

5점		AI, 그림을 맞춰라		20초	
1번 플레이어	2	2번 플레이어	4	채팅	
				1: 정말 잘그리시네요	
				2: 감사합니다^^	
3번 플레이어	0	4번 플레이어	1	채팅내용	

# 구현 기술 스택

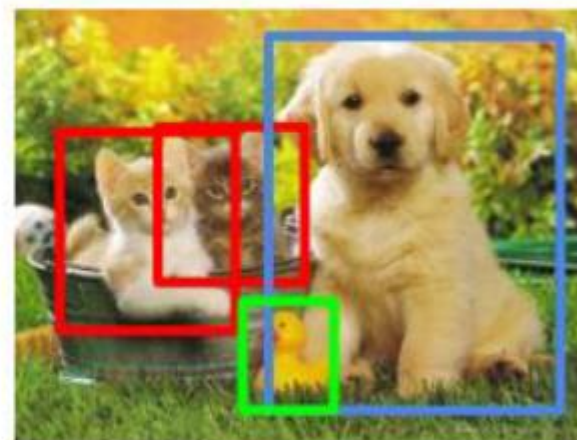
<Image Classification 기술> vs <Object Detection 기술>

**Classification**



CAT

**Object Detection**

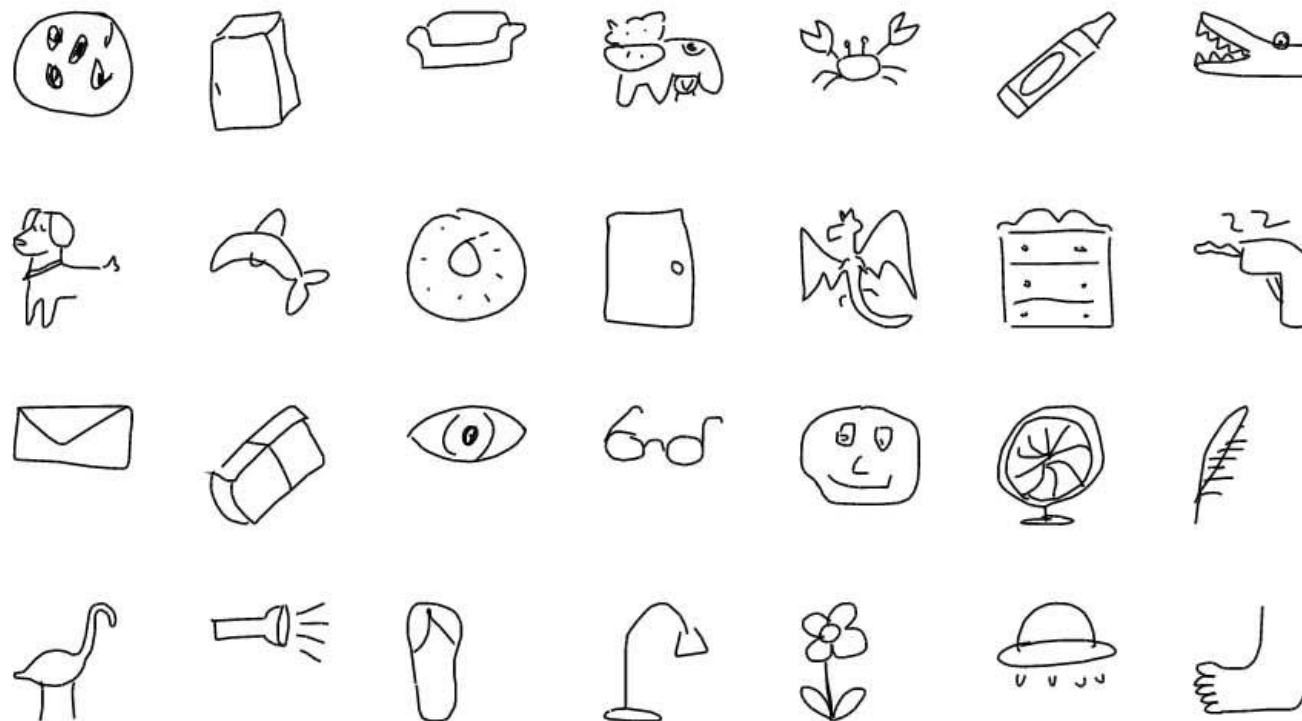


CAT, DOG, DUCK

**Image Classification** 입력으로 하나의 객체만 있는 사진을 받아 사진 속 개체가 어떤 카테고리에 속하는지 판단

**Object Detection** : 입력으로 여러개의 객체가 있는 사진을 받아 사진 속 각 개체가 어떤 카테고리에 속하는지 판단

# 구현 기술 스택



**“Quick, Draw! The data”** 오픈 소스 데이터 셋  
345 카테고리, 5000만 개의 낙서 데이터

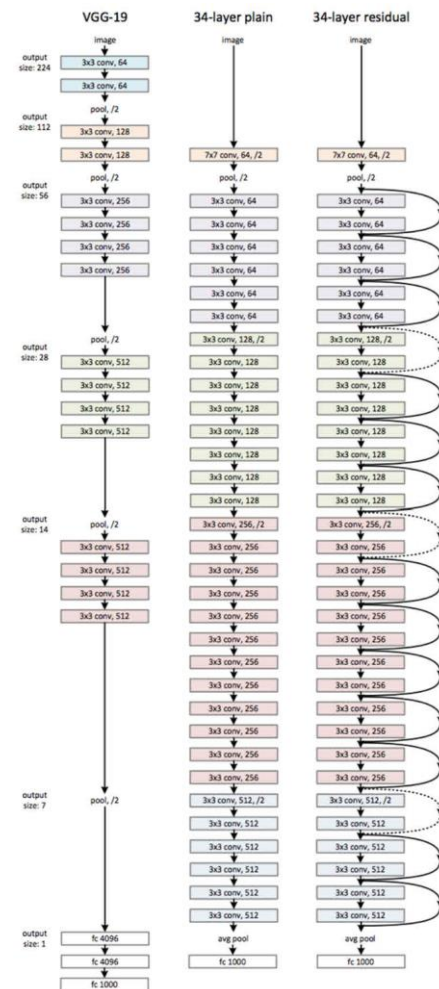
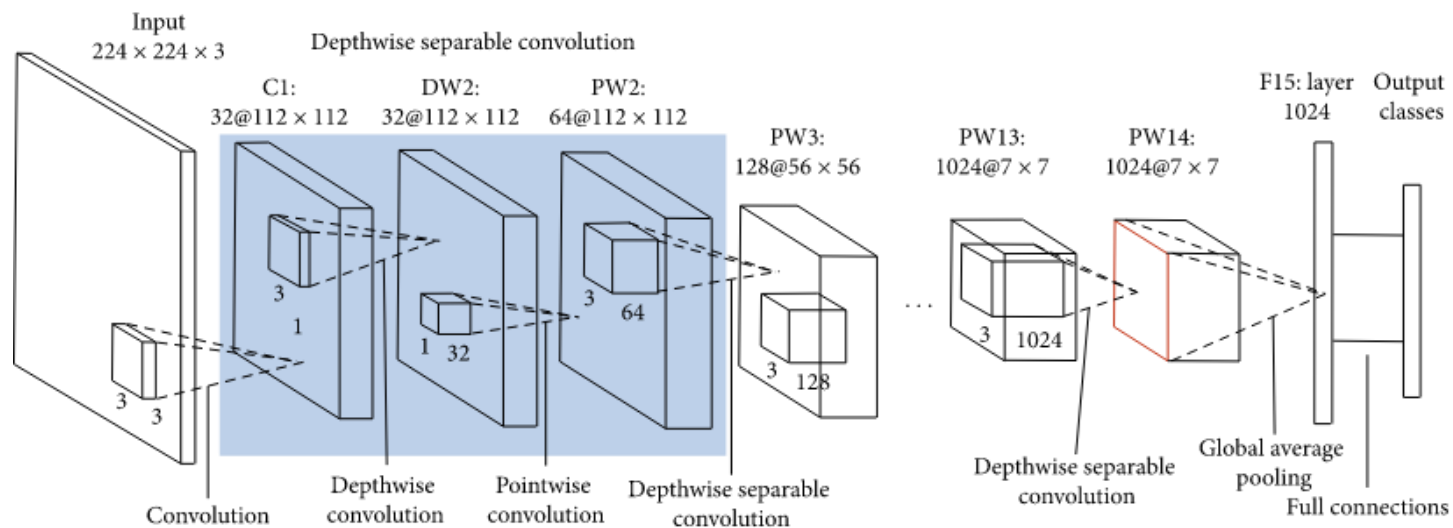
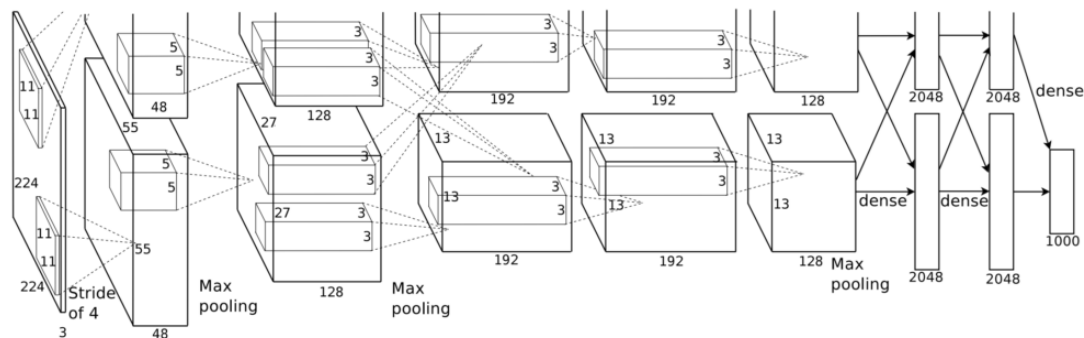
# 구현 기술 스택

