Damian Hupka
801091092
ECGR 4090 - Real-Time AI

Homework #1

Problem 1

Problem 1.1.

The YOLO.v5 model ran across each of the available sizes (s,m,l,x) on an image (batch size 1) on both Google Collab and a personal computer. Both options were chosen to further examine the differences in execution time between a cloud service with virtualized GPUs and a home system. Table 1 below shows the results of the measured execution times. Similarly, these results can be observed by running the "hw1p1.py" in the submission package and modifying the "results" line to run the model on the batch size 1 image (imgs1) or the intersection image (intersection).

	Small	**Medium**	**Large**	**Extra Large**
Google Collab	.64 s	1.17 s	2.14 s	3.51 s
Personal Desktop	.52 s	1.35 s	1.41 s	1.99 s

Table 1: YOLO.v5 Execution times

As examined above, the execution time increases as the size of the model increases. This correlation logically follows since each model size will be increasing in number of parameters and computational complexity directly. However, it is expected that by using an increasingly larger model, the accuracy of the inference will be similarly greater. This improved accuracy will also come at a cost of decreased FPS due to larger inference (execution) times.

Problem 1.2.

As in the table above it was observed that the execution time on a personal desktop was typically slightly less than that of Google Collab, so the same desktop was used in this problem. As in the last problem, the YOLO.v5 model ran across each of the available sizes; however, the input image as given on canvas was used for inference. Figure 1 below shows this input image. The results of Problem 1.2 can also be observed by running "hw1p1.py" in the submission package and passing in the intersection image into the model, but modifying the size of the model to examine varying results.

Damian Hupka
801091092
ECGR 4090 - Real-Time AI

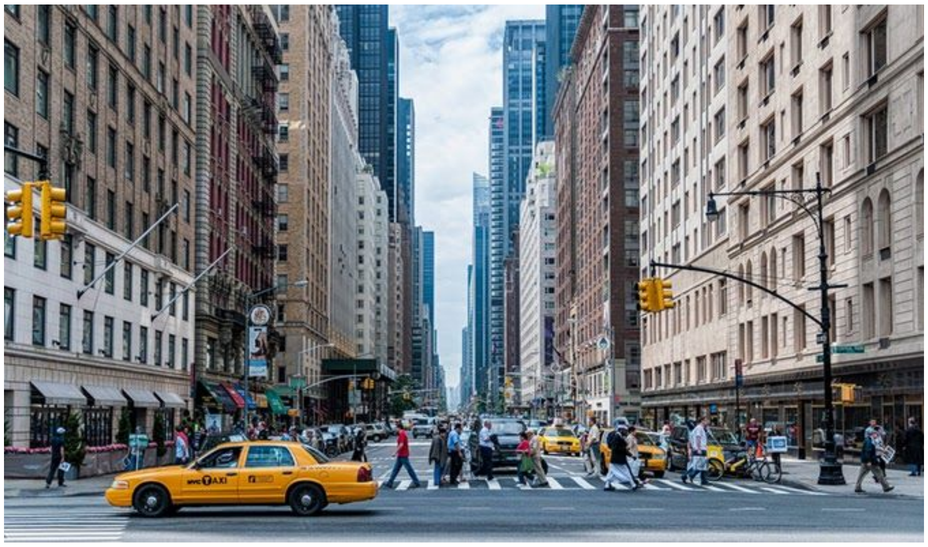
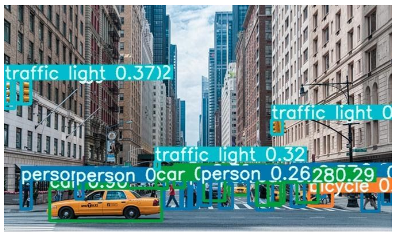
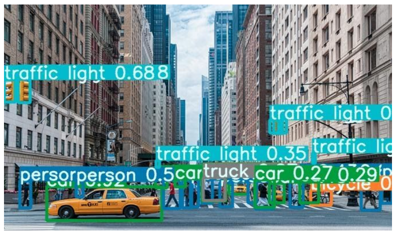
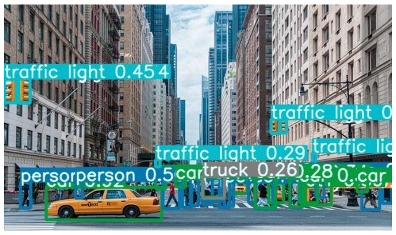


Figure 1: NYC Intersection

Furthermore, Table 2 below shows the output (bounding box, classification, and number of identifications)of the inference for each model size based on the input image in Figure 1.

