Department of Electrical and Computer Engineering
North South University

Senior Design Project

Leveraging Deep Learning to Estimate the Damage Caused by Natural Disasters

Md. Rifat Ahmed	**ID# 1931725042**
Saifur Rahman Rubayet	**ID# 1921870042**
Gazi Shoaib	**ID# 1921198042**

Faculty Advisor:

Dr. Shahnewaz Siddique

Associate Professor

ECE Department

Summer, 2023

LETTER OF TRANSMITTAL

November, 2023

To

Dr. Rajesh Palit

Chairman,

Department of Electrical and Computer Engineering

North South University, Dhaka

Subject: **Submission of Capstone Project Report on “Leveraging Deep Learning to Estimate the Damage Caused by Natural Disasters”**

Dear Sir,

With due respect, we would like to submit our **Capstone Project Report** on **“Leveraging Deep Learning to Estimate the Damage Caused by Natural Disasters”** as a part of our BSc program. The report deals with various deep learning models, that are used to classify and segment building damage from satellite images. This project was very much valuable to us as it helped us gain experience from practical field and apply in real life. We tried to the maximum competence to meet all the dimensions required from this report.

We will be highly obliged if you kindly receive this report and provide your valuable judgment. It would be our immense pleasure if you find this report useful and informative to have an apparent perspective on the issue.

Sincerely Yours,

....

Md. Rifat Ahmed

ECE Department

North South University, Bangladesh

....

Saifur Rahman Rubayet

ECE Department

North South University, Bangladesh

....

Gazi Shoaib

ECE Department

North South University, Bangladesh

APPROVAL

Md. Rifat Ahmed (ID # 1931725042), Saifur Rahman Rubayet (ID # 1921870042) and Gazi Shoaib (ID # 1921198042) from Electrical and Computer Engineering Department of North South University, have worked on the Senior Design Project titled "Leveraging Deep Learning to Estimate the Damage Caused by Natural Disasters" under the supervision of Dr. Shahnewaz Siddique partial fulfillment of the requirement for the degree of Bachelors of Science in Engineering and has been accepted as satisfactory.

Supervisor's Signature

……………………………………….

Dr. Shahnewaz Siddique

Associate Professor

Department of Electrical and Computer Engineering

North South University

Dhaka, Bangladesh.

Chairman's Signature

……………………………………….

Dr. Rajesh Palit

Professor

Department of Electrical and Computer Engineering

North South University

Dhaka, Bangladesh.

DECLARATION

This is to declare that this project is our original work. No part of this work has been submitted elsewhere partially or fully for the award of any other degree or diploma. All project related information will remain confidential and shall not be disclosed without the formal consent of the project supervisor. Relevant previous works presented in this report have been properly acknowledged and cited. The plagiarism policy, as stated by the supervisor, has been maintained.

Students’ names & Signatures

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

1. Md. Rifat Ahmed

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

2. Saifur Rahman Rubayet

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

3. Gazi Shoaib

ACKNOWLEDGEMENTS

The authors would like to express their heartfelt gratitude towards their project and research supervisor, Dr. Shahnewaz Siddique, Associate Professor, Department of Electrical and Computer Engineering, North South University, Bangladesh, for his invaluable support, precise guidance and advice pertaining to the experiments, research and theoretical studies carried out during the course of the current project and also in the preparation of the current report.

Furthermore, the authors would like to thank the Department of Electrical and Computer Engineering, North South University, Bangladesh for facilitating the research. The authors would also like to thank their loved ones for their countless sacrifices and continual support.

ABSTRACT

Leveraging Deep Learning to Estimate the Damage Caused by Natural Disasters

Accurate estimation of natural disaster damage is crucial for effective response and recovery planning. This study focuses on creating a sophisticated deep learning system that combines YOLOv8 and SegFormer for segmentation and ResNet-50 and YOLOv8 for classification in order to accurately estimate the damage caused by natural disasters. Having been trained on an extensive dataset, the system makes use of transfer learning strategies to improve performance. Preliminary results indicate the system's efficacy in accurately estimating damage extent and severity, presenting a robust framework for holistic damage assessment. By offering fast and precise assessments, facilitating optimal resource allocation, facilitating effective response planning, and minimizing overall damage on infrastructure and communities, this strategy has the potential to completely transform the management of natural disasters.

Keywords: *deep learning, natural disasters, damage estimation, convolutional neural networks (CNN), satellite imagery, disaster management.*

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1 Introduction

1.1 Background and Motivation

Natural disasters, such as earthquakes, hurricanes, heavy rain leading to floods, and droughts, can cause widespread damage and pose significant threats to human life. Deep learning algorithms offer a promising solution for natural disaster management by enabling the development of systems that can predict, monitor, and respond to such events. Using deep learning algorithms, we can analyze various data sources, including weather forecasts, satellite imagery, and sensory data, to build models that accurately predict and monitor the impact of natural disasters. These models can provide valuable insights and damage assessments, empowering disaster management officials to make informed decisions and allocate resources effectively. By integrating deep learning algorithms into the disaster management process, officials can swiftly assess the severity and extent of a disaster, enabling them to prioritize response efforts efficiently. This technology-driven approach enhances the speed and accuracy of decision-making, leading to more effective recovery attempts. Moreover, the proposed system has the potential to transform the way natural disasters are managed. Leveraging advanced data analytics and predictive modeling techniques enables proactive measures to be taken, reducing the potential impact on human lives and infrastructure. Additionally, the system facilitates improved coordination among multiple stakeholders involved in disaster response, enhancing overall efficiency and response time.

1.2 Purpose and Goal of the Project

The primary goal of this work is to develop a cutting-edge deep learning system utilizing YOLOv8 and ResNet-50 for building damage classification, along with SegFormer and YOLOv8 for building segmentation from the satellite images. The system aims to accurately estimate the extent and severity of damage resulting from natural disasters. Leveraging the YOLO models bounding box prediction system and ResNet models' capacity to handle deep networks and capture intricate spatial patterns, the system is trained on a comprehensive dataset consisting of pre- and post-disaster satellite imagery, contextual information, and labeled damage data. The objective is to enhance the models' capability to identify damaged areas and assess the severity of destruction caused by various types of disasters. By incorporating transfer learning techniques, the project strives to improve the model's performance and generalization capabilities. Ultimately, this endeavor seeks to revolutionize natural disaster management by providing precise and timely damage estimations, offering valuable insights for efficient resource allocation and informed response planning.

1.3 Organization of the Report

Several topics had to be covered for the development of the system. In Section III, the methodology – the architectures of the Classification Models are explained, followed by the Segmentation models and then the details of the implementation are discussed, finishing it up with the details of the Dataset. Then in Section IV, the results obtained from the different models are discussed. Finally concluding everything in the conclusion in Section V with a brief summary of the whole work and future scopes that lies with it.

Chapter 2 Research Literature Review

2.1 Existing Research and Limitations

Leveraging deep learning algorithms in natural disaster management holds great promise. By harnessing the power of data analysis, prediction, and monitoring, we can create systems that revolutionize how we respond to and recover from these catastrophic events. This technology-driven approach empowers decision-makers and improves resource allocation, leading to more effective and responsive disaster management efforts.

In [1], Danu Kim and his colleagues focused on using computer vision and satellite imagery in disaster assessment to detect water-related structural damages. They proposed a binary classification model to identify the damaged areas by detecting the structural change in an area using the pre- and post-disaster satellite images. The authors implemented transfer learning to train their model efficiently. So before fine-tuning the model to identify damages from a pair of satellite photos, the model is first pre-trained using satellite photos without a disaster. To pre-train their model, the authors used the ResNet-18 convolutional neural network using the ImageNet-1000 dataset and then used the xBD dataset to fine-tune it. Their model succeeded in identifying areas with structural damages with an accuracy of 85.9% and a reliable accuracy of 80.3% in non-domain conditions.

R. F. Ahmad and his colleagues worked on one of the early works in disaster monitoring using satellite stereo images. They used satellite images from Quick Bird for preliminary studies. Then, using a depth estimation algorithm, they tried to compare the pre- and post-disaster satellite images of different areas to detect disaster-affected areas [2].

In 2017, Amit and Aoki proposed an automated natural disaster detector specializing in landslide and flood detection in their paper. They created their dataset by clipping and resizing satellite images of pre- and post-disaster from Google Earth aerial imagery, focusing on Thailand and Japan. The framework for their system was “Caffe,” and the CNN architecture used was AlexNet. The accuracy of disaster detection in their system was around 80–90% for both disasters [3].

In [4], F. Zhao et al. implemented a Mask R-CNN-based damage assessment model where they first trained ResNet 101 in Mask R-CNN as a "Building Feature Extractor" and further trained it to build a Siamese-based semantic segmentation model to differentiate damaged buildings from undamaged ones at a pixel level.

Chapter 3 Methodology

3.1 Classification Models

ResNet-50: The first classification model chosen for damage classification is the ResNet-50, a type of convolutional neural network (CNN). ResNet-50 is a 50-layered CNN architecture with over 25 million parameters [5]. The architecture of the ResNet-50 model is shown in Table I [6].

Table I. ResNet-50 Architecture

Layer Name	**Output Size**	**ResNet-50**
conv1	112×112×64	7×7, 64, stride 2
conv2_x	56×56×256	3×3 max pool, stride 2
		$\begin{bmatrix} 1\times 1, 64 \\ 3\times 3, 64 \\ 1\times 1, 256 \end{bmatrix} \times 3$
conv3_x	28×28×512	$\begin{bmatrix} 1\times 1, 128 \\ 3\times 3, 128 \\ 1\times 1, 512 \end{bmatrix} \times 4$
conv4_x	14×14×1024	$\begin{bmatrix} 1\times 1, 256 \\ 3\times 3, 256 \\ 1\times 1, 1024 \end{bmatrix} \times 6$
conv5_x	7×7×2048	$\begin{bmatrix} 1\times 1, 512 \\ 3\times 3, 512 \\ 1\times 1, 2048 \end{bmatrix} \times 3$
Average Pool	1×1×2048	7×7 average pool
Fully Connected	1000	2048×1000
Softmax		

Deep-layered architectures were designed to get more efficient results using the high number of convolutional layers. However, adding several deep layers to a network leads to a degradation of the output, introducing the problem of "Vanishing Gradient." During the training, algorithms such as gradient descent use backpropagation to calculate the loss function and update the weights. However, with numerous layers, the gradient calculation undergoes many multiplications, causing the gradient to get smaller and smaller, eventually "vanishing," resulting in the network's degraded performance [7].

