Enhancing the Performance of Deep Learning Model Based Object Detection using Parallel Processing (Work In Progress Paper)

Omar Imran
Carleton University
Ottawa, ON, Canada
omarimran@cmail.carleton.ca

Shikharesh Majumdar
Carleton University
Ottawa, ON, Canada
majumdar@sce.carleton.ca

Sreeraman Rajan
Carleton University
Ottawa, ON, Canada
sreeramanr@sce.carleton.ca

ABSTRACT

The need for accelerated object detection is paramount for safety critical applications such as autonomous vehicles. This paper focuses on leveraging parallel processing techniques for enhancing the performance of object detection. Specifically, this research engineers system performance by timely detection of common objects encountered by vehicles, such as other automobiles, pedestrians, and bicycles. Deploying popular pretrained deep learning models like the You Only Look Once (YOLO) model within the Apache Spark framework, the potential enhancements in detection speed achieved through parallel processing are investigated. The capability of the system to efficiently handle large datasets and distribute time-critical applications across multiple nodes is explored to improve both latency and scalability. The one-factor-at-a-time method is used to assess the impact of different system and workload parameters on performance. Of particular interest is the impact of Spark data partitioning on performance, especially for driving scenarios where the number of objects are changing rapidly. A novel data partitioning technique that uses the principles of entropy is utilized. The overall performance objective of this research will be to improve speed for object detection in cars which can improve safety in time critical events such as sudden braking or turning.

CCS CONCEPTS

• **Computing methodologies** → **Massively parallel algorithms**; **Object detection**.

KEYWORDS

Parallel Processing, Object Detection, Deep Learning, Apache Spark, Video Processing

ACM Reference Format:
Omar Imran, Shikharesh Majumdar, and Sreeraman Rajan. 2024. Enhancing the Performance of Deep Learning Model Based Object Detection using Parallel Processing (Work In Progress Paper). In *Companion of the 15th ACM/SPEC International Conference on Performance Engineering (ICPE '24 Companion), May 7–11, 2024, London, United Kingdom.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3629527.3651427

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPE '24 Companion, May 7–11, 2024, London, United Kingdom
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0445-1/24/05
https://doi.org/10.1145/3629527.3651427

1 INTRODUCTION

Timely detection of objects is crucial in safety critical applications such as autonomous vehicles and has become an important subject of research due to increase of accidents happening with autonomous vehicles such as self driving cars [10, 21, 23]. Particularly, the incident when a Tesla in autopilot mode failed to stop at a red light and crashed it into another car killing two people questioned the safety of autonomous vehicles [24]. Object detection is crucial for ensuring the safety of autonomous vehicles and other safety critical applications, such as the detection of objects by Unmanned Aerial Vehicles [32]. The focus of this paper is to use parallel processing to improve object detection.

This work in progress paper specifically concentrates on the identification of objects like pedestrians, traffic lights, and trucks encountered by moving vehicles. The research will utilize the MIT DriveSeg dataset, featuring video frames captured by a camera mounted on a moving vehicle [6]. These video frames capture various driving environments including rural roads, busy highways, and city streets, enabling the simulation of diverse scenarios encountered by vehicles in daily life. Then, using the You Only Look Once Version 3 (YOLOv3) pretrained deep learning model and the Apache Spark parallel computing framework, object detection on the frames will be carried out in parallel. This research will utilize Amazon Web Services (AWS) cloud servers to create scalable Spark clusters which will facilitate in the deployment of the proof-of-concept prototype used in the different experiments. The experiments will use the one-factor-at-a-time approach [27] to systematically alter various system parameters, including the number of worker nodes and the average number of objects in each scenario, to assess their individual impact. This paper will also explore the effect of data partitioning strategies on performance when evenly distributing dynamically changing frames across the various nodes. The paper will introduce a unique method for estimating workload and allocating frames across partitions using frame entropy.

The contributions of this paper include:

- A frame entropy based novel technique for workload estimation and frame allocation across the different nodes in a parallel processing platform such as Spark.
- A proof-of-concept prototype deployed on Spark for analyzing the performance of the concurrent object detection techniques discussed in the paper. This includes combining the YOLOv3 pretrained deep learning model with Apache

Spark's distributed computing framework thus enabling the performance engineering of deep learning based object detection on parallel platforms.
- Initial insights into the impact of different system and workload parameters on the performance of the object detection techniques.

Further contributions to the state of the art are expected from continuation of the research as discussed in Section 6.

