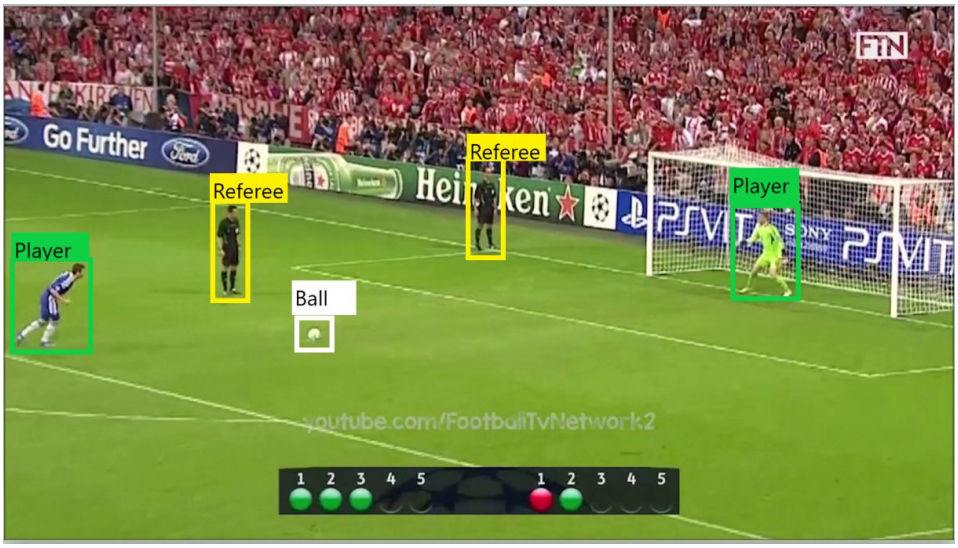
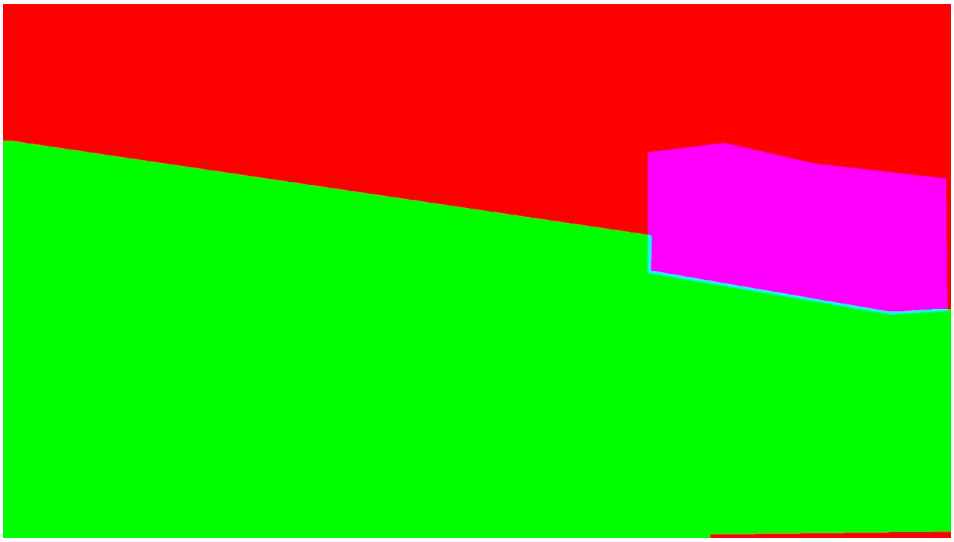
Dataset Generation:

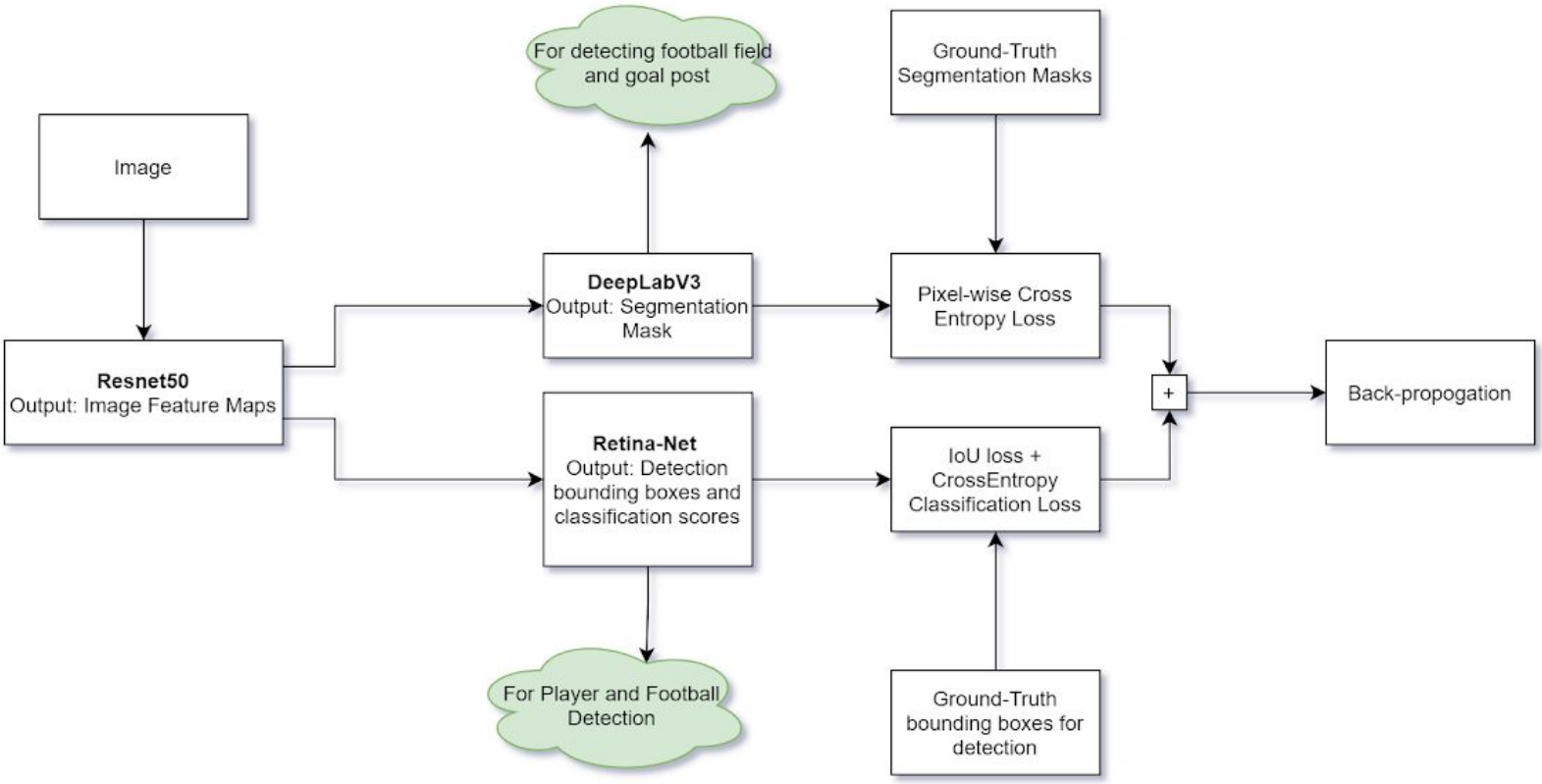
- Images were sampled at 7 frames per second from videos.
- 450 frames were selected out of sampled images.
- For each image, annotations included, semantic mask for goal and field, bounding box coordinates for ball, players and referee.
- Detection labeling was done using Labelimg in PascalVOC format.
- Mask Jsons were created using the Labelme annotation tool.



