An Analysis on Using Lossy Compression to Increase Bandwidth Within Pedestrian Safety Applications

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 commonplace and becoming increasingly more accurate, there is much research today focused on trying to create a network capable of communicating 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 of which 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. Specifically, I focused on segmenting images into separate regions, compressing each section according to its likelihood of containing a pedestrian, and creating new, further compressed image sets to run through a machine-learning module built for 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 [5]. 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 while 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 [7].

When dealing with compression there are two general types, lossy and lossless. In this project I used only lossy compression for my images. While there are a handful of differences between the two, the main distinction is lossy compression loses data that cannot be recovered during compression while lossless can be returned to its exact original state [2]. Lossy compression is the preferred technique when one wishes to reduce the file size and does not care to revert a file back to its original form.

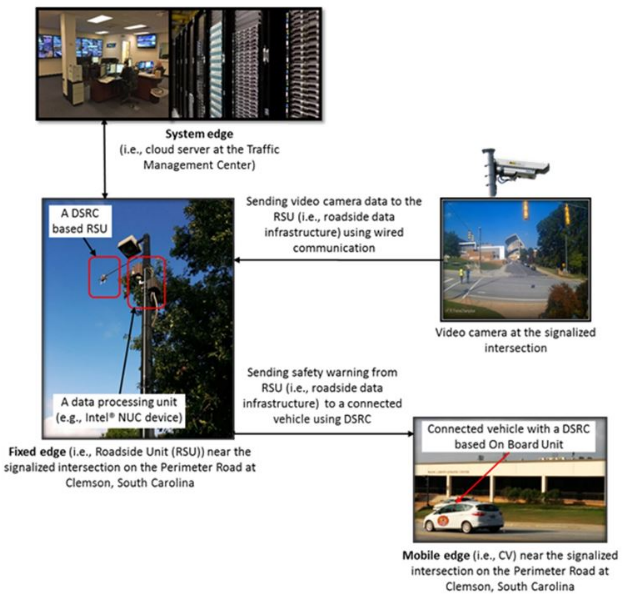
System edge
(i.e., cloud server at the Traffic Management Center)
A DSRC based RSU
Sending video camera data to the RSU (i.e., roadside data infrastructure) using wired communication
Video camera at the signalized intersection
Sending safety warning from RSU (i.e., roadside data infrastructure) to a connected vehicle using DSRC
A data processing unit (e.g., Intel® NUC device)
Connected vehicle with a DSRC based On Board Unit
Fixed edge (i.e., Roadside Unit (RSU)) near the signalized intersection on the Perimeter Road at Clemson, South Carolina
Mobile edge (i.e., CV) near the signalized intersection on the Perimeter Road at Clemson, South Carolina

Figure 1 - Basic overview of the pedestrian-detection module

II. Background

A. Related Work

The work in this paper is served to be an extension of the research conducted by Rahman et al. in their paper *Dynamic*

Error-bounded Lossy Compression (EBLC) to Reduce the Bandwidth Requirement for Real-time Vision-based Pedestrian Safety Applications [6]. 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 ratio (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 2.

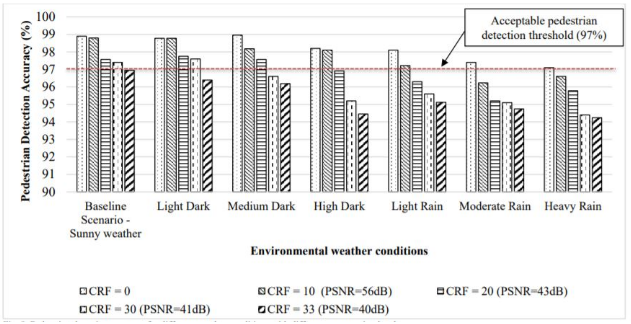

Figure 2 - Pedestrian detection accuracy for different weather conditions with different compression levels

B. *Summary of the Machine Learning Module*

For the purposes of my project, I used the same machine-learning module that Rahman et al. used in their experiments. This served to help keep the experiment controlled and take out any unnecessary variation in the results. The module they used was the YOLOv3 which uses machine-learning to create bounding boxes around features detected in the image and identify which ones are pedestrians [4]. 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. 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 [8].