So, the accuracy of the model starts to get saturated when increasing the depth of the model, and to solve this problem of the "vanishing gradient," residual blocks were introduced, which are the building blocks of a ResNet architecture. The ResNet architecture is formed by stacking these residual blocks on top of each other. These residual blocks are the shortcut connections, also called skip connection blocks, which work by connecting a layer's input directly to another layer's output after skipping a few connections, as we can see here in Fig. 1 [6].

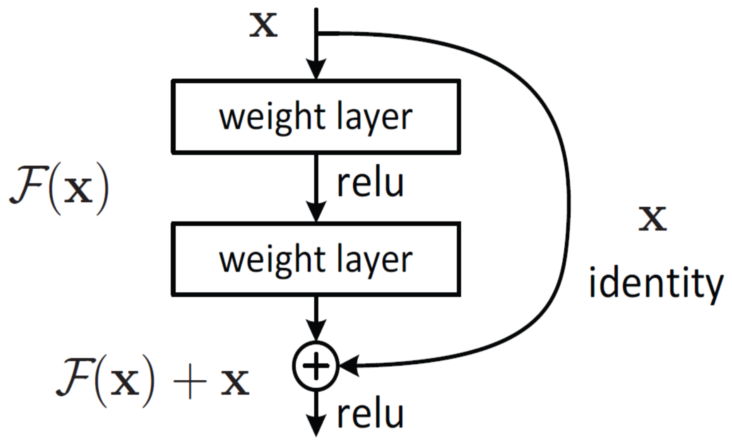
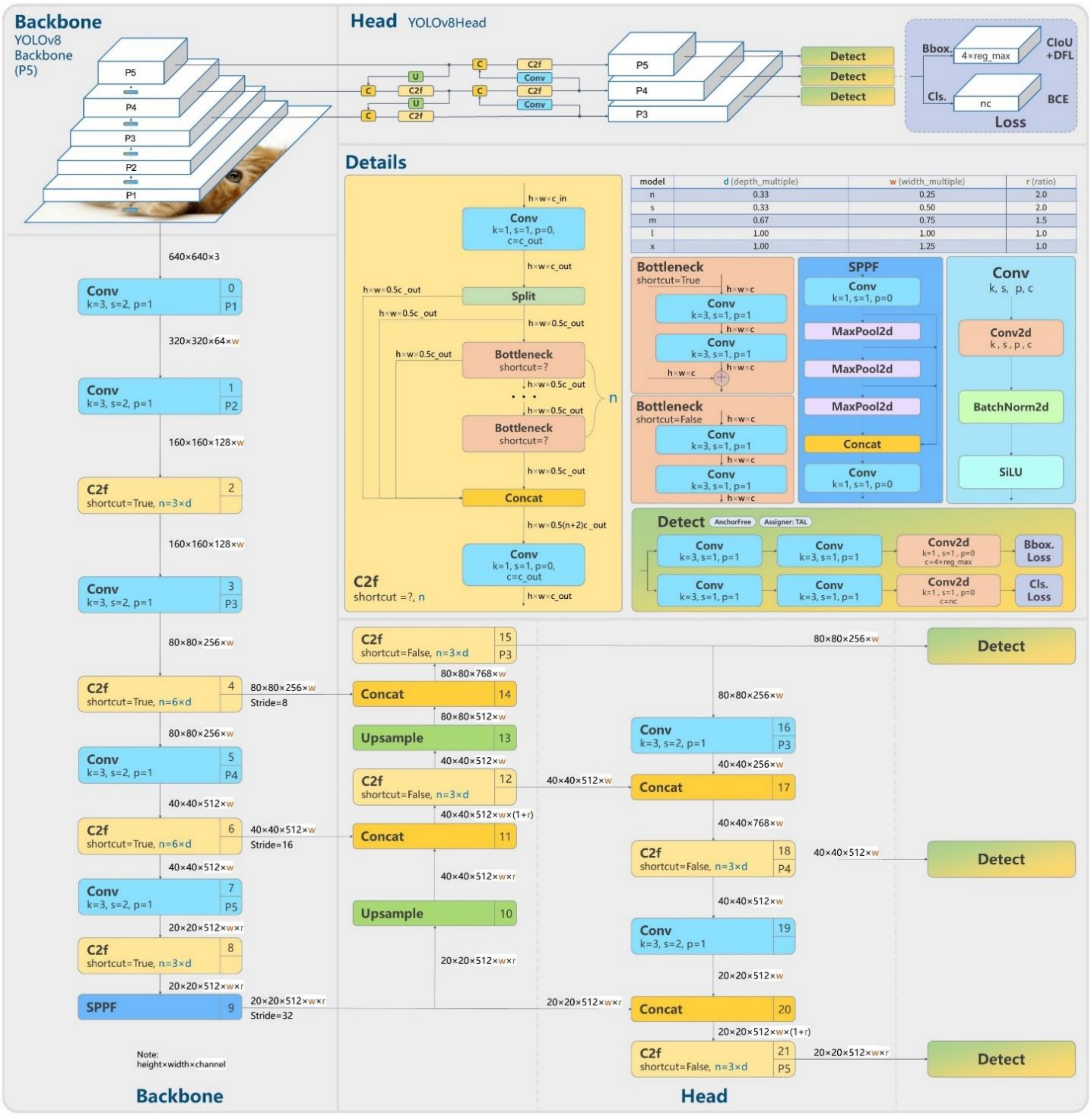


Figure 1. Residual Block.

YOLOv8: In 2016 Joseph et al. came up with the state-of-the-art object detection algorithm called "YOLO" short for "You Only Look Once". Their model fared better than other popular object detection techniques such as DPM and R-CNN. [8]. Then they went on to add features like anchor boxes, dimension clusters, pyramid pooling and further improve their backbone network [9] [10]. Alexey Bochkovskiy and his team then developed YOLOv4, which featured mosaic data augmentation and was capable of anchor-free detection [11]. Subsequently, the Vision AI-focused company Ultralytics created YOLOv5, which had better performance than its predecessors and it composed of three parts the Backbone, Neck, and Head [12]. They are also the creators of the most recent, cutting-edge YOLOv8 model, which is quite similar to YOLOv5 in terms of the architecture except for YOLOv8 the neck is not exclusively mentioned in the documentation [13].



model	d (depth_multiple)	w (width_multiple)	r (ratio)
n	0.33	0.25	2.0
s	0.33	0.50	2.0
m	0.67	0.75	1.5
l	1.00	1.00	1.0
x	1.00	1.25	1.0

Figure 2. YOLOv8 Architecture.

The Backbone is the deep learning architecture that works as the feature extractor while the neck combines the various features acquired from the backbone and the head predicts the classes and the bounding boxes which is the final output produced by the model. While the YOLOv5 used C3 module based on Cross Stage Partial (CSP) in the backbone, YOLOv8 used C2f module. CNN's learning ability is improved by the CSP architecture, which also reduces the model's computing load [14].

Several model variants of YOLOv8 are available for various applications such as pose detection, segmentation, and classification. And the model selected in this study for classification was the YOLOv8x-cls model which is pretrained on the ImageNet dataset [13].

3.2 Segmentation Models

SegFormer: SegFormer is a semantic segmentation approach based on Transformers. Figure 3 shows its architecture which consists of two main modules: a hierarchical Transformer encoder and an All-MLP decoder [15].

