# AI를 활용한 화재탐지 시스템 개발

김상현 김초명 노지윤 안세현 이종희

# 팀 소개

김초명

chomyung8912@gmail.com
https://github.com/MonicaKim89

이종희

www.youtube.com/channel
/UC7p5ZRmaVuYq5p8UYD5SppQ
https://github.com/K-107

김상현

hanseo3939@gmail.com
https://github.com/lenka88

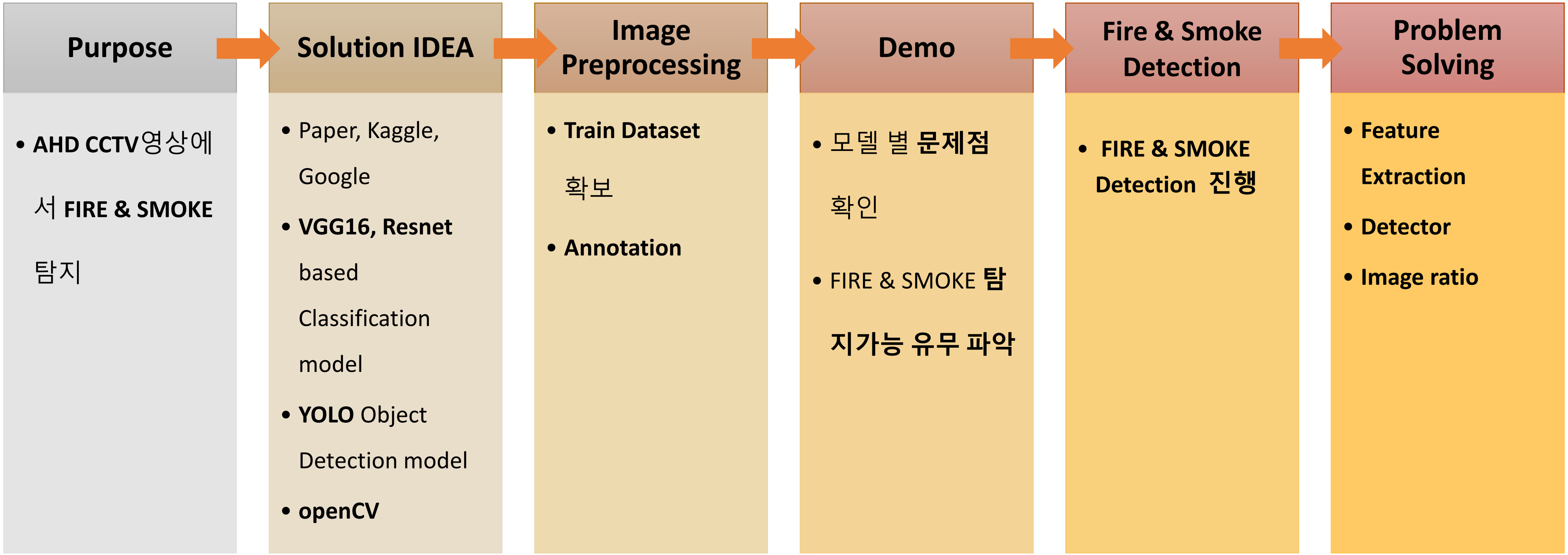
안세현

copigletan@naver.com
https://github.com/copiglet

노지윤

neutral.sth@gmail.com
https://github.com/nojiyoon

# Process & Content



**Purpose**

- **AHD CCTV**영상에서 **FIRE & SMOKE** 탐지

**Solution IDEA**

- Paper, Kaggle, Google
- **VGG16, Resnet** based Classification model
- **YOLO** Object Detection model
- **openCV**

**Image Preprocessing**

- **Train Dataset** 확보
- **Annotation**

**Demo**

- 모델 별 **문제점** 확인
- FIRE & SMOKE **탐지가능 유무 파악**

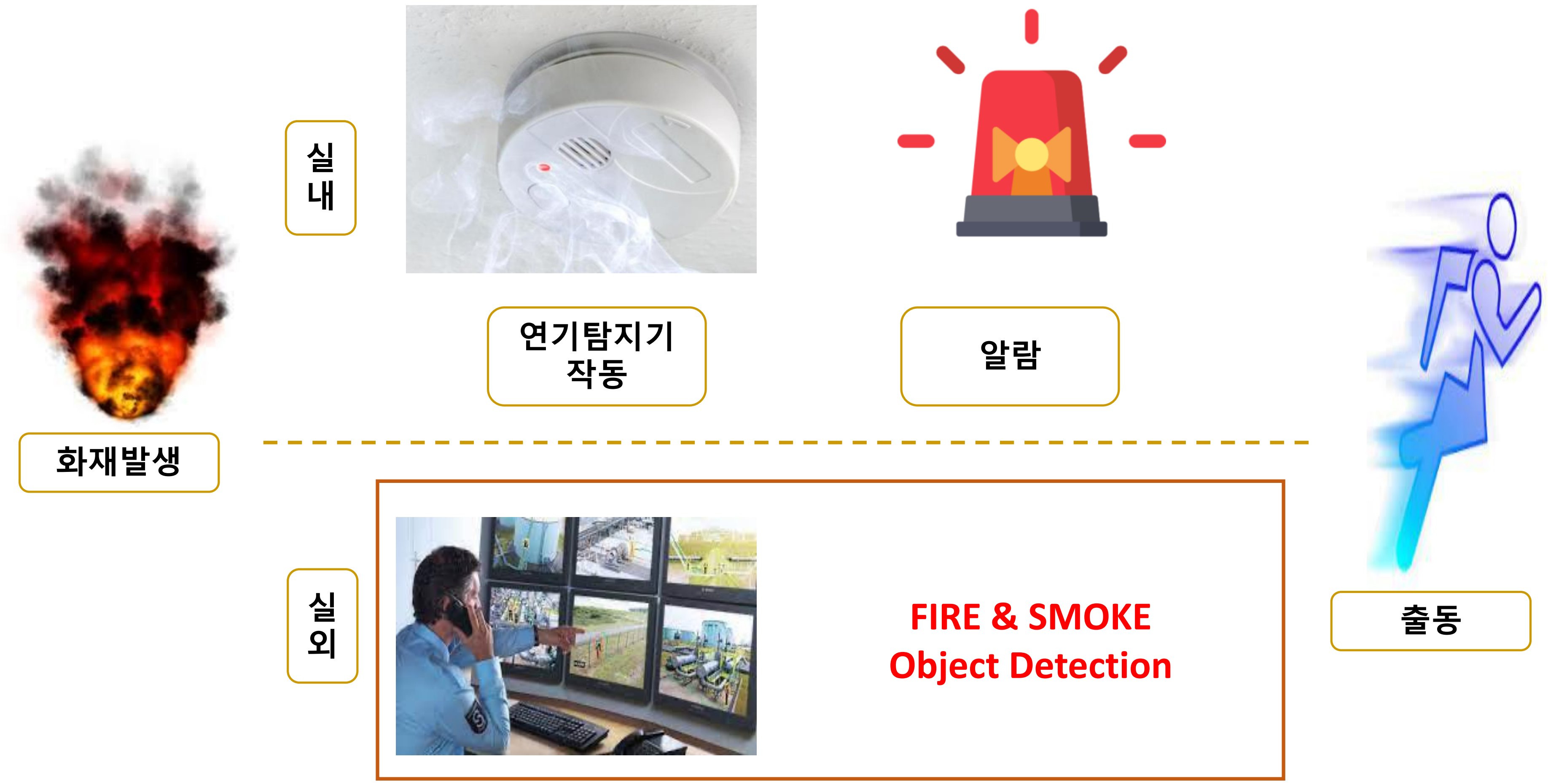
**Fire & Smoke Detection**

- **FIRE & SMOKE Detection 진행**

**Problem Solving**

- **Feature Extraction**
- **Detector**
- **Image ratio**

# Background: Fire detection



화재발생

실내

연기탐지기 작동

알람

실외

FIRE & SMOKE Object Detection

출동

실내의 경우 화재탐지를 위한 '연기탐지기'설치 되어있음

실외에는 연기탐지기 설치 어려움, CCTV영상처리를 통해 FIRE & SMOKE 객체 검출 필요

실외에서의 연기는 화재발생을 뜻함, 화재와 연기가 동시에 발생하지 않을 경우를 대비하여 각각의 학습 필요

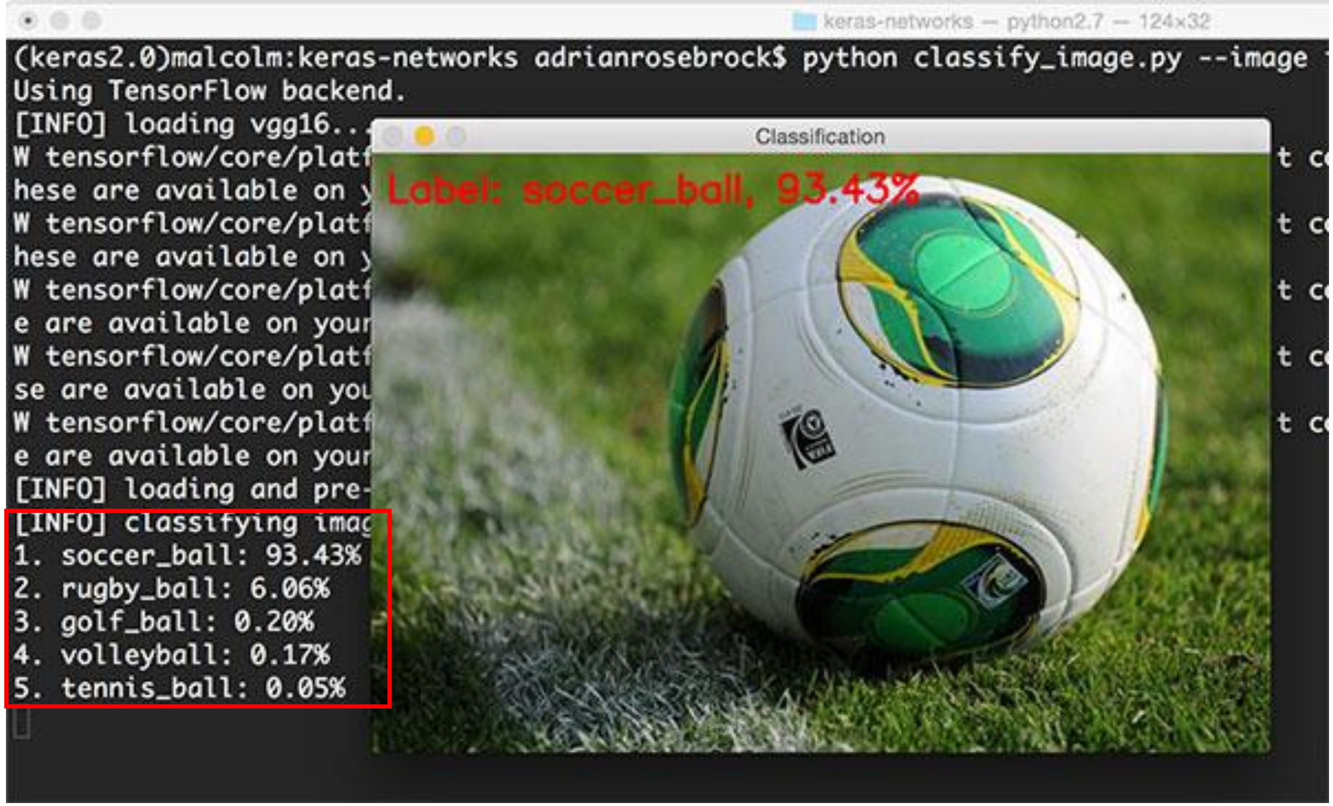
# IDEA for FIRE & SMOKE detection

| | | |
|---|---|---|
| K | Fire – 1000 images<br>Smoke – 1000 images<br>Neutral – 1000 images | **VGG16** based classification model<br>**Resnet** based classification model |
| YOLO | FIRE & SMOKE<br>1000 images | **YOLOv3 – Basic object detection method** |
| | | **YOLOv5 – Latest object detection method** |
| OpenCV | Fire video | **HSV converting(cv2.COLOR_BGR2HSV)**<br>**Fire color extraction (upper_lower color threshold)** |
| | Fire – 200 images<br>(open source) | **Deep Neural Network** |

1) **Classification : AHD 화면을 "화재발생", "연기발생"으로 구분할 수 있음**
2) **YOLOv3 : Fire detection paper에 우수한 성능을 보임, YOLOv5 성능개선**
3) **openCV : (optional) low-powered computer 사용 시**

# 솔루션 소개

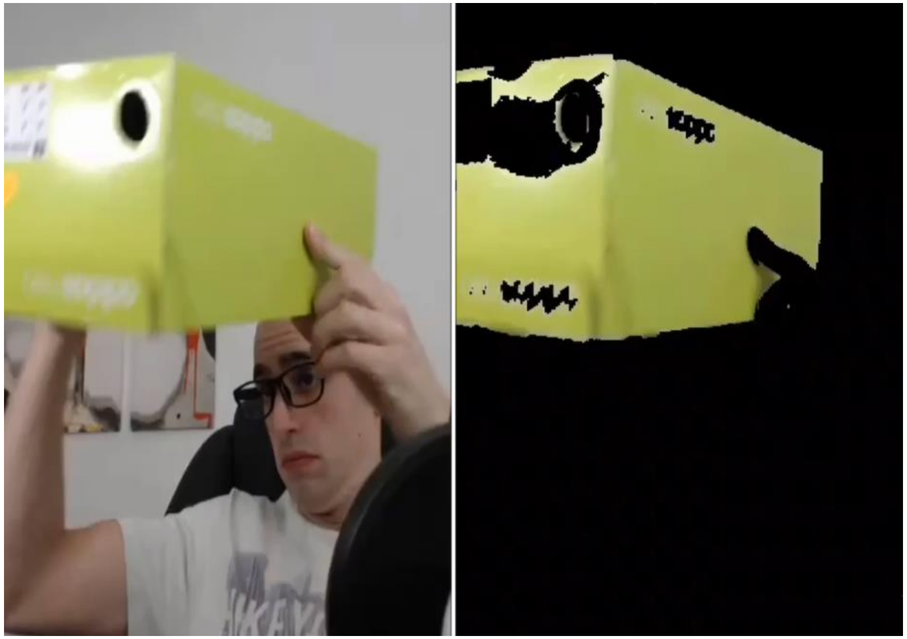
# VGG16, Resnet based Classifier

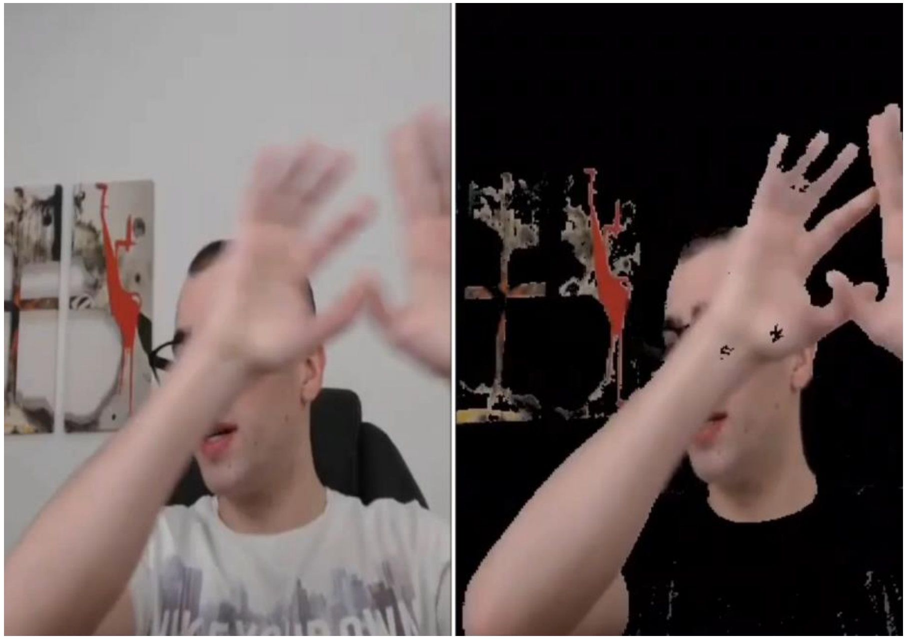

keras-networks — python2.7 — 124×32
Classification
Label: soccer_ball, 93.43%

```
(keras2.0)malcolm:keras-networks adrianrosebrock$ python classify_image.py --image
Using TensorFlow backend.
[INFO] loading vgg16...
W tensorflow/core/platf                                                  t c
hese are available on y
W tensorflow/core/platf                                                  t c
hese are available on y
W tensorflow/core/platf                                                  t c
e are available on your
W tensorflow/core/platf                                                  t c
se are available on you
W tensorflow/core/platf                                                  t c
e are available on your
[INFO] loading and pre-
[INFO] classifying imag
1. soccer_ball: 93.43%
2. rugby_ball: 6.06%
3. golf_ball: 0.20%
4. volleyball: 0.17%
5. tennis_ball: 0.05%
```