Key	Type	Description
key_id	64-bit unsigned integer	A unique identifier across all drawings.
word	string	Category the player was prompted to draw.
recognized	boolean	Whether the word was recognized by the game.
timestamp	datetime	When the drawing was created.
countrycode	string	A two letter country code (ISO 3166-1 alpha-2) of where the player was located.
drawing	string	A JSON array representing the vector drawing

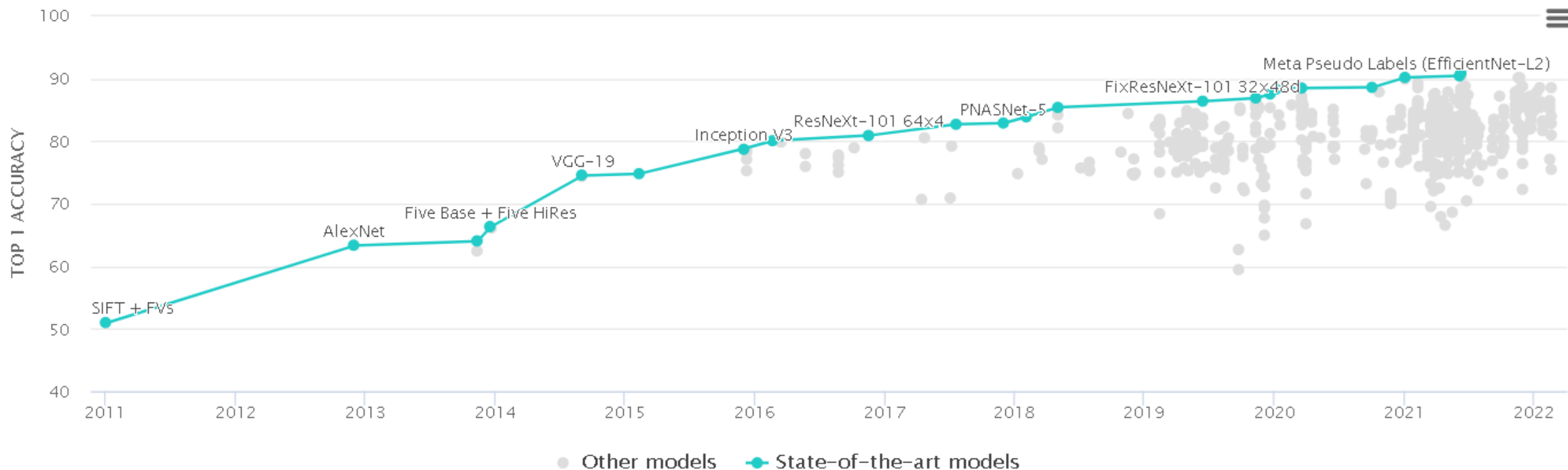
<https://github.com/googlecreativelab/quickdraw-dataset>

# 구현 기술 스택

## <Image Classification 딥러닝 모델>



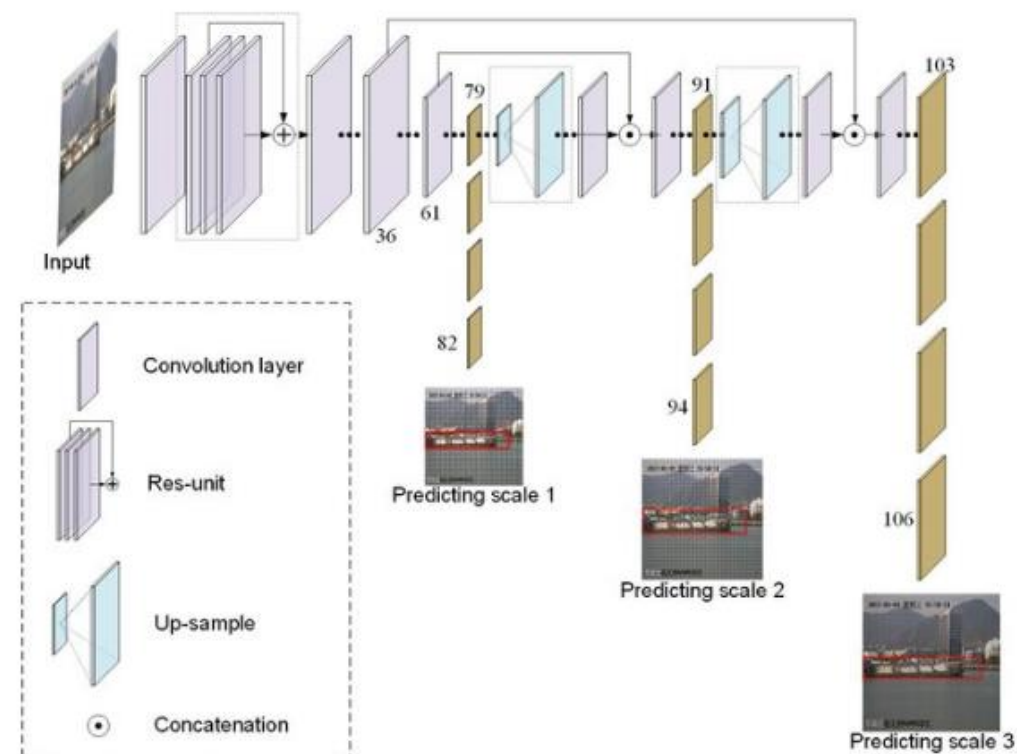
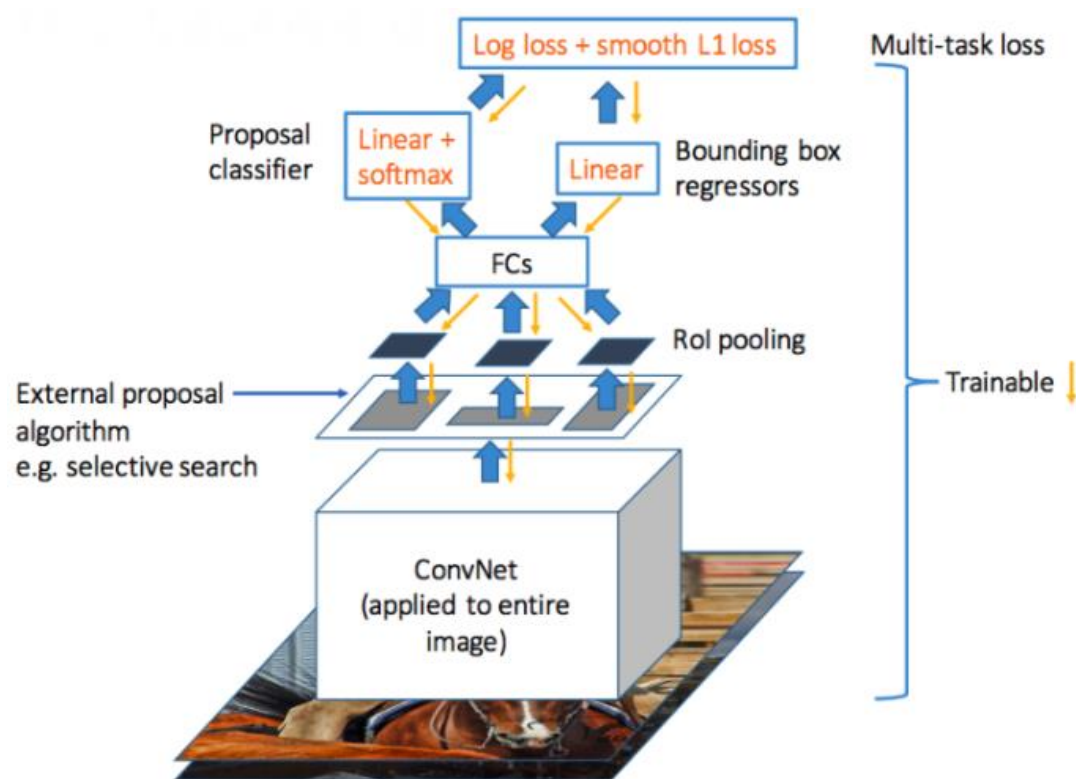
# 구현 기술 스택