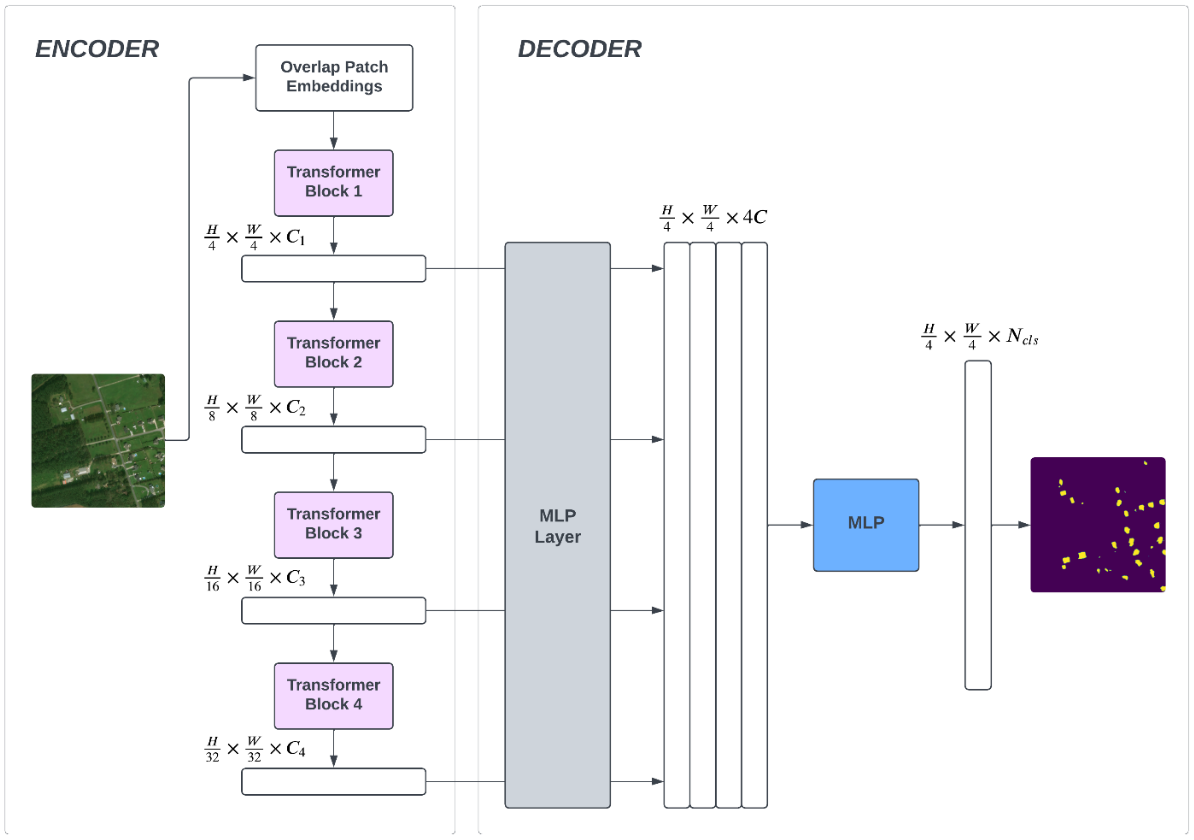

ENCODER
Overlap Patch Embeddings
Transformer Block 1
$\frac{H}{4} \times \frac{W}{4} \times C_1$
Transformer Block 2
$\frac{H}{8} \times \frac{W}{8} \times C_2$
Transformer Block 3
$\frac{H}{16} \times \frac{W}{16} \times C_3$
Transformer Block 4
$\frac{H}{32} \times \frac{W}{32} \times C_4$
DECODER
MLP Layer
$\frac{H}{4} \times \frac{W}{4} \times 4C$
MLP
$\frac{H}{4} \times \frac{W}{4} \times N_{cls}$

Figure 3. SegFormer Architecture.

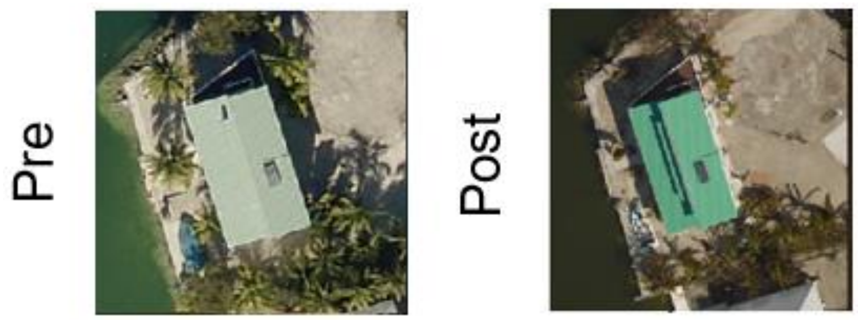
The hierarchical Transformer encoder takes an image as input and produces a set of multi-level features. These features are then passed into the All-MLP decoder, which predicts the segmentation mask for the image. The All-MLP decoder is a simple and lightweight decoder that consists of only MLP layers. This makes it much more efficient than other decoders that use hand-crafted and computationally demanding components. The key to enabling such a simple decoder is that the hierarchical Transformer encoder has a larger effective receptive field (ERF) than traditional CNN encoders. This means that the encoder can capture more context information from the image, which makes it easier for the decoder to predict the segmentation mask.

However, due to a lack of computational resources, the smallest variant of the SegFormer model (SegFormer-B0) had to be chosen for segmentation.

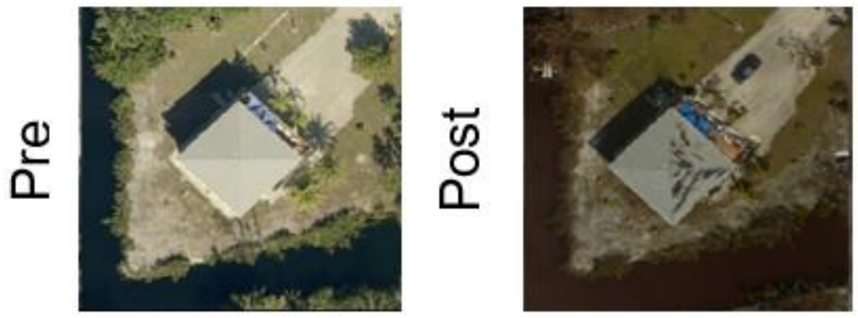
Coming back to YOLO, all of the YOLO models slated for segmentation are pretrained on the COCO dataset [13]. And the model chosen for this study was the YOLOv8n-seg which is the nano (smallest and fastest) variant of the YOLO segmentation model.

3.3 Dataset

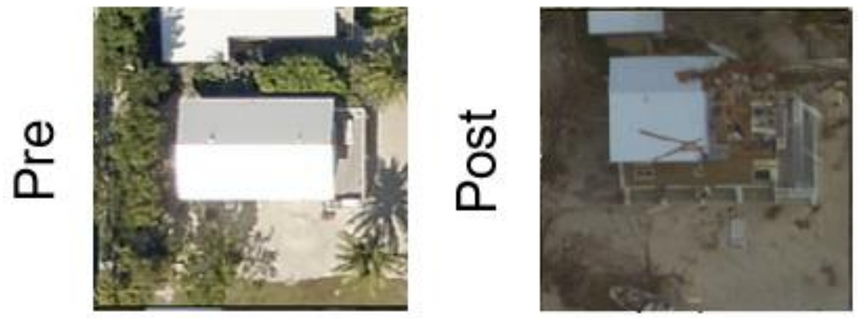
The xBD dataset, collected from the xView website, is a precious resource for computer vision research. It offers a vast and diverse collection of satellite imagery encompassing urban and rural landscapes, with 850,736 annotated buildings spanning over the 45,362 km2 area of imagery, which are split for train, test, and holdout, respectively, with a split ratio of 80/10/10% [16]. Additionally, the dataset offers a fine-grained assessment of damage levels for buildings, spanning from "no damage" to "minor damage," "major damage," and even "destroyed." This comprehensive coverage of disaster scenarios and detailed damage classification truly sets the dataset apart, making it an invaluable asset for research and analysis in the field.



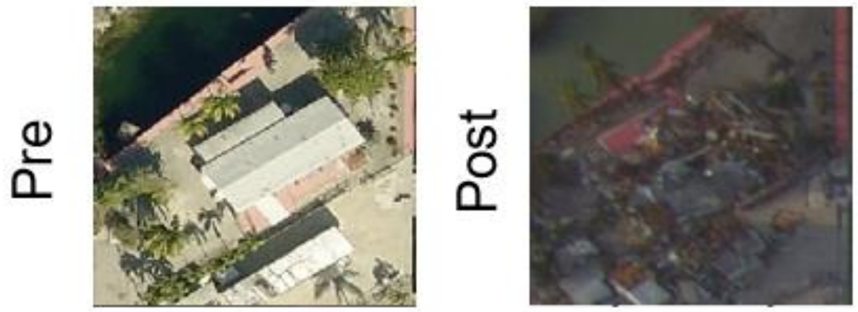
a) No Damage



b) Minor Damage



c) Major Damage



d) Destroyed

Figure 4. Different Types of Damages in the Dataset.

The xBD dataset team employed experts to annotate buildings in pre-disaster images using polygons. The ground truth polygons in post-disaster images were derived by projecting them from the pre-disaster images and aligning them based on satellite coordinates. For the post-disaster image annotation, damage evaluation experts were invited to label the polygons with four damage levels. Figure 4 provides sample images representing each damage level with pre- and post-images. The evaluation standard for post-disaster image annotation can be summarized as follows: Buildings showing no signs of water, structural damage, or burn marks are categorized as having no damage. Buildings labeled as destroyed have completely collapsed or are covered with water or mud. The minor damage classification is assigned when a building is partially burnt, surrounded by water, missing roof elements, or exhibits visible cracks. Major damage refers to a partial wall, roof collapse, or buildings surrounded by water. Although there is a distinct contrast between buildings with no damage and those that are destroyed, distinguishing between minor and majorly damaged buildings is often challenging when relying solely on the visual inspection of satellite images. [16]

However even with all that the Dataset is heavily imbalanced, most of the building crops being Undamaged type as it can be seen in Figure 5, which makes training the dataset a lot more complex.

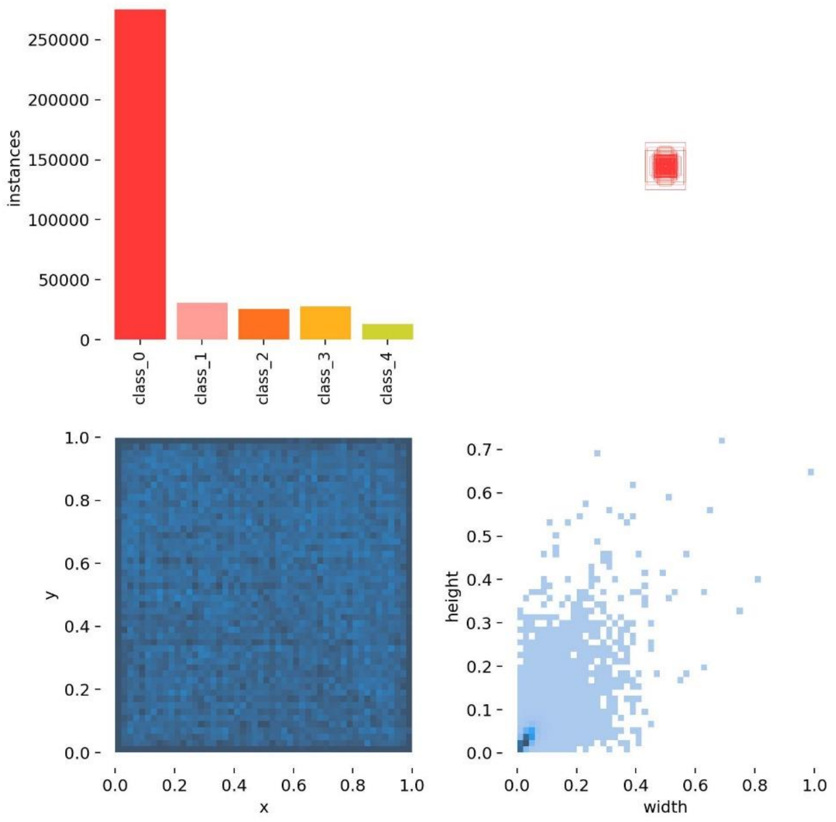


Figure 5. Class Imbalances in the Dataset.

