UAV Ground Scanning System: Human Detection with Deep Learning

Xiaxin Shen
Computer and Information Technology
Purdue University
West Lafayette, IN, USA
shen452@purdue.edu

Haeun Ko
Industrial Security
Chung-Ang University
Seoul, In, Korea, South
hekoh99@naver.com

Taeuk Gwak
Computer Engineering
Chung-Ang University
Seoul, In, Korea, South
gwak2837@kakao.com

Jihyeon Noh
Software Engineering
Chung-Ang University
Seoul, In, Korea, South
jihyeon6754@gmail.com

Yeji Gong
Software Engineering
Chung-Ang University
Seoul, In, Korea, South
yejigong00@gmail.com

Hyonjun Kang
Software Engineering
Chung-Ang University
Seoul, In, Korea, South
hyonjun.kang.00@gmail.com

Minji Lee
Computer and Information Technology
Purdue University
West Lafayette, IN, USA
lee3450@purdue.edu

Eric T. Matson
Computer and Information Technology
Purdue University
West Lafayette, IN, USA
ematson@purdue.edu

***Abstract*—As technology advances, people and their governments will find more practical uses for UAV ground scanning system. As surveillance cameras become more obtainable for the average person, several drawbacks are uncovered for these consumer-grade products. Drawbacks such as lack of night vision (i.e. infrared imaging) and the lack of mobility. A raspberry pi, GPS sensor, and infrared imaging camera are utilize to develop a mobile, automatic camera in this paper. Once the camera detects the shape of a human being, it will take an infrared image of the person and utilize the GPS sensor to provide the exact location of the culprit. This paper presents a deep learning-based convolutional neural network (CNN) to implement the object detection for images taken from a high-altitude downward angle. The implementation of the hardware and sofware are described in this paper, and the infrared dataset made by the research team is introduced as well, which is open-source and ready to be used for any related machine learning tasks. Both YOLO and SSD algorithms are utilized in this research, and weights from pre-trained models including YOLOv3 and SSD mobile net are used in the phase of transfer learning. The system is expected to be deployed and push warning messages with GPS information when human are detected when searching or rescuing events are operated.**

***Index Terms*—UAV, YOLO, SSD, computer vision, convolutional neural network, deep learning, GPS sensor, embedded system development, surveillance, infrared imaging.**

I. Introduction

After examined more than 840,000 police incidents from 2017 across 10 major American cities, a report made by TheSleepJudge [1] concludes that felonies like DWI/DUI, murder and non-negligent manslaughter, rape/sexual assault, robbery, aggravated assault, motor vehicle theft occurred more likely during night time 7 pm – 6:59 am than daytime 7 am – 6:59 pm, while burglaries happen equally likely. Tracking people by the UAV during the night can help increase the rate of the solved criminal cases and decrease the criminal rate to a remarkable extent. Secure surveillance systems established based on the UAV can help recognize objects which are moving or static and show necessary data like the information from Global Positioning System (GPS).

The purpose of the project is to build a system one UAV that will use infrared camera to detect humans on the ground and track them using GPS location. The main use of the application would be in the area of security; as a potential anti-terrorism application that could find and return the GPS location of moving people during the night. To implement this, cameras that can recognize human heads need to be deployed on the UAV with motion detection and low battery cost as well as long battery life.

There are previous research done in the field of Unmanned Aerial Vehicles (UAVs) with utilizing object detection. William Andrew [2] proposed three deep convolutional neural network architectures and applied them to the UAV navigation. The species detector is created based on YOLOv2. This system is applied in the scene of individual animal identification. MohaimenianPour and Vaughan [3] presented the UAV-based simultaneous detection of hands and faces including both RGB and gray-scale images. Kouris, Kyrkou and Bouganis. [4] addressed the challenge of deployment of region-based object detection with aerial imagery. Kannadaguli and Prashanth [5] built a huamn detection system with the deep

learning approach You Only Look Once (YOLO) v4 [6]. Both thermal imaging and videos were used in their applications. Instead of using traditional semantic segmentation method, their work used the proposed single shot detection. Wong *et al*. [7] conducted experiments several tiny object detection models designed for embedded devices, including tiny SSD and tiny YOLO. They achieved an mAP of 61.3% on VOC 2007 dataset, which is higher than tiny YOLO. With this achievement, the model size if tiny SSD is 2.3 MB, which is around 26 times smaller than tiny YOLO. Das *et al*. [8] utilized faster R-CNN SSD, YOLO to do the object detection task on a UAV which was customized with stabilized flight.