Sota [ Imagenet dataset ] (neurips, 2021) “CoAtNet:: Marrying Convolution and Attention for All Data Sizes”

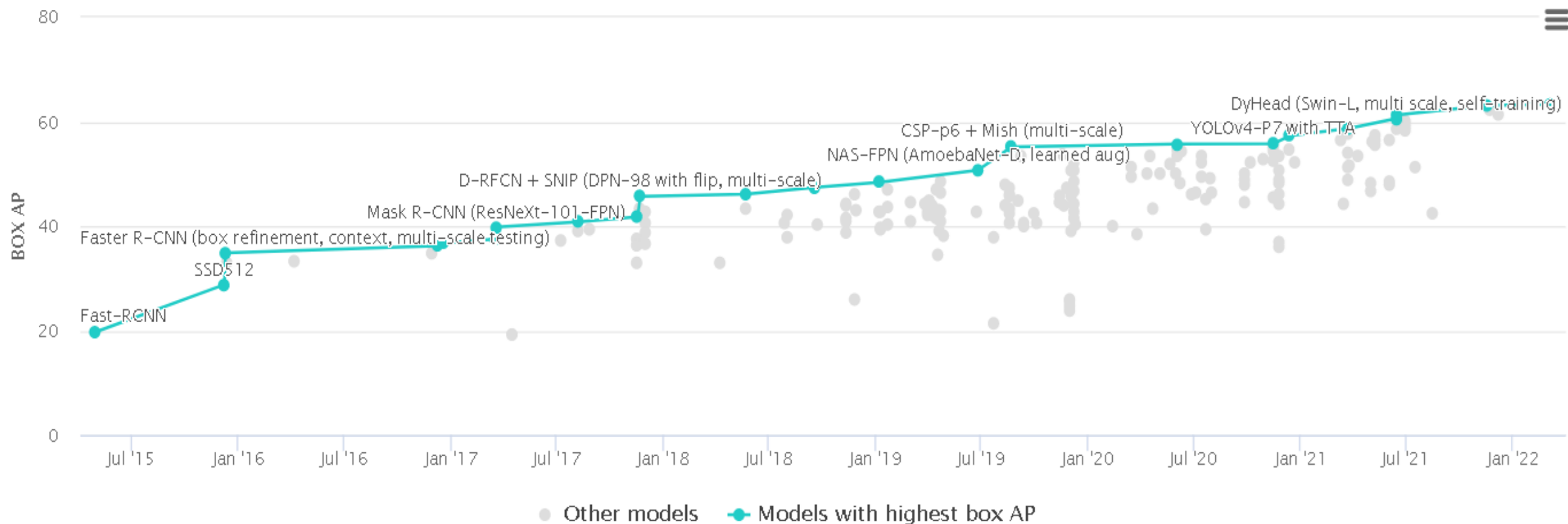
# 구현 기술 스택

## <Object Detection 모델>





# 구현 기술 스택



Sota [ CoCo dataset ]:"DINO: DETR with improved DeNoising Anchor Boxes for End-to-End Object Detection"

# 기술 스택 - 딥러닝 서버

---



# 기술 스택 - 게임 구현



# 역할 분담

## 게임환경

김동한

게임서버 구축

최형규

게임 프론트엔드/ ui

## AI 구축

김영현

딥러닝

김동우

딥러닝 서버 개발

***By Adding***

# 프로젝트 개요

---

**A.I, GO DOODLE!**

# 개선 사항

현존하는 모델의 활용 / 여러 모델의 테스트 / 모델의 개선  
데이터셋을 학습할 수 있을만한 리소스가 있는지에 대한 고민

사람이 시간내에 " " 같은 선으로 된 그림 데이터셋이 있어야 할 것  
기존에 존재하는 게임들에 비해 차별성이 두드러지지 않는 것 같다.  
약간 튜링테스트 같은 것을 게임으로 구현하는 프로젝트라고 생각합니다.  
사람이 시간내에 빠르게 그린 선으로 된 그림 데이터셋이 있어야 할 것  
같습니다.

## 3줄 요약

- 1) 차별점이 무엇인가
- 2) 데이터셋은 무엇을 쓸 것인가
- 3) 모델은 어떤 것을 사용할 예정인가

+ ) 간략한 프로젝트 일정

# 차별점



머신 러닝 기술이 학습을 통해 낙서를 인식할 수 있을까요?  
여러분의 그림으로 머신 러닝의 학습을 도와주세요. Google은 머신 러닝 연구를 위해  
[세계 최대의 낙서 데이터 세트](#)를 오픈소스로 공유합니다