3.4 Hyperparameters

For ResNet-50 and SegFormer models, the Cross-Entropy Loss function was used to adjust the weights during the training, the formula of which is given in equation (1), where ‘y_i’ is the actual value, ‘$\widehat{y_i}$’ is the predicted value, and ‘n’ is the output size.

$$Cross\ Entropy\ Loss = -\sum_{i=1}^{n} \quad y_i \log \log \widehat{y_i} \qquad (1)$$

For the optimization algorithm Adam optimizer was used. Adam, short for adaptive moment estimation, computes adaptive learning rates individually for various parameters. Adam combines two popular optimization methods: AdaGrad, which is good with sparse gradients, and RMSProp, which works well with non-convex optimization problems. Thus, it works so well with computer vision-related works [17].

Table II. Hyperparameters for ResNet-50 and SegFormer

Hyperparameter	**Value**
Optimizer	Adam
Loss Function	Cross-Entropy
Learning Rate	0.001

YOLOv8 on the other hand uses 2 loss functions called Bbox loss and Cls loss while the optimization algorithm and learning rate to start with was kept the same at 0.001.

Table III. Hyperparameters for YOLOv8

Hyperparameter	**Value**
Optimizer	Adam
Loss Function	Bbox loss, Cls loss
Learning Rate	0.001

For classification, both ResNet-50 and YOLOv8 were trained for 150 epochs each, while for segmentation, the YOLOv8 and the SegFormer models were trained for 15 epochs.

3.5 Implementation

Since YOLO models specialize in rectangle-shaped bounding boxes, the initial step in implementation was to transform the polygon labels into bounding boxes of rectangle shape to achieve better classification and segmentation results.

Then for segmentation the SegFormer model was wrapped using LoRA (Low Rank Adaptation) to train the segmentation model faster since the size of the xBD Dataset is pretty huge. In the

SegFormer model, the number of trainable parameters can be cut down to just 14% of the initial set using LoRA [18].

The low-rank "update matrices" that LoRA adds to particular model blocks—like the attention blocks—help to achieve this reduction. During the process of fine-tuning only the update matrices are trained while the initial model parameters remain unaltered. The update matrices and the initial model parameters are combined at inference time to get the final output.

After having all the models trained, using a script the classification output of the YOLOv8 was merged with the segmentation output of the SegFormer model to generate a Segmented output that showed both the classification and the segmentation in the same image like in Figure 6 where the green colors in the output indicate the buildings were undamaged.

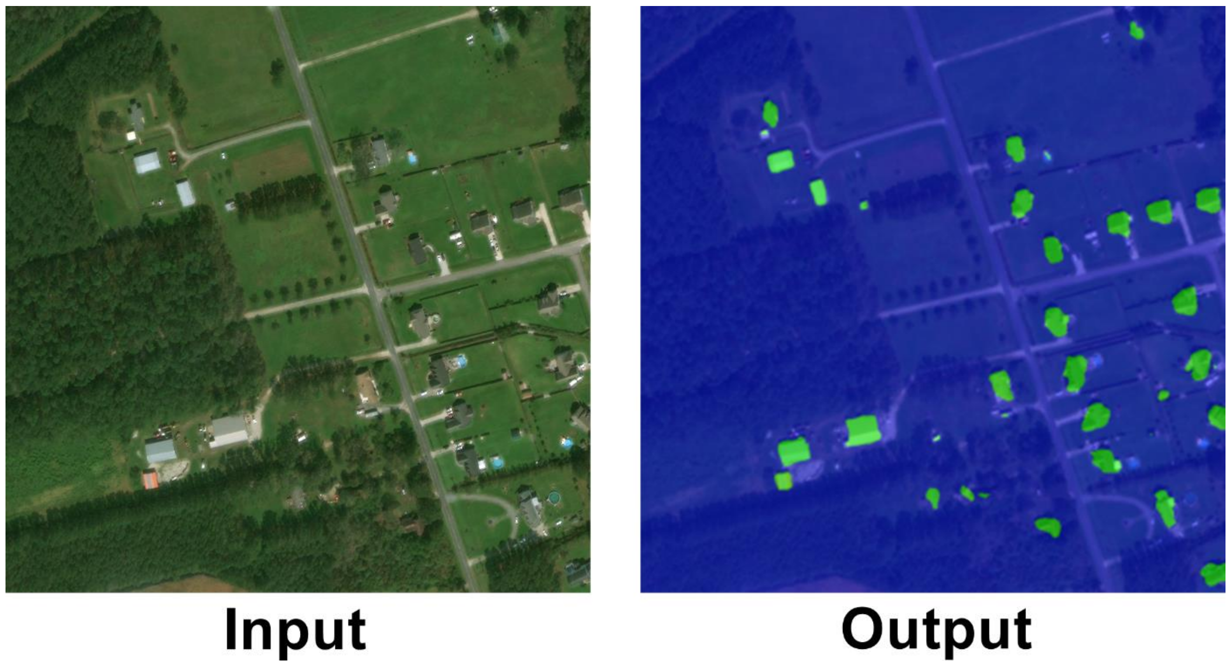


Figure 6. Segmented Classified Output.

Chapter 4 Investigation/Experiment, Result, Analysis and Discussion

From the confusion matrix in Figure 7, it can be seen that the model is predicting the results here and there being a bit messy.

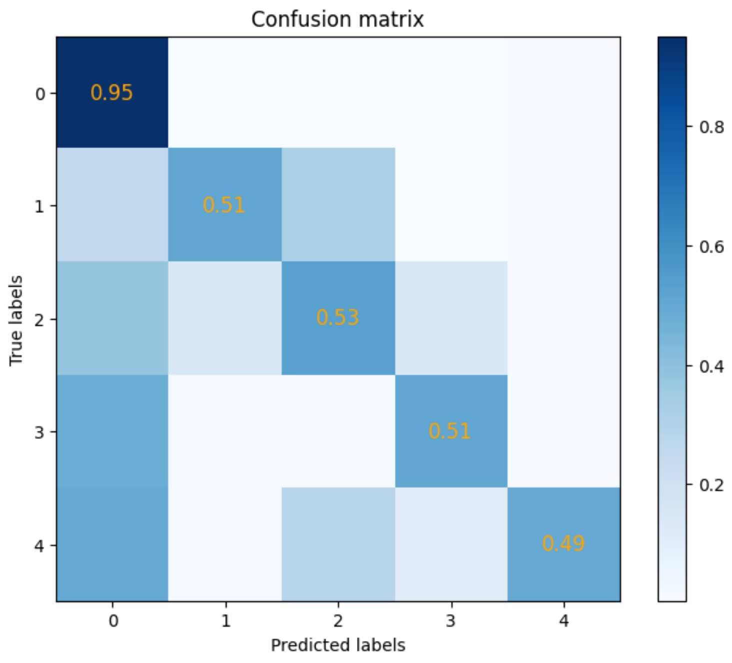


Figure 7. ResNet-50 Classification - Confusion Matrix.

But looking at the confusion matrix in Figure 8 it can be seen that the YOLOv8 model is predicting class 0 which is for the damage type “Undamaged” most of the times which is actually pretty logical if we look at the imbalance in Figure 4.

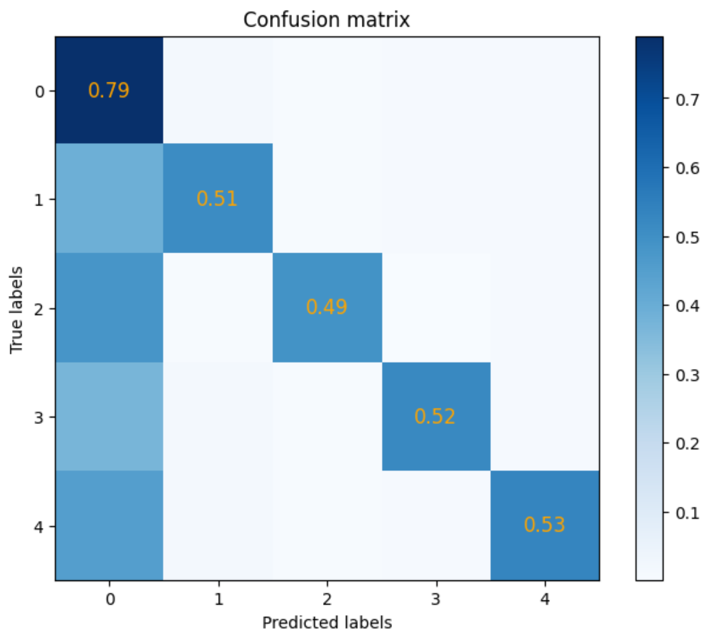


Figure 8. Classification - Confusion Matrix.

As shown in table IV, a precision of 0.530 and a mAP50 score of 0.333 were obtained while classifying using YOLOv8. In YOLO the Bbox loss refers to the Bounding Box Prediction loss and the Cls loss refers to the Class detection loss.

Table IV. Classification Model 1 (YOLOv8x-cls) Results

Precision	**Recall**	**mAP50**	**Bbox Loss**	**Cls Loss**
0.530	0.312	0.333	1.676	1.210

Conversely, a precision of 0.598 and a mean accuracy of 52.5% were obtained using the ResNet-50 model.

