Final Project Report

CPSC 8650

GAMESCAPING

Team Members:

Benafsh Husain
bhusain@clemson.edu

Kushal Hebbar
khebbar@g.clemson.edu

1. Introduction

Automated user feedback for videos in real time is fast becoming a possibility using the technology of deep learning algorithms and the computation power of GPUs. User feedback for videos can have several flavors, where a popular use case is automatic annotation of objects that appear on the screen. For example, in Figure 1 we observe frames with annotated boxes, identifying and classifying objects into their respective categories.

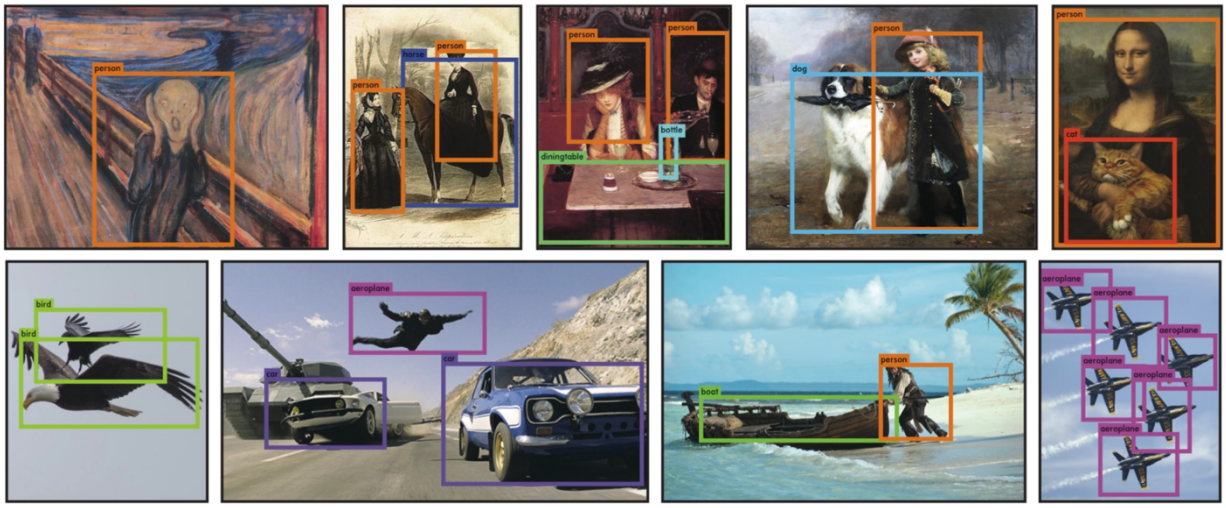


Figure 1 YOLO [1] running on sample artwork and natural images from the internet. It is mostly accurate although it does think one person is an airplane.

Based on the high accuracy and speed of object detection in still images, and in certain cases video feeds, researchers are attempting to further this technique to real-time video streams. An example of the above is utilizing traffic camera data/ car camera data to annotate objects such as persons, cars, traffic signs etc. This research thrust is the basis for several applications such as autonomous driving, real-time face detection and recognition, etc.

With the onset of advanced game design, development, and graphics, it can be observed that newer released games resemble closely to live action scenarios. In this project we leverage the advances made in fast, real-time object detection in videos to extend to games with graphics advanced enough to be recognized by standard object detection algorithms. We evaluate the efficacy of pre-trained real world image object detection algorithms on images from the game. We further extend the application of these object detection models during live gameplay and provide annotation and feedback to the user.

We utilize three pre-trained models using the COCO dataset [2], and apply the model to test the speed and accuracy of detection on the game GTA-V [3]. Throughout the project we have utilized the open source Tensorflow object detection api [4] to test both the still images, and perform annotation during live gameplay.