1) VGG16, Resnet은 화면에 나타난 객체를 분류하는데 사용됨
2) 영상의 각 프레임마다 객체를 분류함

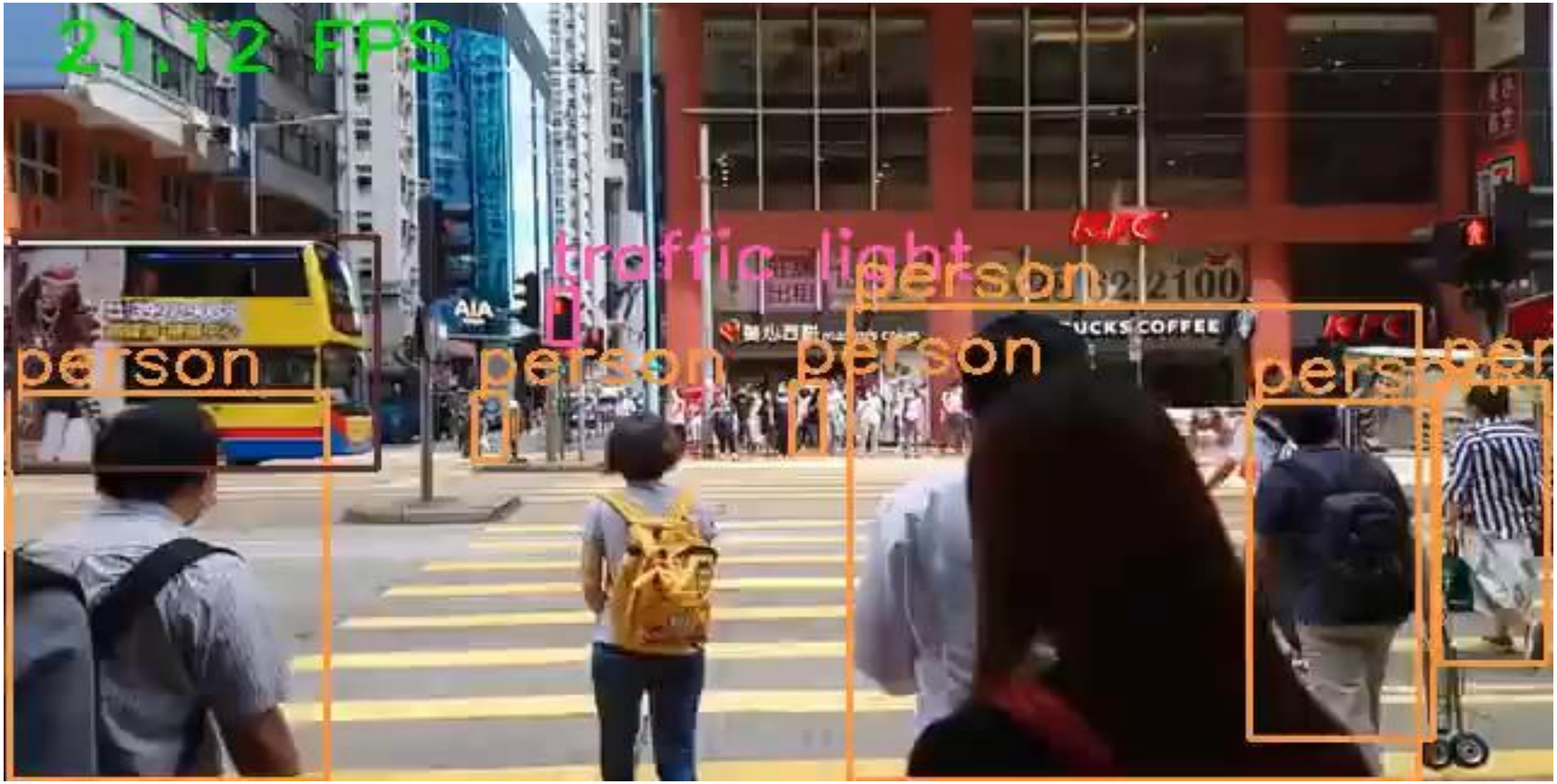
# openCV Color Extraction





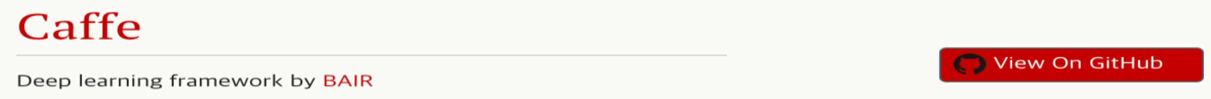
1) 영상 각 프레임을 HSV로 컨버팅 후, 추출하고 싶은 색의 임계치를 설정
2) 원하는 색상의 객체의 색상을 추출함

# openCV Deep Neural Net



1) OpenCV 에서도 TensorFlow, Caffee, PyTorch와 같은 딥러닝 프레임 워크에서 사전학습 된 모델을 로드 한 후 객체 탐지 가능

