RCNN&VGG를 활용한 이상탐지 구현

정보융합전공 정은지

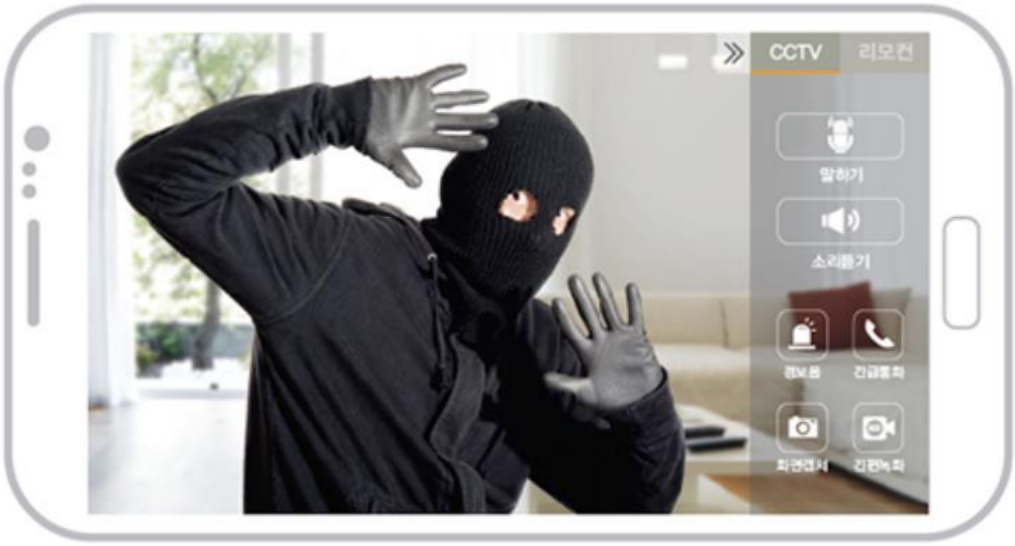
1. 요약

이미지 객체 인식 방법론 중 하나인 RCNN을 활용하여 배경만 있던 사진 프레임들 사이에서 어떠한 객체가 등장했을 때 이상 객체를 탐지하는 모델을 제작하고 싶어 진행된 프로젝트이다.

2. 주제 선정 이유 및 소개

해당 주제는 카드사의 이상 거래 탐지 시스템에서 착안하였다. 이상 거래 탐지 시스템은 고객의 평소 거래 데이터와 상이한 데이터가 발생했을 때 이상 거래를 탐지하고 카드 거래를 제한하는 시스템으로 이를 이미지에 접목시켜 생각하였다.

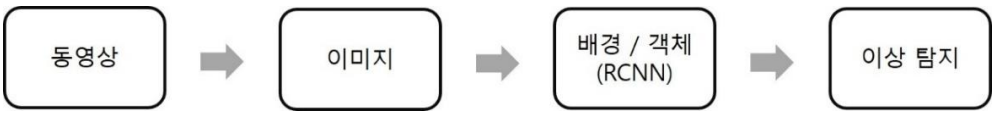
배경만 있던 이미지들의 모임에서 특이한 객체가 출현했을 때 이상 상황이 발생했음을 탐지하는 것인데 이는 현재 홈 CCTV에서 활용되고 있다.



<홈 CCTV 예시>

평소 홈 CCTV를 이용하며 어떤 방식으로 구현할 수 있을지 궁금증이 있었는데 인공지능1 수업 시간에 배운 다양한 알고리즘 중 이미지 객체 인식으로 구현 가능하다고 생각했고, 이번 프로젝트 주제로 선정하게 되었다.

기본 아이디어는 배경만 있다가(일반 가정 환경) 무언가 출현했을(이상 객체) 상황이 있는 동영상을 이미지로 변환하여 배경만 있으면 이상 없는 상황으로 인식하고 객체가 인식되었을 때는 이상 상황으로 탐지하는 방식이다.



3. 데이터

1) 설명 및 현황

데이터는 배경만 있는 이미지와 사람이 배경에 포함된 이미지의 데이터를 찾고 싶었으나 완벽하게 일치하는 데이터로 학습된 모델을 찾을 수 없었고 사람을 객체로 학습 데이터를 구축하기에는 어려움이 있어 배경과 배경에 포함된 객체를 학습하는 데이터를 활용했다.

