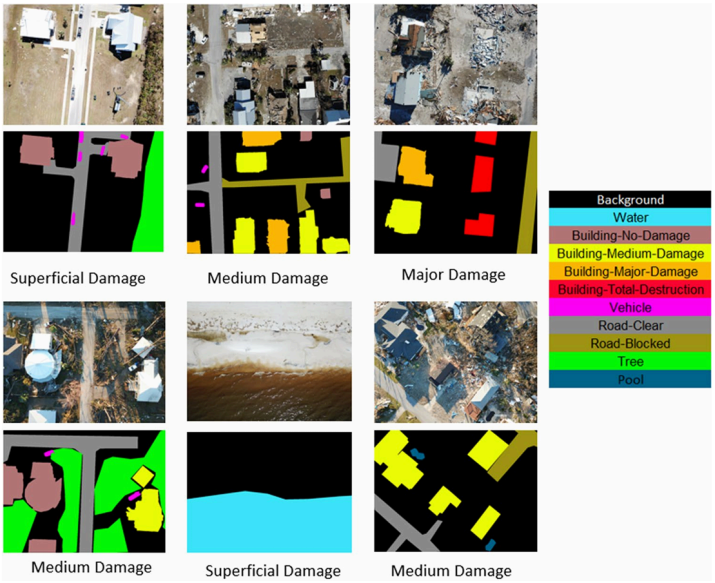
Background

My learning and development around semantic image segmentation started in summer 2025 during attendance of Remote Sensing class at Beaver Works Summer Institute@ MIT.

The course project is around a simulated hurricane scenario, aptly named “Hurricane Beaver”. My classmates formed different teams and used different types of technology to solve challenges including planning, logistics, operation, and public info.

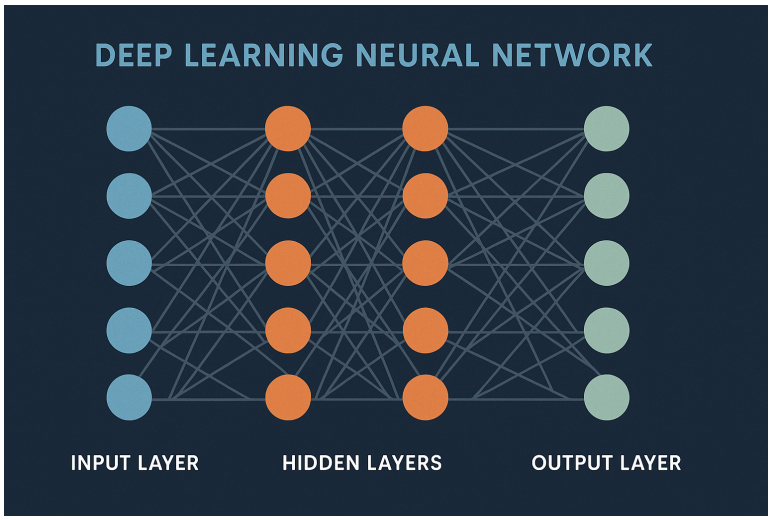
As part of the Operations team, our mission was to design and implement a deep learning model to identify infrastructure damage from post-disaster aerial images.



Picture from Rahnemoonfar, M., Chowdhury, T. & Murphy, R. RescueNet: A High Resolution UAV Semantic Segmentation Dataset for Natural Disaster Damage Assessment. *Sci Data* **10**, 913 (2023).

Examples of output from the model are illustrated in the picture above. The semantic segmentations from humans are referred to as “Ground Truth”. Different objects and their

boundaries/area as well as damage levels are masked. Such information is important for any post-disaster operations. We used the images from RescueNet to train our model.