시작하기



# 게임 모드

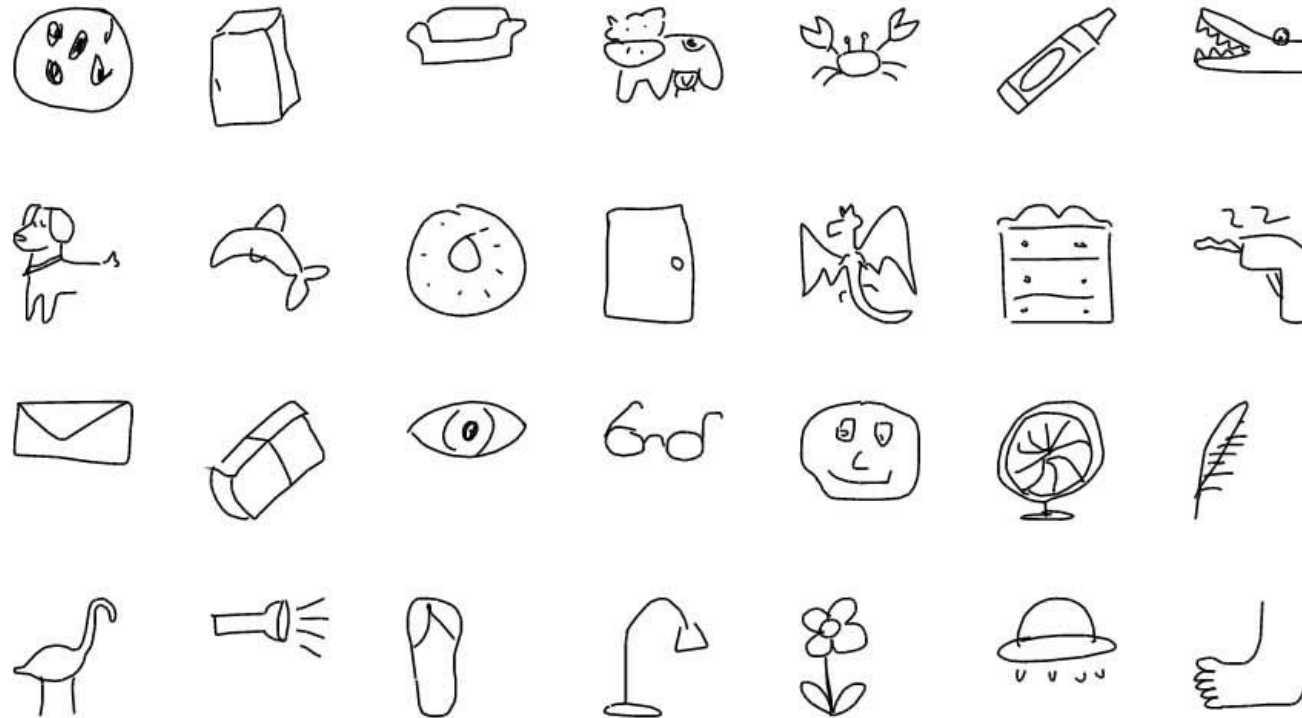
## GAME MODE 1

1. 키워드 주어짐
2. 시간 제한 이내에 키워드에 맞게 각자 그림을 그림
3. AI가 각 그림을 점수매겨 포인트를 줌
4. 정해진 점수 먼저 도달하면 승리

## GAME MODE 2

1. 키워드 주어짐
2. 첫번째 사람이 시간 제한 이내에 키워드에 맞게 스케치
3. AI가 이를 입력받아 도출한 키워드를 다음사람에게 전달
4. 2~3과정을 반복
5. 종료 후 변하는 과정 출력

# 사용 데이터 셋



“Quick, Draw! The data” 오픈 소스 데이터 셋

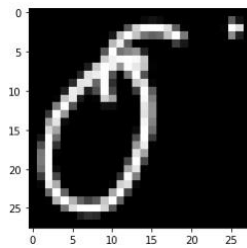
345 카테고리, 5000만 개의 낙서 데이터 중 선정 + 데이터 증강 ( rotation, transition, flip, scale...)

# 사용 데이터 셋

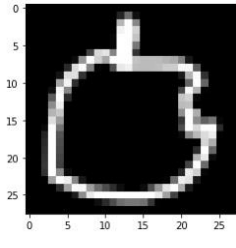
```
import matplotlib.pyplot as plt

for i, image in enumerate(train_images):
    print(class_list[i])
    plt.imshow(image, cmap='gray')
    plt.show()

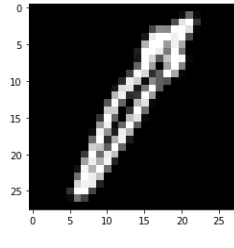
plt.show()
```



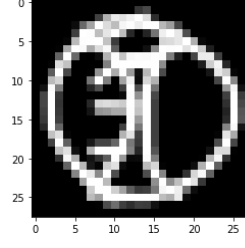
(사과)



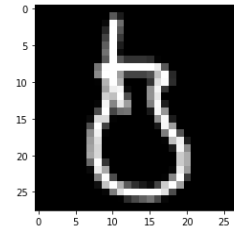
(아스파라거스)



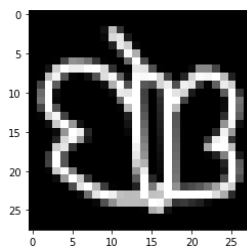
(야구공)



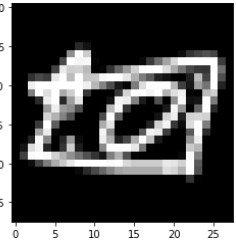
(꽃병)



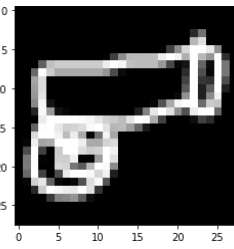
(티셔츠)



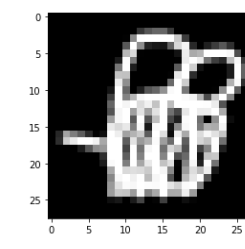
(나비)



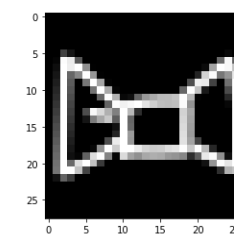
( 카메라)



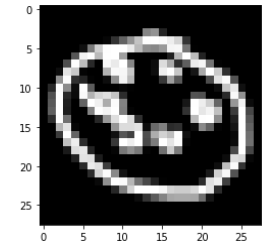
(대포)



(벨)




(리본)



(쿠키)

# Google Colab

 Colaboratory에 오신 것을 환영합니다

파일 수정 보기 삽입 런타임 도구 도움말

공유 설정 영현

연결 수정 가능


목차

시작하기  
데이터 과학  
머신러닝  
추가 리소스  
추천 예시  
섹션

+ 코드 + 텍스트 Drive로 복사

Colab 시작 페이지

Colab에 이미 익숙하다면 이 동영상을 통해 양방향 테이블, 코드 실행 기록 보기, 명령어 팔레트에 관해 알아보세요.



Colab이란?

Colaboratory(줄여서 'Colab'이라고 함)을 통해 브라우저 내에서 Python 스크립트를 작성하고 실행할 수 있습니다.

- 구성이 필요하지 않음
- GPU 무료 액세스
- 간편한 공유

학생이든, 데이터 과학자든, AI 연구원이든 Colab으로 업무를 더욱 간편하게 처리할 수 있습니다. [Colab 소개 영상](#)에서 자세한 내용을 확인하거나 아래에서 시작해 보세요.

시작하기

지금 이 글 게시 모서리 적는 웹페이지가 아니라 코드를 작성하고 실행하는 이노브 대하현 하려의 Colab 메모장이니디

# Colab 프로와 프로+

## Colab

무료

현재 요금제

- ✓ 구독이 필요하지 않습니다.

권장

## Colab Pro

\$9.99/월

- ✓ 더 빠른 GPU  
더 빠른 GPU 및 TPU에 액세스하므로 코드를 실행하는 중에 기다리는 시간이 적어집니다.
- ✓ 추가 메모리  
RAM과 디스크 용량이 크면 더 많은 데이터를 저장할 수 있습니다.
- ✓ 더 긴 런타임  
메모장 런타임이 길어지고 비활성 시간 초과가 적게 발생하면 연결이 끊기는 빈도가 낮아집니다.

