International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 11, No.1 (Jan-2022)

https://dx.doi.org/10.12785/ijcds/110114

Deep Learning-Based Object Recognition in Video Sequences

Ashwith A.[1], Anurag Sethuram[1], Dr. Azra Nasreen[1], Dr. Shobha G.[1] and Dr. S. S. Iyengar[2]

[1]*Department of Computer Science and Engineering, R. V. College of Engineering, Bangalore, India*
[2]*Ryder Professor of Computer Science and Director of the School of Computing and Information Sciences, Florida International University (FIU), Miami*

Received 16 Mar. 2021, Revised 13 Oct. 2021, Accepted 13 Nov. 2021, Published 9 Jan. 2022

Abstract: This paper proposes to detect, recognize, and track a specific object in video sequences using Convolutional Neural Networks (CNNs). The CNNs used are the TensorFlow SSD model and Inception model, with a use case of airplane detection as a test subject, although it can be widely extensible to any class of objects as per the application. For the SSD model, images of planes were downloaded and annotated using bounding boxes to identify regions of interest. Training and test sets were split, after which TensorFlow specific records were generated. Whereas, for the Inception model, the last layer of the Neural Network was trained with multiple images of airplanes and random images to obtain a classifier for identify planes vs. no planes. The SSD model was accurate, generating crisp bounding boxes with a relatively high accuracy. The Inception model had a higher accuracy than the SSD model in terms of false positives and false negatives. But it does not display bounding boxes as the model is not meant to find the region of interest. The GPU outperformed the CPU in training and testing by a wide margin. The Inception model is suitable to extract frames in which a specific object is present if the position of the object is not of importance. The SSD model is suitable if the specific object needs to be detected with its position in a video frame.

Keywords: : TensorFlow, Object Recognition, Bounding Boxes, CNN, Neural Network, SSD, Inception, Plane, CPU, GPU

1. Introduction

Object recognition is used in computer vision, video analytics and image processing tasks, wherein the instances of semantic objects of a certain class (such as humans, buildings, cars and so on) in digital images and videos [1] are studied for various use cases. Object recognition has profound applications in various domains and is also a challenging task as only the target object from multiple other objects present in a frame must be identified and tracked regardless of the pose, variations in viewpoints, illuminations etc. It remains an open research problem even after several years of research in the field. One of the primary and extremely useful applications of object recognition is its use case in Physical Security Systems. It can be used to detect intruders and trigger alarms to enhance security in areas of high sensitivity. It can also be used to detect foreign objects such as drones in high surveillance and high security areas. Its utility also includes surveillance and detection near country borders against intruding vehicles or people. These systems help in taking corrective measures in real time to avoid any untoward incidents.

In any traditional object recognition problem, the main task is to identify different categories of objects in an image. It also seldom involves figuring out the pose of the so recognized objects [2]. A key method used in object recognition involves extracting features from images or video frames and are extremely important, as they describe characteristics of the object in question. The features are fed into a classifier to perform recognition. However, in most cases, extracting these features is not an easy task, due to a variety of factors such as noise in the image. A more commonly used method for object recognition is picture segmentation. In picture segmentation, a picture is divided into a variety of associated, important dimensions and a pixel grid in a specific dimension with respect to application. This process is dependent on the measurements of the image. In a segment of a picture, all pixels in a dimension are qualitatively like each other with respect to the hues, intensity, and surface consistency [3]. After this, feature extraction is performed to obtain important features. Finally, the features are fed to a classifier and object recognition is performed.

A more sophisticated technique which is used in certain applications is object recognition via contorted luminescence and form. In this technique, the shape of an object is guessed based on some approximations obtained from the scene object. This guess is usually called a model shape.

E-mail address: Ashwith_Atluri@intuit.com, anuragsethuram@gmail.com, azranasreen@rvce.edu.in, shobhag@rvce.edu.in, iyengar@cis.fiu.edu

These paradigms can actively relegate a model shape to a scene shape or vice versa. Thereafter, the cost of the distortion can be used to measure the similarity between shapes [4]. The same principle is extended to perform object recognition for a set of objects.

Apart from the techniques listed above, quite a few other techniques are also common in real time scenarios. In some cases, video object partitioning by making use of a unified Bayesian is employed. This technique helps segment objects from a rough form to a finer form. Thereafter, there are a few segment models which help in enhancing fragmentation results [5]. Another technique uses an algorithm which improves recognition and segments articles in the fore of an egocentric video. This paradigm is especially useful in a SIFT based recognition system [5]. A more modern approach involves automatically generating a semantic map on the scene data. Using this information, camera sensors are boosted to identify the article and dynamically programmed partitioning [5]. Semantic texton forests are employed for the binning of the image and segmentation of the object. This process involves the usage of hierarchical clustering and local image classification techniques [5].

