Background

This working document is an updated summary for a BDD project, titled Pedestrian Detection for Autonomous Driving using Deep Learning. The purpose of this document is to provide the latest status, with descriptions of the technical approach that have been explored in this phase of the project. It is intended to provide an outline of next steps and serve as a foundation for future research work.

Problem Statement

Perception is a key challenge for self-driving cars. Unlike vehicles that are big metal objects, pedestrians can be difficult to detect. Pedestrian detection is now being intensively studied as a vision problem. While advanced apparatuses like binocular cameras and data fusion with lidar/radar are still considered, attention is being shifted to monocular RGB cameras.

Pedestrian detection falls under the vision problem of image classification and object detection. Recently, deep learning techniques have proven to be very efficient, closing in on human benchmarks. Pedestrian detection however, has proved to be a harder problem. Real-world pedestrian detection involves pedestrians that are small, sometimes occluded or in groups, and in a lot of cases, no pedestrian will be present.

Discussions of Technical Approaches

For detection tasks, best performing deep-learning algorithms involve Region-based CNNs (or R-CNNs) and its advanced forms such as Fast/Faster R-CNN. The current leading pedestrian detectors are so-called hybrid methods that use R-CNNs after handcrafted features such as Selective Search (SS) or Histogram of Gradients (HOG).

Failures of deep learning models for pedestrian detection often result from low resolution feature maps and false positives. Where handcrafted features generate high resolution features, resolution decreases across layers in CNNs, which leads to poor detection for small objects such as pedestrians far away. Moreover, although object detection errors come from misclassification, pedestrian detection errors are largely dominated by false positives caused by similar background objects (poles, mailboxes, signs, etc.).

Proposed Methodologies

![Proposed Methodologies](image-url)
**Figure 1. Proposed Architecture with RPN and LSTM**

We present a deep learning detector similar to Faster R-CNN, without Region-of-Interest pooling and the additional classifier to deal with the low resolution feature map limitation. In addition, we propose to investigate the use of mixed Long Short-Term Memory (LSTM) units and CNNs to provide context both in space and time. Figure 1 illustrates our proposed setup.

We use the VGG architecture as our main CNN, specifically the model 16-D. We also make the RPN more complex to cope with the removal of the classifier, by adding a second shared layer before the regression and classification layers, and making those layers deeper.

Our second contribution is the idea of using LSTM units to add temporal context to the detector. Our proposed architecture, by removing the classifier network, is much more suited to have LSTM units: the output of the classification layer of the RPN is always constant over time, which means we have the required continuity to use LSTMs.

Finally, it is worth noting that, similarly to how a convolutional layer is essentially one unit applied over the input map with a stride, we will use a similar method for the LSTM layer. This means that the number of added parameters remains very small, while enabling temporal context to be integrated.

**Implementation**

We implement our approach in Python, using the open-source library TensorFlow\(^1\) by Google. TensorFlow is a numerical computation library that uses data flow graphs. It provides ready-to-use implementation of several machine learning algorithms, while retaining a lot of flexibility. We rely on Amazon’s AWS Cloud for training our RPN architecture. We set up an on-demand GPU instance (g2.2xlarge\(^2\)) with Ubuntu 16.04 LTS. For remote monitoring of our experiments, we used TensorFlow’s visualization software TensorBoard. Due to the limited time, as of this report, the LSTM layers are not included in the current implementation. The codes are publicly available on GitHub: [https://github.com/buffer51/lstm-rcnn-pedestrian-detection](https://github.com/buffer51/lstm-rcnn-pedestrian-detection)

**Experiments & Results**

We evaluate our method on the Caltech Pedestrian Dataset detection benchmark. This dataset consists of 10 hours of 30 fps recordings, with a 640x480 resolution. There are around 2,300 pedestrians annotated by 350k bounding boxes. The benchmark described in sets half of the frames for training while reserving the rest for evaluation. We will also set aside ⅓ of the training set for validation. 4,000 examples were used in the training process using AWS instances with GPU support. Examples were randomly sampled from the complete 120k training set, but retrospectively could have been selected as 1 frame per second of the videos. Training the model with 4,000 examples takes an hour per epoch, and 15 hours for a complete training, which limits trial and error cycles. The aspect of computational limitations is expected to be overcome as we implement a high-end computing station in coming months.

To conform with their evaluation procedure, we will output bounding boxes of detected pedestrians for all those frames, along with a confidence score. Those boxes are matched with the ground truth for pedestrians; if a detection box has an IoU over 0.5 with a ground truth, it is considered as a positive detection. Missed ground truths are counted as false negatives, while unmatched detections are false positives. Final results are represented as Miss Rate (percentage of ground truths not found) vs False Positives per Image (FPPI).

Figure 2 shows the performance on the Caltech Pedestrian Dataset benchmark, along with a

---

1. [https://www.tensorflow.org](https://www.tensorflow.org)
2. [https://aws.amazon.com/fr/ec2/instance-types/](https://aws.amazon.com/fr/ec2/instance-types/)
broad range of other detectors, including current leading detector RPN+BF. The final log-average miss-rate of our detector is high (70%, the left in Figure. 2), which is quite poor (the lower the better). It performs worse than HOG, which is a standard but an old detector that relies on handcrafted features. However, we diagnosed our system with a poor localization problem; as shown in Figure 6b, results on the easier benchmark with a 0.25 IoU threshold are encouraging (36%, the right in Figure. 2). This means that detections from our method are not entirely wrong, but are loosely located on the pedestrians. The precision of the localization of detection has room for improvement. Where other detectors’ log-average miss-rate decreases by 2-3% between the easier 0.25 IoU threshold and the reasonable set, ours drops by 29%.

![Figure 2. (left) Benchmark on the Reasonable Set (Ours is LSTM-RCNN), and (right) Benchmark on an Easier Set with IoU Threshold of 0.25](image)

**Conclusion & Future Works**

We presented a deep learning approach to pedestrian detection, making use of Faster R-CNN’s Region Proposal Network as a standalone detector. We exposed several issues that arisen from the specificity of pedestrian detection in the object detection field. While we were able to achieve good results on the classification front, our results are harmed by too many false positives and less than desirable localization of detections. An independent research showed that our basic idea, which was to use RPN as a pedestrian detector, can reach state-of-the-art performance, provided hard negative examples are mined.

Though we named our method LSTM-RCNN, we have not yet implemented LSTM units as described in our architecture. We strongly believe that, combined with a properly trained RPN, this method could improve on the current state-of-the-art. As several analyses of best performing detectors on Caltech Pedestrian Dataset have shown, detecting small and occluded pedestrians is still an open challenge. Temporal context has not been explored except for optical flow, and LSTMs represent an opportunity to outdo current detectors.

We postulate that simple end-to-end techniques such as RPNs and LSTMs could, unlike complex methods, produce great detections while keeping a low footprint, hopefully towards real-time processing.

**Next Steps**

We are currently building a computing workstation with four latest GPU products, and expect to speed up the training processing and algorithm developments. We will implement LSTM elements in our next round of coding and testing.