2) Caffe 딥러닝프레임워크사용

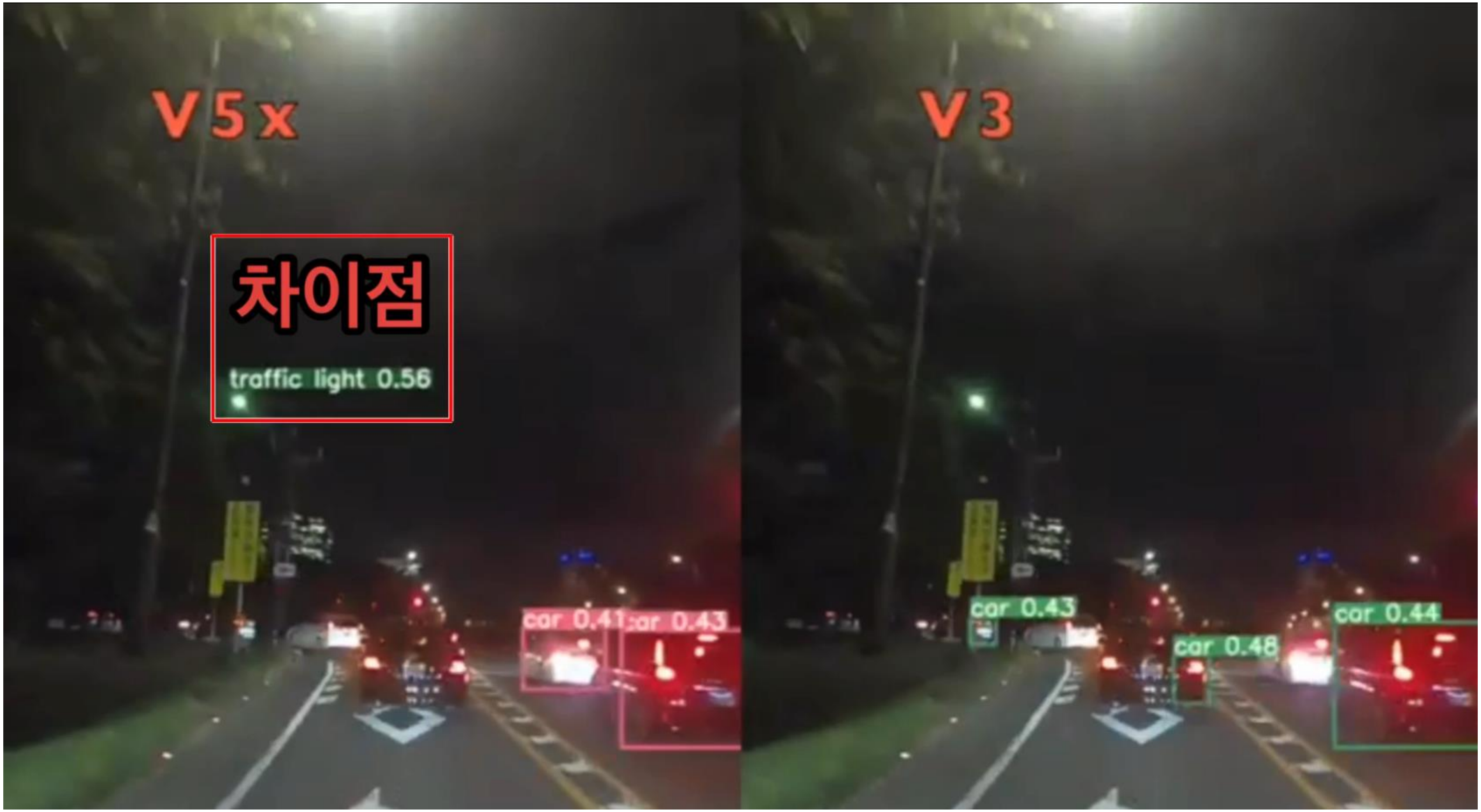


Caffe

Deep learning framework by BAIR

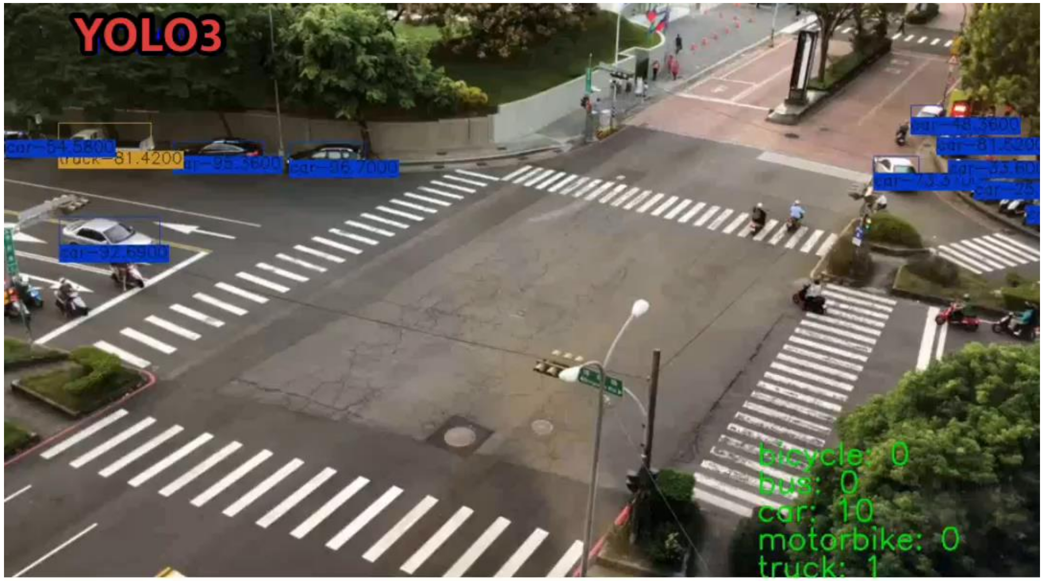
View On GitHub

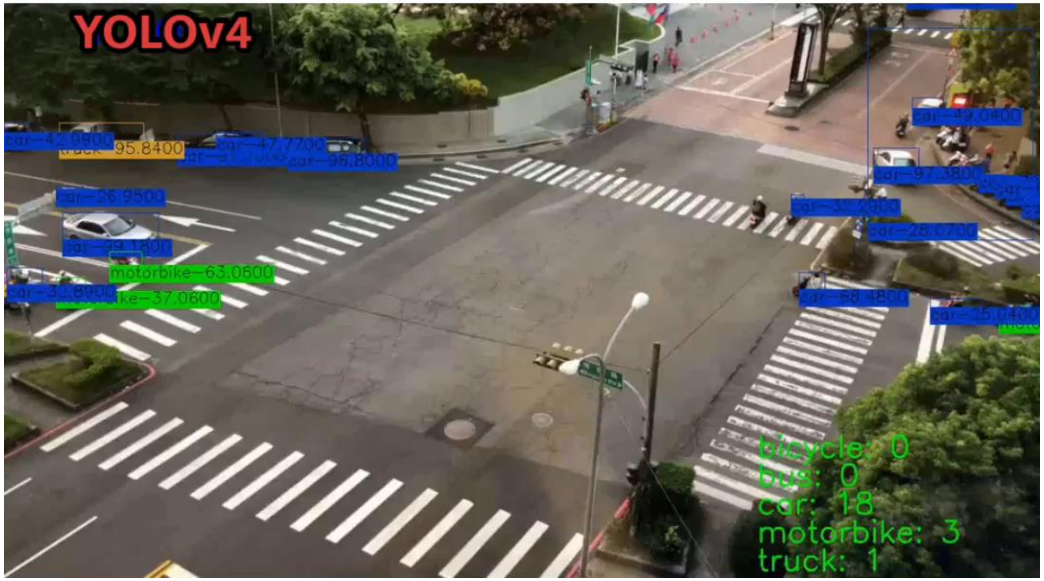
# Comparing YOLO v3, v5



차이점 : YOLOv3 작은 객체 탐지 불가 → YOLOv5탐지가능

# Comparing YOLO v3, v4

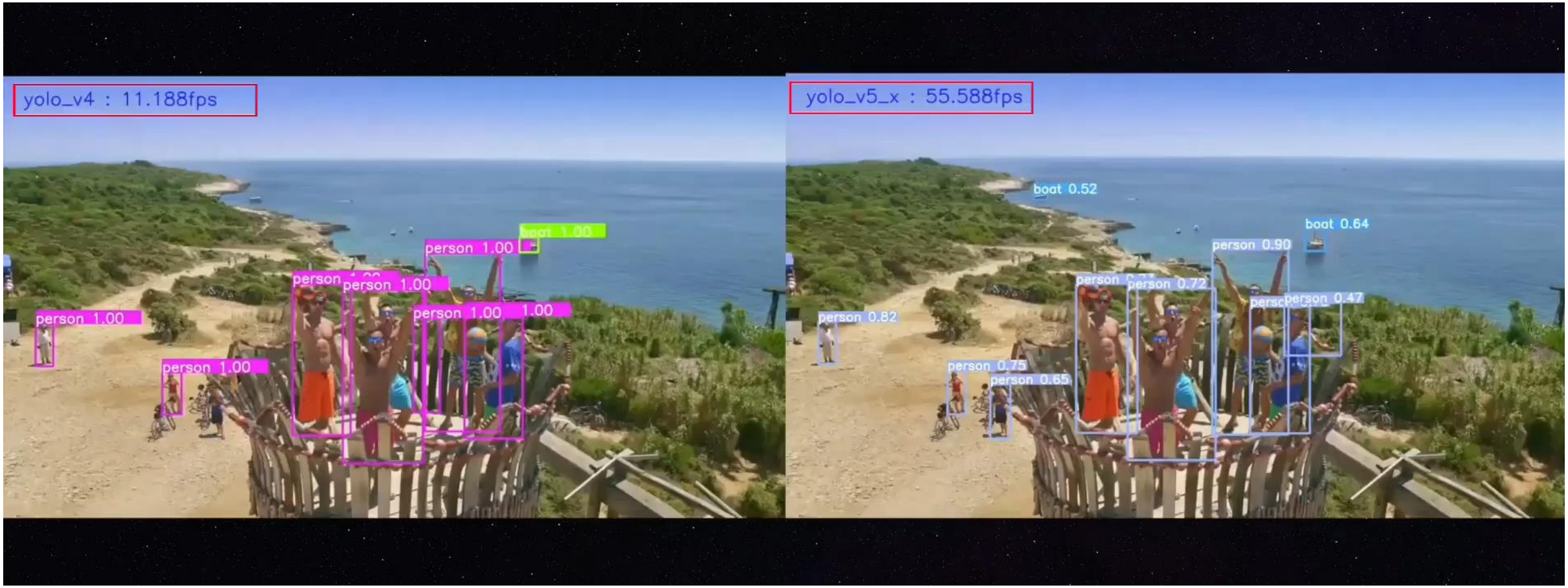




**차이점 : YOLOv4 > YOLOv3 탐지 객체 수**

**YOLOv3 작은 객체 탐지 불가**

# Comparing YOLO v4, v5



차이점 : YOLOv5 > YOLOv4 FPS 속도 높음

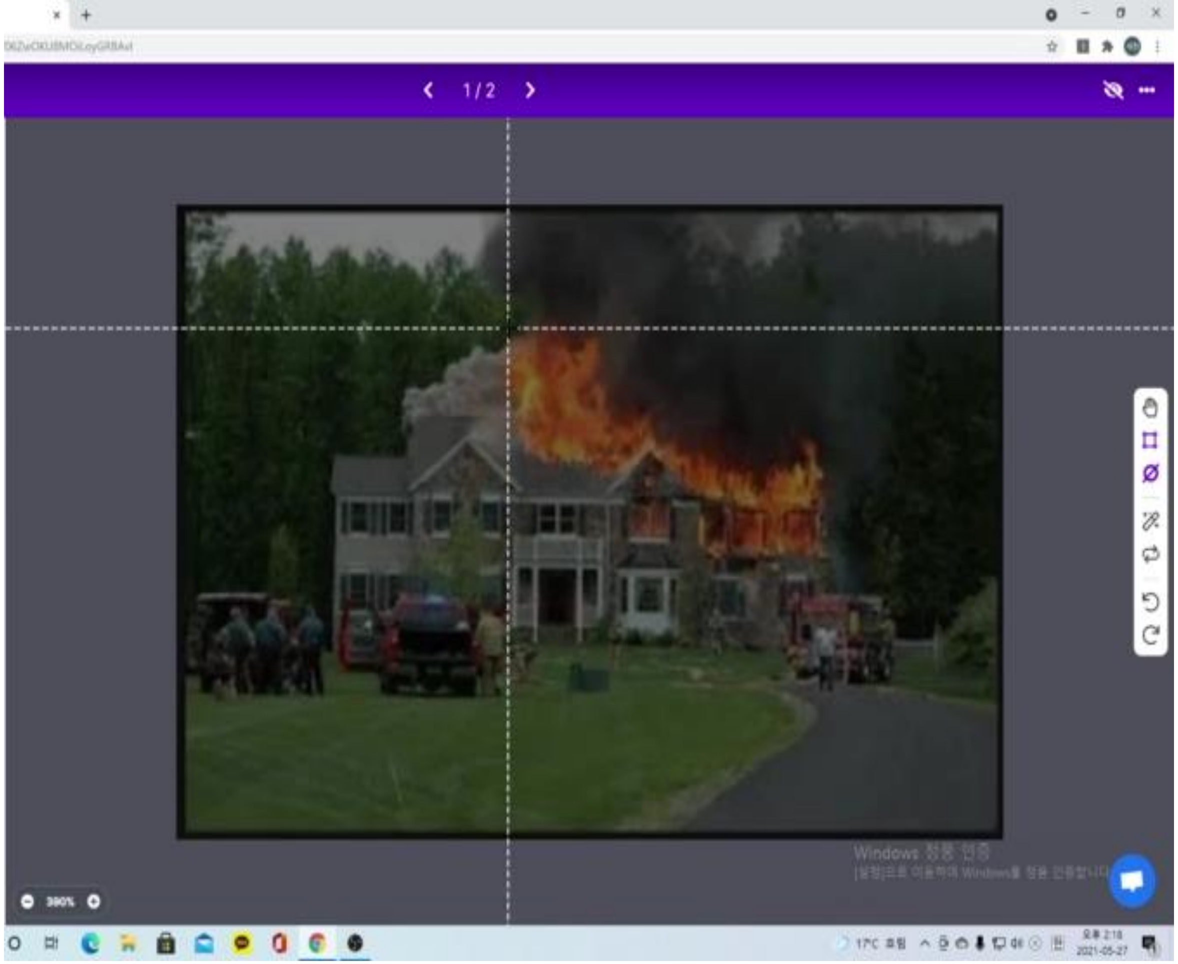
YOLOv5 > YOLOv4 탐지 객체 수

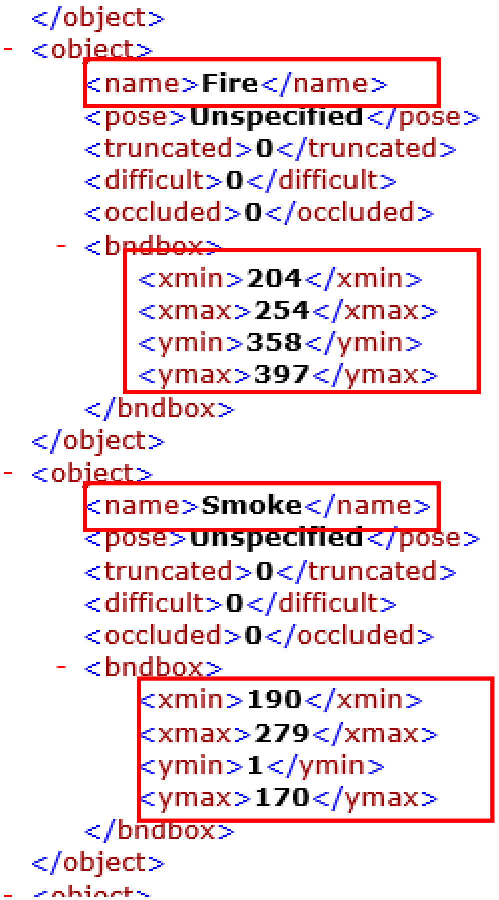
# 솔루션 시현

## Test video

CGTN
Speed 10X
Police tried putting out the fire, but it kept coming back.

# Image Preprocessing (Annotation)





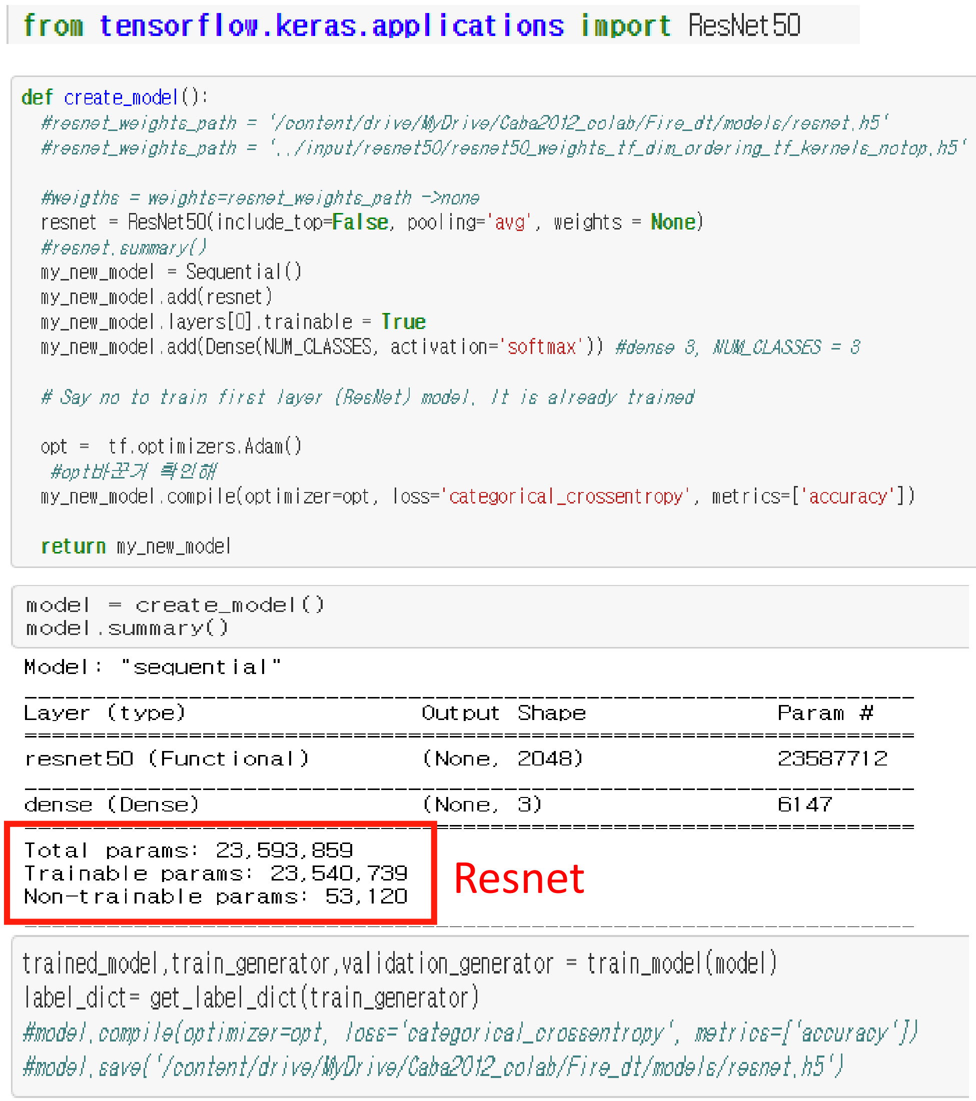
```
  </object>
- <object>
    <name>Fire</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
  - <bndbox>
      <xmin>204</xmin>
      <xmax>254</xmax>
      <ymin>358</ymin>
      <ymax>397</ymax>
    </bndbox>
  </object>
- <object>
    <name>Smoke</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
  - <bndbox>
      <xmin>190</xmin>
      <xmax>279</xmax>
      <ymin>1</ymin>
      <ymax>170</ymax>
    </bndbox>
  </object>
- <object>
```

.XML

1차 – Fire dataset (FIRE & SMOKE) 1000images

각 위치에 Bounding Box표시 후 좌표(x,y,w,h)지정 → Labeling(FIRE, SMOKE)

# FIRE & SMOKE detection using YOLO – RESNET vs VGG16



```
from tensorflow.keras.applications import ResNet50
```

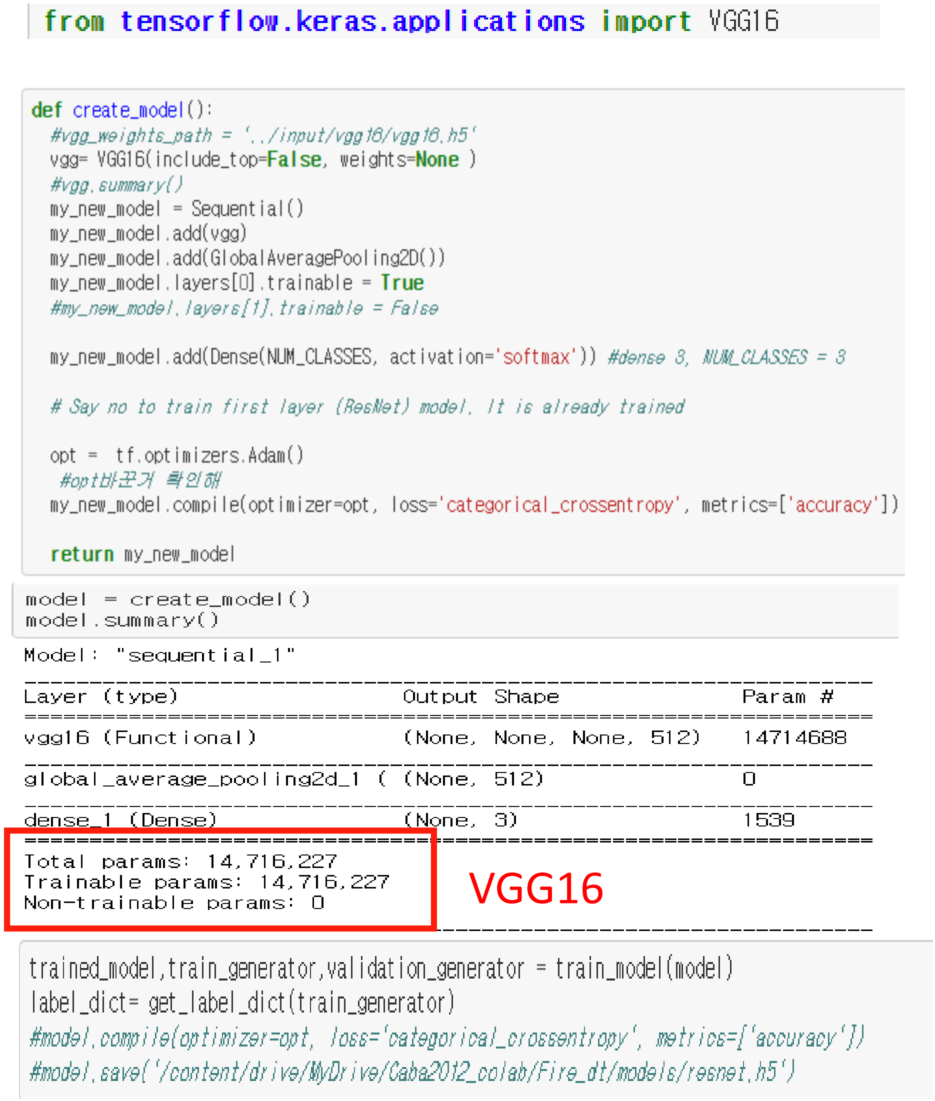
```
def create_model():
  #resnet_weights_path = '/content/drive/MyDrive/Caba2012_colab/Fire_dt/models/resnet.h5'
  #resnet_weights_path = '../input/resnet50/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h5'

  #weigths = weights=resnet_weights_path ->none
  resnet = ResNet50(include_top=False, pooling='avg', weights = None)
  #resnet.summary()
  my_new_model = Sequential()
  my_new_model.add(resnet)
  my_new_model.layers[0].trainable = True
  my_new_model.add(Dense(NUM_CLASSES, activation='softmax')) #dense 3, NUM_CLASSES = 3

  # Say no to train first layer (ResNet) model. It is already trained

  opt =  tf.optimizers.Adam()
   #opt바꾼거 확인해
  my_new_model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

  return my_new_model
```

```
model = create_model()
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 2048)              23587712
_________________________________________________________________
dense (Dense)                (None, 3)                 6147
=================================================================
Total params: 23,593,859
Trainable params: 23,540,739
Non-trainable params: 53,120
_________________________________________________________________
```

Resnet

```
trained_model,train_generator,validation_generator = train_model(model)
label_dict= get_label_dict(train_generator)
#model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
#model.save('/content/drive/MyDrive/Caba2012_colab/Fire_dt/models/resnet.h5')
```



```
from tensorflow.keras.applications import VGG16
```

```
def create_model():
  #vgg_weights_path = '../input/vgg16/vgg16.h5'
  vgg= VGG16(include_top=False, weights=None )
  #vgg.summary()
  my_new_model = Sequential()
  my_new_model.add(vgg)
  my_new_model.add(GlobalAveragePooling2D())
  my_new_model.layers[0].trainable = True
  #my_new_model.layers[1].trainable = False

  my_new_model.add(Dense(NUM_CLASSES, activation='softmax')) #dense 3, NUM_CLASSES = 3

  # Say no to train first layer (ResNet) model. It is already trained

  opt =  tf.optimizers.Adam()
   #opt바꾼거 확인해
  my_new_model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])

  return my_new_model
```

