CS6811 – Project Work

Second Review

UAV-based post-disaster 3D scene reconstruction for efficient survivor detection

Team Number: 17 (Batch 1)

Project Guide: Dr. R. Gunasekaran

Team Members:

Abhishek Manoharan	Nishanthini S	Vamsi Raju M
2019503502	2019503537	2019503568

Domains: Unmanned Aerial Vehicles, Computer Vision, Ensemble Network, 3D Reconstruction

Problem Definition:

Post-disaster scene understanding frameworks are becoming increasingly crucial in search and rescue operations and damage assessment initiatives. The use of Unmanned Aerial Vehicles (UAVs) provides an efficient method to complete the task of scene understanding. However, complex environments present in post-disaster scenarios make it difficult for UAVs to detect humans or objects accurately. Moreover, inefficient object detection mechanisms lead to low accuracy and a long time for object detection tasks. Hence, to mitigate these issues, we propose a UAV-based scene understanding scheme involving a GAN-aided 3D reconstruction mechanism. This approach deploys a Generative Adversarial Network (GAN)-based model to denoise and remove occlusion in the images obtained from the UAVs. The framework classifies objects present in the visual scope of the UAV using a 3D reconstruction of the images obtained from the UAV, followed by semantic segmentation, resulting in pixel-level prediction and classification of entities present in the 3D model. Furthermore, an ensemble network consisting of a combination of single-stage and multi-stage detectors is to be used to improve the performance of the survivor detection model. This will help reduce the false negative rate and improve the system's overall accuracy.

Literature Survey:

S.No	Publication Venue and Year	Title	Proposed Work	Limitations
1.	*IEEE Journal on Miniaturization for Air and Space Systems*, vol. 2, no. 4, pp. 209-219 **(2021)**	**UAV-Based Real-Time Survivor Detection System in Post-Disaster Search and Rescue Operations**	❖ This paper proposes a new thermal image dataset consisting of 6447 thermal images designed for survivor detection using UAVs in post-disaster scenarios. ❖ The paper also describes optimal values to prune survivor detection models in order to reduce the complexity of the models. ❖ The model applies knowledge distillation techniques to fine-tune them and improve accuracy. ❖ The performance of several survivor detection models based on YOLOv3 and YOLOv3-MobileNetV1 were compared with and without pruning and fine-tuning.	Older and inferior detection models have been used for survivor detection, thereby resulting in models with high mean average precision (mAP) loss and low accuracy.
2.	*IEEE Transactions on Geoscience and Remote Sensing*, vol. 60, pp. 1-16 **(2022)**	**Swarm UAV SAR for 3-D Imaging**	❖ This paper implements a 3D imaging mechanism for 2D images obtained from a swarm of UAVs. ❖ The proposed work involves the 3D imaging of a scene by	❖ A considerable amount of data must be transmitted from the UAV swarm, as images obtained from each node in

			the usage of 2D images obtained from several UAVs present in the UAV Swarm at different perspectives with a few points of overlap. ❖ The point cloud obtained is then triangulated, and Bundle Adjustment is used to create the 3D rendering of the image.	the swarm are used to produce the 3D rendering. ❖ Multiple UAVs also need to exchange information in order to efficiently collect data of the scenario.
3.	*IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, no. 11, pp. 8879-8889 **(2019)**	**Method for 3-D Scene Reconstruction Using Fused LiDAR and Imagery From a Texel Camera**	❖ In order to create greater fidelity terrain models, this study describes a bundle adjustment technique for aerial texel images. ❖ The model enables relatively low-accuracy navigation systems to be employed with inexpensive LiDAR and camera data.	Outliers present in the point cloud are not identified and mitigated, thereby leading to lower accuracy.
4.	*25th IEEE International Conference on Pattern Recognition (ICPR)*, pp. 10227-10234 **(2021)**	**SyNet: An ensemble network for object detection in UAV images**	❖ With the goal of lowering the high false negative rate of multi-stage detectors and improving the quality of the single-stage detector proposals, the authors of this research propose an ensemble network called SyNet that combines a multi-	❖ According to the investigation, detecting objects in drone images is more challenging than detecting them in images that were taken from the ground, even

			stage method with a single-stage one.	with the most advanced object detection algorithms. ❖ Hence, the accuracy of the model trained on UAV images is still low compared to models trained on ground images.
5.	*IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 11, pp. 6047-6067 **(2022)**	**Vehicle Detection From UAV Imagery With Deep Learning**	❖ The article provides a review of vehicle detection from UAV imagery using deep learning techniques. ❖ It begins by outlining the various deep learning architectures, including ➢ generative adversarial networks ➢ autoencoders ➢ recurrent neural networks ➢ convolutional neural networks and their contributions to the challenge of improving vehicle detection. ❖ The paper then focuses on examining various vehicle detection techniques	Videos captured in the UAVs are sent to on-ground workstations or to the cloud for processing rather than being implemented on the UAV itself, thereby leading to the absence of a lightweight system for vehicle detection.