III. Major Contributions

A. *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 3.

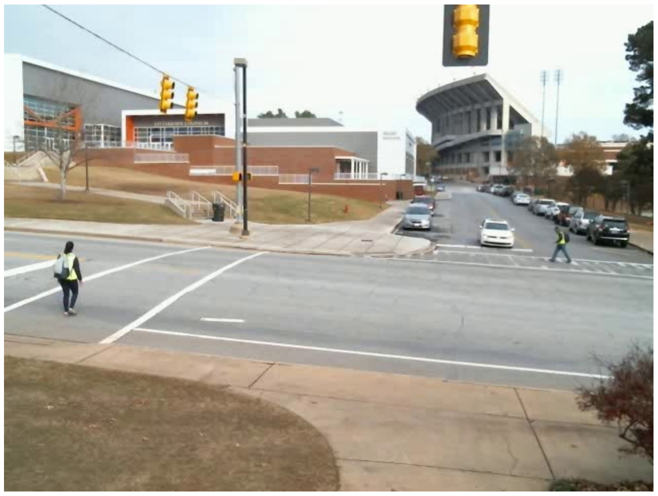

Figure 3 - Example image from experimental data set

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.

B. *Initial Compression Techniques*

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 [9]. 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. When I started this project I was given six image sets each of which compiled with different a CRF value: 0, 10, 20, 30, 33, and 35. The CRF value controls the quality of the image, with 0 being the highest quality [1]. In my initial experiments I simply cropped out irrelevant sections of the JPEG image in the CRF 35 data set and replaced it with the same section in the corresponding CRF 0 image. This in theory should have had a smaller file size than an uncompressed image. However, images with a higher CRF value do not always have smaller file sizes, as CRF is not a guaranteed form of file compression. This, along with the fact the JPEG has a complicated compression algorithm, led me to believe I needed find a practical way to compress these images in software.

C. *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 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.

D. 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. This corresponds to the likelihood of a pedestrian being in that section of the image. My experiments consisted of trying to find the best compression thresholds for each section to run most successfully through our pedestrian-detection module.

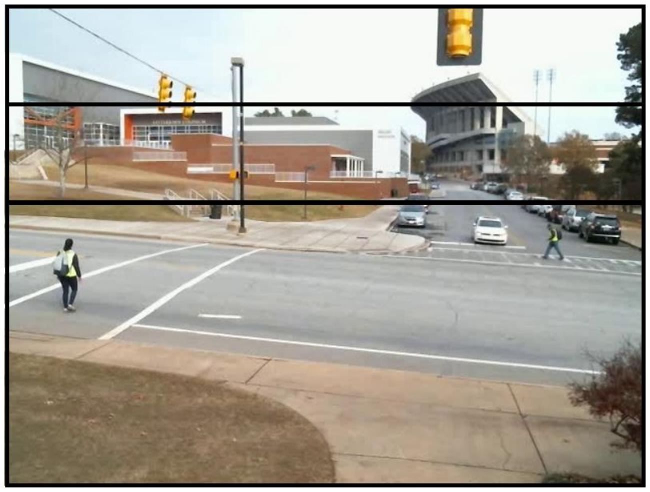

Figure 4 - Breakdown of the various layers of compression

IV. Experimental Evaluation

A. Description of Palmetto and the Setup Process

The YOLOv3 modules takes an immense amount of time to pass a data set of 426 images, so for this project I was granted access to Clemson's supercomputer *Palmetto*. The module uses Tensorflow, a library used for machine-learning applications. Since Palmetto does not have this library installed by default, I had to install the library via an Anaconda environment. I also had to ensure Jupyterhub was compatible with this environment. Once I completed the setup of this environment I could then run the YOLOv3 module on my image sets on Palmetto. The significance in this is Palmetto grants users to run tests on up to 40 CPU's with 370GB of memory and two GPU's. This cut down runtimes for each job to just a few hours.

B. Running the Image Sets Through the YOLOv3 Module

As part of my research, I experimented with using different PSNR values utilizing the SZ compressor of Libpressio to create several uniquely compressed image sets. After creating each data set of 426 images, I was then able to run the set through the pedestrian-detection module and determine how the compression affected detection accuracy. Fortunately, Palmetto allows users to submit PBS scripts, so I could submit my script and exit my SSH session without interrupting the test.

Using my Python script that implemented Libpressio, I created image sets with hybrid compression levels as well as the same compression in all three sections to help create base cases. My data sets and the corresponding accuracy results are detailed in section five of this report. An example of how my script compresses the three sections of an image is depicted below in Figure 5. The code used to create my image sets is also available for reference on my GitHub [3].