The deep convolutional neural networks (CNN) can be used to recognize objects. The Convolutional Neural Network (CNN) [9] is commonly used at the field of computer vision and pattern recognition. In this project, Convolutional Neural Network is used to do object detection. The key point for utilizing it is to compress large data such as images. The CNN include convolutional layers, pooling layers and dense layers. This phase is supervised learning with images and labels. The labels include the class name, which is represented in the index in the output, and four numbers for describing the box around the object including X_CENTER, Y_CENTER, WIDTH and HEIGHT. There are several different networks established based on CNN. For the purpose of image classification, LeNet [10], AlexNet [11], VGG [12], GoogLeNet [13] are created. For detecting the objects, RCNN [14], Fast RCNN [15], Faster RCNN [16], YOLO [17] , SSD [18] are trained and established. Considering the superb speed of the YOLO and SSD, the pre-trained model of YOLO and SSD are used as the base of transfer learning in this project. The deep transfer learning [19] can help solve the problem of insufficient training data. In this way, independent training data and identically distributed test data are not mandatory.

II. Environment Setting

A. Raspberry Pi Setting

The application is currently using a raspberry pi model 3B [20] running the 32-bit Raspberry Pi OS. It has VNC setup on it to allow for remote connection. For use on an UAV, the Raspberry Pi is secured in a weather proof case to keep it protected from the elements.

B. GPS Sensor Setting

The GPS sensor that is utilized for this application is the Adafruit Ultimate GPS with gpsd [21]. It is connected to the Raspberry Pi using a USB to TTL serial cable. The GPS is setup using python and outputs the longitude, latitude and altitude of the current location of the UAV. Although the GPS sensor is always active, it only transmits a location back to the user when a human is detected by the infrared camera sensor.

C. Camera Sensor Setting

The original camera that was being used was the raspberry pi NOIR camera [22]. It was setup using python commands to take pictures every second until 50 pictures have been taken.

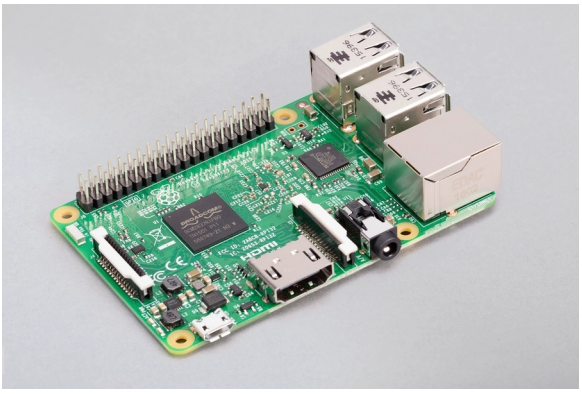

Fig. 1. Picture of Raspberry Pi Model 3B

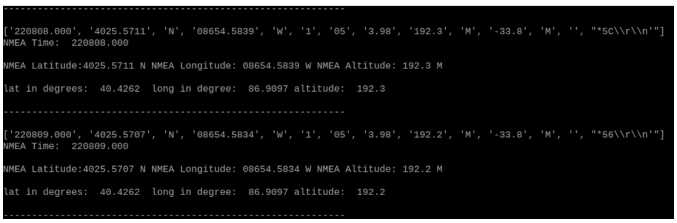

Fig. 2. GPS Output

Upon testing the NOIR camera, it was found that the camera required the use of a IR illuminator to be able to detect bodies in the dark. Therefore, the team needed to find a way to work around this for the project. The reason for this was because the main application of the UAV is during nighttime and the NOIR camera is unable to detect bodies in the dark.

Fig. 3. Pi NoIR Camera V2

Also it should be prepared infrared filter that can be installed on the DSLR camera in order to take infrared photographs, because it must get permission from authorities to take pictures in the air with drones in South Korea. The infrared filter has the effect of blocking electromagnetic waves below 720nm including visible light, allowing us to take infrared photos.

D. Code Environment Setting

Libraries including ipython 7.16.1, jupyter-http-over-ws 0.0.8, Keras 2.3.1, matplotlib 3.3.4, notebook 6.3.0, numpy 1.19.5, opencv-python 4.5.1.48, pandas 1.1.5, Pillow 8.2.0,

Fig. 4. HOYA Infrared filter 67mm

TensorFlow 1.15 are used in this project. Python 3.6 is used for establishing the model.

