Training an SSD object detector on infrared imagery

Robert Short
University of Central Florida
4000 Central Florida Blvd.
rshort@knights.ucf.edu

Abstract

There is substantial interest in using longwave infrared image sensors for detection of pedestrians, bicyclists, vehicles, etc. in autonomous vehicle applications. There are many advantages to infrared over visible imagery (background rejection, vision through smoke/fog, and others), but little research into applying modern deep-learning computer vision techniques to infrared sensors. In this work, we train an SSD object detector on a dataset of infrared imagery of common driving scenarios and present the results.

1. Introduction

Object detection in RGB imagery, for self-driving vehicles and other applications, is an exciting but well-explored field. Large datasets such as PASCAL VOC or MSCOCO have allowed researchers to develop algorithms with performance on par with that of humans. However, the application of similar techniques to more exotic types of imagery has remained largely unexplored.

Automatic object detection is critical for driverless car applications, and that is the focus of this project. There is a need for accurate detection and localization of cars, trucks, pedestrians, animals, etc., to enable safe operation of such vehicles. Relatedly, visible cameras may not be ideal for this usage, since visible imagery varies strongly with ambient illumination and does not work well in fog, among other concerns.

Infrared (IR) object detection has several advantages for driverless car sensing. Objects of interest (people, animals, running vehicles) are self-illuminating, and background regions are less bright. This leads to easier discrimination. Also, IR has better propagation through obscurants like smoke or fog, which is important in safety-critical systems.

Unfortunately, there are no datasets of comparable size or scope for imaging modalities other than visible (i.e. RGB). Fortunately, transfer learning provides a solution for applying models trained on large datasets to narrower contexts. We use transfer learning extensively in this work.

2. Related Work: RefineDet512

To establish a baseline for detection performance on this dataset, FLIR Systems trained a RefineDet512 [3] detector on this dataset. Detection performance is summarized in Figure 2.

Object class	**AP**
person	0.794
bicycle	0.58
car	0.856
dog	0.118
mAP	0.587

Table 1: Class APs and mAP for the RefineDet baseline detector.

Unfortunately, weights are not publicly available, so we could not reproduce this configuration. We will use these results as a baseline for performance evaluations.

RefineDet is a single-shot detector that uses multiple layers of regression on the anchor boxes to improve localization and classification. The authors claim that this method achieves a throughput (in frames per second) on par with SSD300, while achieving better performance on the VOC and COCO datasets.

3. Proposed Approach

For its simplicity and multitude of implementations, we chose an SSD300 object detector as the basis for our experiments. Although SSD512 would possibly yield better performance, limited access to computational resources meant that SSD300 was a better choice.

3.1. SSD Implementation: ssd_keras

Although the original Caffe-based implementation of SSD is available, the difficulty of installing and running it made this option impractical, especially since we primarily used Google's Colaboratory service for computation.

For our project, we chose to use an SSD-architecture object detector. There are several Python implementations SSD available, but we chose the ssd_keras (available on GitHub at [1]), a Keras-based port of the SSD architecture, due to the quality of associated documentation.

During initial experiments with the framework, it became clear the models run in inference mode (where the

TensorFlow graph does non-max-suppression and decoding of relative coordinates) causes TensorFlow to crash. We did not extensively investigate this, as running models in training mode is a simple solution and was necessary for most of our experiments.

The framework also includes an average precision evaluation script. Unfortunately, the main branch has a bug which triggers for any class with no detections, crashing with an “IndexError”. There is a patch to fix this issue [2], which has not been merged into the master branch. We applied this fix for our experiments since we encountered this error quite frequently when evaluating untrained and partially-trained networks.

[PLACEHOLDER]: Detection coordinate bug

Commit activity suggests that maintenance stopped around April of 2018; This was not apparent when we started the project. Overall, the framework was functional, but unreliable. If starting from scratch, we would recommend a different choice.

3.2. Pre-trained weights

As a starting point, we chose to use pretrained SSD300 weights for the MSCOCO dataset. As we have mentioned, the classes in the FLIR ADAS dataset map onto COCO classes with few alterations, so there is good reason to believe that transfer learning will be productive.

3.3. Data augmentation strategy

The original SSD [4] was trained on the ImageNet,COCO, and VOC datasets using a data augmentation strategy that is adapted for RGB color images, of the type commonly produced by consumer cameras. Initial experiments using this strategy showed that the training loss levelled off quickly and did not yield good performance.

