Distributed Edge Cloud R-CNN for Real Time Object Detection

Joshua Herrera, Mevlut A Demir, 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, 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 but these performance improvements only focus on a single contiguous object detection model. In this paper, we propose a distributed Edge-Cloud R-CNN by splitting the model into components and dynamically distributing these components in the cloud for optimal performance for real time object detection. As a proof of concept, we evaluate the performance of the proposed system on a distributed computing platform encompasses cloud servers and edge embedded devices for real-time object detection on 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 has progressed down a path of combination-of-operations. This is to say, current architectures try to reduce the number of distinct operations by sharing and utilizing the same operations across multiple stages of the object detection pipeline.

However, upon inspection of the R-CNN model, it's architecture can be resolved into distinct components. This paper serves as a proposal for the deployment of a distributed model based on the components native to the R-CNN model. Some preliminary results are also presented.

Another motivation for this model is the potential for a GPU-less deployment. This would allow for reasonable performance from an object detection model without the need for expensive, dedicated GPU hardware.

As a comparison to the standard R-CNN object detection model, a Faster R-CNN model using the inception v2 architecture will be deployed undistributedly. This will allow for quantitative benchmark metrics to be compared between the two models.

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

II. R-CNN Based Object Detection

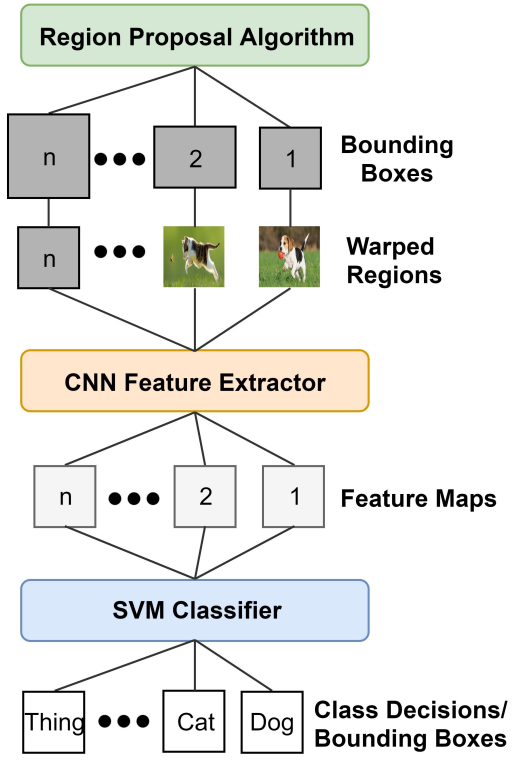

Fig. 1. R-CNN - Girshik et al. [1]

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

The first stage consists of a region proposal algorithm 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 [4] to generate approximately 2000 boxes per image.

The second stage is a Convolutional Neural Network [14, 17]. Bounding boxes from the region proposal algorithm 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. Keep in mind, 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 [8] dataset compared to previous methods using SIFT [9] and HOG [10] features. Additionally, when compared to the OverFeat [11] model, an architecture which did utilize CNN features but implemented a sliding window detector [15, 16] instead of a region proposal algorithm, R-CNN outperformed by a significant margin on the ILSVCR [12, 13] dataset.

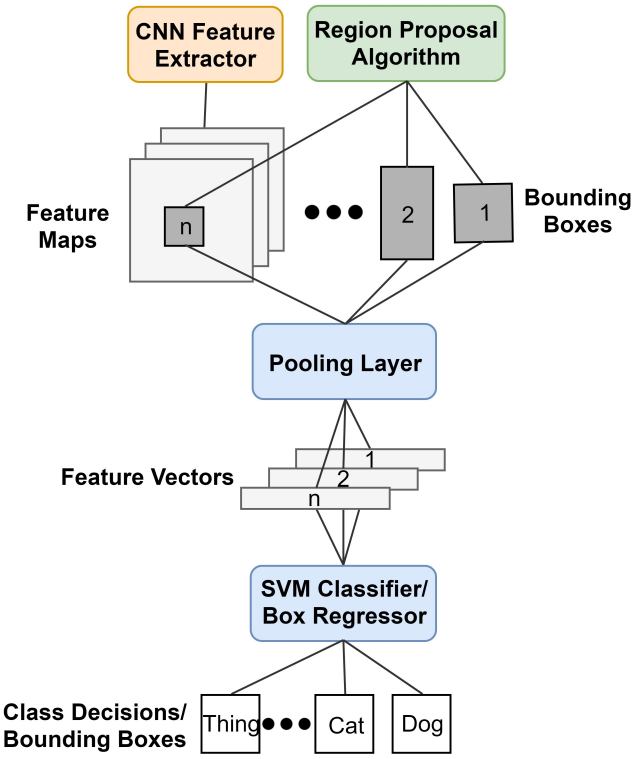

Fig. 2. Fast R-CNN - Girshik [2]

Less Convolutions. The next evolution in R-CNNs were models like Fast R-CNN [2] and SPPnet [5] which, instead of operating on each region proposal box individually, compute a single convolutional feature map for an entire image. 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 [18] 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.