배경과 배경에 포함된 객체(비행기)를 RCNN을 통해 학습한 모델이 있어 이를 활용하였다.

학습 데이터는 단순 배경만 있는 이미지, 배경과 객체(비행기)를 포함한 이미지 233개이고, 'Images' 폴더에 저장되어 있다. 그리고 각 이미지의 객체의 좌표를 학습하기 위한 데이터는 csv 파일 형식으로 'Airplanes_Annotations' 폴더에 저장되어 있고, 이미지 수와 동일하게 233개이다.

<이미지 데이터 예시>

	A	B
1	4	
2	15 69 40 96	
3	10 173 42 203	
4	74 212 100 243	
5	233 197 256 226	

<좌표 설정에 활용하기 위한 데이터 예시>

2) 전처리 과정

개발 환경은 GPU 사용을 위해 'google colaboratory'를 사용하였다.

```
[2]  import os,cv2,keras
     import pandas as pd
     import matplotlib.pyplot as plt
     import numpy as np
     import tensorflow as tf
```

패키지 리스트

```
Using TensorFlow backend.
```

```
[3]  path = "Images"
     annot = "Airplanes_Annotations"
```

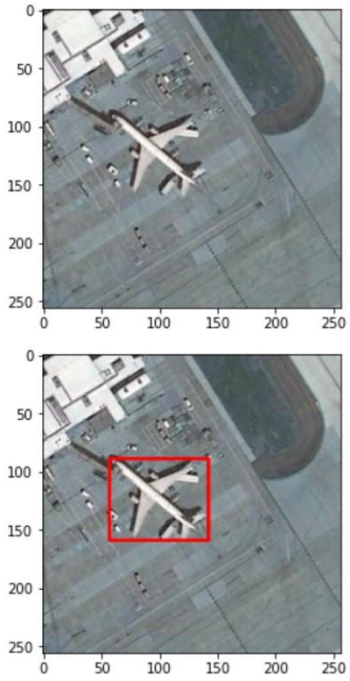
데이터 저장

패키지 import 및 데이터 저장 후 객체 인식을 위한 학습 데이터 구축 과정은 아래와 같다.

```
[4]  for e,i in enumerate(os.listdir(annot)):
         if e < 10:
             filename = i.split(".")[0]+".jpg"
             print(filename)
             img = cv2.imread(os.path.join(path,filename))
             df = pd.read_csv(os.path.join(annot,i))
             plt.imshow(img)
             for row in df.iterrows():
                 x1 = int(row[1][0].split(" ")[0])
                 y1 = int(row[1][0].split(" ")[1])
                 x2 = int(row[1][0].split(" ")[2])
                 y2 = int(row[1][0].split(" ")[3])
                 cv2.rectangle(img,(x1,y1),(x2,y2),(255,0,0), 2)
             plt.figure()
             plt.imshow(img)
             break
```

Annot(좌표 설정에 활용하기 위한 데이터) 활용하여 객체의 위치를 설정하는 과정

오른쪽은 좌표가 설정되어 디텍트된 결과

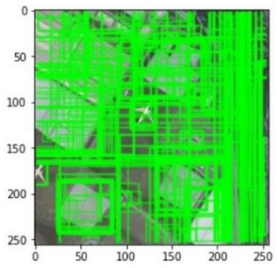


학습을 위한 regions을 생성하는 과정 및 생성된 결과 이미지이다.

```
[5]  cv2.setUseOptimized(True);
     ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
```

```
[6]  im = cv2.imread(os.path.join(path,"42850.jpg"))
     ss.setBaseImage(im)
     ss.switchToSelectiveSearchFast()
     rects = ss.process()
     imOut = im.copy()
     for i, rect in (enumerate(rects)):
         x, y, w, h = rect
     #     print(x,y,w,h)
     #     imOut = imOut[x:x+w,y:y+h]
         cv2.rectangle(imOut, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
     # plt.figure()
     plt.imshow(imOut)
```

```
[6]  <matplotlib.image.AxesImage at 0x7f3ae2169278>
```



regions을 생성

각 이미지 데이터에 검색을 위한 설정을 진행하였다.

