Saving the Geese in London Ontario

Justin Chuang
Department of Electrical and Computer Engineering
Western University
Jchuang7@uwo.ca

Theodoros Kassa Aragie
Department of Electrical and Computer Engineering
Western University
taragie@uwo.ca

Ahmad Abu Shawarib
Department of Electrical and Computer Engineering
Western University
aabusha3@uwo.ca

Gabriel Olivotto
Department of Electrical and Computer Engineering
Western University
golivott@uwo.ca

Abstract — An abundance of geese have been showing up on London roads, and car traffic threatens the protected animals. Without recognizing the areas where the geese tend to idle, there is a higher risk of harming the geese. We propose an image detection model that can be used to detect geese from an active camera feed to inform the active presence of the geese and to prevent harm to them in traffic-dense areas. Vision transformers seem to provide state-of-the-art performance on image detection problems, however, they are very computationally heavy. Therefore, we plan to use convolutional neural networks to create a model that can run off of systems with lower computational prowess and use training images of geese to train the model.

I. INTRODUCTION

If you visited the Western University campus, there is a good chance you encountered a group of geese crossing roads causing a traffic holdup. This is because the Canadian goose is classified as a protected animal, and with their abundance, the need to identify large groups becomes necessary for their protection. The geese will frequently accumulate on streets, in and around campus, and haphazardly cross across roads, posing a risk to drivers and the protected species.

Object detection is a crucial task in the field of computer vision. It involves identifying instances of specific target classes of various objects, such as humans, animals, cars, or buildings, within digital imagery like photographs or videos[1].

This project intends to identify Canadian geese from video frames using a Canadian geese training set to train the YoloV8 model. While obstructions may occur, using multiple video frames allows for a larger margin of time to identify geese. This project aims to provide the university or other researchers with a possible tool they could use to monitor the activity of Canadian geese in urban settings.

II. RELATED WORKS

Object detection has been a very active field of research for a while now with its capabilities having been improved significantly due to the introduction of various deep-learning techniques in the past decade. In addition to researchers, many industries have begun utilizing object detection in many different applications ranging from heatmap and checkout traffic for retail stores, to road sign and pedestrian identification in autonomous driving applications, to medical feature detection in the healthcare specialization.

In recent years, much progress has been made in the real-time object detection field of study and many models have been developed that can perform inference on live video frames. One such study from Reis et al. used the YoloV8 model to identify flying aircraft in real-time, their approach was able to

achieve an average inference speed on 1080p videos of 50 frames per second [2].

Another research paper from Fernández-Carrión et al. used object detection to identify symptoms of swine flu in a herd of pigs based on changes in the pigs' mobility over time. His research shows that object detection can effectively monitor animals in an automated and non-intrusive fashion over long periods [3].

III. METHODS

A. Dataset Creation (Objective 1)

Before training a model to perform geese detection, we first had to construct a dataset. This dataset includes a large collection of geese images and the bounding boxes the detection model would use for training. Due to the novelty of this application, there are no existing publicly available datasets that contain all the required information, necessitating the development of a new one. This objective is significant as it lays the foundation for the research and has practical implications for similar future projects.

a) Data Collection: A collection of 1800 geese images was sourced from images.cv [4]. This platform was chosen due to its extensive collection of images for various computer vision tasks. The images were all resized to be 256 by 256 pixels. The size was chosen as it preserved enough detail for identifying geese while keeping the size of the dataset manageable. This decision was also influenced by the desire to mitigate the need for powerful hardware during training.

b) Sorting Data: The geese images collected contained various species of geese, but our application only required images containing Canadian geese. As such, the collected images had to be manually sorted. Due to a lack of manpower, we only sorted the first 1000 images from the dataset. The criteria for selecting images was if the image contained a Canadian goose, it should be included in the final dataset. This resulted in some images containing other types of birds. We hoped that this would help the model to better distinguish between geese and other types of birds. After the data was sorted, the final number of images was whittled down to 440 images.

c) Annotating Data: The 440 images of Canadian geese that remained after the sorting needed to be annotated by drawing bounding boxes around all the geese in each image. To complete this task, we used cvat.ai, a free web-based computer vision annotation tool [5]. This tool was chosen for its user-friendly interface and robust features. When drawing bounding boxes, care was given to ensure the bounding boxes closely matched the visible part of the goose. This had to be done since geese are typically seen in flocks huddled closely together, which can result in many geese being partially obscured by other geese.

d) Splitting the dataset: The final data was randomly split into 360 images for training, 40 images for validation, and 40 images for testing. The split prioritized training images since we wanted to give our model most of the data to train on and our dataset is relatively small. However, we still wanted to set aside some images for validation and testing so that we could collect metrics on how well the model can identify the geese. This split ratio, 8:1:1, was chosen based on common practices in machine learning and considering the size of our dataset.

