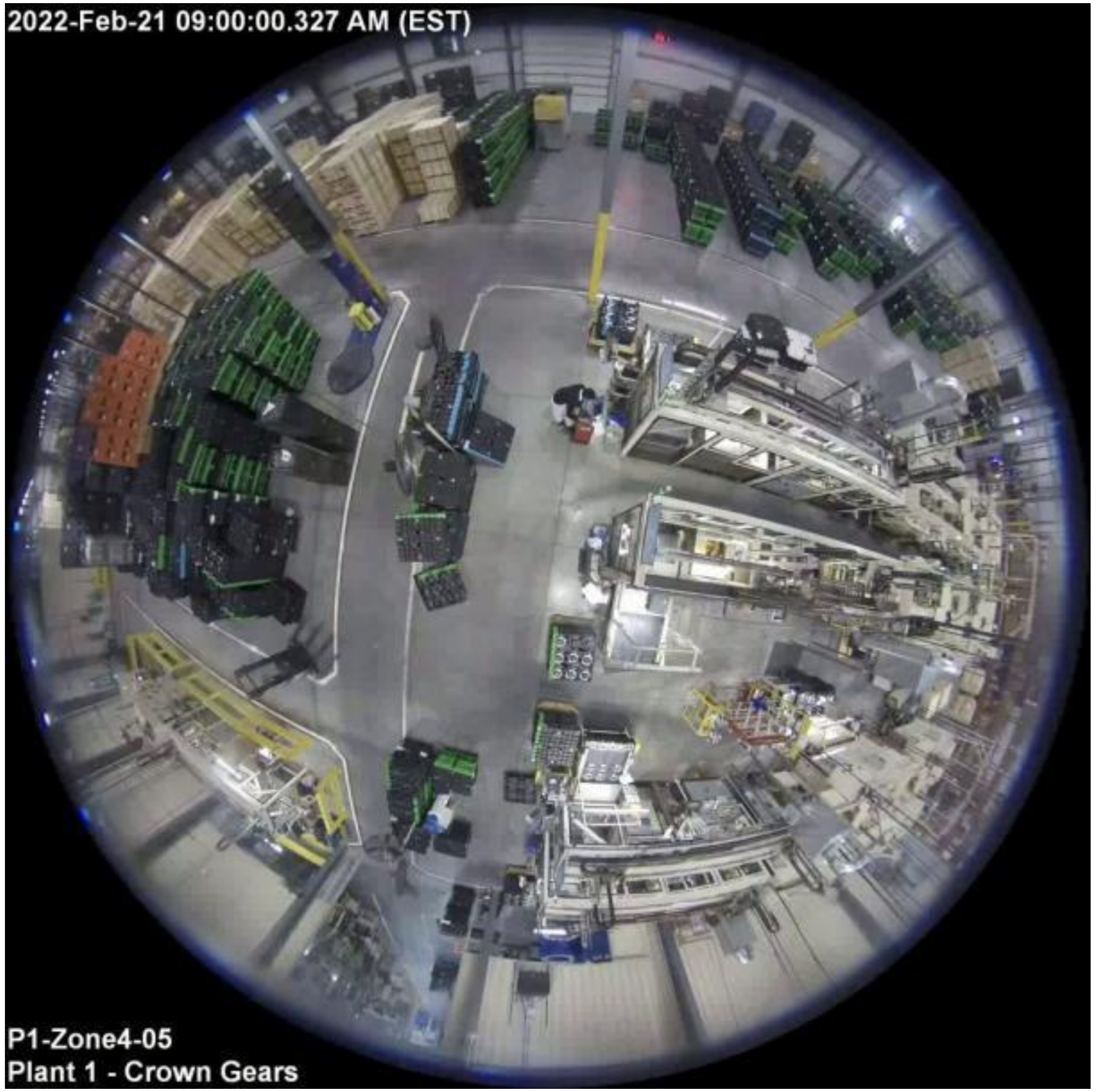
Semantic Segmentation

Hackathon 11/12/2022

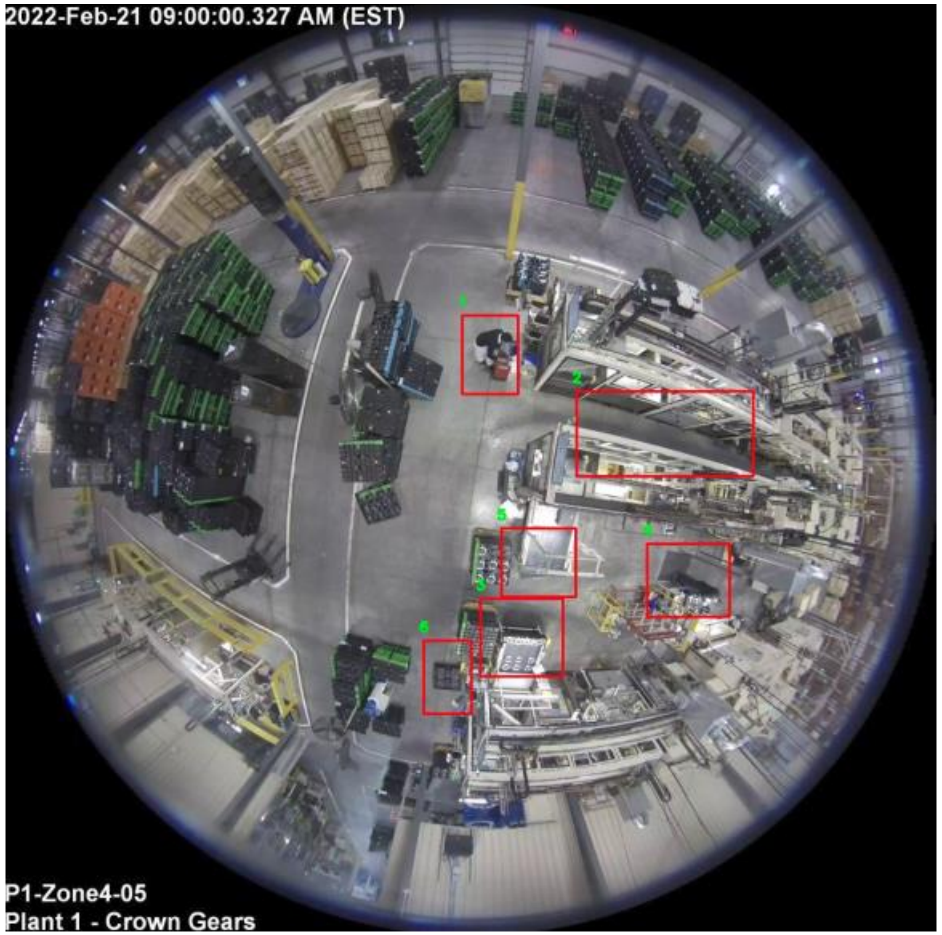
Input

- Fish-eye camera tape of factory



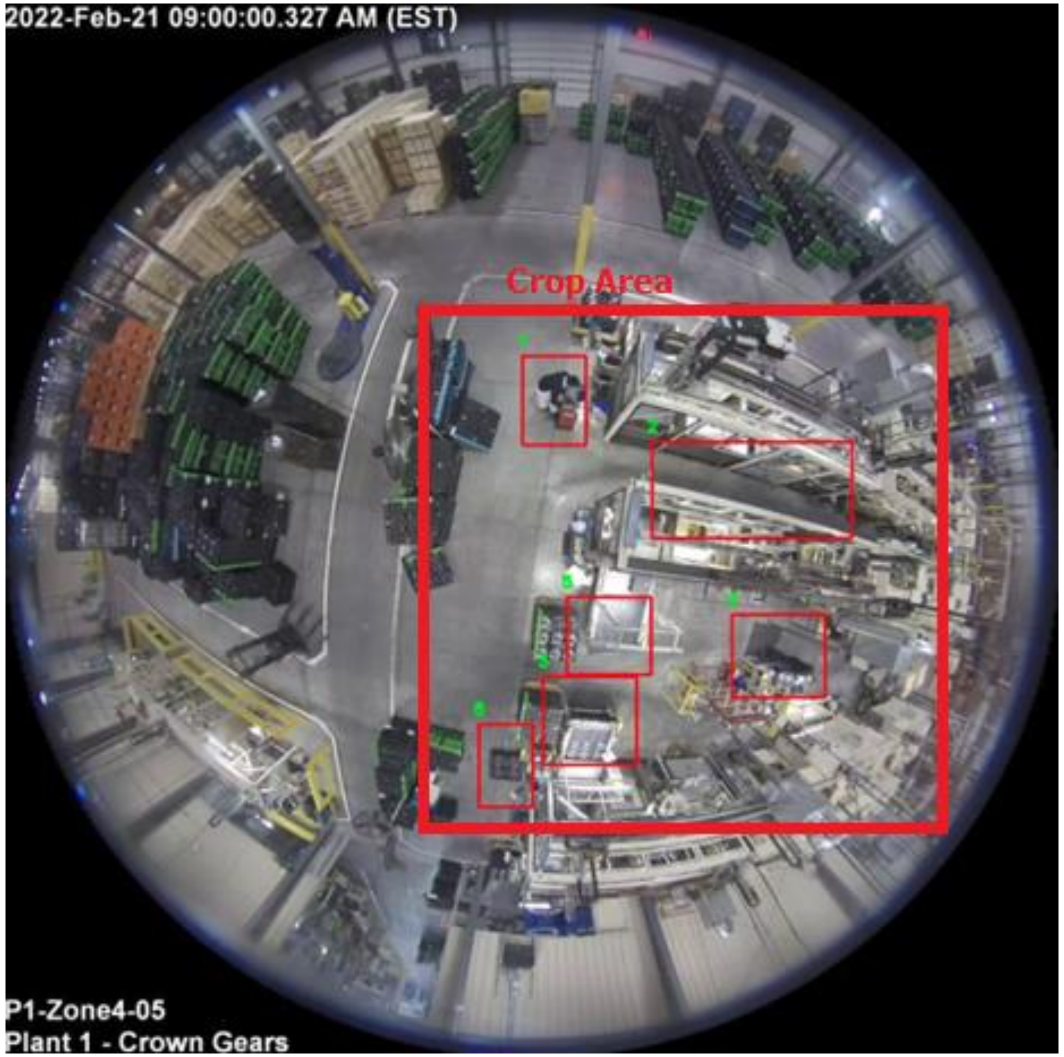
