Distributed Edge Cloud R-CNN for Real Time Object Detection

Joshua Herrera, Mevlut A Demir, Parsa Yousefi, John J Prevost, and Paul Rad
Department of Electrical and Computer Engineering
The University of Texas at San Antonio
One UTSA Circle, San Antonio, TX 78249
Email: joshua.herrera@utsa.edu, mevlut.demir@utsa.edu, parsa.yousefi@utsa.edu, jeff.prevost@utsa.edu, paul.rad@utsa.edu

***Abstract*—Cloud computing infrastructures have become the de-facto platform for data driven machine learning applications. However, these centralized models of computing are unqualified for dispersed high-volume real-time edge data intensive applications such as real time object detection, where video streams may be captured at multiple geographical locations. While many recent advancements in object detection have been made using Convolutional Neural Networks, these performance improvements only focus on a single, contiguous object detection model. In this paper, we propose a distributed Edge-Cloud R-CNN pipeline. By splitting the object detection pipeline into components and dynamically distributing these components in the cloud, we can achieve optimal performance to enable real time object detection. As a proof of concept, we evaluate the performance of the proposed system on a distributed computing platform including cloud servers and edge-embedded devices for real-time object detection on live video streams.**

***Index Terms*—Machine learning, Object detection, CNN, R-CNN, Region proposal, Edge Computing, Distributed computing.**

I. Introduction & Motivation

The current evolution of object detection using Convolutional Neural Networks (CNN) has progressed down a path of combination-of-operations. This is to say, current models attempt to reduce the number of distinct operations by sharing and utilizing computations across multiple stages of the object detection pipeline.

However, without a dedicated Graphics Processing Unit (GPU) or other hardware accelerator, the performance benefits seen while deploying these models can not be realized. As a solution to this issue, this paper proposes a distributed model, where components from existing object detection pipelines are containerized and deployed on various devices in the cloud to leverage all available compute power. This distributed computing approach to an object detection pipeline is the first the authors are aware of in literature.

This distributed pipeline would allow for reasonable performance from an object detection model without the need for expensive, dedicated GPU servers. This is useful in scenarios where bandwidth and/or power is limited, as less data needs to be sent to the cloud, and computation can be distributed onto devices as portable as a laptop CPU and a power-efficient GPU-enabled edge device.

The distributed pipeline in this paper will be based on the region-proposal algorithm (RPA) with CNN features model (R-CNN). As a comparison to the contemporary R-CNN object detection pipeline, a standard (undistributed) Faster R-CNN model was trained to serve as a benchmark for future work. Details of this model will be discussed in Section V.

The rest of the paper progresses as follows. First, section II provides a literature review of past and current R-CNN pipelines. Section III uses the concepts discussed in section II to define a new, distributed R-CNN pipeline. Section IV discusses the progress made specifically with the RPA component of the distributed R-CNN pipeline. Section V describes the un-distributed model that is being used as a benchmark for future work. Section VI lays out some future work for the implementation and testing of the distributed R-CNN model, and finally, section VII concludes this paper.

II. R-CNN Based Object Detection

Disjointed R-CNN. The first pipeline to successfully apply region-proposal with CNN features was the R-CNN object detection pipeline proposed by Girshik et al. in [1]. It is termed disjointed in this paper because it takes distinct, previously existing components and combines them into a unified object detection pipeline. It's architecture can be found in Fig. 1.

The first stage consists of a RPA that takes an image as input and returns a number of bounding boxes predicted to contain an object of interest. The original R-CNN model used Selective Search [1] to generate approximately 2000 boxes per image.

The second stage is a Convolutional Neural Network [12] [15]. Bounding boxes from the RPA are used to crop the input image. These crops are then warped to a fixed, square resolution in order to accommodate the CNN. Feature maps are computed for each warped region. Note that this is happening up to 2000x per image.

Finally, an SVM (Support Vector Machine) is used to classify each region based on the features extracted by the CNN.

