Human and Seat Belt Object Detection for Tracking

Abstract

We built an object detection model that detects and localizes humans with a precision of 99.5% and recall of 97.6% and seat belts with a precision of 97.2% and recall of 89.7% in the backseat of vehicles. The model ran on NVIDIA Jetson Xavier[1] module with an FPS of 45 in isolation. We selected a pretrained SSD Inception v2[2] trained on COCO[3] dataset. The model was then fine-tuned on our training dataset of 12000 and tested on 3500 images of the hold-out test set and 32 videos.

Figure 1: Examples of human detection 1a (left) and seat belt detection 1b (right) in images

Introduction

At Passenger AI, we want to track passengers and determine if they are seated with seat belts on. We need to detect and localize humans to track them[4]. Seat belt detection is used to detect if a passenger has their seat belt on or not.

One of our other requirements is to be able to run any solution on the edge, for our case, NVIDIA Jetson Xavier. An edge device has limited computing capability, it has a less powerful GPU compared to most desktop and/or commercial GPUs in the market. Hence the model we train also needs to be able to run on the edge.

Object detection is a common approach for detecting and localizing objects in images. It is straightforward to collect and annotate data for and well supported in the Machine Learning community.

Requirements

The goal of the project was to have a precision of 95%+ for both human and seat belt class and recall of 95%+ and 70%+ for human and seat belt class respectively. The requirements were set by the Engineering and Product team to achieve a suitable performance from the passenger tracker for the human class. Since the task of tracking a passenger is more mission-critical then detecting if a seat belt is worn by the said passenger, the requirements on the human class are higher.

Data

The most important part for any Deep Learning model is the data, collection and annotation. We collected data for humans in different lighting conditions, seating positions, different cameras and camera angles, different cars and using different actors.

Data set split	Actors	Cars	Count
Train	19	9	9000
Validation	4	4	3000
Test	6	5	3500

Table 1: Breakup of actors, cars and total images in each dataset split.

	Human	Seat belt
Train	12891	6783
Validation	1751	818
Test	2750	1981

Table 2: Breakup of total instances of human vs seat belt class in each dataset split



Figure 2: Different scenarios, fighting in 2a (left), hugging in 2b (left center), leaning forward in 2c (right center) and sideways in 2d (right)

Cameras



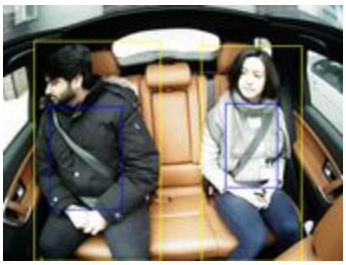

Figure 3: Different cameras, fisheye camera in 3a (left) and non-fisheye camera in 3b (center) with a fisheye from a higher perspective in 3c (left)

We used a variety of different cameras and camera angles to help with model generalization. We don't have a split of how many data points per camera. Figure 3a shows the fisheye camera ELP 2.1mm w/ Sony IMX322[5] sensor, it has a 170-degree field of view (FOV). Figure 3b shows non-fisheye ELP camera with a 100-degree FOV[6] and figure 3c shows the same fisheye camera but from a higher perspective and looking down at a steeper angle.



Figure 4: Human annotation from head to toe in 4a (left) and seat belt annotation for only seat belt visible on a human's torso in 4b (right)

Annotation

For object detection, annotations are done in the form of bounding boxes around the item of interest. The annotations usually follow a convention, which dictates in what format the annotations are stored. We followed the Pascal/VOC[7] format for this project.

Humans are annotated from toe to head, which is convention in the COCO dataset, and they are annotated only if they are completely inside the vehicle cabin and in the back seat of the vehicle, since we only wanted track humans inside the vehicle. Figure 4a shows a human annotated.

Seat belt is annotated only if they are worn across the torso of a human, this was decided to reduce the cost of data annotation of having to annotate each visible seat belt in the cabin which would not have any impact on the model training. From Figure 4b, you can see the seat belt on the right not annotated while the seat belt on the left annotated as 'seatbelt' class as it is worn by a human.

