Comparative Evaluation of Human and Object Detection Models for Low-light Images

Group A:

1. **Anket Sah**
2. **Amala Chirayil**
3. **Ksheeraj Vepuri**
4. **Kriti Gupta**
5. **Sanmesh Bhosale**

I. Introduction

Many state-of-the-art convolutional neural networks (CNNs) have rightfully earned the title of "state-of-the-art" after making significant breakthroughs. However, according to researchers Yuen Peng Loh and Chee Sang Chen, these breakthroughs were a result of the large availability of bright images. In fact, "less than 2% of total images were low-light data in successful public benchmark datasets such as PASCAL VOC, ImageNet, and Microsoft COCO" [1]. In order to reduce the lack of low-light images, these researchers produced a low-light image dataset called Exclusively Dark.

Inspired by the observations made by these researchers, in this report, we will be examining the results produced by YOLO [3], SqueezeDet [4], CenterNet [5], and Faster R-CNN [6] after feeding in the same 5 low-light images from the Exclusively Dark dataset. In addition, we will also be showcasing the results of each of these models on visible-light images to demonstrate how each model works. Furthermore, we will be doing a comparative study on each of these models by examining the detection speed and accuracy of objects detected. More specifically, we will first take a look at the overall project idea by examining the architecture diagram. In the subsequent sections, we will explore each model closely with a high-level overview of the model, its implementation details followed by the results. Lastly, we provide a table which lists all the models and their respective accuracy and detection speeds.

II. Architecture Diagram

A. High-Level Overview

For our project, we will be feeding in low-light input images into a deep-learning based model called EnlightenGAN [2] which will result in enhanced/enlightened output images. These output images will then be fed into each of the CNN models we are examining. Please refer to *Figure 1* in the Appendix which provides an architecture diagram of our overall project.

Before examining the results of each CNN model, an overview of EnlightenGAN is required. EnlightenGAN is a deep learning-based model that implements unsupervised training on low-light images utilizing attention-guided U-Net as the generator and uses dual-discriminator to direct the global and local information. The steps involved in this process are:

1. *Global-Local Discriminators*: Global loss and local loss functions are calculated.
2. *Self-Feature Preserving Loss:* Constraint the VGG-feature distance between input low-light and its enhanced normal-light output.
3. *U-Net Generator Guided with Self-Regularized Attention:* Implemented with 8 convolutional blocks each consisting of 3x3 convolutional layers, followed by LeakyRelu and batch normalization layer.

We have implemented EnlightenGAN (*details can be found in the "Implementation Details" section*). Discussions and implementations of YOLO, SqueezeDet, CenterNet, and Faster R-CNN can be found in sections III, IV, V, and VI of the report, respectively. At the present moment, we are only feeding in low-light images into each CNN model to examine how each CNN model detects objects in low-light images. In other words, we are not feeding in images that have been processed by the EnlightenGAN model yet.

B. Implementation Details

We implemented EnlightenGAN on an AWS EC2 Ubuntu instance of type p2.8xlarge along with torch (0.3.1), torchvision (0.2.0), visdom server, dominate Python library, and vncserver/vncviewer.

C. Results

Figure 2 provides the output of EnlightenGAN after running [2a] code.

III. YOLO: Real-time object detection

A. High-level Overview

YOLO processes an image in 3 major steps. First, the model resizes the image into 448 x 448. Secondly, a convolution network is run on the image where each CNN layer applies 32/64/256/128 filters to an image and filters are run on the image after it is resized into grids of size 3 x 3. Lastly, the threshold of the resulting detection is determined by the model's confidence.

B. Implementation Details

For YOLO, the project was run on a local Linux system with the following high-end specifications:

Hardware - CPU: Intel core i7 6th gen, RAM: 16GB, GPU: Nvidia GeForce gtx-960x
Software – Linux OS, Python3 [3a].

C. Results

i. Visible-light Image

Figure 3 shows the result of running YOLO model on a visible-light image.

ii. Low-light Image

Figures 4a, 4b, 4c, 4d, and 4e showcase the result of running YOLO model on 5 dark/low-light image.

IV. SqueezeDet

A. High-Level Overview

SqueezeDet uses SqueezeNet as the "backbone" CNN architecture and performs "region proposition and classification in one single network simultaneously. After feeding an input image into a CNN, a low-resolution, high dimensional feature map is extracted. Subsequently, the extracted feature map is fed into the *ConvDet* layer. The trained *ConvDet* convolutional layer works as a "sliding window that moves through each spatial position on the feature map." It is trained by a multi-task loss function to learn detection, localization, and classification in order to output bounding box coordinates and class probabilities.

B. Implementation Details

An AWS EC2 instance with the Deep Learning Base AMI (Ubuntu 16.04) Version 19.3 that used the p2.xlarge GPU instance was set up in order to run this model. The implementation code that we used was executed on Tensorflow. [4a].

C. Results

i. Visible-light Image

Figure 5 showcases the result of running SqueezeDet model on a visible light image.

ii. Low-light Image

Figures 6a, 6b, 6c, 6d, and *6e* showcase the result of running SqueezeDet model on 5 low-light images.

V. CenterNet

A. High-Level Overview

CenterNet uses keypoint estimation to find center points and regresses all the other object properties such as size, 3D location, orientation, and even pose. It is end-to-end differentiable, simpler, faster and more accurate than corresponding box-based detectors. We simply feed the input image to a fully convolutional network that generates a heatmap and peaks in the heatmap correspond to object centers. The model is trained using standard dense supervised learning. Inference is a single network forward-pass, without non-maximal suppression for post-processing.

B. Implementation Details

An AWS EC2 instance with the Deep Learning Base AMI (Ubuntu 16.04) Version 24.3 that used the p2.xlarge GPU instance was set up in order to run this model. PyTorch v0.4.1, CUDA 9.0, Python 3.6 and cudNN 7.1 were used to run and test this model. [5a].

C. Results

i. Visible-light Image

Figure 7 showcases the result of running CenterNet model on a visible-light image. The model detected 15 instances in 3.87 seconds.

ii. Low-light Image

Figures 8a, 8b, 8c, 8d, and 8e showcase the result of running CenterNet model on a low-light images

.