The geometric transforms (scaling, cropping, etc.) are valid for all imaging modalities, but some of the photometric transforms did not make sense on infrared data. In particular, color transformations (hue jitter, saturation jitter, and channel swapping) are not valid on this type of data. Channel swapping is an identity operation, because the data is inherently monochromatic, but the other two operations risk producing nonsensical data.

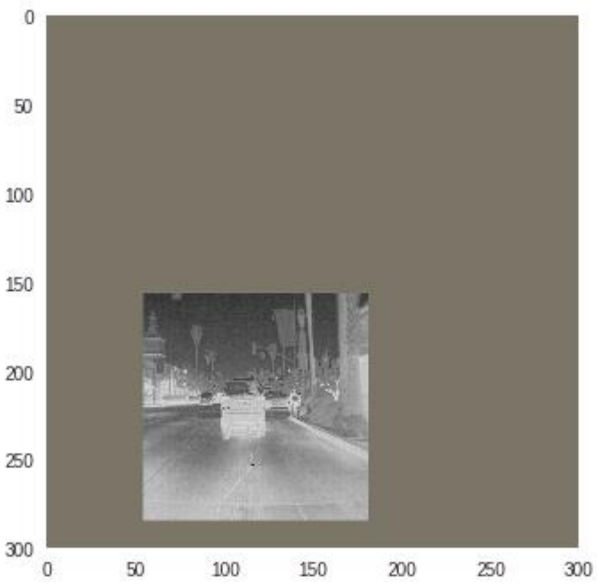

Figure 1: An example of the original SSD data augmentation chain, as applied to an example image. Note that this is one of several transform sequences that may be applied.

In our final implementation, we retained SSD’s geometric transformations, random brightness adjustment, and random contrast adjustment, but discarded the remaining photometric transformations.

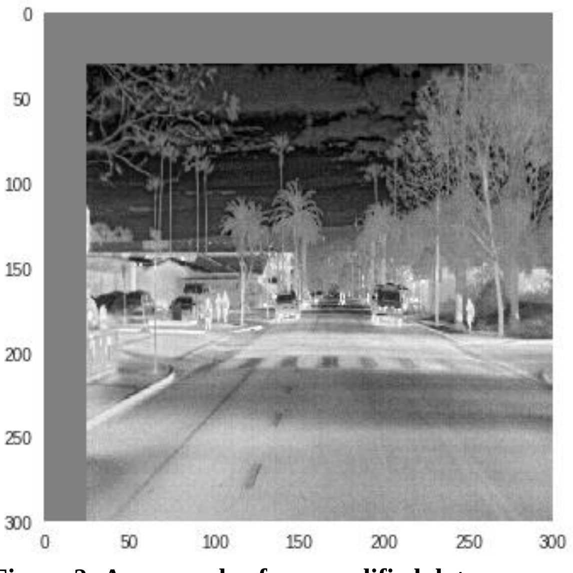

Figure 2: An example of our modified data augmentation chain, as applied to an example image. Hue variation has been removed, and the background color is (127, 127, 127).

3.4. Training

During most of the training process, we used a batch size of eight. Increasing the batch size did not seem to increase training speed (in terms of images per second), possibly due to the data augmentation process (implemented in OpenCV) being CPU-bound.

The original implementation of SSD used Stochastic Gradient Descent (SGD) for optimization of the loss function. Initially, we tried using the Adam optimizer [5] on the belief that it would converge faster than SGD. During training, we encountered several instances where Adam would get “stuck” and fail to decrease the loss after several thousands of iterations.

Subsequently, we tried using SGD with Keras’ functionality for reducing learning rate when a plateau in the validation loss was encountered. This configuration seemed to more reliably converge than Adam, and was used for the rest of our training.

Our working theory to explain the poor performance of Adam is that the small batch size did not provide a consistent loss landscape for the adaptive parameters to take advantage of. SGD does not suffer from this issue, and there is some evidence that SGD exhibits better generalization with smaller batch sizes. Regardless of the reason, we found SGD to be superior for this particular application.

Our overall optimization strategy was SGD with a starting learning rate of 0.001, batch size of eight. After every thousand training steps, the model’s loss on the validation set was evaluated. If three thousand steps went by without a decrease in validation loss (a plateau), the learning rate was halved.

3.5. Loss function

While still in the process of training the detector, we plotted the detections of the network as a diagnostic. When the mAP failed to increase, we tried improving localization to fix the issue, on the assumption that a stronger localization loss might be necessary to adapt to the slightly unusual bounding box conventions of the dataset.