Table V. Classification Model 2 (ResNet-50) Results

Precision	**Recall**	**Mean Accuracy**	**Cross Entropy Loss**
0.598	0.518	0.525	1.126

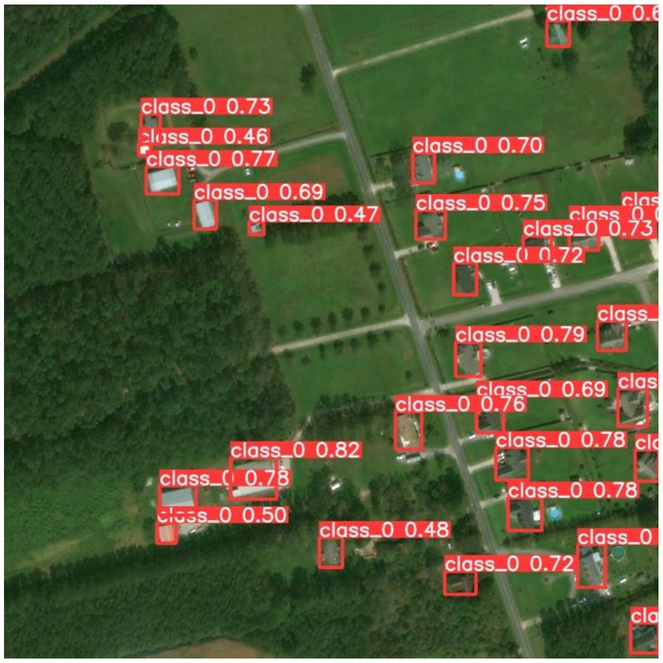


Figure 9. Classification Prediction.

Given that the SegFormer model is solely being used for segmenting the buildings crops, Table VI shows that the IoU for Building is 0.6845 also giving the same amount mean accuracy for the model at 68.5%.

Table VI. Segmentation Model 1 (SegFormer-B0) Results

Cross Entropy Loss	**IoU (Building)**	**Mean Accuracy**
0.0719	0.6845	0.6845

The final segmentation model using the YOLOv8 gives an accuracy of 48.6% which seems pretty low but having low resources to compute and heavy imbalances in the dataset that is to be expected.

Table VII. Segmentation Model 2 (YOLOv8n-seg) Results

Precision	**Recall**	**Accuracy**	**Bbox Loss**	**Cls Loss**
0.342	0.215	0.486	2.012	2.477

In the following figure the left displays the original satellite image randomly chosen from the xBD dataset and the segmented output is on the right.

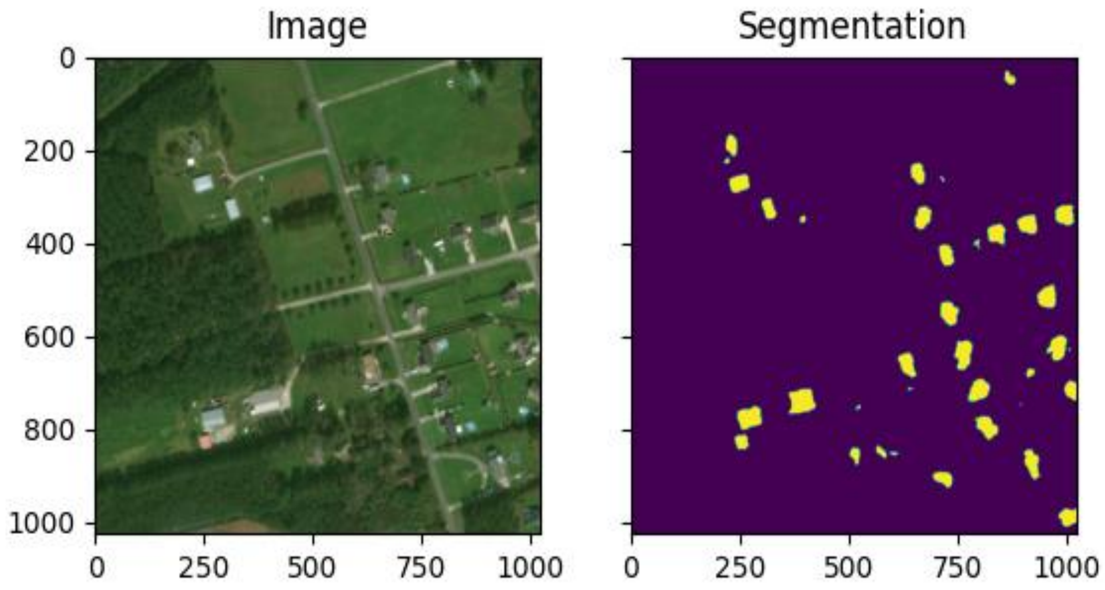


Figure 10. Segmentation Prediction.

Lastly, the segmented output in the original image is shown in Figure 11. And now going back to Figure 6, the combined damage classification and segmentation output can be understood easily

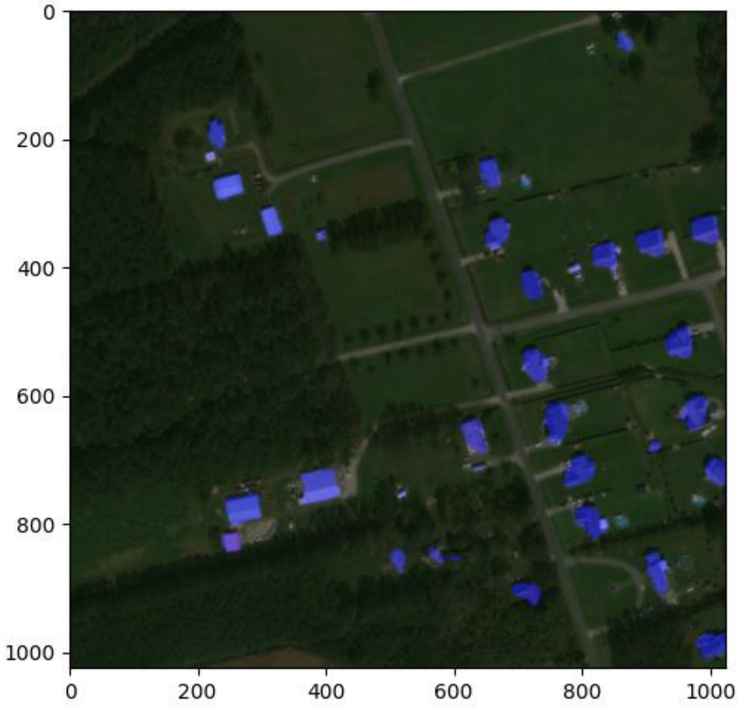


Figure 11. Segmentation Prediction in Original Image.

Consequently, the SegFormer is clearly producing superior results in the segmentation part, while the YOLOv8 appears to be performing similar to ResNet-50 among the two classification models. Although both models shown some promise in categorising damage types, they faced limitations as a result of class imbalance in the dataset. The model's performance was negatively impacted by an unbalanced distribution caused by the lack of data from various damage categories. Lower accuracy and recall were seen for those particular categories of building damage because the model had difficulty learning and differentiating these less represented classes due to the substantial decrease in the number of instances of those sorts of damage. In order to improve the model's performance and provide fair representation of different damage types to enable more accurate and balanced predictions, it becomes necessary to address the issue of class imbalance. Nevertheless, even if the results are now a little below average, they may still be enhanced by testing out larger models with a sizable number of additional parameters and by fine-tuning the current model even more. Another possible solution to lessen the effect of class imbalance on both the classification model’s effectiveness in identifying building damage is to apply data augmentation techniques [19]. Data augmentation may be used to rebalance the dataset by artificially creating more instances of the underrepresented classes using techniques like rotation, flipping, or adding noise to the current data. By giving the model more varied instances of the minority classes, this augmentation step would contribute to the enrichment of the training set.

Chapter 5 Impacts of the Project

5.1 Impact of this project on societal, health, safety, legal and cultural issues

1. **Societal Influence:**
   A. Prompt Response: Deep learning models can expedite and enhance damage evaluation, leading to quicker responses in the aftermath of natural disasters. This has the potential to save lives and mitigate the overall impact on affected communities.

   B. Optimized Resource Allocation: Improved damage estimation enables more effective resource distribution, ensuring that assistance is directed to the most critical areas. This can boost the efficiency of relief efforts and aid in community recovery.

2. **Health Consequences:**
   A. Emergency Medical Intervention: Accurate damage assessment assists in identifying regions requiring urgent medical aid, improving the targeting of resources to areas with higher injury and health risks.

3. **Safety Outcomes:**
   A. Enhanced Evacuation Planning: Deep learning models contribute to more precise evacuation planning by pinpointing high-risk zones and potential hazards, enhancing individual safety during natural disasters and reducing the likelihood of casualties.

4. **Legal Implications:**

A. Streamlined Insurance Claims: Precise damage estimation facilitates smoother processing of insurance claims, crucial for individuals and businesses seeking compensation for losses incurred during natural disasters.

B. Liability Determination: The technology aids in determining liability in legal proceedings by providing objective data on the extent of damage and potential causes.

5. **Cultural Ramifications:**

A. Preservation of Heritage: Deep learning models play a crucial role in evaluating damage to cultural and historical sites, guiding efforts to preserve and restore cultural heritage that is often at risk during natural disasters.

6. **Ethical Considerations:**

A. Privacy Challenges: The use of deep learning models for damage estimation may raise concerns about privacy, especially when the technology involves analyzing images or data containing sensitive information. Balancing technological benefits with privacy considerations is crucial.

B. Bias and Equity: Ensuring that deep learning models used for damage assessment are trained on diverse and representative data is essential to prevent unintended biases that could affect the fair distribution of resources and aid.

5.2 Impact of this project on environment and sustainability

1. **Environmental Impact**
   A. Reduced Reliance on Traditional Damage Assessment Methods: Conventional damage assessment techniques, such as aerial surveys and ground-based inspections, can be time-consuming, costly, and environmentally harmful. Deep learning-based systems provide an efficient and sustainable alternative, minimizing the environmental impact associated with traditional methods.

   B. Enhanced Monitoring of Environmental Damage: Deep learning's ability to analyze vast amounts of data can be leveraged to effectively monitor the environmental consequences of natural disasters, including deforestation, water pollution, and soil erosion. This information can empower communities to implement proactive measures to mitigate the environmental impact of these events.

   C. Promoting Sustainable Disaster Response and Recovery: Deep learning can be employed to develop sustainable disaster response and recovery strategies. For instance, deep learning algorithms can identify suitable areas for reforestation or optimize resource allocation for disaster relief efforts, minimizing the environmental footprint of these activities.