With the advent of artificial intelligence and re-emergence of deep learning, many new techniques have come into the light for solving traditional object detection problems. This paper focuses on the specific object recognition problem of airplane detection in varying backgrounds. We will be employing Google's TensorFlow framework and applying Convolutional Neural Networks to perform this task.

With most traditional computer vision techniques, we face a few hurdles in pursuit of the optimal solution. For starters, the complexity of an object recognition problem increases once we bring in varying scene backgrounds, especially if it matches the object to be recognized in a scene. Secondly, the application of additional tracking algorithms if needed, after the object recognition process, is harder and more computationally intensive for each run of the system. These two problems are addressed rather well by using CNNs. The background of an object does not affect the prediction made and using techniques such as Single Shot Detection on a MobileNet model, the run-time processing time also reduces.

The other problem addressed with use of CNNs is the possibility of producing an extremely lightweight model (Inception), which can run on live videos with high frame rates. This is especially important for high security requirements, since for fast paced targets, a single frame can make or break the whole purpose of the system.

With more focus on deep learning-based applications, more and more techniques based on convolutional neural networks are coming into the scene. Efficient deep learning frameworks such as Google's TensorFlow, Caffe and so on are gaining a lot of traction due to their robustness and abstracted views.

Proposed system uses TensorFlow to detect and track specific objects in videos. Two models are compared after the training procedure to highlight the pros and cons of each. Also, a comparative analysis of the training and object recognition phases using CPUs (Intel Core series processors) and GPUs (NVIDIA GTX 970M) is performed. TensorFlow is an open-source library which is used for calculations using dataflow graphs and is especially useful for machine learning applications. Statistical features are represented by nodes in the graph. Tensors, which are multi-dimensional arrays shared between operations, form the edges in the graph. The architecture allows for users and developers to distribute complex arithmetic tasks between one or more cores (either CPU or GPU) in a desktop, server, or mobile device. This is all enabled through a single API. TensorFlow was originally envisioned to be employed for ML and Deep Learning-based research internally within Google's ML Research group. But the same was scalable enough to be applied for other use-cases as well. The Single Shot Detector (SSD) model [6], [7] and the Inception model are Convolutional Neural Networks (CNN) developed for precise processing of images. For this project, the above two models will be trained to detect and track airplanes in each video.

2. Proposed System

In brief, the methodology incorporated is as follows: For the SSD Model, images of airplanes were downloaded and were annotated using bounding boxes to identify regions of interest. Images were split into training and test sets, after which TensorFlow specific records were generated for training and test sets. The SSD model was trained independently on the CPU and GPU and tested at multiple checkpoints. For the Inception model, the last layer of Google's pertained Inception Neural Network was retrained with two sets of images, one of "Airplanes" and the other of "Generic" images. This generated a classifier that classifies images as either Planes or Not-Planes. The model was tested and compared with the SSD model on multiple criteria [8].

Although the experiment was to specifically detect planes in a video, the principle used here is easily extensible to the context of other objects as well for other applications [9]. It's also possible to detect and track more than one specific object class in a video. The only drawbacks being an increase in time spent annotating the training images and training the models.

The following are the design considerations:

- Appropriate split of the images into training and testing batches to ensure the model has an appropriate mix of different images

- Finding the optimal hardware on which the training of the model must be done which is effectively identifying the tradeoffs between speed and complexity of hardware

in the case of a CPU and GPU

• Identifying the optimal video quality on which the video must be run to ensure minimal lag and maximum accuracy

• Identifying the appropriate differences and use cases for TensorFlow SSD and TensorFlow Inception models

A. *System Architecture*

The system architecture is as depicted by Figure 1. The system can be divided into 3 main modules: Training, Models and Run-Time. The Training module is responsible for data pre-processing and training model checkpoints. The Model module is the frozen graph, which is run using the Runtime module. Finally, the Runtime module uses the frozen variables from the modules to classify and detect objects in the video.