## Colab Pro+

\$49.99/월

- ✓ 백그라운드 실행  
노트북은 브라우저를 닫은 후에도 계속 실행됩니다.
- ✓ 더 빠른 GPU  
더 빠른 GPU 및 TPU에 우선적으로 액세스하므로 코드를 실행하는 중에 기다리는 시간이 적어집니다.
- ✓ 더욱 넉넉한 메모리  
그 어느 때보다도 많은 메모리를 문제 없이 사용할 수 있습니다.
- ✓ 더욱 긴 런타임  
Colab에서 가장 긴 노트북 실행 시간이 제공되므로 작업을 완료할 수 있습니다.

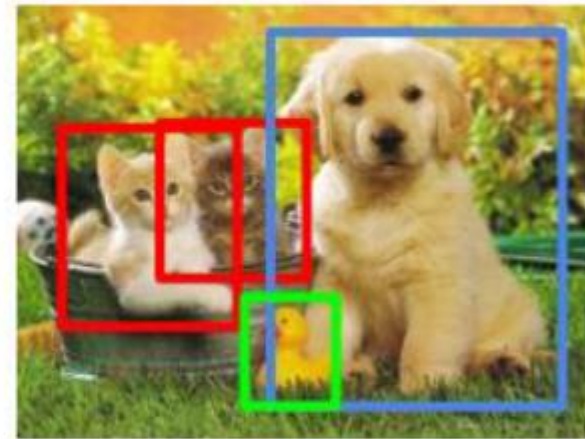
# 구현 기술 스택

## Classification



CAT

## Object Detection

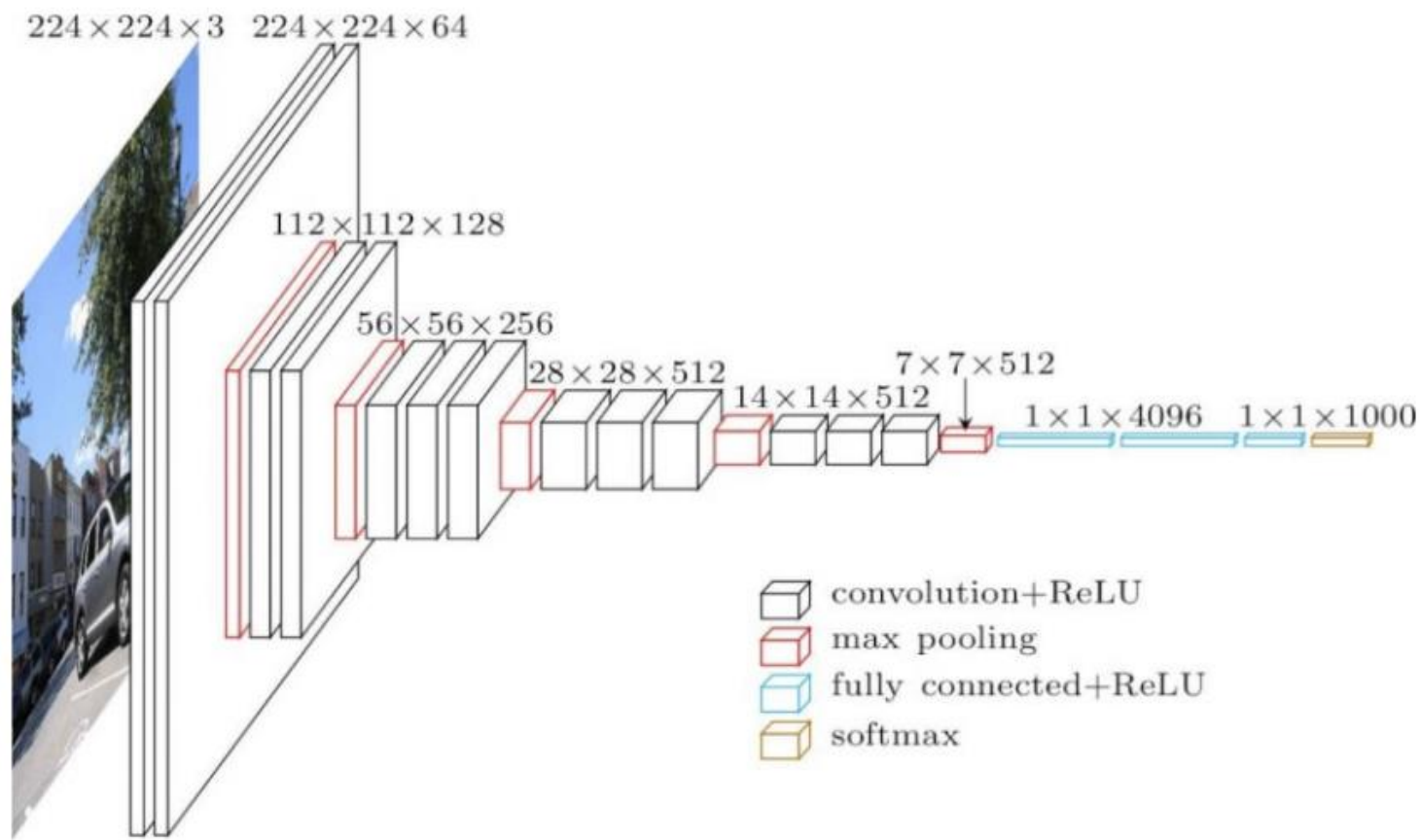


CAT, DOG, DUCK

**Image Classification** : 입력으로 하나의 객체만 있는 사진을 받아 사진 속 객체가 어떤 카테고리에 속하는지 판단

**Object Detection** : 입력으로 여러개의 객체가 있는 사진을 받아 사진 속 각 객체가 어떤 카테고리에 속하는지 판단

# VGGNet-16,19



VGG16 Architecture

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input ( $224 \times 224$ RGB image)					
conv3-64	conv3-64 <b>LRN</b>	conv3-64 <b>conv3-64</b>	conv3-64	conv3-64	conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128	conv3-128	conv3-128
maxpool					
conv3-256	conv3-256	conv3-256	conv3-256	conv3-256	conv3-256
conv3-256	conv3-256	conv3-256	conv3-256 <b>conv1-256</b>	conv3-256 <b>conv3-256</b>	conv3-256
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512 <b>conv1-512</b>	conv3-512 <b>conv3-512</b>	conv3-512
maxpool					
conv3-512	conv3-512	conv3-512	conv3-512	conv3-512	conv3-512
conv3-512	conv3-512	conv3-512	conv3-512 <b>conv1-512</b>	conv3-512 <b>conv3-512</b>	conv3-512
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

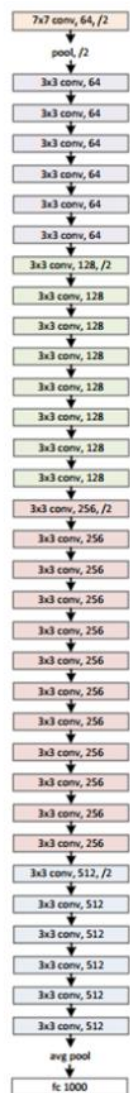
Table 2: Number of parameters (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