2. Background

2.1. Convolutional Neural Network: Object detection

The state of the art technique currently for object detection is the convolutional neural network (CNN). The CNN is a class of deep, feed forward artificial neural network, generally with fully connected hidden layers. CNN, as the name suggests typically consist of convolutional layers, pooling layers, fully connected layers and normalization layers as shown in Figure 2. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. The output of a CNN generally is a list of probabilities of class assignment. Within the class of CNN there are several algorithms that have been created for object detection. In this project we leverage two of the more common methods; Single Shot Detector (SSD) and Faster-RCNN

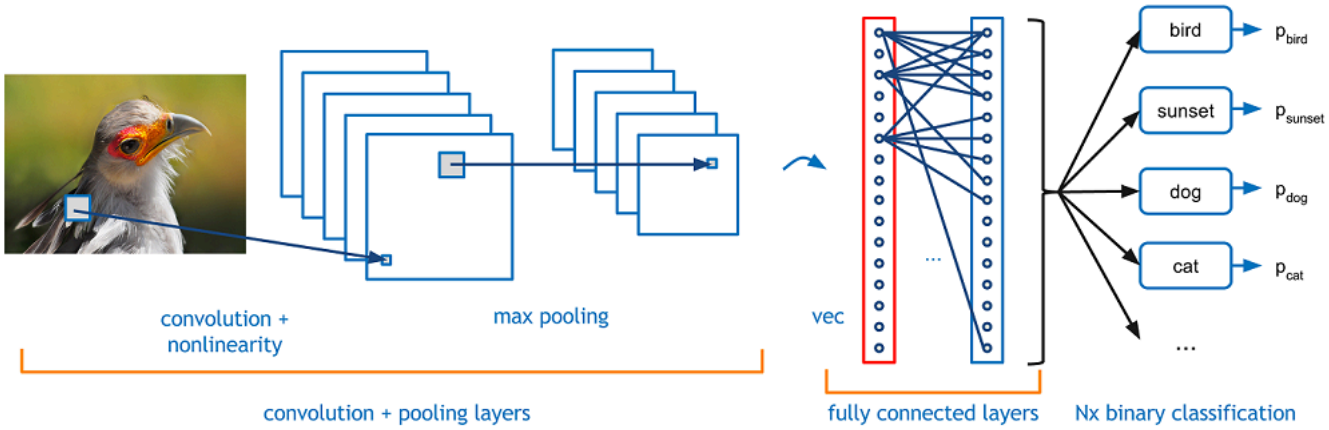


Figure 2 Example of Convolutional Neural Network representation

2.1.1 SSD: Single Shot MultiBox Detector

SSD [5] discretizes the output space of bounding boxes into a set of default boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object category in each default box and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps with different resolutions to naturally handle objects of various sizes. SSD is simple relative to methods that require object proposals because it completely eliminates proposal

generation and subsequent pixel or feature resampling stages and encapsulates all computation in a single network.

