SSD Shufflenet V2 Autonomus vehicle object detection on Embedded systems

Corona Alvarez
College of Engineering
Colorado State Univertiy
Fort Collins, Colorado
pcorona@rams.colostate.edu

I. Introduction

Computer vision is a growing sector of the machine learning subsection and industry. One application of computer vision is object detection which can have a varying application and uses. Ranging from captioning videos and images to indoor navigation as well as real time object detection for Autonomous vehicles. The wide number of applications for object detection mean that it can be found in lower power systems as well as higher end systems. For this reason, many autonomous vehicles use different approaches for solving the problem of real time object detection.

Largely the standard for object detection has relied on various Lidar and Radar technologies, however, these technologies are expensive raising the question if this is an applicable technology for autonomous vehicles. Regardless of this some questions can still come to rise whether multiple cars using lidar technology would produce substantial interference with one another. Because of the unknowns with Lidar technology as well as the expenses that come along with it looking into potential other solutions avoiding costs is the path this paper seeks to look down.

Looking to reduce the cost and potential overhead of object detection led me to research object detection models which were supportive of lower power embedded systems. Each of which had their own issues, namely being that performance is hindered by the low computational and memory capabilities embedded systems. For this reason, we are testing on a Raspberry Pi 4B with a Coral edge TPU device. Allowing the Responsiveness of the model to be accelerated and increased

II. Problem

Largely the problem that we are looking to overcome is the capabilities of a low power low memory device. The latency between scanning for an object and finding said object is the key factor which we are looking to improve. Object detection for autonomous vehicles must be real time. Embedded systems typically do not have the power to perform real time object detection for autonomous vehicles.

The solution that we are attempting find is creating a model which can be used on low power embedded systems for real time object detection maintaining a reasonable accuracy. Looking as to what real time object detection is, we define real time object detection as being able to process a minimum of 30 frames per second. One of the struggles which will come to fruition is maintaining or achieving this level of performance on a small embedded system. The severity of this issue came to light when studying the models which many mobile devices use. Typically, the performance of the model is greatly bottlenecked by the need for a GPU or TPU device. When one of these devices is present the models are capable of inference at a significantly faster speeds, in some cases processing 43x the images at the same time.

We will be looking at training a Raspberry Pi 4b with 8 GB of ram using a Coral USB Edge TPU Device, using the Udacity Self driving car dataset pertaining of 15,000 images with labels for cars pedestrians and streetlights. We are expecting to see similar results to that which can be found on the Coral USB for SSD using mobile net v2 which is around 14 Frames per second. However, It can be noted that because of use of different models or datasets we can see potential increases or decreases in performance

III. Approach

There are many approaches and models which can be used for Real time object detection each of which provides its own respective benefits. However, there are two strict guidelines which I used in order to determine which model would be picked for training.

1. Models should minimize the amount of space and memory required to run inference.
2. Inference speed is critical over increasing the accuracy of the model within a reasonable range.

We look to adhere to these two guidelines for deciding a model as memory on an embedded system is at a premium, as well as inference speed being a critical portion of Real time object detection. With that being said inference speed does not reign king in relation to inference accuracy, we are willing to decrease the accuracy of the model (a range of 5%) in order to increase the inference speed of the model by an acceptable amount (increasing by a range of 5%). This is due to the inherent nature that inference speed is critical to the performance of a real time object detection model. A model which only process 1 frame per second at 80% accuracy does not provide as much utility in a real world application as a model which process 8 frames per second at 76% accuracy.

For these reasons through looking at many models which have been designed for mobile applications we sorted out those which meet our criteria of being viable on a low power embedded device, as well as ranked them based on inference speed on datasets. Primarily two models came to light for being tested Mobile net v3 as well as ShufflenetV2. Two models came to mind particularly for one reason. Originally Shuffle net was the model which caught my eye due to the faster inference compared to Mobilenetv2. However, authors of Shuffle net state that depth wise convolutions are difficult to efficiently implement on low power mobile devices. For this reason, I looked into the updated models of both MobileNetv3 as well as ShuffleNetv2 both of which use depth wise convolutional layers. ShuffleNetV2 has been tested against MobilenetV2 which It maintains a higher accuracy as well as a faster inference speed. However, they do not compare the model to the newer more accurate MobileNetv3. For this reason, we are looking to test on both models to determine which is overall better for inference accuracy and speed.

The models which we have chosen are bases for the segmentation models which we apply to for object detection. There are few models which are used for object segmentation; one in particular stood out SSD. SSD or Single Shot Detection performed inference and detection at better rates when compared to other models such as YOLO. SSD also provides a framework for it to be built on top of an already pretrained model allowing us to interface with both MobileNetv3 and ShuffleNetV2.

After having trained the model we will then be looking at compressing the model to fit on an embedded system. This will be taken through a few different steps.

1. Quantization allowing us to compress the model further while potentially being able to squeeze out more performance.
2. Clustering the weights which reduces the amount of unique weight values in the model reducing inference time.
3. Pruning the model removing unnecessary weights in the model which decreases the overall size of the model

I will be looking to implement these 3 techniques together with potentially using Huffman encoding in order to increase the inference speed as well as managing to fit the model on a low power mobile device. Another step which can be taken in order to increase the speed and accuracy of the model is to look into potential hardware and software accelerators for these specific models which I have decided upon. This can be potentially done through looking into MAESTRO as well as DEEPX.