Previous research done has shown a 60% performance increase when deploying deep learning models on Spark clusters, suggesting the potential of parallel programming in object detection tasks [20]. The YOLOv3 model has been vastly used in the literature for object detection for applications such as the detection of unmanned aerial vehicles [11]. Furthermore, research has been done in the literature as well on different partitioning algorithms in Spark. They analyzed different partitioning approaches for a textual dataframe and concluded that Spark's standard random Hash Partitioning could be optimized further with custom partitioning strategies [22]. This further indicates that the choice of partitioning algorithm could also play a significant role in other applications such as enhancing performance in video processing. Notably, while most research papers emphasize the quantity of data points in partitioning studies, this paper uniquely concentrates on the specific content within each partition to estimate workload.

This paper is organized as follows. Section 2 gives a background on the methodologies used in this research. Section 3 gives an overview on the design of the system and the experiments. Section 4 goes over the experimental results. Section 5 concludes the paper and Section 6 gives a summary of the steps for further work.

2 BACKGROUND

This section will provide a background on the different topics used in this paper.

2.1 Apache Spark

Apache Spark can be used to efficiently distribute workloads and data across a cluster of machines or nodes and is used for distributing the processing of the video frames. This distributed computing framework specializes in parallel task execution, enabling fast processing and analysis of big data tasks such as video processing [18, 30]. Its in-memory computing capability reduces disk access, significantly boosting processing speeds compared to alternatives such as the Hadoop Distributed File System [31]. Spark supports multiple programming languages and offers various libraries for data processing tasks, such as PySpark.

2.1.1 Partitioning . A key component of Apache Spark concerns partitioning which is fundamental in determining the processing speed of the job at hand. Partitioning involves splitting up the data and tasks into smaller partitions that will ultimately be processed in parallel across the various nodes [22]. Ensuring a balanced distribution of tasks across partitions is critical to prevent any single node from becoming a bottleneck due to an excessive workload compared to others.

In this paper, the frames will be split up into different partitions that will be distributed across the various nodes. Traditional approaches typically divide data into partitions of equal size, basing the division on the quantity of data [8], which can be useful for tabular or time series data. However, in this research, information from each data point (i.e., the video frames) will be leveraged to dynamically create the partitions and better split up the workload.

2.2 Dataset

In this paper, the MIT DriveSeg dataset, which contains images of different driving scenarios that were acquired using a camera mounted on top of a car during the daylight was utilized. The dataset contains 20,100 video frames from 68 different driving scenarios [6]. The different scenarios depict different situations such as driving through a busy downtown street with a large number of potential objects or driving on a rural road with very few objects. The images have annotations for the different objects that have been labeled using semi-automated methods [7]. Some annotations may not be accurate because of semi-automatic labeling. Therefore, the annotations and labels have not been used in this paper.

2.3 Object Detection Techniques

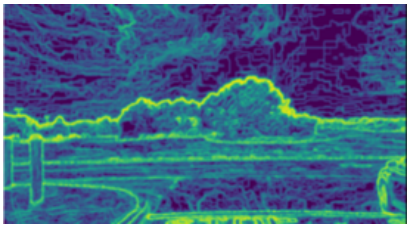
As mentioned in Section 2.2, the annotations that came with the MIT DriveSeg dataset, were shown not to be accurate due to the use of semi-automated methods which provided the segmentation of the images. Therefore, in this work, object detection of the driving scenarios is done through the utilization of bounding boxes. A bounding box is a rectangular frame that can be used to identify objects within a given video frame [17].

2.3.1 YOLO Pretrained Model. The YOLOv3 model was used to eliminate the need to train a deep learning model from scratch. The use of YOLOv3 resulted in saving time by eliminating the need for precise labels and allowing for the implementation of transfer learning.

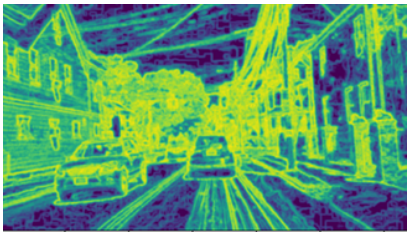
YOLO models are typically trained and optimized on a large dataset such as the Common Objects in Context (COCO) dataset [16]. Then, the model's weights can be further optimized or reused in other smaller datasets for object detection. YOLOv3 has shown improved metrics such as mean Average Precision (mAP) and also better processing speed when compared with other YOLO and pretrained models [16]. Additionally, YOLOv3 has been effective in maintaining a balance between detection speed and accuracy [13]. The YOLOv3 model has 80 possible classes of objects that can be detected within each image [12]. YOLOv3 is capable of detecting various objects that can be encountered in a driving scenario, such as cars, traffic lights, buses, trucks, street signs, pedestrians, and more. Each object found in the image can be represented using a bounding box.