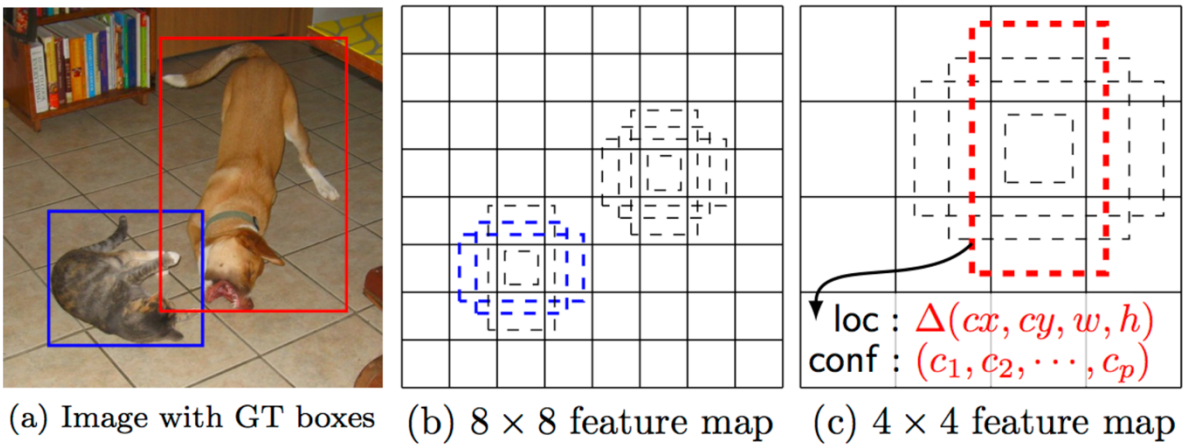


Figure 3 SSD Algorithm for CNN object detection

2.1.2 Faster- RCNN

Region Proposal Network (RPN) [6] that shares full-image convolutional features with the detection network enables nearly cost-free region proposals. An RPN is a fully convolutional network that simultaneously predicts object bounds and objectness scores at each position. The RPN is trained end-to-end to generate high-quality region proposals, which are used by Fast R-CNN for detection. Merging RPN and Fast R-CNN into a single network by sharing their convolutional features—using the recently popular terminology of neural networks with "attention" mechanisms, the RPN component tells the unified network where to look.

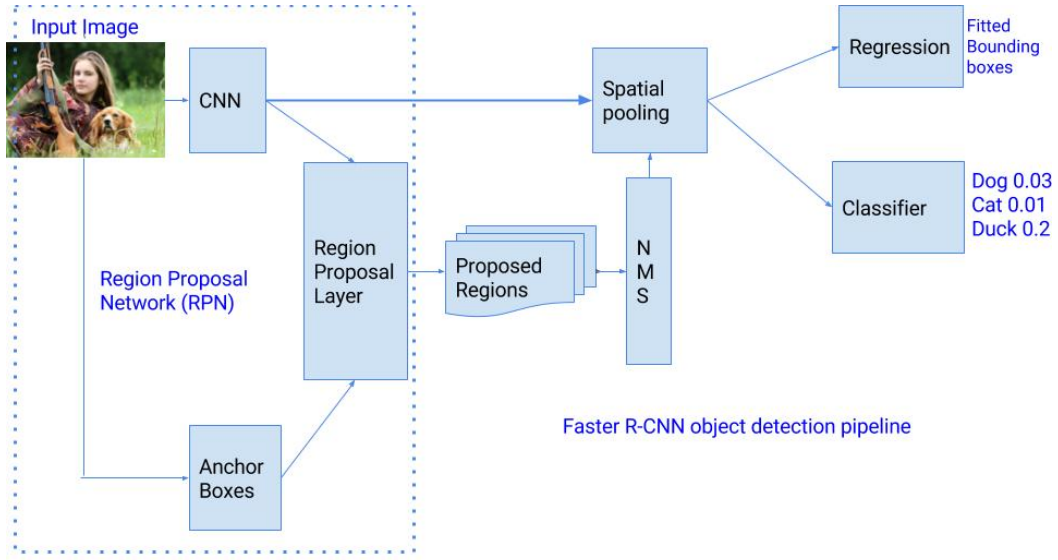


Figure 4 Faster-RCNN model for CNN object detection

3. COCO dataset

All the models we tested were trained using the COCO dataset [2] in order to stay consistent while evaluating the model accuracy. The dataset consists of images of complex everyday scenes containing common objects in their natural context. Objects are labeled using per-instance segmentations to aid in precise object localization. The dataset contains photos of *91 objects* types that would be easily recognizable by a 4 year old. With a total of *2.5 million labeled instances in 328k images*.

Figure 5 Sample images and categories from the COCO dataset

4. Architectures Utilized

4.1. Mobilenets

An efficient model class called MobileNets [7] was developed for mobile and embedded vision applications. MobileNets are based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks. The two global hyperparameters were introduced that trade-off between latency and accuracy. These hyper-parameters allow the model builder to choose the right sized model for their application based on the constraints of the problem.