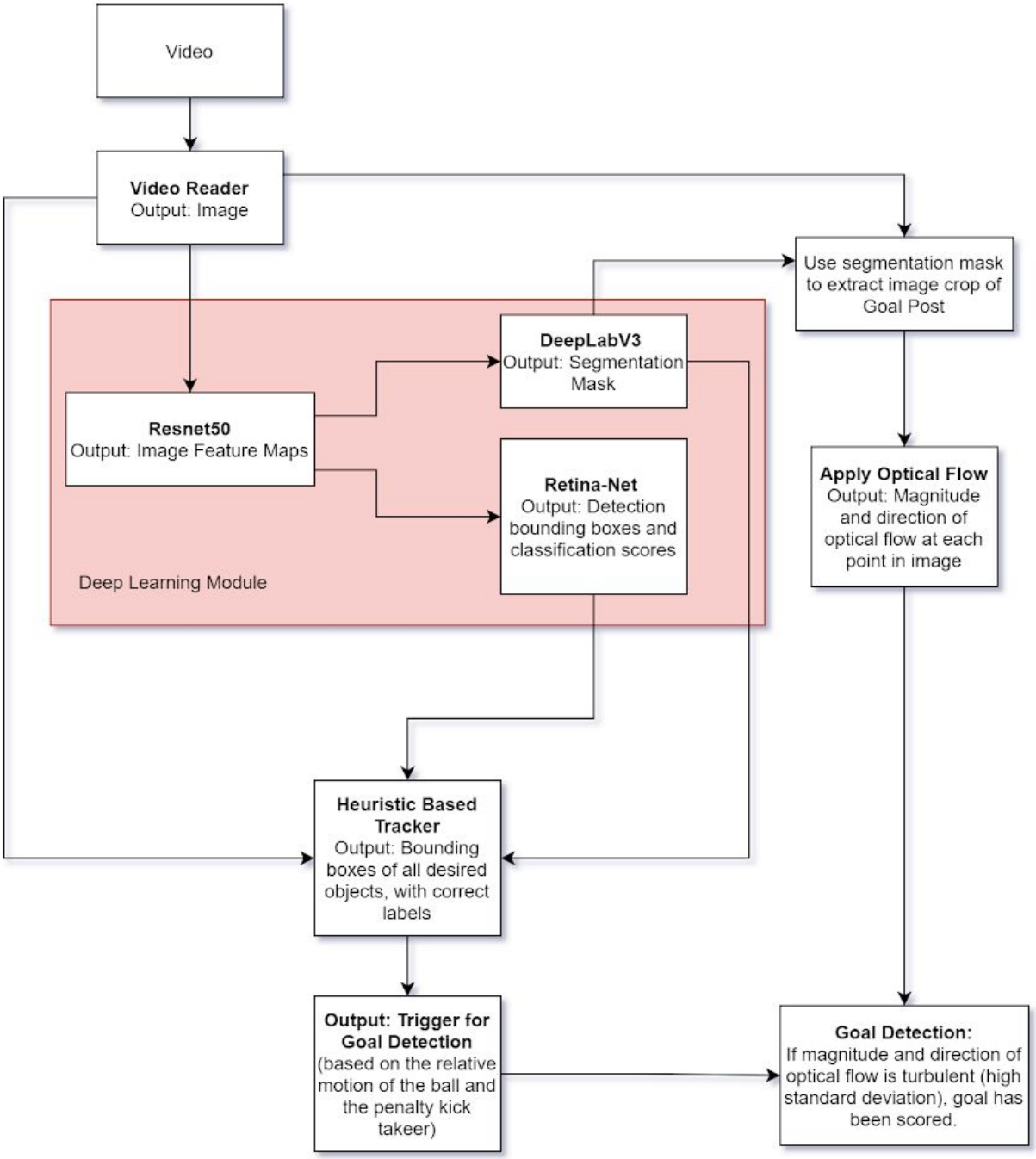


Architectures:

During Training:



During Inference:



Detailed Description of the Modules:

- **Resnet50:**

  Resnet50 is used as the backbone of the entire Deep Learning module. The reasons for using ResNet50 include the fact that its architecture is simpler to implement as compared to other models, besides having the possibility to reduce model complexity in the form of its variants ResNet50 and ResNet18. Besides, it has been established in the research community that Convolutional Neural Networks like ResNet can be trained on smaller datasets, to perform well on tasks like feature extraction because of having skip connections in the architecture because of which it is possible for features to flow from the top layers even to the deepest ones. We didn't want our gradients to vanish away, on which our deeplab and retinanet was going to train to extract masks and bounding boxes so Resnet50 was a suitable choice. Further Resnet50 was chosen over others (like Resnet18, Resnet101 and Resnet152) as it provides an optimal trade-off between performance and computational cost and model size.