We used different annotation tools for annotating the dataset. The most common tools in the industry are LabelImg[8], RectLabel[9] and LabelMe[10].

Collection

When collecting the dataset we considered different scenarios that the model would have problems detecting. The model has been pretrained on the COCO dataset that has common objects in context. Although one of the classes was human class, the variety of humans that the model is used to seeing would be drastically different from what it would encounter in practice with Passenger AI due to the fact that the COCO dataset would have humans in images would always be at a farther perspective compared to the closer perspective of the back seat of the vehicle. Keeping that in mind we tried collecting images of different actors in several different cars. The distribution of how many actors we used across how many different vehicles are given in Table 1.

One of the issues we faced during training is the class imbalance between instances of the human class and the seat belt class, summarized in Table 2. As seen from the annotation section, a positive instance of a seat belt class is annotated only if it is worn across the torso of a human, hence even if collect 1000 more images of seat belt class, it will add at least 1000 more instances of the human class as well. To help with this we used the concept of class weights, which assigns a higher cost to a misclassification of a particular class so during training the model assigns a higher loss value and hence allows that class to have a higer importance during training. Not only does it help with the class imbalance, we can use it to artificially assign an added weight to the human class as that class has more stringent requirements.

As seen from Table 1, we collected upwards of 3000 images to gather metrics on the precision and recall of the model.

Videos

We use the signal from object detection, bounding box coordinates of detected humans, to track passengers. The tracker can gracefully handle sporadic false positives and false negatives, but it cannot handle consecutive false positives or false negatives. An image-based test set can give us metrics on precision and recall of the model but to test for consistent failure, we collected a dataset of 32 videos with normal and difficult conditions to test for consecutive failures in the tracker.

Methods

Object Detection

Object detection fits the acceptance criteria for this project and have several open source implementations on github. Since we want to detect humans so that we can track them, the solution needs to reliably detect and localize the instances of the human class. Object detection gives us exactly that.

We use object detection along with pose estimation[11] to aid in the reliable detection of humans inside vehicles, but pose estimation fails consistently in cases where the majority of the keypoints are occluded. Keeping this in mind, we preemptively collected images of failure cases of pose estimation, examples can be seen in Figure 4.

Backbone Selection

The solution we select needs to run on the edge. The most reliable way to run an object detection model on the edge was to use NVIDIA's TensorRT[12] to optimize the model layers to run efficiently on the selected platform. At the time of training, only two models could be reliably converted to TensorRT engines - YOLO v3[13] and SSD Inception v2. In Table 3, we compare these pretrained models on COCO dataset along with a pretrained RetinaNet ResNet 50, as at the time it was close to the theoretical maximum and we had used the model before and would give us an appropriate baseline. The models were evaluated on our hold out test set. This would give us a signal if the model improved when we evaluate our trained model on the same hold-out test set, which can be seen in Table 3.

Person Detection	F1 Score - pretrained	F1 Score - fine-tuned
RetinaNet ResNet 50 (512x512)	0.87	0.98
SSD-Inception v2 (300x300)	0.84	0.967
YOLO v3 (416x416)	0.85	0.94

Table 3: Performance of three pretrained models on our hold out test set and the performance of the same models fine-tuned on our dataset on the same hold-out test set

As you can see from Table 3, the SSD-Inception v2 was a winner against YOLO v3 and hence was chosen for this task.

Training

We used the Tensorflow Object Detection API[14] (TODAPI) to train the SSD-Inception v2. It takes away all the code implementations for different training artefacts like optimizers, learning rate and schedule and replaces them with a single configuration file.Implementing the above from scratch would have taken up man hours and also lead to implementation errors as with any software. Using the API we can rest assured that the software has been well tested and is supported by the developers. We used the default SSD-Inception v2 config that was used to train on the COCO dataset with minor modifications that we will discuss below.