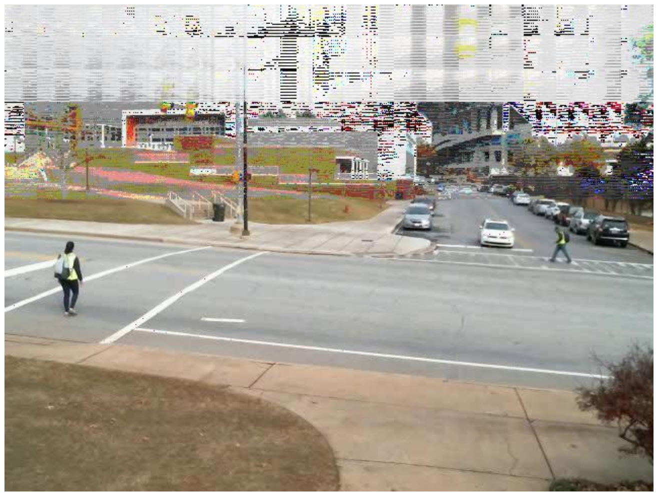

Figure 5 - Compressed image using SZ compressor and PSNR values of 8, 20, and 50, respectively

V. Analysis and Results

A. Data Reduction and Detection Comparison

The results from my experiments are summarized in Table 1 below. The mAP in the table stands for the Mean Average Precision, which is a metric used for measuring accuracy in object detection applications. This is the mean of the average precision (AP) scores for each query in the data set. We can translate this to determine how accurate the machine-learning module is at detecting pedestrians. The PSNR values all correspond to the PSNR used to compress each section of the images with the SZ compressor in Libpressio.

PSNR values (Top-Middle-Bottom)	mAP score
Uncompressed	0.718
Uncompressed	0.6985
Uncompressed	0.737
20-20-20	0.7608
50-50-50	0.699
100-100-100	0.7672
8-20-50	0.7419
5-15-40	0.723
3-13-30	0.7149
3-30-50	0.7198

Table 1 - Results from passing the SZ compressed images through the YOLOv3 module

The table above illustrates that the highest accuracy I was able to obtain was with an image compressed with a PSNR of 100 over all three sections of the image. This is virtually no compression, so that is expected it would be the most accurate. However, the results do not depict any major trends, and in the following subsection I will explain why.

B. Possible Errors

Unfortunately, I did not get the results I expected after conducting my experiments. The team I based my work off of had results up to 97% accurate, while my highest accuracy rating was only 76%. After further analysis, I can pinpoint a few possibilities as to why my results are incomplete and how than can be fixed.

One factor that I ran into with my experiments was using the correct anchor sizes. The three different uncompressed sets in Table 1 were all ran with differing anchors, which one can see has an effect on the final accuracy. Anchors are unique to each data set and are used to further improve accuracy. Since I was creating my own images, it was difficult to pinpoint which anchor set would best be used with my data.

Another possibility is failure to calibrate the non-max suppression algorithm. Like previously stated, the NMS algorithm is used to eliminate the number of false positives the module detects. This algorithm must be calibrated with the proper threshold values to further increase accuracy. Unfortunately I did not discover this until late and did not have time to run multiple tests using different thresholds to perform the calibration.

VI. Conclusion

To summarize my research, I was able to successfully create a conda environment to run the YOLOv3 module on Palmetto, decreasing the necessary time to run the module by a magnificent proportion. I was able to install and apply Libpressio to apply unique compression to separate regions of an image all of which I defined. While my results do not show further improvement in increasing bandwidth, I do believe this project can be expanded upon to find the anticipated results. Once a base case can be identified and the proper accuracy is reproduced, there is no reason to believe that the methods used in this experiment cannot yield further compression while maintaining reasonable accuracy. In addition, this experiment could be taken even further by changing the different compressors used and creating new sections of compression within the images.

Acknowledgment

I would like to thank the entire team consisting of Mizanur Rahman, Mhafuzul Islam, Jon Calhoun, and Mashrur Chowdhury for their prior and ongoing research that this project is based off of. Dr. Calhoun in particular assisted me throughout the entirety of my research and this project would not be possible without him. Additionally, I would like to thank Robert Underwood for his work on Libpressio and assistance in helping me implement the program on my data. Lastly, I would like to credit the Department of Electrical and Computer Engineering at Clemson University for sponsoring this work.

References

[1] CRF Guide (Constant Rate Factor in x264, x265 and Libvpx), 24 Feb. 2017, slhck.info/video/2017/02/24/crf-guide.html.

[2] "Difference between Lossy Compression and Lossless Compression." GeeksforGeeks, 22 May 2019, www.geeksforgeeks.org/difference-between-lossy-compression-and-lossless-compression/.

[3] Heath Gerrald, video_compression, https://github.com/hgerrald/video_compression

[4] J. Redmon, A. Farhadi, and C. Ap, "YOLOv3: An Incremental Improvement," arXiv preprint arXiv:1804.02767, 2018.

[5] 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.

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

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

[8] 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.

[9] 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/.