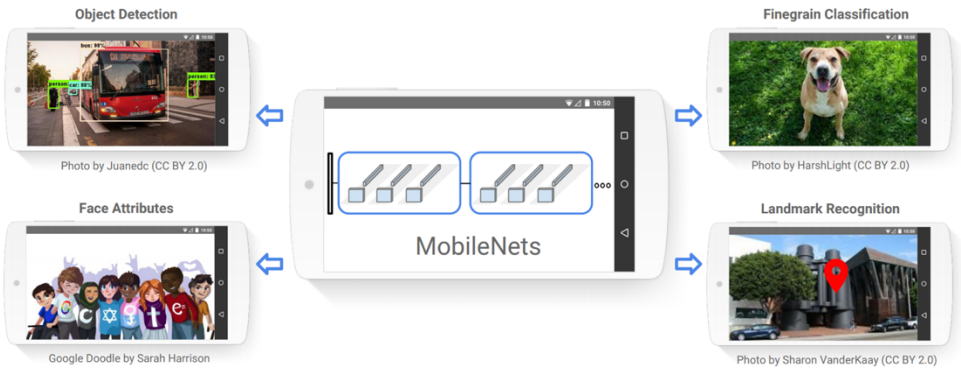


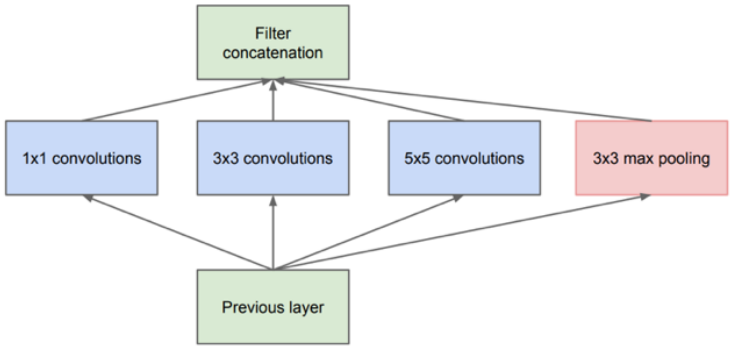
Figure 6 MobileNet model applied to various recognition tasks for on device intelligence

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

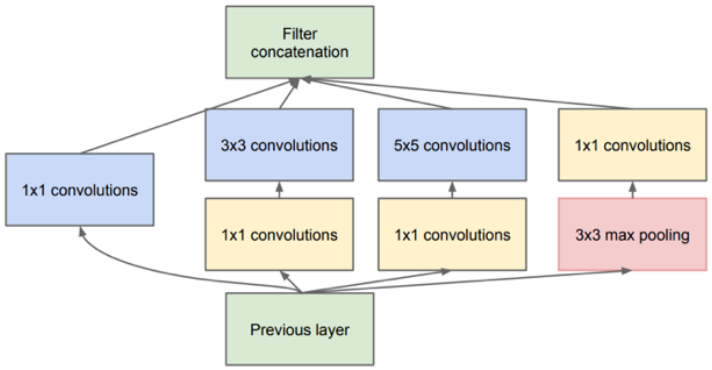
Figure 7 MobilNet Body Architecture

4.2. Inception (GoogLeNet)

GoogLeNet [8] is a 22 layer CNN and was the winner of ILSVRC 2014 with a top 5 error rate of 6.7%. It used Inception modules in the whole architecture, with over 100 layers in total. A salient feature was no use of fully connected layers. They use an average pool instead, to go from a 7x7x1024 volume to a 1x1x1024 volume. This saves a huge number of parameters. It also utilized 12x fewer parameters than AlexNet. During testing, multiple crops of the same image were created, fed into the network, and the softmax probabilities were averaged to give the final solution.



(a) Inception module, naïve version



(b) Inception module with dimensionality reduction

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 8 Inception Body Architecture

5. Models Utilized

For the purpose of this project we utilized three pre-trained on the COCO dataset using the open source tensorflow object detection api:

1) SSD using the mobilenet architecture
2) SSD using the inception architecture
3) Faster-RCNN using the inception architecture

We compared the three models using still images of the GTA-V game to specifically detect three categories of most prominent objects:

a) Humans
b) Cars/ vehicles
c) Traffic signals

The test set contained a combination of object with 50 humans, 50 vehicles, and 50 traffic signals. We recorded all the objects that were correctly and incorrectly identified. Along with recording the identified objects, we also recorded the percent probability of detection for every correctly identified object for the three models.

6. Results

6.1. Detection of Humans

Human detection in all three models seemed to have a high precision, that is low false positive rates observed, but the recall for SSD_Inception model was fairly low in comparison to the other two, indicating that several true object were not detected. Also, as shown in Figure 9 the average probabilities associated with correct detection was higher in F_RCNN_Inception model, and lowest in the SSD_Inception model.