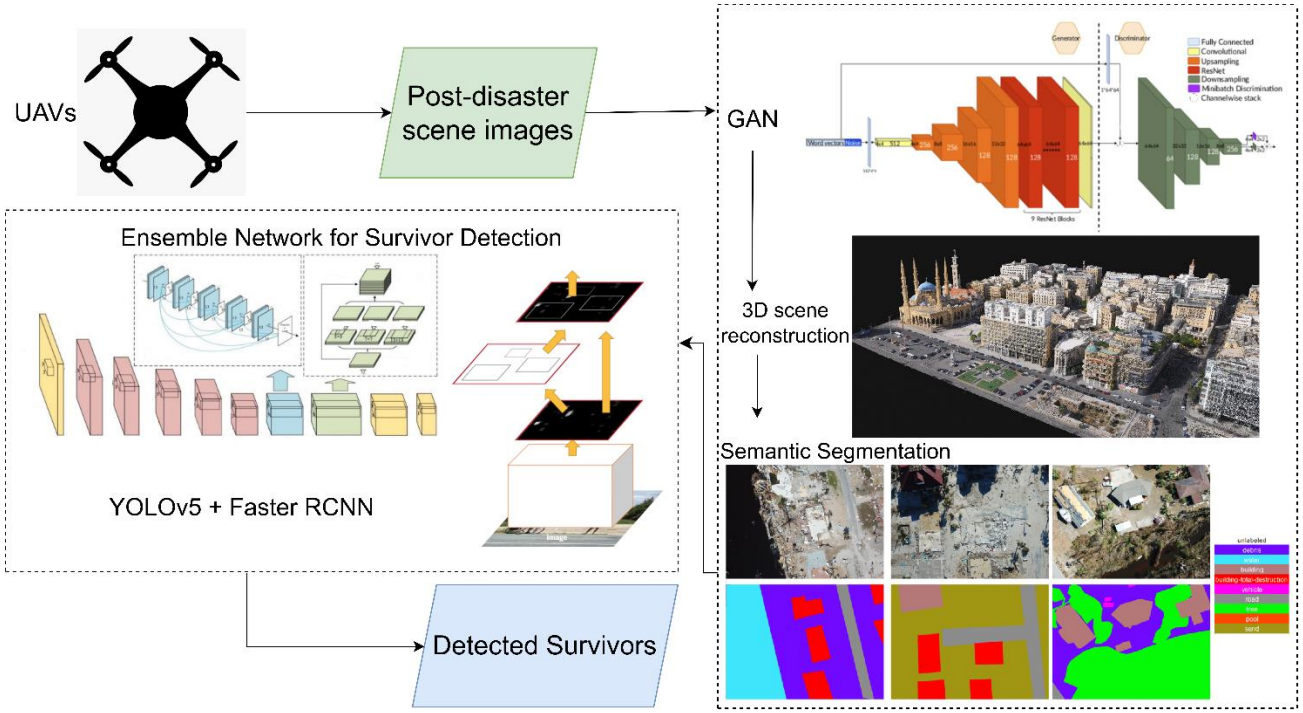
			and presents different benchmark datasets and problems that have been discovered, along with possible remedies.	
6.	*IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1-13 **(2022)**	**Dense and Small Object Detection in UAV-Vision Based on a Global-Local Feature Enhanced Network**	❖ This paper proposes a global-local feature-enhanced network (GLF-Net) to alleviate issues when detecting small and dense objects using UAVs. ❖ A feature-fusion module has been proposed to tackle the presence of numerous small objects. ❖ GLF-Net achieves 86.52% mean Average Precision (mAP) on the RO-UAV dataset.	The scalability of the framework is poor, and the application of GLF-Net on post-disaster UAV images leads to lower mAP, thereby requiring better frameworks.
7.	*IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1223-1232 **(2021)**	**Unmanned aerial vehicle visual detection and tracking using deep neural networks: A performance benchmark**	❖ This paper executes and compares various UAV detection mechanisms using air-borne UAVs that deploy deep neural networks. ❖ 4 datasets have been used and performance has been compared, namely MAV-VID, Drone-vs-Bird, Anti-UAV RGB, and Anti-UAV IR. ❖ The performance of 4 models was compared using the datasets mentioned, namely	❖ Long-distance detection of small UAVs was not taken into consideration. ❖ Deep neural networks for re-identification of UAVs were not considered as well.

			Faster RCNN, SSD512, YOLOv3, and DETR (Detection Transformer). Overall, Faster RCNN performed best.	
8.	*UMBC Student Collection* **(2021)**	**RescueNet: A High-Resolution Post Disaster UAV Dataset for Semantic Segmentation**	❖ This paper introduces a high-resolution post-disaster UAV dataset named RescueNet, which contains comprehensive pixel-level annotation of 11 classes for semantic segmentation to assess damage after a natural disaster. ❖ The dataset collection and annotation process are discussed, along with the challenges it poses.	❖ RescueNet contains a small number of classes. ❖ As a result, smaller objects like "vehicles" and "pools" make it difficult to get a good segmentation compared to larger objects like buildings and roads. ❖ Besides that, since UAV images include only the top view of a scene, it is difficult to assess the actual damage since the horizontal view also brings information regarding all sides of a building.
9.	IEEE/CVF Conference on Computer Vision and	**Uav-human: A large benchmark for human**	❖ This paper proposes a UAV-Human dataset for human action,	❖ The UAV-Human dataset poses a limitation for

	Pattern Recognition (CVPR), pp. 16266-16275 **(2021)**	**behavior understanding with unmanned aerial vehicles**	pose, and behavior understanding. ❖ The proposed UAV-Human contains ➢ 67,428 multi-modal video sequences ➢ 119 subjects for action recognition ➢ 22,476 frames for pose estimation ➢ 41,290 frames ➢ 1,144 identities for person re-identification ➢ 22,263 frames for attribute recognition which encourages the exploration and deployment of various data-intensive learning models for UAV-based human behavior understanding.	attribute recognition because the dataset is captured over a relatively long period of time. ❖ As a result, the subjects have been diversified with different dressing types and large variations of viewpoints caused by multiple UAV altitudes.
10.	*2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pp. 2325-2328 **(2021)**	**Attention-Based Semantic Segmentation on UAV Dataset for Natural Disaster Damage Assessment**	❖ This paper proposes and evaluates a novel self-attention segmentation model named ReDNet on a new high-resolution UAV natural disaster dataset named HRUD. ❖ The challenges of semantic segmentation on the HRUD dataset are discussed, along with the excellent	❖ HRUD is a very challenging dataset due to its variable-sized classes along with similar textures among different classes. ❖ Debris, textures of debris, sand, and building

			performance of the proposed model.	with destruction damage make a great impact on the segmentation performance of the evaluated network models.