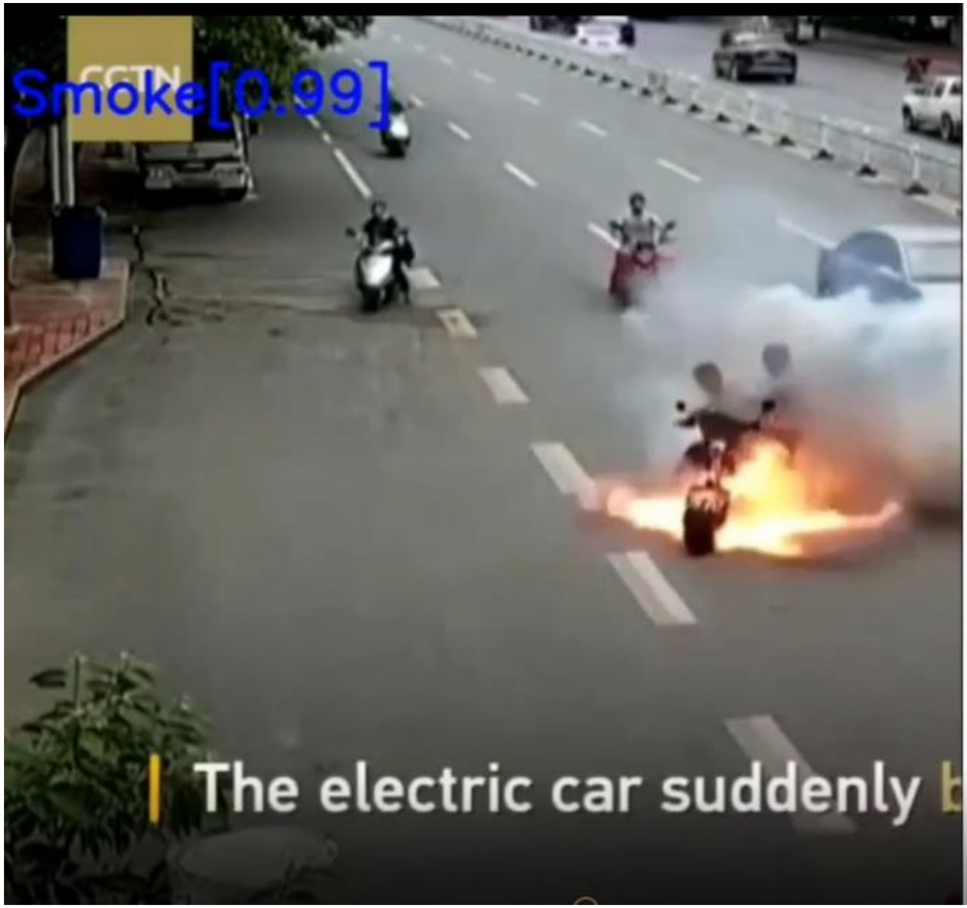
```
model = create_model()
model.summary()
```

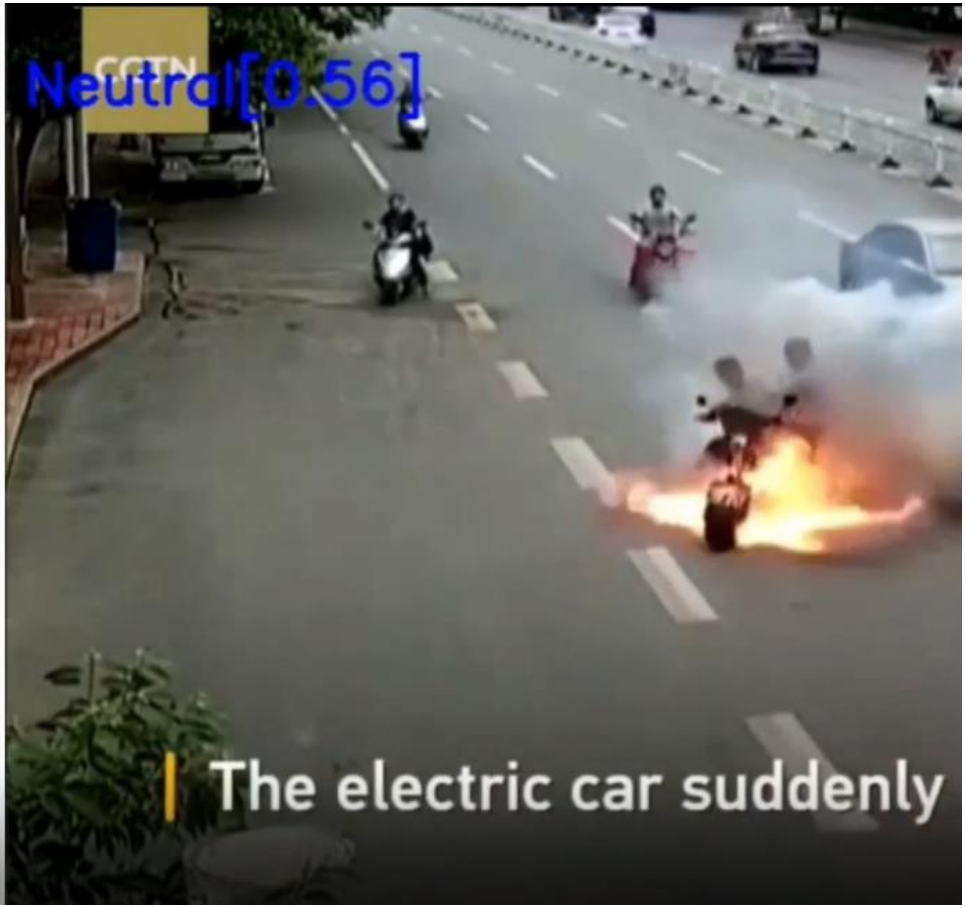
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, None, None, 512)   14714688
_________________________________________________________________
global_average_pooling2d_1 ( (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 1539
=================================================================
Total params: 14,716,227
Trainable params: 14,716,227
Non-trainable params: 0
_________________________________________________________________
```

VGG16

```
trained_model,train_generator,validation_generator = train_model(model)
label_dict= get_label_dict(train_generator)
#model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
#model.save('/content/drive/MyDrive/Caba2012_colab/Fire_dt/models/resnet.h5')
```

# FIRE & SMOKE detection using YOLO – RESNET vs VGG16

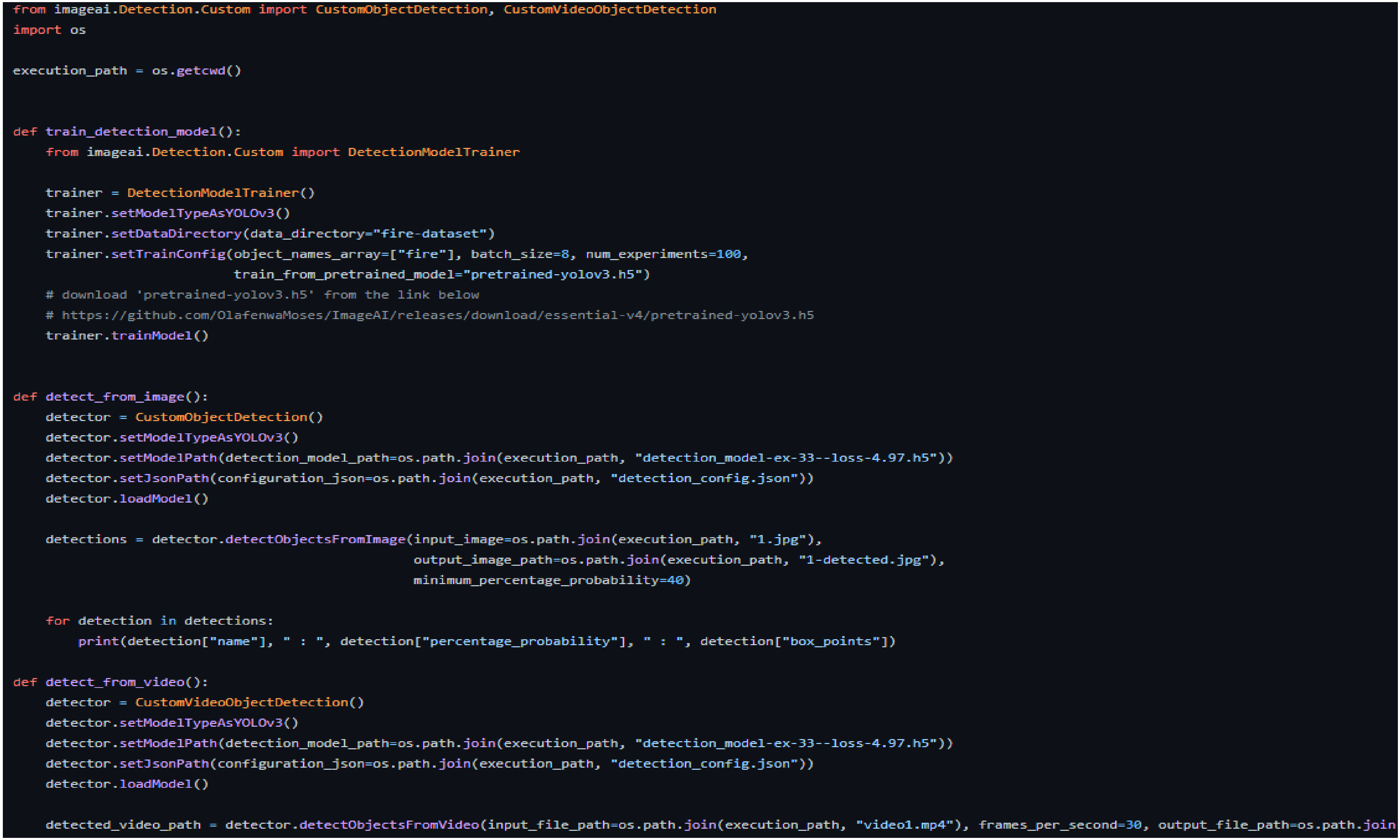




문제점: 1) Classification을 위한 방법

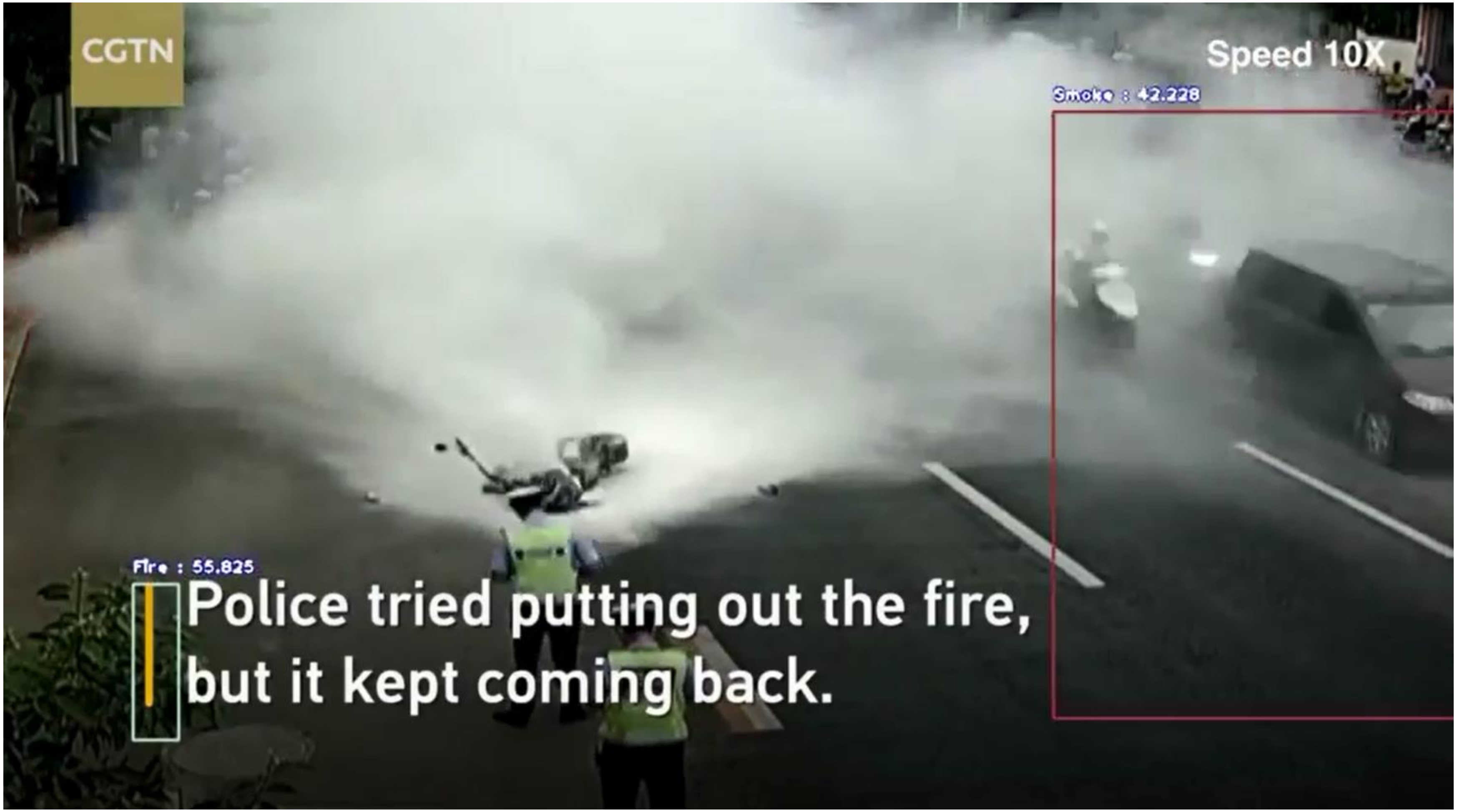
2) Fire detection 실패

# FIRE & SMOKE detection using YOLO – YOLOv3



