YOLO Modeling to Detect Uranium

Diego Ramon, Hector Arvizu, Myriam Boutros, Chanel Williams

I. Abstract

The YOLO model, renowned for its real-time object detection capabilities, has been adapted and fine-tuned to address the challenges posed by the detection of uranium, a material crucial to both peaceful and potential nefarious applications. The proposed model leverages convolutional neural networks (CNNs) to efficiently process high-dimensional data, allowing for swift and precise identification of uranium signatures within various imaging modalities.

Our methodology involves the collection of diverse datasets encompassing different scenarios, including varying lighting conditions, backgrounds, and shieldings. The YOLO model is trained on these datasets, employing transfer learning to enhance its adaptability to specific uranium detection tasks. The results demonstrate the model's effectiveness in accurately localizing uranium sources while minimizing false positives.

Furthermore, the study evaluates the model's robustness against potential adversarial attacks and its generalization across different sensor platforms. The implications of deploying YOLO-based uranium detection systems in real-world nuclear security applications are discussed, emphasizing the importance of real-time monitoring and rapid response to potential threats.

II. Description

The YOLO model/program is significantly influenced by the hardware on which it operates. The efficiency of YOLO is closely tied to the computational power and architecture of the underlying hardware. Different hardware configurations can have profound effects on the model's speed, accuracy, and overall performance.

When deploying YOLO on various hardware platforms, the choice of hardware can impact the inference speed of the model. High-performance GPUs (Graphics Processing Units) are often favored for their parallel processing capabilities, allowing YOLO to rapidly analyze and detect objects in real-time. On the other hand, deploying YOLO on less powerful hardware, such as CPUs (Central Processing Units) or less capable GPUs, may result in slower inference speeds, potentially compromising the model's real-time detection capabilities. This can be extremely critical when being used for security reasons and potential safety scenarios where speed and accuracy are the most important.

The amount of available memory on the hardware also plays a crucial role. YOLO processes images in their entirety, and larger image resolutions or complex scenes may require more memory for efficient processing. Inadequate memory can lead to performance bottlenecks and may necessitate adjustments to the model's configuration to accommodate hardware limitations.

As technology evolves, specialized hardware accelerators like TPUs (Tensor Processing Units) and NPUs (Neural Processing Units) are increasingly integrated to enhance the performance of deep learning

models, including YOLO. These accelerators are designed to efficiently execute the complex computations involved in neural network inference, potentially unlocking new possibilities for real-time object detection with YOLO across various applications.

In summary, the impact of different hardware on YOLO modeling is multifaceted. From inference speed to memory considerations and the utilization of specialized accelerators, the choice of hardware profoundly shapes the practicality and effectiveness of YOLO in diverse deployment scenarios. As hardware technology continues to advance, YOLO's adaptability to varying hardware configurations remains a critical consideration for optimizing its performance across different applications and use cases.

III. History

In 2015 the first version of YOLO was released by both Joseph Redmon and Andrew Farhadi, and was already outpacing its competitors at the time. While other programs were built with conventional object detection algorithms, YOLO was performing under conventional neural network algorithms. This allowed YOLO to operate at faster speeds and also detect objects more efficiently than competing programs. Since its release there have been eight versions of this program.

Released in 2016, YOLOv2 made some slight improvements over the first model. While the first version would generate one or two boxes to identify an object, YOLOv2 was able to generate up to five boxes. At the same time, the user was also able to designate the number of boxes they could use.

In YOLOv3, which came out in 2018, this newest model was slightly slower than its predecessors but offered more features. The most notable feature was that this version of YOLO was able to classify and label objects for a specific object. The reason on how this was made possible was mainly due to the program replacing its original backbone network with another.

Years later, YOLOv4 was made available to the public in 2020 with already addressing some of the problems from the past - that being the ability to detect smaller objects. At the same time the main network for this version of the program was altered and redesigned to allow higher accuracy and faster readings. Finally, YOLOv4 added data augmentation in order to perform better.

In 2021, YOLOv5 was released with an upgraded conventional neural network. However with the fifth version of this program, came with several sub modes that differed in speed and accuracy. While some models were able to perform at faster speeds, the accuracy of detecting objects was poor. The same can be said for models that were slower but offered higher accuracies.