	Precision	**Recall**
SSD_Mobilenet	**96.5%**	**89.1%**
SSD_Inception	**93.1%**	**79.3%**
F_RCNN_Inception	**97.3%**	**93%**

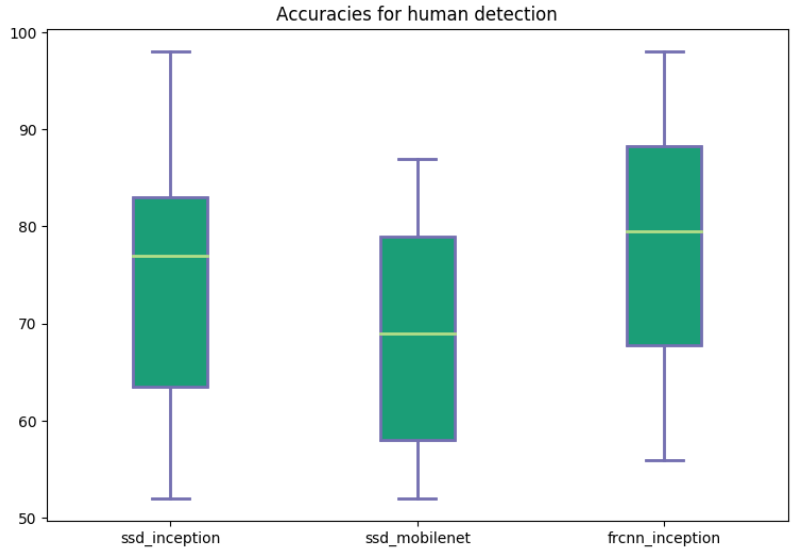


Figure 9 Boxplot representing accuracies for human detection

6.2.Detection of Cars

Detection of cars/ vehicles was overall much better supported in the SSD model, but as the table below indicates the recall for all models was low in comparison to human detection. F-RCNN_Inception model seemed to perform the worse in this case, specifically when trying to detect larger vehicles such as buses or trucks. Similarly, Figure 10 also corroborates lower probabilities of detected vehicles in F_RCNN_Inception model. Although, it is interesting to note that SSD_Inception model did not perform significantly better than F_RCNN_Inception.

	Precision	**Recall**
SSD_Mobilenet	**92.1%**	**88.6%**
SSD_Inception	**93%**	**79.3%**
F_RCNN_Inception	**91%**	**74%**

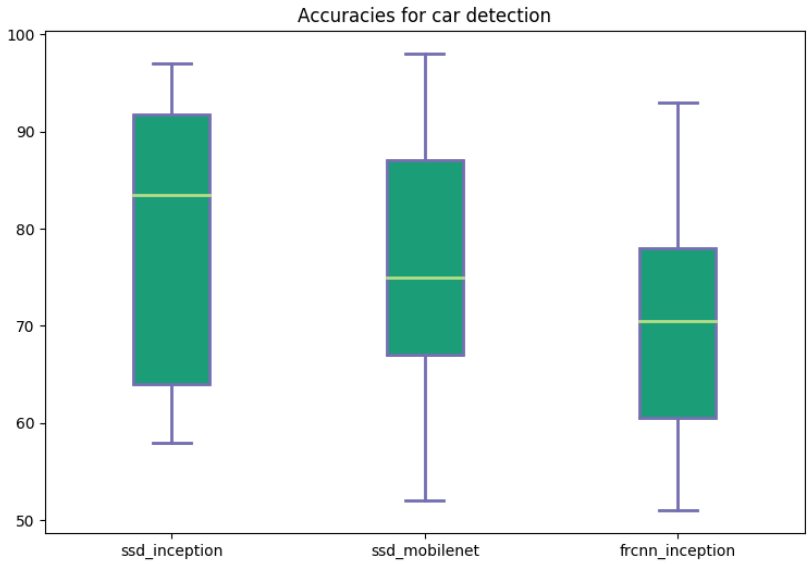


Figure 10 Boxplot representing accuracies for Car detection

6.3.Detection of traffic Signals

Detection of traffic signal was both the trickiest and also the most determinate in selecting the model most suited for GTA-V. The precision for all models was fairly high, as it seemed very rarely would incorrect object be classified as traffic signals, but as the table below indicates the recall for the 3 models varied significantly. F_RCNN_Inception far outperformed the other two at detection rate, along with detection probabilities as shown in Figure 11.