III. Dataset

A. Data Collecting

The dataset were collected by photographing the ground on the rooftop or the high places of the buildings. The dataset were taken in The Hyundai Seoul Yeouido, Chung-Ang University, Nodeul Island, and Times Square Yeongdeungpo, South Korea. Also, there are two types of infrared photographs and normal photographs, and examples are Figure 5 and figure 6. The number of pictures taken at each location is as follows.

- Nodeul Island, Chung-Ang University: 159
- The Hyundai Seoul: 95
- Infrared photos in daytime: 114
- Infrared photos in twilight: 74

Fig. 5. Picture at Korea's Yeongdeungpo Times Square

Fig. 6. Infrared picture at Korea's Nodeul Island

B. Image Preprocessing

- Labelling

Both images created by Siyah [23] and the our own created dataset were used. The model needs the images and the labels. Annotation files, include two types: one is in YOLO format and the other is in VOC format.

The study was conducted as follows: First, the labeling tool was used to create an image in YOLO format and then convert it to VOC format. YOLO and VOC format are respectively .txt and .xml format.

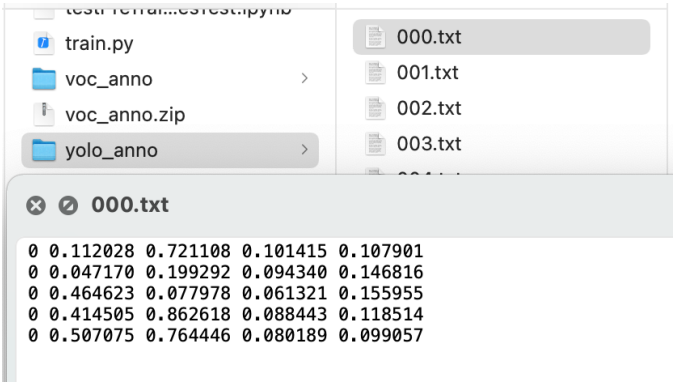

Fig. 7. YOLO annotation

Labeling images was done with Labelimg [24], which is a graphical image annotation tool developed. In this phase, the person object is identified in the image with its bounding box (bbox) information which includes 4 parameters used for coordinating as well as the class name, namely person.

- Data Cleaning

The naming of Aerial Semantic Segmentation Drone Dataset [23] is not consistent. The program is developed to analyze which number lacks, and saved them to the csv file. Based on this result, files are generated with empty content to make matching exists for every image and its annotation files.

There are some errors made when annotating images manually, like wrong label index written in the annotation files, and the lack of files which does not include any object. The