System Architecture:



Novelty:

A GAN denoiser results in images having lower occlusion and optimal brightness, thereby highlighting the important features of the object. Furthermore, using GAN improves the detection of small and dense objects, which is the case of survivors in images obtained from a UAV. A 3D reconstruction of the scene using the enhanced images obtained from the GAN-based model will be used to map and extract useful information from the scene. The GAN-based 3D reconstruction framework incorporating the Structure-From-Motion (SFM) and bundle adjustment algorithms enhances occluded survivors in the 3D model, thereby resulting in a more efficient survivor detection

mechanism compared to existing frameworks. Semantic segmentation on the 3D model leads to a pixel-level prediction of various entities or objects present in the image. The ensemble model, a hybrid architecture consisting of single-stage and multi-stage detectors, overcomes the disadvantages of both frameworks. Deploying an ensemble network comprising the CenterNet and Cascade R-CNN frameworks improves the performance and efficiency of survivor detection. The overall framework will increase the accuracy and performance of the survivor detection task, thereby resulting in efficient Search-And-Rescue operations.

Expected Outcomes:

- To develop an efficient post-disaster scene understanding framework using UAVs for survivor detection and Search-And-Rescue (SAR) operations.
- To deploy a Generative Adversarial Network (GAN) framework to improve the detection of survivors, who are generally present as small and dense objects in post-disaster UAV images. A GAN denoiser will result in images having lower occlusion and optimal brightness, thereby highlighting the important features of the object.
- To devise a 3D scene-reconstruction mechanism based on the Structure-From-Motion (SFM) and bundle adjustment algorithms using images obtained from a swarm of UAVs to map and extract useful information from the scene.
- To deploy a semantic segmentation mechanism on the 3D model, leading to a pixel-level prediction of various entities or objects in the 3D model. This will improve the detection of survivors present in the post-disaster scene.
- To implement a hybrid single-stage and multi-stage ensemble network for survivor detection on the previously obtained semantic entities. The model will comprise the CenterNet and Cascade R-CNN mechanisms to combine the benefits of both, thereby decreasing the high false negative rate of multi-stage mechanisms and improving the performance of single-stage detectors.

Modules identified:

Module 1: Ensemble Model

Hybrid single-stage and multi-stage survivor detection model for efficient Search-And-Rescue operations.

Module 2: Semantic Segmentation

Semantic segmentation mechanism for pixel-level prediction and entity classification.

Module 3: GAN-infused 3D reconstruction mechanism

GAN-based denoiser and occlusion removal mechanism for better survivor detection from UAV images. 3D scene reconstruction framework that creates a 3D model based on the enhanced images obtained from the GAN mechanism, incorporating the Structure-From-Motion (SFM) and bundle adjustment algorithms.

Overall Performance Evaluation Approach:

Three model execution pipelines will be implemented and compared for performance, namely:

1. Regular 2D images passed through an Ensemble network comprising single-stage and multi-stage networks for survivor detection
2. Regular 2D images pre-processed through the Semantic Segmentation module, followed by the Ensemble network-based survivor detection model.
3. 2D images passed through a GAN-infused 3D reconstruction mechanism to obtain a 3D model of the post-disaster scene, upon which semantic segmentation is executed. The Ensemble network-based survivor detection model will be implemented on the output obtained from the previous step.

Algorithms:

1. **Algorithm 1: Object Detection using a hybrid Single-Stage and Multi-Stage Ensemble Model**

   Input:

   - **single_stage_model**: A PyTorch model trained with a single-stage detector architecture.

- **multi_stage_model**: A PyTorch model trained with a multi-stage detector architecture.
- **images**: A list of images to detect objects in.
- **conf_threshold**: A confidence threshold below which predictions are suppressed.
- **iou_threshold**: An IoU threshold above which overlapping predictions are suppressed.

Output:

- **results**: A list of dictionaries containing the detection results for each image.

Procedure:

1. Initialize an empty list of results.
2. For each image in images, do the following:
    a) Use the single-stage model to detect objects in the image and store the detection results in a list 'single_stage_results'.
    b) Use the multi-stage model to detect objects in the image and store the detection results in a list 'multi_stage_results'.
    c) Concatenate the detection results from the single-stage and multi-stage models into a single tensor.
    d) Apply non-maximum suppression to the tensor with confidence threshold conf_threshold and IoU threshold iou_threshold.
    e) Convert the detection results to a dictionary format with keys 'boxes', 'scores', and 'labels', and append the dictionary to results.
3. Return the results list.

2. <u>Algorithm 2: Semantic Segmentation</u>

Input:

- **model**: A PyTorch model trained for semantic segmentation.
- **images**: A list of images to perform segmentation on.
- **batch_size**: An integer specifying the batch size for inference.

Output:

- **results**: A list of dictionaries containing the segmentation results for each image.

Procedure:

1. Initialize an empty list **results**.
2. For each batch of size **batch_size** in **images**, do the following:

a) Preprocess the images by normalizing and converting them to tensors.
b) Feed the batch of images to the **model** and obtain the segmentation results.
c) Convert the segmentation results to a dictionary format with keys 'mask' and 'class', and append the dictionary to **results**.

3. Return the **results** list.

3. Algorithm 3: GAN-based Occlusion Remover

Input:

- **generator**: A PyTorch generator model trained to remove occlusions.
- **images**: A list of images with occlusions.
- **batch_size**: An integer specifying the batch size for inference.

Output:

- **results**: A list of images with occlusions removed.

Procedure:

1. Initialize an empty list **results**.
2. For each batch of size **batch_size** in **images**, do the following:
   a) Preprocess the images by resizing, normalizing and converting them to tensors.
   b) Feed the batch of images to the **generator** and obtain the occlusion-removed images.
   c) Convert the occlusion-removed images back to numpy arrays and append them to **results**.
3. Return the **results** list.

Preprocessing:

- Resize the input images to the required size for the **generator** model.
- Normalize the pixel values of the images to be in the range [-1, 1].
- Convert the images to tensors.

Occlusion Removal:

- Feed the pre-processed batch of images to the **generator** model.
- The generator model generates occlusion-free versions of the input images.
- Convert the generated occlusion-free images back to numpy arrays.