	Precision	**Recall**
SSD_Mobilenet	**99%**	**81%**
SSD_Inception	**99%**	**72%**
F_RCNN_Inception	**99%**	**96.2%**

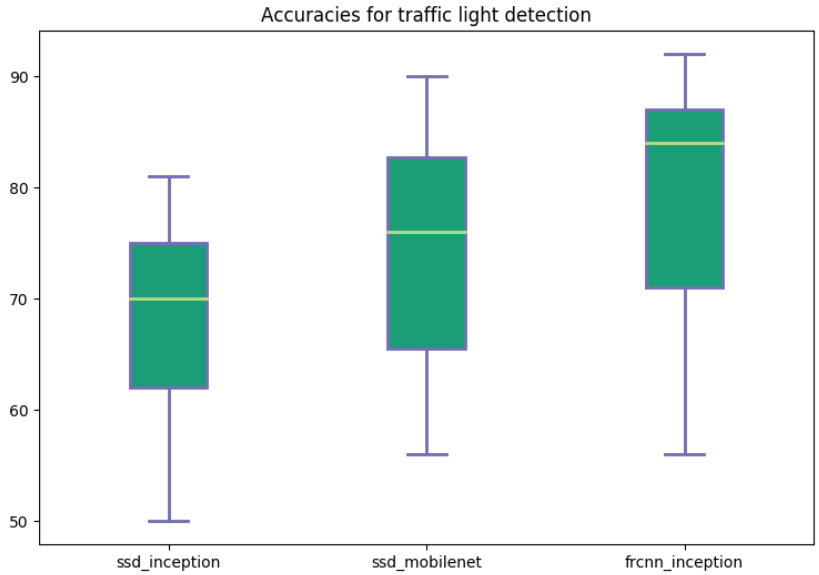


Figure 11 Boxplot representing accuracies for Traffic signal detection

Based on the comparison of the three models for human, car, and traffic signal detection on still images it can be concluded that SSD_mobilenet did not perform as well as the other two models, but SSD_inception and F_RCNN_Inception were close enough in performance to not make a conclusive decision. We therefore extended the experimentation of all three models on active gameplay.

7. Active Game Play

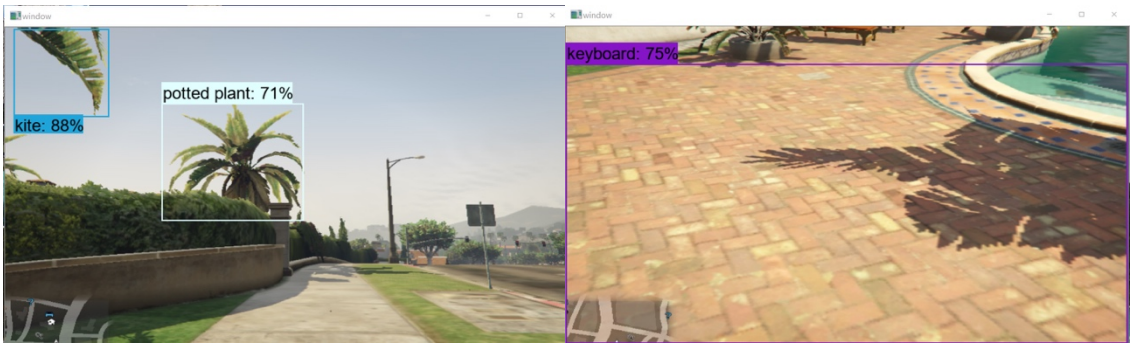
Upon evaluating the three models on still images, we extended the testing for active gameplay. Calculating accuracies and speed during live gameplay for the three models was difficult due to the limitation of the low available frame rate without a GPU. To run the testing model algorithm for active gameplay we used the tensorflow object detection api and the screengrab function [9].