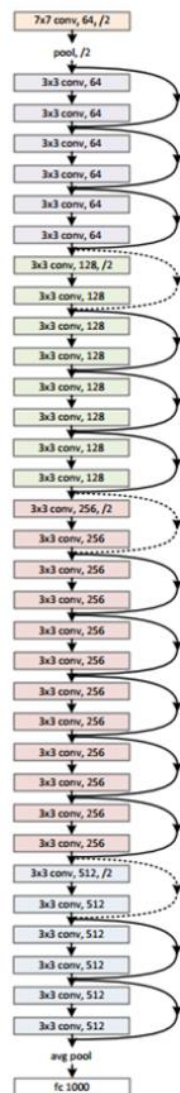


# ResNet-50,101,152

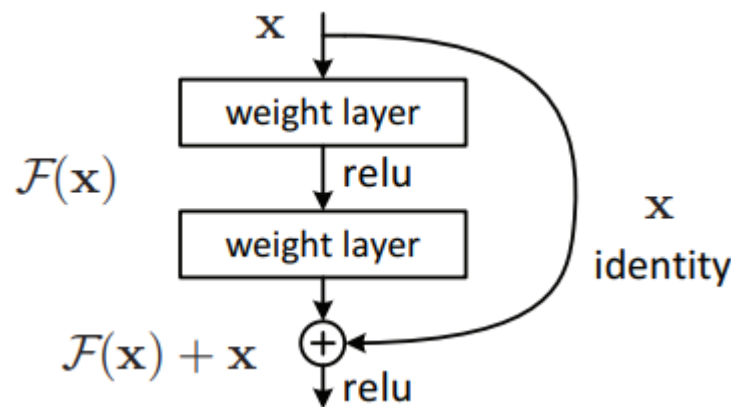
plain net



ResNet



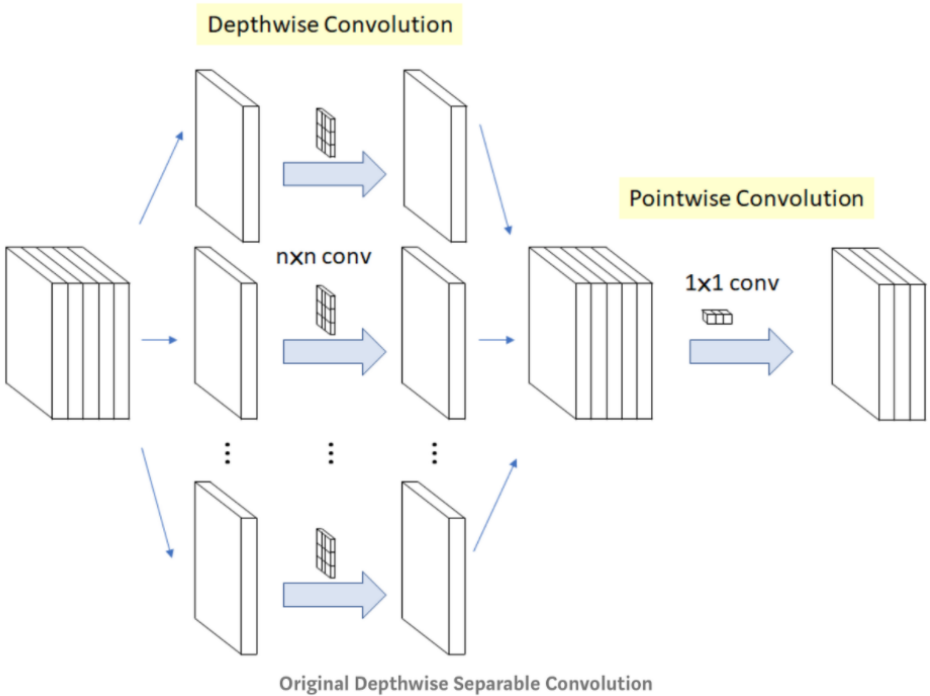
We argue that this optimization difficulty is *unlikely* to be caused by **vanishing gradients**. These plain networks are trained with BN [16], which ensures forward propagated signals to have non-zero variances. We also verify that the backward propagated gradients exhibit healthy norms with BN. So neither forward nor backward signals vanish. In



model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PRelu-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	<b>21.43</b>	<b>5.71</b>

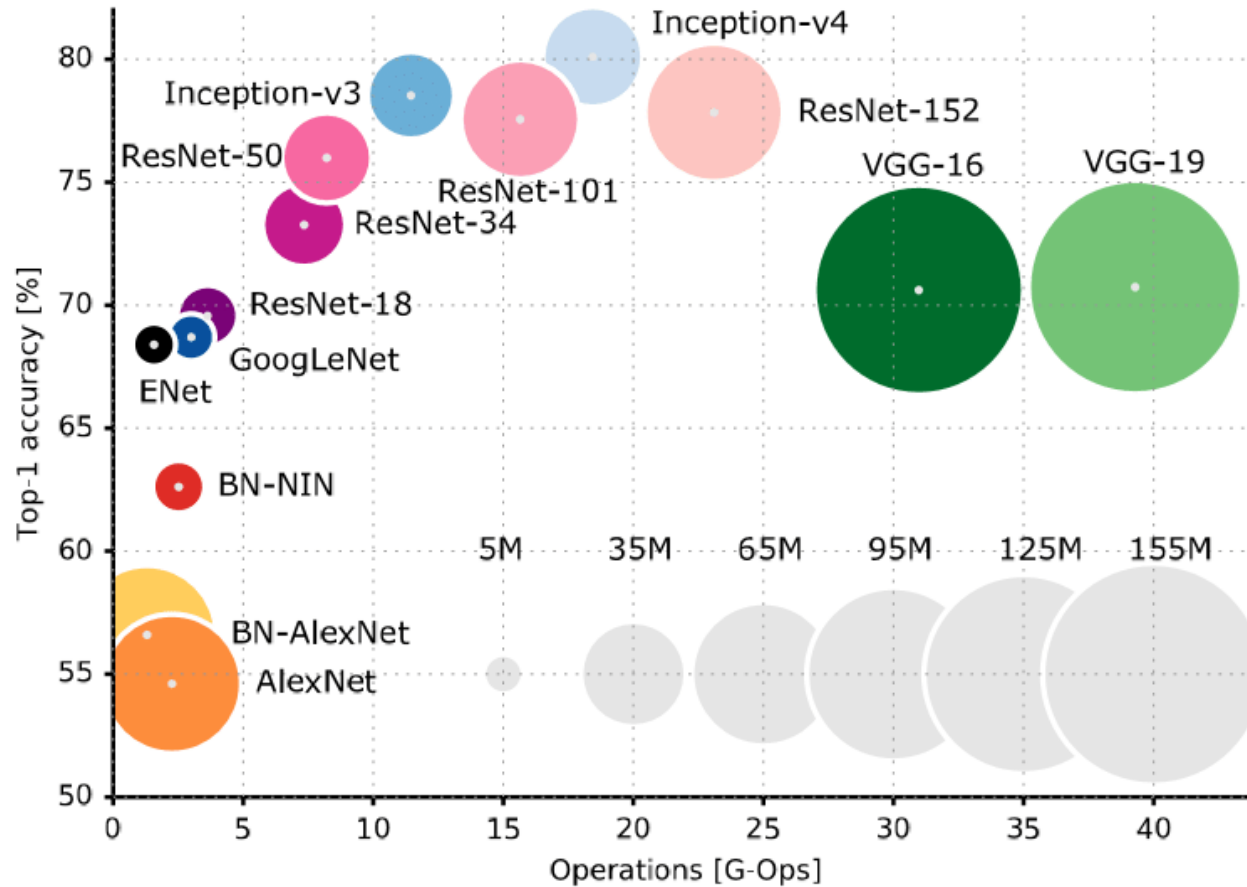


# MobileNet



Network	Top 1	Params	MAdds	CPU
MobileNetV1	70.6	4.2M	575M	113ms
ShuffleNet (1.5)	71.5	<b>3.4M</b>	292M	-
ShuffleNet (x2)	73.7	5.4M	524M	-
NasNet-A	74.0	5.3M	564M	183ms
MobileNetV2	<b>72.0</b>	<b>3.4M</b>	<b>300M</b>	<b>75ms</b>
MobileNetV2 (1.4)	<b>74.7</b>	6.9M	585M	<b>143ms</b>