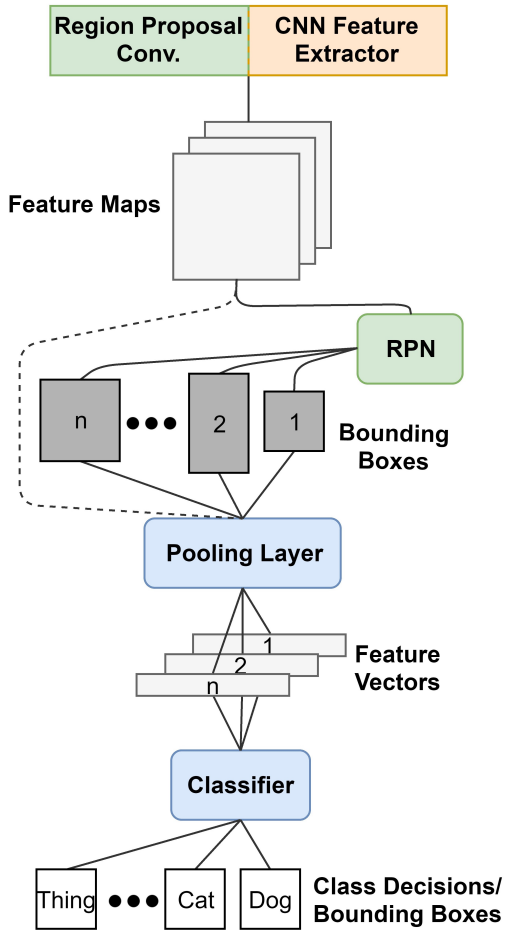

Fig. 3. Fast R-CNN - Ren et al. [3]

A Moving/Sharing of Operations. The final iteration of R-CNN, Faster R-CNN [3], replaces the arbitrary region proposal network 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.

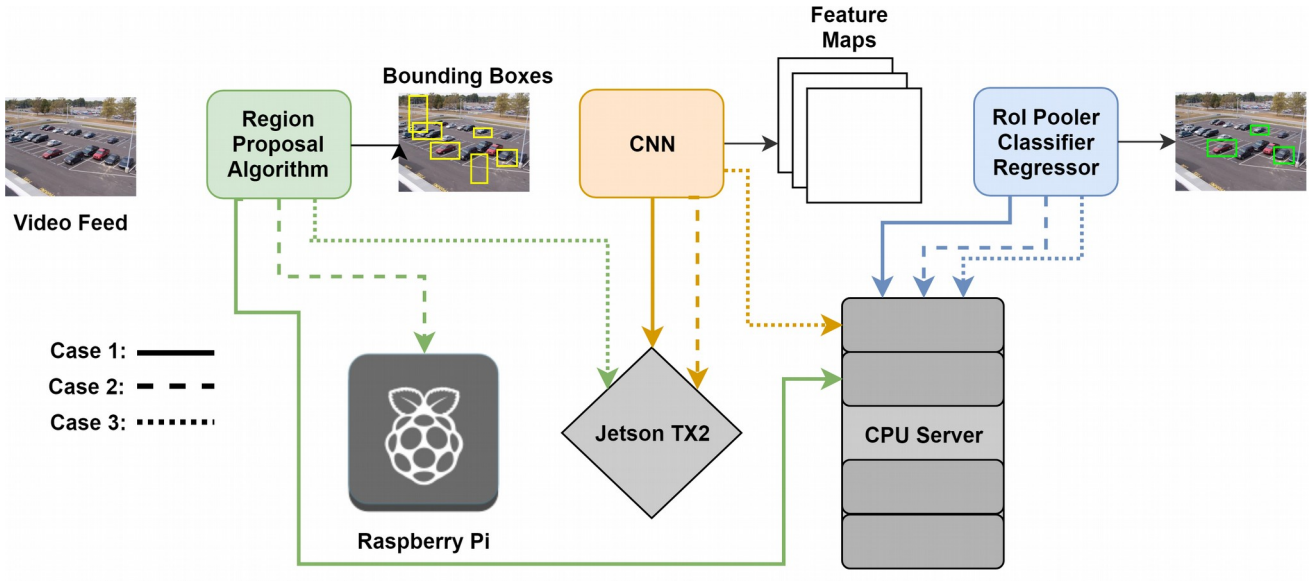

Fig. 4. Distributed R-CNN

Additional performance benefits were 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 [7] and YOLO [6] replace the 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 a dynamically distributed R-CNN model based on Fast R-CNN. As noted in other object detection architectures in literature [6, 7], the need for a region proposal algorithm in the R-CNN model creates a bottleneck for the overall detection speed of the model. Most, if not all, approaches to mitigating this bottleneck have aimed to refine a single, contiguous model by reducing and sharing operations across the architecture. Other architectures take unique approaches such as a cascade of more efficient, lighter networks to improve accuracy while maintaining efficiency [39].