For active gameplay we did not limit detection only to human, cars, and traffic signals, since the models were trained on the COCO dataset that contained 91 labels, we explored the space of object detection in the world of the game. Below we present some of the correctly and incorrectly classified objects for each model.

7.1 SSD_Inception

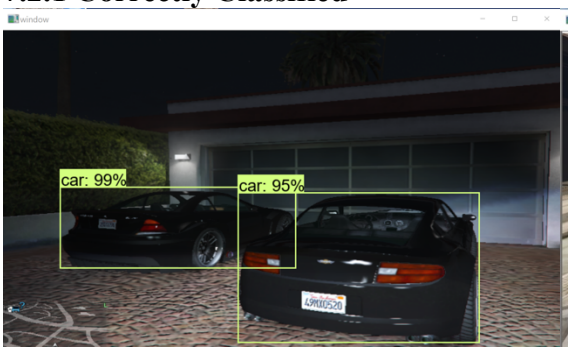
7.1.1 Correctly classified

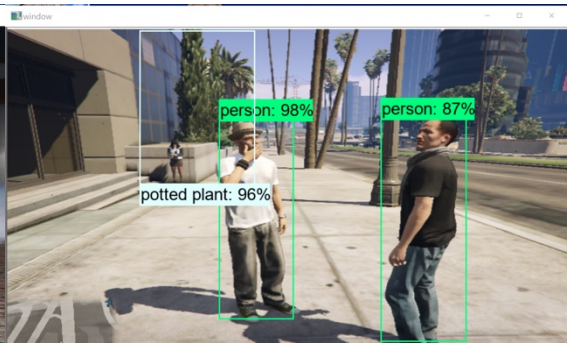
7.1.2 Incorrectly classified



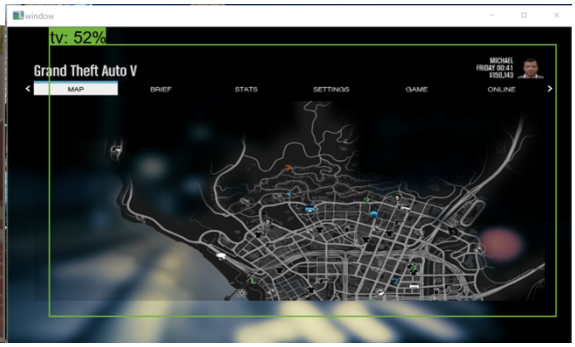
7.2 SSD_Mobilenet

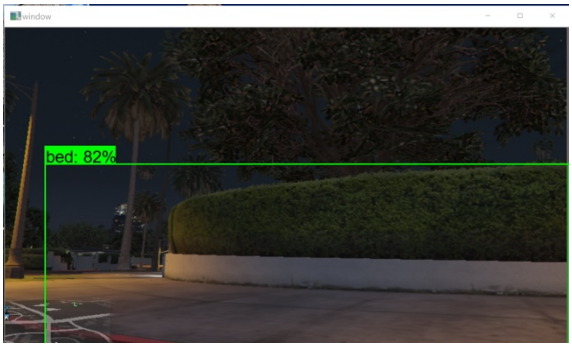
7.2.1 Correctly Classified





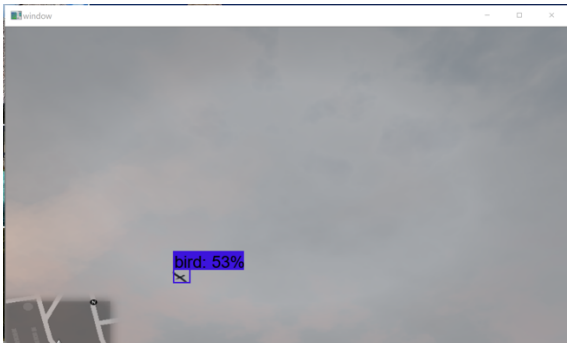
7.2.2 Incorrectly Classified

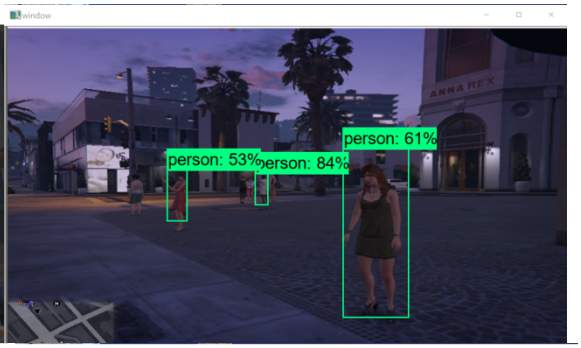


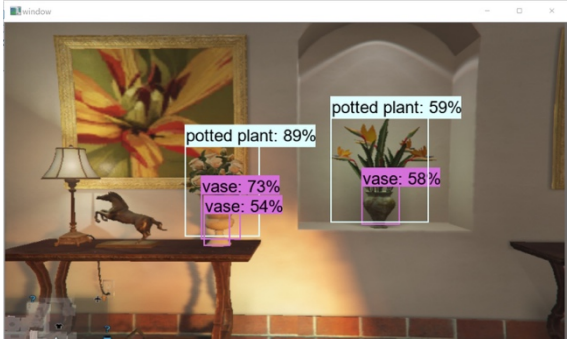


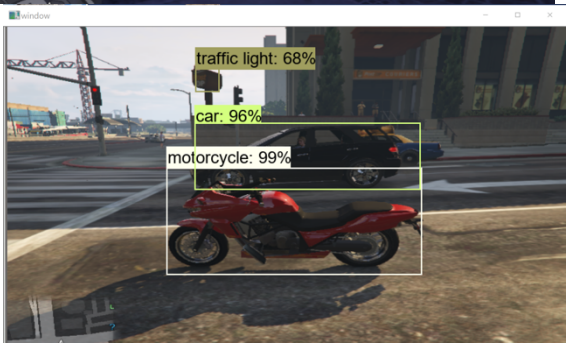
7.3 F_RCNN_Inception

7.3.1 Correctly classified









8. Conclusion

Overall, It was observed that during active gameplay the model F_RCNN_Inception outperformed the other two models. Primarily as it maintained the best balance of precision and recall. The other two models, specifically SSD_Inception in order to maintain a high recall would significantly misclassify objects. Also, SSD_inception performed the worst on one of our core tests of traffic signal detection. It was difficult to obtain exact precision and recall values for all three models during live gameplay since the game generates random characters and objects and getting a consistent set of characters every iteration was not possible.