4. **Algorithm 4: 3D Reconstruction of 2D Images using Structure-From-Motion**

Input:

- **images**: A set of 2D images of an object from different views.
- **camera_params**: The camera parameters used to capture the images.
- **matching_threshold**: The threshold for feature matching.
- **reconstruction_threshold**: The threshold for 3D point reconstruction.
- **bundle_adjustment_iterations**: The number of iterations for bundle adjustment.

Output:

- **point_cloud**: A 3D point cloud of the object.

Procedure:

1. Extract feature points from each image using a feature detector (SIFT).
2. Match the feature points between image pairs using a feature descriptor (ORB) and a matching algorithm (FLANN).
3. Estimate the fundamental matrix for each image pair using the matched feature points.
4. Estimate the essential matrix for each image pair using the camera parameters and the fundamental matrix.
5. Compute the relative camera poses (i.e., rotation and translation) for each image pair using the essential matrix and the camera parameters.
6. Triangulate the 3D points using the relative camera poses and the matched feature points.
7. Perform bundle adjustment to refine the camera poses and the 3D points using all images and their matched feature points.
8. Filter the 3D points based on the reconstruction threshold to remove outliers.
9. Return the 3D point cloud.

Feature Detection and Matching:

- Feature detection is the process of identifying distinctive image features, such as corners or blobs, that can be matched between images.
- Feature matching is the process of finding corresponding feature points between image pairs.

Fundamental Matrix and Essential Matrix:

- The fundamental matrix describes the geometric relationship between two views of a scene, given the corresponding image points.

- The essential matrix relates the camera poses between two views of a scene, given the intrinsic camera parameters and the fundamental matrix.

Triangulation:

- Triangulation is the process of computing the 3D position of a point in space, given its projections onto multiple images and the camera poses.

Bundle Adjustment:

- Bundle adjustment is a nonlinear optimization algorithm that jointly refines the camera poses and the 3D points to minimize the reprojection error.

Implementation:

Ensemble Model:

- An ensemble model comprising the YOLOv5 single-stage detector and the Faster RCNN multi-stage detector was implemented.
- The main advantage of the proposed combination of single-stage and multi-stage detectors is the improved object detection performance over an independent single-stage detector and lower false negative rate than an independent multi-stage detector.
- Our choice of CNN mechanisms: YOLOv5, being one of the most popular and powerful single-shot detectors, is a stable version of the 'You Only Look Once' family of CNN-based object detection models. Faster RCNN is a very powerful two-stage detector that is widely used for object detection mechanisms.

YOLOv5:

- The YOLOv5 model is instantiated by downloading all dependencies from the official 'ultralytics' github repository.

Installing Dependencies

```
# clone YOLOv5 repository
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
!git reset --hard 064365d8683fd002e9ad789c1e91fa3d021b44f0
```

```
Cloning into 'yolov5'...
remote: Enumerating objects: 15529, done.
remote: Counting objects: 100% (136/136), done.
remote: Compressing objects: 100% (93/93), done.
remote: Total 15529 (delta 49), reused 119 (delta 43), pack-reused 15393
Receiving objects: 100% (15529/15529), 14.59 MiB | 13.31 MiB/s, done.
Resolving deltas: 100% (10577/10577), done.
/content/gdrive/My Drive/Final_Year_Project/YOLOv5/yolov5
HEAD is now at 064365d Update parse_opt() in export.py to work as in train.py (#10789)
```

```
[ ]  # install dependencies as necessary
!pip install -qr requirements.txt  # install dependencies (ignore errors)
import torch

from IPython.display import Image, clear_output  # to display images
from utils.downloads import attempt_download  # to download models/datasets

# clear_output()
print('Setup complete. Using torch %s %s' % (torch.__version__, torch.cuda.get_device_properties(
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 184.3/184.3 kB 13.7 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 7.5 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.6/1.6 MB 78.8 MB/s eta 0:00:00
Setup complete. Using torch 2.0.0+cu118 _CudaDeviceProperties(name='Tesla T4', major=7, minor=5, t
```

- Our dataset of choice, namely RescueNet, is annotated and augmented using the Roboflow software. Additional pre-processing techniques are incorporated within the Roboflow workspace, and a unique api key is generated for our dataset. This api key is then used to download the dataset and the respective annotations in PyTorch YOLOv5 format.

Downloading Formatted Custom Dataset

Formatted and Augmented Post disaster dataset is downloaded from Roboflow using the "**YOLOv5 PyTorch**" export format.

```
[ ]  #follow the link below to get your download code from from Roboflow
!pip install -q roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="9KG0WhvnpHGdnlOuXdur")
project = rf.workspace("final-year-project-deylr").project("ensemble-3ipxs")
dataset = project.version(1).download("yolov5")
```

```
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in Ensemble-1 to yolov5pytorch: 86% [30875648 / 35596042] bytesExt
```

- The configuration for the YOLOv5 model is defined, wherein various parameters, including the number of classes, batch size, and the number of epochs. Furthermore, The head, backbone, and neck of YOLOv5 are implemented in this stage. The final configuration is then verified.

```
%cd ..
%cat models/yolov5s.yaml
```

```
/content/gdrive/MyDrive/Final_Year_Project/YOLOv5/yolov5
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license

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

# YOLOv5 v6.0 backbone
backbone:
  # [from, number, module, args]
  [[-1, 1, Conv, [64, 6, 2, 2]],  # 0-P1/2
   [-1, 1, Conv, [128, 3, 2]],  # 1-P2/4
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],  # 3-P3/8
   [-1, 6, C3, [256]],
```