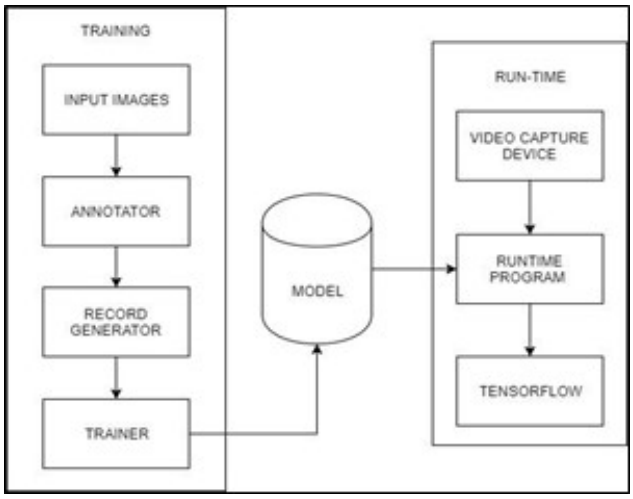

Figure 1. System Architecture

B. *Single Shot Detector*

For the SSD model the images of planes are downloaded from ImageNet and annotated to make them ready for use with the TensorFlow learning scripts. The dataset generator is the module that generates the dataset in the format, as required by the TensorFlow API. They were split into train and test sets and the corresponding .csv files for training were generated which were subsequently converted into tfRecords (TensorFlow compatible format) for training the SSD model.

The web scraper program is used to download relevant images from a Google Image Search. Once these images are downloaded, we manually go through them to ensure that they have the required features for our dataset. In our case, we downloaded images from the search for "Airplane".

These images are then run through the annotation program that lets us draw boxes around the area of interest in the images and store these in a generic XML format. These images and annotations are used for the training process with the TensorFlow training scripts. The Model trainer takes our generically annotated images and uses it to generate a model that is compatible with our runtime program.

The XML annotations are first fed to a CSV converter, which translates the XML annotations into a format that is accepted by the training scripts. These CSV annotations, along with the images, are fed to the training scripts. The training scripts use these to generate TFRecords, a TensorFlow specific storage format before the training phase. The training graph is used by the object detector, which is the runtime program, to detect images with features like those of the dataset [10]. The runtime uses a video capture device to extract frames from either a camera or a video file. These images are processed individually in the program loop.

Within the program loop, the individual frames are processed by feeding them to the trained model graph that provides the bounding box and confidence levels. These outputs are used to draw the bounding box and print details on the final output image, which is displayed as the output video stream [11], [12].

The SSD model was trained for about 100,000 steps on CPU and GPU and tested at multiple checkpoints.

C. *Inception Model*

For the Inception model, the last layer of Google's pertained Inception Neural Network was retrained with two sets of images, one of "Airplanes" and the other of "Generic" images. Since Inception is a classifier, one set of 600 images of planes was used to build the "plane" class and one set of 600 images was used to create a "generic" class. The inception model internally splits the input image set into training and validation sets in every step. The given image set is used to retrain the last layer of the inception model and generate a new TensorFlow Graph File which can be used in the runtime program. A total of 8000 steps were used to retrain the model. The model obtained here was of high accuracy.

3. Experimental Setup

For training the TensorFlow SSD model, the experimental setup on the hardware side included the MacBook Pro, with an Intel 2.9 GHz i5 Processor with 8 GB RAM and on an ASUS RoG with an Intel 2.5 GHz i7 Quad Core Processor, and a GeForce GTX 970M GPU.

The plane's images were downloaded from ImageNet, which is an open-source data repository for image-based datasets. To label the images, the labelImg annotation software was used. The software generated an xml file for each image with the edge coordinates of the regions of interest in the image.

For training the TensorFlow Inception model, the experimental setup used a MacBook Pro with an Intel 2.6 GHz

quad core i7 with 16 GB RAM. Since the training wasn't as rigorous as the SSD training process, only CPU computing was utilized for training the Inception model training. The images used to train the model were from the Common Objects in Context dataset (COCO).

A. Batch Size Selection

The batch size is essentially the number of images the hardware can simultaneously support in each step while training the model in parallel to ensure faster training. The optimal batch size for a given hardware specification would be the one that would prevent memory overflow error while ensuring that the training time is optimized [12]. For the CPU and the GPU, the batch sizes that were tested were 1, 2, 10, 14 and 16. The results were observed as follows: a memory overflow error was encountered for a Batch size of 16 or greater on the GPU; the same error was encountered with a batch size of 4 or greater on CPU. It was decided that for the CPU, a batch size of 2 was optimal, while for the GPU, a batch size of 16 was optimal. Each step took around 1.35 seconds on the GPU and around 20.57 seconds on the CPU.