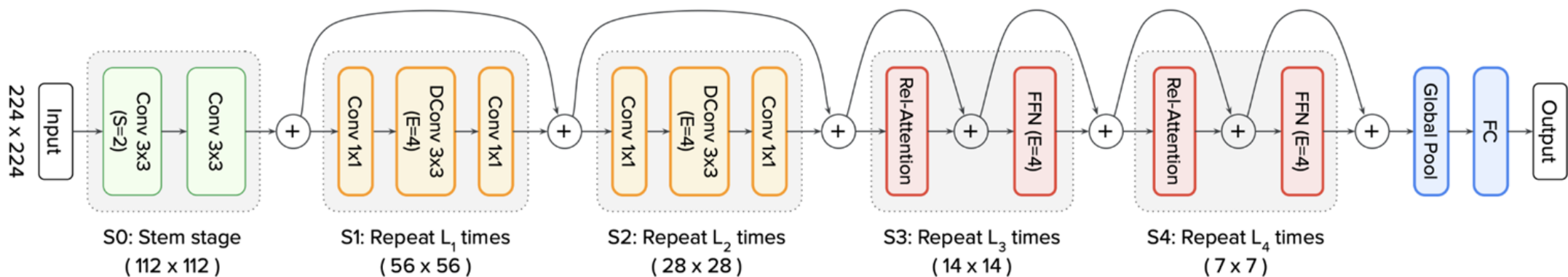
# Image Classification 결론



< 모델선택 >

- VGG16/19
- ResNet 50
- MobileNet

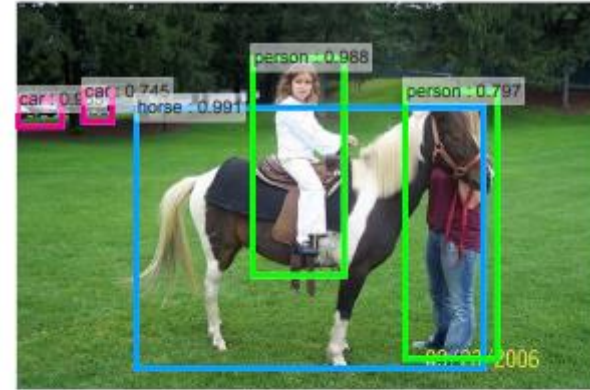
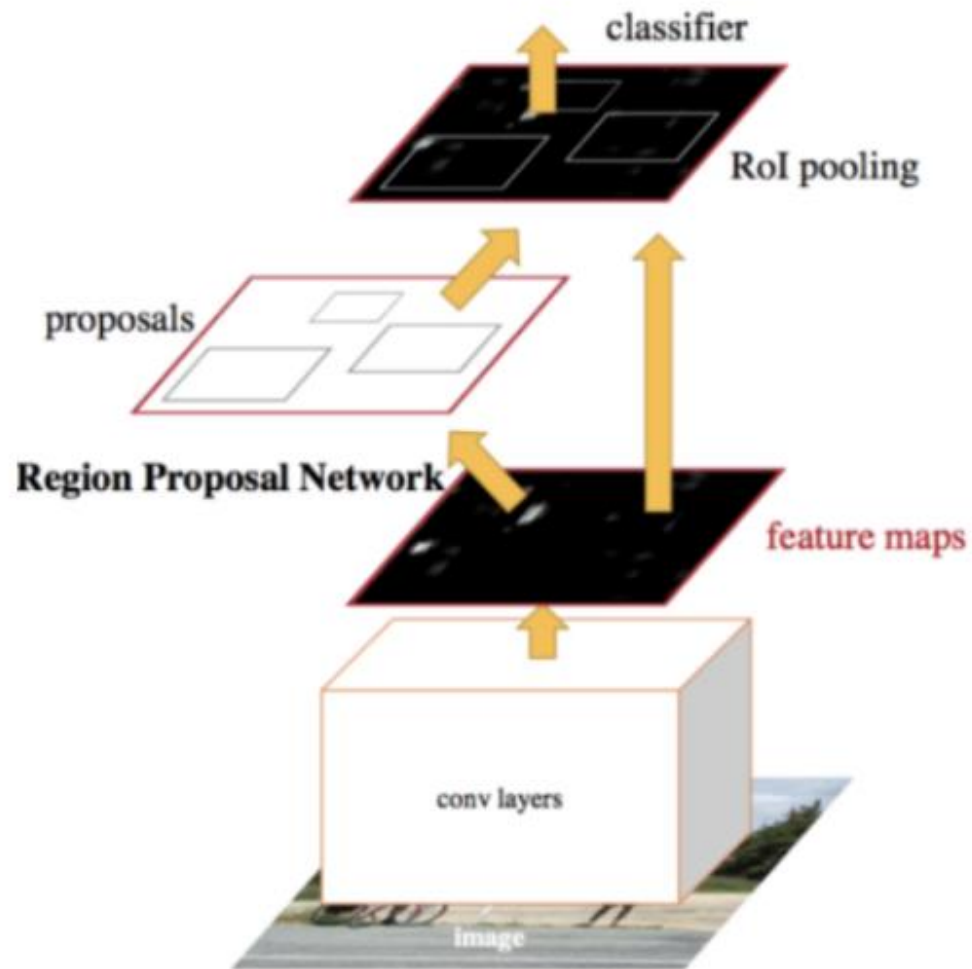
# CoAtNet (SOTA)



Models	Eval Size	#Params	#FLOPs	TPUv3-core-days	Top-1 Accuracy
ResNet + ViT-L/16	384 <sup>2</sup>	330M	-	-	87.12
ViT-L/16	512 <sup>2</sup>	307M	364B	0.68K	87.76
ViT-H/14	518 <sup>2</sup>	632M	1021B	2.5K	88.55
NFNet-F4+	512 <sup>2</sup>	527M	367B	1.86K	89.2
CoAtNet-3 <sup>†</sup>	384 <sup>2</sup>	168M	114B	0.58K	88.52
CoAtNet-3 <sup>†</sup>	512 <sup>2</sup>	168M	214B	0.58K	88.81
CoAtNet-4	512 <sup>2</sup>	275M	361B	0.95K	89.11
CoAtNet-5	512 <sup>2</sup>	688M	812B	1.82K	89.77
ViT-G/14	518 <sup>2</sup>	1.84B	5160B	>30K <sup>o</sup>	90.45
CoAtNet-6	512 <sup>2</sup>	1.47B	1521B	6.6K	90.45
CoAtNet-7	512 <sup>2</sup>	2.44B	2586B	20.1K	<b>90.88</b>