Particularly when creating the model for vehicle detection using SSD object detection there was a few issues that I ran into when making my model. Creating the model was initially based on using a Keras port of the work done by W, Liu intended to be used to build a SSD model on top of an existing model which has already been trained or use the same weights from the paper. However, in the implementation of this model and port was not compatible with the current versions of keras and TensorFlow. Much of the code for the Anchor boxes for annotations of the image were using old formatting for accessing certain methods resulting in the code not running. Much time was spent trying to debug this code and find solutions online as to the errors that would come up in using the code.

Eventually I decided to use the old TensorFlow and keras libraries which supported the specific Keras port of the SSD framework. However, these versions of TensorFlow and keras were no longer available for download. Further complicating my creation of the models.

IV. Results

Ultimately the solution for creating a model that used SSD was through the Object detection model zoo which TensorFlow supported natively. In this there were few models to chose from, the SSD Mobilenetv2 300x300 was chosen for the model to remain as close as possible to initial model design choices. Though this proved to be an issue as one of the major guidelines for my decisions was that accuracy must be over an acceptable accuracy when detecting vehicles otherwise a new model must be chosen. The Mobilenetv2 300x300 model proved to have inferior accuracy not being able to reach an accuracy over 60% in my tests. Because of this I decided to change me model to the SSD Resnet v1 640x640.

From this Point following the tutorial from the Tensorflow Object Detection zoo the model was trained twice separately on the Udacity self driving dataset. The first time that the model was trained, using the pretrained weights from the MS COCO 2017 dataset, only 5 images were used for fine tuning. On the second time training the model 78 images were used for finetuning the model prior to testing on the same dataset. Fine tuning the model took about 3 hours in both passthrough regardless of the training dataset.

Once the model was trained it was tested on the same model twice. Once visualizing whenever the accuracy of the model reached above 60% on its guesses and then another with guesses over 80% being annotated on in the test images. This was done to see how having a tighter error tolerance would affect the amount of objects that it would detect in any case.

Ultimately I would say that from the results from training on the two sizes of datasets proves to be about the same regardless of what data was used, this may be because the original 5 images used for training were reused for training in the 78 images. However from watching the test images which are annotated I can say there are a few things that do differ. It seems that the model which was trained on 78 images does better in the testing case in detecting cars while they are turning and are far away however, the model which was trained on solely 5 images seems to do better on detecting cars in the same side of the road and which are closer.

It is also interesting to see how changing the tolerance of what is plotted or treated as an object changes the amount of annotations done in the test images. Its clear to see that changing the boundaries and tolerance for accuracy means more is annotated showing that the model is detecting the objects however struggles in accurately determining if the object Is a car or not.

Another point of interest is to see where each model does and doesn't detect vehicles. In some cases where the model detects the car with good accuracy on one training set the other model trained with different data couldn't detect the object but would detect vehicles the different model couldn't detect.

V. Conclusions

There is still much work which can be done for this model. Firstly there is the option to add more classes for detection. Notable classes which could be implemented which would be important for autonomous driving are, pedestrians, bicycles, Light post, stop signs, speed signs. In the same scope because time did not permit quantizing the model aswell as applying other techniques in order to run the model on an embedded system is a strong step to take in the future as well.

References

[1] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for MobileNetV3," *arXiv.org*, 20-Nov-2019. [Online]. Available: https://arxiv.org/abs/1905.02244. [Accessed: 14-Nov-2021].

[2] D. Li, X. Wang, and D. Kong, "DeepRebirth: Accelerating deep neural network execution on mobile devices," *arXiv.org*, 10-Jan-2018. [Online]. Available: https://arxiv.org/abs/1708.04728. [Accessed: 14-Nov-2021].

[3] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," *arXiv.org*, 21-Mar-2019. [Online]. Available: https://arxiv.org/abs/1801.04381. [Accessed: 14-Nov-2021].

[4] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN Architecture Design," *arXiv.org*, 30-Jul-2018. [Online]. Available: https://arxiv.org/abs/1807.11164. [Accessed: 14-Nov-2021].

[5] N. D. Lane et al., "DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices," 2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN), 2016, pp. 1-12, doi: 10.1109/IPSN.2016.7460664.

[6] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," *arXiv.org*, 29-Dec-2016. [Online]. Available: https://arxiv.org/abs/1512.02325v5. [Accessed: 14-Nov-2021].

[7] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," *arXiv.org*, 07-Dec-2017. [Online]. Available: https://arxiv.org/abs/1707.01083. [Accessed: 14-Nov-2021].

[8] Tensorflow. (2020,July 11). *Models/eager_few_shot_od_training_tf2_colab.ipynb at master · Tensorflow/models*. GitHub. Retrieved December 16, 2021, from https://github.com/tensorflow/models/blob/master/research/object_detection/colab_tutorials/eager_few_shot_od_training_tf2_colab.ipynb

[9]