Hard example miner

For most detectors like SSD, we make far more predictions than the number of objects present. So there are much more negative matches than positive matches. This creates a class imbalance which negatively affects training. We are training the model to learn background space rather than detecting objects. However, we need negative sampling so it can learn what constitutes a bad prediction.

We saw that false positives start to rise up so we added more negatives in our training and accordingly adjusted the `min_negatives_per_image` to `10` from `1` to make the model learn the background class.

Learning rate and optimizer

For optimizer, we went with the default choice of RMSProp[15] that the SSD-Inception v2 was pretrained with. We lowered the learning rate by a factor of 10 to `0.004` to avoid the sudden spikes we saw for loss during training.

Augmentations

We chose the augmentations that we expected to see in practice. They include random cropping, rgb to grey, random brightness, hue, saturation, contrast, saturation and color adjustments and horizontal flipping[16].

Class weights

We explicitly assigned class weights for training as we wanted the model to penalize wrong prediction on the human class more than the seat belt class, as we discussed in the requirements section above.

Results

Below, in Table 4, we share the results of training an SSD-Inception v2 on our dataset and evaluating on the hold out test set.

Class	F1 Score	Recall	Precision
Human	0.986	0.976	0.995
Seat belt	0.933	0.897	0.972

Table 4: F1 score, precision and recall of the human and seat belt class in the test set.

From Table 5 below, we can see that the model performed well on videos too except for two videos that were considered challenging for this task. A particular video is considered failed if there are more than 10 false positives or more than 10 false negatives consecutively detected by the model on that particular video in question.

Video difficulty	Number of videos	Number of videos failed due to false positives	Number of videos failed due to false negatives
normal	17	0	0
hard	15	1	1

Table 5: Number of videos in each category of difficulty and number of videos failed

In Table 6, we can see the two examples in red as failure cases, the rest are considered to have passed the evaluation. The goal of the table below is to contrast different videos and how many frames are present in each video, number of passengers and the difficulty of the video for the model. In case you want to to look at the whole table with all videos mentioned, you can find it here[17].

Video name	Difficulty	Number of people	Number of Frames	Max Consecutive False Positives	Max Consecutive False Negatives
Video-1	hard	2	331	82	4
Video-2	hard	1	710	1	14
Video-3	hard	2	501	6	2
Video-4	hard	1	555	0	0
Video-5	normal	1	537	0	0
Video-6	normal	3	497	0	0

Table 6: Example videos from the video test set

We were able to convert the model to TensorRT and deploy it on the Inference Engine[18]. In isolation, we were able to run the model at 45FPS. When we ran the model alongside other models and detectors, we were able to achieve 22FPS.

Discussion

The model fulfils all the project requirements, but it still has room for improvement. In this section, we look at some of the edge cases and talk about how we can mitigate them. The most common notable edge cases are discussed below

Edge case analysis



Figure 5: Showing the different failure cases for the model. 5a (right) showing the failure case due to humans being merged together. 5b (center) showing the failure case due to an added bounding box being detected when fighting. 5c (left) showing the failure case due to a false positive on the hand being mistaken as a seat belt.

Figure 5a shows the edge case where the bounding box of the human on the left and the human on the right get merged into one box making the model thinking that there is only one human in the cabin. This can be solved through SoftNMS[19] which would lower the confidence score of the lower confidence human detection as opposed to merging the two human detections into one as done by normal NMS[20].

Figure 5b shows how the model can be confused when there is a lot of movement in the cabin, the model picks up three humans in the cabin instead of of the two that are present. This is also caused by having too high of a threshold for merging detections using NMS. By using SoftNMS we can bypass the problem of choosing the correct IOU threshold. In this case, since the third false positive is lower confidence, softNMS will lower the confidence of this detection even further and cause it to get suppressed.

Figure 5c shows a false positive of the seat belt class, this mostly happens due to the arm of the human being across their torso confusing the model into thinking that this too is a seat belt. This can be mitigated by collecting more data of humans with their arms across their torsos and not annotating the seat belt on the arm, hence feeding the model more instances of the negative class.