However, the R-CNN architecture can be split into discrete parts, parts that can be containerized and deployed [36, 37, 38] in the cloud to take advantage of the compute resources of multiple devices. Distributing the computational requirements for the model is expected to improve performance, especially on limited hardware platforms such as those without a dedicated GPU. This would serve to improve detection speed while maintaining the accuracy benefits of R-CNN.

For the purposes of this distributed architecture, 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.

Figure 4 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 feature maps. Case 2 is the most distributed case, but the Raspberry Pi's limited compute capabilities may pose a bottleneck to the system. Case 3 attempts to move convolution to the CPU bound server while doing region proposals on the Jetson TX2. Further testing is needed to evaluate the most efficient configuration.

A. Region Proposal

Several region proposal algorithm were considered for the development of this distributed R-CNN model. In particular, selective search, objectness, and EdgeBoxes. Hosang et al.'s evaluation of current detection proposal methods [25] was heavily utilized in deciding what method to use.

Selective Search [20, 21], as used in the original R-CNN [1] model, uses the concept of superpixels to segment an image at different levels of granularity, where superpixels are regions of an image where the original pixels have been merged. No model is trained to achieve this. Instead, specifically designed features and score functions define which pixels get merged into superpixels.

Objectness [22, 23] 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 [24] is a relatively newly proposed region proposal method that makes bounding box predictions based on an edge map generated by Structured Forests [26, 27]. 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 region proposal algorithm'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 different image. This was tested in [25] 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 an object detection algorithm "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 would work with a detection framework. From best to worst, the 3 algorithms scored as follows; EdgeBoxes, selective search, then objectness.

Conclusion. Edgeboxes and selective search are the most attractive region proposal algorithms because of their high recall. The deciding factor for a region proposal algorithm choice comes down to computational performance; EdgeBoxes is up to two orders of magnitude faster than selective search. Although its predicted bounding boxes may have poor IoU with objects in a scene, a well designed RoI pooling layer and box regressors may be able to make up for its shortcomings.

B. CNN Feature Extractor

Since the hardware for deploying this system will be primarily CPU bound, with the exception of the Jetson TX2's small onboard GPU, the CNN feature extractor should be fittingly computationally (un)complex. Following suite after tensorflow's fastest faster r-cnn model in the object detection API [28], the CNN feature extractor for this distributed R-CNN will be modeled after Szegedy et al.'s inception_v2 architecture.

C. Additional Layers

Additional layers such as the RoI pooling layer, classifying layers, and box regressors will be deployed together since they are the less computationally intense portions of the model.

IV. EDGE BOXES

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.

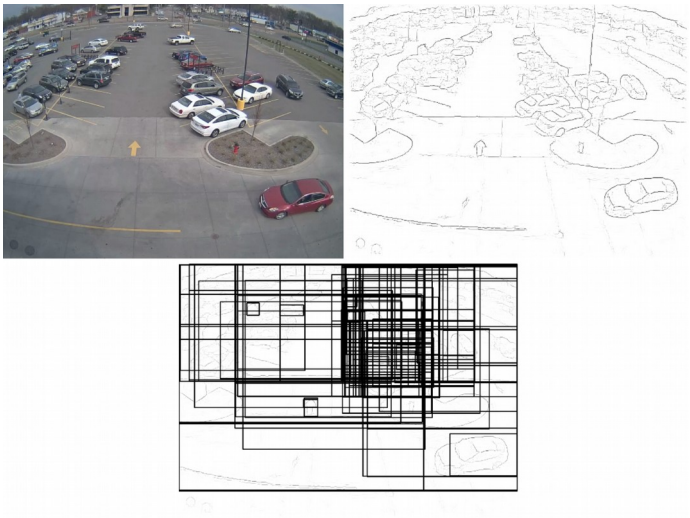

Fig. 5. Edge Map Generation and Bounding Box Proposals

The EdgeBoxes region proposal algorithm consists of two steps. First, an edge map and orientation map is generated for an input image using a Structured Forests model. This can be seen in figure 5, 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 figure 5. It should be noted that, for visualization purposes, only 100 boxes were proposed in figure 5, whereas the actual number of boxes proposed will be an order of magnitude greater.