Objective

- 6 Regions of interests (ROI)
- A ROI is occupied if there is a person inside the bounding box (or part of a person)
- Compute the number of occupancy frames for each ROI



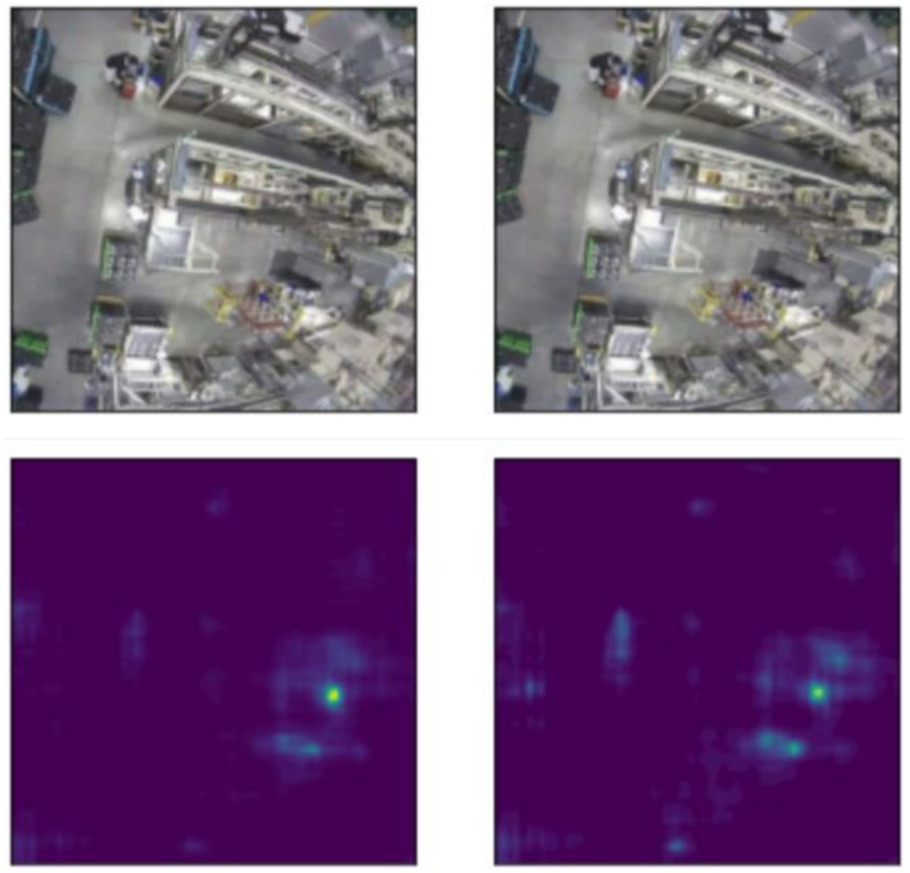
Preprocessing

- Crop the image to include all 6 ROI & remove the fisheye after crop
- This also reduces the image size from 1024x1024 to 500x500
- Standard normalization (required for any torchvision models)



