An Analysis on Increasing Bandwidth Within Pedestrian Safety Applications Using Machine Learning and Lossy Compression

Heath W. Gerrald
Department of Electrical and Computer Engineering
Clemson University
Clemson, SC, USA
hgerral@g.clemson.edu

***Abstract*—As vehicles continually transition to becoming more autonomous, concerns over their safety and potential hazards are growing more prevalent. One such concern is their ability to accurately detect pedestrians in real time. While pedestrian detection alone is already common and becoming increasingly more accurate, our team is working to create a network that can relay knowledge of the presence of pedestrians. The goal is to install a camera at potential intersections, continually collect images to run through a machine learning algorithm for pedestrian detection, and relay this information to nearby vehicles - all as close to real time as possible. One of the main issues with such a network is a limited bandwidth which prevents images, and even worse so videos, from transmitting at fast speeds. To combat this, I contributed to this project by finding methods of compressing image data to allow data transfer to be closer to real time while maintaining accuracy in pedestrian detection.**

***Index Terms*—lossy compression, libpressio, pedestrian detection, machine learning, bandwidth**

I. Introduction

Pedestrian detection is an absolute necessity in autonomous vehicles, as without this feature the resulting injuries could be catastrophic. This past year (2019) we experienced a 30-year high in pedestrian deaths due to automobiles with 6,590 reported deaths [1]. Should society transition to a standard use of self-driving cars, pedestrian detection technology will be crucial as it should decrease the number of deaths theoretically to zero.

The goal of this project is to help combat pedestrian deaths by implementing a camera network capable of both communicating in real time and detecting pedestrians with machine learning. Unfortunately, the communication quickness is limited by bandwidth and must be compressed to travel at reasonable speeds. Compression, however, deters accuracy in detection, and with human lives on the line, accuracy should be as close to perfect as possible. In this project, I worked to find methods of compressing these pedestrian images, but maintaining accuracy rates as high as possible. I worked heavily with a software called *Libpressio*, which was developed and is currently maintained by Robert Underwood [3].

II. Background

A. Related Work

The work in this paper is served to be an extension of the research conducted by Mizanur Rahman, Mhafuzul Islam, Jon Calhoun, and Mashrur Chowdhury in their paper *Dynamic Error-bounded Lossy Compression (EBLC) to Reduce the Bandwidth Requirement for Real-time Vision-based Pedestrian Safety Applications* [2]. Their paper describes the methods they used to perform an error-bounded lossy compression on images and how it affected their machine-learning module's detection rates. Since environmental factors play a significant role in camera quality, they discovered ways to dynamically adapt video compression levels based on several weather conditions. Their software automatically determines the constant rate factor (CRF) and corresponding peak-signal noise ration (PSNR) that best fits the weather conditions and still performs video compression with an acceptable accuracy. They decided any compression that made the module below 97% accurate would not be practical. Their concluding results are shown below in **Figure 1**.

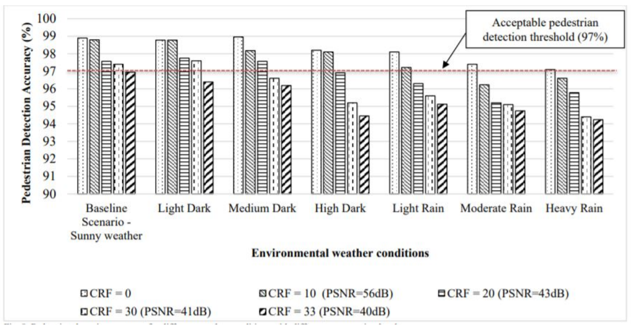

Figure 1 - Pedestrian detection accuracy for different weather conditions with different compression levels

B. Expansion of This Work

To take their research a step further, this past year I researched ways to implement intra-frame compression tolerances to further improve the compression ratio. In my image data set, there are certain regions of the image where pedestrians would likely never be, and if so would not be in any danger of being hit by a vehicle. An example of one of the images from my data set is shown in **Figure 2**.