VI. Faster R-CNN

A. High-Level Overview

Faster R-CNN is a region-based CNN, that is composed of two networks, a Regional Proposal Network (RPN) and an object detection network. The RPN is used to generate region proposals that most likely contain objects and these region proposals are fed into an object detection network layer to detect the objects. In comparison with its predecessor, Fast R-CNN which uses Selective Search to generate region proposals, Faster R-CNN introduce "novel RPNs that share convolutional layers with state-of-the-art object detection networks." Hence, the time it takes to compute region proposals is small (e.g., 10ms per image).

B. Implementation Details

An AWS EC2 instance with the Deep Learning Base AMI (Ubuntu 16.04) Version 19.3 that used the p2.xlarge GPU instance was set up in order to run this model. The implementation code that we used incorporated Detectron2, which is the software system built by Facebook AI Research to run various object detection models. [6a].

C. Results

i. Visible-light Image

Figure 9 showcases the result of running Faster R-CNN model on a visible light image. The model detected 18 instances in 0.62s.

ii. Low-light Image

Figures 10a, 10b, 10c, 10d, and 10e showcase the result of running Faster R-CNN model on a low-light image.

VII. Comparison Between Models

Model	Accuracy	Time Required to Predict
YOLO	34.68%	31.3 seconds
SqueezeDet	14.66%	4 seconds
CenterNet	62%	3.5 seconds
Faster R-CNN	60%	1second

For future work, we will be working closely with one model based on the results in addition to incorporating EnlightenGAN.

VIII. References

1. Getting to know low-light images with Exclusively Dark dataset (http://www.cs-chan.com/doc/cviu.pdf)
2. EnlightenGAN: Deep Light Enhancement without Paired Supervision (https://arxiv.org/pdf/1906.06972.pdf)
    a. Code: https://github.com/TAMU-VITA/EnlightenGAN
3. You Only Look Once: Unified, Real-Time Object Detection (https://arxiv.org/abs/1506.02640)
    a. Code: https://pjreddie.com/darknet/yolo/
4. SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving (https://arxiv.org/pdf/1612.01051v4.pdf)
    a. Code: https://github.com/BichenWuUCB/squeezeDet
5. Objects as Points (https://arxiv.org/pdf/1904.07850v2.pdf)
    a. Code: https://github.com/xingyizhou/CenterNet
6. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (https://arxiv.org/pdf/1506.01497v3.pdf
    a. Code: https://github.com/facebookresearch/detectron2

IX. Appendix

The following low-light images were fed into each of the convolutional networks. The result of each convolutional network is showcased in their respective sections.