B. Model Creation (Objective 2)

After curating and preparing our dataset, the next critical step is to design and train a model that can accurately detect Canadian geese in images. This involves selecting an appropriate model architecture, configuring its parameters, and then training it using our dataset. The model's performance is then evaluated and optimized through several iterations of training and validation. This objective is significant as the effectiveness of our application hinges on the accuracy and robustness of this model.

a) Model Selection: After researching various object detection models we decided to use YOLOv8 [6]. YOLOv8 is a state-of-the-art object detection model that is known for its efficiency and accuracy. YOLOv8 is particularly suitable for real-time object detection. It treats object detection as a regression problem, which makes it faster than other models that treat object detection as a classification problem.

Firstly, YOLOv8 offers various sizes of the model, allowing us to tailor the computational power required to our available hardware. This flexibility makes it suitable for our hardware constraints. Secondly, it has been pre-trained on a large dataset (COCO dataset), which means it has already learned a variety of features from different objects. This pre-training can help in detecting geese, even though they were not explicitly part of the training set.

Lastly, YOLOv8 has a good balance between speed and accuracy, which is crucial for our application.

b) Training Methodology: The training was conducted using the training and validation sections of the dataset. After the training was complete, the best model throughout the training process was tested on the test set. This allowed us to gather metrics for comparing different model configurations.

We tested four different configurations to find the optimal setup for our specific use case:

1. YOLOv8 Nano with no data augmentation and default image size of 640 by 640 pixels: This configuration was chosen to evaluate the performance of the base model without any data augmentation techniques. The image size of 640 by 640 pixels was selected as it is the default size for YOLOv8 Nano.
2. YOLOv8 Nano with data augmentation and default image size of 640 by 640 pixels: Data augmentation techniques were introduced in this configuration to increase the diversity of our training data and potentially improve the model's ability to generalize. The augmentations applied included:
    - Hue, Saturation, and Value (HSV) Adjustments: These adjustments were made to help the model generalize across different lighting conditions.
    - Translation and Scaling: These techniques were used to aid in the detection of partially visible objects and objects at different distances.
    - Horizontal Flips, Image Cropping, and Mosaics: These methods were employed to adjust the composition of scenes, thereby increasing the variability of the training data.
3. YOLOv8 Nano with data augmentation and image size of 256 by 256 pixels: This configuration was similar to the previous augmentations but with a reduced image size. The aim was to investigate whether a smaller image size would affect the model's performance.
4. YOLOv8 Small with data augmentation and default image size of 640 by 640 pixels: This configuration was chosen to evaluate the performance of a larger YOLOv8 model with data augmentation.

c) Model Evaluation: In the model evaluation phase, we used several metrics to assess the performance of our model. These metrics provide different perspectives on the model's ability to detect objects accurately and efficiently.

1. Precision: This is the proportion of true positive predictions (geese correctly identified as geese) among all positive predictions (all detections identified as geese by the model). High precision means that the model made fewer false positive errors.
2. Recall: This is the proportion of true positive predictions among all actual positives (all actual geese in the images). High recall means that the model detected most of the geese present in the images.
3. Mean Average Precision (mAP): This is a popular metric in object detection tasks. It calculates the average precision (AP) for each class and then takes the mean over all classes. The AP is the area under the precision-recall curve for each class.
    - mAP50: This is the mAP calculated at an Intersection over the Union (IoU) threshold of 0.5. IoU is a measure of overlap between the predicted bounding box and the ground truth bounding box. An IoU of 0.5 means the predicted bounding box and the ground truth bounding box have 50% overlap.
    - mAP50-95: This is the mAP calculated by averaging the mAP values at IoU thresholds from 0.5 to 0.95 with a step size of 0.05. This gives a more comprehensive view of the model's performance across different levels of overlap.
4. Average Inference Time: This is the average time it takes for the model to process an image and make predictions. We tested this across various export formats on both the CPU and GPU. This metric is crucial for understanding the model's real-time effectiveness. If the inference time is too long, it may not be feasible to use the model for real-time geese detection in the field.