The following year, YOLOv6 and YOLOv7 were released several months apart in 2022 with slight adjustments. While YOLOv6 was able to mimic previous models, YOLOv7 could predict models using a new labeling system. This was made possible with a new network structure around the computational blocks.

Finally, moving onto the latest version of YOLO, YOLOv8 was released in 2023 with more enhanced features. Such

features were given a more integrated framework in order to improve object detection, and image classification. Much like YOLOv5, this version of the program offered several sub models tailored for different needs. As of now YOLOv8 offers the user a chance to remove detection boxes in order to improve speed on detecting classes.

IV. Why?

The importance of utilizing YOLO modeling for the detection of uranium in nuclear security applications lies in the critical need for swift and accurate identification of radioactive materials to prevent potential threats and ensure public safety. Here are some key reasons why this research is significant.

YOLO's real-time object detection capabilities enable the rapid identification of uranium sources within complex and dynamic environments. This speed is crucial for timely responses to potential threats, allowing security personnel to take appropriate actions swiftly.

The YOLO model, with its ability to process high-dimensional data using convolutional neural networks (CNNs), offers a high level of accuracy in identifying uranium signatures. This is essential to minimize false positives and negatives, ensuring that security systems provide reliable information for decision-making.

YOLO's adaptability to diverse datasets, encompassing various lighting conditions, backgrounds, and shieldings, makes it well-suited for the complex and unpredictable nature of nuclear security scenarios. This adaptability enhances the model's performance in real-world applications where conditions may vary.

In summary, the application of YOLO modeling in uranium detection addresses the urgent need for advanced, real-time, and accurate detection systems in the field of nuclear security, thereby enhancing our ability to respond effectively to potential threats and protect public safety.

V. Code

The implementation of our object detection model was carried out using the YOLOv8 model, a state-of-the-art model known for its speed and accuracy. We used the Ultralytics implementation of YOLOv8, which provides a Python package as well as a command-line interface. The Ultralytics package was installed in a Python>=3.8 environment with PyTorch>=2.1.0+cu118 CUDA:0 (Tesla T4, 15102MiB) on Google Colab, a cloud-based Python notebook environment that offers free access to GPUs.

In terms of accuracy, YOLOv8 has been shown to outperform its predecessors. For example, when measured on the COCO dataset, the YOLOv8m model – the medium model – achieves a 50.2% mAP. When evaluated against Roboflow 100, a dataset that specifically evaluates model performance on various task-specific domains.

VI. Performance And Implementation

The implementation of our object detection model was carried out using the YOLOv8 model. YOLOv8 is a state-of-the-art model that builds upon the success of previous YOLO versions, introducing new features and improvements to further boost performance and flexibility.

It is designed to be fast, accurate, and easy to use, making it an excellent choice for a wide range of object detection tasks.

We used the Ultralytics implementation of YOLOv8, which provides a Python package as well as a command-line interface. The Ultralytics package was installed in a Python>=3.10 environment with PyTorch>=2.1.0. This environment was set up both locally and on Google Colab, a cloud-based Python notebook environment that offers free access to GPUs. This allowed us to leverage the power of cloud computing and GPU acceleration for training our model, which significantly improved the efficiency of our work.

For our dataset, we utilized Roboflow, a platform that provides tools for creating datasets, training models, and deploying to production. Roboflow allowed us to bring in images in various annotation and image formats via API. We were able to filter, tag, segment, preprocess, and augment image data by metadata, train/test split, or location of image. This helped us to create a robust and diverse dataset for training our model.

In all three graphs, an epoch is one complete pass through the entire training dataset. The model's accuracy is a measure of how well the model performs on the validation set. The different colors in each graph represent different metrics used to measure the model's performance. The goal is to have the highest possible accuracy, which would mean the model is doing a good job predicting the validation data.