This method for object detection was shown to have significant accuracy improvements on the canonical PASCAL VOC [7] dataset compared to previous methods using SIFT [8] and HOG [9] features. Additionally, when compared to the OverFeat [10] model, a model which did utilize CNN features, but implemented a sliding window detector [13] [14] instead

978-1-5323-7791-4/2018/ ©2018 TSI Press, Inc

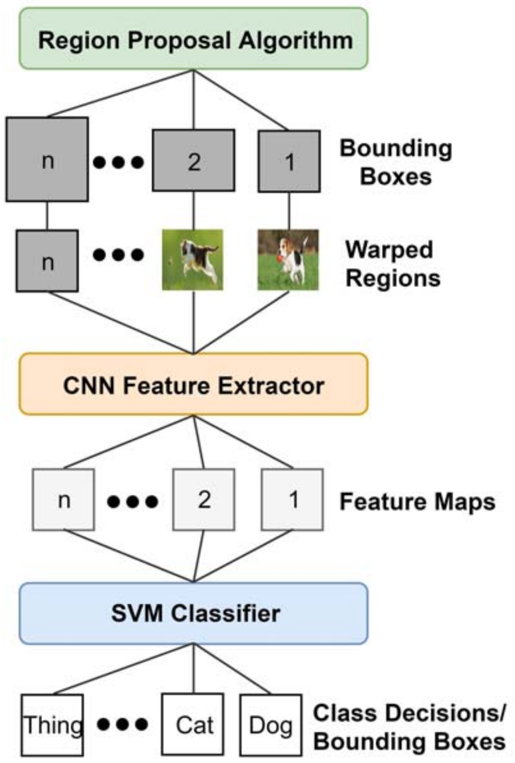

Fig. 1: R-CNN - Girshik et al. [1]

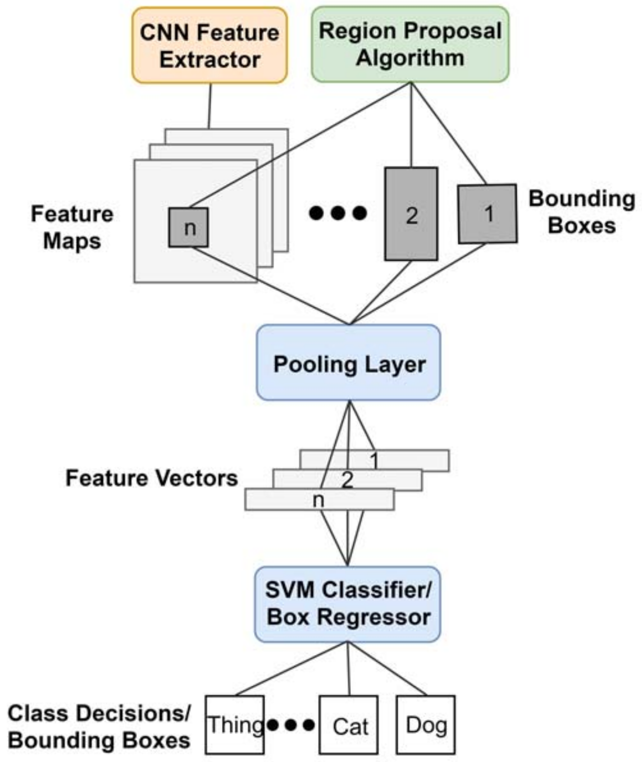

Fig. 2: Fast R-CNN - Girshik [2]

of a RPA, R-CNN outperformed by a significant margin on the ILSVCR [11] dataset.

Less Convolutions. The next evolution in R-CNNs were models like Fast R-CNN [2] and SPPnet [4] which, instead of operating on each region proposal box individually, compute a single convolutional feature map for an entire image (Fig. 2). This dramatically reduces the number of operations performed by the CNN for a single image, allowing it room to grow in depth (more layers) and, consequently, achieve higher accuracy.

After predicting bounding boxes and computing the image's feature map, the second stage maps bounding boxes to the image's feature map. The nomenclature in [2] refers to these as Regions of Interest (RoI). Each of these regions are pooled to form 1D feature vectors.

Finally, these feature vectors are fed to an SVM and bounding-box regressors [16] for classification and localization refinement respectively.

Fast R-CNN showed a 213x improvement in performance over the original R-CNN model at run-time, and a 9x improvement in training speed.