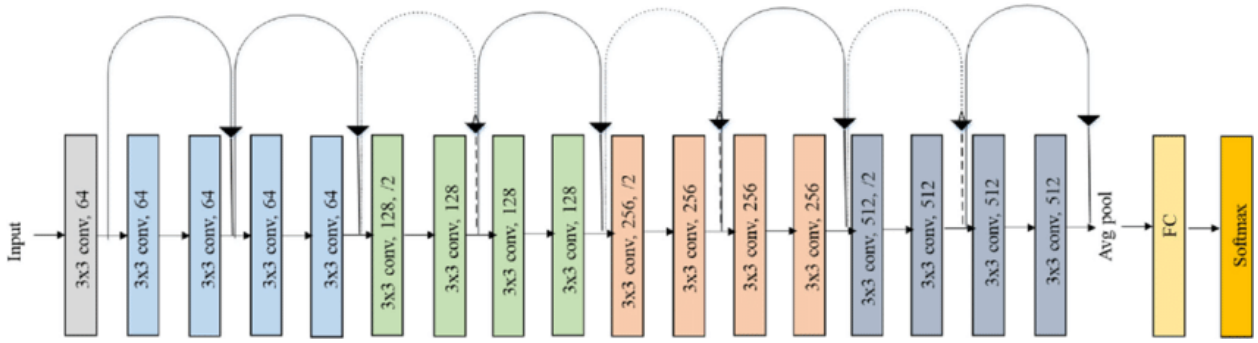
Deep learning was the type of machine learning my team and I used to create our model. The model is a multi-layer neural network made up of millions of yes/no switches called neurons (just like our brains). Each neuron looks for small details and clues that could help explain why the initial leads to the result. Together, they combine to make a decision – for example: “this fruit is probably an apple”. Training means adjusting how strongly each of those little neurons reacts, so the overall prediction gets closer to the right answer every time.

After learning from the training data, you give the model new images it’s never seen before. If it can still tell the difference between damaged and undamaged buildings, that means it’s learned the pattern, not just memorized the pictures. This step checks whether the model can generalize; that’s what makes it useful in the real world.

If the model gets confused (say, by shadows or reflections), you might add more examples or tweak the settings. This process — test, adjust, retrain — continues until the model performs well enough for real-world use.

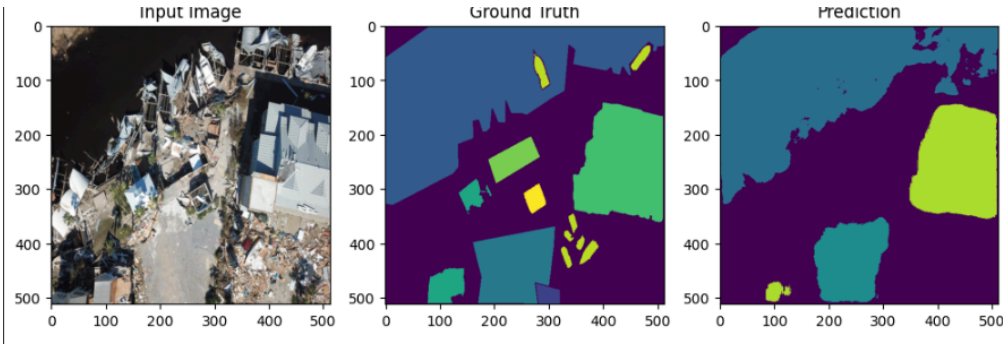
Initial Work

During the class, I used the free version of Google Colab. The limited computing resource together with the short time in preparing for this project led to the decision of using a lighter model, i.e., U-Net with ResNet18 as the backbone.