To achieve this, we looked to the SSD loss function, which takes the form:

$$L = \frac{1}{N}\left(L_{confidence} + \alpha L_{localization}\right)$$

In this equation, L is the overall loss, N is the number of matched (IoU above 0.5) anchor boxes, the confidence term is the loss component due to classification, and the localization term is loss component due to bounding box regression. Lastly and most importantly, alpha controls the weight of the localization loss relative to the classification loss.

The original SSD used a value of 1.0 for alpha. During our experiments, we tried both the original 1.0 and 4.0 for alpha.

4. Experiments

4.1. Metrics used: Precision, Recall, and Average Precision

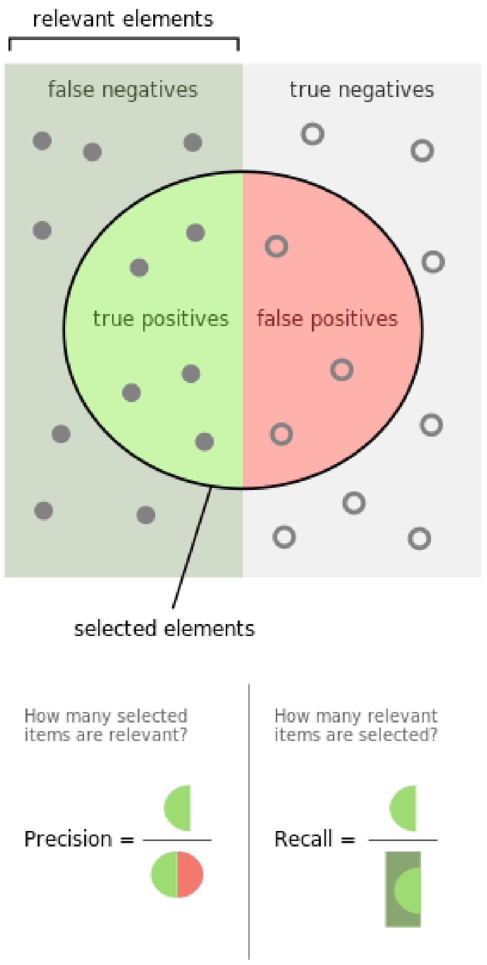

Figure 3: Visualization of precision and recall. Source: [1]

The primary metrics used for quantifying the performance of an object detector are precision, recall, and average precision.

As shown visually in Figure 3, precision is the fraction of true detections (i.e. GT boxes which have been correctly classified and acceptably localized) that the detector made over all detections (including false positives). Recall is the fraction of true detections over the number of GT boxes in the scene.

For any given detector, there is generally a tradeoff between precision and recall. By making the detector less selective, we can increase the number of objects detected, but we will also increase the false positive rate, driving down the precision. Similarly, we can increase precision by making the detector more selective, but this risks missing some small or indistinct objects, hurting recall.

To quantify the detector's quality independent of the selectivity (which can be varied), we plot precision against recall (which is varied, in the case of SSD, by changing the confidence threshold for detection) and integrating under the curve. A precision-recall curve can be generated for each object class.

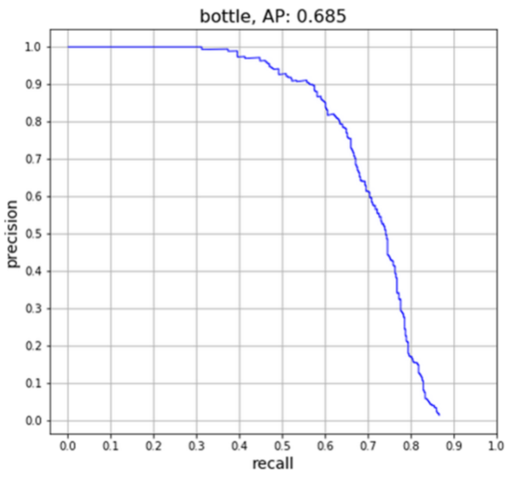

Figure 4: An example of a precision-recall curve. Note that precision decreases as recall increases. The area under the curve is the AP value.

The area under the precision-recall curve is called the average precision. (Meaning that it is the precision value averaged over all possible recall values). Mean average precision is simply the mean of average precisions for each class of interest.

It should be noted that the classification of a detection as either a true positive or a false positive depends on an intersection over union (IoU) threshold. In this paper, we take the IoU threshold as 0.5, corresponding to COCO's AP50 calculations. All other aspects of the performance metrics (integration method, bounding box edges, etc.) are handled in a manner that is consistent with COCO's conventions.