'ss.switchToSelectiveSearchFast()' 와 'ssresults = ss.process()'를 통해 초기화 및 regions를 세팅한다. 검색 결과 중 처음 2,000개 결과 전체를 반복한 후 'get_iou()'를 통해 regions의 IOU를 계산한다. 이미지 내에 배경(negative)이 많을 경우 인식하고 싶은 객체(positive) 인식에 어려움이 생길 수 있으므로 하나의 이미지에 최대 30개의 negative와 positive를 수집하도록 설정하였다. 아래는 해당 코드이다.

```
for e,i in enumerate(os.listdir(annot)):
    try:
        if i.startswith("airplane"):
            filename = i.split(".")[0]+".jpg"
            print(e,filename)
            image = cv2.imread(os.path.join(path,filename))
            df = pd.read_csv(os.path.join(annot,i))
            gtvalues=[]
            for row in df.iterrows():
                x1 = int(row[1][0].split(" ")[0])
                y1 = int(row[1][0].split(" ")[1])
                x2 = int(row[1][0].split(" ")[2])
                y2 = int(row[1][0].split(" ")[3])
                gtvalues.append({"x1":x1,"x2":x2,"y1":y1,"y2":y2})
            ss.setBaseImage(image)
            ss.switchToSelectiveSearchFast()
            ssresults = ss.process()
            imout = image.copy()
            counter = 0
            falsecounter = 0
            flag = 0
            fflag = 0
            bflag = 0
            for e,result in enumerate(ssresults):
```

```
            for e,result in enumerate(ssresults):
                if e < 2000 and flag == 0:
                    for gtval in gtvalues:
                        x,y,w,h = result
                        iou = get_iou(gtval,{"x1":x,"x2":x+w,"y1":y,"y2":y+h})
                        if counter < 30:
                            if iou > 0.70:
                                timage = imout[y:y+h,x:x+w]
                                resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                                train_images.append(resized)
                                train_labels.append(1)
                                counter += 1
                        else :
                            fflag =1
                        if falsecounter <30:
                            if iou < 0.3:
                                timage = imout[y:y+h,x:x+w]
                                resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                                train_images.append(resized)
                                train_labels.append(0)
                                falsecounter += 1
                        else :
                            bflag = 1
                    if fflag == 1 and bflag == 1:
                        print("inside")
                        flag = 1
```

4. 방법론

학습 모델에는 VGG16을 변형하여 활용하였다. X의 size는 아래와 같다.

```
[11] X_new = np.array(train_images)
     y_new = np.array(train_labels)
```

```
[12] X_new.shape
```

```
(8909, 224, 224, 3)
```

VGG16 사용을 위한 패키지를 import하였고, VGG16은 output을 1,000개의 클래스로 반환하는데 이 모델의 경우 배경(negative), 객체(positive) 두 개의 클래스로 결과값이 필요하므로 모델을 변환하였다.

```
[14] vggmodel = VGG16(weights='imagenet', include_top=True)
     vggmodel.summary()
```

```
[16] X= vggmodel.layers[-2].output
```

```
[17] predictions = Dense(2, activation="softmax")(X)
```

```
[18] model_final = Model(vggmodel.input, predictions)
```

```
[19] from keras.optimizers import Adam
     opt = Adam(lr=0.0001)
```

```
[20] model_final.compile(loss = keras.losses.categorical_crossentropy, optimizer = opt, metrics=["accuracy"])
```

```
[21] model_final.summary()
```

아래와 같이 기존 VGG16은 output이 1,000개의 클래스이고 변경한 VGG는 output이 2개의 클래스이다.

```
model_final.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

기존 VGG16

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 8194
=================================================================
Total params: 134,268,738
Trainable params: 126,633,474
Non-trainable params: 7,635,264
_________________________________________________________________
```

변경한 VGG

<최종 모델의 구성>

Label을 one hot representation하는 과정이며 MyLabelBinarizer ()사용하여 데이터를 인코딩하는 과정을 거쳤다.

```
[22] from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import LabelBinarizer
```

```
[23] class MyLabelBinarizer(LabelBinarizer):
         def transform(self, y):
             Y = super().transform(y)
             if self.y_type_ == 'binary':
                 return np.hstack((Y, 1-Y))
             else:
                 return Y
         def inverse_transform(self, Y, threshold=None):
             if self.y_type_ == 'binary':
                 return super().inverse_transform(Y[:, 0], threshold)
             else:
                 return super().inverse_transform(Y, threshold)
```