ResNet-18 Architecture

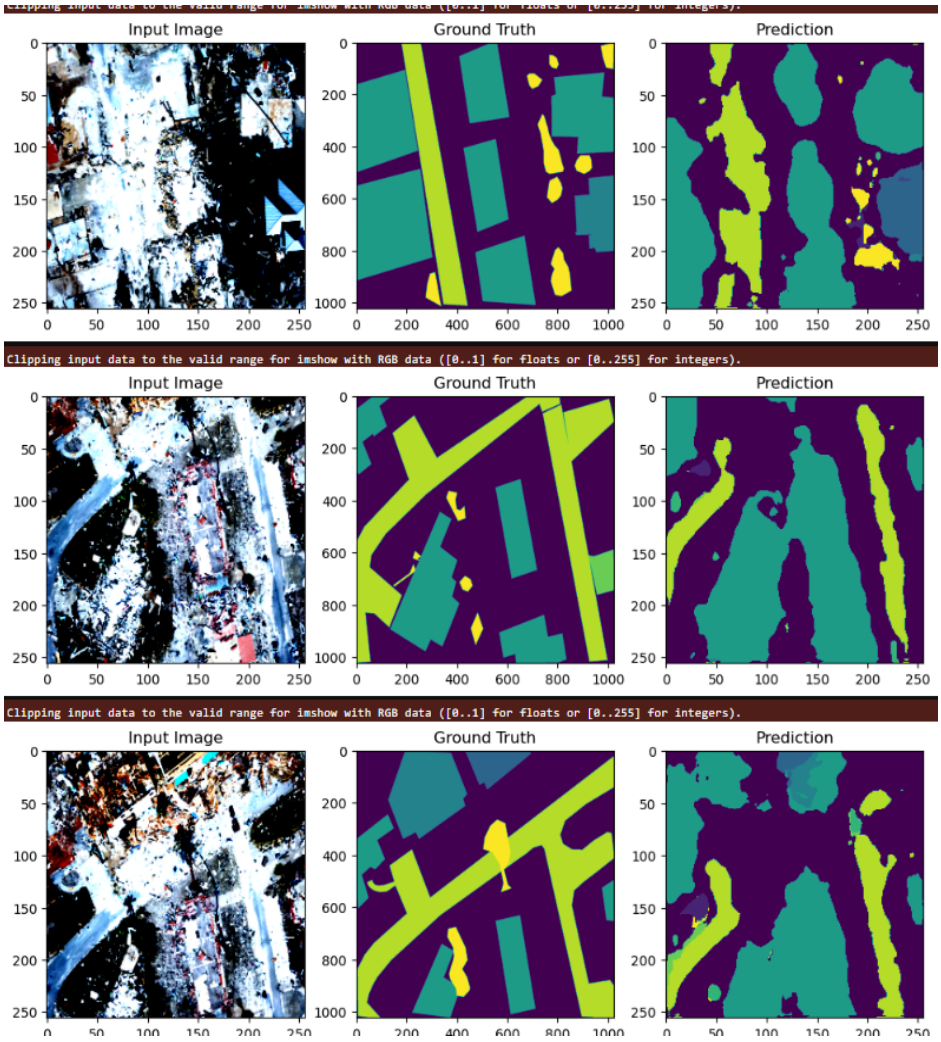
ResNet18 is a convolutional neural network that is 18 layers deep. ResNet stands for **Residual Network**. Its innovation was introducing **skip connections**, which let information flow more freely between neural network's layers. This helps very deep models avoid the "vanishing gradient" problem — where early layers forget what they learned as the network gets deeper. Think of it as a shortcut highway that keeps features from earlier layers (like "roof edges" or "water textures") alive all the way to the end.



At a Flat learning rate of 0.0001, the model picked up on larger object shapes but didn't classify them well. It also struggled a bit with recognizing finer features.

Although there was limited time for the model training in this project, I applied hyperparameter tuning to improve the results together with my teammates.

- We initialized the model with pre-trained weights from the ImageNet dataset. This provided our model a good starting point as it learned low level features quickly
- Cross Entropy Loss Function - The more confident the model is in a wrong answer, the higher the penalty ○ Optimizer
- We used AdamW (adaptive optimizer), which shrinks training weights of the model to prevent the model overfitting on training data
- Learning Rate Scheduler - Decreases learning rate of model as epochs go on - as a model trains further it starts to focus and hone in more and more on details



The improvement of semantic segmentation is shown above. However, again, due to the limitation of the model and the time for training/tuning, our model only reached 70% accuracy (accuracy = number of pixels matched with the ground truth/number of total pixels).

After the program I wondered – what if I could continue this spark in deep learning that I learned in BWSI?