A Moving/Sharing of Operations. Faster R-CNN [3], replaces the arbitrary RPA present in R-CNN and Fast R-CNN with a unified Region Proposal Network (RPN). This network generates bounding boxes by performing convolution over the feature maps produced by the CNN feature extractor of Fast R-CNN. This is particularly impactful on performance because using convolutions for the Region Proposal Network allows it to be moved from the CPU to the GPU.

Additional performance benefits are gained from sharing convolution operations across the RPN and CNN stages of the model.

Further developments in object detection have seen a departure from the dedicated region proposal method. Models such as SSD [6] and YOLO [5] forgo a dedicated region proposal algorithm to improve computational performance, but this often sacrifices accuracy for detection speed.

III. Distributed R-CNN

This paper's contribution to literature is the use of distributed computing to deploy a R-CNN object detection pipeline as discussed above. As noted in literature [5] [6], the need for a RPA in the R-CNN pipeline creates a bottleneck at the CPU. Approaches to mitigating this bottleneck have aimed to refine a single, contiguous model by reducing the total number of operations across the pipeline, and by moving RPA operations to the GPU. Other pipelines take unique approaches such as a cascade of more efficient, lighter networks, or a color based detection metric, to improve accuracy while maintaining efficiency [36] [37] [38].

The approach in this paper splits the R-CNN pipeline into discrete components. These components can be containerized and deployed [33] [34] [35] in the cloud to take advantage of the compute resources of multiple devices. Distributing the

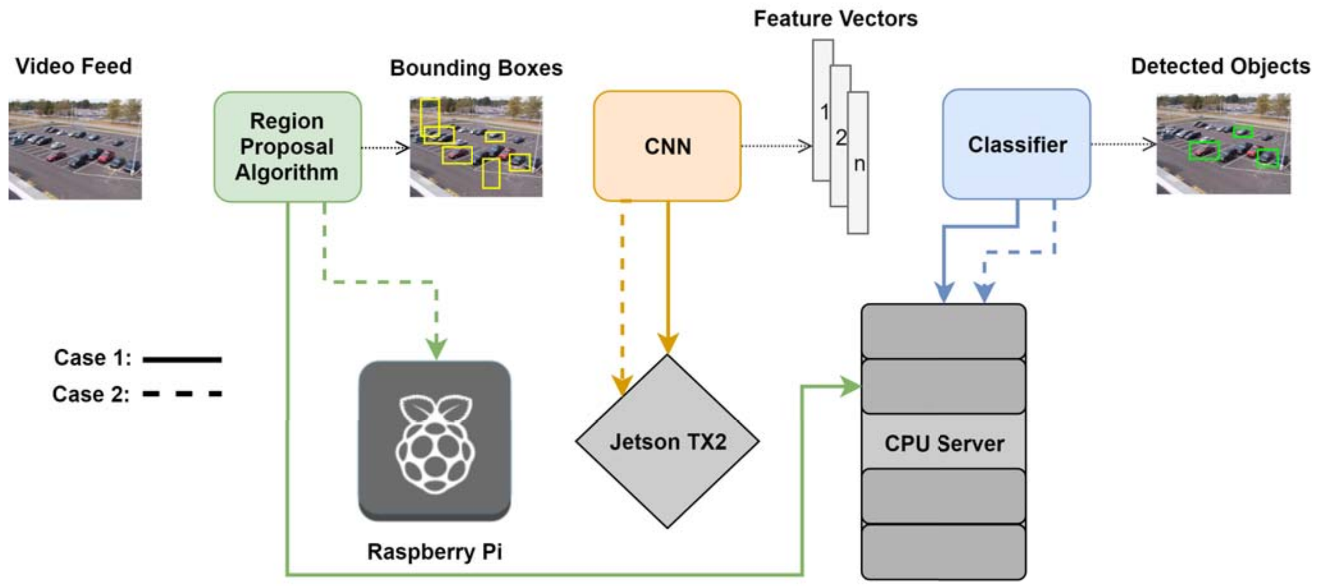

Fig. 3: Distributed R-CNN

computational requirements for the pipeline is expected to improve performance, especially in cases where limited hardware infrastructure prohibits the use of a dedicated, centralized GPU infrastructure. This would serve to improve detection speed and latency while maintaining the accuracy benefits of R-CNN.