By evaluating these metrics, we can gain insights into the model's performance and make informed decisions about its deployment. For instance, if the model performs well on a CPU with acceptable inference times, it could potentially be deployed on relatively weak hardware in real-time scenarios. Conversely, if the model requires a GPU for efficient operation, the data might need to be accumulated and then processed on a more powerful machine.

C. Geese Counting Proof of Concept (Objective 3)

The final objective of our research is to develop a proof of concept for geese counting. This involves creating test programs that utilize our trained model to count the number of geese in an image or video. The process begins by feeding the image or video into our model. The model then identifies and counts the number of geese present. For video inputs, this process is repeated for each frame, providing a dynamic count of geese over time.

In addition to counting geese, our model provides another significant feature: it outputs the (x,y) coordinates of each detected goose in the image or video frame. This allows us to map the geese to specific locations within the frame. For instance, if we have a static camera monitoring a particular area, such as a road, we can define this area within the frame. By checking whether the coordinates of the detected geese fall within this defined area, we can identify when and where geese are accumulating.

This feature could be particularly useful for monitoring areas where the presence of geese could pose a risk, such as roads or airport runways. By providing real-time alerts when geese are detected in these defined areas, our model makes contributions towards efforts that prevent accidents and enhance safety.

This objective is significant as it demonstrates the practical application of our research. The ability to accurately count geese in different environments can provide valuable data for ecological studies and wildlife management groups. For instance, it can help track where geese accumulate, which informs the implementation of deterrent measures to keep geese away from dangerous areas. This proof of concept, therefore, has the potential to make a significant contribution to both wildlife research and safety practices regarding Canadian geese.

V. RESULTS

The various models described in III. METHODS, B. Model Creation, were trained for 500 epochs on an Ubuntu machine using a Nvidia Tesla K80. However, if model performance on the validation set did not improve after 100 epochs, training ended early.

A. Model Performance Metrics

After training the various models on the training and validation sets, the best instance of the model was tested on the test set to get an idea of their performance across various metrics on new unseen data. The results are tabulated in table 1.

TABLE 1: Performance metrics for models tested (**bold**: best score, underline: second best score)

Name	Data Augmented	Image Size (px)	BaseModel (Params)	Precision(%)	Recall (%)	mAP50(%)	mAP50-95(%)
No Augmentation		640	YOLOv8n (3.2M)	45.8	3.97	8.03	2.48
Augmentation	✓	640	YOLOv8n (3.2M)	87.0	**60.3**	68.4	45.0
imgz256	✓	256	YOLOv8n (3.2M)	80.3	51.6	58.8	32.8
Small	✓	640	YOLOv8s (11.2M)	**91.9**	58.7	**70.1**	**48.7**

a) YOLOv8 Nano with no data augmentation and a default image size of 640 by 640 pixels: This model served as our baseline and was crucial as it provided a reference point for comparing the effects of various parameter changes on the model's performance. Row 2 of Table 1 (referenced above) labelled "No augmentation", this row presents the metrics we recorded on the test set for the best-performing instance of our model. Unfortunately, this version of the model did not perform well across all categories. Figure 1 (referenced below) illustrates the bounding box and classification loss on both the training and validation sets for the non-augmented model. From this figure, it is evident that the lack of data augmentation techniques causes the model to overfit the training data set. Overfitting is a modelling error that occurs when a function is too closely aligned to a limited set of data points. More specifically, in Figure 1, we observe that the classification loss on the validation set starts to increase dramatically after approximately 80 epochs. An epoch is a complete pass through the entire training dataset. The best version of this model was obtained at epoch 83. However, beyond this point, the model began to overfit significantly to the training data set, indicating that it might not generalize well to new unseen data.