실험을 위해 학습 데이터와 테스트 데이터는 9 : 1 비율로 구성하였고, 그 결과는 아래와 같다.

```
[24] lenc = MyLabelBinarizer()
     Y =  lenc.fit_transform(y_new)
```

```
[25] X_train, X_test , y_train, y_test = train_test_split(X_new,Y,test_size=0.10)
```

```
[26] print(X_train.shape,X_test.shape,y_train.shape,y_test.shape)
```

```
(8018, 224, 224, 3) (891, 224, 224, 3) (8018, 2) (891, 2)
```

```
[27] trdata = ImageDataGenerator(horizontal_flip=True, vertical_flip=True, rotation_range=1)
     traindata = trdata.flow(x=X_train, y=y_train)
     tsdata = ImageDataGenerator(horizontal_flip=True, vertical_flip=True, rotation_range=1)
     testdata = tsdata.flow(x=X_test, y=y_test)
```

```
[28] from keras.callbacks import ModelCheckpoint, EarlyStopping
```

5. 실험 결과

초기 실험 진행 시 'google colaboratory'에서 GPU를 이용할 경우 제한된 사용량을 초과하면 GPU 연결이 자동으로 해제되고 데이터가 소실되는 문제가 있었다. 따라서, 원래 구현된 모델의 코드에서는 'steps_per_epoch'가 10이고, 'epochs'가 100이었으나 GPU 연결 문제로 부득이하게 1과 10으로 줄여 실험하였다.

```
[29] checkpoint = ModelCheckpoint("ieeercnn_vgg16_1.h5", monitor='val_loss', verbose=1, save_best_only=True, save_weights_onl
     early = EarlyStopping(monitor='val_loss', min_delta=0, patience=0, verbose=1, mode='auto')

[30] hist = model_final.fit_generator(generator= traindata, steps_per_epoch= 1, epochs= 10, validation_data= testdata, valida

Epoch 1/10
1/1 [==============================] - 12s 12s/step - loss: 0.8020 - accuracy: 0.6250 - val_loss: 0.2549 - val_accuracy:
Epoch 2/10
1/1 [==============================] - 1s 754ms/step - loss: 0.2942 - accuracy: 0.8750 - val_loss: 0.2801 - val_accuracy:
Epoch 3/10
1/1 [==============================] - 1s 769ms/step - loss: 1.0842 - accuracy: 0.8750 - val_loss: 0.0488 - val_accuracy:
Epoch 4/10
1/1 [==============================] - 1s 755ms/step - loss: 2.5154 - accuracy: 0.5625 - val_loss: 0.9391 - val_accuracy:
```