4.2. Dataset used: FLIR ADAS

The dataset used in this project is published by FLIR Systems, primarily a manufacturer of infrared cameras. It is called the FLIR ADAS (Advanced Driver Assistance System) dataset and will be referred to by this name in the remainder of the paper.

The FLIR ADAS imagery, predictably for a dataset geared towards driverless car applications, is a set of 10228 thermal images taken from a forward-facing IR camera mounted on a moving vehicle. The images are annotated with bounding boxes for selected categories. Many (but not all) images have paired RGB images from a parallel visible camera, although the bounding boxes are not valid for the RGB images, due to the difference in camera geometry.

The published data consists of 8862 training images, 1366 validation images, and 4224 frames from a continuous video. These images are provided in both the original 16-bit lossless format, and 8-bit compressed JPEG versions.

The 8-bit images were used in this work, primarily for ease of transfer to the compute server. The uncompressed 16-bit TIFF files presumably contain more information, although it remains to be seen if this information can be used to improve detection.

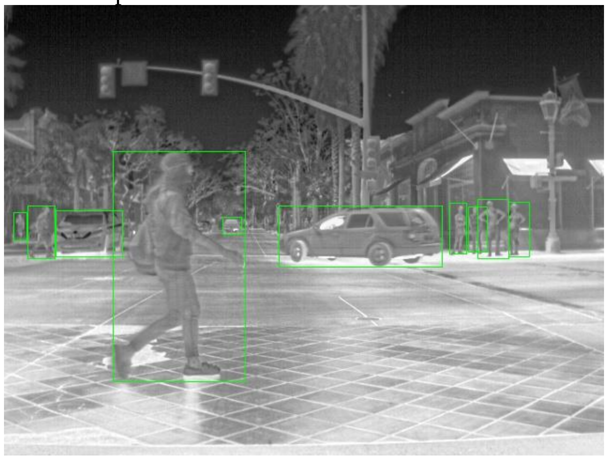

Figure 5: An example image from the validation portion of the FLIR ADAS dataset, with ground truth boxes plotted. People (class 1) and cars (class 3) are visible in this example.

Annotations are provided in an MSCOCO-like format, with one JSON file per image (e.g. FLIR_0001.JPEG has its ground truths defined in FLIR_0001.JSON). Bounding boxes and segmentation polygons are included.

The one-file per image annotation format is not exactly compatible with the COCO format, which provides all annotations in a single JSON file. Although the framework used in this paper supports the COCO format, this discrepancy meant that a custom converter had to be written to compile the annotations into a CSV format, which would then be fed into the framework.

Of note is that the "file_name" metadata in the JSON files was frequently incorrect, sometimes even referring to image filenames that did not exist. It is our assumption that some images were renamed before the public version of the data was released, and others were not published. Accordingly, we use the filename of the JSON files to match to images.

Another idiosyncrasy of the data is the convention for annotation of overlapping bounding boxes; objects which are partially occluded have multiple boxes, one for each visible portion of the object. This is an unconventional annotation scheme, but we assume that the detector will adapt after training.

According to the README document provided with the dataset, there are 5 classes, 4 of which match up with MSCOCO categories, allowing the use of pretrained networks. The fifth class (91 - other large vehicles) would have been mapped to the COCO class for trucks (class 8), but did not occur in any of the images, contrary to the documentation.

Class ID	Class Description
1	People
2	Bicycles and motorcycles
3	Cars and small trucks
18	Dogs

Table 2: Class IDs and descriptions for the FLIR ADAS dataset.

Note that the classes are only partially consistent with MS COCO object classes. This is unfortunate, but the classes are still close enough that transfer learning can be utilized to hasten training.

For many ML algorithms (SSD included), the class balance of classes in the training data influences performance. The balance of the validation data always does. For this reason, we have included counts of the number of ground truth (GT) boxes for each class present in each portion of the dataset.

Class ID	Description	Count of GT boxes
1	People	22372
2	Bicycles	3986
3	Cars	41260
17	Dogs	226

Table 3: Class frequencies for the training data.

Class ID	Description	Count of GT boxes
1	People	5779
2	Bicycles	471
3	Cars	5432
17	Dogs	14

Table 4: Class frequencies for the validation data.

Class ID	Description	Count of GT boxes
1	People	21965
2	Bicycles	1205
3	Cars	14013
17	Dogs	0

Table 5: Class frequencies for the supplementary video data.