```
from imageai.Detection.Custom import CustomObjectDetection, CustomVideoObjectDetection
import os

execution_path = os.getcwd()


def train_detection_model():
    from imageai.Detection.Custom import DetectionModelTrainer

    trainer = DetectionModelTrainer()
    trainer.setModelTypeAsYOLOv3()
    trainer.setDataDirectory(data_directory="fire-dataset")
    trainer.setTrainConfig(object_names_array=["fire"], batch_size=8, num_experiments=100,
                           train_from_pretrained_model="pretrained-yolov3.h5")
    # download 'pretrained-yolov3.h5' from the link below
    # https://github.com/OlafenwaMoses/ImageAI/releases/download/essential-v4/pretrained-yolov3.h5
    trainer.trainModel()


def detect_from_image():
    detector = CustomObjectDetection()
    detector.setModelTypeAsYOLOv3()
    detector.setModelPath(detection_model_path=os.path.join(execution_path, "detection_model-ex-33--loss-4.97.h5"))
    detector.setJsonPath(configuration_json=os.path.join(execution_path, "detection_config.json"))
    detector.loadModel()

    detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path, "1.jpg"),
                                                 output_image_path=os.path.join(execution_path, "1-detected.jpg"),
                                                 minimum_percentage_probability=40)

    for detection in detections:
        print(detection["name"], " : ", detection["percentage_probability"], " : ", detection["box_points"])

def detect_from_video():
    detector = CustomVideoObjectDetection()
    detector.setModelTypeAsYOLOv3()
    detector.setModelPath(detection_model_path=os.path.join(execution_path, "detection_model-ex-33--loss-4.97.h5"))
    detector.setJsonPath(configuration_json=os.path.join(execution_path, "detection_config.json"))
    detector.loadModel()

    detected_video_path = detector.detectObjectsFromVideo(input_file_path=os.path.join(execution_path, "video1.mp4"), frames_per_second=30, output_file_path=os.path.join
```

# FIRE & SMOKE detection using YOLO – YOLOv3

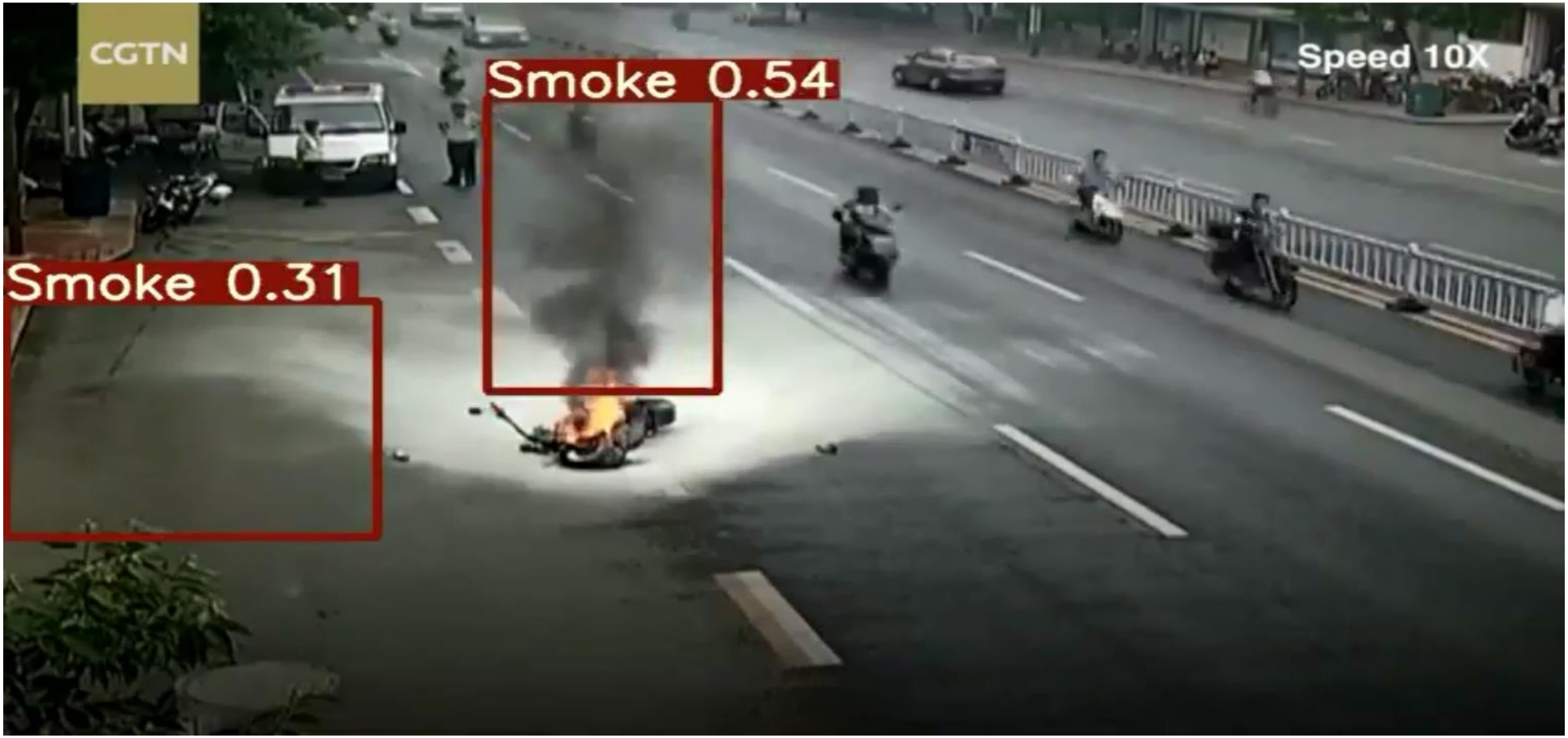


문제점: 1) 흰 연기 탐지 불가능 → 학습데이터부족

2) FIRE 색상 추정 객체 화재로 탐지

3) 작은 객체 탐지 불가능

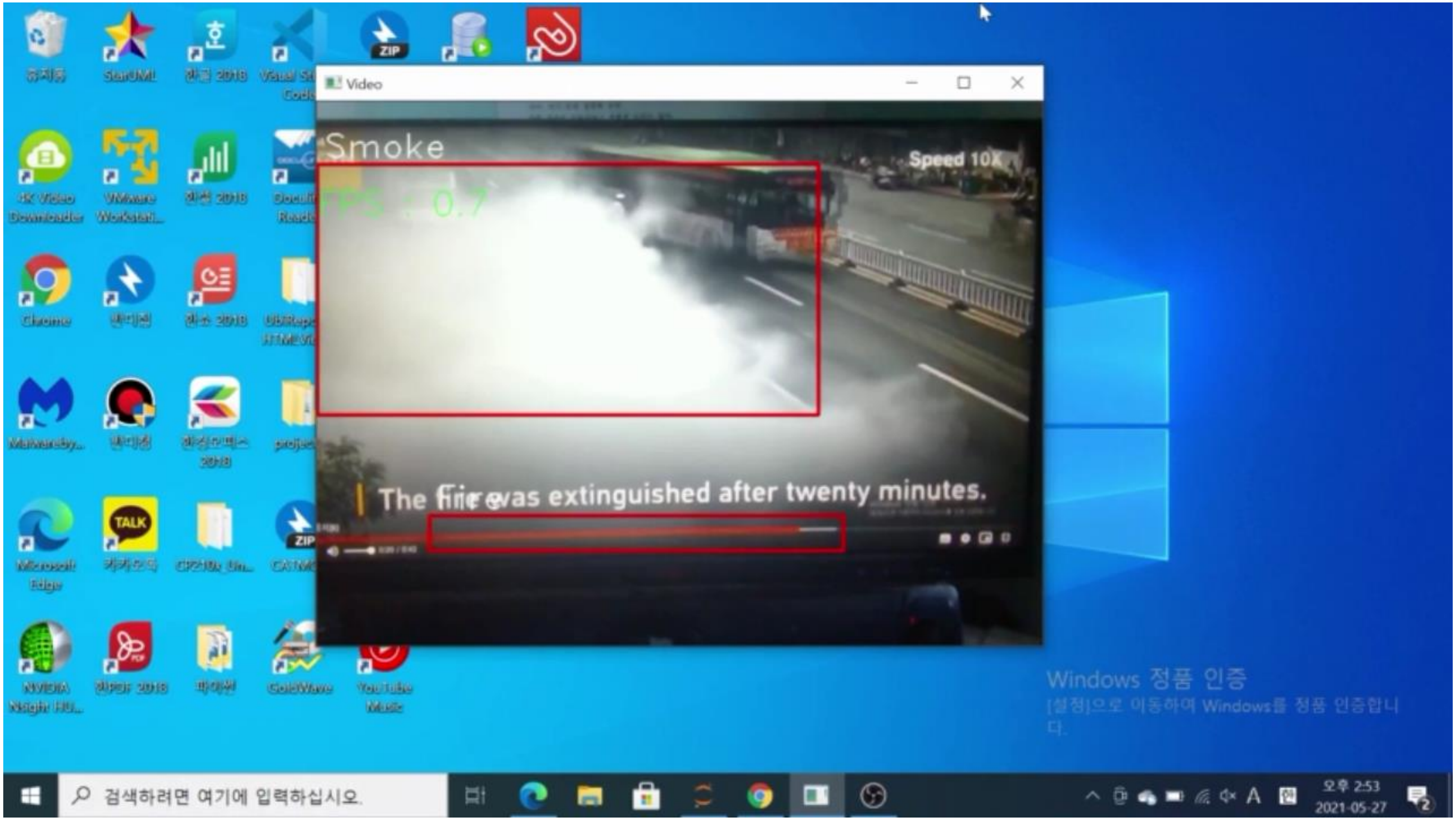
# FIRE & SMOKE detection using YOLO – YOLOv5



문제점: 1) 연기가 아닌 부분 → 연기로 탐지 : 탐지오류

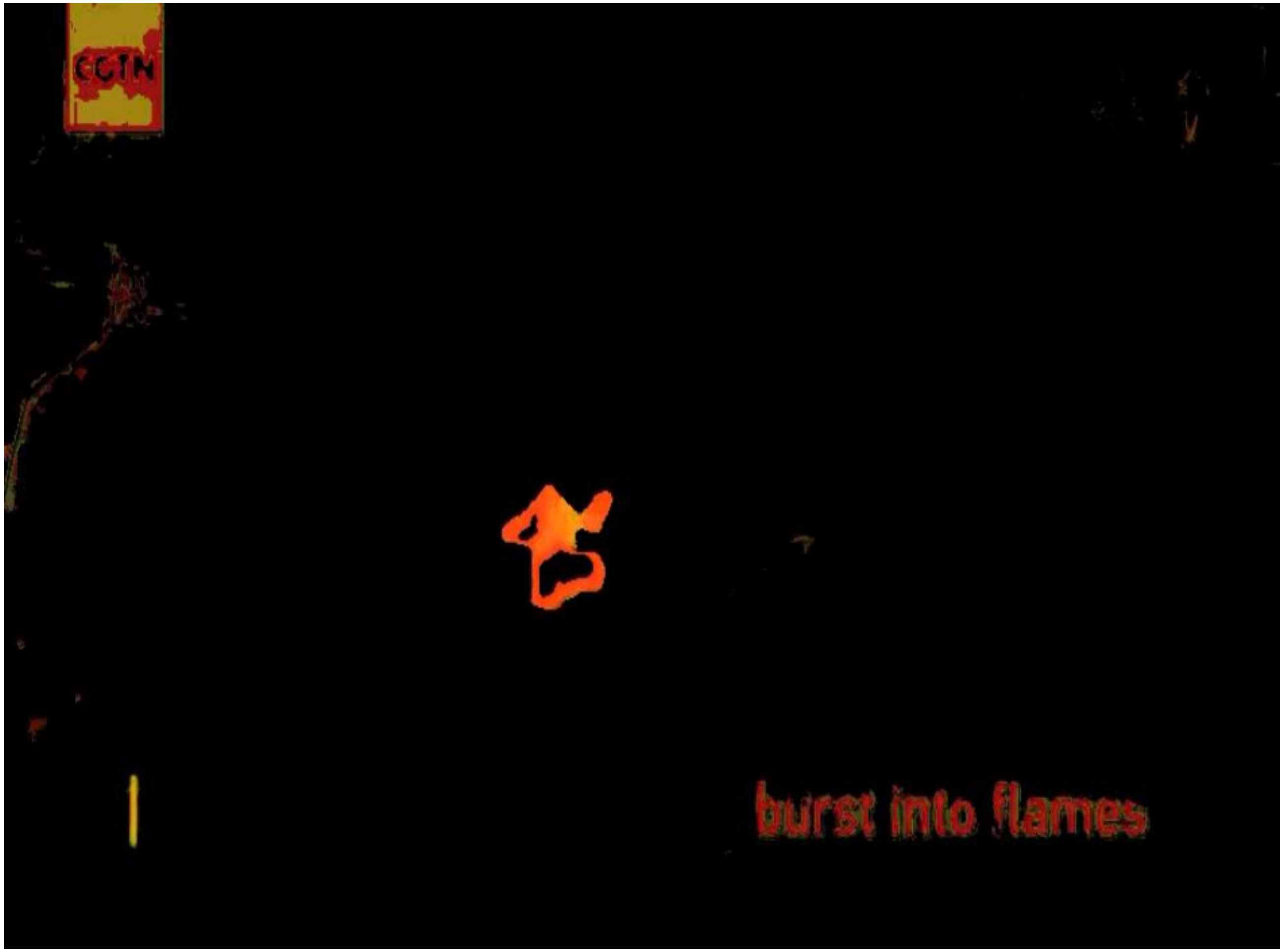
2) 객체 추출 성능 저하

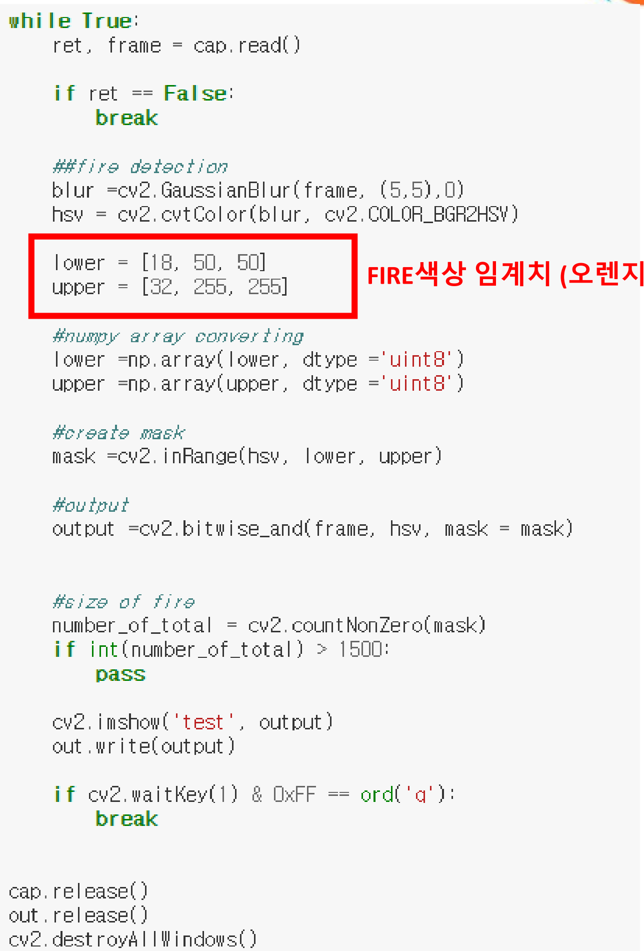
# Real time YOLOv3



1) YOLOv3 실시간 탐지 불가능

# FIRE detection using openCV – color extraction





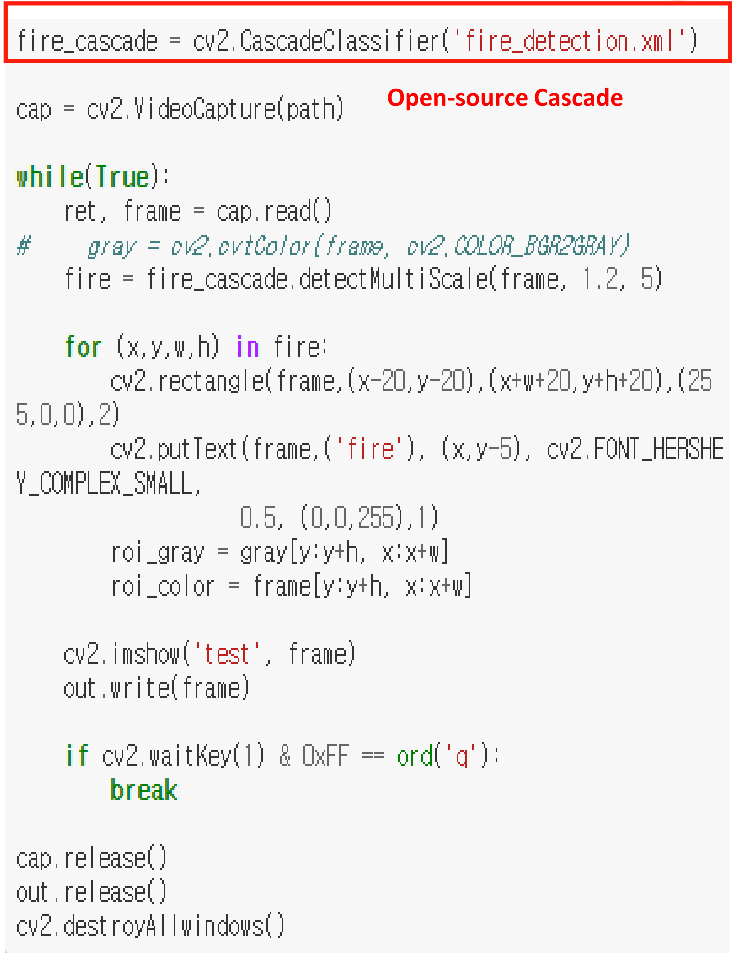
```
while True:
    ret, frame = cap.read()

    if ret == False:
        break

    ##fire detection
    blur =cv2.GaussianBlur(frame, (5,5),0)
    hsv = cv2.cvtColor(blur, cv2.COLOR_BGR2HSV)

    lower = [18, 50, 50]
    upper = [32, 255, 255]