Group A

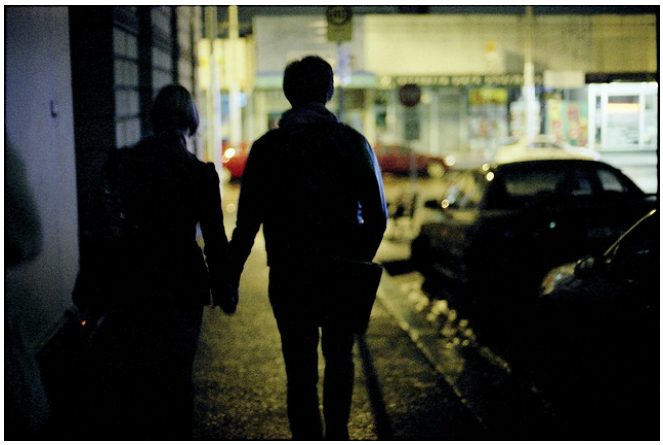


Figure 1: Architecture Diagram

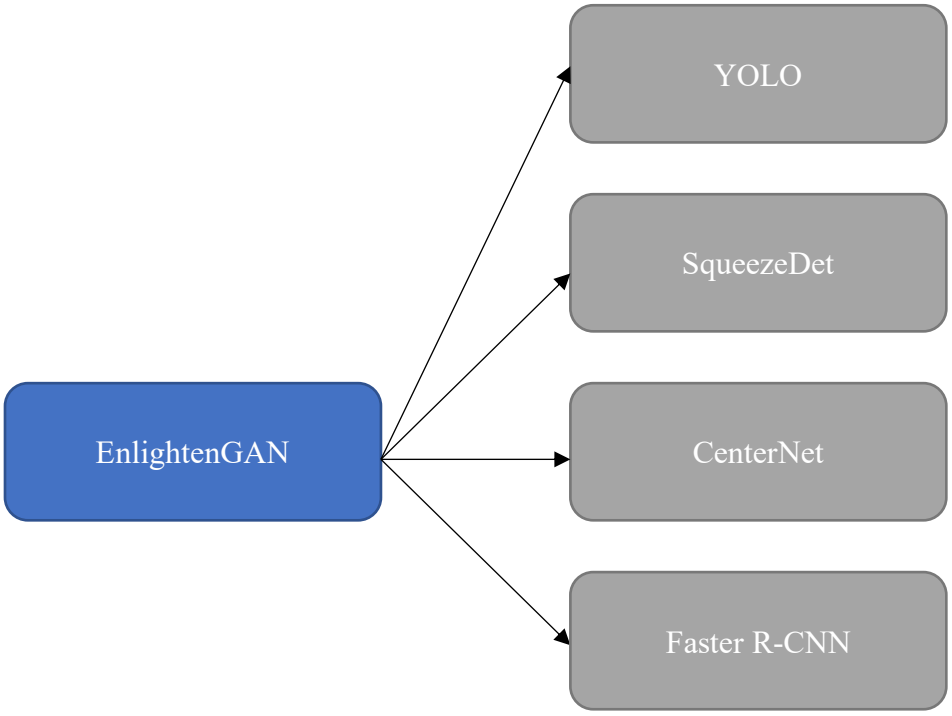
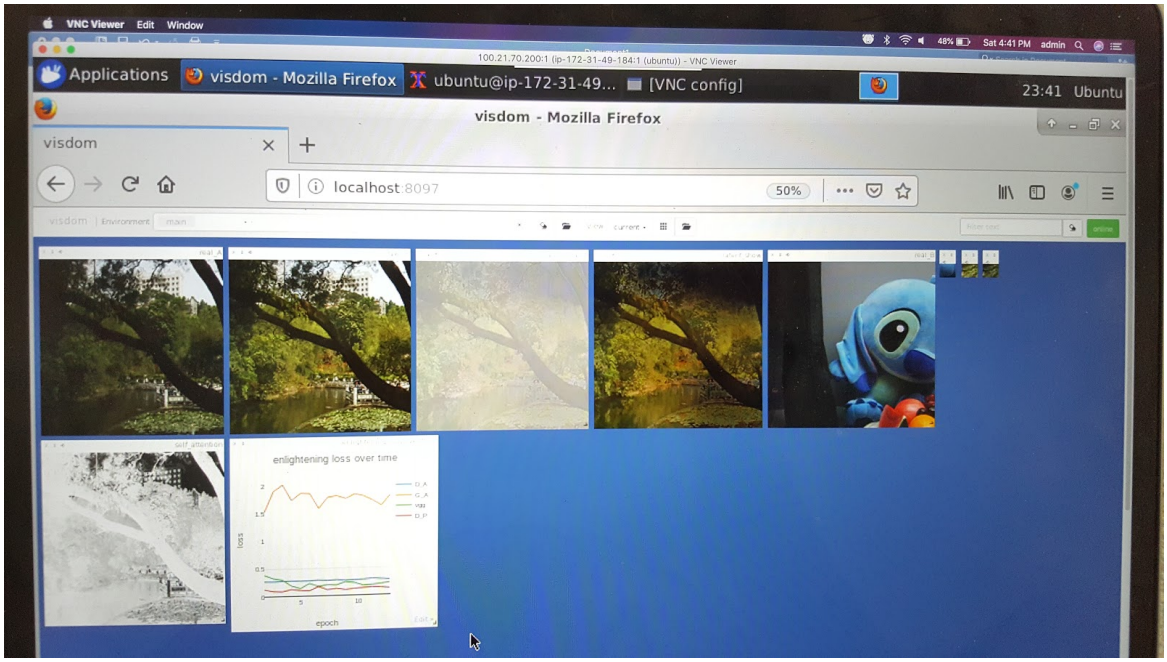
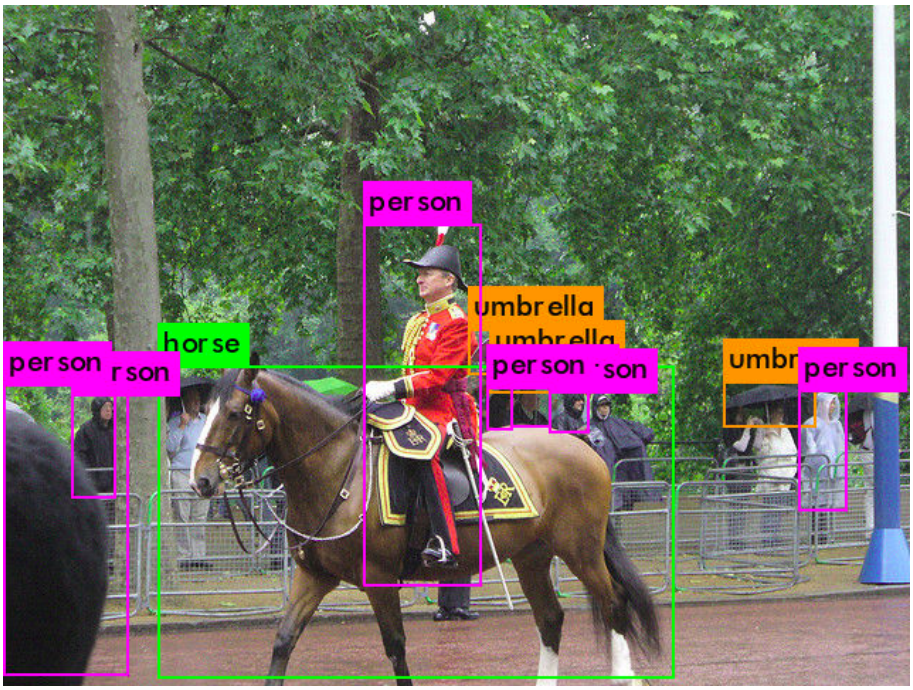


Figure 2: EnlightenGAN Output



YOLO

Figure 3: Result of YOLO on a visible-light image



Low-light Image Results:

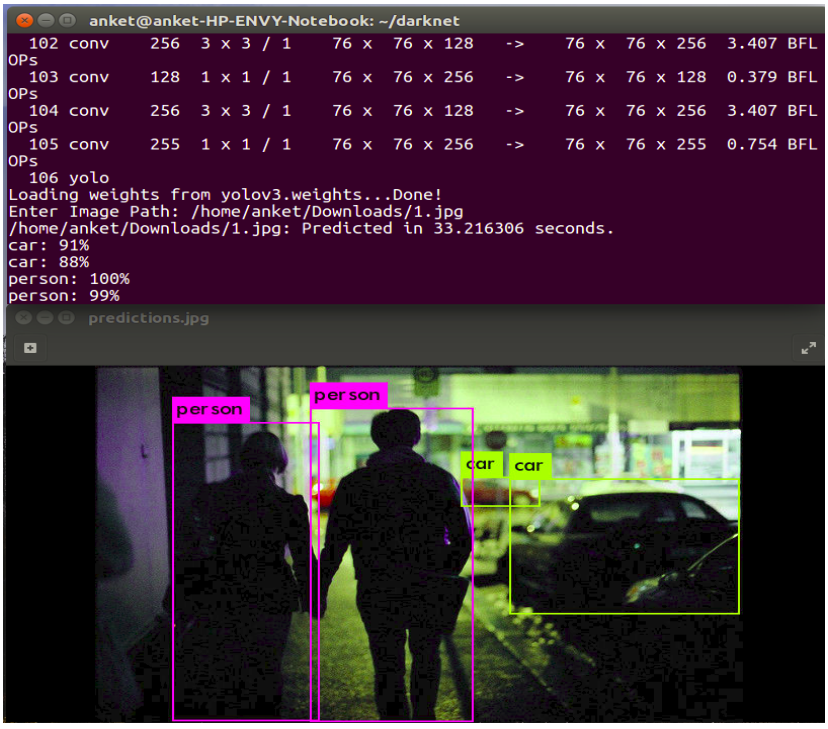


Figure 4a: Detected 4 instances in 33.216306s

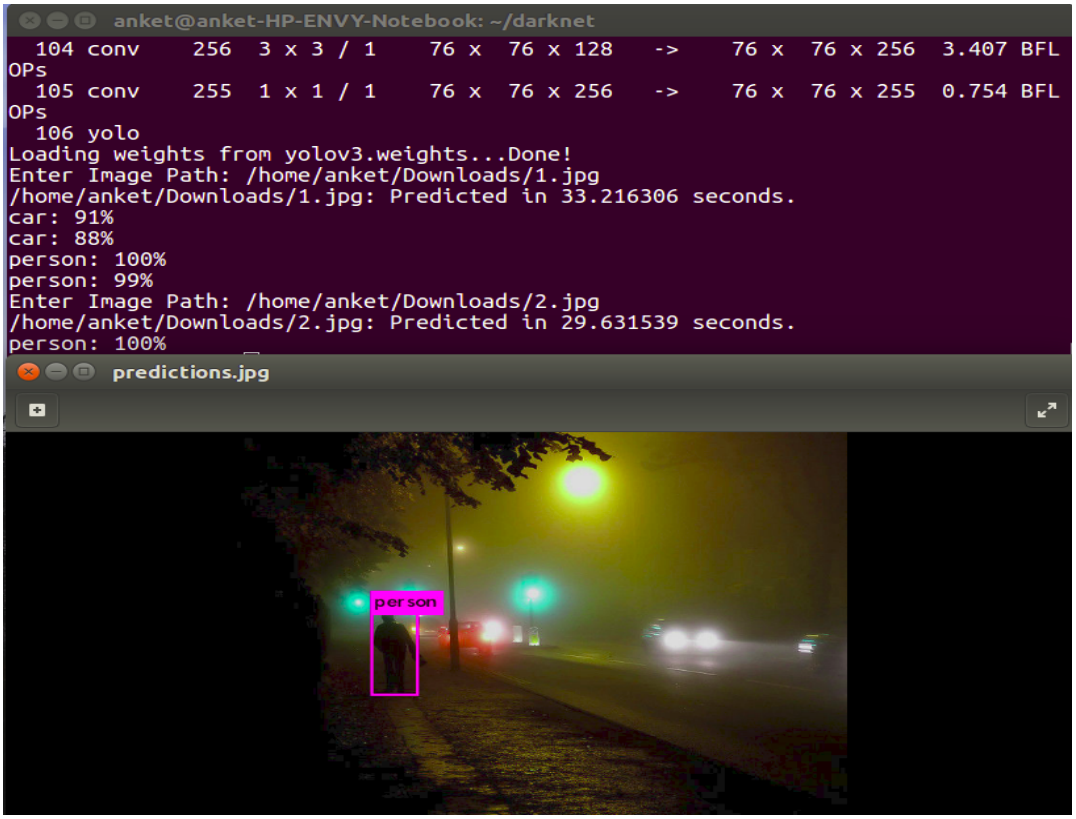


Figure 4b: Detected 4 instances in 33.216306s

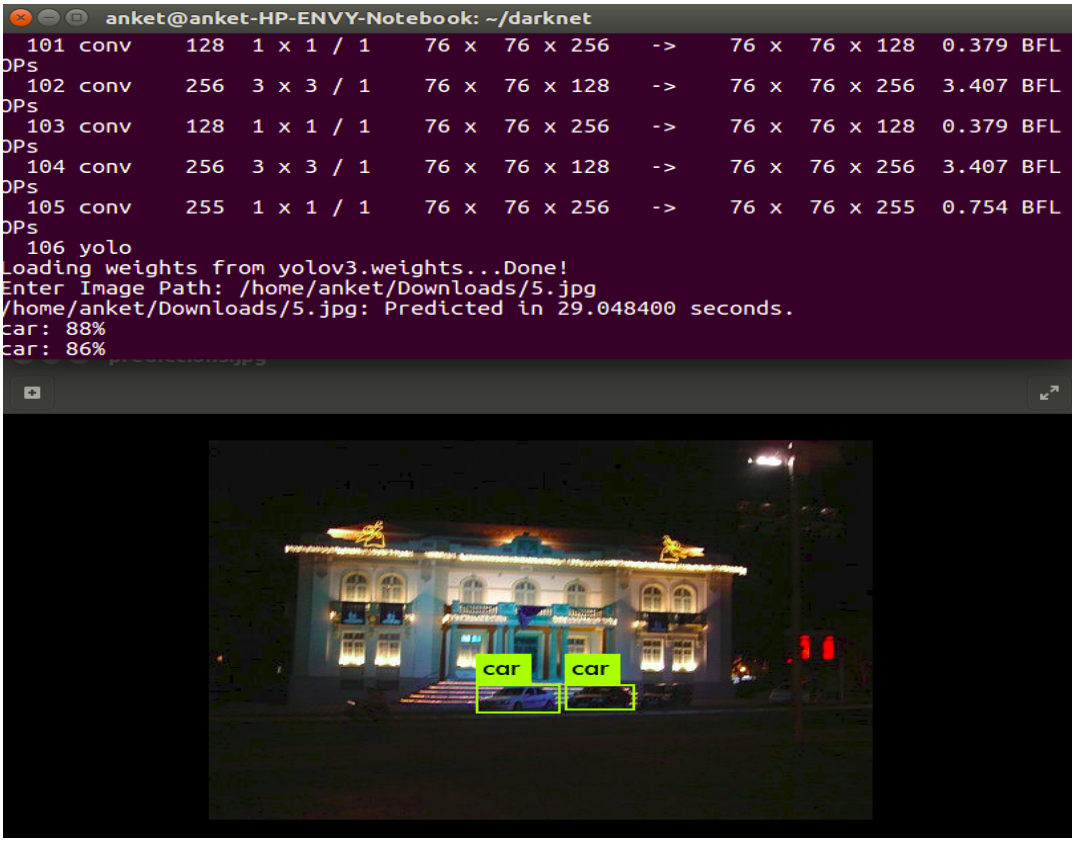


*Figure 4c:*Detected 2 instances in 29.0484s

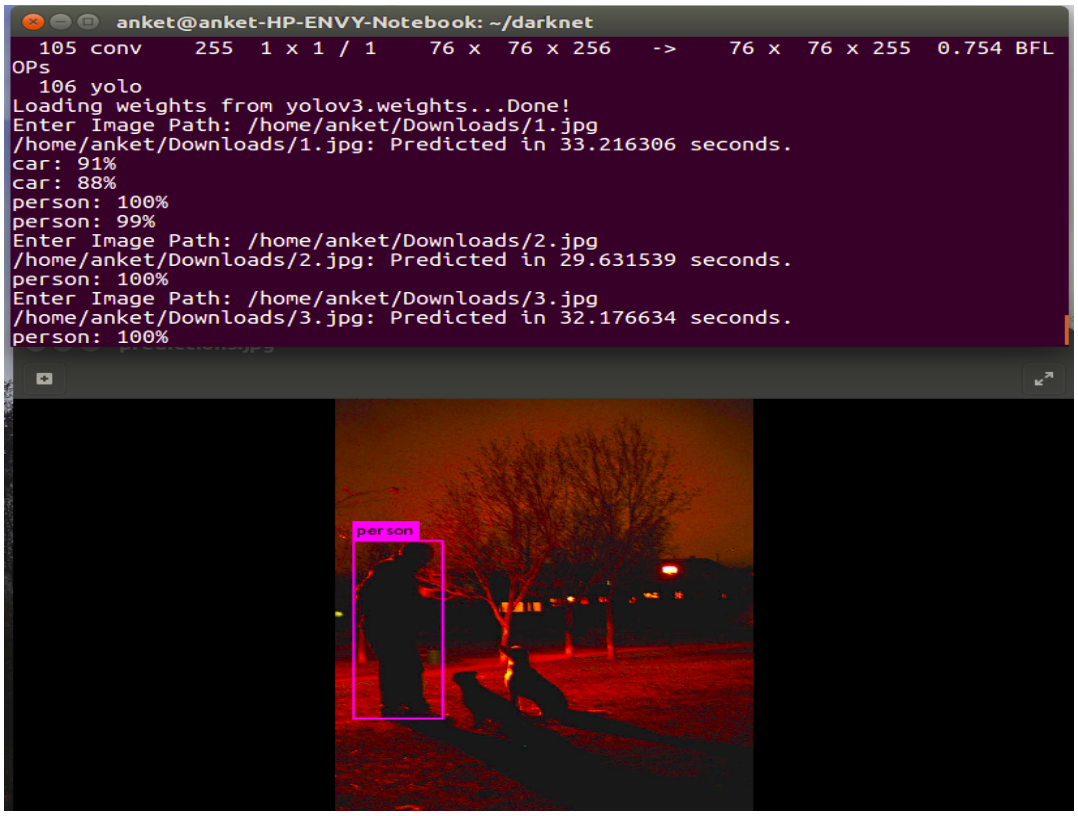


Figure 4d: Detected 1 instance in 32.176634s

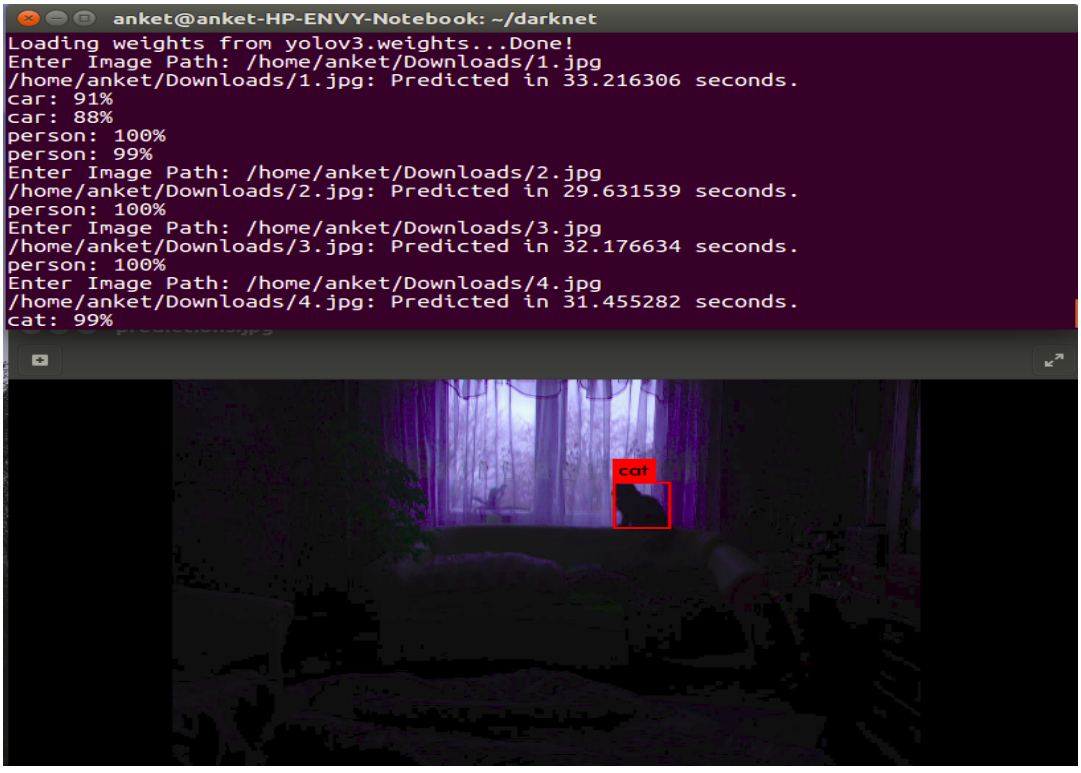


Figure 4e: Detected 1 instance in 31.455282s

SqueezeDet

Figure 5: Output Image as a result of SqueezeDet model

Low-Light Image Results:

Input images are scaled to 1242x375 which is why the images below are elongated

Figure 6a: Detected 2 instances where one instance (blue-bound box) is a cat with probability 0.75 and second instance (purple-bound box) is a pedestrian with probability 0.47 upon closer inspection. The object detected as a cat is incorrect.

Figure 6b: Detected 1 instance where the instance (blue-bound box) is a car with probability 0.48 upon closer inspection.

Figure 6c: 0 instances were detected upon closer inspection.

Figure 6d: Detected 1 instance where the instance (blue-bound box) is a cat with probability 0.78 upon closer inspection. This detection is incorrect for two reasons. First, there are two objects within the bounding box and second, both objects within the bounding box are dogs.

Figure 6e: 0 instances were detected upon closer inspection.