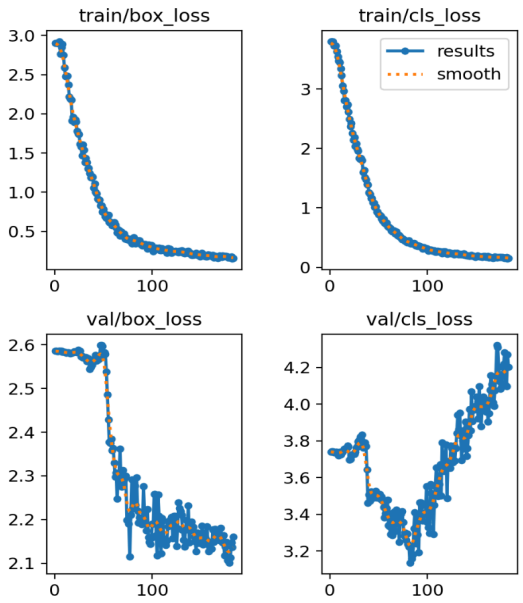

Fig. 1. Bounding box and classification loss on training and validation sets for the *No Augmentation* model

b) YOLOv8 Nano with data augmentation and default image size of 640 by 640 pixels: After applying the data augmentation techniques to the training set described in III. METHODS, B. Model Creation (Objective 2), b. Training Methodology, performance significantly improved. Row 3 of Table 1 (referenced above) labelled "Augmentation", this row presents the metrics we recorded on the test set for the best-performing instance of our model. This version performed significantly better than the baseline achieving a respectable 68.4% mAP50 score with good showings in other metrics as well. This can largely be attributed to the various data augmentation techniques randomly altering the input training data allowing the model to better generalize leading to better performance on new data. By looking at Figure 2 (referenced below), we can see this model was able to train for much longer and never showed signs of over-fitting to the training data. Loss on the validation set did seem to plateau around epoch 200 only improving marginally past that point.

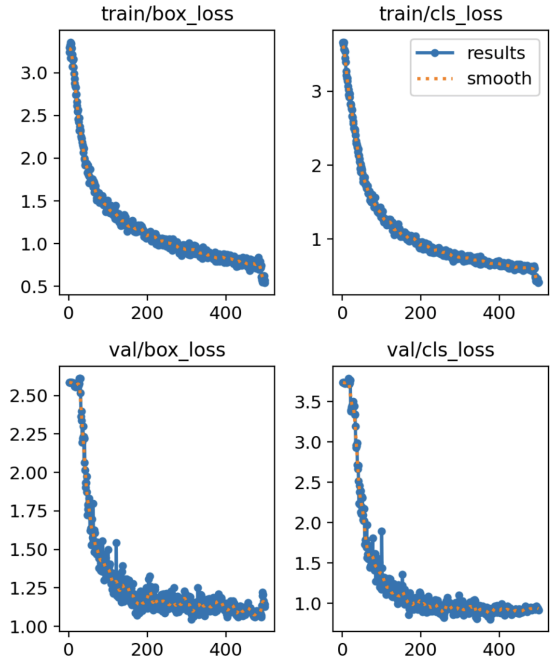

Fig. 2. Bounding box and classification loss on training and validation sets for the *Augmentation* model

c) YOLOv8 Nano with data augmentation and an image size of 256 by 256 pixels: After resizing the images from 640 by 640 pixels to 256 by 256 pixels as described in III. METHODS B. Model Creation (Objective 2) b. Training Methodology, there was a small yet noticeable decline in the performance metrics. Row 4 of Table 1 (referenced above) labelled "imgz256", this row presents the metrics we recorded on the test set for the best-performing instance of our model. Although this model performed significantly better than the baseline model, reaching 80.3% on its precision score and having decent scores all-round, it was marginally worse than the previous model "Augmentation" in all aspects. Since the only difference between the two models is the resizing of the training set's input images, this proved that downscaling the images, while negligible, was not a necessary step. The image resize step was considered due to a hypothesis that took into consideration the fact that the filtered images from the data collection step in III. METHODS A. Dataset Creation (Objective 1) a. Data Collection, were all 256 by 256 pixels. Moving forward, the images remained the default size of 640 by 640 pixels. Examining Figure 3 (referenced below), the "imgz256" model follows a similar pattern to the above "Augmentation" model but takes smaller varying steps and ends with bigger loss scores as a result. Interestingly, although the model found a local minima in the box and CLS loss, it found the true minima later on.