4.3. Baseline Algorithm: RefineDet512

The claimed performance of a trained RefineDet512 on the FLIR ADAS dataset is given in Table 1. The RefineDet architecture (published in [2]) is based on SSD-style predictors running on multiple feature layers, but unifies information from each feature layer using what the authors call “transfer connection blocks”.

Unlike SSD, there are two “stacks” of convolutional layers, one being the “Anchor Refinement Module” (ARM) and the other being the “Object Detection Module” (ODM). Information flows from the ARM to the ODM, and the ARM, as the name suggests, produces “refined” bounding boxes which are the basis for further regression in the ODM. This approach retains end-to-end trainability, but mimics two-stage object detection methods in accuracy and operation.

Method	FPS	VOC 2007 mAP	VOC 2012 mAP
SSD300	46	0.772	0.758
SSD512	19	0.798	0.785
RefineDet320	40.3	0.8	0.781
RefineDet512	24.1	0.818	0.801

Table 6: mAP (on VOC) and FPS for RefineDet and SSD, as measured by the RefineDet authors. Summarized from [2].

As can be seen from Table 6, the authors of RefineDet assert that it achieves better accuracy with similar FPS. Notably, the more complex architecture outperforms SSD at the 512 by 512 input size, in terms of both FPS and mAP. Certainly, the performance of this architecture on the ADAS dataset is better than we were able to achieve using SSD.

4.4. Quantitative Results (AP and Precision-recall curve)

Object class	AP
person	0.206
bicycle	0.139
car	0.597
dog	0
mAP	0.236

Table 7: Per-class and mean AP values for the final network.

The maximum mAP achieved was for the configuration with alpha set as 1.0, the custom data augmentation strategy, and SGD optimization.

As can be seen by comparing Table 7 with Table 1, the AP scores are significantly below RefineDet512. Potential reasons for this disparity will be discussed in-depth in the final section.

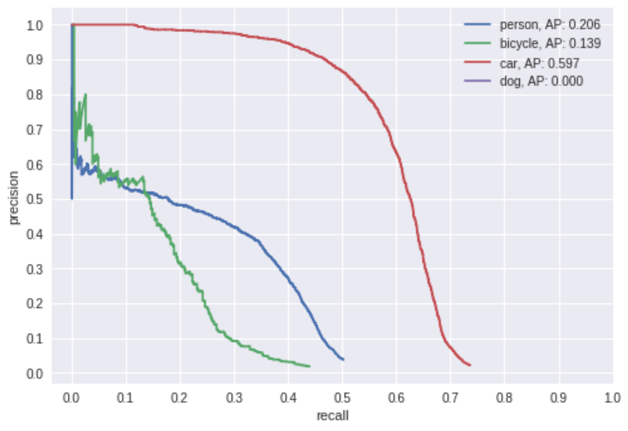

Figure 6: Precision-recall curves for each object class, calculated on the validation set.

4.5. Qualitative Results

In this section, we present the plotted detections, classes, and confidence values for the same network that produced Table 7 and Figure 6. The confidence threshold was set at 0.2, to suppress excess bounding boxes which would have made the images illegible.

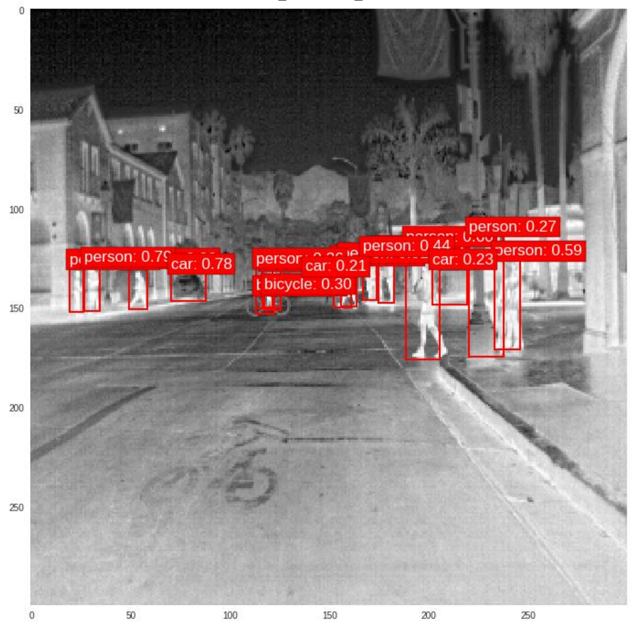

Figure 7: A good example of network performance. Note that the network has split the bounding box for bicycles. As hypothesized, the network has learned to adapt to the dataset's unusual convention for occluded bounding boxes.