For the purposes of this distributed pipeline, 3 primary components are considered; a **region proposal** algorithm, a **CNN feature extractor**, and then any **additional layers** used for final classification and bounding box regression. As a proof of concept, the first implementation of the distributed pipeline will be modeled after the simplest R-CNN model, the original R-CNN.

Fig. 3 illustrates several possible distribution cases for the R-CNN model. Each case has a set of potential advantages and disadvantages. Case 1 takes advantage of the high CPU compute power provided by a CPU bound cloud to accelerate the region proposal algorithm, while simultaneously taking advantage of the Nvidia Jetson TX2's GPU for computing CNN feature vectors. Case 2 is the most distributed case, but the Raspberry Pi's limited compute capabilities may pose a bottleneck to the system.

A. Region Proposal Algorithm

Several region proposal algorithm were considered in the development of this distributed R-CNN model. In particular, selective search, objectness, and EdgeBoxes were evaluated using Hosang et al.'s paper on current detection proposal methods [22].

Selective Search [17] [18], as used in the original R-CNN [1] model, uses the concept of superpixels to segment an image at different levels of granularity. Superpixels are regions of an image where the original pixels have been merged to express the presence of an object. No model is trained to achieve this. Instead, specifically designed features and score functions define which pixels get merged into superpixels.

Objectness [19] [20] refers to how likely it is that a bounding box contains an object. The algorithm to predict objectness uses a variety of indicators such as edge density, saliency and color contrast to generate bounding boxes.

EdgeBoxes [21] is a relatively new RPA that makes bounding box predictions based on an edge map generated by Structured Forests [23] [24]. Structured Forests uses a trained model to generate edges from an input image, which is then fed to EdgeBoxes for bounding box prediction. These predictions are made based on how many contours are contained wholly within a region of the edge map.

Comparison Hosang et al. judged region proposal algorithms on three criteria: repeatability, recall, and detection. In addition, execution time of the algorithm was recorded.

Repeatability is defined as a RPA's propensity to re-localize similar image content within a variety of different images. That is, if an algorithm predicts the location of something within an image, that same prediction will be repeatable if fed a modified image. This was tested in [22] by adjusting an image's brightness, saturation, crop, etc. and checking for re-detection of objects. The 3 algorithms under consideration scored as follows, from best to worst; EdgeBoxes, selective search, then objectness.

Recall refers to how many ground truth detection annotations the RPA "hits" with its bounding boxes. The value of recall was tested by Hosang et al. at several Intersection over Union (IoU) values and with differing numbers of proposals. Objectness consistently scored the worst compared to EdgeBoxes and selective search. When considering a low IoU, EdgeBoxes consistently scores the best of all methods. However, as IoU values increase to around 0.8, selective search scores better.

Finally, the detection metric measures how well a region proposal algorithm works in practice with a detection pipeline. From best to worst, the 3 algorithms scored as follows; EdgeBoxes, selective search, then objectness.

Conclusion. Edgeboxes and selective search are the most attractive RPAs because of their high recall. However, EdgeBoxes is up to two orders of magnitude faster than selective search, so it was selected for this pipeline. Although its predicted bounding boxes may have poor IoU with objects in a scene, well designed final layers can refine the bounding boxes to more accurately encompass objects.

B. CNN Feature Extractor

To provide a good basis for comparison, the CNN used for the distributed pipeline will be the same as that used in the original R-CNN model [12]. Further work for creating a faster distributed pipeline will adapt the inception_v2 architecture to a Fast R-CNN architecture. The CNN's layer architecture for the current implementation can be found in Table I.

type	**kernel shape/stride**	**input shape**
conv	11x11/4	225x225x3
pool	3x3/2	54x54x96
conv	5x5/1	26x26x96
pool	3x3/2	22x22x256
conv	3x3/1	10x10x256
conv	3x3/1	8x8x384
conv	3x3/1	6x6x384
dense	2048	4x4x256
linear	logits	1x2048
softmax	classifier	1x80

TABLE I: CNN Layer Architecture

C. Additional Layers

The final dense, logits, and softmax layers will be separated from the CNN classifier discussed above. They will receive the feature vectors from the CNN and predict classes for each bounding box.