CenterNet

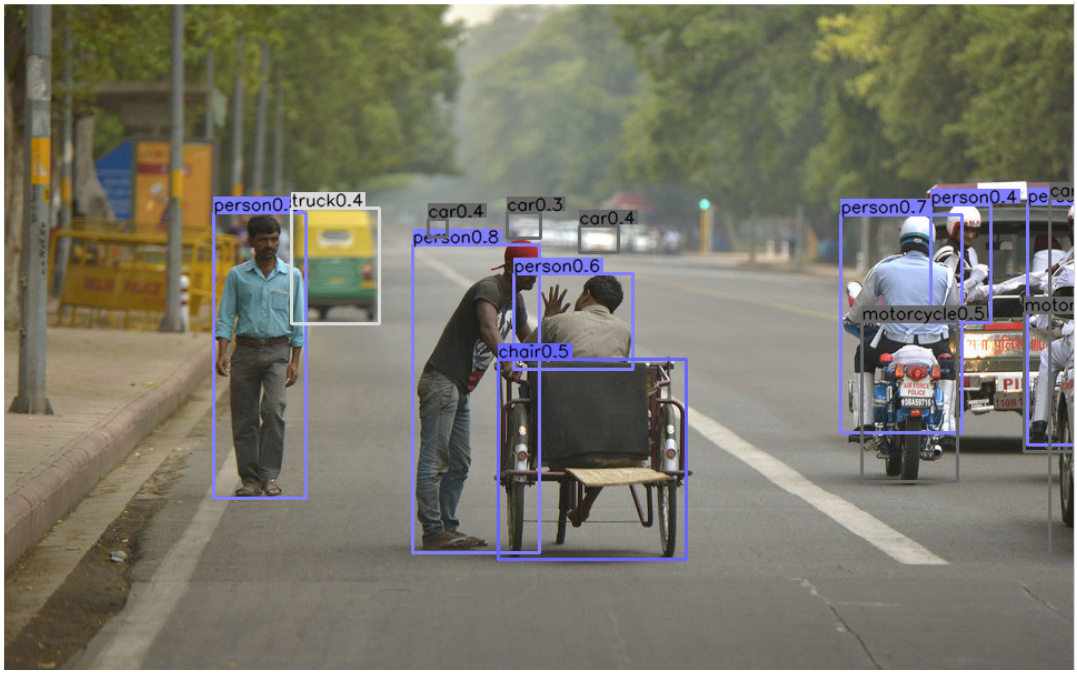


Figure 7: Output Image as a result of CenterNet model (detected 15 instances in 3.87s)

Low-Light Image Results:

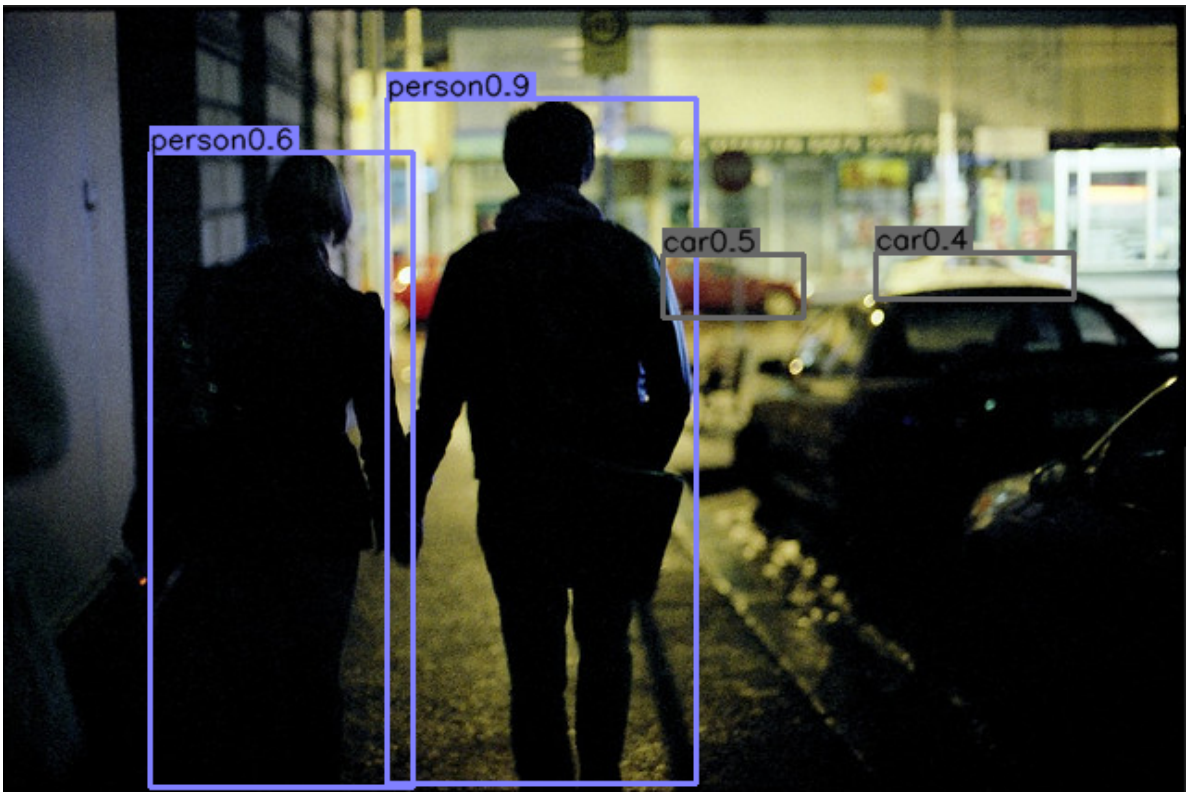


Figure 8a: Detected 4 instances in 3.78s

Figure 8b: Detected 5 instances in 3.80s

Figure 8c: Detected 6 instances in 3.66s

Figure 8d: Detected 3 instances in 3.70s

Figure 8e: Detected 2 instances in 3.71s

Faster R-CNN

Figure 9: Output Image as a result of Faster R-CNN model

Low-Light Image Results:

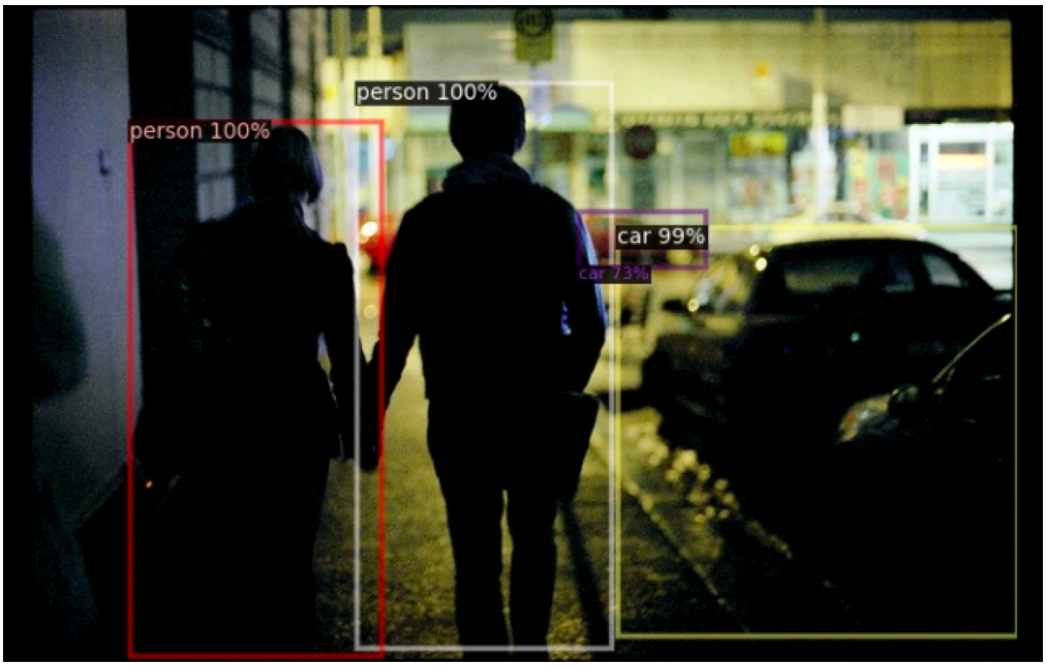


Figure 10a: Detected 4 instances in 1.19s

Figure 10b: Detected 5 instances in 0.62s

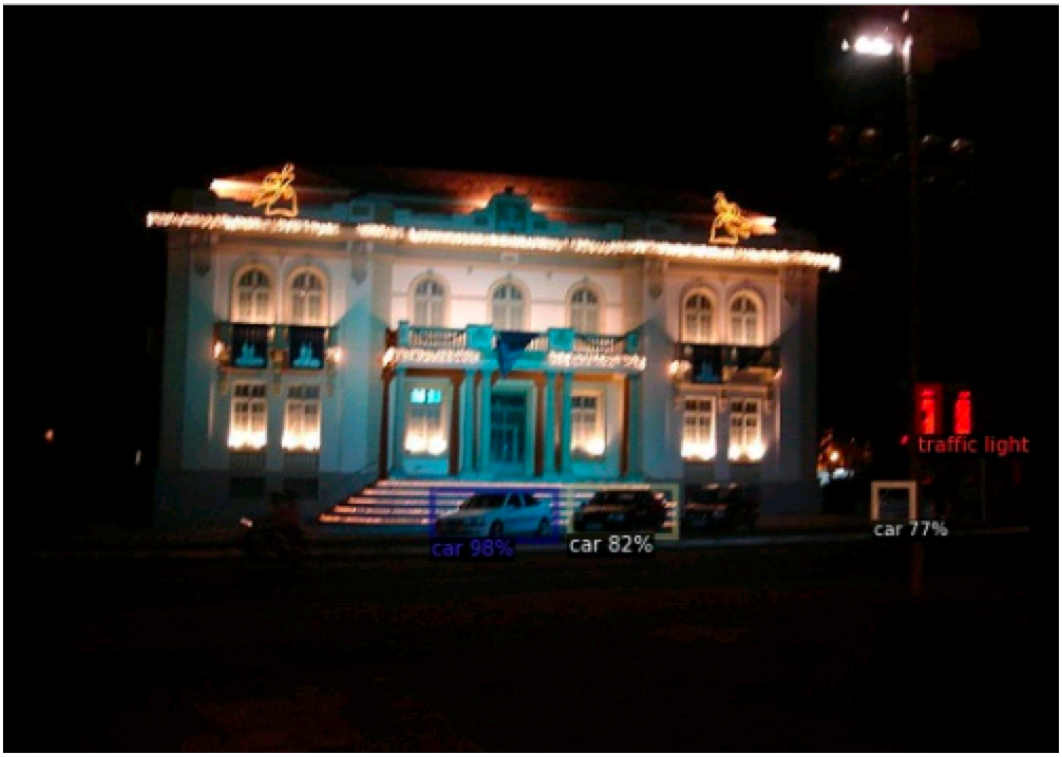