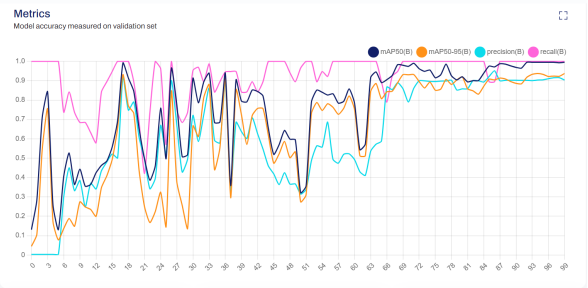

Figure 1: Model accuracy measured on validation 3rd set

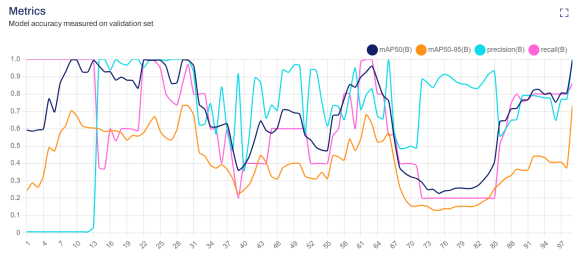

Figure 2: Model accuracy measured on validation 2nd set

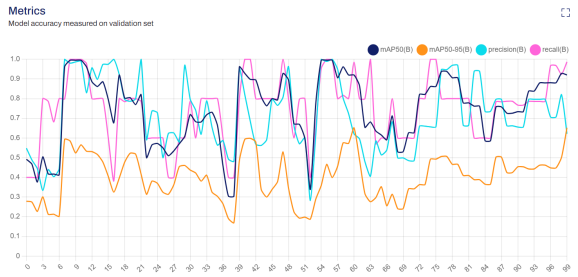

Figure 3: Model accuracy measured on validation 1st set

These three graphs show the Box Loss of a machine learning model over a number of training iterations. The x-axis represents the number of training iterations, and the y-axis represents the loss value. The orange line represents the training loss, and the blue line represents the validation loss

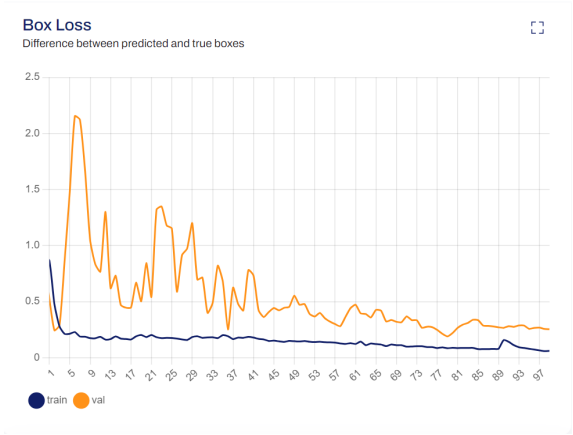

Figure 4: Box loss 3rd set

The graph from Figure 4 shows that the object loss for the training dataset decreases over time, while the object loss for the validation dataset increases over time. This suggests that the model may be overfitting on the training data and not generalizing well to unseen data in the validation set.

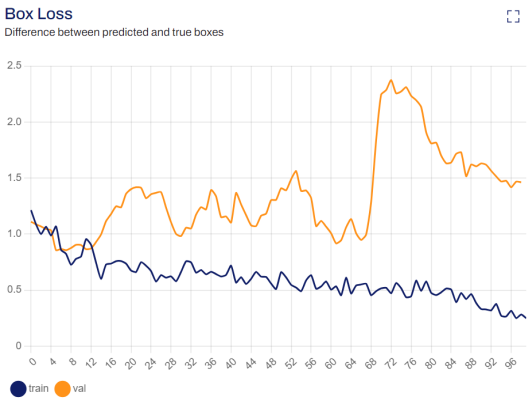

Figure 5: Box loss 2nd set

The graph from Figure 5 shows that the object loss for the training dataset decreases over time, while the object loss for the validation dataset increases over time. This suggests that the model may be overfitting on the training data and not generalizing well to unseen data in the validation set.

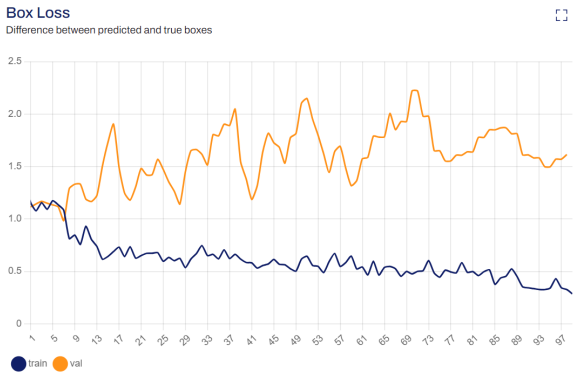

Figure 6: Box loss 1st set

The graph from Figure 6 shows that the training loss is decreasing steadily, which is a good sign. However, the validation loss is fluctuating, which might suggest the model is having trouble generalizing to new data. The goal is to have both the training and validation loss decrease to a point of stability with a minimal gap between the two.