IV. EdgeBoxes

A. Implementation

As a first step in implementing the distributed R-CNN model, EdgeBoxes was used for region proposal on a video of a parking lot.

The EdgeBoxes RPA consists of two steps. First, an edge map and orientation map are generated for an input image using Structured Forests. This can be seen in Fig. 4, where the upper left image is the original video feed and the upper right image is the edge map generated by Structured Forests.

The second stage feeds the edge map and orientation map into the EdgeBoxes algorithm for bounding box proposal. This can be seen in the bottom image of Fig. 4. It should be noted that, for visualization purposes, only 100 boxes were proposed in Fig. 4, whereas the actual number of boxes proposed will be an order of magnitude greater.

Implementation details for hardware and software parameters are as follows:

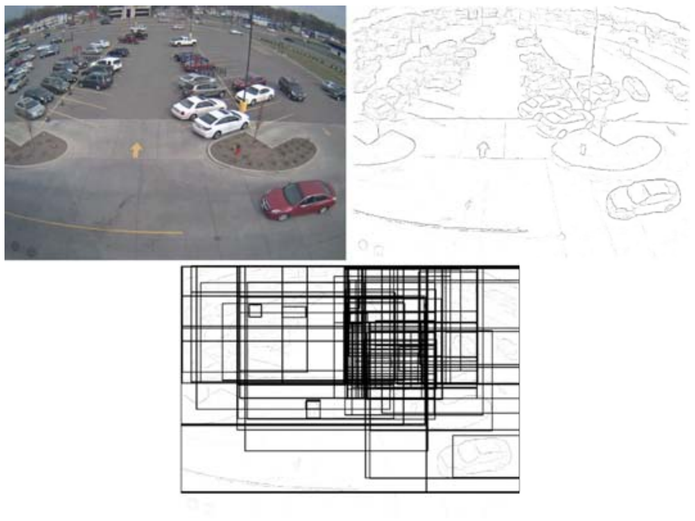

Fig. 4: Edge Map Generation and Bounding Box Proposals

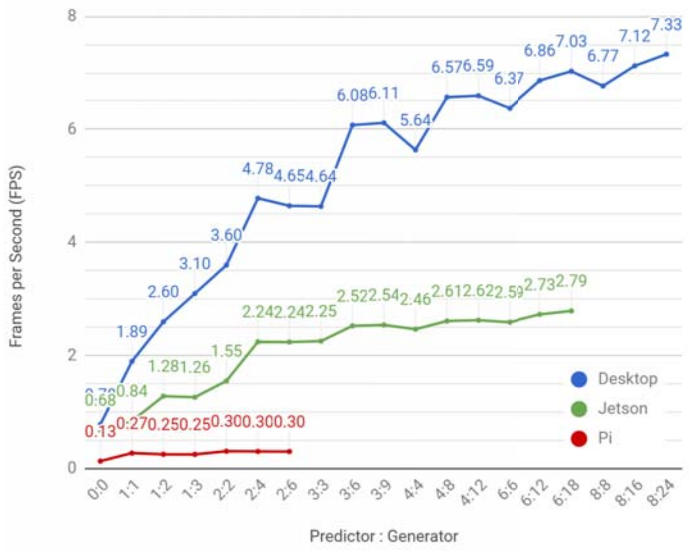

Fig. 5: EdgeBoxes Performance on Different Platforms

- Desktop: workstation, quad-core Intel i7-4770 CPU
- RPi: Raspberry Pi 3 Model B, quad-core ARM CPU
- Jetson: NVIDIA Jetson TX2, dual-core Denver CPU + quad-core ARM
- Structured Forests: OpenCV [31] implementation and model from [32].
- EdgeBoxes: OpenCV implementation. Parameters: alpha 0.65, beta 0.75, minscore 0.03, + OpenCV defaults.

The primary efforts for this implementation were focused on improving the computational performance of the region proposal algorithm.

B. Results

Initially, a consecutive execution approach was taken, where Structured Forests would first generate an edge map and orientation map, then EdgeBoxes would process these maps for bounding box proposal. However, this method was unacceptably slow. It was observed that EdgeBoxes consistently ”waited” on Structured Forests to generate the necessary maps; so, a method using multi-threaded computation was implemented. While EdgeBoxes predicts bounding boxes for the current frame, Structured Forests generates the next frame's maps, serving as a frame buffer.