    #numpy array converting
    lower =np.array(lower, dtype ='uint8')
    upper =np.array(upper, dtype ='uint8')

    #create mask
    mask =cv2.inRange(hsv, lower, upper)

    #output
    output =cv2.bitwise_and(frame, hsv, mask = mask)


    #size of fire
    number_of_total = cv2.countNonZero(mask)
    if int(number_of_total) > 1500:
        pass

    cv2.imshow('test', output)
    out.write(output)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break


cap.release()
out.release()
cv2.destroyAllWindows()
```

FIRE색상 임계치 (오렌지색)

**문제점: 1) 불과 비슷한 색상 모두 추출**

**2) 불은 주황색 외에도다양한 색으로 발화 가능**

# FIRE detection using openCV – fire XML



```
fire_cascade = cv2.CascadeClassifier('fire_detection.xml')

cap = cv2.VideoCapture(path)

while(True):
    ret, frame = cap.read()
#     gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    fire = fire_cascade.detectMultiScale(frame, 1.2, 5)

    for (x,y,w,h) in fire:
        cv2.rectangle(frame,(x-20,y-20),(x+w+20,y+h+20),(25
5,0,0),2)
        cv2.putText(frame,('fire'), (x,y-5), cv2.FONT_HERSHE
Y_COMPLEX_SMALL,
                    0.5, (0,0,255),1)
        roi_gray = gray[y:y+h, x:x+w]
        roi_color = frame[y:y+h, x:x+w]

    cv2.imshow('test', frame)
    out.write(frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
out.release()
cv2.destroyAllwindows()
```

Open-source Cascade

문제점: 1) 7-8초 잠시 탐지

2) 학습 데이터(images 200장) 부족으로 보임

# Comparing YOLO v3, v5 for FIRE & SMOKE detection model

| | 문제점 | 해결방안 |
|---|---|---|
| **VGG16, RESNET** | - Classification에 특화된 모델<br>- FIRE와 SMOKE 동시 발생 시 화면을 채우는 객체로 탐지 | - Object Detection으로는 사용 불가 |
| **openCV** | - FIRE의 색상으로 지정한 색상 임계치에 포함되는 모든 객체를 탐지<br>- FIRE 의 색상은 다양함<br>- 탐지 성능이 떨어짐 | - FIRE색상 임계치 변화<br>- 학습 이미지 추가<br>- Centroid 및 화재의 특성을 파악하고 적용 |
| **YOLOv3** | - **오렌지색상 객체를 화재로 탐지**<br>- **작은 객체 탐지불가 (YOLOv3특징)**<br>- **Real-Time 활용 불가 (YOLOv3 문제)** | - **학습 이미지 추가** (SMOKE 이미지 추가)<br>- **Feature map 강화** (이미지 특성 추출 강화)<br>- **Detector 성능 강화** (최적의 weight 및 모델 선택) |
| **YOLOv5** | - **연기가 아닌 부분을 연기로 탐지**<br>- **연기탐지성능저하**<br>- **후반부 화재탐지불가** | |

# 학습 이미지 추가

Fire Dataset
1000 images

**FIRE** >>> Smoke

Add Smoke Dataset
1000 images

Fire <<< **SMOKE**

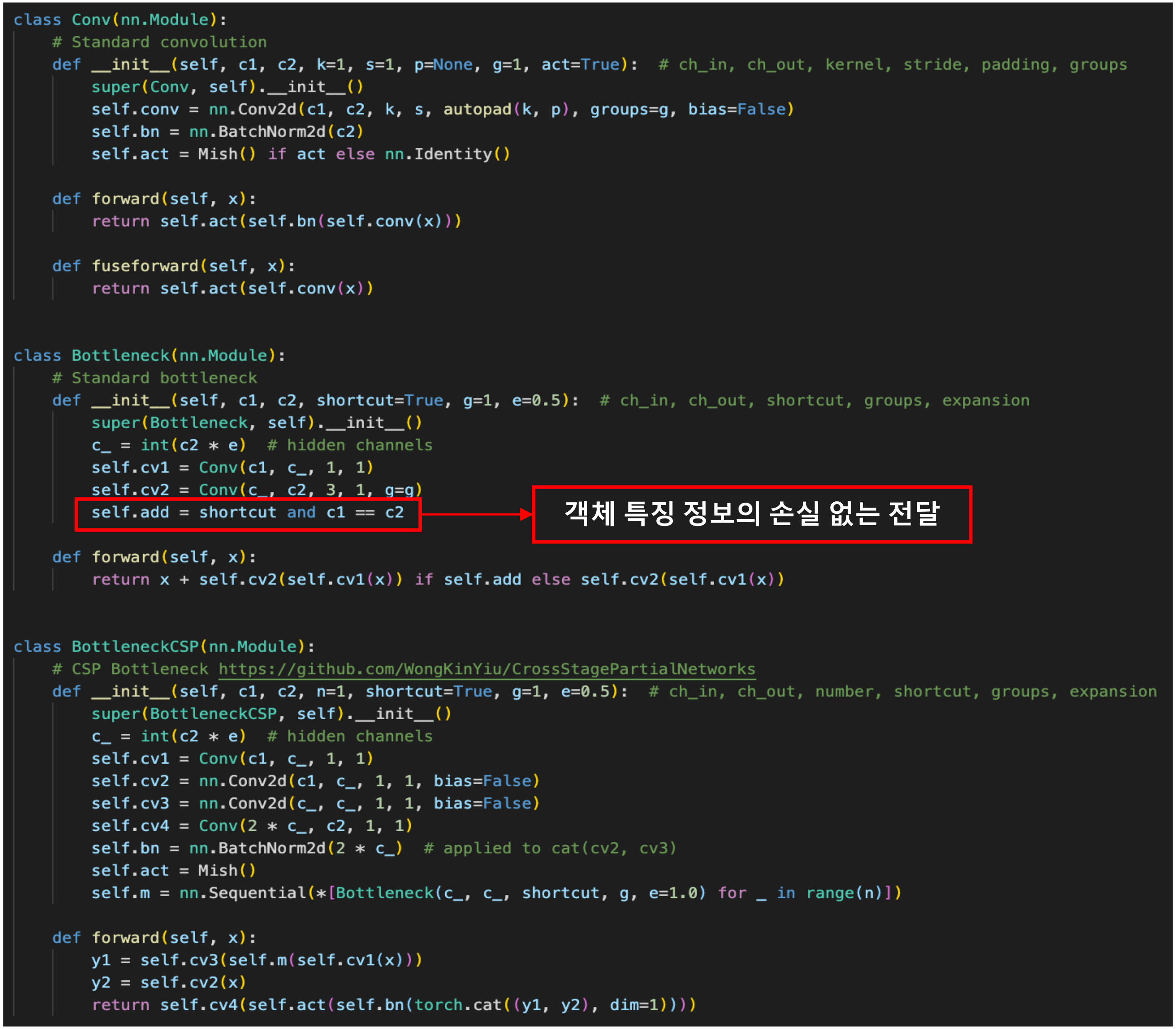
(문제) SMOKE 탐지 기능 저하

→ SMOKE 객체의 특징 학습↓

(해결) SMOKE 특징 학습

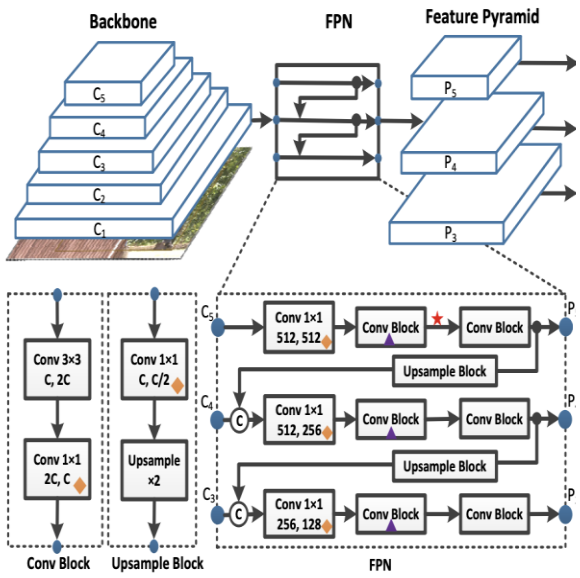
→ **SMOKE 데이터 1000장 추가**

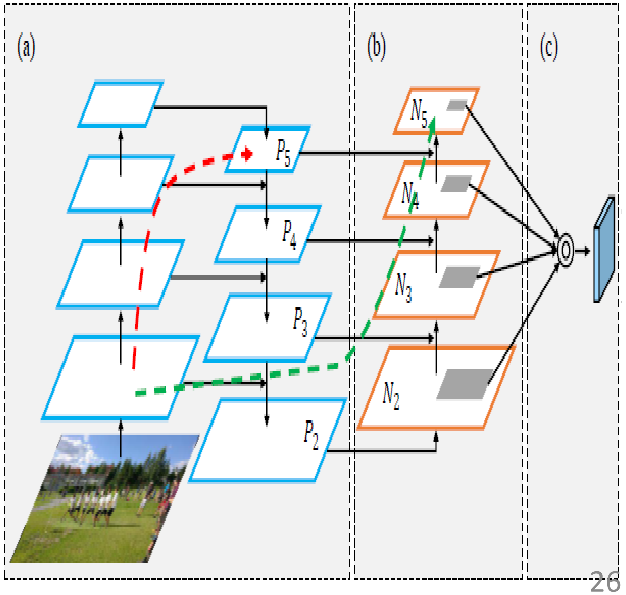
# Feature Map 강화



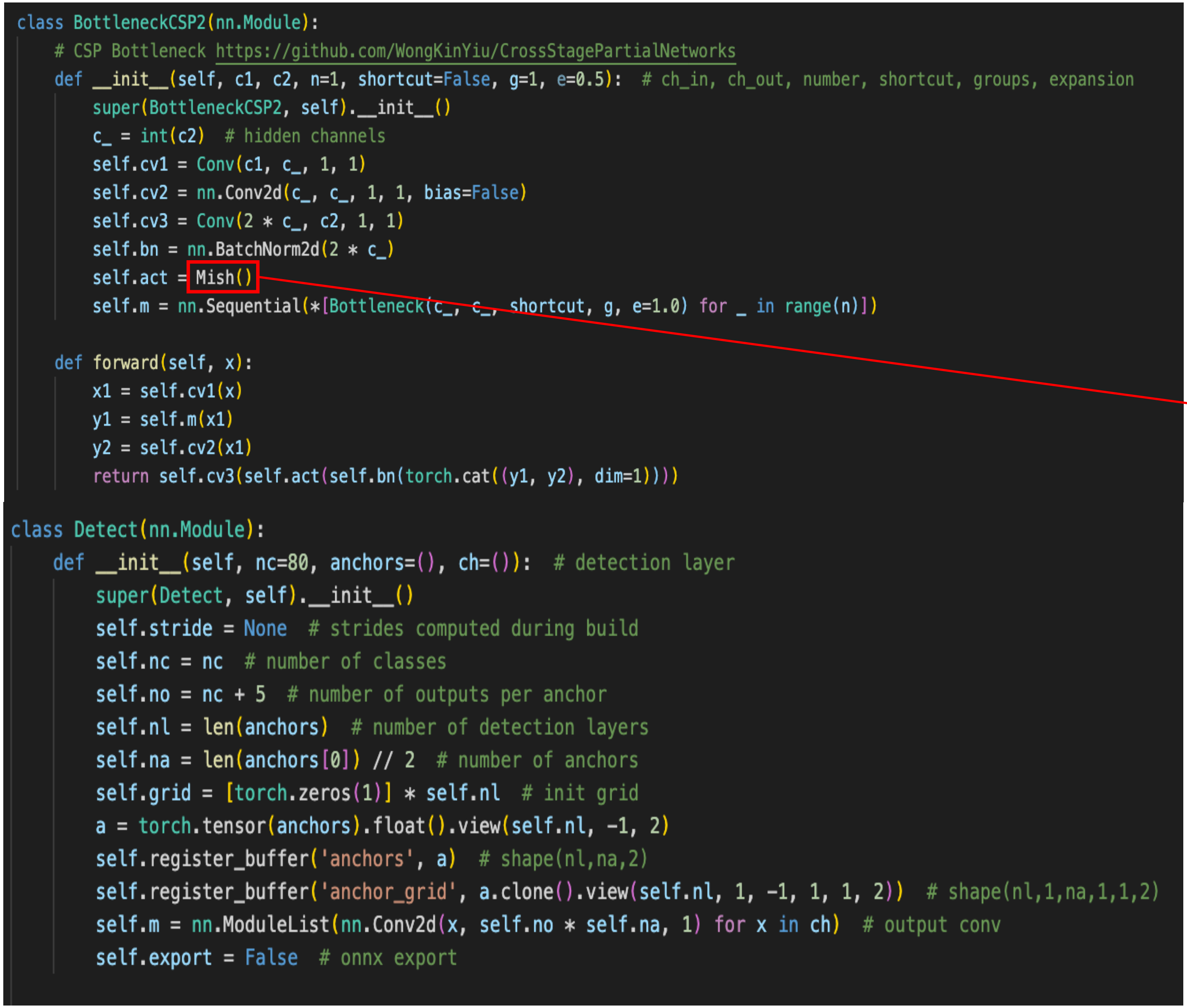
```
class Conv(nn.Module):
    # Standard convolution
    def __init__(self, c1, c2, k=1, s=1, p=None, g=1, act=True):  # ch_in, ch_out, kernel, stride, padding, groups
        super(Conv, self).__init__()
        self.conv = nn.Conv2d(c1, c2, k, s, autopad(k, p), groups=g, bias=False)
        self.bn = nn.BatchNorm2d(c2)
        self.act = Mish() if act else nn.Identity()

    def forward(self, x):
        return self.act(self.bn(self.conv(x)))

    def fuseforward(self, x):
        return self.act(self.conv(x))


class Bottleneck(nn.Module):
    # Standard bottleneck
    def __init__(self, c1, c2, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, shortcut, groups, expansion
        super(Bottleneck, self).__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = Conv(c_, c2, 3, 1, g=g)
        self.add = shortcut and c1 == c2

    def forward(self, x):
        return x + self.cv2(self.cv1(x)) if self.add else self.cv2(self.cv1(x))


class BottleneckCSP(nn.Module):
    # CSP Bottleneck https://github.com/WongKinYiu/CrossStagePartialNetworks
    def __init__(self, c1, c2, n=1, shortcut=True, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super(BottleneckCSP, self).__init__()
        c_ = int(c2 * e)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = nn.Conv2d(c1, c_, 1, 1, bias=False)
        self.cv3 = nn.Conv2d(c_, c_, 1, 1, bias=False)
        self.cv4 = Conv(2 * c_, c2, 1, 1)
        self.bn = nn.BatchNorm2d(2 * c_)  # applied to cat(cv2, cv3)
        self.act = Mish()
        self.m = nn.Sequential(*[Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)])

    def forward(self, x):
        y1 = self.cv3(self.m(self.cv1(x)))
        y2 = self.cv2(x)
        return self.cv4(self.act(self.bn(torch.cat((y1, y2), dim=1))))
```

객체 특징 정보의 손실 없는 전달



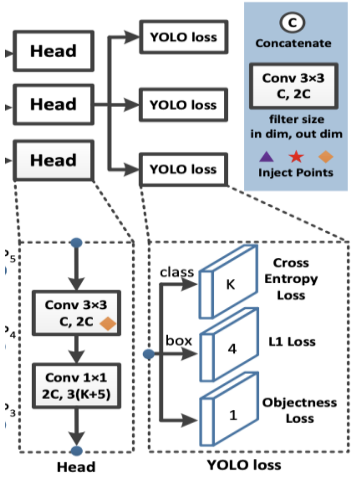


# Object Detector 강화