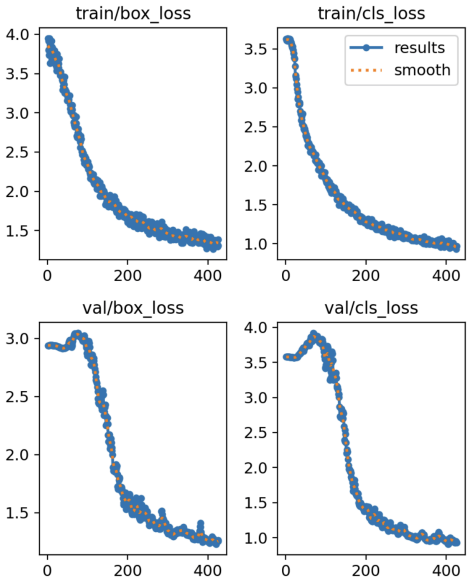

Fig. 3. Bounding box and classification loss on training and validation sets for the *imgz256* model

d) YOLOv8 Small with data augmentation and default image size of 640 by 640 pixels: Observing that the use of an image size of 256 by 256 pixels appeared to negatively impact the model's performance, we reverted to using an image size of 640 by 640 pixels. We then trained a version of the model using the YOLOv8 Small pre-trained model to assess the impact of a more complex model on our metrics. As shown in Row 5 of Table 1 (referenced above) labelled "Small", this row presents the metrics we recorded on the test set for the best-performing instance of this model. As expected, this model outperformed all other models tested across nearly all metrics. However, it's worth noting that despite the YOLOv8 Small model having nearly four times the number of parameters as the YOLOv8 Nano model, the performance improvement was marginal, only about 3 to 4 points. This suggests that for our specific task, the simpler Nano model is sufficient to identify most of the geese in the images, and the additional complexity of the Small model unnecessarily consumes considerably more computing resources and time. Figure 4 (referenced below) shows the loss on the training and validation sets. Interestingly, they are very similar to the plots for the Nano model with data augmentation seen in Figure 2 (referenced above).

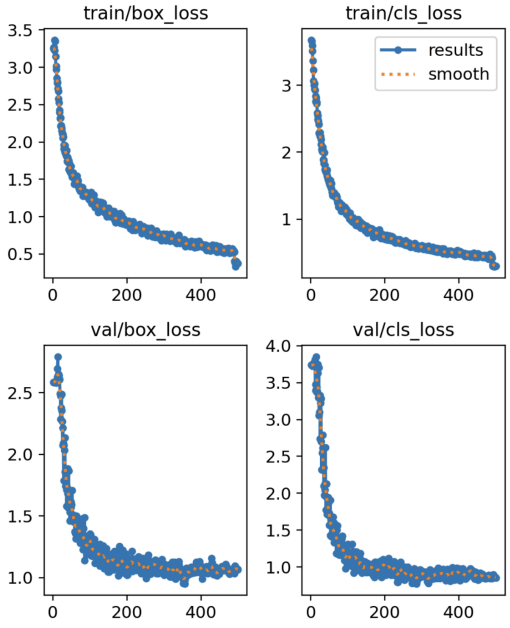

Fig. 4. Bounding box and classification loss on training and validation sets for the *small* mode

TABLE 2: Average inference time comparison for different models on the test set
(green highlight for results under 41.6 ms, yellow highlight for results over 41.6 ms, red highlight for results near double 41.6 ms)

Model				Inference time(ms/im)	
Name	Export Format	Size (MB)	mAP50-95 (%)	CPU (i7-9700k)	GPU (Tesla K80)
No Augmentation	PyTorch	6.0	2.48	285.05	21.22
	ONNX	11.7	3.47	81.67	72.58
	TensorFlow	29.2	3.47	55.55	51.89
Augmentation	PyTorch	6.0	45.01	571.14	20.72
	ONNX	11.7	40.91	83.44	70.99
	TensorFlow	29.2	40.91	53.95	47.21
imgz256	PyTorch	6.0	32.79	299.51	12.39
	ONNX	11.5	34.10	23.80	10.39
	TensorFlow	29.0	34.10	15.90	25.06
Small	PyTorch	21.5	48.71	1407.52	42.96
	ONNX	42.6	42.87	140.33	135.20
	TensorFlow	106.7	42.87	105.42	65.40