Model Size:	**Number of classifications**	**Output Image**
Small	● 13 people ● 1 bicycle ● 8 cars ● 6 traffic lights	
Medium	● 12 people ● 1 bicycle ● 8 cars ● 1 truck ● 7 traffic lights	 traffic light 0.68 traffic light 0.35 truck car 0.27
Large	● 15 people ● 1 bicycle ● 11 cars ● 1 truck ● 7 traffic lights	

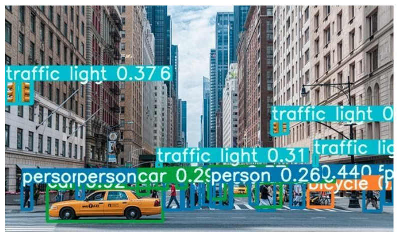
Extra Large	● 13 people ● 1 bicycle ● 7 cars ● 7 traffic lights	 traffic light 0.376 traffic light 0 traffic light 0.31 traffic lig persorpersoncar 0.2 person 0.26 0.440 bicycle

Table 2: Inference Results of YOLO.v5 Model Sizes

It is difficult to gain substantial insight from the results of varying YOLO.v5 model size on the given input image. This difficulty is largely based on the density of objects within the image. It is to be expected that the number of classifications, confidence, and validity of prediction for each increasing model size will similarly increase. Although this may be difficult to see in the output images and number of classifications in the results above, the larger the model size that was used lead to more accurate classifications on each object in the image, and decreased misclassifications of said objects. The greater accuracy in prediction with a larger model did have a detrimental effect on the execution time of the model which is expected as observed in Problem 1.1.

Problem 2

Problem 2.1.

Similar to problem 1, the pretrained YOLO.v5 was used, but in this case across varying batch sizes and using the NVIDIA Jetson Nano. The execution time of the YOLO.v5s model was measured on the board with batch sizes of 1, 8, and 16. These results are found below in Table 3 and Figure 2. The results of Problem 2.1 can also be examined by running the "hw1p2.py" file included in the submission package and modifying the batch size that is being passed into the model (imgs1, imgs8, imgs16).

Batch Size:	Execution Time (seconds)
1	1.79 s
8	65.35 s
16	149.70 s

Table 3: Execution time for Varying Batch Size Inference (Nano)