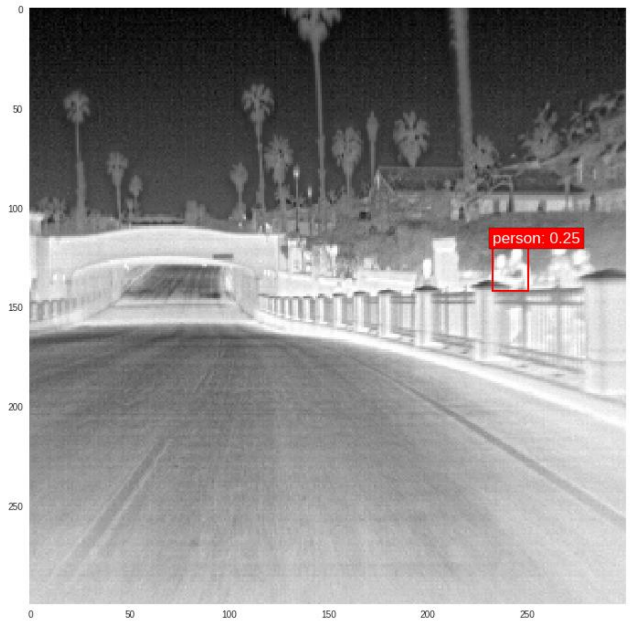

Figure 8: A poor example of network performance. Only one person out of the group is detected, and with very low confidence. The partial occlusion and low contrast background are likely causes.

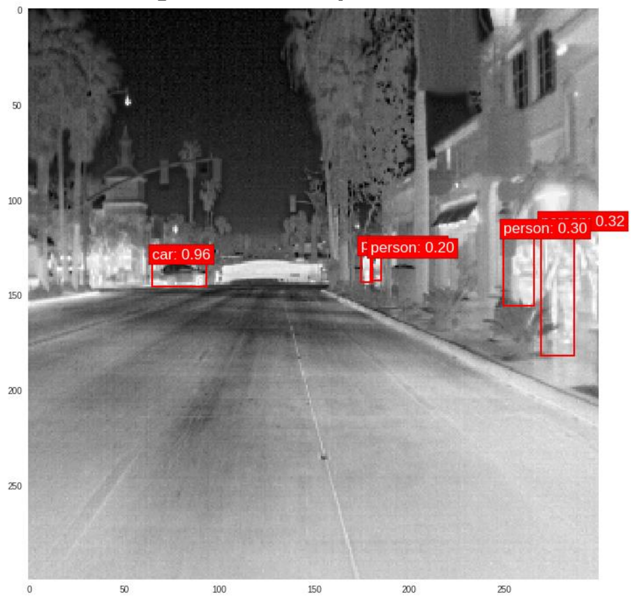

Figure 9: A middling example of network performance. Some human figures are detected, but with very low confidence.

As alluded to in Figure 9, low confidence for all but vehicles (class 3) was a persistent problem. Low contrast backgrounds seem to challenge the detector, but this is expected.

Curiously, there are few problems with small objects, although the literature suggests that this should be an issue. The people in Figure 9 are detected and well localized despite being only tens of pixels wide.

4.6. Computing setup

Google's Colaboratory interface was used for training and evaluating our object detector. It provides access to an NVIDIA Tesla K80 on a shared VM.

Although the GPU resources are sufficient for training many modern ML models, there are some difficulties getting access to large datasets within Google Colab. The VMs are ephemeral (the maximum continuous runtime is 12 hours, after which the notebook must be interactively restarted), and the filesystem is created anew for each runtime. Fortunately, there is an API for connecting to Google Drive, which we used to store our data and model checkpoints. This allowed us to access our training data without re-uploading it every time, and to resume training even if our VM timed out.

Because the VMs were repeatedly interrupted, we do not have an exact value for the training time. However, the final network (that is, the one showing the best mAP on the validations set) trained for over 200 iterations (of 1000 steps each, at a batch size of 8). Assuming approximately 20 minutes per iteration (this varied depending on the amount of data augmentation, and presumably the shared server load), we arrive at a rough estimate of 67 cumulative hours, not counting extra experiments.

5. Discussion

By comparing Table 1 (RefineDet performance, mAP: 0.587) with Table 7 (SSD300 performance, mAP: 0.236) we can see that plain SSD has a performance deficit. We can speculate as to the cause.