B. Video Optimality Test

The next testing module was to determine the optimal video resolution range for best detection and minimal latency. Different video qualities were used for detection on the GPU to compare the video latencies and detection accuracies and hence determine a range of video qualities for the purpose of this test [13]. The video qualities (resolutions) used were 144p, 360p, 480p, 720p, 1080p. The model was found to be inaccurate with 144p, and a slight improvement was observed with 360p. There were a few errors with 480p and a good performance improvement was observed with 720p. While it was found to be more accurate, 1080p showed a significant increase in latency. The video quality from 480p to 720p was optimal for speed on the above specified GPU hardware.

In the above test, though 480p and 720p showed optimal speed among all other resolutions, it is fairly apparent that a higher resolution would mean better object detection accuracy, since more pixels are available around the desired object [14], [15]. However, there were a few exceptions to this rule, which were more random than explanatory.

4. Results and Analysis

The following chapter highlights the results and analysis of the proposed system, which includes the CPU vs GPU training time, the analysis of weights and biases, training checkpoint analysis, comparison of TensorFlow SSD and TensorFlow Inception model.

A. CPU vs. GPU Training Time

An important aspect is to do comparative analysis of the training performance of the TensorFlow SSD model on a CPU and GPU. For analysis, the CPU on which training was performed was an Intel i5 (8 GB RAM, 2.9 GHz). The GPU on which the model was trained was an NVIDIA GTX 970M. The Figure 2 clearly indicates that at a training step of approximately 3,500 on the above CPU, the time taken to reach the above point was around 20 hours and 5 minutes.

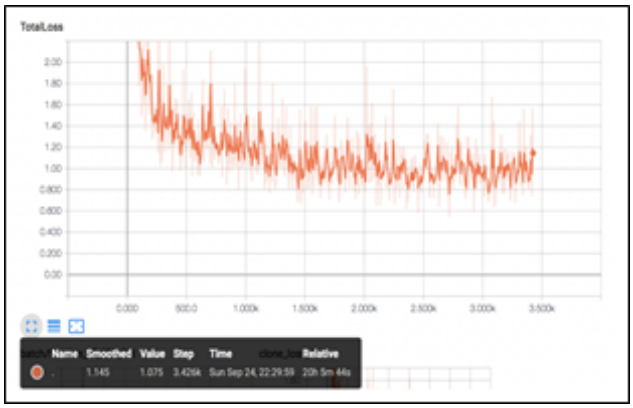

Figure 2. CPU Training Graph

However, the same training process for a checkpoint of close to 2500 steps was observed when the model was trained with a GPU. As can be observed from Figure 3, we obtained a remarkable speedup, with the checkpoint being reached in about 1 hour and 20 minutes. This speedup is a welcome result for larger datasets.

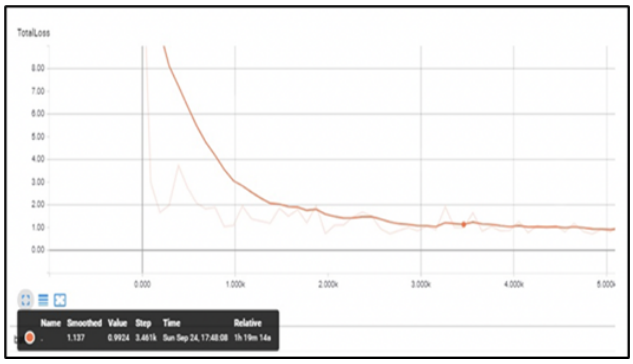

Figure 3. GPU Training Graph

B. Weights and Biases

For training visualization, weights and biases are sought out for the layers in the model as training progresses. The visualization is shown below.

The BW graphs stand for biases and weights. The Figures 4, 5, 6, 7, 8 are the biases and weights for the 5 fully connected layers of the neural network. The Box Predictor number indicates the layer number of the fully connected layers of the neural network. Each of the graphs is basically a histogram. The x axis indicates the value of the bias in the bias graph and value of weights in the weights graph. The y axis indicates the distribution of the values, that is, the number of instances which have

the specific value of weight/bias. The z axis indicates the training step at which the data is recorded.