2.3.2 Entropy . As mentioned in Section 2.1.1, one of the objectives of this research is to dynamically create partitions based on the estimated workload. A frame that has an abundant number of objects will take longer to process compared to a frame that has fewer objects. Therefore, there is a need to swiftly estimate the workload so that it can be split up into balanced partitions.

$$Entropy = -\sum_{i=1}^{L} p_i \log_2(p_i) \tag{1}$$



(a) Frame with low entropy



(b) Frame with high entropy

Figure 1: Two different driving scenarios illustrating low and high entropy.

Entropy can be used to determine the uncertainty or complexity in an image. Entropy can also be mathematically defined as the probability of occurrence p_i at gray level i where L is the maximum pixel value [5, 19, 26]. Overall, equation 1 is computing the uncertainty in each pixel of the frame. Furthermore, local entropy can be used for images to examine the variance in images given a window or a neighbourhood [26]. In Figure 1, there are two frames from the MIT DriveSeg shown with their entropy overlaid on top of each other. Pixels that are brighter have a higher local entropy and vice versa. Homogeneous and less complex images (see Figure 1a) have a lower entropy when compared to heterogeneous and more complex images (see Figure 1b). Therefore, entropy can serve as a quick indicator of the number of objects in a frame, under the assumption that more complex images typically contain a greater number of objects, and simpler ones contain fewer.

3 PROPOSED APPROACH

This section will provide the overall approach used in devising the proposed technique and a description of how the proof-of-concept (POC) was implemented.

3.1 System Design

This system POC for the proposed technique was hosted on Amazon Web Services (AWS). AWS is a cloud platform which provides scalable computing and storage services [3]. AWS offers an ideal platform for rapidly deploying different resources which can have varying hardware specifications like the number of cores.

The MIT DriveSeg dataset was first uploaded to a Simple Storage Service (S3) bucket which is a durable and scalable data storage tool in AWS [4]. After the data was uploaded, a Spark cluster was created. AWS provides a service known as Elastic Map Reduce (EMR) which can be used to create and manage a Spark cluster with a main node and multiple worker nodes. EMR uses the Hadoop Yet Another Resource Negotiator (YARN) manager to organize the different nodes. The main node requests different resources such as memory or CPU cores from the YARN manager which then distributes the tasks among the worker nodes [4].

Each node in the cluster represents an AWS Elastic Cloud Computing (EC2) instance which is a cloud server containing various libraries including Python and Spark [4]. Additionally, PySpark, the Python library for Apache Spark, is installed on every node in the cluster, serving as the main library for the program in this paper. EC2 instances can be scaled up to include varying number of nodes which have varying numbers of cores. For the POC, the m5 family of EC2 instances which can have anywhere from 2 to 96 cores and from 8 to 384 GB of memory for each node were used [2].

Figure 2 gives an illustration of the system and can be described with the following steps.

(1) The EMR cluster will read from the S3 bucket containing frames from a driving scenario.
(2) The main node will load in the YOLOv3 model with weights trained on the COCO dataset using TensorFlow, a Python deep learning library which can be used to create and load neural network models [14].
(3) The YOLOv3 model is broadcast in the main node. In Spark, memory-intensive variables can be broadcast to provide a read-only version on the main node instead of replicating them across all worker nodes [29].
(4) The tasks and frames are submitted to the main node. The frames are split into partitions based on the chosen partitioning algorithm that are outlined in Table 1.
(5) The YARN manager performs task scheduling with the worker nodes.
(6) The worker nodes concurrently detect the objects in their assigned frames using the YOLOv3 model.
(7) The bounding box detections are sent back to the main node.
(8) Once all worker nodes are done with their tasks, the results are stored in a S3 bucket.

3.2 Experimental Design

The experiments used the one-factor-at-a-time approach where all the factors in the experiment are set to their default values while one factor was varied at a time [27]. The processing time required by the worker nodes to complete all of the detections is the measured metric for evaluating performance. The measurement of processing time is the duration from the moment the frames are dispatched to the worker nodes until the completion of all detections. The one-factor-at-a-time approach helps in identifying the key factors that impact the processing time. The different factors and their values are shown in Table 1, with the default value shown in bold. The selection of each factor will be further explained in this section.

3.2.1 Distribution Framework. The distribution framework is a simple factor that will compare whether the use of distributed computing is useful for the problem. Using no distribution will mean running the program on a single node and with no parallelism which will be compared to running it using multiple nodes on Apache Spark.