5.1. Data balance

One issue with this data set is that it is poorly balanced. The frequency ratio between the most common class (cars) and the least common class (dogs) is 183 for the training set, and 388 on the validation set. For reference purposes, the hard negative mining ratio (limiting the background class) for SSD is 3 [3], which indicates that SSD does not perform well with class imbalances greater than 3.

The problem with this hypothesis is that RefineDet is also vulnerable to class imbalance, since it also uses hard negative mining during training [4]. Table 1 shows that RefineDet achieves over 0.5 mAP on all classes, suggesting that the imbalance did not affect training.

One potential explanation is that the baseline detector was trained on more (and presumably better balanced) data than was made publicly available by FLIR systems. As mentioned in Section 4.2, some image metadata is incorrect in a way that indicates data was removed and renamed prior to publication. There is also the matter of the missing object class mentioned in the same section.

5.2. Alpha value

Another possibility that we explored was that the bounding box annotations of this dataset were more demanding than those of VOC or COCO, and required a stronger localization loss.

The ADAS dataset splits occluded bounding boxes and prioritizes tight bounding boxes over complete inclusion. For example, the bounding box for a bicycle might be split into two by its rider, and a bounding box for a person with arms raised might only include the head and torso.

To test this, we fine-tuned our network for several tens of thousands of steps using an alpha value of 4. (That is, the localization loss was assigned 4 times the weight of the classification loss). In its published and default configuration [3] [5], SSD uses an alpha value of 1.0.

Object class	**AP**
person	0.201
bicycle	0.131
car	0.594
dog	0
mAP	0.231

Table 8: The AP and mAP values for the network with increased localization loss.

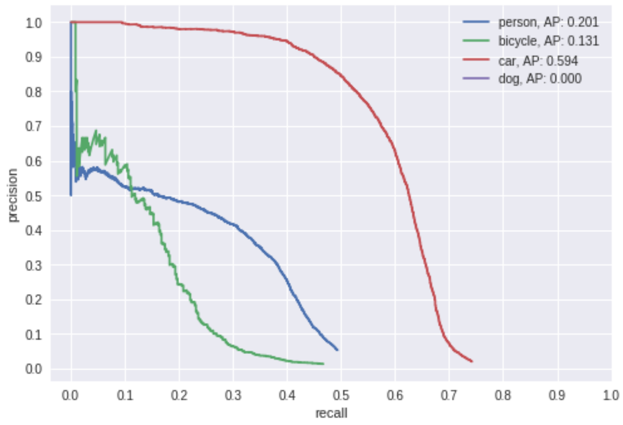

Figure 10: The precision-recall curves for the network with increased localization loss.

Performance does not increase with the higher localization loss, and instead regresses slightly. We can try to explain this by looking at the predicted bounding boxes for a few objects.

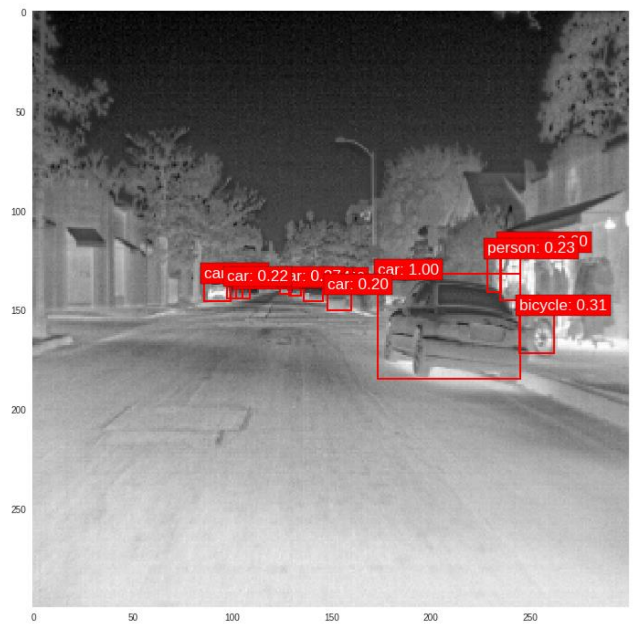

Figure 11: Example detections plotted for the network with increased localization loss.

In Figure 11, we see that the greater weight for the bounding box loss term leads to notably tighter bounding boxes. Compare to Figure 7, which leaves significant "slack space" in the bounding boxes for nearby figures.

Still, this leads to worse performance in terms of mAP, probably because fitting the bounding box better than IoU 0.5 does not increase AP.