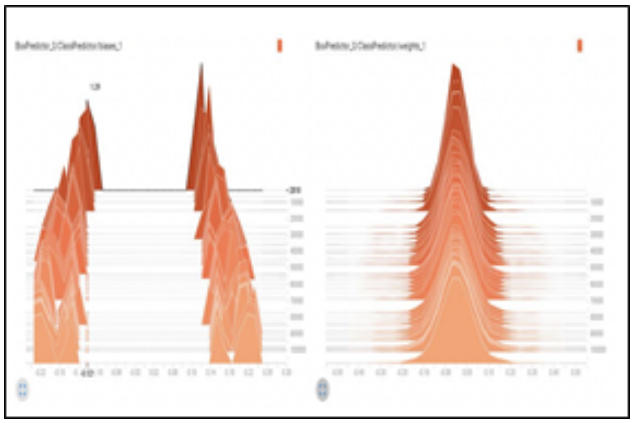

Figure 4. Bias and Weight for BoxPredictor0

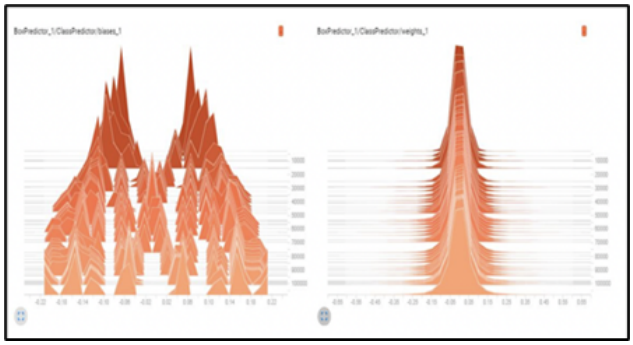

Figure 5. Bias and Weight for BoxPredictor1

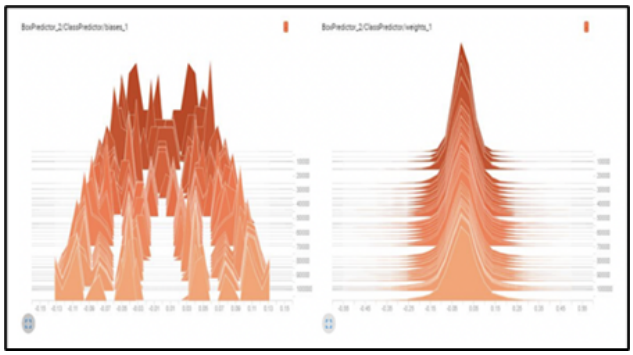

Figure 6. Bias and Weight for BoxPredictor2

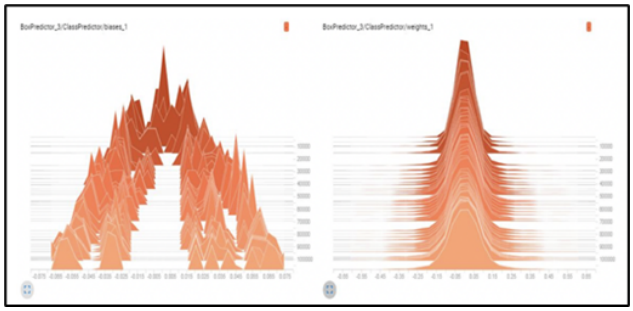

Figure 7. Bias and Weight for BoxPredictor3

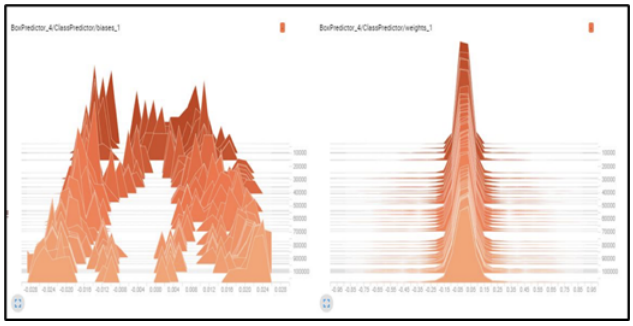

Figure 8. Bias and Weight for BoxPredictor4

C. Training Checkpoint Analysis

As indicated by the experimental setup above, the training was accomplished on the plane dataset for about 100,000 steps until the model converged to a loss less than 1.0 on an average. For the SSD Model, the various checkpoints obtained above were tested to verify the best model as the iterations progress. For analysis, the models were tested on the test set as well as videos of planes from the open-source YouTube-8M video data set maintained by Google. The criteria for analysis were testing the number of false positives and false negatives in each video which consisted of multiple planes and a video which had no planes and consisted of people and other vehicles. The length of the videos was about 50 seconds each.

Clearly as the total loss in training decreased with the increase in number steps until the point of convergence, which was around 41117 steps, the model performed well with a very small number of false positives and false negatives. Beyond 41117 the model started producing full frame errors with almost no significant recognition. Table I summarizes all our findings with respect to checkpoint analysis.

D. Inception vs. SSD

Table II compares SSD and the Inception model. It brings about the pros and cons of each model and their typical use cases.