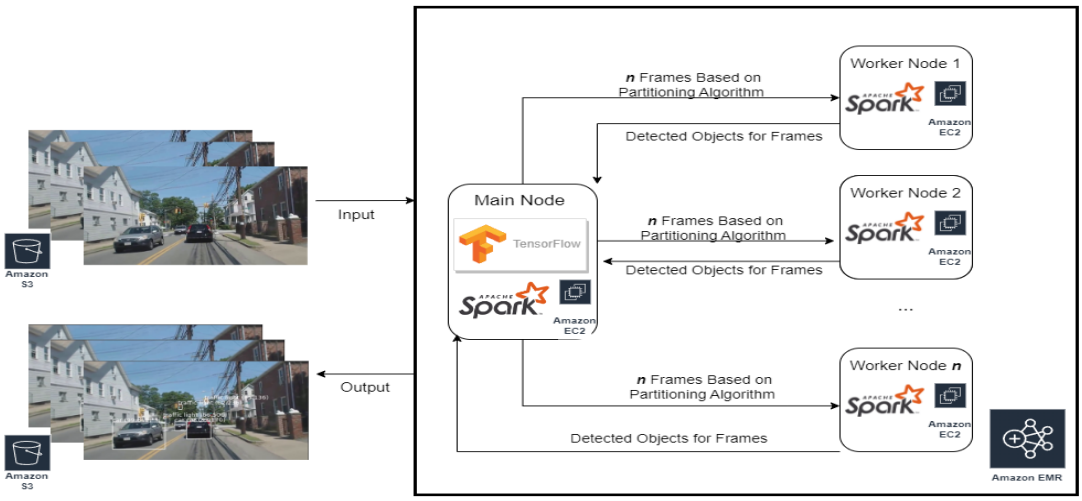


Figure 2: Overall system design for object detection.

3.2.2 Number of Objects . The number of objects within the scenario will be compared to evaluate whether this led to a difference in processing time. The hypothesis that needs to be verified is that as the average number of objects in a scenario increases, it would increase the overall workload for the system. For example, a driving scenario in a busy urban street, would require a lengthier duration for object detections, unlike a sparsely populated rural road with few objects.

3.2.3 Number of Worker Nodes . The number of worker nodes will be varied to analyze the impact of parallelism on performance.

3.2.4 Number of CPU Cores. Similar to Section 3.2.3, the number of CPU cores within each worker node will be changed to investigate the impact of core parallelism on performance.

3.2.5 Total Number of Frames . The total number of frames in a video being processed will be varied to assess concomitant changes in throughput. System throughput is defined as the number of frames processed per second [27].

3.2.6 Partitioning Algorithm . As mentioned in Section 2.1.1, one of the important objectives of this paper is to experiment with different partitioning algorithms. The number of partitions in Spark can be set manually. In most cases, setting the number of partitions to the number of total cores has been observed to yield good results [9].

For experimenting with this parameter, two different video scenarios were combined; one scenario concerning a busy street with a lot of objects and the other concerning a rural road with fewer objects. The combined video is meant to replicate a scenario where the number of objects varies throughout the video. For this case, partition splits should be based on the estimated workload in each partition and not based on the number of frames. One partition could have frames from the scenario that has numerous objects meaning it could become the performance bottleneck for the overall system. This is because as mentioned in Section 3.2.2, as the number of objects increase in a frame, the overall processing time for the frame is expected to increase.

The four partitioning algorithms are experimented with include the following.

(i) **Video-based**: In Video-based partitioning, each partition split will have frames from the same video scenario.
(ii) **Spark-based**: In Spark-based partitioning, the partitions will be assigned frames using the default Spark settings. Spark's default partitioning mechanism uses Hash Partitioning [9]. For this, a hash function is used to randomly assign the frame to a partition. However due to its randomness, it could result in a skewed data distribution as some partitions may contain larger portions of the data while some partitions could be empty [25, 28].
(iii) **Object-based**: In Object-based partitioning, each partition will have an equal number of frames from each video. Therefore, this implies that each partition should contain roughly an equal number of objects, based on the assumption that every frame from the each scenario possesses a similar quantity of objects. To estimate the workload, one random frame was sampled from each scenario with the assumption that frames with higher number of objects will have a higher workload and vice versa. To minimize overheads, the YOLOv3 model was deployed on this one frame to estimate whether there is a high or low number of objects within the whole scenario. A threshold of 7 objects was established through experimentation, categorizing a frame with 7 or more objects as an indicator of high workload and and as an indicator of low workload otherwise.
(iv) **Entropy-based**: As mentioned in Section 2.3.2, entropy can be used to estimate the workload resulting from a given frame, using the observation that frames with more objects will generally have a larger entropy value than those that do not. Therefore, similar, to Object-based partitioning, Entropy-based partitioning can be used to estimate the workload based on the contents of the frames. One of the benefits of using Entropy-based partitioning is its fast computation