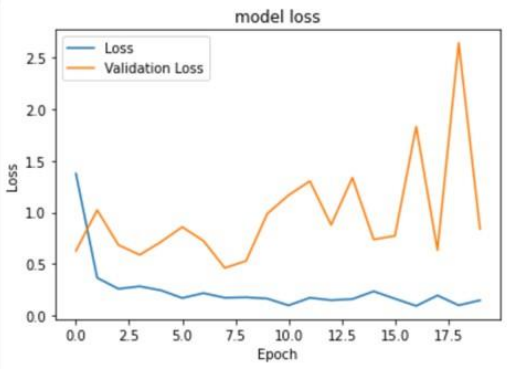
그러나 위 실험 결과 accuracy가 0.5625로 너무 낮게 나와 여러 차례 GPU 연결 최대치가 얼마인지 확인하였다. 그 결과 'steps_per_epoch = 10'가 'epochs = 20'으로 설정하여 실험을 진행했고 accuracy 0.9542의 결과를 얻었다.

```
[30] hist = model_final.fit_generator(generator= traindata, steps_per_epoch= 10, epochs= 20, validation_data= te

Epoch 1/20
10/10 [==============================] - 17s 2s/step - loss: 0.9389 - accuracy: 0.7844 - val_loss: 0.3881 -
Epoch 2/20
10/10 [==============================] - 5s 533ms/step - loss: 0.3102 - accuracy: 0.8813 - val_loss: 0.1329
Epoch 3/20
10/10 [==============================] - 5s 531ms/step - loss: 0.2881 - accuracy: 0.8969 - val_loss: 0.3063
Epoch 4/20
10/10 [==============================] - 5s 531ms/step - loss: 0.3826 - accuracy: 0.8750 - val_loss: 0.1242
Epoch 5/20
10/10 [==============================] - 5s 528ms/step - loss: 0.2326 - accuracy: 0.9187 - val_loss: 0.1563
Epoch 6/20
10/10 [==============================] - 5s 531ms/step - loss: 0.1650 - accuracy: 0.9406 - val_loss: 0.0862
Epoch 7/20
10/10 [==============================] - 5s 532ms/step - loss: 0.1844 - accuracy: 0.9438 - val_loss: 0.1846
10/10 [==============================] - 5s 531ms/step - loss: 0.1261 - accuracy: 0.9438 - val_loss: 0.1626
Epoch 17/20
10/10 [==============================] - 5s 529ms/step - loss: 0.1351 - accuracy: 0.9563 - val_loss: 0.0378
Epoch 18/20
10/10 [==============================] - 5s 531ms/step - loss: 0.1848 - accuracy: 0.9375 - val_loss: 0.0451
Epoch 19/20
10/10 [==============================] - 5s 531ms/step - loss: 0.1397 - accuracy: 0.9500 - val_loss: 0.0561
Epoch 20/20
10/10 [==============================] - 8s 751ms/step - loss: 0.1065 - accuracy: 0.9542 - val_loss: 0.2860
```

Model loss의 결과는 다음과 같고 epochs를 최대로 높였으나 원래 값보다 작은 값이기 때문에 이러한 결과가 나온 것으로 예상된다.

```
[31] import matplotlib.pyplot as plt
     # plt.plot(hist.history["acc"])
     # plt.plot(hist.history['val_acc'])
     plt.plot(hist.history['loss'])
     plt.plot(hist.history['val_loss'])
     plt.title("model loss")
     plt.ylabel("Loss")
     plt.xlabel("Epoch")
     plt.legend(["Loss","Validation Loss"])
     plt.show()
     plt.savefig('chart loss.png')
```



6. 결론

테스트 데이터의 regions 중 배경인 경우는 '이상 없음', 객체가 인식된 경우 '이상객체 탐지'로 결과를 출력하였다.

```
for i in range(len(X_test)):
  im = X_test[i]
  plt.imshow(im)
  img = np.expand_dims(im, axis=0)
  out= model_final.predict(img)
  if out[0][0] > out[0][1]:
    print(i)
    print("이상객체 탐지")
  else:
    print(i)
    print("이상 없음")
```

```
0
이상객체 탐지
1
이상객체 탐지
2
이상객체 탐지
3
이상 없음
4
이상 없음
5
이상 없음
6
이상객체 탐지
```