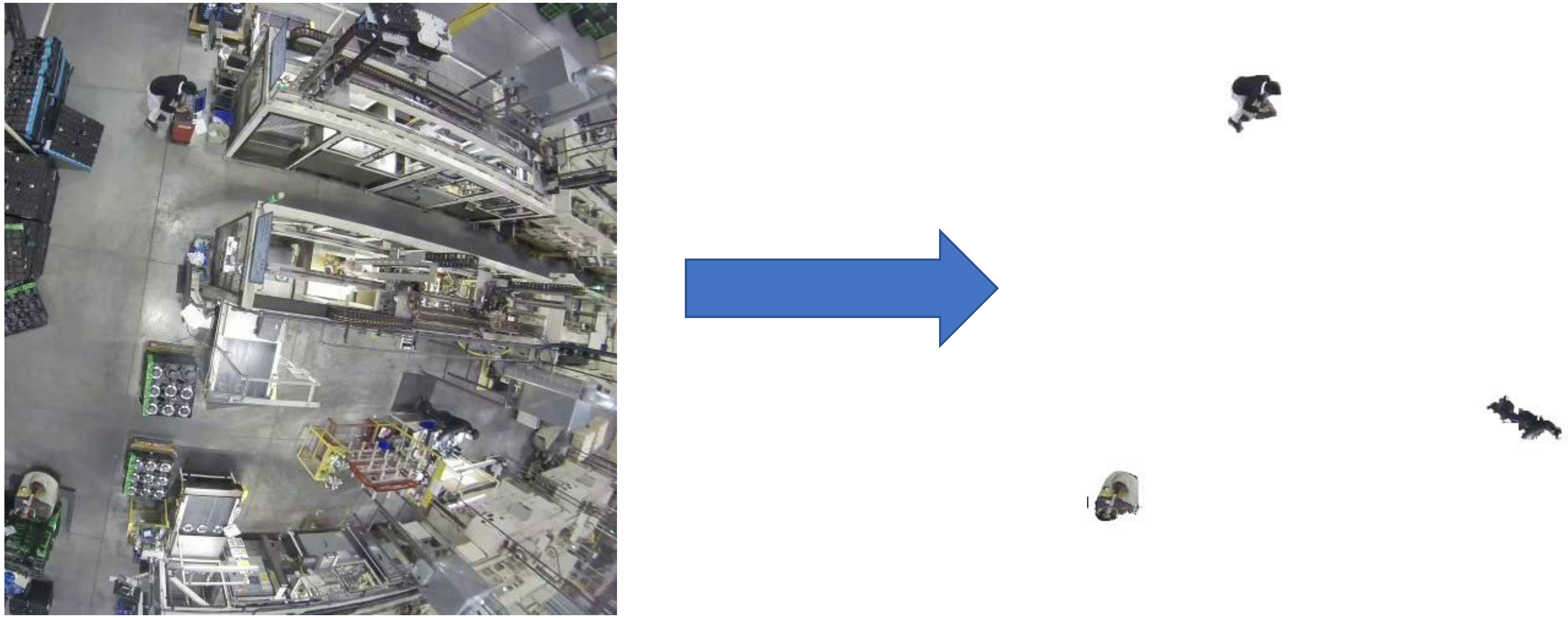
Naïve attempt:
Using pre-trained model directly

- Using pre-trained deeplabv3_resnet50 on the preprocessed data
- Extract "person" mask from the final layer
- Result is unacceptable: completely unable to detect human



Attempt #2: Fine-tuning pretrained model

- Firstly, manually label the training images using Microsoft Paint 😖 (labelled 8/13500 such images)



Fine-tuning pretrained model (cont.)

- Secondly, replace the last layer of deeplabv3_resnet50 with a single layer of Conv2d with 2 classification images (background and person)

```
model = deeplabv3_resnet50(pretrained=True)
model.classifier[4] = nn.Conv2d(256, 2, kernel_size=(1,1), stride=(1,1))
```

Fine-tuning pretrained model (cont.)