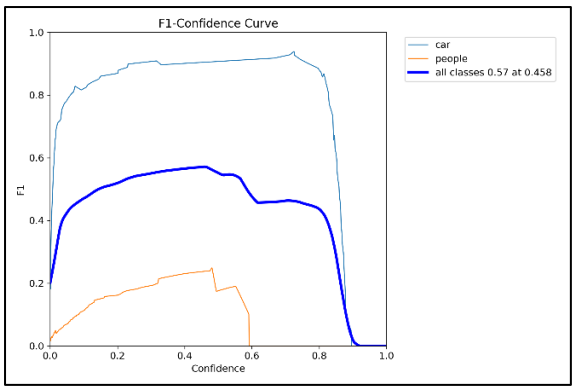
- The YOLOv5 model is then trained with batch size 16 for 492 epochs. The configurations previously defined in custom_yolov5s.yaml are also given as input parameters. In our case, Early stopping was observed as the model performance did not vary over a period of epochs, thereby stopping the training process in the 492nd epoch. The Mean Average Precision (mAP) is calculated for every epoch/iteration and displayed in the output. Finally, the total number of instances that the model visualized under each class is tabulated and presented, and the weights are stored in the yolov5s_results folder as ‘best.pt’.

```
      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
    490/999       1.7G    0.05502    0.02816   0.002194         33        416: 100% 14/14 [00:02<00:00,  5.13it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  3.55it/s]
                   all         10         60      0.681      0.534       0.53      0.264

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
    491/999       1.7G    0.05301     0.0274   0.001798         12        416: 100% 14/14 [00:03<00:00,  4.46it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  5.75it/s]
                   all         10         60      0.518      0.567      0.505      0.247

      Epoch    GPU_mem   box_loss   obj_loss   cls_loss  Instances       Size
    492/999       1.7G    0.05517    0.03003   0.002003         23        416: 100% 14/14 [00:02<00:00,  6.20it/s]
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00,  7.62it/s]
                   all         10         60      0.539      0.555      0.498      0.246
Stopping training early as no improvement observed in last 100 epochs. Best results observed at epoch 392, best model saved as best.pt.
To update EarlyStopping(patience=100) pass a new patience value, i.e. `python train.py --patience 300` or use `--patience 0` to disable

493 epochs completed in 0.456 hours.
Optimizer stripped from runs/train/yolov5s_results/weights/last.pt, 14.8MB
Optimizer stripped from runs/train/yolov5s_results/weights/best.pt, 14.8MB

Validating runs/train/yolov5s_results/weights/best.pt...
Fusing layers...
custom_YOLOv5s summary: 182 layers, 7249215 parameters, 0 gradients
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 1/1 [00:00<00:00, 10.73it/s]
                   all         10         60      0.661      0.555       0.55      0.272
                   car         10         41       0.86      0.951      0.959      0.511
                people         10         19      0.462      0.158      0.142     0.0323
Results saved to runs/train/yolov5s_results
```

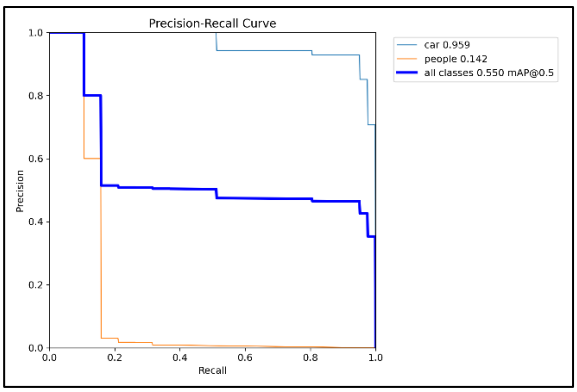
- Tensorboard is then made use of to visualize the performance of the model. Tesnorboard plots all performance metrics observed while training the model over the range of epochs for which the model was trained. The overall training loss of the model was found to be 0.002 for identifying various classes. The mean average precision (mAP) of the model after 492 epochs as found to be 0.55.

Results obtained:

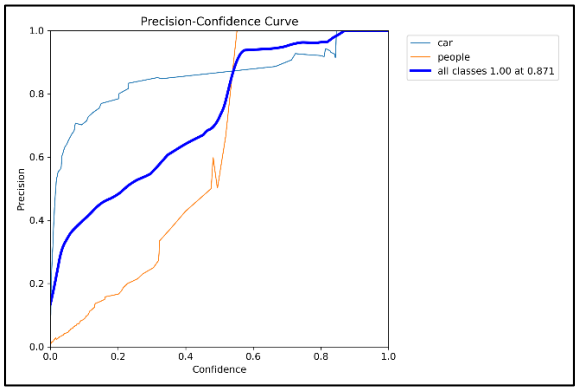
F1-confidence curve



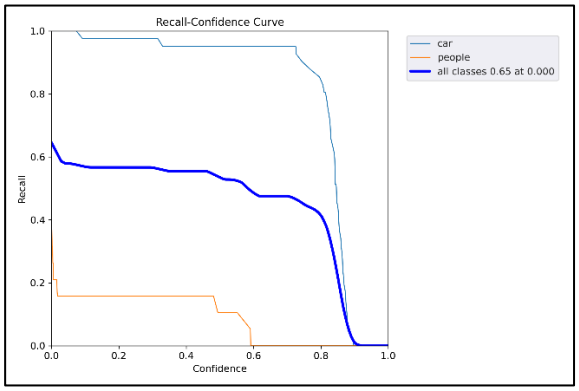
Precision-Recall curve



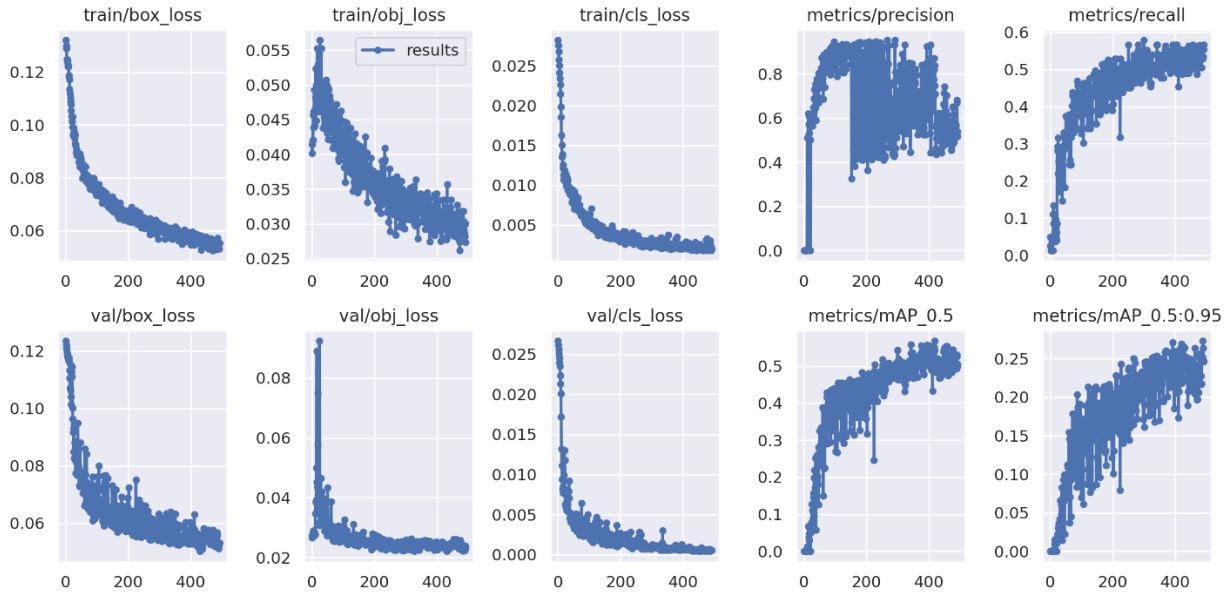
Precision-confidence curve



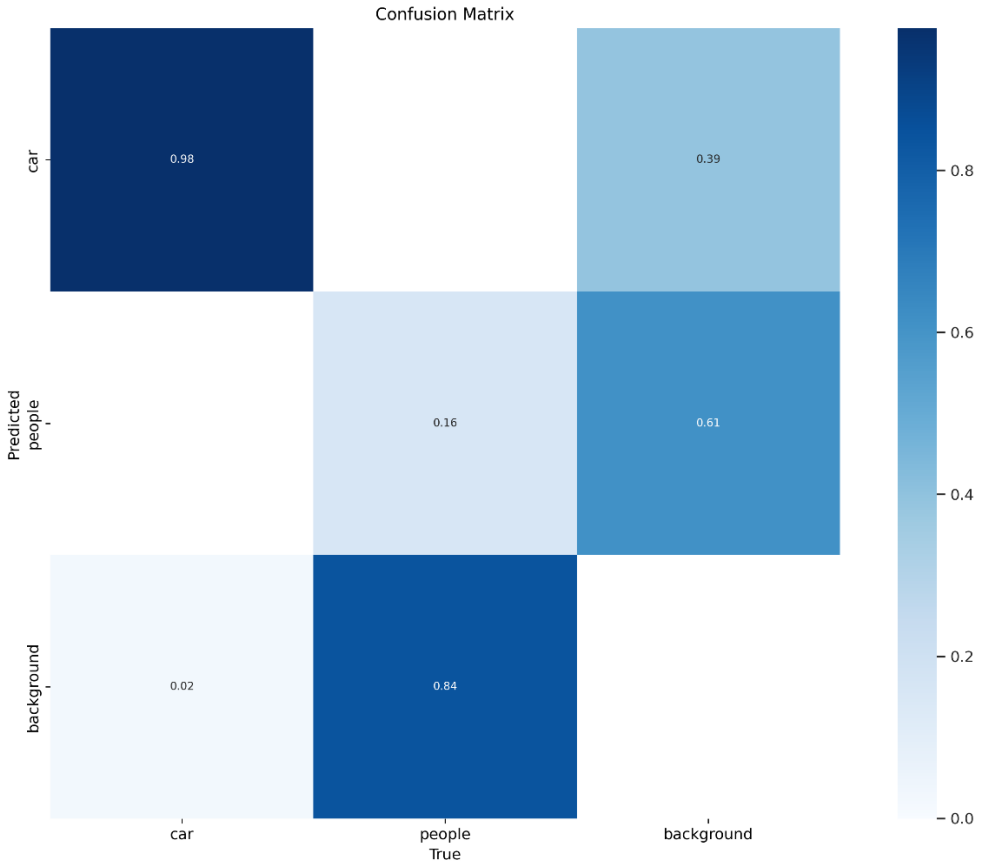
Recall-confidence curve



Performance Metrics:



Confusion Matrix:



- The trained model is then tested by visualizing results obtained when test images are passed through the model. The bounding boxes constructed over detected classes are displayed.

- The images present in the test split are then passed through the trained model loaded with the best weights obtained during the training process. The results are then obtained and visualized.

```
[ ]  # %cd /content/yolov5/
     !python detect.py --weights runs/train/yolov5s_results/weights/best.pt --img 416 --conf 0.4 --sou

     detect: weights=['runs/train/yolov5s_results/weights/best.pt'], source=Ensemble-1/test/images, dat
     YOLOv5 🚀 v7.0-72-g064365d Python-3.9.16 torch-2.0.0+cu118 CUDA:0 (Tesla T4, 15102MiB)

     Fusing layers...
     custom_YOLOv5s summary: 182 layers, 7249215 parameters, 0 gradients
     image 1/20 /content/gdrive/MyDrive/Final_Year_Project/YOLOv5/yolov5/Ensemble-1/test/images/10841_j
     image 2/20 /content/gdrive/MyDrive/Final_Year_Project/YOLOv5/yolov5/Ensemble-1/test/images/10845_j
     image 3/20 /content/gdrive/MyDrive/Final_Year_Project/YOLOv5/yolov5/Ensemble-1/test/images/10865_j
     image 4/20 /content/gdrive/MyDrive/Final_Year_Project/YOLOv5/yolov5/Ensemble-1/test/images/10900_j
     image 5/20 /content/gdrive/MyDrive/Final_Year_Project/YOLOv5/yolov5/Ensemble-1/test/images/10910_j
     image 6/20 /content/gdrive/MyDrive/Final_Year_Project/YOLOv5/yolov5/Ensemble-1/test/images/10911_j
```

- The model along with the best weights are then saved in a folder named 'Saved_Models' for it to be used and deployed in the final Ensemble model.