Figure 10c: Detected 4 instances in 0.68s

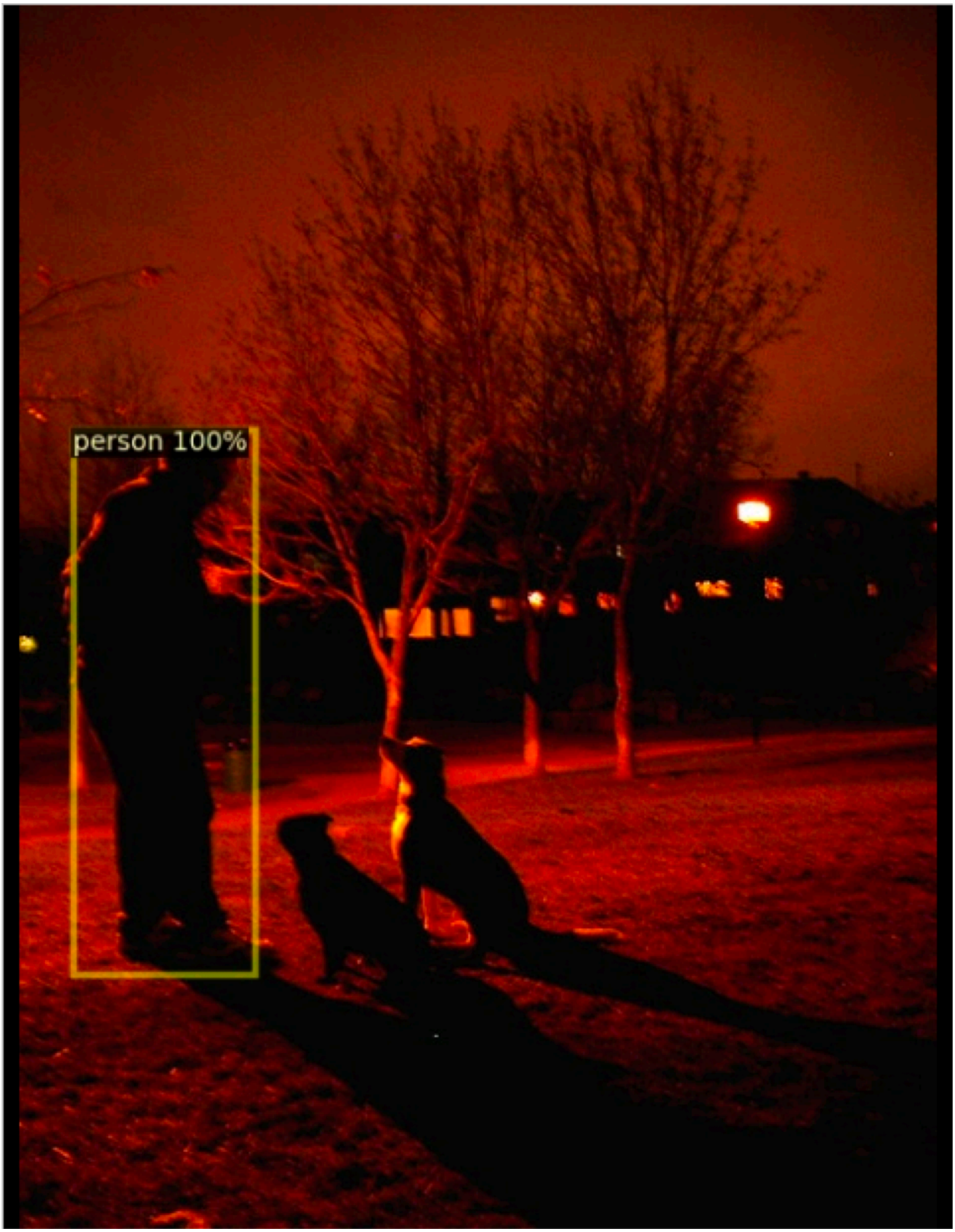


Figure 10d: Detected 1 instance in 0.63s

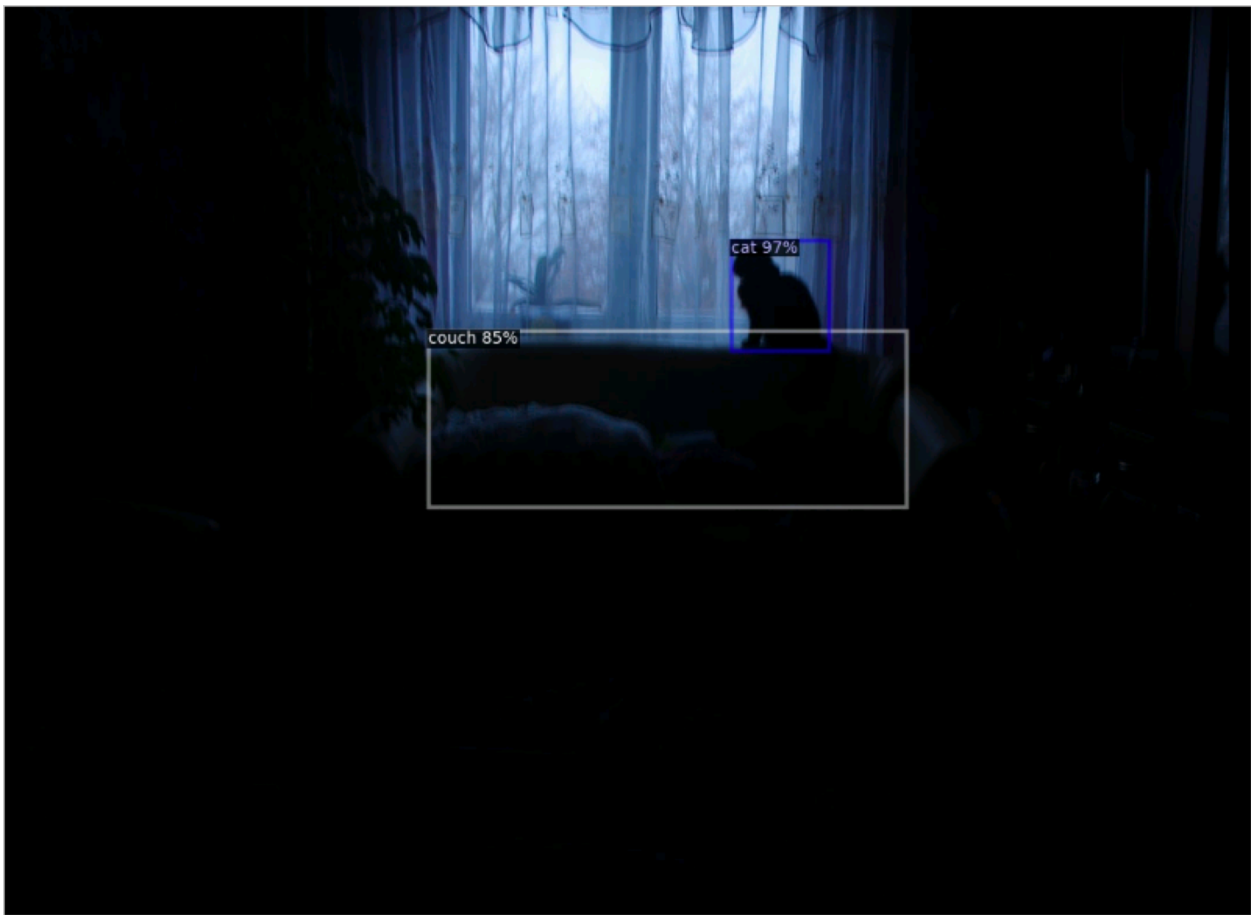


Figure 10e: Detected 2 instances in 0.65s