2. **Sustainability Impact**
   A. Enhanced Disaster Preparedness and Resilience: Accurate and timely damage assessments provided by deep learning systems enable communities to better prepare for and respond to natural disasters, leading to reduced environmental harm and promoting sustainable development.

B. Improved Land Use Planning and Development: Deep learning can identify and map areas vulnerable to natural disasters, providing valuable information for informed land use planning and development decisions, thereby minimizing environmental risks.

C. Promoting Sustainable Insurance and Risk Assessment: Deep learning-based disaster insurance and risk assessment models enhance accuracy and sophistication, enabling individuals and businesses to make informed decisions regarding insurance coverage and disaster preparedness planning, ultimately reducing environmental damage.

In conclusion, the application of deep learning to estimate natural disaster damage has the potential to make a substantial positive impact on the environment and sustainability. By providing accurate and timely information, deep learning can empower communities to prepare for, respond to, and recover from natural disasters in a more environmentally responsible manner.

Chapter 6 Project Timeline

Starting in January 2023, this study conducted during the two senior design courses – 499A and 499B, that ended in December 2023. The project's roadmap and task progression are displayed in Figure 12 using a Gantt Chart.

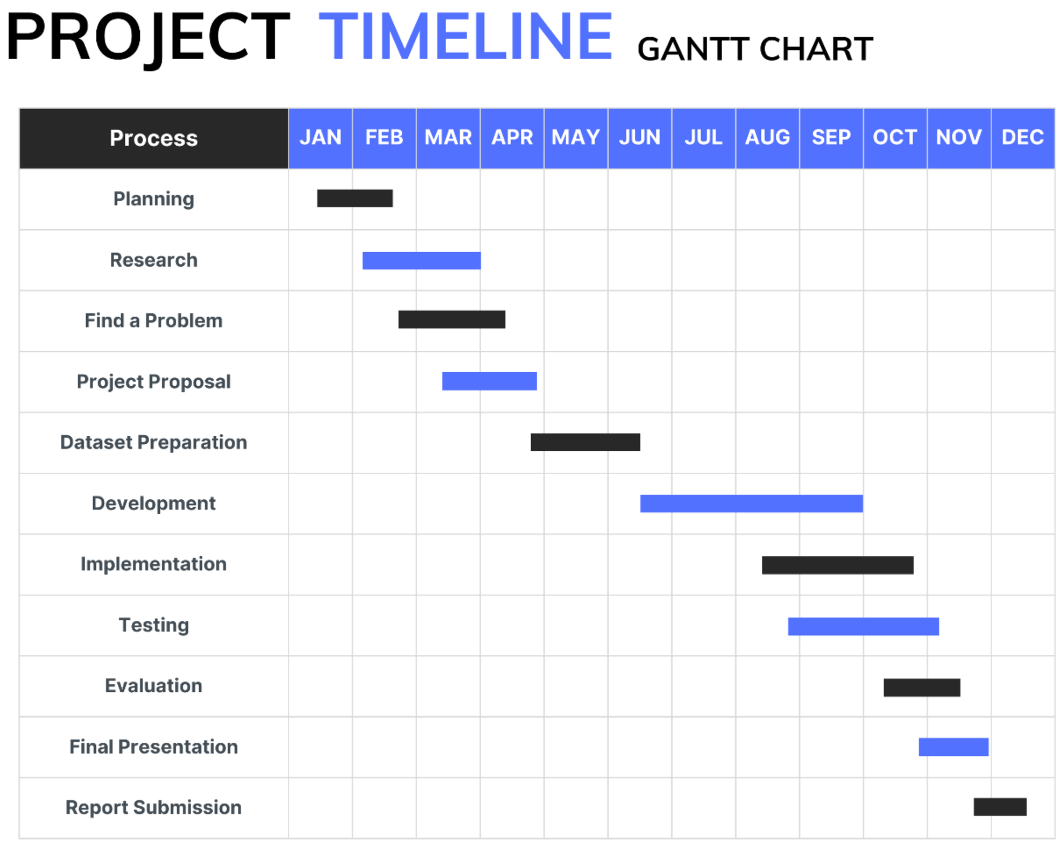


Figure 12. Project Timeline Gantt Chart.

Chapter 7 Conclusions

7.1 Summary

The study utilized state-of-the-art models tailored for diverse tasks in identifying building damage. The YOLOv8 model, known for spotting objects with BBox Loss and Cls Loss, performed well in finding damaged structures. Simultaneously, the ResNet-50 model effectively classified damage types without showing inferior performance compared to the YOLOv8 model. Additionally, the SegFormer model, incorporating LoRA, exhibited commendable expertise in segmentation tasks, aligning closely with the results achieved by the YOLOv8 model. Adjusting input sizes and using unique features like YOLO's rectangle predictions helped create a robust toolkit for understanding building damage. These models offer practical applications in responding to disasters and aiding recovery efforts, marking a significant step forward in the field.

7.2 Future Improvement

Due to the lack of computational resources smaller models had to be used for segmentation which compromised the output accuracy for image segmentation so using deeper or bigger variants of the models is the first thing that can be done in the future. And trying out other classification and segmentation models is another route that can be taken. Also enhancing the dataset through data augmentation techniques is one of the things that must be done to improve the results along the way. These future steps are geared towards enhancing the precision and effectiveness of our approach in evaluating building damage, contributing to the ongoing development of this research area.

References

1. D. Kim, J. Won, E. Lee, K. R. Park, J. Kim, S. Park, H. Yang, and M. Cha, "Disaster assessment using computer vision and satellite imagery: Applications in detecting water-related building damages," Frontiers in Environmental Science, Korea, vol. 10, 2022.
2. R. F. Ahmad, A. S. Malik, A. Qayyum, and N. Kamel, "Disaster monitoring in urban and remote areas using satellite stereo images: A depth estimation approach," IEEE 11th International Colloquium on Signal Processing & Its Applications (CSPA), Kuala Lumpur, Malaysia, pp. 150–155, 2015.

3. S. N. K. B. Amit and Y. Aoki, "Disaster detection from aerial imagery with convolutional neural network," International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC), Surabaya, Indonesia, pp. 239–245, 2017.

4. F. Zhao and C. Zhang, "Building Damage Evaluation from Satellite Imagery using Deep Learning," IEEE 21st International Conference on Information Reuse and Integration for Data Science (IRI), Las Vegas, NV, USA, pp. 82–89, 2020.

5. "resnet50 - Torchvision main documentation," pytorch.org, https://pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html (accessed Jun. 6, 2023).

6. K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, pp. 770-778, 2016.

7. Y. Chandola, J. Virmani, H.S. Bhadauria, and P. Kumar, "End-to-end pre-trained CNN-based computer-aided classification system design for chest radiographs," Deep Learning for Chest Radiographs, pp. 117–140, 2021.

8. J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, pp. 779-788, 2016.

9. J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, pp. 6517-6525, 2017.

10. J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," arXiv:1804.02767 [cs.CV], 2018.

11. A. Bochkovskiy, C. Y. Wang, H. Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” arXiv:2004.10934 [cs.CV], 2020.

12. Ultralytics, “YOLOv5: A state-of-the-art real-time object detection system,” Ultralytics, https://docs.ultralytics.com, 2021 (accessed Nov. 11, 2023).

13. G. Jocher, “Ultralytics YOLOv8 Docs,” Ultralytics, https://docs.ultralytics.com, 2023 (accessed Nov. 11, 2023.

14. C. Y. Wang, H. Y. M. Liao, I H. Yeh, Y. H. Wu, P. Y. Chen, J. W. Hsieh, “CSPNet: A New Backbone that can Enhance Learning Capability of CNN,” arXiv:1911.11929 [cs.CV], 2019.

15. E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, P. Luo, "SegFormer: Simple and Efficient Design for Semantic Segmentation with Transformers," arXiv:2105.15203 [cs.CV], 2021.

16. R. Gupta, R. Hosfelt, S. Sajeev, N. Patel, B. Goodman, J. Doshi, E. Heim, H. Choset, and M. Gaston, “xBD: A Dataset for Assessing Building Damage from Satellite Imagery,” arXiv:1911.09296 [cs.CV], 2019.

17. D. P. Kingma, and J. L. Ba, "Adam: A method for stochastic optimization," arXiv:1412.6980 [cs.LG], 2014.

18. “Semantic segmentation using LoRA,” Hugging Face, https://huggingface.co/docs/peft/task_guides/semantic_segmentation_lora (accessed Nov. 11, 2023.

19. J. Wang, L. Perez, “The Effectiveness of Data Augmentation in Image Classification using Deep Learning,” arXiv:1712.04621 [cs.CV], 2017.