These three graphs show the Class Loss of a machine-learning model set over a number of training iterations. The x-axis represents the number of training iterations, and the y-axis represents the loss value. The orange line represents the training loss, and the blue line represents the validation loss.

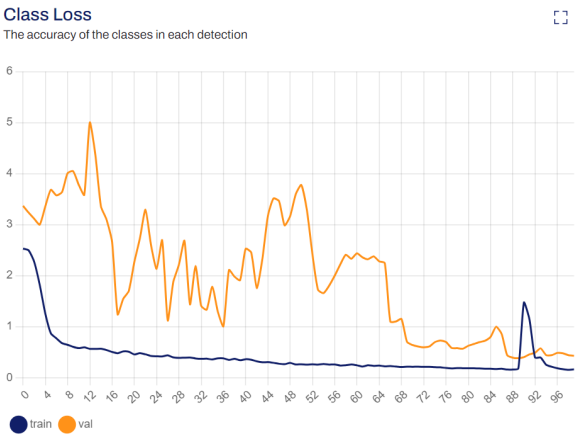

Figure 7: Class loss 3rd set

The graph from figure 7 shows that the accuracy of the classes decreases as the number of detections increases, which might suggest the model is having trouble generalizing to new data. The goal is to have both the training and validation loss decrease to a point of stability with a minimal gap between the two.

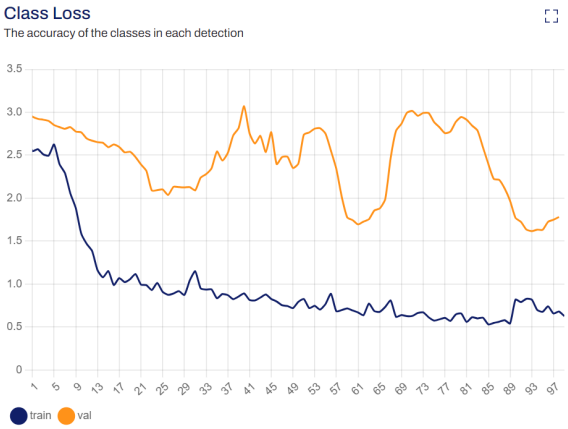

Figure 8: Class loss 2nd set

The graph of Figure 8 shows that the accuracy of the classes decreases as the number of detections increases, which might suggest the model is having trouble generalizing to new data.

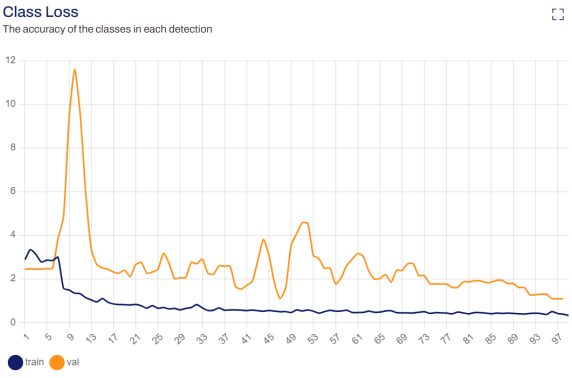

Figure 9: Class loss 1st set

The graph Figure 9 shows that the accuracy loss for the training data is higher than the validation data, which might suggest

the model is not overfitting and is generalizing well to the validation data.

The Object Loss graph represents the performance of a machine learning model in detecting objects in an image. The x-axis represents the number of training iterations, and the y-axis represents the loss value. The orange line represents the training loss, and the blue line represents the validation loss.