Based on this project, it can be concluded that even the worse performing model, SSD_Inception is able to correctly detect objects in a game environment with significantly high accuracies. This enables integration of robust real world object detection algorithms to games with advanced graphics easily. As observed, the user feedback during gameplay can be developed to detect and classify specific objects, such as cars, or predict event such as a collision course, or crash warning. Gameplay experience can be enhanced through user feedback based on real world trained data which seems to be easily translated to the game environment.

References

[1] Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi, You Only Look Once: Unified, Real- Time Object Detection, arXiv:1506.02640v5, June 2015. https://arxiv.org/pdf/1506.02640.pdf

[2] Tsung-Yi Lin et.al "Microsoft {COCO:} Common Objects in Context"CoRR, abs/1405.0312, http://cocodataset.org/

[3] https://www.rockstargames.com/V/

[4] "Speed/accuracy trade-offs for modern convolutional object detectors." Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S, Murphy K, CVPR 2017

[5] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, SSD: Single Shot MultiBox Detector, arXiv:1512.02325v5, December 2015. https://www.cs.unc.edu/~wliu/papers/ssd.pdf

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: towards real-time object detection with region proposal networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1 (NIPS'15), C. Cortes, D. D. Lee, M. Sugiyama, and R. Garnett (Eds.), Vol. 1. MIT Press, Cambridge, MA, USA, 91-99.

[7] Andrew G. Howard et. al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications, CoRR, abs/1704.04861, 2017, http://arxiv.org/abs/1704.04861]

[8] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In CVPR, 2015.

[9] https://stackoverflow.com/questions/14489013/simulate-python-keypresses-for-controlling-a-game