speed, meaning it can be applied for all the frames. In contrast, Object-based partitioning can only be applied to one frame since it requires running the YOLOv3 model to get an estimate of the number of objects, which requires significantly more computation power compared to calculating the entropy of a frame. Additionally, to minimize system overheads, it is necessary to select a simple metric like entropy for estimating workload. Entropy-based partitioning is presented in *Algorithm 1*. First, in *Algorithm 1*, the videos and their frames are loaded into a variable known as *df* as shown in lines 1 to 4. Then the entropy of each frame is found on line 5. From lines 6 to 12, the frames are classified as having a low or high number of objects based on if they have a low or high entropy. The threshold value, found experimentally, will distinguish whether a frame has more or less objects. Once the groups are made, each partition will contain an equal number of *LESS_OBJECTS* and *MORE_OBJECTS* frames, as shown in lines 13 and 14.

Algorithm 1 High Level Algorithm for Partitioning Frames Based on Entropy

```
1:  for video in videos do
2:      files = files.append(load_video(video))
3:  end for
4:  df = spark.read.format("binaryFile").load(files)
5:  df["entropy"] = getEntropy(df["frames"])
6:  for row in df do
7:      if row["entropy"] < THRESHOLD then
8:          row["group"] = LESS_OBJECTS
9:      else
10:         row["group"] = MORE_OBJECTS
11:     end if
12: end for
13: partitions = df.rdd.getNumPartitions()
14: df = partitionFrames(df, partitions)
```

4 EXPERIMENTAL RESULTS

The results for the experiments will be summarized. Each experiment was executed 10 times to calculate a mean and a standard deviation for the processing time.

Table 1: System and Workload Parameters.

Factor	Value
Distribution Framework	(None, **Apache Spark**)
Number of Objects	(1-2, **7-8**, 12-13)
Number of Worker Nodes	(1, 2, **3**, 4)
Number of CPU Cores in Worker Nodes	(4, **8**, 16, 32)
Number of Total Frames	(150, **300**, 900, 2700)
Partitioning Algorithm	(Video, **Spark**, Object, Entropy

Figure 3 shows that using Spark results in 50% decrease in processing time, compared with using no distribution framework. This establishes that using distributed computing is proven to be beneficial for the given problem.

Figure 4 depicts that as the number of objects increase in the frames, the processing time increases. This means that the workload in the system is related to the average number of objects in each scenario. This observation is important as it forms a necessary criterion for differentiating between various partitioning algorithms.

Figure 5 show that as the computation power increases, the processing times decreases. As the number of worker nodes increases, the performance improves. Similar results were found when increasing the number of cores.

Figure 6 shows that as the number of frames increase, the system throughput also increases. This is an expected result because as the size of the dataset increases, the throughput is expected to increase as the impact on overall performance of fixed overheads due to Spark, decreases with a larger dataset [1, 15].

Figure 7 shows the comparative performance of the different partitioning algorithms for two different scenarios, one on a busy street with a large number of objects and one on a rural street with no objects. By combining these scenarios a simulated dynamic video is created where the number of objects are varying from one frame to another. As shown, the worst performance was achieved by the Video-based algorithm where each partition contains frames from only one scenario which leads to uneven workload distribution. In Video-based partitioning, certain partitions may turn into bottlenecks due to the allocation of frames from scenarios with higher workloads. Spark's default partitioning algorithm was the second worst in terms of performance. As described in Section 3.2.6, Spark's default partitioning algorithm relies on a random Hash partitioner which may lead to a skewed data distribution. The randomness in results can be seen as the standard deviation after 10 runs of the algorithm is high, compared to the other algorithms analyzed in this research. This implies that relying on Spark's default algorithm for data distribution in a dynamic video scenario may not always give effective results. Object-based partitioning produced the best results in Figure 7. However, as mentioned in Section 3.2.6, using a single frame from each video to assess workload may lead to inaccuracies in dynamic scenarios and may not be a good choice to use for workload partitioning. If the video has varying number of objects, then it is not sufficient to use one frame to estimate the workload. Entropy-based partitioning produced comparable performance results as Object-based. The entropy for each frame was computed, making this approach the most accurate partitioning algorithm as it is frame-specific. Furthermore, as shown in Figure 7, Entropy-based partitioning achieved an improvement of ~13% in processing time when compared to Spark's default partitioning algorithm.

As mentioned in Section 3.2.6, for Entropy-based partitioning a threshold parameter needs to be set so that the workload of the frame can be classified. To experimentally find the threshold value, three scenarios that had a large number of objects and three scenarios that had a few objects were analyzed to calculate the entropy of each frame (see Figure 8). Each bin in the histogram is the number of frames that correspond to the energy value captured in the x-axis. As shown in the figure, there is a clear difference

between the two groupings. From these results, the threshold value used in *Algorithm 1* was set at 2.5×10^6, since this is the value that can differentiate a high workload frame from a low workload frame.