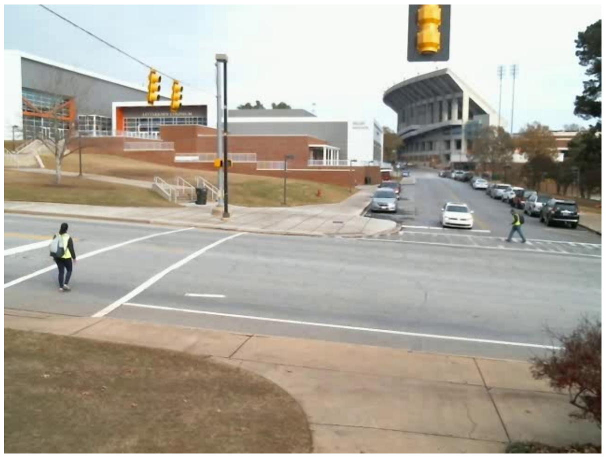

Figure 2 - Example image from experimental data set

Obviously, pedestrians are only in danger to autonomous vehicles if they are in the road, and as one can tell the road only takes up a fraction of the above image. Therefore, these sections are not important and can be more heavily compressed than the rest of the image.

III. Compression Techniques

A. JPEG Cropping

The image data I was experimenting with were JPEGs, short for Joint Photographic Experts Group. JPEGs are actually a lossy codec (compressor/decompressor) designed for photographs that use a highly sophisticated compression algorithm [5]. Whenever a file is saved as a .jpeg extension, operating systems today know to compress these files using the JPEG compression algorithm and know how to decompress them when need be. My initial attempts to compress different regions of an image involved cropping a JPEG image, compressing that specific section, and reattaching the newly compressed image to the original. While this is achievable, this method will not in fact save any disk storage and will many times make the image take up even more memory. This is due to the sophisticated JPEG compression technique being implemented on top of each other.

Another immediate idea would be to convert the JPEGs to a different file type to perform the compression, or use a different image format entirely. However, these methods are either too time consuming or do not perform enough compression; there would be no way to perform calculations in real time. For these reasons, it was determined that I needed to write some software to compress separate regions before JPEG perform its own compression.

B. Libpressio

Libpressio is a C++ library with Python bindings used to implement SZ, ZFP, MGARD, BLOSC, FPZIP, and a handful of other compressors on image files. It allows users to write application level code that requires only minute changes to switch between compressors. For this reason, it is a great tool to analyze the effects of different compression techniques. For the majority of my project, I used the SZ compressor and experimented with how differing PSNR values affected compression and detection accuracy. A lower PSNR will have more compression on an image as it decreases the variance of the values pixels can hold.

To implement Libpressio, I wrote Python code to read in an image, tag the sections of the various compression layers, and apply Libpressio with differing PSNR values to each section. It would have been better to code in C since Libpressio is primarily meant for C programs, but the Python library *Pillow*, a fork of the Python Imaging Library, allows for easy reading in and writing out of JPEG images. To do this in C would have required manipulating the JPEG source code, something I did not have time for in the course of a year.

C. Explanation of Compression Layers

As mentioned earlier, some areas of the image need more visibility than others, so I did a simple breakdown on my images to create different regions. Each image contains 3 horizontal regions, the top being the most compressed, the middle having slightly less compression, and the lower with only little to no compression.