- **RetinaNet:**

  RetinaNet is a one-stage object detection model which has a focal loss. It is a single unified network which is further divided into backbone network and two subnetworks. Backbone network extract feature maps. One subnetwork is used for classification of objects and the second subnetwork performs regression for the bounding box coordinates.

  RetinaNet was chosen for object (player, referee and ball) detection in this system. This network was chosen as it is a Single Shot detector which improves the computational complexity. It also features a BiFPN (Bi-directional Feature Pyramid Network), which is used to fuse together features from multiple blocks of the backbone (Resnet50) so as to allow detection at multiple scales. This results in a model which is fast as well as robust. Also, as mentioned above it has focal loss which solves the class imbalance problem. In Faster RCNN this problem is solved by adding Region Proposal network which adds computational inefficiency. RetinaNet is faster as compared to Faster-RCNN and more accurate than single shot detectors like Yolo, SSD which also has class imbalance problems.

- **DeepLabV3:**

  DeepLabV3 has been used for semantic segmentation of goalpost, ground and crowd in the videos. This architecture has a ResNet50 as backbone network as a feature extractor. Last block of the backbone network has astrous convolutions which control the size of feature maps at different scales. On top of extracted features from the backbone, an Atrous Spatial Pyramid Pooling (ASPP) network is added to classify each pixel corresponding to their classes. The output from the ASPP network is passed through a 1 x 1 convolution to get the actual size of the image which will be the final segmented mask for the image.
  DeepLabV3 uses Astrous Convolutions because of which it has capacity to perform well at large fields of view without increasing the number of parameters. It is also capable of learning multi-scale information because of the introduction of the ASPP layer.

- **Heuristics Based Tracker:**

  Tracking Algorithm:

```
Initialize a seed-position for the object based on assumptions for each object and set velocity to zero.
for each frame:
    - Predict future state of the tracked object based on its current velocity.
    - Calculate distance of this future state to all the predicted objects of that class.
    if (the distance is below a threshold):
        - Select the object which has minimum distance to the predicted future state.
        - Update the tracked object's position and velocity based on the selected object.
    else:
        - if the tracked object has not been matched/updated for the last 5 frames,
          generate a new seed-position for the object.
```

  Note: Before generating the seed-positions, we determine the direction of the video using the segmentation mask. If the goal-mask is majorly on the right side of the image midpoint, penalty is being taken from the left and the goal is towards the right.

  Note: '**w**' and '**h**' are used in place of image width and height in the text below

  Seed-Positions for each class:

  1. **Ball:** If goal is on the right we assume the ball to be on left side of the image and below certain height (0.4 * **h**). Among the detections of ball class in this region, the one with highest confidence is selected as the seed-position.
  2. **Goal-Keeper** (GK): Bounding box for the goal is calculated based on the segmentation mask and "player" detections are taken as candidate seed-points if it overlaps with the goal bounding box horizontally. Among the candidates, the object with highest overlap with the goal mask is selected as the seed-point
  3. **Penalty Kick-Taker** (PK): To get a seed-point for penalty kick-taker we first wait for "Ball" to be tracked first. Once the ball is tracked we take all player detections as candidates which are on the left side of the ball, if the goal is on the right side and vice versa. Distance of all the candidates is calculated to the ball, while giving more weightage to vertical distance than horizontal distance (distance = sqrt(3*v + h)). The candidate with minimum weighted distance is selected as the seed-point.
  4. **Referees**: All of the rest of bounding boxes are assumed to be referees.