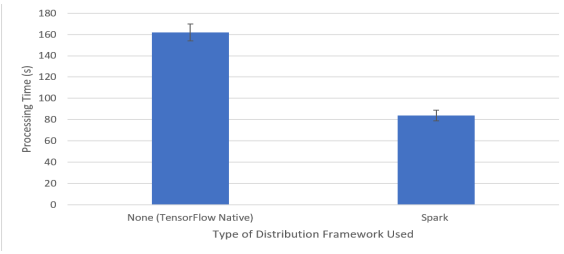


Figure 3: Processing time for object detection versus distribution framework.

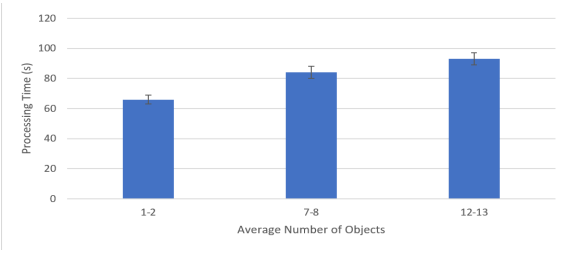


Figure 4: Processing time for object detection versus average number of objects in each frame.

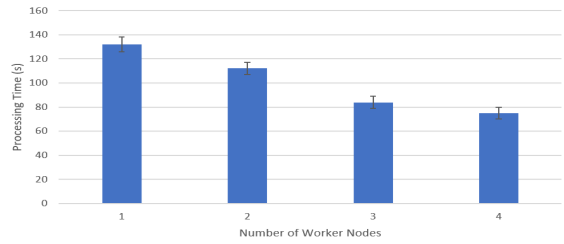


Figure 5: Processing time for object detection versus number of worker nodes.

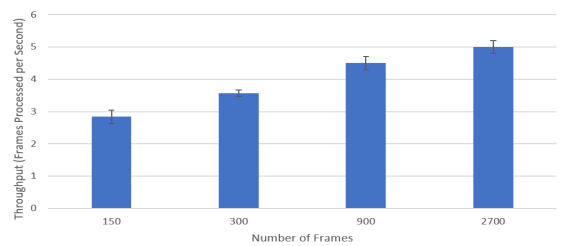


Figure 6: Throughput versus total number of frames being processed.

5 CONCLUSIONS

This paper demonstrated how parallel processing techniques deployed on Apache Spark can be used to improve performance of the deep learning based YOLOv3 model for detecting objects in videos. First it was shown that using parallel processing is significantly faster than using only TensorFlow with no parallel programming. The results showed that as the number of objects in the videos increased, the processing time also increased, this demonstrating the correlation between the workload intensity and the number of objects in the scenarios. As the computation power is increased by adding more worker nodes or CPU cores, the performance improved as well. Throughput was also found to be related to the size of the videos. Different partitioning algorithms for dynamic workload distribution were studied. The Entropy-based partitioning algorithm showed improved performance in processing the detections when compared to Spark's random partitioning.

6 FUTURE WORK

The preliminary results presented in this research open up several items for future work which will include the following:

(i) The investigation of longer videos is one of the important components of future work. For the initial work presented in this pilot project, each video was 300 frames long. In the results, it was shown that as the size of the dataset increases, parallel processing becomes more beneficial. Therefore, experimentation with larger videos, that give rise to larger datasets, need to be carried out. Furthermore, longer videos

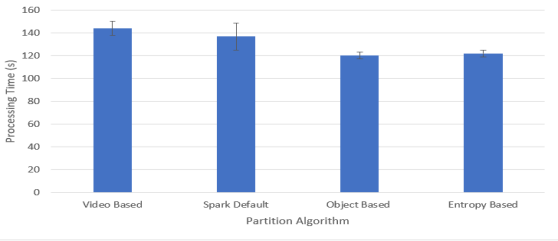


Figure 7: Processing time for detection for different partitioning algorithm.

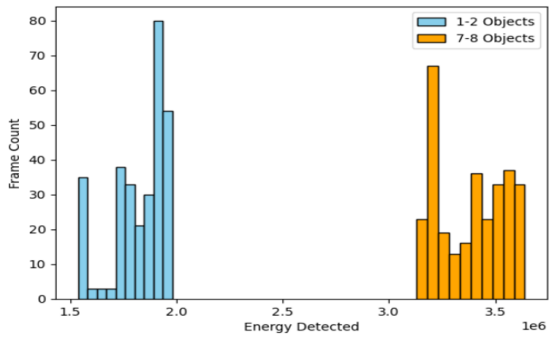


Figure 8: Distribution of entropy grouped by number of objects.

that have dynamically changing scenes need to be used to further confirm the usefulness of Entropy-based partitioning.