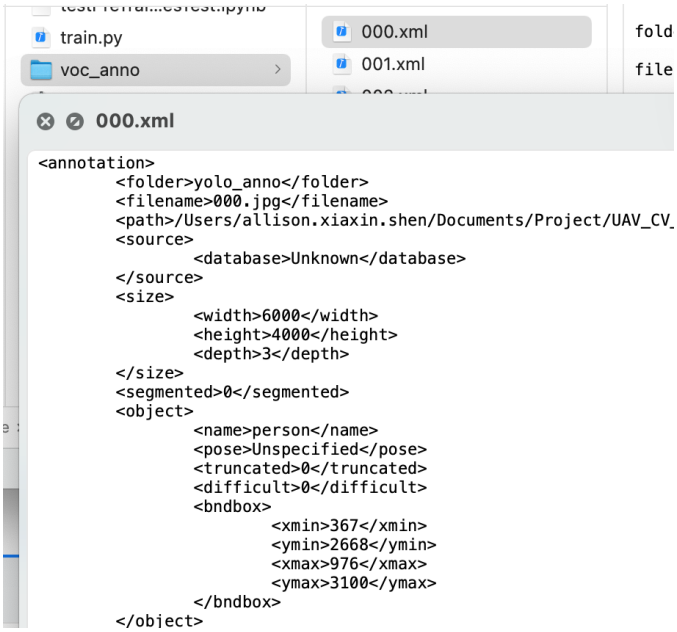

Fig. 8. VOC annotation

necessary information from images which does not include any object will be set as missing values, also referred to as NA, in pandas after converting them to CSV files. All those rows including NA need to be dropped before the next step: converting to TFRecords.

```
[##################---] 84% No person in this image!
[##################---] 85% No person in this image!
[###################--] 88% No person in this image!
[###################--] 89% No person in this image!
[####################-] 93% No person in this image!
[####################-] 93% No person in this image!
[####################-] 93% No person in this image!
[####################-] 95% No person in this image!
[####################-] 96% No person in this image!
[####################-] 96% No person in this image!
[####################] 98% No person in this image!
[####################] 100%
Successfully converted xml to csv.
   filename  width  height   class    xmin    ymin    xmax    ymax
0   000.jpg   6000    4000  person   367.0  2668.0   976.0  3100.0
1   000.jpg   6000    4000  person     0.0   503.0   566.0  1090.0
2   000.jpg   6000    4000  person  2603.0     0.0  2971.0   623.0
3   000.jpg   6000    4000  person  2221.0  3213.0  2752.0  3687.0
4   000.jpg   6000    4000  person  2801.0  2859.0  3283.0  3255.0
5   001.jpg   6000    4000  person  3219.0   354.0  3459.0   630.0
6   001.jpg   6000    4000  person  4308.0  2321.0  4613.0  2767.0
7   001.jpg   6000    4000  person  2681.0  2527.0  2985.0  2845.0
8   001.jpg   6000    4000  person   955.0  2272.0  1507.0  2732.0
9   001.jpg   6000    4000  person  2179.0  3213.0  2518.0  3538.0
10  001.jpg   6000    4000  person  2462.0  3411.0  2794.0  3800.0
```

Fig. 9. Handling NA Cases

Critical information including filename, class, width, height, xmin, xmax, ymin, and ymax are extracted from .xml files and stored in csv files. The reason of converting the data format from VOC xml files to CSV (Comma-Separated Values) files is that pandas [25] works friendly with CSV files. Figure 10 shows an example how CSV files are organized. Two chunks of data: train and test are split from the original dataset and saved to CSV files. Based on the data in CSV format, it is easier to work with in the phase of generating TFRecords, which can combine the byte string of the image and image label together, with TensorFlow Object Detection API's dataset_util library.

filename	width	height	class	xmin	ymin	xmax	ymax
032.JPG	5184	3456	person	151	395	566	975
032.JPG	5184	3456	person	2117	1	2322	475
032.JPG	5184	3456	person	4462	185	4892	765
033.JPG	5184	3456	person	2137	1	2352	440
033.JPG	5184	3456	person	4472	160	4897	730
033.JPG	5184	3456	person	2792	2830	3192	3350
033.JPG	5184	3456	person	3117	2530	3467	3075
034.JPG	5184	3456	person	2342	1655	2617	2215
034.JPG	5184	3456	person	2042	1975	2467	2545
034.JPG	5184	3456	person	2157	1	2337	375
034.JPG	5184	3456	person	4517	105	4957	675

Fig. 10. Saving as CSV files

- Images Resizing

After labeling images, resizing was needed. The figure 11 shows that the image and the bounding box are resized with the same scale. The aerial drone dataset [23] is resized from 6000x4000 to 320x320 considering the YOLO and SSD network downsamples up to 32 inputs and the limitation of memory.

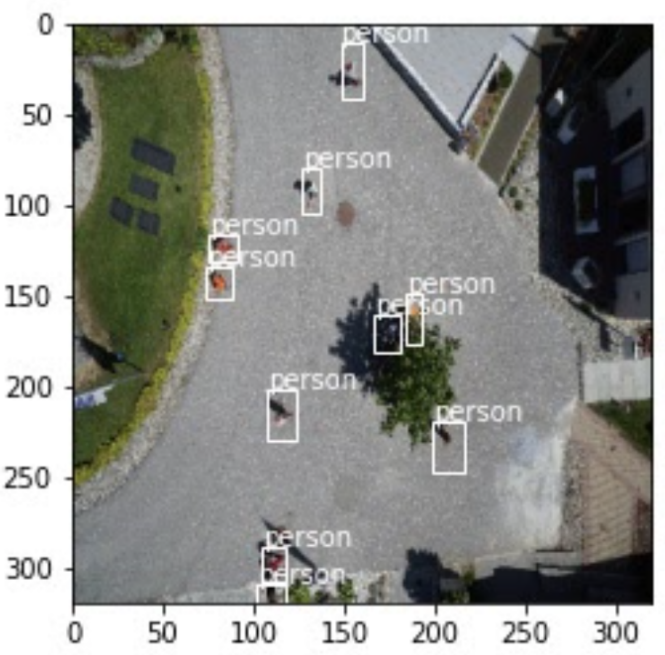

Fig. 11. Resizing Images

The library cv2 is used to resize images, the same scale needs to be used to resize the bounding box based on the new height and width of images. Figure 12 shows how bounding boxes are resized.