- **Optical Flow calculation and Goal Detection:**

  Note: The goal detection module is only triggered and invoked after the tracked football reaches close to the goal bounding box.

  At first we estimate the bounding box for the goal based on the segmentation mask. Based on the bounding box, we take a crop of the image containing goal-post from both the current and previous frame. Afterwards we apply the "**calcOpticalFlowFarneback**" function of openCV which gives the vertical and horizontal optical flow values for each point. Using the vertical and horizontal component we compute the angle and magnitude of the optical flow. Next, we take the bounding box of Goal-Keeper and set optical flow inside it to 0, so as to ignore the motion of

the keeper. On the rest of the optical flow values we compute standard deviation *(std)* for both the angle and the magnitude. Using the std(angle) and std(magnitude) we calculate a score using the formula “score = std(magnitude) * 3 + std(angle)”. If the score exceeds 90, we detect that a goal has been scored.

Instructions for usage of Code-Base:

Three scripts have been provided for this project, a training script, and two inference scripts for images and videos respectively. For training it is recommended to use google colab, whereas inference can be done on the PC with ease.

Training:

If you run “python train.py -h” you will receive the following prompt, showing the command line arguments that can be provided to it. I will provide explanations of the command line arguments below:

- **INPUT_DATA_PATH:** The folder should contain the training data provided beforehand. The structure of the directory is supposed to be as follows.

  {INPUT_DATA_PATH}
  |-labels
  |-images
  |-val_images

  - “images” subdirectory will contain all the images to be used for training.
  - “val_images” subdirectory will contain all the images to be used for validation.
  - “labels” sub-directory will contain the .xml and .png labels (for detections and segmentations respectively) for both the validation and training images.
- **OUT_PATH:** Path to the directory for outputting the results of training. The model weights will be saved after every 5 epochs to this directory. Training summary (containing all the losses and validation accuracy over-time) will be saved to this directory as well, which can be visualized by running “tensorboard --logdir {OUT_PATH}”.
- **LEARNING_RATE:** Default value has already been provided and so it is not required to pass this flag.
- **EPOCHS:** Recommended number of epochs to run is 35-40, as the accuracies saturate at that point.

```
usage: train.py [-h] [-i INPUT_DATA_PATH] [-o OUT_PATH] [-lr
LEARNING_RATE] [-e EPOCHS]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT_DATA_PATH, --input_data_path INPUT_DATA_PATH
                        Path to directory containing input dataset.
  -o OUT_PATH, --out_path OUT_PATH
                        Path for outputting model weights and
tensorboard summary.
  -lr LEARNING_RATE, --learning_rate LEARNING_RATE
                        Learning Rate
  -e EPOCHS, --epochs EPOCHS
                        Number of epochs to train
```

Inference on Videos:

If you run “python infer_video.py -h” you will receive the following prompt, showing the command line arguments that can be provided to it. I will provide explanations of the command line arguments below:

- **INPUT_VIDEO:** Path to the video file to be used for goal-detection inference of the system.
- **OUT_PATH:** Path to the directory for outputting the results of inference. 2 separate files will be outputted.
    - {INPUT_VIDEO}-seg.mp4: A video file containing the segmentation results of the video.
    - {INPUT_VIDEO}-det.mp4: A video file containing the object + goal detection results.
- **CHECKPOINT:** Path to the model checkpoint to be used for inference. The checkpoints will be obtained from the training script in the {OUT_PATH} directory. An example for a checkpoint is epoch35.pth which is trained by me and provided to you.

```
usage: infer_video.py [-h] [-i INPUT_VIDEO] [-o OUT_PATH] [-c
CHECKPOINT]

optional arguments:
  -h, --help            show this help message and exit
  -i INPUT_VIDEO, --input_video INPUT_VIDEO
                        Path to directory containing input dataset.
  -o OUT_PATH, --out_path OUT_PATH
                        Path for outputting model weights and
tensorboard summary.
  -c CHECKPOINT, --checkpoint CHECKPOINT
                        Path to checkpoint file to be used for
inference.
```