(ii) The removal of redundant frames is expected to improve system performance. In this paper, every single frame in the video was used for detection; however, many of these frames have overlapping information. Performance would improve by sampling a fixed number of frames once a substantial change has taken place. Algorithms for identifying redundant frames will be investigated. Improving the latency of object detection will also be useful for real time applications.

(iii) Incorporating the use of other deep learning models will be another direction for future work. In this paper, the YOLOv3 model was used for object detection but there are newer versions of YOLO models, the performance of which can be compared with the results from this research. Other pre-trained models such as EfficientNet will also be investigated.

(iv) Computing the accuracy of object detections would be included in future work. In this research, the accuracy of the bounding boxes were not computed. Metrics like mean Average Precision (mAP) and Jaccard index are used to determine the accuracy of the bounding box. Analysis of the potential trade-offs between speed and accuracy will be performed.

(v) Future work will also look to distribute pixels from individual frames among the different nodes using parallel processing.

(vi) The use of Graphical Processing Units (GPUs) will be used in future work to study its impact in performance, given their proven ability to speed up processing times in deep learning applications [27].

ACKNOWLEDGMENTS

This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Nasim Ahmed, Andre LC Barczak, Teo Susnjak, and Mohammed A Rashid. 2020. A comprehensive performance analysis of Apache Hadoop and Apache Spark for large scale data sets using HiBench. *Journal of Big Data* 7, 1 (2020), 1–18.

[2] AWS. [n. d.]. Amazon EC2 M5 Instances. https://aws.amazon.com/ec2/instance-types/m5/

[3] Ignacio Bermudez, Stefano Traverso, Marco Mellia, and Maurizio Munafo. 2013. Exploring the cloud from passive measurements: The Amazon AWS case. In *2013 Proceedings IEEE INFOCOM*. IEEE, 230–234.

[4] Lin Chen, Rui Li, Yige Liu, Ruixuan Zhang, and Diane Myung-kyung Woodbridge. 2017. Machine learning-based product recommendation using Apache Spark. In *2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI)*. IEEE, 1–6.

[5] Sandipan Dey. 2018. *Hands-On Image Processing with Python: Expert techniques for advanced image analysis and effective interpretation of image data.* Packt Publishing Ltd.

[6] Li Ding, Michael Glazer, Jack Terwilliger, Bryan Reimer, and Lex Fridman. 2020. MIT DriveSeg (Semi-auto) Dataset. https://doi.org/10.21227/nb3n-kk46

[7] Li Ding, Jack Terwilliger, Rini Sherony, Bryan Reimer, and Lex Fridman. 2020. MIT DriveSeg (Semi-auto) Dataset: Large-scale Semi-automated Annotation of Semantic Driving Scenes. *Massachusetts Institute of Technology AgeLab Technical Report* 2 (2020).

[8] J Geetha and N. G. Harshit. 2019. Implementation and Performance Comparison of Partitioning Techniques in Apache Spark. In *2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*. 1–5. https://doi.org/10.1109/ICCCNT45670.2019.8944759

[9] Anastasios Gounaris, Georgia Kougka, Ruben Tous, Carlos Tripiana Montes, and Jordi Torres. 2017. Dynamic Configuration of Partitioning in Spark Applications. *IEEE Transactions on Parallel and Distributed Systems* 28, 7 (2017), 1891–1904. https://doi.org/10.1109/TPDS.2017.2647939

[10] Edward Helmore. 2022. Tesla behind eight-vehicle crash was in "full self-driving" mode, says driver. https://www.theguardian.com/technology/2022/dec/22/tesla-crash-full-self-driving-mode-san-francisco

[11] Bowen Li, Nat Shineman, Jayson Boubin, and Christopher Stewart. 2021. Comparison of Object Detectors for Fully Autonomous Aerial Systems Performance. In *Companion of the ACM/SPEC International Conference on Performance Engineering* (Virtual Event, France) *(ICPE '21)*. Association for Computing Machinery, New York, NY, USA, 165–166. https://doi.org/10.1145/3447545.3451170

[12] Tao Li, Yitao Ma, and Tetsuo Endoh. 2020. A Systematic Study of Tiny YOLO3 Inference: Toward Compact Brainware Processor With Less Memory and Logic Gate. *IEEE Access* 8 (2020), 142931–142955. https://doi.org/10.1109/ACCESS.2020.3013934

[13] Ignacio Martinez-Alpiste, Gelayol Golcarenarenji, Qi Wang, and Jose Maria Alcaraz-Calero. 2021. A dynamic discarding technique to increase speed and preserve accuracy for YOLOv3. *Neural Computing and Applications* 33, 16 (2021), 9961–9973.