Performance was tested by spawning different ratios of Structured Forest threads to EdgeBoxes threads. Since Structured Forest threads serve to buffer for EdgeBoxes threads, a number of EdgeBoxes threads were spawned, and then either 1, 2, or 3 Structured Forests threads were spawned for each EdgeBoxes thread. ie. 1 Edgeboxes : 2 Structured Forests threads.

Performance increases are shown in Fig. 5, measured in frames per second (FPS). The horizontal axis is the number of EdgeBoxes threads and Structured Forests threads spawned (ie 2:6 - 2 EdgeBoxes:6 Structured Forests), where 0:0 represents consecutive execution. A steady increase in FPS can be seen from consecutive execution to multiple threads on the Desktop and Jetson TX2 platforms. With an execution FPS of 0.78 at 1 thread (0:0), a value of 6.08 FPS at 9 threads (3:6) represents a 85% increase in performance per thread on the desktop. The Jetson TX2 performance gains, while notable, provide a poor analogy to a CPU cloud due to the Jetson's mix of faster and slower cores.

The Raspberry Pi sees an improvement in performance from consecutive execution to multi-threading, but these improvements taper off quite quickly due to the Pi's insufficient CPU overhead. A slight disparity can be seen between the bare-metal performance of the desktop vs a containerized deployment, but this disparity is reduced as more threads are spawned. Further investigation is required to explain this performance difference.

Regardless, results from the desktop implementation suggest that region proposal using a multi-threaded EdgeBoxes would scale well with the high core count of a CPU-bound cloud. Since the efforts described here are primarily to improve computational performance, further testing with the completed distributed pipeline is needed to evaluate accuracy.

V. CNN Feature Extractor

In a manner similar to that used by Google's pre-trained feature extractors (Section VI), the CNN will first be trained broadly on a publicly available dataset, Microsoft's COCO [26]. Then, the model's weights will be saved and re-trained to identify cars in a parking lot using a fine-tuning dataset. This is to ensure that the model is proficient at recognizing cars from an elevated perspective, such as that of a drone above a parking lot, for the use case described in Section VI.

Because COCO is a dataset designed to test object detection algorithms, training the CNN as a classifier requires preprocessing of each image into separate objects. This is accomplished by cropping and warping each object in an image and feeding them individually to the CNN. This is identical to how the completed pipeline will utilize a RPA's predicted bounding boxes.

A. Results

The CNN was implemented using Google's Tensorflow library [30]. Initial training of the CNN has achieved 67% accuracy across all 80 classes of the COCO dataset. This level of accuracy was achieved after 60 hours of training on a GTX 1080.

Considering recall values at different IoUs for EdgeBoxes from [22], we expect accuracy results on the COCO dataset for the distributed pipeline to approach those in Table II. # Candidates refers to the number of boxes proposed by EdgeBoxes at run time.

Execution time on a GTX 1080 for 1000 candidate bounding boxes was 1.12 seconds and 29.2 seconds on the Jetson TX2. Further improvements to this execution time will be made with the adoption of a Fast-CNN architecture.

# Candidates	mAP@0.5 IoU	mAP@0.7 IoU
1000	51.0	43.2
10000	57.0	51.0

TABLE II: Expected Accuracy Results

VI. Faster R-CNN

As a comparison to the standard R-CNN object detection pipeline, a Faster R-CNN model using the inception_v2 [29] CNN architecture will be deployed on a CPU bound server. This model will not be split into components, and so will run its RPA, CNN, and other operations on a single device. This will allow for quantitative benchmark metrics to be compared to the novel, distributed pipeline's performance.

A. Training

Due to the fairly high cost of training, with regards to both hardware and time requirements, a method using Google's pre-trained feature extractors is explored [25]. The model used as a benchmark in this paper is the faster rcnn inception_v2 model pre-trained on the coco dataset [26].

To ensure the model would be proficient at recognizing cars from an elevated perspective (such as that of a camera over a parking lot) for the test case (section VI), the model was fine-tuned with a small set of images pulled from two datasets.