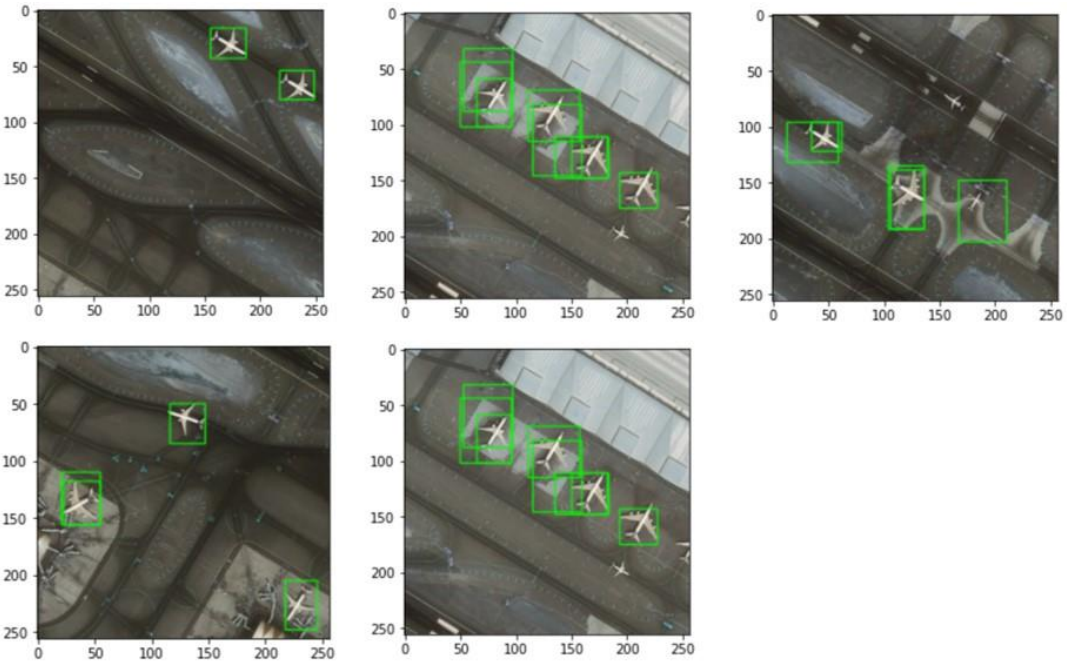
```
이상 없음
878
이상 없음
879
이상 없음
880
이상객체 탐지
881
이상 없음
882
이상객체 탐지
883
이상 없음
884
이상 없음
885
이상객체 탐지
886
이상객체 탐지
887
이상객체 탐지
888
이상 없음
889
이상객체 탐지
890
이상 없음
```

이상 객체 탐지의 최종 목적은 배경에서 특정 물체가 발견된 시점이기 때문에 객체를 포함하고 있는 이미지의 파일명을 출력하였다. 해당 테스트 데이터에서는 총 17개의 이상 객체가 검출되었다.

```
z=0
for e,i in enumerate(os.listdir(path)):
    if i.startswith("4"):
        z += 1
        img = cv2.imread(os.path.join(path,i))
        ss.setBaseImage(img)
        ss.switchToSelectiveSearchFast()
        ssresults = ss.process()
        imout = img.copy()
        for e,result in enumerate(ssresults):
            if e < 2000:
                x,y,w,h = result
                timage = imout[y:y+h,x:x+w]
                resized = cv2.resize(timage, (224,224), interpolation = cv2.INTER_AREA)
                img = np.expand_dims(resized, axis=0)
                out= model_final.predict(img)
                if out[0][0] > 0.65:
                    cv2.rectangle(imout, (x, y), (x+w, y+h), (0, 255, 0), 1, cv2.LINE_AA)
        plt.figure()
        plt.imshow(imout)
        print("이상 객체가 탐지되었습니다. 검출된 이미지 명 : " + i)
```

```
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428452.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 42847.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428482.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428451.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428481.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 42850.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 42848.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428492.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428491.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428501.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428503.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428462.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428472.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 42845.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428461.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 428483.jpg
이상 객체가 탐지되었습니다. 검출된 이미지 명 : 42849.jpg
```

아래는 실제 인식된 객체의 모습이며 총 17개의 이미지 중 샘플 몇 개에 대한 결과를 첨부하였다.



이번 프로젝트를 통해 이론으로 학습했던 CNN의 한 종류인 RCNN을 구현해볼 수 있었고, 한 가지 아쉬운 점은 사람으로 학습 데이터 구축이 가능했다면 더 완성도 높은 결과를 얻을 수 있었을 것으로 생각된다. 또한, 실제로 촬영된 영상을 이미지로 변환하여 구현해보고 싶었으나 학습 데이터를 구축할 수 없었던 점이 가장 아쉬움으로 남는다. 무료 GPU 사용 시 RAM의 제한된 사용량을 사용하면

연결이 끊어져 데이터가 전부 초기화되고 다시 처음부터 실험을 진행해야 하는 어려움이 있었고, 이러한 문제 때문에 부득이하게 데이터의 수와 model의 fit 진행 시 epochs를 줄여서 진행했고, 그 결과로 나온 model loss 그래프 결과가 아쉬웠다.

7. 참고문헌

- Step-by-Step R-CNN Implementation From Scratch In Python

  (R-CNN 코드 https://url.kr/RBgqDV)

- Colaboratory 사용법 관련 (https://url.kr/Jn2H6u)

- cv2 사용법 관련 (https://url.kr/lNqpmE)