Codes

#YOLO Classification:

```
# Ultralytics YOLO :rocket:, AGPL-3.0 license


import torch

import torchvision


from ultralytics.data import ClassificationDataset, build_dataloader

from ultralytics.engine.trainer import BaseTrainer

from ultralytics.models import yolo

from ultralytics.nn.tasks import ClassificationModel, attempt_load_one_weight

from ultralytics.utils import DEFAULT_CFG, LOGGER, RANK, colorstr

from ultralytics.utils.plotting import plot_images, plot_results

from ultralytics.utils.torch_utils import is_parallel, strip_optimizer, torch_distributed_zero_first



class ClassificationTrainer(BaseTrainer):

    """

    A class extending the BaseTrainer class for training based on a classification model.
    Notes:
        - Torchvision classification models can also be passed to the 'model' argument, i.e.
model='resnet18'.

    Example:

        from ultralytics.models.yolo.classify import ClassificationTrainer

        args = dict(model='yolov8n-cls.pt', data='imagenet10', epochs=3)
        trainer = ClassificationTrainer(overrides=args)
        trainer.train()

    """

    def __init__(self, cfg=DEFAULT_CFG, overrides=None, _callbacks=None):
        """Initialize a ClassificationTrainer object with optional configuration overrides and
callbacks."""
        if overrides is None:
            overrides = {}
        overrides['task'] = 'classify'
        if overrides.get('imgsz') is None:
            overrides['imgsz'] = 224
        super().__init__(cfg, overrides, _callbacks)

    def set_model_attributes(self):
        """Set the YOLO model's class names from the loaded dataset."""
        self.model.names = self.data['names']

    def get_model(self, cfg=None, weights=None, verbose=True):
        """Returns a modified PyTorch model configured for training YOLO."""
        model = ClassificationModel(cfg, nc=self.data['nc'], verbose=verbose and RANK == -1)
        if weights:
            model.load(weights)

        for m in model.modules():
            if not self.args.pretrained and hasattr(m, 'reset_parameters'):
                m.reset_parameters()
            if isinstance(m, torch.nn.Dropout) and self.args.dropout:
                m.p = self.args.dropout  # set dropout
        for p in model.parameters():
            p.requires_grad = True  # for training
        return model

    def setup_model(self):
        """Load, create or download model for any task."""
        if isinstance(self.model, torch.nn.Module):  # if model is loaded beforehand. No setup needed
            return

        model, ckpt = str(self.model), None
        # Load a YOLO model locally, from torchvision, or from Ultralytics assets
        if model.endswith('.pt'):
            self.model, ckpt = attempt_load_one_weight(model, device='cpu')
            for p in self.model.parameters():
                p.requires_grad = True  # for training
        elif model.split('.')[-1] in ('yaml', 'yml'):
            self.model = self.get_model(cfg=model)
        elif model in torchvision.models.__dict__:
            self.model = torchvision.models.__dict__[model](weights='IMAGENET1K_V1' if
self.args.pretrained else None)
        else:
            FileNotFoundError(f'ERROR: model={model} not found locally or online. Please check
model name.')
```

```
        ClassificationModel.reshape_outputs(self.model, self.data['nc'])

        return ckpt

    def build_dataset(self, img_path, mode='train', batch=None):
        """Creates a ClassificationDataset instance given an image path, and mode (train/test etc.)."""
        return ClassificationDataset(root=img_path, args=self.args, augment=mode == 'train',
prefix=mode)

    def get_dataloader(self, dataset_path, batch_size=16, rank=0, mode='train'):
        """Returns PyTorch DataLoader with transforms to preprocess images for inference."""
        with torch_distributed_zero_first(rank):  # init dataset *.cache only once if DDP
            dataset = self.build_dataset(dataset_path, mode)

        loader = build_dataloader(dataset, batch_size, self.args.workers, rank=rank)
        # Attach inference transforms
        if mode != 'train':
            if is_parallel(self.model):
                self.model.module.transforms = loader.dataset.torch_transforms
            else:
                self.model.transforms = loader.dataset.torch_transforms
        return loader


    def preprocess_batch(self, batch):
        """Preprocesses a batch of images and classes."""
        batch['img'] = batch['img'].to(self.device)
        batch['cls'] = batch['cls'].to(self.device)
        return batch


    def progress_string(self):
        """Returns a formatted string showing training progress."""
        return ('\n' + '%11s' * (4 + len(self.loss_names))) % \
            ('Epoch', 'GPU_mem', *self.loss_names, 'Instances', 'Size')


    def get_validator(self):
        """Returns an instance of ClassificationValidator for validation."""
        self.loss_names = ['loss']
        return yolo.classify.ClassificationValid
```

#ResNet-50 Classification:

```
import cv2
import os

import numpy as np

import torch

import torchvision as tv

from PIL import Image

from torchvision import transforms

from torchvision.models.detection.faster_rcnn import FastRCNNPredictor

from torchvision.models.detection import FasterRCNN

from torchvision.models.detection.rpn import AnchorGenerator

from torch.utils.data import DataLoader, Dataset

from torch.utils.data.sampler import SequentialSampler

from matplotlib import pyplot as plt



class CustomDataset(Dataset):


    def __init__(self, root_dir):

        self.root_dir = root_dir

        self.images = os.listdir(os.path.join(root_dir, 'images'))

        self.image_dir = os.path.join(root_dir, 'images')
        self.labels_dir = os.path.join(root_dir, 'labels')



    def __len__(self) -> int:
        return len(self.images)

    def __getitem__(self, index: int):
        image_name = self.images[index]
        image = Image.open(os.path.join(self.image_dir, image_name)).convert('RGB')
        width, height = image.size
        # image = cv2.imread(os.path.join(self.image_dir, image_name), cv2.IMREAD_COLOR)
        # image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB).astype(np.float32)
        # image = cv2.resize(image, (1024, 1024))
        # print(image.shape)
        # image = image / 255.0
        with open(os.path.join(self.labels_dir, image_name[:-4] + '.txt')) as f:
            lines = f.readlines()
        boxes, targets = [], []
        for line in lines:
            target, x, y, w, h = line.split()
```

```
            targets.append(int(target) + 1)
            boxes.append([(float(x) - float(w)/2)*width, (float(y) - float(h)/2)*height, (float(x) + float(w)/2)*width , (float(y) + float(h)/2)*height])

        if not boxes:
            return None, None
        target = {}
        target['boxes'] = torch.tensor(boxes)
        target['labels'] = torch.tensor(targets)
        return transforms.ToTensor()(image), target

class Averager:
    def __init__(self):
        self.current_total = 0.0
        self.iterations = 0.0

    def send(self, value):
        self.current_total += value
        self.iterations += 1

    @property
    def value(self):
        if self.iterations == 0:
            return 0
        else:
            return 1.0 * self.current_total / self.iterations

    def reset(self):
        self.current_total = 0.0
        self.iterations = 0.0



def custom_collate(data):
    return tuple(zip(*data))



custom_dataset_train_path, custom_dataset_val_path, batch_size = r'selected\train', r'selected\val', 1
custom_dataset_train = CustomDataset(root_dir=custom_dataset_train_path)
custom_dataset_val = CustomDataset(root_dir=custom_dataset_val_path)
data_loader_train = DataLoader(custom_dataset_train, batch_size=batch_size, shuffle=True, collate_fn=custom_collate, pin_memory=True if torch.cuda.is_available else False)
```

```
data_loader_val = DataLoader(custom_dataset_val, batch_size=batch_size, shuffle=True,
collate_fn=custom_collate, pin_memory=True if torch.cuda.is_available else False)



model = tv.models.detection.fasterrcnn_resnet50_fpn(pretrained = True)
num_classes = 6
in_features = model.roi_heads.box_predictor.cls_score.in_features
model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)
device = 'cuda' if torch.cuda.is_available else 'cpu'
model.to(device)
params = [p for p in model.parameters() if p.requires_grad]
optimizer = torch.optim.SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)
lr_scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)
# lr_scheduler = None

num_epochs = 150
loss_hist = Averager()
itr = 1

for epoch in range(num_epochs):
    loss_hist.reset()
```

```
for images, target in data_loader_train:
    images = list(image.to(device) for image in images)
    target = [{k: v.to(device) for k, v in t.items()} for t in target]
    loss_dict = model(images, target)


    losses = sum(loss for loss in loss_dict.values())
    loss_value = losses.item()


    loss_hist.send(loss_value)


    optimizer.zero_grad()
    losses.backward()
    optimizer.step()


    if itr % 50 == 0:
        print(f"Iteration #{itr} loss: {loss_value}")


    itr += 1


# update the learning rate
```

```
    if lr_scheduler is not None:

        lr_scheduler.step()


    print(f"Epoch #{epoch} loss: {loss_hist.value}")


torch.save(model.state_dict(), 'test.pb')

```

#SegFormer Segmentation:

```
import torch

import torch.nn as nn

import torch.nn.functional as F


from mmseg.core import add_prefix

from mmseg.ops import resize

from .. import builder

from ..builder import SEGMENTORS

from .base import BaseSegmentor



@SEGMENTORS.register_module()
```

```
class EncoderDecoder(BaseSegmentor):
    """Encoder Decoder segmentors.


    EncoderDecoder typically consists of backbone, decode_head, auxiliary_head.
    Note that auxiliary_head is only used for deep supervision during training,
    which could be dumped during inference.
    """


    def __init__(self,
                 backbone,
                 decode_head,
                 neck=None,
                 auxiliary_head=None,
                 train_cfg=None,
                 test_cfg=None,
                 pretrained=None):
        super(EncoderDecoder, self).__init__()
        self.backbone = builder.build_backbone(backbone)
        if neck is not None:
            self.neck = builder.build_neck(neck)
        self._init_decode_head(decode_head)
        self._init_auxiliary_head(auxiliary_head)

        self.train_cfg = train_cfg
        self.test_cfg = test_cfg

        self.init_weights(pretrained=pretrained)

        assert self.with_decode_head

    def _init_decode_head(self, decode_head):
        """Initialize ``decode_head``"""
        self.decode_head = builder.build_head(decode_head)
        self.align_corners = self.decode_head.align_corners
        self.num_classes = self.decode_head.num_classes

    def _init_auxiliary_head(self, auxiliary_head):
        """Initialize ``auxiliary_head``"""
        if auxiliary_head is not None:
            if isinstance(auxiliary_head, list):
                self.auxiliary_head = nn.ModuleList()
                for head_cfg in auxiliary_head:
                    self.auxiliary_head.append(builder.build_head(head_cfg))
            else:
                self.auxiliary_head = builder.build_head(auxiliary_head)

    def init_weights(self, pretrained=None):
        """Initialize the weights in backbone and heads.

        Args:
            pretrained (str, optional): Path to pre-trained weights.
                Defaults to None.
        """

        super(EncoderDecoder, self).init_weights(pretrained)
        self.backbone.init_weights(pretrained=pretrained)
        self.decode_head.init_weights()
        if self.with_auxiliary_head:
            if isinstance(self.auxiliary_head, nn.ModuleList):
                for aux_head in self.auxiliary_head:
```

```
            aux_head.init_weights()
        else:
            self.auxiliary_head.init_weights()

def extract_feat(self, img):
    """Extract features from images."""
    x = self.backbone(img)
    if self.with_neck:
        x = self.neck(x)
    return x

def encode_decode(self, img, img_metas):
    """Encode images with backbone and decode into a semantic segmentation
    map of the same size as input."""
    x = self.extract_feat(img)
    out = self._decode_head_forward_test(x, img_metas)
    out = resize(
        input=out,
        size=img.shape[2:],
        mode='bilinear',
        align_corners=self.align_corners)
    return out


def _decode_head_forward_train(self, x, img_metas, gt_semantic_seg):
    """Run forward function and calculate loss for decode head in
    training."""
    losses = dict()
    loss_decode = self.decode_head.forward_train(x, img_metas,
                                                 gt_semantic_seg,
                                                 self.train_cfg)

    losses.update(add_prefix(loss_decode, 'decode'))
    return losses


def _decode_head_forward_test(self, x, img_metas):
    """Run forward function and calculate loss for decode head in
    inference."""
    seg_logits = self.decode_head.forward_test(x, img_metas, self.test_cfg)
    return seg_logits
```

```
def _auxiliary_head_forward_train(self, x, img_metas, gt_semantic_seg):
    """Run forward function and calculate loss for auxiliary head in
    training."""
    losses = dict()
    if isinstance(self.auxiliary_head, nn.ModuleList):
        for idx, aux_head in enumerate(self.auxiliary_head):
            loss_aux = aux_head.forward_train(x, img_metas,
                                              gt_semantic_seg,
                                              self.train_cfg)
            losses.update(add_prefix(loss_aux, f'aux_{idx}'))
    else:
        loss_aux = self.auxiliary_head.forward_train(
            x, img_metas, gt_semantic_seg, self.train_cfg)
        losses.update(add_prefix(loss_aux, 'aux'))

    return losses


def forward_dummy(self, img):
    """Dummy forward function."""
    seg_logit = self.encode_decode(img, None)
```

```
        return seg_logit
```