The first dataset was provided by [27] and was used in the testing of their system. It consists of images from several different parking areas and from multiple perspectives. The second dataset was provided by [28] and consists of several thousand images from three different cameras perspectives at three different parking lots during various times throughout the day. After cleaning up annotation values, the final dataset used for fine-tune training and evaluation consists of 155 images, containing 5837 cars.

The faster r-cnn inception_v2 model was fine-tune trained for 6.5 hours on the dataset described above, where the dataset was split 4:1, training:evaluation (124 training. 31 evaluation). Accuracy plateaued at around 80% on the evaluation dataset.

VII. Further Work

While progress has been made in implementing individual components of the distributed pipeline, more work is needed to further improve and evaluate the accuracy of the combined pipeline. In particular, the IoU of Edgeboxes on the COCO dataset needs to be improved with tweaks to its detection parameters. The Accuracy of the CNN feature extractor on the COCO dataset in deployment also needs to be improved. Additionally, further performance optimizations by adopting a Fast-CNN architecture will be made for the CNN.

As an ”in the wild” test case, a camera based infrastructure using R-CNN models proposed in this paper will be used to track cars in a parking lot. This data will be used to monitor parking lot occupancy conditions in real-time. Both the distributed and undistributed R-CNN models will be deployed and tested on a series of metrics including latency and accuracy.

VIII. Conclusion

This paper presents the components for a distributed R-CNN pipeline. This pipeline is based on the original R-CNN architecture [1], with future work to adopt a Fast R-CNN architecture [2]. EdgeBoxes [21] was chosen for the Region Proposal Algorithm component of the pipeline. Performance increases were achieved through multithreading the different stages [23] [24] of EdgeBoxes. A CNN based on the original R-CNN's model was designed and trained as the feature extractor and classifier of the model. Finally, as a benchmark for future work, an undistributed Faster R-CNN model was trained on the parking lot test discussed in section VI.

References

[1] Girshick Ross, et al. ”Rich feature hierarchies for accurate object detection and semantic segmentation.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2014.

[2] Girshick Ross. ”Fast r-cnn.” arXiv preprint arXiv:1504.08083 (2015).

[3] Ren Shaoqing, et al. ”Faster r-cnn: Towards real-time object detection with region proposal networks.” Advances in neural information processing systems. 2015.

[4] He Kaiming, et al. ”Spatial pyramid pooling in deep convolutional networks for visual recognition.” european conference on computer vision. Springer, Cham, 2014.

[5] Redmon, Joseph, et al. ”You only look once: Unified, real-time object detection.” Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[6] Liu Wei, et al. ”Ssd: Single shot multibox detector.” European conference on computer vision. Springer, Cham, 2016.

[7] Everingham, Mark, et al. ”The pascal visual object classes (voc) challenge.” International journal of computer vision 88.2 (2010): 303-338.

[8] Lowe, David G. ”Distinctive image features from scale-invariant keypoints.” International journal of computer vision 60.2 (2004): 91-110.

[9] Dalal, Navneet, and Bill Triggs. ”Histograms of oriented gradients for human detection.” Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on. Vol. 1. IEEE, 2005.

[10] Sermanet, Pierre, et al. ”Overfeat: Integrated recognition, localization and detection using convolutional networks.” arXiv preprint arXiv:1312.6229 (2013).

[11] Deng, J., et al. ”Imagenet large scale visual recognition competition.” (ILSVRC2012) (2012).

[12] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. ”Imagenet classification with deep convolutional neural networks.” Advances in neural information processing systems. 2012.

[13] Rowley, Henry A., Shumeet Baluja, and Takeo Kanade. ”Neural network-based face detection.” IEEE Transactions on pattern analysis and machine intelligence 20.1 (1998): 23-38.

[14] Sermanet, Pierre, et al. ”Pedestrian detection with unsupervised multi-stage feature learning.” Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on. IEEE, 2013.

[15] LeCun, Yann, et al. ”Backpropagation applied to handwritten zip code recognition.” Neural computation 1.4 (1989): 541-551.

[16] Felzenszwalb, Pedro F., et al. ”Object detection with discriminatively trained part-based models.” IEEE transactions on pattern analysis and machine intelligence 32.9 (2010): 1627-1645.