[14] Bo Pang, Erik Nijkamp, and Ying Nian Wu. 2020. Deep learning with tensorflow: A review. *Journal of Educational and Behavioral Statistics* 45, 2 (2020), 227–248.

[15] Md Armanur Rahman, J Hossen, and C Venkataseshaiah. 2018. SMBSP: a self-tuning approach using machine learning to improve performance of spark in big data processing. In *2018 7th International Conference on Computer and Communication Engineering (ICCCE)*. IEEE, 274–279.

[16] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. arXiv:1804.02767 [cs.CV]

[17] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. 2019. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 658–666.

[18] Sajad Sameti, Mea Wang, and Diwakar Krishnamurthy. 2018. Stride: Distributed video transcoding in spark. In *2018 IEEE 37th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 1–8.

[19] scikit image. [n. d.]. Entropy. https://scikit-image.org/docs/stable/auto_examples/filters/plot_entropy.html

[20] Arindrajit Seal and Arindam Mukherjee. 2019. Real Time Accident Prediction and Related Congestion Control Using Spark Streaming in an AWS EMR cluster. In *2019 SoutheastCon*. 1–7. https://doi.org/10.1109/SoutheastCon42311.2019.9020661

[21] David Shepardson. 2023. GM's cruise recalling 950 driverless cars after pedestrian dragged in ... https://www.reuters.com/business/autos-transportation/gms-cruise-recall-950-driverless-cars-after-accident-involving-pedestrian-2023-11-08/

[22] Tinku Singh, Shivam Gupta, Manish Kumar, et al. 2023. Performance analysis and deployment of partitioning strategies in apache spark. *Procedia Computer Science* 218 (2023), 594–603.

[23] Lauren Smiley. 2023. The legal saga of Uber's fatal self-driving car crash is over. https://www.wired.com/story/ubers-fatal-self-driving-car-crash-saga-over-operator-avoids-prison/

[24] Hayley Smith and Russ Mitchell. 2022. A Tesla on autopilot killed two people in Gardena. is the driver guilty of manslaughter? https://www.latimes.com/california/story/2022-01-19/a-tesla-on-autopilot-killed-two-people-in-gardena-is-the-driver-guilty-of-manslaughter

[25] H. S. Sreeyuktha and J. Geetha Reddy. 2019. Partitioning in Apache Spark. In *Innovations in Computer Science and Engineering*, H. S. Saini, Rishi Sayal, Aliseri Govardhan, and Rajkumar Buyya (Eds.). Springer Singapore, Singapore, 493–498.

[26] Badri Narayan Subudhi, Pradipta Kumar Nanda, and Ashish Ghosh. 2011. Entropy based region selection for moving object detection. *Pattern recognition letters* 32, 15 (2011), 2097–2108.

[27] Azhar Talha Syed and Shikharesh Majumdar. 2022. Parallel Processing Techniques for Analyzing Large Video Files: a Deep Learning Based Approach. In *2022 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCloud/SocialCom/SustainCom)*. 270–279. https://doi.org/10.1109/ISPA-BDCloud-SocialCom-SustainCom57177.2022.00041

[28] Zhuo Tang, Wei Lv, Kenli Li, and Keqin Li. 2018. An intermediate data partition algorithm for skew mitigation in spark computing environment. *IEEE Transactions on Cloud Computing* 9, 2 (2018), 461–474.

[29] Isaac Triguero, Mikel Galar, D Merino, Jesus Maillo, Humberto Bustince, and Francisco Herrera. 2016. Evolutionary undersampling for extremely imbalanced big data classification under apache spark. In *2016 IEEE congress on evolutionary computation (CEC)*. IEEE, 640–647.

[30] Md Azher Uddin, Aftab Alam, Nguyen Anh Tu, Md Siyamul Islam, and Young-Koo Lee. 2019. SIAT: A distributed video analytics framework for intelligent video surveillance. *Symmetry* 11, 7 (2019), 911.

[31] Ankush Verma, Ashik Hussain Mansuri, and Neelesh Jain. 2016. Big data management processing with Hadoop MapReduce and spark technology: A comparison. In *2016 symposium on colossal data analysis and networking (CDAN)*. IEEE, 1–4.

[32] Haijun Zhang, Mingshan Sun, Qun Li, Linlin Liu, Ming Liu, and Yuzhu Ji. 2021. An empirical study of multi-scale object detection in high resolution UAV images. *Neurocomputing* 421 (2021), 173–182.