Faster RCNN:

- The Faster RCNN multi-stage CNN model is implemented using Detectron2, which is a computer vision library that allows the implementation of various CNN models through the usage of PyTorch.
- Initially, the GPU instantiated in the Google Colab environment is displayed using the nvidia-smi command.

```
[ ]  !nvidia-smi

     Sun Apr 16 12:23:21 2023
     +-----------------------------------------------------------------------------+
     | NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
     |-------------------------------+----------------------+----------------------+
     | GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
     | Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
     |                               |                      |               MIG M. |
     |===============================+======================+======================|
     |   0  Tesla T4            Off  | 00000000:00:04.0 Off |                    0 |
     | N/A   42C    P8    10W /  70W |      0MiB / 15360MiB |      0%      Default |
     |                               |                      |                  N/A |
     +-------------------------------+----------------------+----------------------+

     +-----------------------------------------------------------------------------+
     | Processes:                                                                  |
     |  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
     |        ID   ID                                                   Usage      |
     |=============================================================================|
     |  No running processes found                                                 |
     +-----------------------------------------------------------------------------+
```

- The dependencies of Detectron2 are installed to be able to setup Detectron2 and Faster RCNN. The versions of all dependencies are confirmed.

Installing Detectron2 and dependencies

```
[ ]  !python -m pip install 'git+https://github.com/facebookresearch/detectron2.git'
```

```
  Downloading omegaconf-2.3.0-py3-none-any.whl (79 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.5/79.5 kB 10.0 MB/s eta 0:00:00
Collecting hydra-core>=1.1
  Downloading hydra_core-1.3.2-py3-none-any.whl (154 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 154.5/154.5 kB 20.0 MB/s eta 0:00:00
Collecting black
  Downloading black-23.3.0-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 79.8 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from det
Requirement already satisfied: numpy in /usr/local/lib/python3.9/dist-packages (from fvcore<
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-packages (from f
Collecting antlr4-python3-runtime==4.9.*
  Downloading antlr4-python3-runtime-4.9.3.tar.gz (117 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 117.0/117.0 kB 15.3 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
```

- All libraries required to implement the detectron2 environment and the Faster RCNN model are defined. The libraries required for dataset preparation, visualization of the performance of the model on test images, and training of the model are defined.

```
[ ]  # COMMON LIBRARIES
     import os
     import cv2

     from datetime import datetime
     from google.colab.patches import cv2_imshow

     # DATA SET PREPARATION AND LOADING
     from detectron2.data.datasets import register_coco_instances
     from detectron2.data import DatasetCatalog, MetadataCatalog

     # VISUALIZATION
     from detectron2.utils.visualizer import Visualizer
     from detectron2.utils.visualizer import ColorMode

     # CONFIGURATION
     from detectron2 import model_zoo
     from detectron2.config import get_cfg

     # EVALUATION
     from detectron2.engine import DefaultPredictor

     # TRAINING
     from detectron2.engine import DefaultTrainer
```

- The dataset previously annotated and augmented on Roboflow is downloaded along with annotations in the COCO JSON format. Roboflow being a versatile software allows exporting various annotation formats for the same dataset. The COCO JSON format is used while training a Faster RCNN model.

Formatting Survivor Detection Dataset to COCO JSON format

```
[ ]  !pip install roboflow

     from roboflow import Roboflow
     rf = Roboflow(api_key="9KG0WhvnpHGdnlOuXdur")
     project = rf.workspace("final-year-project-deylr").project("ensemble-3ipxs")
     dataset = project.version(1).download("coco")
```

- The dataset is then registered to detectron2 to be able to use the dataset for training the Faster RCNN model.

Confirming that the survivor detection dataset is correctly registered using MetadataCatalog

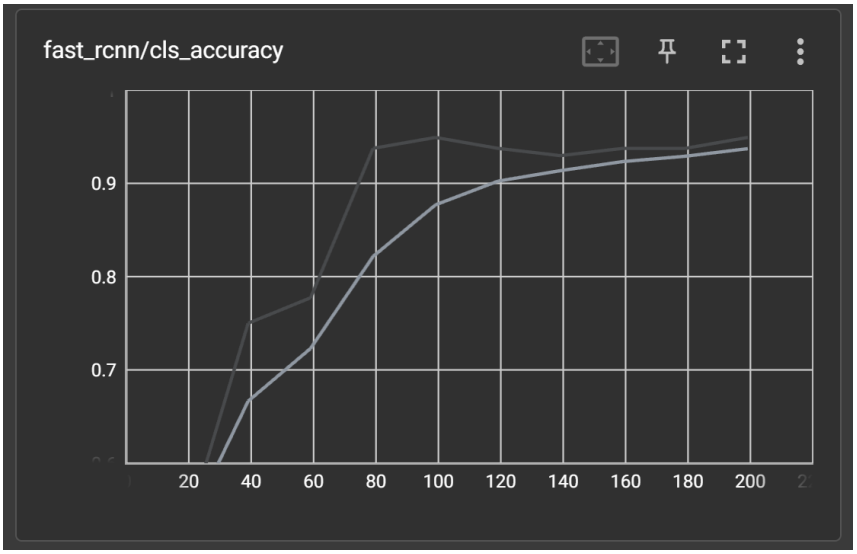
```
[ ]  [
         data_set
         for data_set
         in MetadataCatalog.list()
         if data_set.startswith(DATA_SET_NAME)
     ]

     ['Ensemble-train', 'Ensemble-test', 'Ensemble-valid']
```