#YOLO Segmentation:

```
# Ultralytics YOLO :rocket:, AGPL-3.0 license


from copy import copy



from ultralytics.models import yolo

from ultralytics.nn.tasks import SegmentationModel

from ultralytics.utils import DEFAULT_CFG, RANK

from ultralytics.utils.plotting import plot_images, plot_results




class SegmentationTrainer(yolo.detect.DetectionTrainer):
    """
    A class extending the DetectionTrainer class for training based on a segmentation model.


    Example:
        from ultralytics.models.yolo.segment import SegmentationTrainer


        args = dict(model='yolov8n-seg.pt', data='coco8-seg.yaml', epochs=3)
        trainer = SegmentationTrainer(overrides=args)
        trainer.train()

    """


    def __init__(self, cfg=DEFAULT_CFG, overrides=None, _callbacks=None):
        """Initialize a SegmentationTrainer object with given arguments."""
        if overrides is None:
            overrides = {}
        overrides['task'] = 'segment'
        super().__init__(cfg, overrides, _callbacks)


    def get_model(self, cfg=None, weights=None, verbose=True):
        """Return SegmentationModel initialized with specified config and weights."""
        model = SegmentationModel(cfg, ch=3, nc=self.data['nc'], verbose=verbose and RANK == -
1)
        if weights:
            model.load(weights)
        return model

    def get_validator(self):
        """Return an instance of SegmentationValidator for validation of YOLO model."""
        self.loss_names = 'box_loss', 'seg_loss', 'cls_loss', 'dfl_loss'
        return yolo.segment.SegmentationValidator(self.test_loader, save_dir=self.save_dir,
args=copy(self.args))

    def plot_training_samples(self, batch, ni):
        """Creates a plot of training sample images with labels and box coordinates."""
        plot_images(batch['img'],
                    batch['batch_idx'],
                    batch['cls'].squeeze(-1),
                    batch['bboxes'],
                    batch['masks'],
                    paths=batch['im_file'],
                    fname=self.save_dir / f'train_batch{ni}.jpg',
                    on_plot=self.on_plot)

    def plot_metrics(self):
        """Plots training/val metrics."""

        plot_results(file=self.csv, segment=True, on_plot=self.on_plot)  # save results.png
```

#Convert json to Yolo:

```
import os

import json

import cv2

import shutil

from shapely.geometry import Polygon

from shapely.wkt import loads


LABEL_DIR = r'dataset\train\jsons'

SAVE_DIR = r'dataset\train\labels'

IMG_DIR = r'dataset\train\images'

BASE_DIR = r'dataset\train\all_images'

os.makedirs(SAVE_DIR, exist_ok=True)

os.makedirs(IMG_DIR, exist_ok=True)


damage_dict = {

    'no-damage' : 0,
    'minor-damage' : 1,

    'major-damage' : 2,

    'destroyed' : 3,

    'un-classified' : 4

}


for json_name in os.listdir(LABEL_DIR):

    if 'post_disaster' in json_name:

        print(f'Processing Json -> {json_name}')

        with open(os.path.join(LABEL_DIR, json_name), 'r') as f:

            data = json.load(f)

        metadata = data['metadata']

        xys = data['features']['xy']

        if len(xys) != 0:

            img = cv2.imread(os.path.join(r'dataset\train\all_images', json_name[:-5] + '.png'))

            shutil.copyfile(os.path.join(BASE_DIR, json_name[:-5] + '.png'), os.path.join(IMG_DIR,
json_name[:-5] + '.png'))

            lines = []

            for xy in xys:

                coordinates = list(loads(xy['wkt']).exterior.coords)
                max_x, min_x = round(max(coordinates, key=lambda x: x[0])[0]), round(min(coordinates, key=lambda x: x[0])[0])

                max_y, min_y = round(max(coordinates, key=lambda x: x[1])[1]), round(min(coordinates, key=lambda x: x[1])[1])

                img = cv2.rectangle(img, (min_x, min_y), (max_x, max_y), (255,0,0), 3)

                line = str(damage_dict[xy['properties']['subtype']])

                x, y = round((max_x + min_x) / 2), round((max_y + min_y)/2)

                width, height = max_x - min_x, max_y - min_y


                line += f" {str(x/metadata['original_width'])} {str(y/metadata['original_height'])} {str(width/metadata['original_width'])} {str(height/metadata['original_height'])}"

                lines.append(line)


            with open(os.path.join(SAVE_DIR, json_name[:-5] + '.txt'), 'w') as file:

                for line in lines:

                    file.write(line + '\n')


        # cv2.imwrite(json_name[:-5] + '.png', img)

        # break
```

```
#Segmented Classified Output Inference:

import torch

import time

import numpy as np

import transformers

import matplotlib.pyplot as plt

from PIL import Image

from transformers import Trainer, TrainingArguments, EarlyStoppingCallback

from transformers import AutoImageProcessor

from transformers import AutoModelForSemanticSegmentation, TrainingArguments


from peft import PeftConfig, PeftModel


from ultralytics import YOLO




img_path = r'hurricane-florence_00000173_post_disaster.png'

# Load a model

yolo_model = YOLO(r'D:\AI Models\SAT PROJECT (YOLO Cls)\best.pt')  # pretrained
YOLOv8n model

result = yolo_model.predict(img_path, save=False, conf=0.1, imgsz=1024, boxes=True)[0]
```

```
boxes = result.boxes  # Boxes object for bbox outputs

classes  = boxes.cls

print(boxes.conf)

cords = boxes.xywh




# Red: (255, 0, 0)

# Green: (0, 255, 0)

# Blue: (0, 0, 255)

# Yellow: (255, 255, 0)

# Purple: (128, 0, 128)


checkpoint = "nvidia/mit-b0"


id2label = {0: "background", 1: "building"}

label2id = {label: idx for idx, label in id2label.items()}

config = PeftConfig.from_pretrained('mit-b0-building-damage-lora')

model = AutoModelForSemanticSegmentation.from_pretrained(

    checkpoint, id2label=id2label, label2id=label2id, ignore_mismatched_sizes=True

)
```

```
inference_model = PeftModel.from_pretrained(model, 'mit-b0-building-damage-lora')

image = Image.open(img_path)

image_processor = AutoImageProcessor.from_pretrained(checkpoint, do_reduce_labels=False)
encoding = image_processor(image.convert("RGB"), return_tensors="pt")

with torch.no_grad():
    outputs = inference_model(pixel_values=encoding.pixel_values)
    logits = outputs.logits

upsampled_logits = torch.nn.functional.interpolate(
    logits,
    size=image.size[::-1],
    mode="bilinear",
    align_corners=False,
)

pred_seg = upsampled_logits.argmax(dim=1)[0]
```

```
for i,cord in enumerate(cords):
    print(cord)
    for x in range(int(cord[0]), int(cord[0]) + int(cord[2])):
        if int(cord[0]) + int(cord[2]) >= 1024:
            continue
        for y in range(int(cord[1]), int(cord[1]) + int(cord[3])):
            if int(cord[1]) + int(cord[3]) >=1024:
                continue
            if pred_seg[x][y] == 1:
                pred_seg[x][y] = classes[i]



f, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
# ax1.imshow(image)
# ax1.set_title('Image')
# ax2.imshow(pred_seg)
# ax2.set_title('Segmentation')
# plt.show()
```

```
color_seg = np.zeros((pred_seg.shape[0], pred_seg.shape[1], 3), dtype=np.uint8)



# color_seg[pred_seg == 1, :] = np.array([255, 0, 0])


for i in range(1024):
    for j in range(1024):
        if pred_seg[i][j] == 0:
            color_seg[i][j] = [255, 0, 0]
        elif pred_seg[i][j] == 1:
            color_seg[i][j] = [0, 255, 0]
        elif pred_seg[i][j] == 2:
            color_seg[i][j] = [0, 0, 255]
        elif pred_seg[i][j] == 3:
            color_seg[i][j] = [255, 255, 0]
        elif pred_seg[i][j] == 4:
            color_seg[i][j] = [128, 0, 128]
color_seg = color_seg[..., ::-1]  # convert to BGR
```

```
img = np.array(image) * 0.5 + color_seg * 0.5  # plot the image with the segmentation map

img = img.astype(np.uint8)

print(img)



# plt.figure(figsize=(15, 10))

plt.imshow(img)

plt.show()
```