- Thirdly, fix the pretrained weights, and only train the last layer with a high learn rate (1e-4)
- Trained on 5 images for 500 iterations (only 5 labelled at the time)
- Train loss decreases from 0.5312 to 0.0598
- Results still unacceptable (noisy and unclear)

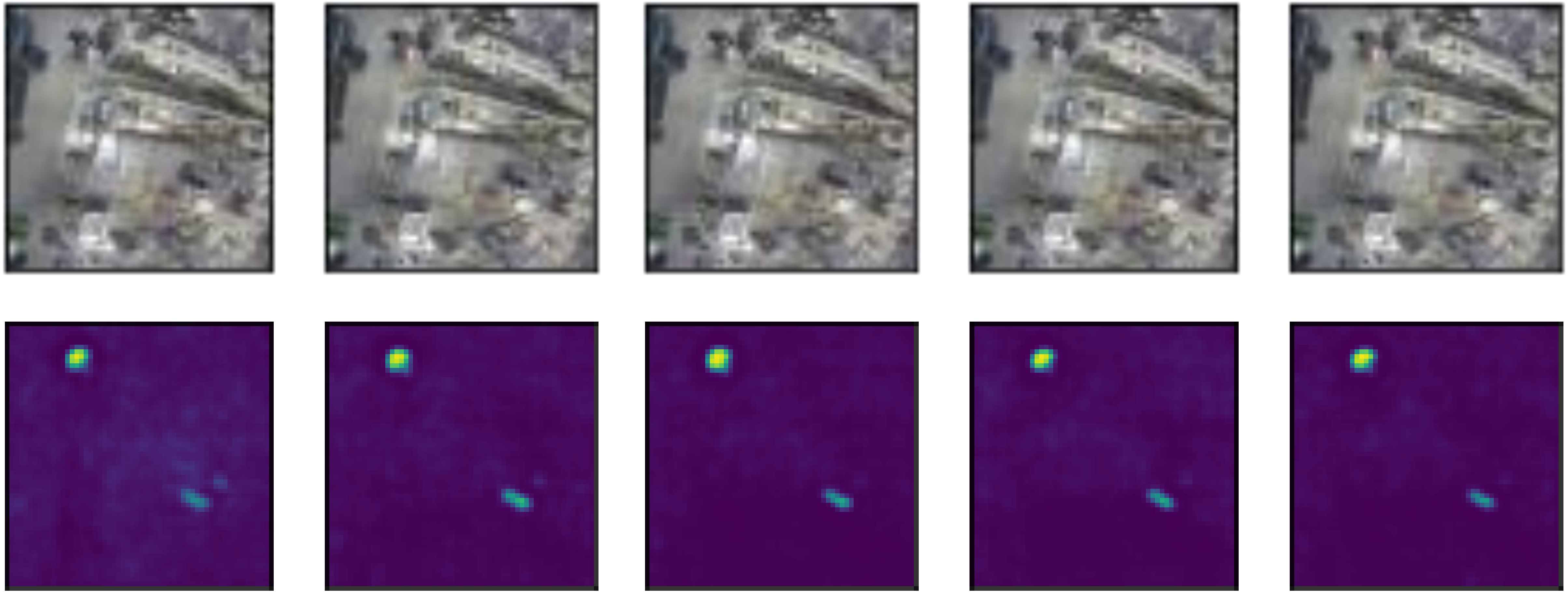
```
optimizer = torch.optim.Adam(model.classifier[4].parameters(), lr=1e-4)
criterion = torch.nn.CrossEntropyLoss()
## (…) standard training loop
```

Fine-tuning pretrained model (cont.)

- Fourthly, train the entire model (including pretrained weight) with a much lower learn rate (1e-5)
- Trained on 8 images for 150 iterations (batch size 8)
- Train loss decreases from 0.0673 to 0.0220
- Dramatic improvement (see next slide)

```
optimizer = torch.optim.Adam(model.parameters(), lr=1e-5)
criterion = torch.nn.CrossEntropyLoss()
## (…) standard training loop
```

Results on first 5 frames



Counting Algorithm

For each frame image & for each ROI box:

1. Preprocess the frame image
2. Make a 2D array mask, coordinates inside the ROI are 1, and coordinates outside are 0
3. Perform a Hadamard product between the ROI mask and the human mask from the model. Note that each grid value of the human mask indicate the probability that the pixel coordinate is part of a person: ~1 means is human and ~0 means not human)
4. If sum the Hadamard product > some threshold, it meaning many pixels inside the bounding box are likely parts of a person. And we count the current frame for the current ROI.

Thank you