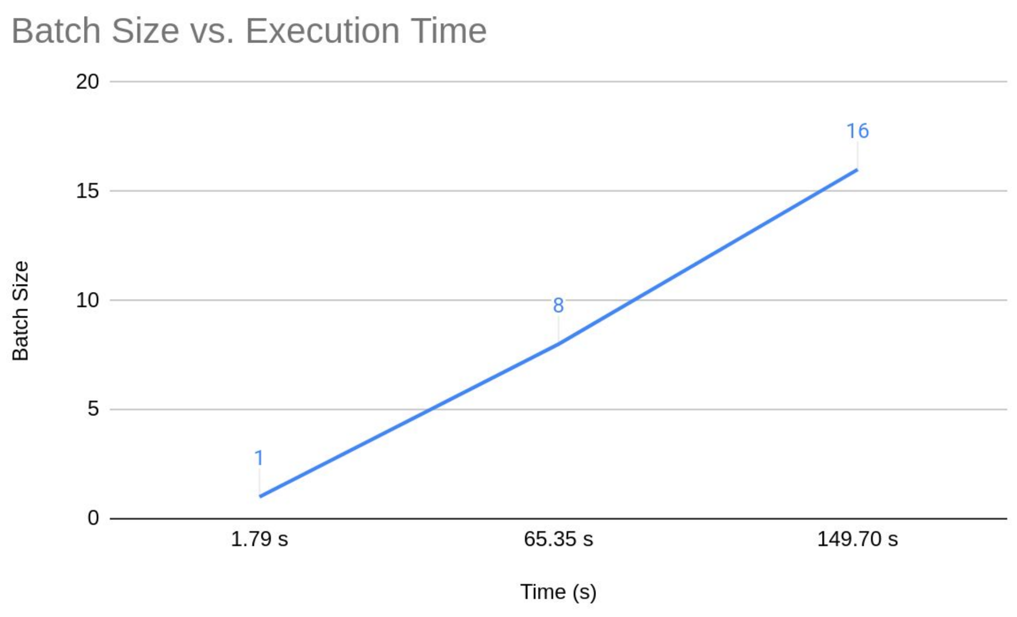


Figure 2: Execution Time Plot Across Batch Sizes

The results in the table and figure above demonstrate that as the batch size increases (1->16) the execution time will experience a significant increase as well. This relationship is correct as by having an input of a larger batch size for the model to conduct inference upon will now be many sets of inputs, to optimize many sets of weights within the model layers.

Problem 2.2.

Problem 2.1 was repeated, but across all YOLO.v5 model sizes (s, m, l, x) and the results for these respective execution times are below in Table 3.

Model Size:	**Batch Size:**	**Execution Time (seconds)**
Small	**1**	1.79 s
	8	65.35 s
	16	149.70 s
Medium	**1**	6.06 s
	8	131.63 s
	16	379.47 s
Large	**1**	6.21 s
	8	222.26 s

Damian Hupka
801091092
ECGR 4090 - Real-Time AI

	16	636.79 s
Extra-Large	**1**	18.60 s
	8	416.88 s
	16	1347.79 s

Table 2: Execution time for Varying Batch Size and Model Size Inference (Nano)

Similar to Problem 2.1, YOLO.v5 was run across varying batch sizes, but this time on each of the available YOLO.v5 model sizes. Again, there is a direct relationship between increasing the model size/batch size leading to an increased execution time. Interestingly, the execution time increase in using the largest model and largest batch size causes an exceptionally substantial increase in the execution time being approximately 1347 seconds (22 minutes). In running the "extra-large" model with 8 or 16 batch size the Jetson Nano was nearly depleted of cached memory, but the model was able to successfully complete, surprisingly. However, it is very likely if the batch size was increased to be any greater, the inference would not be completed as the Jetson would be out of available memory for computation.

Problem 3

Problem 3.1.

Lastly, the NVIDIA Jetson Nano was used to run real-time inference using the YOLO.v5s model. Initially, this was done using the default camera option provided by the YOLO.v5s directory using the "detect.py" framework and is included in the "HW1/yolov5" directory in the submission package.. Adapted from https://github.com/ultralytics/yolov5. In this template, the inference time (Ending time per inference frame - Start time per inference frame) is being measured. In order to capture and display the FPS, the calculation is as follows $\frac{1}{inference\ time}$ and this value was then displayed to the terminal. Typically, when running the YOLO.v5s model on the NVIDIA Jetson Nano the average FPS was approximately 5. Out of curiosity, the model was run on the same PC as used in Problem 1.1 and averaged an outstanding 90 FPS on inference. This is certainly a product of the PC having a much more substantial CUDA enabled GPU compared to the GPU cores on the NVIDIA Jetson Nano. A greater understanding of "detect.py" is necessary to implement the FPS measurement in such a way to display embedded on the live camera inference video. A demonstration of the YOLO.v5 real-time inference is here: **Youtube Link.**

Problem 3.2.

Jetcam was not utilized in implementation due to difficulty in versioning and enabling jetcam to recognize USB camera. Attempts were made to modify jetcam source and rebuild setup.py with python3 to enable functionality, but this fix did not work. Thus, the camera was read directly using the "detect.py" framework provided by YOLO.v5 code was adapted from such https://github.com/ultralytics/yolov5 greater details of implementation are discussed above in section 3.1.