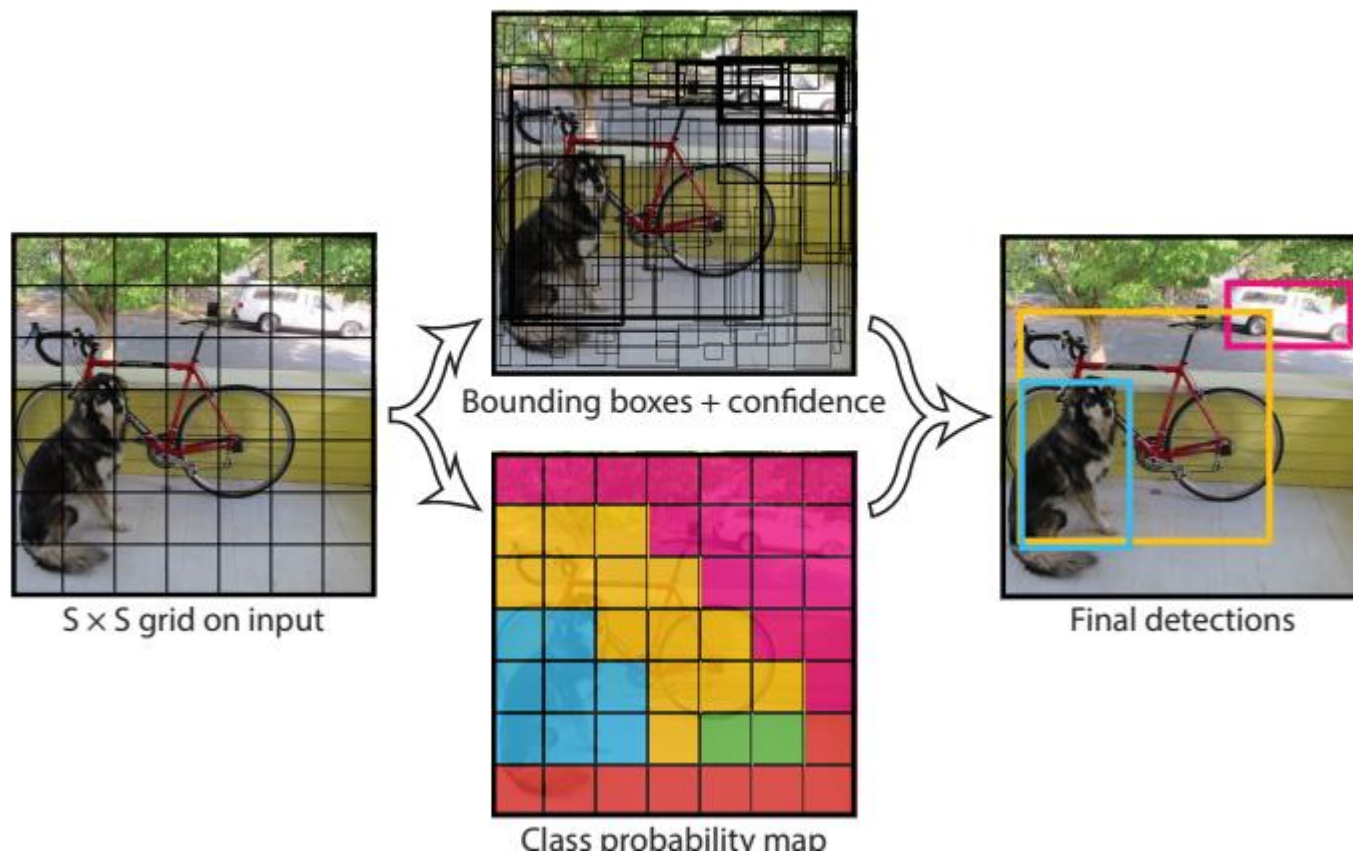
Too Heavy!

# Faster R-CNN



(CVPR, 2016) "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Network"

# YOLO v1



Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	<b>155</b>
YOLO	2007+2012	<b>63.4</b>	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21



# Object Detection 결론

VOC 2012 test	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
MR_CNN_MORE_DATA [11]	<b>73.9</b>	<b>85.5</b>	<b>82.9</b>	<b>76.6</b>	<b>57.8</b>	<b>62.7</b>	<b>79.4</b>	77.2	86.6	<b>55.0</b>	<b>79.1</b>	<b>62.2</b>	87.0	<b>83.4</b>	<b>84.7</b>	78.9	45.3	73.4	65.8	80.3	74.0
HyperNet_VGG	71.4	84.2	78.5	73.6	55.6	53.7	78.7	<b>79.8</b>	87.7	49.6	74.9	52.1	86.0	81.7	83.3	<b>81.8</b>	<b>48.6</b>	<b>73.5</b>	59.4	79.9	65.7
HyperNet_SP	71.3	84.1	78.3	73.3	55.5	53.6	78.6	79.6	87.5	49.5	74.9	52.1	85.6	81.6	83.2	81.6	48.4	73.2	59.3	79.7	65.6
<b>Fast R-CNN + YOLO</b>	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	<b>89.4</b>	49.4	75.5	57.0	<b>87.5</b>	80.9	81.0	74.7	41.8	71.5	68.5	<b>82.1</b>	67.2
MR_CNN_S_CNN [11]	70.7	85.0	79.6	71.5	55.3	57.7	76.0	73.9	84.6	50.5	74.3	61.7	85.5	79.9	81.7	76.4	41.0	69.0	61.2	77.7	72.1
Faster R-CNN [28]	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
DEEP_ENS_COCO	70.1	84.0	79.4	71.6	51.9	51.1	74.1	72.1	88.6	48.3	73.4	57.8	86.1	80.0	80.7	70.4	46.6	69.6	<b>68.8</b>	75.9	71.4
NoC [29]	68.8	82.8	79.0	71.6	52.3	53.7	74.1	69.0	84.9	46.9	74.3	53.1	85.0	81.3	79.5	72.2	38.9	72.4	59.5	76.7	68.1
Fast R-CNN [14]	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	<b>87.5</b>	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
UMICH_FGS_STRUCT	66.4	82.9	76.1	64.1	44.6	49.4	70.3	71.2	84.6	42.7	68.6	55.8	82.7	77.1	79.9	68.7	41.4	69.0	60.0	72.0	66.2
NUS_NIN_C2000 [7]	63.8	80.2	73.8	61.9	43.7	43.0	70.3	67.6	80.7	41.9	69.7	51.7	78.2	75.2	76.9	65.1	38.6	68.3	58.0	68.7	63.3
BabyLearning [7]	63.2	78.0	74.2	61.3	45.7	42.7	68.2	66.8	80.2	40.6	70.0	49.8	79.0	74.5	77.9	64.0	35.3	67.9	55.7	68.7	62.6
NUS_NIN	62.4	77.9	73.1	62.6	39.5	43.3	69.1	66.4	78.9	39.1	68.1	50.0	77.2	71.3	76.1	64.7	38.4	66.9	56.2	66.9	62.7
R-CNN VGG BB [13]	62.4	79.6	72.7	61.9	41.2	41.9	65.9	66.4	84.6	38.5	67.2	46.7	82.0	74.8	76.0	65.2	35.6	65.4	54.2	67.4	60.3
R-CNN VGG [13]	59.2	76.8	70.9	56.6	37.5	36.9	62.9	63.6	81.1	35.7	64.3	43.9	80.4	71.6	74.0	60.0	30.8	63.4	52.0	63.5	58.7
<b>YOLO</b>	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
Feature Edit [33]	56.3	74.6	69.1	54.4	39.1	33.1	65.2	62.7	69.7	30.8	56.0	44.6	70.0	64.4	71.1	60.2	33.3	61.3	46.4	61.7	57.8
R-CNN BB [13]	53.3	71.8	65.8	52.0	34.1	32.6	59.6	60.0	69.8	27.6	52.0	41.7	69.6	61.3	68.3	57.8	29.6	57.8	40.9	59.3	54.1
SDS [16]	50.7	69.7	58.4	48.5	28.3	28.8	61.3	57.5	70.8	24.1	50.7	35.9	64.9	59.1	65.8	57.1	26.0	58.8	38.6	58.9	50.7
R-CNN [13]	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9	26.5	48.7	39.5	66.2	57.3	65.4	53.2	26.2	54.5	38.1	50.6	51.6

< 모델 선정 >

- Faster R-CNN
- YOLO
- Faster R-CNN + YOLO

# 역할 분담 & 프로젝트 일정

## 게임환경

김동한

게임서버 구축

최형규

게임 프론트엔드/ ui

## AI 구축

김영현

딥러닝

김동우

딥러닝 서버 개발

주차	목표	
4주차	발표 준비 및 proposal 작성	
5주차	proposal 마무리 + 스터디	
6주차	Figma 게임 UI/UX 프로토타이핑 + 스터디	AI 서버 설계 + 스터디
7주차	게임 서버 설계 및 구현	AI 서버 구현
8주차		
9주차	게임 프론트 구현	AI 구현
10주차		
11주차	게임 서버 프론트 연동 작업	AI 서버 작업 마무리, 게임 연동 준비 작업
12주차	게임 서버 & AI 서버 연동 작업 (프로젝트 완성 데드라인)	
13주차	VM 테스트 + 리팩토링	
14주차	리팩토링 + 디버깅	
15주차	최종 발표	



	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13
제안서 작성 및 스터디									
게임 서버 구현									
게임 프론트 구현									
AI 서버 구현									
AI 작업									
리팩토링 및 구현 마무리, 디버깅									

**감사합니다**