Since the size of the our own created dataset is also inconsistent, they need to be resized. In this phase, PIL(Python Imaging Library) was used. By resizing function in the Image module, the dataset was resized into 320X320 successfully.

```python
def resize_images_boundingbox(pathOriginalImages, newSize, pathResize,df):
  tempList = []
  for filename in glob.glob(f"{pathOriginalImages}/*.jpg"):
    basename = os.path.basename(filename)
    image_name = basename
    print(f"<<<<<<<<<resizing {image_name}>>>>>>>>")
    image = cv2.imread(filename)
    scale_x = newSize[0] / image.shape[1]
    scale_y = newSize[1] / image.shape[0]
    image = cv2.resize(image, (newSize[0], newSize[1]))
    cv2.imwrite(f"{pathResize}/{image_name}", image)  #save the resized images

    allLinesSameImage = df[df.filename == image_name]
    for i, row in allLinesSameImage.iterrows():
      new_width, new_height = newSize[0], newSize[1]

      ymin, xmin, ymax, xmax = row['ymin'], row['xmin'], row['ymax'], row['xmax']

      new_xmin = int(np.round(float(xmin) * scale_x))
      new_ymin = int(np.round(float(ymin) * scale_y))
      new_xmax = int(np.round(float(xmax) * scale_x))
      new_ymax = int(np.round(float(ymax) * scale_y))

      value = (image_name,new_width,new_height,row['class'],new_xmin,new_ymin,new_xmax,new_ymax)
      tempList.append(value)
  column_name = ['filename', 'width', 'height', 'class', 'xmin', 'ymin', 'xmax', 'ymax']
  df_resize = pd.DataFrame(tempList, columns=column_name)
  return df_resize
```

Fig. 12. Resizing Bounding Boxes

C. TFRecord Conversion

TFRecord is used for the dataset. Before converting the dataset to TFRecord, the original dataset need to be split to two parts: train_labels.csv and test_labels.csv as shown in figure 13. The dataset was shuffled and divided into train part and test part. After dividing, the each dataset was converted to TFRecord.

```
total len: 347
len of train (# images): 277
len of test (# images): 70
len of train (#rows): 1002
   filename  width  height   class  xmin  ymin  xmax  ymax
0  000.jpg    320     320  person    20   213    52   248
1  000.jpg    320     320  person     0    40    30    87
2  000.jpg    320     320  person   139     0   158    50
3  000.jpg    320     320  person   118   257   147   295
4  000.jpg    320     320  person   149   229   175   260
len of test (#rows): 256
    filename  width  height   class  xmin  ymin  xmax  ymax
21  003.jpg    320     320  person    58    35    86   102
22  003.jpg    320     320  person   100   121   136   172
25  005.jpg    320     320  person   197    77   206   104
26  005.jpg    320     320  person   184   170   195   187
27  005.jpg    320     320  person   189   164   200   181
```

Fig. 13. Train and Test Splitting

D. Preparement

- Test in dark

Taking pictures in a dark room was done to test the quality of the IR camera. This method was to ensure that the quality of the photograph was adequate not only under time but also under minimal lighting. Testing outside would result in a lot of lighting from secondary sources, such as the moon or other buildings.

E. Determination of the field of view

- 90 degrees

The camera will be attached to the UAV at a 45 degree angle between the X-axis and Y-axis. This will, ideally, give the camera the best field of view for object detection, providing as close to 90 degrees of view as possible.

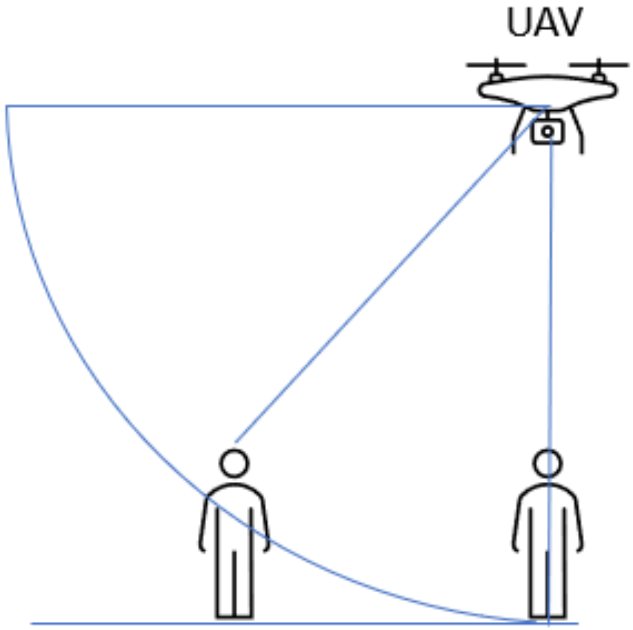

Fig. 14. Theoretical Diagram

F. Color of clothes for testers

When it comes to testing the camera it was recommended to have as much skin showing on the person. This is to enable the most amount of their body heat to be shown on the sensor without anything blocking heat. Therefore for testing, people wore t-shirts and shorts if applicable; otherwise, they had pants on with a t-shirt.