Table III gives a brief comparison of R-CNN, R-FCN and the SSD during Real-Time Usage, the results of which

TABLE I. CHECKPOINT ATTRIBUTES

Model Name	False Negative Count	False Positive Count	Model Performance
.ckpt-22222	10	1	Almost no detection
.ckpt-32000	2	7	Few Boxes were obtained accurately, but confidence was lower
.ckpt-41117	3	1	False positives almost eliminated, and heavy reduction in false negative count
.ckpt-72542	2	10	Detection occurred in a few frames but with full frame errors cropping up
.ckpt (more than 100000 steps)	1	12	Abundance of full frame errors. Model cannot be used at all.

TABLE II. INCEPTION VS. SSD – A COMPARISON

Inception	SSD
No Bounding Boxes	Bounding Boxes
Trains only last layer of model	Trains all layers of Model
Very Low Training Time	High Training Time
Can be Trained easily with CPU	CPU training is extremely slow and not recommended. GPU is a must.
Higher Accuracy levels	Accuracy is good but lower than Inception for not very large datasets
Very Good Classifier. Its specialty is to classify.	Specialty is to identify specific objects within a frame and location of objects in the frame.
Cannot find object position in the frame	Can find the objects specific position in the frame

are obtained from a paper [16] by Google Research.

E. CPU vs. GPU Comparison

The comparison of the CPU and GPU that has been used in terms of cost of the device themselves, operational costs and the performance-power metrics for these experiments are illustrated in Table IV.

F. Sample of Obtained Results

Figure 9 shows the identification of a fighter jet in a non-uniform background. The SSD model was run on this frame to obtain the bounding box around the object of interest. The airplane was detected quite accurately with a confidence level of 99%.

Figure 10 shows another application of the SSD model on a frame with a light, uniform background. As with the previous case, the airplane in the frame was identified accurately, with a confidence of 97%.

Figure 11 shows the application of the SSD model on a frame where multiple objects of interest (airplanes) are present in a single frame. The model accurately annotates the airplanes in the frame, with varying confidence levels.

TABLE III. FASTER R-CNN VS. R-FCN VS. SSD (REAL-TIME USAGE COMPARISON)

Factors	Faster R-CNN	R-FCN	SSD
Mean Average Precision (mAP)	Lower than SSD for real-time processing	Lower than SSD for real-time processing	Has the highest mAP amongst the three for real-time processing
Memory Usage	Highest Memory Utilization	Memory Utilization factor in-between the other two models	Lowest memory utilization amongst the three
Accuracy vs GPU Time	Takes the longest time for object recognition on a GPU but most accurate	Takes moderate amount of time for object recognition on a GPU and almost as accurate as SSD	Takes the least time for object recognition on a GPU with good accuracy but accuracy is lesser than that of Faster-CNN

TABLE IV. CPU VS. GPU: COST AND PERF-POWER METRICS)

Factors	CPU Intel i5 8 GB RAM 2.9 GHz	GPU NVIDIA GTX 970M
Cost Price	Approximately 200$	Approximately 400$
Power Usage Cost For Processor Considering Each Proper Training Run	Run time: 1205 mins TDP: 65W Cost per kWh: 0.077$ Total cost: 0.1$ Run time: 80 mins	TDP: 75W Cost per kWh: 0.077$ Total cost: 0.01$
GFLOPS (Single Precision)	71.68 @ TDP (1.102 GFLOPS/watt)	2365 @ TDP (31.53 GFLOPS/watt)
GFLOPS (Double Precision) 31.35 @ TDP	(0.482 GFLOPS/watt) 73.9 @ TDP	(0.985 GLOPS/watt)

Figure 12 shows the SSD model in its working for an instance of airplanes where the angle of viewing the plane is directly below it. The frame is also such that the actual airplane is blurred. Even in this case, the airplane was identified. However, the confidence dropped to 90%.

Figure 13 shows the working of the Inception model on the frame of a video. The Inception model, being a light-weight model, ran much faster than the SSD model. However, no bounding boxes were obtained in this case. Rather, a higher confidence value was obtained.