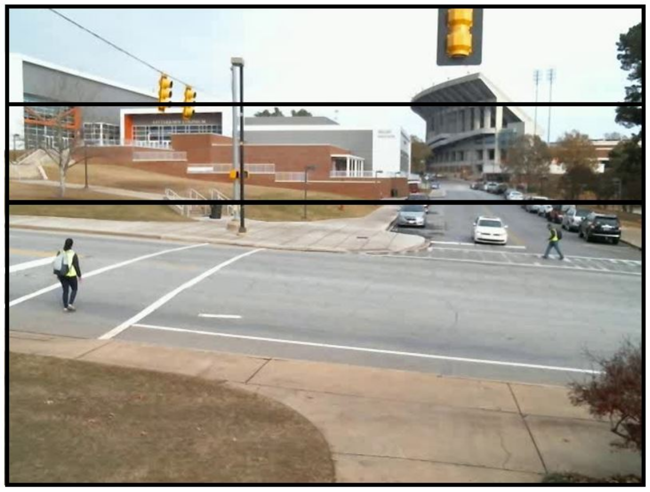

Figure 3 - Breakdown of the various layers of compression

I experimented with using different PSNR values using the SZ compressor of Libpressio, creating several uniquely compressed images. After creating each data set of 426 images, I was then able to run the image set through the pedestrian-detection module and determine how the compression affected detection accuracy.

IV. Running the YOLOv3 Model on Image Sets

A. Summary of the Machine Learning Module

For the purposes of my project, I used the same machine-learning module that Rahman, Islam, Calhoun, and Chowdhury used in their experiments. This helped keep the experiment controlled and take out any unnecessary variation in the results. This module using machine-learning to create bounding boxes around features detected in the image and identify which ones are pedestrians. To train the module, the videos are sampled at 10 frames per second, uses the Pascal Visual Object Class (VOC) to annotate each frame, and generates possibilities of several regions to determine whether a bounding box should be placed around it [2]. Since it is not uncommon for multiple bounding boxes to border a pedestrian, the team uses the non-max suppression algorithm to eliminate false positives [4].

B. Description of Palmetto and the Setup Process

V. Analysis and Results

A. Data Reduction and Detection Comparison

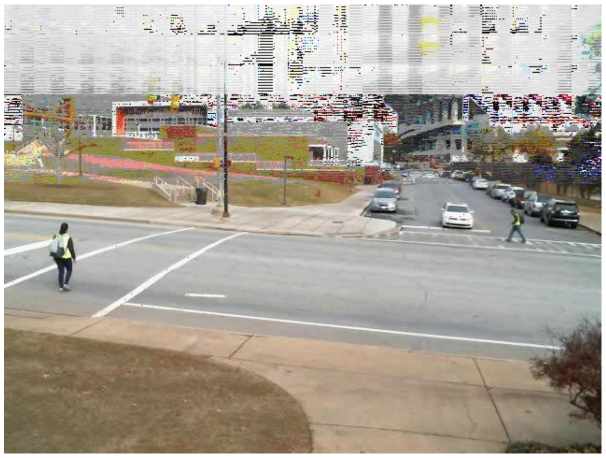

Figure 4 - Compressed image using PSNR values of 8, 20, and 50, respectively

B. Possible Errors

VI. How This Work Could be Further Improved

A. Using Different Compressors

Libpressio is new and installation can be very time consuming - the chance to use some of the other compressors would have been great.

B. Further Research on Different Compression Layers

VII. Conclusion

Acknowledgment

References

[1] Laris, Michael. "Distractions, Drinking and Darkness Contribute to 30-Year High in Pedestrian Deaths, Report Says." The Washington Post, WP Company, 27 Feb. 2020, www.washingtonpost.com/local/trafficandcommuting/distractions -drinking-and-darkness-contribute-to-rise-in-pedestrian-deaths-report-says/2020/02/26/71a93408-58f0-11ea-9b35-def5a027d470_story.html.

[2] Rahman, Islam, Calhoun, Chowdhury. Dynamic Error-bounded Lossy Compression (EBLC) to Reduce the Bandwidth Requirement for Real-time Vision-based Pedestrian Safety Applications, 2019.

[3] Robert Underwood 2019, Libpressio v. 0.38.0, https://github.com/robertu94/libpressio.

[4] R. Rothe, M. Guillaumin, and L. Van Gool, "Non-maximum suppression for object detection by passing messages between windows," In Asian Conference on Computer Vision (pp. 290-306). Springer, Cham. , 2014.

[5] Wake, Lewis. "What Is a JPEG File?" Digital Communications Team Blog, 4 Apr. 2019, digitalcommunications.wp.st-andrews.ac.uk/2019/04/08/what-is-a-jpeg-file/.