IV. Object Detection and Tracking

The goal for the Infrared Camera is to be able to detect a human being while in the dark, automatically take a picture once a human is detected, then send the picture to a database for further review. To do this, an algorithm called YOLO will be implemented to the camera for object detection.

A. YOLO

In order for YOLO to be able to detect a human being, it must first be taught what to look for. To do this, a database of 500 aerial photographs were manually tagged using the program Make Sense, placing extra emphasis on human heads. Make Sense tagged all humans photographed, then exported the database to a format that YOLO can utilize to detect humans. Due to many pictures of humans not entirely in the photograph, only humans with their heads showing were tagged with Make Sense. This was done to ensure the camera could detect a human within the entirety of its field of view and to negate the amount of false positives generated.

B. SSD

TensorFlow Lite [26] at raspberry pi was set and the pre-trained model with the algorithm SSD (Single Shot MultiBox

Detector) was deployed. [18] to do object detection with the infrared camera. The figure 15 shows when a person is recognized successfully, both screen and terminal will show the label name “person”, with a confidence score. The model is a pre-trained model established based on Tensorflow Lite [26] which is commonly used for mobile and IoT devices. The model created with the SSD algorithm was pretrained with the COCO dataset [27].

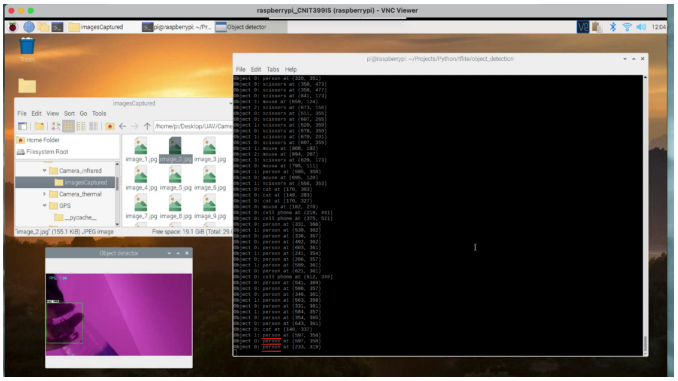

Fig. 15. Object Detection with SSD

C. Pinging the GPS

Once the Pi NoIR Camera detects a human and takes an automatic picture, the images will be tagged with the exact GPS location the picture was taken. The image will then be sent to the host computer and stored for further review. This is better than having the pictures stored on the device itself due to the possibility of losing the UAV and the contents attached to it.

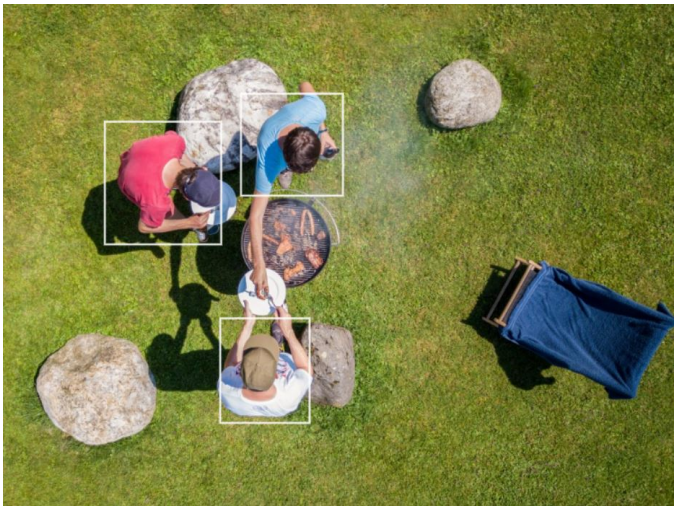

Fig. 16. Example of Make Sense Human Detection

V. Training Phase

For training the model, YOLOv3’s pre-trained weights and pipeline are used with the setting of batch size 8 and epochs 100. TensorFlow 2.x was used for running YOLO based model. figure 17 shows that the loss value goes down from 179 to 0.0778.