5. Conclusion

In conclusion, the SSD and Inception models were successful in detecting planes in a video with minimal lag and high accuracy. The Inception model is suitable to extract frames in which a specific object is present if the position of the object is not of importance. The SSD model is suitable if the specific object needs to be detected and recognized with its position in a video frame. Also, for the use case, the requirement of a GPU for training the model is important as it was clearly observed that the training efficiency with a GPU was higher. This model can

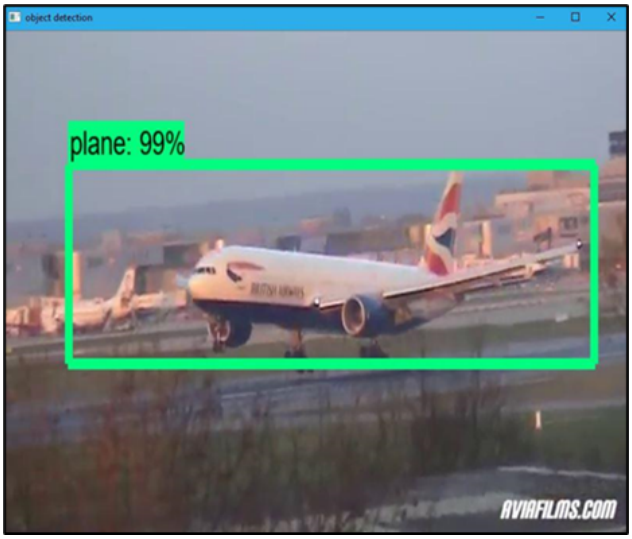

Figure 9. Jet Identified by SSD Model: Case 1

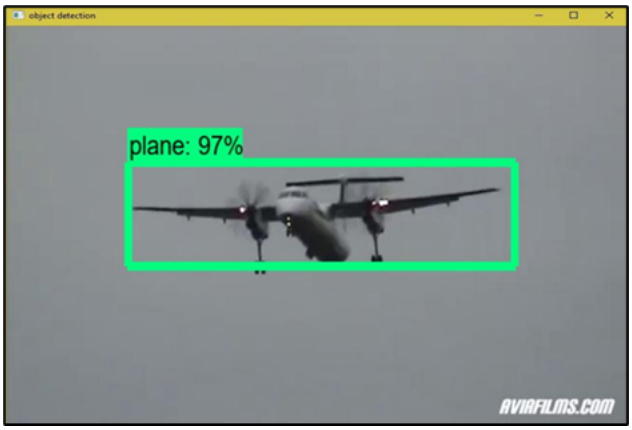

Figure 10. Airplane Identified by SSD Model: Case 2

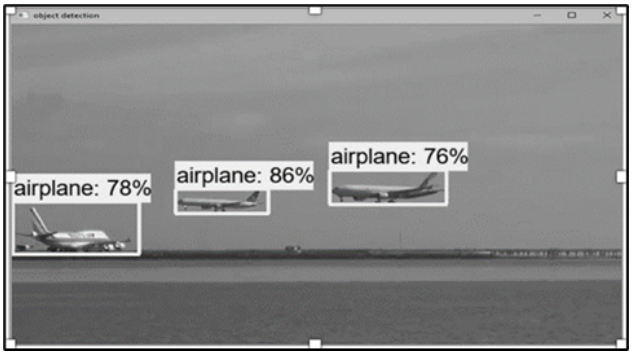

Figure 11. Multiple Airplanes Identified by SSD Model: Case 3

Figure 12. Airplane Identified by SSD Model: Case 4

Figure 13. Fighter Jet Identified by Inception Model: Case 5

track the object even if they come to a standstill suddenly (shortcoming of Temporal Differencing), and it works even if there is an excessive movement (shortcoming of Median Flow Tracker). The higher the occlusion, the lower was the detection accuracy. The future work involves improving accuracy and detection for occluded objects by using a larger and more diverse dataset and detection of war-tanks in a war-field since it is a challenging Computer Vision problem, due to the existence of camouflaging techniques.

References

[1] N. Kumaran and U. S. Reddy, "Object detection and tracking in crowd environment — a review," in *2017 International Conference on Inventive Computing and Informatics (ICICI)*, 2017, pp. 777–782.

[2] Y. Lin and B. Bhanu, "Evolutionary feature synthesis for object recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 2, pp. 156–171, 2005.

[3] N. Neelima, A. SriKrishna, and K. G. Rao, "Prototype analysis

of different object recognition techniques in image processing," in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*. IEEE, 2017, pp. 2882–2892.

[4] Q. Zhou, L. Ma, M. Celenk, and D. Chelberg, "Object detection and recognition via deformable illumination and deformable shape," in *2006 International Conference on Image Processing*. IEEE, 2006, pp. 2737–2740.

[5] A. Sharma, P. K. Singh, and P. Khurana, "Analytical review on object segmentation and recognition," in *2016 6th International Conference-Cloud System and Big Data Engineering (Confluence)*. IEEE, 2016, pp. 524–530.