Implementation details for hardware and software parameters are as follows:

- 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 [34] implementation and model from [35].
- 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. Further testing and refining is needed to evaluate accuracy.

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 edge map generation was implemented. The premise here being that while EdgeBoxes

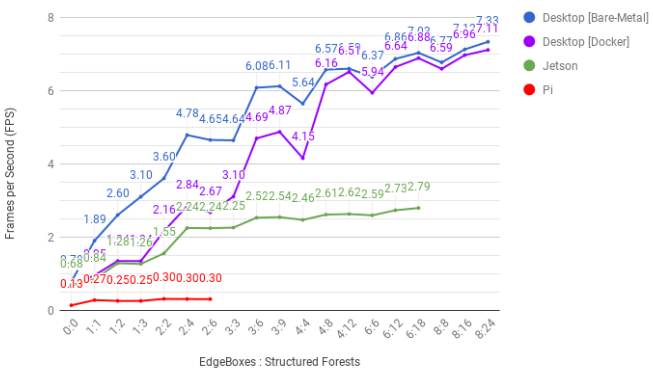

Fig. 6. EdgeBoxes Performance on Different Platforms

was generating proposals for the current frame, Structured Forests would generate the next frame's map to be read once EdgeBoxes was done with the current frame. Further performance gains were made by spawning multiple EdgeBoxes threads in addition to the Structured Forest threads, effectively creating multiple Structured Forests-Edgeboxes pipelines for bounding box prediction.

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 figure 6, 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. The Jetson TX2 sees gradual improvement until its 6 core processor becomes saturated with the number of threads. The Raspberry Pi sees an improvement in performance from consecutive 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, these results suggest that region proposal using EdgeBoxes would benefit from running on the CPU-bound cloud.

V. Undistributed R-CNN

To test the efficacy of a distributed R-CNN model, an undistributed model with similar architecture was trained. This undistributed model will be used as a benchmark for future work.

A. Pre-Trained Feature Extractor

Due to the fairly high cost of training, with regards to both hardware and time requirements, a method using pre-trained feature extractors is explored. These feature extractors are provided by Google with the Object Detection API [28] explicitly for the purposes of initializing models for quick time-to-market detections. These models have been trained on several different datasets using hardware inaccessible to the general public.

Provided models vary in terms of size, speed, accuracy, and the dataset trained on. The model used as a benchmark in this paper is the faster rcnn inception_v2 model trained on the coco dataset [29].

B. Fine-Tuning Dataset

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 [31] 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 [32] and consists of several thousand images from three different cameras perspectives at three different parking lots during various times throughout the day. Both of these datasets came with annotations describing the xy coordinates of bounding boxes for cars in the scene. However, these annotations were particularly inaccurate in the case of the second dataset because of the nature of the dataset (to describe parking lot occupancy instead of car location).

After cleaning up the annotation values, the final dataset used for fine-tune training and evaluation consists of 155 images, containing 5837 cars.

C. Performance/Results

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.

A similar approach to training the distributed R-CNN model will be taken, where a large dataset with several classes will be used for an initial training session. Then, the model will be fine tuned to the parking lot dataset.

VI. Further Work

While some progress has been made in implementing a region proposal network for the model, more work is needed to further improve the performance and evaluate the accuracy of EdgeBoxes. Additionally, the model requires design and implementation of a CNN feature extractor, as well as other remaining layers for pooling, classification, and bounding box regression. The interaction between EdgeBoxes and RoI pooling layers/bounding box regression needs to be investigated for optimization of object localization by the model.

As containerization and deployment for the distributed R-CNN model progresses, performance versus the undistributed

model will be evaluated. A method simulating the latency of wirelessly connected edge devices will be used to evaluate computational performance benefits of the distributed model.

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.

VII. Conclusion

This paper proposes a theoretical architecture for a distributed R-CNN model. This model will be based on the Fast R-CNN architecture [2] which is more efficient than R-CNN [1], but, unlike Faster CNN's [3] combined RPN-CNN layers, still consists of the distinct, separable components necessary in a distributed deployment model. The region proposal algorithm component of the distributed R-CNN model was investigated as a first step in developing and deploying the distributed model. After review of current object detection literature, EdgeBoxes [24] was chosen as the region proposal algorithm of the distributed R-CNN model. Performance increases were gained through multithreading the different stages [26, 27] of EdgeBoxes. 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] Uijlings Jasper RR, et al. "Selective search for object recognition." International journal of computer vision 104.2 (2013): 154-171.

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

[6] 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.

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

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

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

[10] 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.

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

[12] Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on. IEEE, 2009.

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

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

[15] 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.

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

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

[18] 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.

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

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

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

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

[23] 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.

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

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

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

[27] 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.

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

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

[30] 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.

[31] 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.

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

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

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

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

[36] 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.

[37] 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.

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

[39] 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.