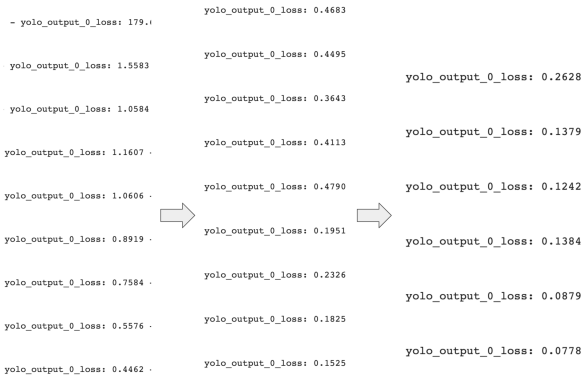

Fig. 17. YOLO v3: Loss Values

Further, SSD mobile net is also utilized. the total steps were set to 100000, and the evaluation steps were set to 50. TensorFlow 1.x was used for running SSD based model. The figure 18 shows that the loss value is converging as the steps go up.

By comparing the results, the YOLO’s curve is more stably decreasing than SSD. More training results will be updated in future work.

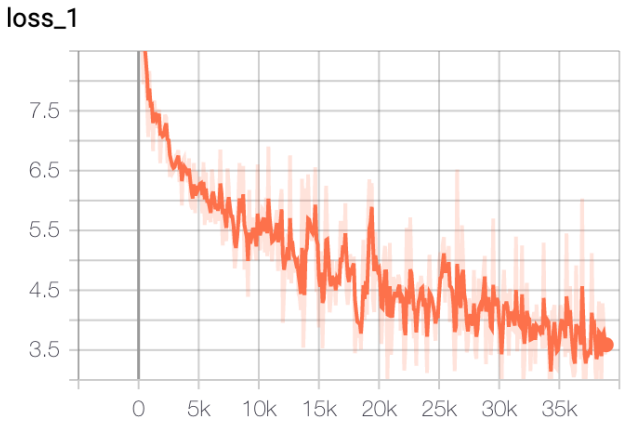

Fig. 18. SSD Mobile Net: Loss Curve

VI. Warning System Built with GPS Informing

The model is created based on the pre-trained files with weights provided by YOLO. The model is first defined based on the structure shown in the figure 19. After creating the model with this structure, the weights from YOLO pre-trained files are loaded to the model. Then, the model is saved as a .h5 file.

As the figure 20 shows, the program can recognize objects like the person, including the probability of this class. The logic of the system is reading the result from the model, converting the label to the human-readable version, which can be seen as a decoding process, and triggering the GPS informing system when the label matches with ”person”.

```
def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))

    # Layer  0 => 4
    x = _conv_block(input_image,
                    [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 0},
                     {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1},
                     {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2},
                     {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])

    # Layer  5 => 8
    x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
                        {'filter': 64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 6},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])

    # Layer  9 => 11
    x = _conv_block(x, [{'filter': 64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 10}])

    # Layer 12 => 15
    x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
                        {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 13},
                        {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 14}])

    # Layer 16 => 36
    for i in range(7):
        x = _conv_block(x, [
            {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 16 + i * 3},
            {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 17 + i * 3}])

    skip_36 = x

    # Layer 37 => 40
    x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 37},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 38},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 39}])
```

Fig. 19. Model Structure

```
person 83.38873982429504
Person is recognized, show the GPS info
person 63.007354736328125
Person is recognized, show the GPS info
skateboard 75.75014233589172
figName: 000.jpg
[(1, 13, 13, 255), (1, 26, 26, 255), (1, 52, 52, 255)]
```

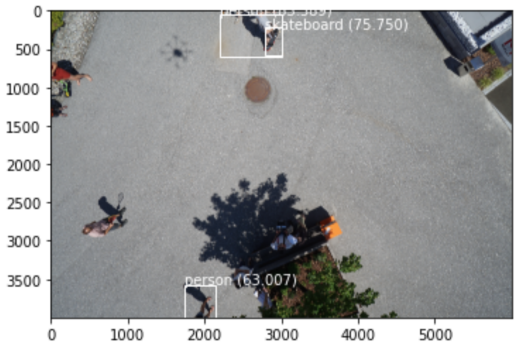

Fig. 20. Warning System

VII. Challenges

A. Data preprocessing

When working with TensorFlow Object Detection API, the readme (tutorial) file provided at Github is out of date, so there are some Tensorflow version problems between TF1 and TF2, which was fixed by modifying the code with specifying the version with "import tensorflow.compat.v1 as tf".