5.3. Pre-trained weights only

As mention in Section 3.4 and 3.2, we began training using pretrained weights for the COCO dataset. We also evaluated the performance of these weights before additional training.

Object class	**AP**
person	0.187
bicycle	0.053
car	0.329
dog	0
mAP	0.142

Table 9: AP score achieved by the pre-trained COCO weights, prior to any training on the ADAS dataset.

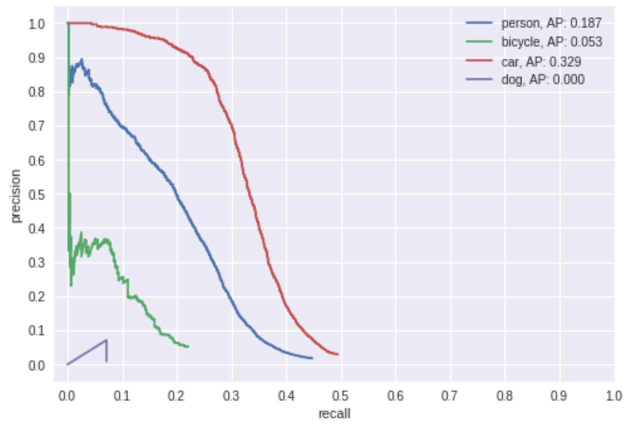

Figure 12: Precision-recall curves for the pre-trained weights.

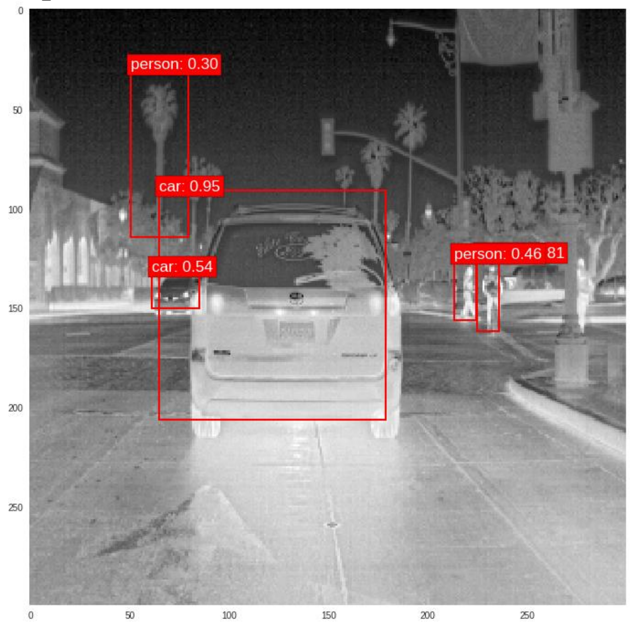

Figure 13: Example detections for the pre-trained weights.

Performance was surprisingly good, which is why we chose to use these weights as a basis for further training. Poor localization and commonly mistaking upright objects (plants, light posts, etc.) as people are issues with these weights.

6. Acknowledgements

The project's GitHub repository may be found at: https://github.com/group17-CAP5415/cap5415-group17-final

References

[1] Walber, "Precisionrecall.svg, CC BY-SA 4.0," [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=36926283.

[2] L. W. X. B. Z. L. S. Z. L. Shifeng Zhang, "Single-Shot Refinement Neural Network for Object Detection," 3 January 2018. [Online]. Available: https://arxiv.org/abs/1711.06897. [Accessed 3 December 2018].

[3] D. A. D. E. C. S. S. R. C.-Y. F. A. C. B. Wei Liu, "SSD: Single Shot MultiBox Detector," 29 December 2016. [Online]. Available: https://arxiv.org/abs/1512.02325. [Accessed 3 December 2018].

[4] L. W. X. B. Z. L. S. Z. L. Shifeng Zhang, "Single-Shot Refinement Neural Network for Object Detection," 29 December 2016. [Online]. Available: https://arxiv.org/abs/1711.06897. [Accessed 3 December 2018].

[5] P. Ferrari, "SSD: Single-Shot MultiBox Detector implementation in Keras," 3 December 2018. [Online]. Available: https://github.com/pierluigiferrari/ssd_keras.

[6] "fix for case where some classes don't have predictions," [Online]. Available: https://github.com/pierluigiferrari/ssd_keras/pull/155. [Accessed 3 December 2018].

[7] "Adam: A Method for Stochastic Optimization," 30 January 2017. [Online]. Available: https://arxiv.org/abs/1412.6980.