# Assignment 3 - Moved Object Detection Using DETR

## Overview

- In this assignment you will build an **end-to-end DETR pipeline** to detect **objects that moved between two frames** of surveillance footage (i.e. the modified VIRAT dataset provided to you during Assignment 2)
- You will:
    1. Prepare and understand the dataset annotation format. You will also be provided with a preprocessing pipeline to create the annotations.
    2. Adapt and fine-tune a pre-trained DETR model (**facebook/detr-resnet-50**) using HuggingFace Transformers
    3. Evaluate your model with quantitative and qualitative visualizations
    4. Write a technical report that documents the entire pipeline from data to results

## Input Data

- The input data is procured from the VIRAT dataset and has been shared earlier with you during assignment 2. It contains pairs and their respective annotation ".txt" files from surveillance camera footage. These frames contain objects labelled: Unknown=0, person=1, car=2, other vehicle=3, other object=4, bike=5.
- Read more about the VIRAT dataset here: https://data.kitware.com/#item/56f578dc8d777f753209bdef
- In case you have not done so earlier, I would encourage you to read about this dataset to get an understanding of how you would parse the annotations for the respective pairs.

### Data Ground Truth Labeling

- You should use the data_ground_truth_labeller.py file provided to you to generate the matched annotations.
- It is advised that you go through the script on your own as well.
- You can change the way the annotations are written, and generate more ground truth labels by making edits to the data_ground_truth_labeller.py file, however, if you do so, you MUST include this information in your report and provide an explanation of the changes you made.

### Working of the data_ground_truth_labeller.py file

- The above mentioned labeller uses feature similarity and centroid distances for cost matrix calculation alongwith the Hungarian method of matching objects in 2 frames. After that, it only keeps objects that have significantly moved (i.e. with IoU = 0).
- The given script also generates some sample visualizations with 1 frame per pair.
- Note: The object IDs from the original annotation.txt files are not all correct, so for the annotation files for these matched GTs, I have replaced the object IDs with new object IDs (different from those in the actual Kitware dataset provided in the link).
- Each annotation file will have 2 rows per object. The first row represents the old coordinates of the object, and the second row represents the new ones.
- The format of each row is: **<object_id> <x> <y> <w> <h> <object_type>**. Here, **x, y, w, and h are the bounding box coordinates.**

# Assignment Task

Here, I will discuss the main component of the assignment — the method architecture options we can try implementing in the finetuning pipeline.

1. Since the DETR model provided has a ResNet50 block, we can pass the 2 images individually to the ResNet50 block and generate their feature representations. We can then take the difference of the 2 feature representations and feed this into the transformer block of the DETR model.
2. Take the pixel-wise image difference and feed it directly to the entire DETR block.

You can then proceed to finetune the DETR model after implementing either/both of these workflows.

## Implementational Information and Training

### Pipeline architecture

- For option 1, you can do 2 things:
    1. you can use your own Resnet model (say, the ImageNet ResNet50) and end it at an intermediate layer (for example - layer 3) prior to the fully-connected layers of the ResNet architecture. You can then run the images through the ResNet architecture and then feed it **directly to the** transformer block of the DETR model.
       IMPORTANT NOTE: if you feed the feature-representation difference of the 2 images to the entire DETR model/object then this will be considered **incorrect.**
    2. You can use a “forward hook” (read about this) to use the ResNet block present within the DETR model. This is more flexible, but may be technically more complicated. However, it is a more powerful implementation and lets you tap into any layer dynamically.
- For option 2, you need to take the pixel-wise image difference (think about whether all images are of the same size or not, what image processing would be required and why) and feed that into your DETR model.

### Fine-tuning methods

For either/both options that you implement, you must provide comparison studies between entire training runs for the following finetuning methods (at the very least):

1. Fine-tune all parameters (please note you don't need to finetune the convolutional parameters this is not required, if your architecture follows option 1)
2. Fine-tune only the convolutional block (you can finetune the entire convolutional block or a subset of layers in the convolutional block) (please note this is not required, if your architecture follows option 1)
3. Fine-tune only the transformer classification head (i.e. the fully-connected network at the end of the transformer used to make the classification)
4. Finetune only the transformer block (you can finetune the entire transformer block or a subset of layers in the transformer block)

### Note

It is important that you justify all design considerations you make in the model architecture choice and the fine-tuning methods used.

### Training and Evaluation

1. Use an 80-20 train-test split on the annotated and matched dataset (note that this doesn't include the size of the evaluation set and you can generate the evaluation set from the training samples)
2. Make sure to provide robust evaluation metrics - make sure to provide precision and recall. You may showcase other evaluation metrics as you deem appropriate.

# Submission Checklist

## Report

- Describe your approach, pipeline, which DETR was used, and the rationale behind the chosen approach (option i or ii).
- Include performance metrics, visual output/screenshots wherever helpful.
- Highlight any assumptions or workarounds you implemented.
- Ablation tests

## Code

- Turn in a zip file containing your program files. (do NOT include your data files and model tensors in your submissions)
- Keep the code modular (i.e. different files for dataloader, trainer, configs, evaluation)
- Make sure your zip file contains the SLURM batch scripts, SLURM output runs, and screenshots demonstrating that the code has been run by you
- Include a Readme file that contains descriptions on how to execute the program.