B. Inference Time Comparison

Table 2 compares the average inference time for different export formats on the test set. This information can give us an idea of the computational efficiency of the different export formats and provide some insight into how our model could be used to monitor geese's behaviour in the field. It is well known that using a GPU significantly speeds up deep learning applications and that is reflected in table 2. For reference, standard video formats export at 24 frames per second which is roughly equal to 41.6 ms per frame, so for our benchmark anything under this value can be considered "real-time." It is worth noting that nearly all GPU inference time met this threshold so with access to a sufficient computer GPU, real implementations of the model could reach real-time performance. However, any study looking into how geese accumulate would likely want to have many cameras set up to record enough area so that enough meaningful information can be recorded. One possible solution could be to use cheap microcomputers such as the Raspberry Pi instead to operate on the individual snapshots as fast as the low-power CPU could run.

C. Qualitative Analysis of Results

The results of this proof of concept application can be seen below in some real-world scenarios taken on the Western University Campuses. The predictions were sourced from the model that used YOLOv8 Small with data augmentation and a default image size of 640 by 640 pixels.

Fig. 5. Predictions using the *small* model for real-world scenarios

Overall, the model seems to work reasonably well. An interesting feature of the model can be seen on the bottom right of Figure 5 where the model did not identify the stylized picture of a goose in the sign. Testing the model's performance on frames from a video about geese by National Geographic[7] also yielded promising results and shows that even in its current state our model could be useful as a tool for wildlife researchers[1].

VI. CONCLUSIONS & FUTURE WORK

In conclusion, this research has demonstrated the feasibility and effectiveness of using the YOLOv8 model for geese detection and counting. Our experiments have shown that even with limited computational resources, it is possible to train a model that performs well on this task.

We found that data augmentation techniques and careful selection of model parameters played a crucial role in improving the model's performance. However, our results also indicated that increasing the complexity of the model did not necessarily lead to significant improvements in performance, suggesting that simpler models may be sufficient for this task.

The real-time geese detection system developed in this study has potential applications in ecological studies and wildlife management. By providing accurate counts of geese in different environments, the model can help track where geese accumulate and inform the implementation of deterrent measures to keep geese away from dangerous areas.

While our results are promising, there is still room for improvement. Future work could explore other model architectures, consider using larger datasets, or apply more advanced data augmentation techniques. Despite these potential areas for future exploration, the findings of this study represent a significant step forward in the application of deep learning techniques to wildlife detection and counting.

[1] Video Output of predictions: https://www.youtube.com/watch?v=nq4RMQWpcg0

REFERENCES

[1] G. Boesch, "Object Detection in 2021: The Definitive Guide," viso.ai, Jul. 09, 2021. https://viso.ai/deep-learning/object-detection/

[2] D. Reis, J. Kupec, J. Hong, and A. Daoudi, "Real-Time Flying Object Detection with YOLOv8," May 2023, doi: https://doi.org/10.48550/arxiv.2305.09972.

[3] E. Fernández-Carrión, M. Martínez-Avilés, B. Ivorra, B. Martínez-López, Á. M. Ramos, and J. M. Sánchez-Vizcaíno, "Motion-based video monitoring for early detection of livestock diseases: The case of African swine fever," PLOS ONE, vol. 12, no. 9, p. e0183793, Sep. 2017, doi: https://doi.org/10.1371/journal.pone.0183793.

[4] "Download Goose labeled image classification dataset labeled image dataset," images.cv. https://images.cv/dataset/goose-image-classification-dataset (accessed Apr. 08, 2024).

[5] "CVAT," www.cvat.ai. https://www.cvat.ai/

[6] G. Jocher, "YOLOv8 Documentation," docs.ultralytics.com, May 18, 2020. https://docs.ultralytics.com/

[7] "Geese Fly Together | National Geographic," www.youtube.com. https://www.youtube.com/watch?v=pdtpI33EXHQ