[6] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*, 2016, pp. 265–283.

[7] A. Sobti, C. Arora, and M. Balakrishnan, "Object detection in real-time systems: Going beyond precision," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. IEEE, 2018, pp. 1020–1028.

[8] Z. Deng, L. Lei, H. Sun, H. Zou, S. Zhou, and J. Zhao, "An enhanced deep convolutional neural network for densely packed objects detection in remote sensing images," in *2017 International Workshop on Remote Sensing with Intelligent Processing (RSIP)*. IEEE, 2017, pp. 1–4.

[9] A. V. Raviteja and N. Sampath, "Detection and tracking of objects using thresholding technique," in *2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon)*. IEEE, 2017, pp. 810–814.

[10] H. Yanagisawa, T. Yamashita, and H. Watanabe, "A study on object detection method from manga images using cnn," in *2018 International Workshop on Advanced Image Technology (IWAIT)*. IEEE, 2018, pp. 1–4.

[11] Y. Yang, Y. Zhuang, F. Bi, H. Shi, and Y. Xie, "M-fcn: Effective fully convolutional network-based airplane detection framework," *IEEE Geoscience and Remote Sensing Letters*, vol. 14, no. 8, pp. 1293–1297, 2017.

[12] H. E. Kim, Y. Lee, H. Kim, and X. Cui, "Domain-specific data augmentation for on-road object detection based on a deep neural network," in *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2017, pp. 103–108.

[13] K. Wu, E. Wu, and G. Kreiman, "Learning scene gist with convolutional neural networks to improve object recognition," in *2018 52nd Annual Conference on Information Sciences and Systems (CISS)*. IEEE, 2018, pp. 1–6.

[14] J. E. Espinosa, S. A. Velastin, and J. W. Branch, "Motorcycle classification in urban scenarios using convolutional neural networks for feature extraction," in *8th International Conference of Pattern Recognition Systems (ICPRS 2017)*. IET, 2017, pp. 1–6.

[15] H. Schilling, D. Bulatov, R. Niessner, W. Middelmann, and U. Soergel, "Detection of vehicles in multisensor data via multibranch convolutional neural networks," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 11, pp. 4299–4316, 2018.

[16] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama *et al.*, "Speed/accuracy trade-offs for modern convolutional object detectors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7310–7311.

[17] S. Qi, X. Ning, G. Yang, L. Zhang, P. Long, W. Cai, and W. Li, "Review of multi-view 3d object recognition methods based on deep learning," *Displays*, p. 102053, 2021.

[18] N. Wang, Y. Wang, and M. J. Er, "Review on deep learning techniques for marine object recognition: Architectures and algorithms," *Control Engineering Practice*, p. 104458, 2020.

Ashwith Atluri Ashwith Atluri has completed his bachelor's in Computer Science and Engineering from R.V College of Engineering, Bangalore. He is currently working at Intuit as a full stack engineer. His research interests lie in the field of machine learning and data science.

Anurag Sethuram Anurag Sethuram received his Bachelor's in Computer Engineering from R. V. College of Engineering, Bangalore in 2018. Since then, he has been working as a System Software Engineer at Qualcomm India Private Limited. At Qualcomm, he has been involved in the architecture and development of CPU system software that currently is used to power millions of phones across the globe. His research interests include Operating System Security, Wireless Networks and Software Defined Networks.

Dr. Azra Nasreen Dr. Azra Nasreen completed her PhD from Visvesvaraya Technological University Belgaum and is working in the Department of Computer Science and Engineering of R. V. College of Engineering, Bangalore, India. Area of research interests include video analytics, high performance computing, and computer vision. She has published articles in reputed Scopus indexed journals and presented number of papers in various international conferences. She has completed and is involved in many consultancies and sponsored research projects from various organizations of international repute.

Dr. Shobha G. Dr Shobha G is a Professor in the Department of Computer Science and Engineering at RVCE Bangalore, India. She has worked on about 75+ publications in International Journals, presented about 72+ papers in international Conferences and has completed sponsored research projects of more than 10 million INR with national and international organization.

Dr. S. S. Iyengar Dr. S. S. Iyengar is a Distinguished University Professor and Founding Director of Discovery Lab at FIU. He served as the Director and was a Distinguished Ryder Professor of Computer Science at the Knight Foundation School of Computing and Information Sciences at Florida International University (FIU), Miami between 2011 and 2020. Prior to joining FIU, Dr. Iyengar was the Roy Paul Daniel's Distinguished Professor and Chairman of the Computer Science department for over 20 years at Louisiana State University.