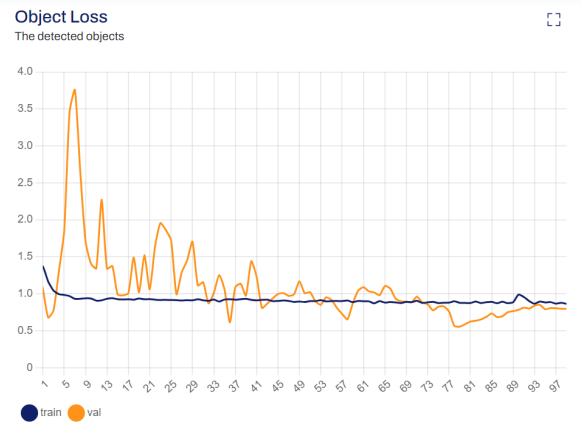

Figure 10: Object loss 3rd set

From the graph of Figure 10, we can see that the training loss (orange line) has a peak at around x=20 and then decreases steadily until x=91. The validation loss (blue line) also has a peak at around x=20 and then fluctuates until x=91. This suggests that the model is learning and improving its ability to detect objects over time.

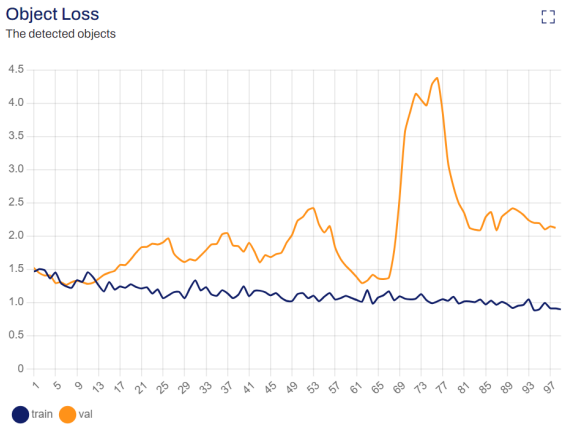

Figure 11: Object loss 2nd set

The graph from Figure 11 shows that the object loss for the validation dataset is higher than the training dataset initially. However, as the number of epochs increases, the object loss for both the datasets decreases. This suggests that the model is learning and improving its ability to detect objects over time.

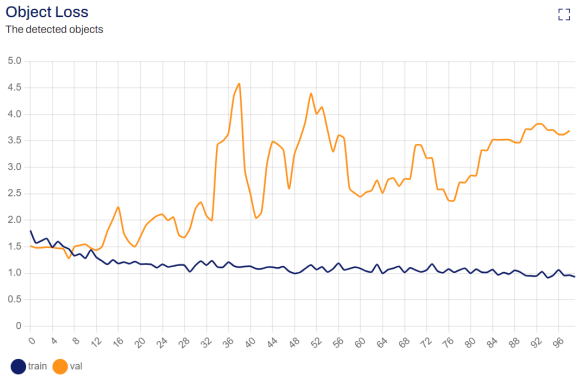

Figure 12: Object loss 1st set

The graph from Figure 12 shows that the object loss for the training dataset decreases over time, while the object loss for the validation dataset increases over time. This suggests that the model may be overfitting on the training data and not generalizing well to unseen data in the validation set.

In machine learning, loss is the difference between the predicted and true values. These graphs are used to evaluate the model's performance and identify areas for improvement. The blue line represents the training loss and the orange line represents the validation loss. The goal is to minimize this loss over the training iterations.

VII. Conclusion

YOLO has been renowned for its real-time object detection capabilities and has been adapted to address the challenges posed by the detection of uranium. Through trial and error, the latest version of this program has

been pushed to its limits in the hopes of detecting uranium by using Ultralytics implementation. As it is shown in Figures 4 through 12, it can be said that the program is gradually learning on how to detect Uranium with each trial being more accurate than before. With the project ending after Figure 12, one could say that, should the trials resume and continue, the program will become more accurate in determining uranium. The overall objective of this project was to train an object detection system, such as YOLO, into locating, determining, and classifying an object as either Uranium or not. From the data that was taken, it can be concluded that the goal was achieved and the methods that were applied to this project can be applied to other YOLO projects in the future.

VIII. Reference

[1] Kang, C.H., Kim, S.Y. Real-time object detection and segmentation technology: an analysis of the YOLO algorithm. *JMST Adv.* **5**, 69–76 (2023). https://doi-org.proxy.library.cpp.edu/10.1007/s42791-023-00049-7

[2] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics," *GitHub*, Jan. 01, 2023. https://github.com/ultralytics/ultralytics

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

[4] "Roboflow: Go from Raw Images to a Trained Computer Vision Model in Minutes.," roboflow.ai. https://roboflow.com/