```python
class BottleneckCSP2(nn.Module):
    # CSP Bottleneck https://github.com/WongKinYiu/CrossStagePartialNetworks
    def __init__(self, c1, c2, n=1, shortcut=False, g=1, e=0.5):  # ch_in, ch_out, number, shortcut, groups, expansion
        super(BottleneckCSP2, self).__init__()
        c_ = int(c2)  # hidden channels
        self.cv1 = Conv(c1, c_, 1, 1)
        self.cv2 = nn.Conv2d(c_, c_, 1, 1, bias=False)
        self.cv3 = Conv(2 * c_, c2, 1, 1)
        self.bn = nn.BatchNorm2d(2 * c_)
        self.act = Mish()
        self.m = nn.Sequential(*[Bottleneck(c_, c_, shortcut, g, e=1.0) for _ in range(n)])

    def forward(self, x):
        x1 = self.cv1(x)
        y1 = self.m(x1)
        y2 = self.cv2(x1)
        return self.cv3(self.act(self.bn(torch.cat((y1, y2), dim=1))))
```

```python
class Detect(nn.Module):
    def __init__(self, nc=80, anchors=(), ch=()):  # detection layer
        super(Detect, self).__init__()
        self.stride = None  # strides computed during build
        self.nc = nc  # number of classes
        self.no = nc + 5  # number of outputs per anchor
        self.nl = len(anchors)  # number of detection layers
        self.na = len(anchors[0]) // 2  # number of anchors
        self.grid = [torch.zeros(1)] * self.nl  # init grid
        a = torch.tensor(anchors).float().view(self.nl, -1, 2)
        self.register_buffer('anchors', a)  # shape(nl,na,2)
        self.register_buffer('anchor_grid', a.clone().view(self.nl, 1, -1, 1, 1, 2))  # shape(nl,1,na,1,1,2)
        self.m = nn.ModuleList(nn.Conv2d(x, self.no * self.na, 1) for x in ch)  # output conv
        self.export = False  # onnx export
```



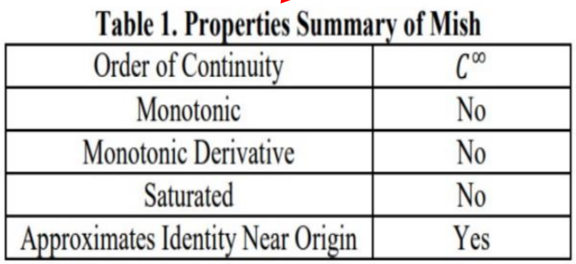
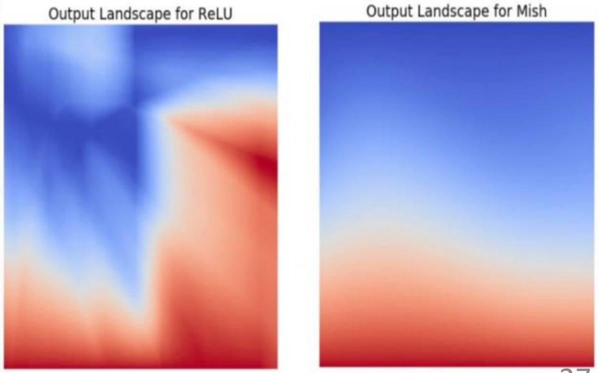


Table 1. Properties Summary of Mish

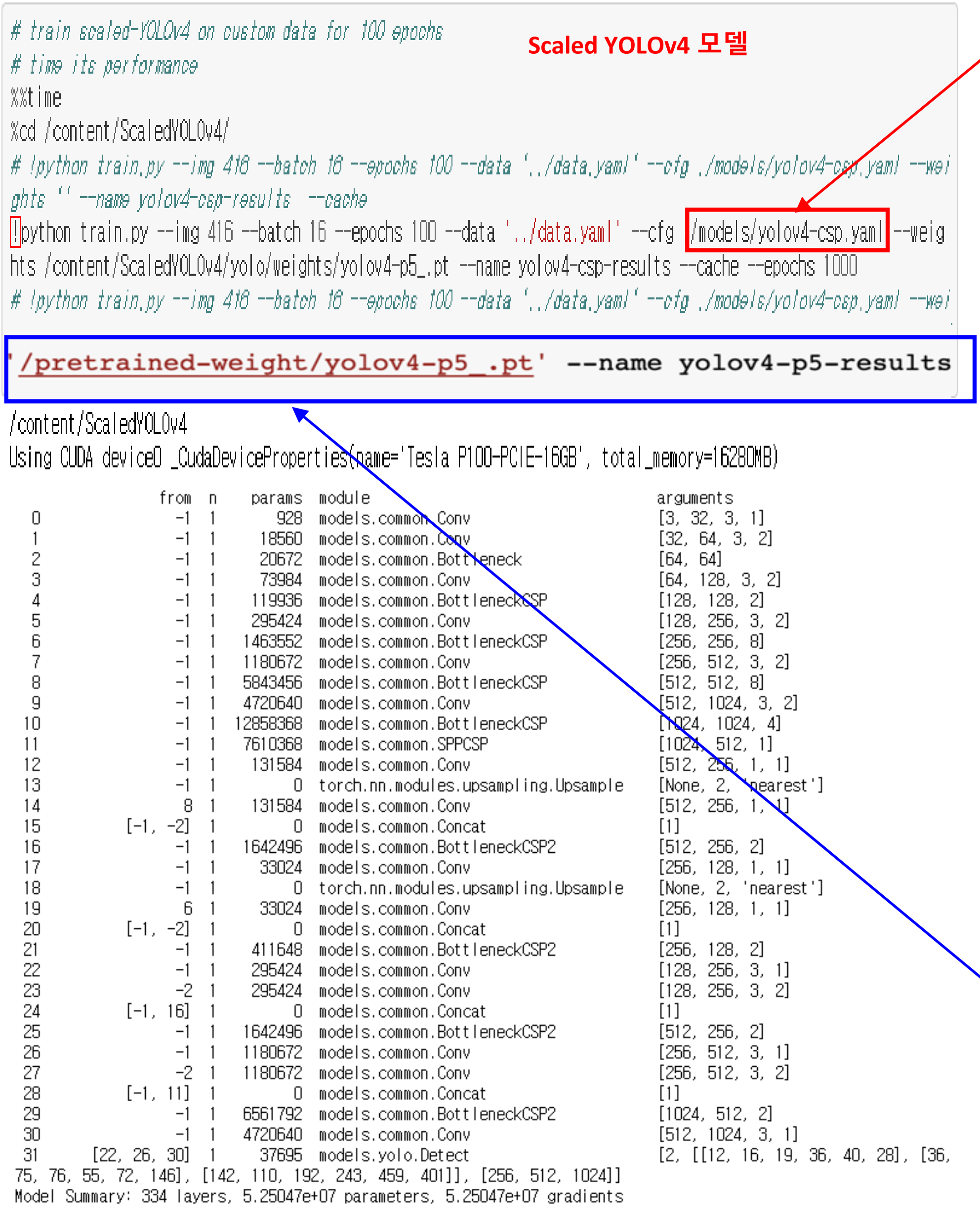
| Order of Continuity | $C^\infty$ |
|---|---|
| Monotonic | No |
| Monotonic Derivative | No |
| Saturated | No |
| Approximates Identity Near Origin | Yes |



Output Landscape for ReLU

Output Landscape for Mish

# Object Detector 강화 – 최적의 모델과 weight 선택



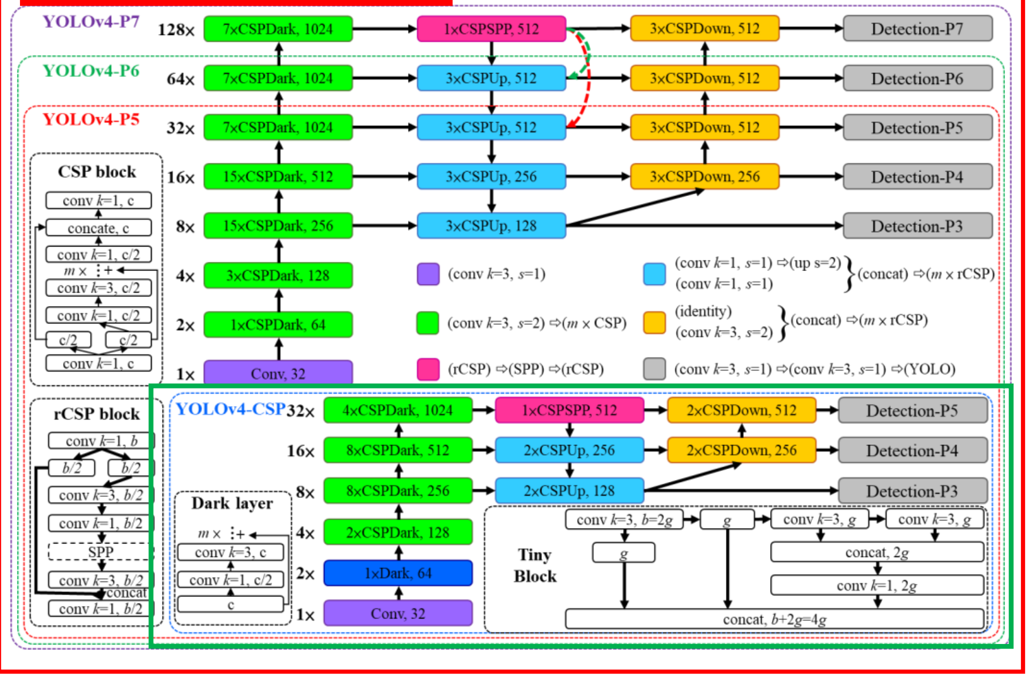
Scaled YOLOv4 모델

```
# train scaled-YOLOv4 on custom data for 100 epochs
# time its performance
%%time
%cd /content/ScaledYOLOv4/
# !python train.py --img 416 --batch 16 --epochs 100 --data '../data.yaml' --cfg ./models/yolov4-csp.yaml --wei
ghts '' --name yolov4-csp-results  --cache
!python train.py --img 416 --batch 16 --epochs 100 --data '../data.yaml' --cfg /models/yolov4-csp.yaml --weig
hts /content/ScaledYOLOv4/yolo/weights/yolov4-p5_.pt --name yolov4-csp-results --cache --epochs 1000
# !python train.py --img 416 --batch 16 --epochs 100 --data '../data.yaml' --cfg ./models/yolov4-csp.yaml --wei
```

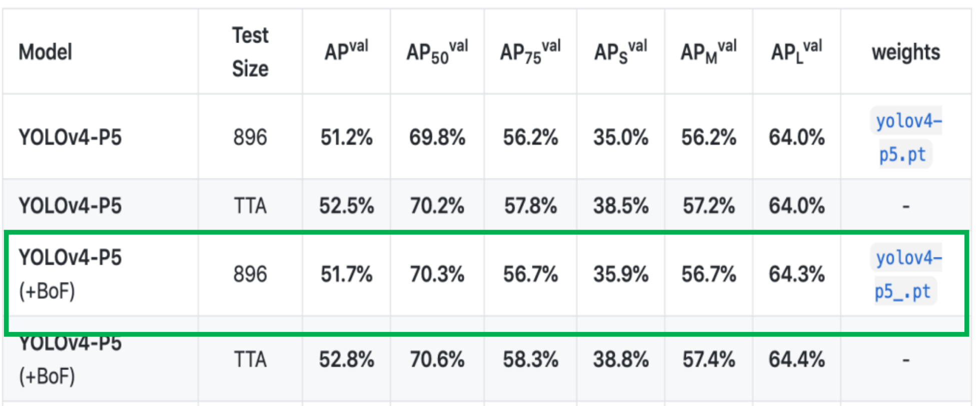
```
'/pretrained-weight/yolov4-p5_.pt' --name yolov4-p5-results
```

```
/content/ScaledYOLOv4
Using CUDA device0 _CudaDeviceProperties(name='Tesla P100-PCIE-16GB', total_memory=16280MB)