Another challenge occurred when drawing the bounding box. Multiple labels were drawn on one image and successfully drawn using PIL and cv2. As pyplots are not close properly, redundant labels are drawn at images densely after the whole iterations. After closing it properly, this problem was solved.

VIII. Conclusion

In conclusion, a novel human detection methodology was presented based on deep convolutional neural network with UAV imagery. The approach combines the deep learning models with the IoT devices, such as the raspberry pi which had the GPS sensor and the camera sensor plugged into it. The project also considered the occasion of tracking objects with night vision, so the infrared are configured and tested in the project. An initial system which implements warning built with GPS informing triggered by the specific result from the model of object detection.

Further, both daytime and night infrared imaging dataset taken from a high-altitude downward angle are created. Training with weights of YOLO v3 and SSD mobile net was done with ADH dataset. More training on our own dataset need to be done in future work, and the model needs to be converted to the TensorFlow Lite version which is compatible with Raspberry Pi's 32 bit operating system. After the deployment, a demo video is expected to be recorded.

References

[1] "Crimes that happen while you sleep," Nov 2020. [Online]. Available: https://www.thesleepjudge.com/crimes-that-happen-while-you-sleep/

[2] W. Andrew, C. Greatwood, and T. Burghardt, "Aerial animal biometrics: Individual friesian cattle recovery and visual identification via an autonomous uav with onboard deep inference," *arXiv preprint arXiv:1907.05310*, 2019.

[3] S. MohaimenianPour and R. Vaughan, "Hands and faces, fast: mono-camera user detection robust enough to directly control a uav in flight," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 5224–5231.

[4] A. Kouris, C. Kyrkou, and C.-S. Bouganis, "Informed region selection for efficient uav-based object detectors: altitude-aware vehicle detection with cycar dataset," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 51–58.

[5] P. Kannadaguli, "Yolo v4 based human detection system using aerial thermal imaging for uav based surveillance applications," in *2020 International Conference on Decision Aid Sciences and Application (DASA)*. IEEE, 2020, pp. 1213–1219.

[6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[7] A. Wong, M. J. Shafiee, F. Li, and B. Chwyl, "Tiny SSD: A tiny single-shot detection deep convolutional neural network for real-time embedded object detection," *Proceedings - 2018 15th Conference on Computer and Robot Vision, CRV 2018*, pp. 95–101, 2018.

[8] L. B. Das, A. Lijiya, G. Jagadanand, A. Aadith, S. Gautham, V. Mohan, S. Reuben, and G. George, "Human target search and detection using autonomous uav and deep learning," in *2020 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*. IEEE, 2020, pp. 55–61.

[9] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*. Ieee, 2017, pp. 1–6.

[10] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[11] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," *arXiv preprint arXiv:1602.07360*, 2016.

[12] A. Sengupta, Y. Ye, R. Wang, C. Liu, and K. Roy, "Going deeper in spiking neural networks: Vgg and residual architectures," *Frontiers in neuroscience*, vol. 13, p. 95, 2019.

[13] Z. Zhong, L. Jin, and Z. Xie, "High performance offline handwritten chinese character recognition using googlenet and directional feature maps," in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 846–850.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 1, pp. 142–158, 2015.

[15] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[16] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.
[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
[18] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.
[19] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," in *International conference on artificial neural networks*. Springer, 2018, pp. 270–279.
[20] "Raspberry pi 3 model b," last accessed 30 March 2021. [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b/
[21] "Adafruit ultimate gps breakout," last accessed 30 March 2021. [Online]. Available: https://www.adafruit.com/product/746
[22] "Pi noir camera v2," last accessed 30 March 2021. [Online]. Available: https://www.raspberrypi.org/products/pi-noir-camera-v2/
[23] "Aerial semantic segmentation drone dataset," last accessed 8 May 2021. [Online]. Available: http://dronedataset.icg.tugraz.at/
[24] "tzutalin/labelImg: LabelImg is a graphical image annotation tool and label object bounding boxes in images." [Online]. Available: https://github.com/tzutalin/labelImg
[25] W. McKinney *et al.*, "pandas: a foundational python library for data analysis and statistics," *Python for high performance and scientific computing*, vol. 14, no. 9, pp. 1–9, 2011.
[26] L. Shuangfeng, "Tensorflow lite: On-device machine learning framework," *Journal of Computer Research and Development*, vol. 57, no. 9, p. 1839, 2020.
[27] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.