[17] Uijlings, Jasper RR, et al. ”Selective search for object recognition.” International journal of computer vision 104.2 (2013): 154-171.

[18] Van de Sande, Koen EA, et al. ”Segmentation as selective search for object recognition.” Computer Vision (ICCV), 2011 IEEE International Conference on. IEEE, 2011.

[19] Alexe, Bogdan, Thomas Deselaers, and Vittorio Ferrari. ”What is an object?.” Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010.

[20] Alexe, Bogdan, Thomas Deselaers, and Vittorio Ferrari. ”Measuring the objectness of image windows.” IEEE transactions on pattern analysis and machine intelligence 34.11 (2012): 2189-2202.

[21] Zitnick, C. Lawrence, and Piotr Dollr. ”Edge boxes: Locating object proposals from edges.” European Conference on Computer Vision. Springer, Cham, 2014.

[22] Hosang, Jan, Rodrigo Benenson, and Bernt Schiele. ”How good are detection proposals, really?.” arXiv preprint arXiv:1406.6962 (2014).

[23] Dollr, Piotr, and C. Lawrence Zitnick. ”Structured forests for fast edge detection.” Computer Vision (ICCV), 2013 IEEE International Conference on. IEEE, 2013.

[24] Dollr, Piotr, and C. Lawrence Zitnick. ”Fast edge detection using structured forests.” IEEE transactions on pattern analysis and machine intelligence 37.8 (2015): 1558-1570.

[25] Huang, Jonathan, et al. ”Speed/accuracy trade-offs for modern convolutional object detectors.” IEEE CVPR. 2017.

[26] Lin, Tsung-Yi, et al. ”Microsoft coco: Common objects in context.” European conference on computer vision. Springer, Cham, 2014.

[27] X. Ling, J. Sheng, O. Baiocchi, X. Liu and M. E. Tolentino, ”Identifying parking spaces & detecting occupancy using vision-based IoT devices,” 2017 Global Internet of Things Summit (GIoTS), Geneva, 2017, pp. 1-6.

[28] Almeida, P., Oliveira, L. S., Silva Jr, E., Britto Jr, A., Koerich, A., PKLot A robust dataset for parking lot classification, Expert Systems with Applications, 42(11):4937-4949, 2015.

[29] Szegedy, Christian, et al. ”Rethinking the inception architecture for computer vision.” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2016.

[30] Dean, J., and R. Monga. ”TensorFlow: Large-scale machine learning on heterogeneous systems.” TensorFlow. org. Google Research. Retrieved 10 (2015).

[31] Bradski, Gary. ”The opencv library (2000).” Dr. Dobbs Journal of Software Tools (2000).

[32] OpenCV, ximgproc: module for extended image processing, github.com/opencv/opencv_extra/blob/master/testdata/cv/ximgproc/model.yml.gz

[33] Benson, James O., John J. Prevost, and Paul Rad. ”Survey of automated software deployment for computational and engineering research.” Systems Conference (SysCon), 2016 Annual IEEE. IEEE, 2016.

[34] Rad, P., Lindberg, V., Prevost, J., Zhang, W., & Jamshidi, M. (2014, August). ZeroVM: secure distributed processing for big data analytics. In World Automation Congress (WAC), 2014 (pp. 1-6). IEEE.

[35] Karim, S., John Prevost, and Paul Rad. ”Efficient real-time mobile computation in the cloud using containers.” (2016): 21-30.

[36] Lwowski, J., Kolar, P., Benavidez, P., Rad, P., Prevost, J. J., & Jamshidi, M. (2017, June). Pedestrian detection system for smart communities using deep Convolutional Neural Networks. In System of Systems Engineering Conference (SoSE), 2017 12th (pp. 1-6). IEEE.

[37] M. Bagheri, M. Madani, R. Sahba, and A. Sahba, ”Real time object detection using a novel adaptive color thresholding method”, International ACM workshop on Ubiquitous meta user interfaces (Ubi-MUI'11), Scottsdale, AZ, November 2011.

[38] A. Sahba, R. Sahba and W. M. Lin, ”Improving IPC in simultaneous multi-threading (SMT) processors by capping IQ utilization according to dispatched memory instructions,” 2014 World Automation Congress (WAC), Waikoloa, HI, 2014, pp. 893-899.