                 from  n    params  module                                  arguments
  0                -1  1       928  models.common.Conv                      [3, 32, 3, 1]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     20672  models.common.Bottleneck                [64, 64]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  1    119936  models.common.BottleneckCSP             [128, 128, 2]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  1   1463552  models.common.BottleneckCSP             [256, 256, 8]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1   5843456  models.common.BottleneckCSP             [512, 512, 8]
  9                -1  1   4720640  models.common.Conv                      [512, 1024, 3, 2]
 10                -1  1  12858368  models.common.BottleneckCSP             [1024, 1024, 4]
 11                -1  1   7610368  models.common.SPPCSP                    [1024, 512, 1]
 12                -1  1    131584  models.common.Conv                      [512, 256, 1, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 14                 8  1    131584  models.common.Conv                      [512, 256, 1, 1]
 15          [-1, -2]  1         0  models.common.Concat                    [1]
 16                -1  1   1642496  models.common.BottleneckCSP2            [512, 256, 2]
 17                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
 18                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 19                 6  1     33024  models.common.Conv                      [256, 128, 1, 1]
 20          [-1, -2]  1         0  models.common.Concat                    [1]
 21                -1  1    411648  models.common.BottleneckCSP2            [256, 128, 2]
 22                -1  1    295424  models.common.Conv                      [128, 256, 3, 1]
 23                -2  1    295424  models.common.Conv                      [128, 256, 3, 2]
 24          [-1, 16]  1         0  models.common.Concat                    [1]
 25                -1  1   1642496  models.common.BottleneckCSP2            [512, 256, 2]
 26                -1  1   1180672  models.common.Conv                      [256, 512, 3, 1]
 27                -2  1   1180672  models.common.Conv                      [256, 512, 3, 2]
 28          [-1, 11]  1         0  models.common.Concat                    [1]
 29                -1  1   6561792  models.common.BottleneckCSP2            [1024, 512, 2]
 30                -1  1   4720640  models.common.Conv                      [512, 1024, 3, 1]
 31      [22, 26, 30]  1     37695  models.yolo.Detect                      [2, [[12, 16, 19, 36, 40, 28], [36,
 75, 76, 55, 72, 146], [142, 110, 192, 243, 459, 401]], [256, 512, 1024]]
Model Summary: 334 layers, 5.25047e+07 parameters, 5.25047e+07 gradients
```

## Scaled YOLOv4 모델

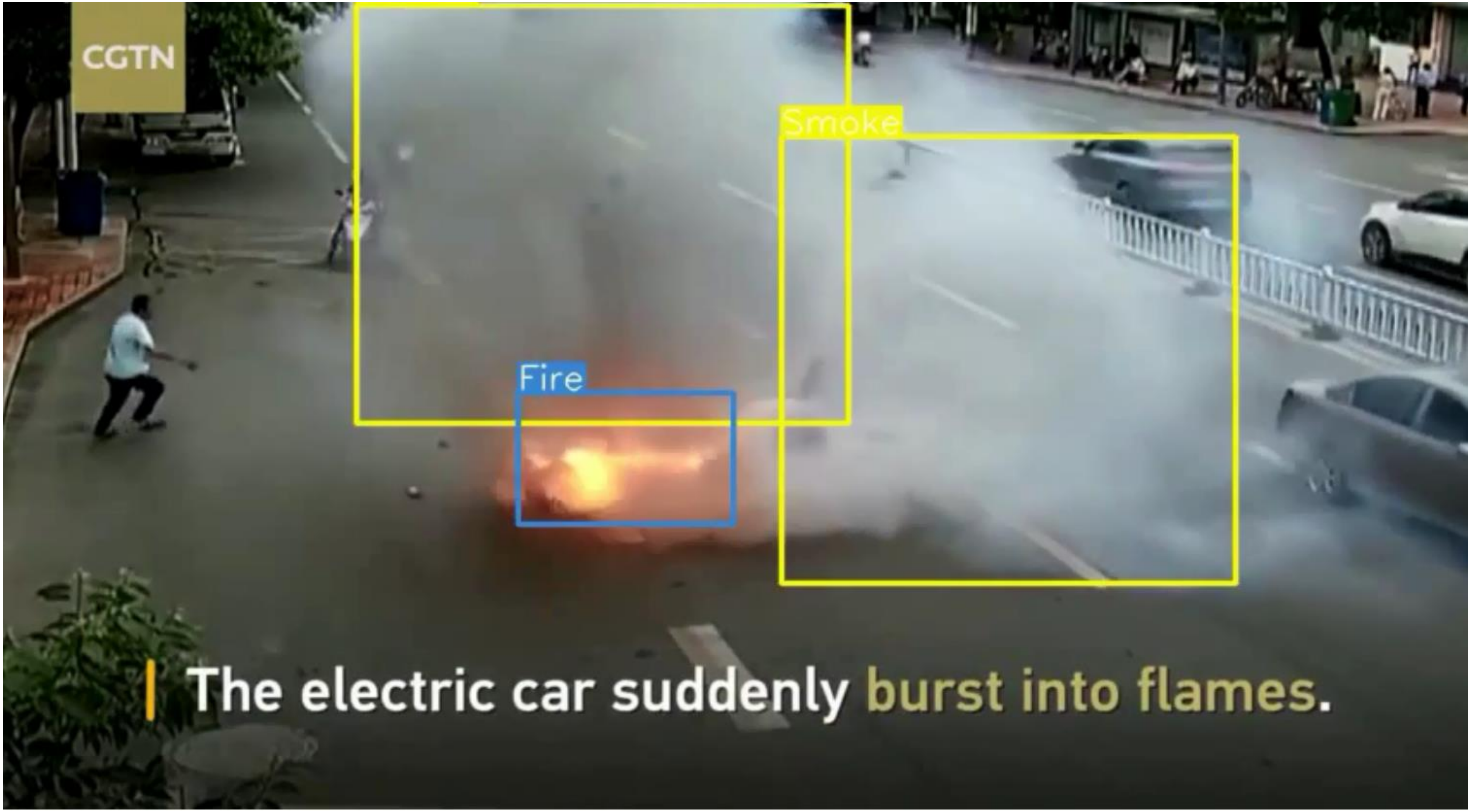


## 전이학습 가중치



| Model | Test Size | $AP^{val}$ | $AP_{50}^{val}$ | $AP_{75}^{val}$ | $AP_{S}^{val}$ | $AP_{M}^{val}$ | $AP_{L}^{val}$ | weights |
|---|---|---|---|---|---|---|---|---|
| YOLOv4-P5 | 896 | 51.2% | 69.8% | 56.2% | 35.0% | 56.2% | 64.0% | yolov4-p5.pt |
| YOLOv4-P5 | TTA | 52.5% | 70.2% | 57.8% | 38.5% | 57.2% | 64.0% | - |
| YOLOv4-P5 (+BoF) | 896 | 51.7% | 70.3% | 56.7% | 35.9% | 56.7% | 64.3% | yolov4-p5_.pt |
| YOLOv4-P5 (+BoF) | TTA | 52.8% | 70.6% | 58.3% | 38.8% | 57.4% | 64.4% | - |

# FIRE & SMOKE detection using YOLO – YOLOv4 scaled

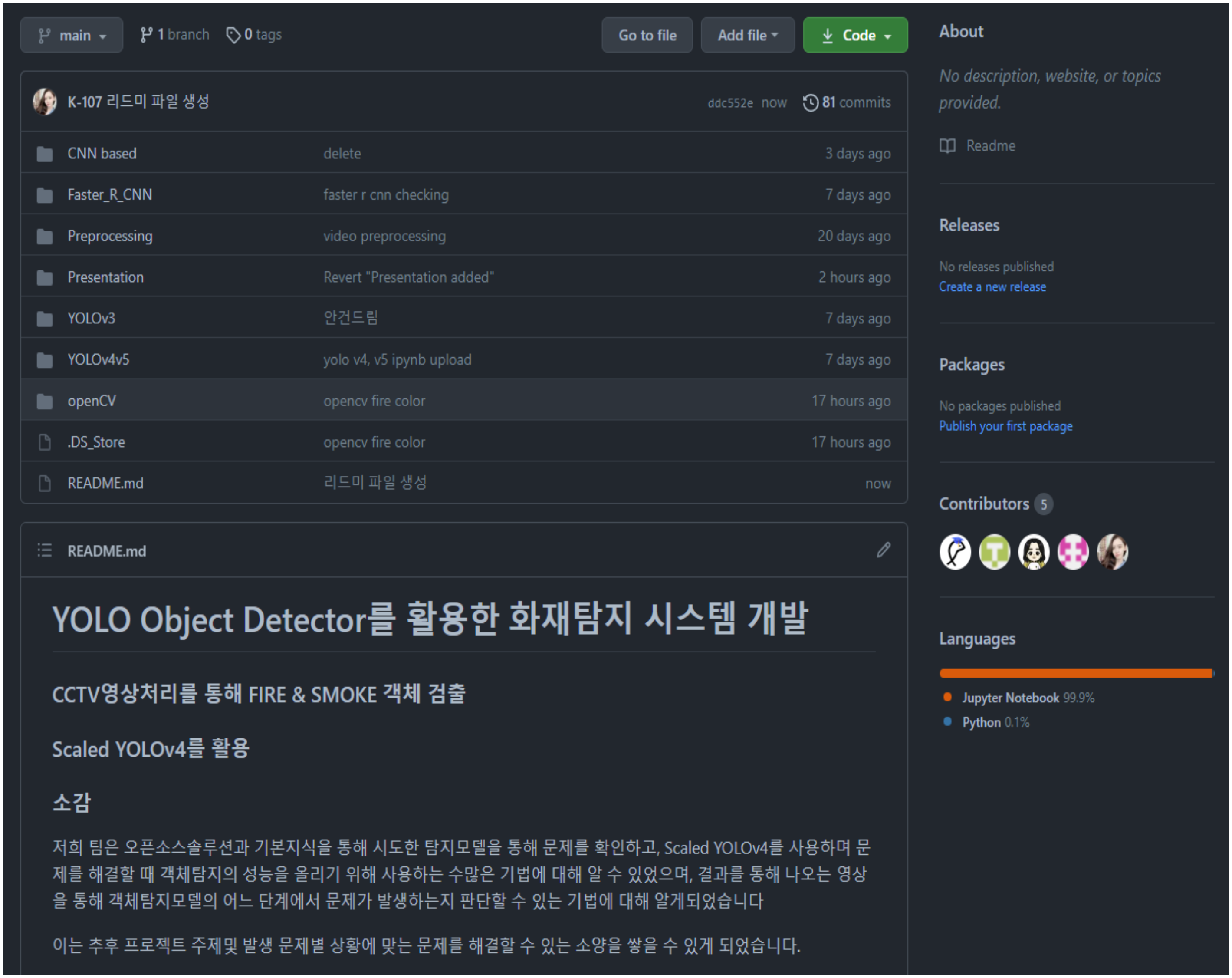


데모 영상 링크: https://youtu.be/pCvd12noNHo

# Problem & Solving

| 문제점 | 원인 | 해결방안 | 해결방법 | 설명 및 대표적 특징 |
|---|---|---|---|---|
| - 오렌지색상 객체를 화재로 탐지<br>- 작은 객체 탐지불가<br>- 후반부 화재탐지불가<br>- 연기가 아닌 부분을 연기로 탐지 | 데이터 쏠림 | SMOKE 이미지 추가 | SMOKE labeling 후 추가 학습 | SMOKE 이미지1000장 추가 |
| | 특징 추출 및 손실로 인한 분류성능저하 | 이미지 특징 손실 최소화 + 전달 | CSPBottleneck<br>(PA-NET, SPP) | CSP, Short-cut 사용하여 레이어의 정보를 전달, 객체 특징 추출 효율 향상 |
| | 탐지기능부족<br>(탐지기 성능부족) | Detector 성능 강화<br>최적 weight 및 모델 선택을 통한 전이 학습 | YOLOv4_CSP<br>YOLOv4_pt5_bof.pt | MISH 활성화 함수로 학습 효율 향상<br>BoF 기법을 통한 성능 향상 |

# GitHub



main · 1 branch · 0 tags · Go to file · Add file · Code

K-107 리드미 파일 생성 · ddc552e now · 81 commits

| | | |
|---|---|---|
| CNN based | delete | 3 days ago |
| Faster_R_CNN | faster r cnn checking | 7 days ago |
| Preprocessing | video preprocessing | 20 days ago |
| Presentation | Revert "Presentation added" | 2 hours ago |
| YOLOv3 | 안건드림 | 7 days ago |
| YOLOv4v5 | yolo v4, v5 ipynb upload | 7 days ago |
| openCV | opencv fire color | 17 hours ago |
| .DS_Store | opencv fire color | 17 hours ago |
| README.md | 리드미 파일 생성 | now |

README.md

## YOLO Object Detector를 활용한 화재탐지 시스템 개발

### CCTV영상처리를 통해 FIRE & SMOKE 객체 검출

### Scaled YOLOv4를 활용

### 소감

저희 팀은 오픈소스솔루션과 기본지식을 통해 시도한 탐지모델을 통해 문제를 확인하고, Scaled YOLOv4를 사용하며 문제를 해결할 때 객체탐지의 성능을 올리기 위해 사용하는 수많은 기법에 대해 알 수 있었으며, 결과를 통해 나오는 영상을 통해 객체탐지모델의 어느 단계에서 문제가 발생하는지 판단할 수 있는 기법에 대해 알게되었습니다

이는 추후 프로젝트 주제및 발생 문제별 상황에 맞는 문제를 해결할 수 있는 소양을 쌓을 수 있게 되었습니다.

About

*No description, website, or topics provided.*

Readme

Releases

No releases published
Create a new release

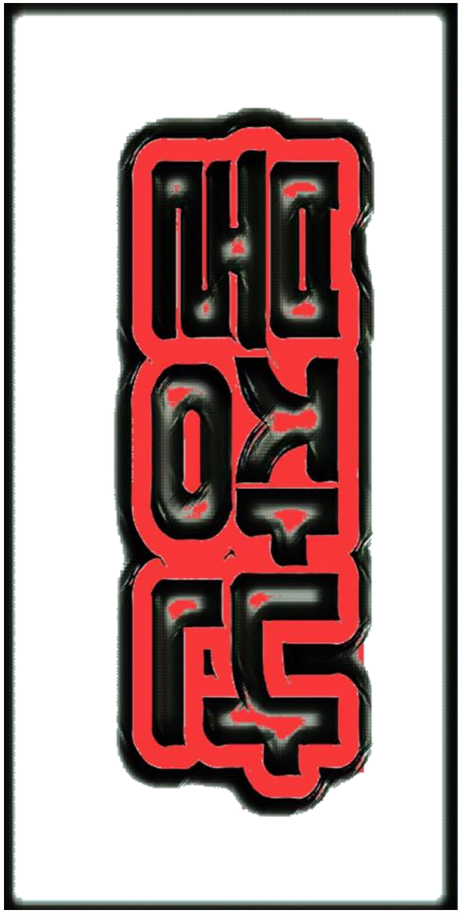
Packages

No packages published
Publish your first package

Contributors 5

Languages

- Jupyter Notebook 99.9%
- Python 0.1%

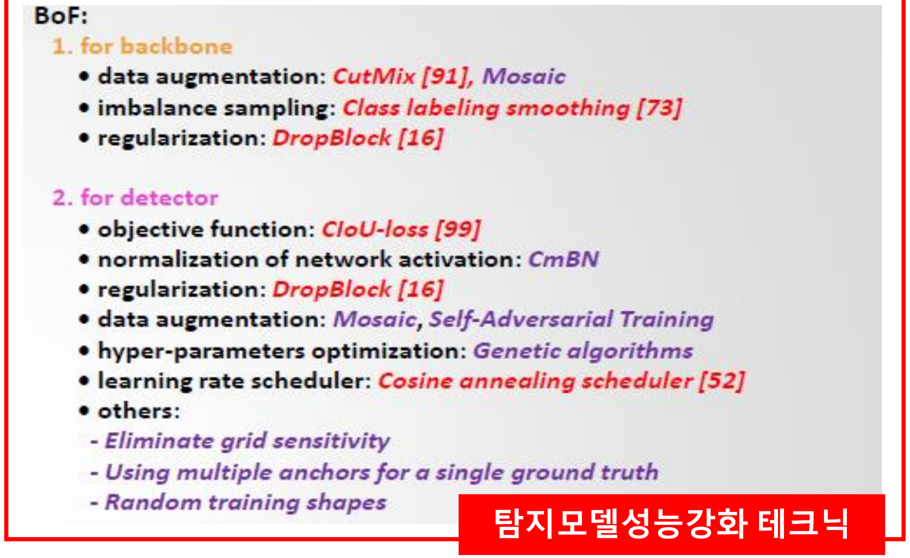


깃헙 주소:
https://github.com/MonicaKim89/Fire_detection/

# 마무리

| Backbone | Neck | Head |
|---|---|---|
| • 객체 특징 추출 | • 객체 정보전달<br>• 레이어가 많거나 이미지 해상도가 낮을 때<br>• 이미지해상도는 높으나 너무 클 때 | • 객체 위치 예측<br>• 객체 분류 |

발생가능문제



BoF:

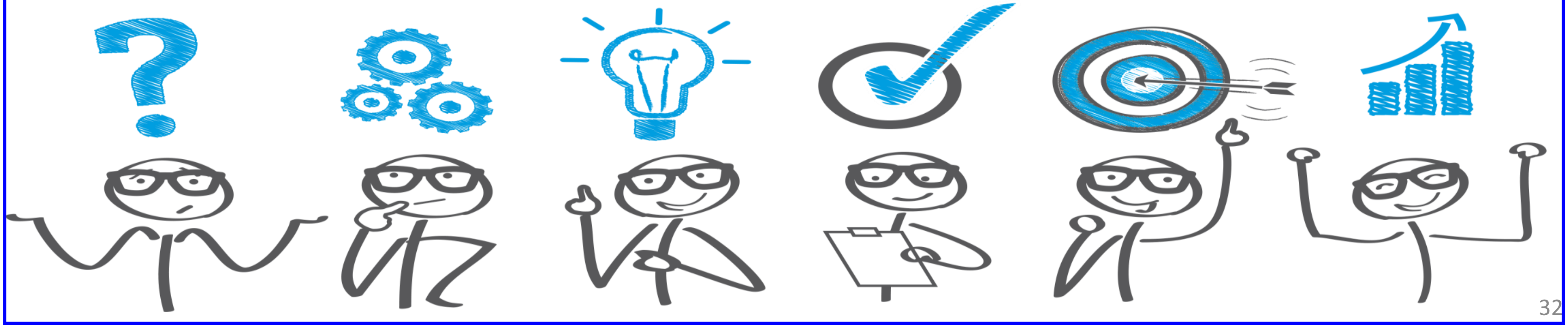
1. for backbone
   - data augmentation: *CutMix [91], Mosaic*
   - imbalance sampling: *Class labeling smoothing [73]*
   - regularization: *DropBlock [16]*

2. for detector
   - objective function: *CIoU-loss [99]*
   - normalization of network activation: *CmBN*
   - regularization: *DropBlock [16]*
   - data augmentation: *Mosaic, Self-Adversarial Training*
   - hyper-parameters optimization: *Genetic algorithms*
   - learning rate scheduler: *Cosine annealing scheduler [52]*
   - others:
     - *Eliminate grid sensitivity*
     - *Using multiple anchors for a single ground truth*
     - *Random training shapes*

탐지모델성능강화 테크닉

## 팀원 능력

Needs파악 · 해결방안모색 · 아이디어 발굴 · 솔루션 확인 · 목표 달성 · 성공



# Q & A

솔루션 모색

openCV

딥러닝 이론

딥러닝 이론

Fast R-CNN

VGG16

Resnet 분류기

YOLOv3

Real-Time

YOLOv4, 5

Scaled YOLOv4