Future work

Torso+Head only detection

From the above work, we can see that merging passengers due to over-extended limbs can be a problem, one of the ways we can solve this is by only annotating head and torso as human and excluding the limbs from the annotation, this will lead to fewer human box merges.

Sharing backbones

One of the ways to speed up execution of multiple models on the edge is by making them share a single backbone, which would work as a feature extractor and the task-specific layers can then do their job on the input of the shared feature extractor

Re-ID

To use the tracker we need to make sure that the passengers are never lost and if they are, then they get assigned back to their original identity. As of now, we use the tracker for assigning identity and if the identity is lost, we assign the last known identity at a physical location to reassign the lost identity. As you can imagine, this can lead to failures where the state of the cabin has changed, for example, the passengers have switched seats. Re-ID would be valuable here as it would use the visual features of detection to reassign a lost identity.

Conclusion

In conclusion, we successfully built an object detection model that detects and localizes humans and seat belts in the back seat of vehicles, and we surpassed the goal of 95%+ precision and recall on human class and 95%+ precision and 70%+ recall on the seat belt class achieved on the test dataset of 3000 images. We began by collecting a dataset of 12000 training and validation images in a variety of cars and with a variety of different actors. We then fine-tuned a COCO-pretrained SSD-Inception v2 model. We conclude that the project was a success as the goal metric was satisfied with a wide margin.

References

[1] -Jetson AGX Xavier, https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit
[2] - Chengcheng Ning, Huajun Zhou, Yan Song, Jinhui Tang: Inception Single Shot MultiBox Detector for object detection, https://ieeexplore.ieee.org/document/8026312
[3] - Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence

Zitnick, Piotr Doll ar: Microsoft COCO: Common Objects in Context, https://arxiv.org/pdf/1405.0312
[4] - Ali Alavi, Simple Tracker, passengerai/device/src/tracker/SimpleTracker.py
[5] - ELP USB Camera Fisheye Sony IMX322, https://www.amazon.ca/ELP-IMX322-Sensor-Camera-Module/dp/B06ZXX1C56
[6] - ELP Autofocus USB camera, http://www.webcamerausb.com/elp-usb-camera-module-1080p-hd-industrial-usb20-camera-with-autofocus-lens-p-86.html
[7] - Pascal Visual Object Classes challenge, http://host.robots.ox.ac.uk/pascal/VOC/voc2012/
[8] - Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg
[9] - RectLabel, https://rectlabel.com/
[10] - Kentaro Wada: Image Polygonal Annotation with Python, Git code (2016), https://github.com/wkentaro/labelme
[11] - Papers With Code: Pose Estimation, https://paperswithcode.com/task/pose-estimation
[12] - NVIDIA: TensorRT, https://developer.nvidia.com/tensorrt
[13] - Joseph Redmon, Ali Farhadi: YOLOv3: An Incremental Improvement, https://arxiv.org/abs/1804.02767
[14] - Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S, Murphy K, CVPR 2017: Speed/accuracy trade-offs for modern convolutional object detectors, https://github.com/tensorflow/models/tree/master/research/object_detection
[15] - Sebastian Ruder: An overview of gradient descent optimization algorithms, https://arxiv.org/pdf/1609.04747
[16] - Connor Shorten, Taghi M. Khoshgoftaar: A survey on Image Data Augmentation for Deep Learning, https://link.springer.com/article/10.1186/s40537-019-0197-0
[17] - Performance of model on videos test set, https://docs.google.com/spreadsheets/d/1KLofmaYcNE8lwXaknIEvbejj0sCR01ZegH8_kwHDXZ4/edit?usp=sharing
[18] - Ali Alavi: Inference Engine, passengerai/inference-engine/README.md
[19] - Navaneeth Bodla, Bharat Singh, Rama Chellappa, Larry S. Davis: Soft-NMS -- Improving Object Detection With One Line of Code, https://arxiv.org/abs/1704.04503
[20] - Jan Hosang, Rodrigo Benenson, Bernt Schiele: Learning non-maximum suppression, https://arxiv.org/abs/1705.02950