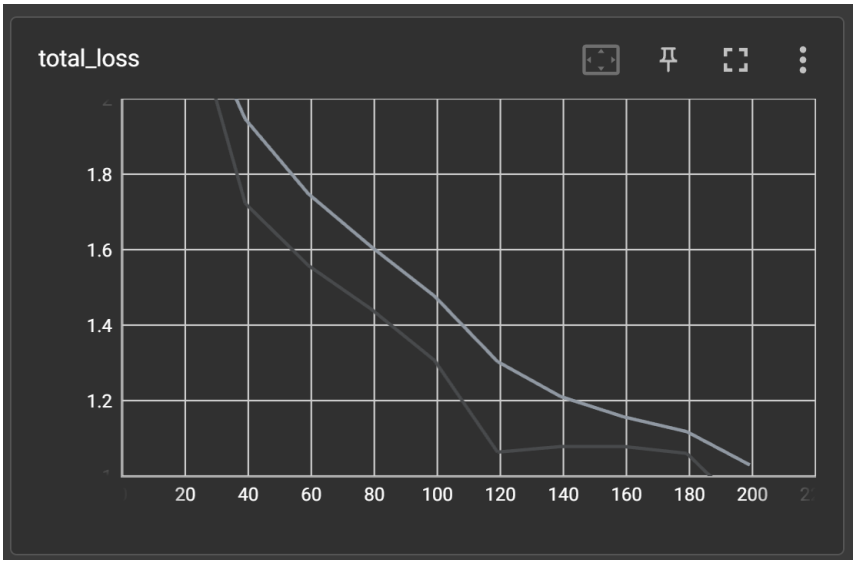
- The Faster RCNN model is then instantiated and trained using the faster_rcnn_X_101_32x8d_FPN_3x architecture present in Detectron2. Other parameters required for training the model are defined as well.

```
[ ]  # HYPERPARAMETERS
     ARCHITECTURE = "faster_rcnn_X_101_32x8d_FPN_3x"
     # ARCHITECTURE = "mask_rcnn_R_101_FPN_3x"
     CONFIG_FILE_PATH = f"COCO-Detection/{ARCHITECTURE}.yaml"
     MAX_ITER = 200
     EVAL_PERIOD = 200
     BASE_LR = 0.001
     NUM_CLASSES = 2

     # OUTPUT DIR
     OUTPUT_DIR_PATH = os.path.join(
         DATA_SET_NAME,
         ARCHITECTURE,
         datetime.now().strftime('%Y-%m-%d-%H-%M-%S')
     )

     os.makedirs(OUTPUT_DIR_PATH, exist_ok=True)
```

- Once the model is trained, Tesnorboard is initialized, thereby giving various performance metrics observed during the training process of the Faster RCNN model. Total loss of the model was observed to be 0.89. The accuracy of detecting various classes was found to be 93.75%.

Results obtained:





- The trained model is then tested using the test set and results are visualized.

- The model and its corresponding weights are then saved and stored in the 'Saved_Models' folder.

Final Ensemble Model:

- The final ensemble model is created using the two trained models obtained in the previous steps.
- Both the models are executed for the same image and their respective outputs are visualized.
- The models are then combined using the ensembling technique named 'simple averaging', wherein multiple models are trained independently and their predictions are combined to improve overall performance.

```python
# Detect objects in each image using each model
for image in images:
    detection_results = []

    for model in models:
        predictions = model.detect(image)
        detection_results.append(predictions)

    # Ensemble the detection results
    ensemble_predictions = torch.cat(detection_results, dim=0)
    ensemble_predictions = non_max_suppression(ensemble_predictions,

    # Convert the detection results to a dictionary format
    detection_dict = {'boxes': [], 'scores': [], 'labels': []}

    for bbox in ensemble_predictions:
        detection_dict['boxes'].append(bbox[0:4].tolist())
        detection_dict['scores'].append(bbox[5].item())
        detection_dict['labels'].append(int(bbox[6].item()))

    results.append(detection_dict)

return results
```

- The ensembled model is then evaluated using the same image used for the previous models and the outputs are compared.

References:

1. J. Dong, K. Ota and M. Dong, "**UAV-Based Real-Time Survivor Detection System in Post-Disaster Search and Rescue Operations**," in *IEEE Journal on Miniaturization for Air and Space Systems*, vol. 2, no. 4, pp. 209-219, 2021
2. H. Ren et al., "**Swarm UAV SAR for 3-D Imaging: System Analysis and Sensing Matrix Design**," in IEEE Transactions on Geoscience and Remote Sensing, vol. 60, pp. 1-16, 2022
3. T. C. Bybee and S. E. Budge, "**Method for 3-D Scene Reconstruction Using Fused LiDAR and Imagery From a Texel Camera**," in IEEE Transactions on Geoscience and Remote Sensing, vol. 57, no. 11, pp. 8879-8889, Nov. 2019
4. Albaba, Berat Mert, and Sedat Ozer, "**SyNet: An ensemble network for object detection in UAV images**," in 25th IEEE International Conference on Pattern Recognition (ICPR), pp. 10227-10234, 2021
5. A. Bouguettaya, H. Zarzour, A. Kechida and A. M. Taberkit, "**Vehicle Detection From UAV Imagery With Deep Learning: A Review**," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 11, pp. 6047-6067, Nov. 2022
6. T. Ye, W. Qin, Y. Li, S. Wang, J. Zhang and Z. Zhao, "**Dense and Small Object Detection in UAV-Vision Based on a Global-Local Feature Enhanced Network**," in *IEEE Transactions on Instrumentation and Measurement*, vol. 71, pp. 1-13, 2022
7. Rahnemoonfar, Maryam, Tashnim Chowdhury, and Robin Murphy. "**RescueNet: A High-Resolution Post Disaster UAV Dataset for Semantic Segmentation**." *UMBC Student Collection*, 2021
8. T. Chowdhury and M. Rahnemoonfar, "**Attention Based Semantic Segmentation on UAV Dataset for Natural Disaster Damage Assessment**," *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, pp. 2325-2328, 2021
9. Li, Tianjiao, Jun Liu, Wei Zhang, Yun Ni, Wenqian Wang, and Zhiheng Li. "**Uav-human: A large benchmark for human behavior understanding with unmanned aerial vehicles**." In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 16266-16275, 2021
10. Isaac-Medina, Brian KS, Matt Poyser, Daniel Organisciak, Chris G. Willcocks, Toby P. Breckon, and Hubert PH Shum. "**Unmanned aerial**

vehicle visual detection and tracking using deep neural networks: A performance benchmark." In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1223-1232, 2021.