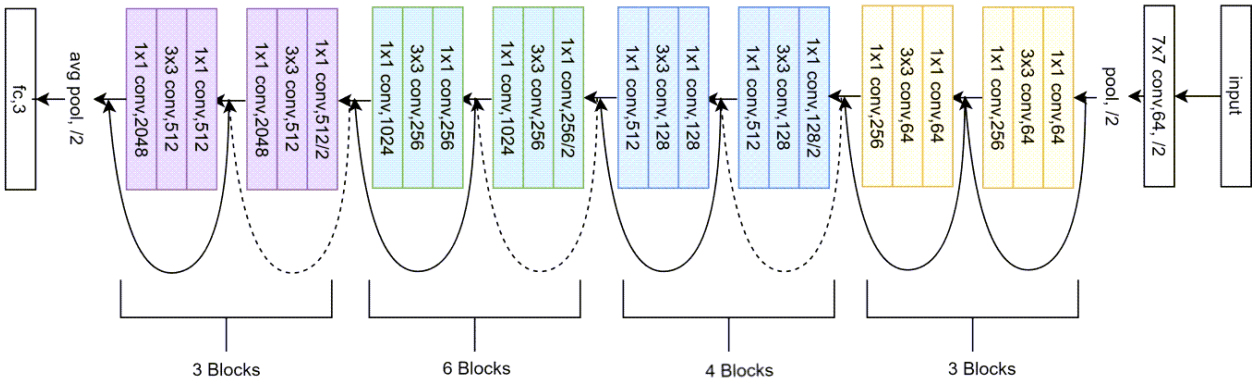
Further work

I continued designing a model with the same dataset. I kept applying optimizations as I continued to explore and dive deeper into the realm of machine learning through this project. The model's evolution timeline can be found in this repo in a Readme called Version_History.md.

With my parent's generous "sponsorship", I was able to access Colab's paid plan. Being able to access more powerful GPUs and have more compute units allows me to try more sophisticated models.

I first rebuilt the whole pipeline. I wrote scripts to automatically download, unzip, and organize the various files of the dataset.

ResNet50



ResNet50 Architecture

I switched from ResNet18 to ResNet50, which is deeper (50 layers), more complex, and capable of capturing more intricate patterns (lines, corners, textures, and object shapes) in the images.

DeeplabV3+

Due to the nature of semantic segmentation for my dataset of post-disaster aerial images, I needed a model which can see both fine local details (like the edge of a damaged roof) and large-scale context (like surrounding floodwater or nearby roads).

I utilized DeeplabV3+, which is a deep learning model created by Google Research for semantic segmentation. It helps classifying every pixel in a scene to a certain type with following features:

Atrous (dilated) convolutions - The model can “see” both nearby pixels and those farther away, helping it recognize context like flooded streets or partially collapsed buildings.

ASPP (Atrous Spatial Pyramid Pooling) – The model analyzes the same image at several scales simultaneously (zoomed in and zoomed out views), e.g., one aerial image contains both small vehicles and large water bodies.

Decoder refinement – A “decoder” stage that sharpens object boundaries, so the output masks have clean edges instead of blurry outlines.

Together these 3 features make DeepLabV3+ good at detecting and classifying large and finer features in images.

Other Techniques

Here are optimization techniques that I evaluated and applied in my model training.

- AdamW optimizer, this stayed the same.
- Hybrid loss functions: combines different ways of measuring error so both large and rare objects are learned correctly:

Final hybrid loss composition (v3–v10):

Component	Description	Weight
Cross-Entropy (weighted)	base per-pixel classification loss, weighted by inverse √(class frequency)	0.35
Lovasz Loss	focuses on IoU optimization, improves boundary sharpness	0.30
Focal CE	emphasizes hard-to-classify pixels; per-class γ	0.20
OHEM (Online Hard Example Mining)	selects hardest 20% of pixels	0.15

- Rarity-Aware Sampling - New sampler that prioritizes giving the model more images that contain rare object classes.
- EMA (Exponential Moving Average) – smooths out short-term fluctuations for stability
- TTA (Test-Time Augmentation) – runs the model on slightly scaled or flipped versions of each image, then averages the results for more reliable predictions. Here are the transformations used:
    - Normalization of image, basically converting the input image into an image the model’s backbone (Resnet50 in our case) can understand

- Crops image into 768x768 snippets of image, if dimensions aren't enough (say there's a 600x600 image), padding (blank image spots the model ignores) is added to reach that minimum image size
- Randomly rotates the image with a set probability
- Random chance to flip the image vertically
- Random chance to flip the image horizontally
- Adds random color jitter and motion blur. The color jitter + motion blur helps the model train on images where the drone taking the photos might not have the best angle, lighting, or position
- These transformations all create diversity through making different versions of each image in our dataset, effectively creating tons of new images for our model to train on

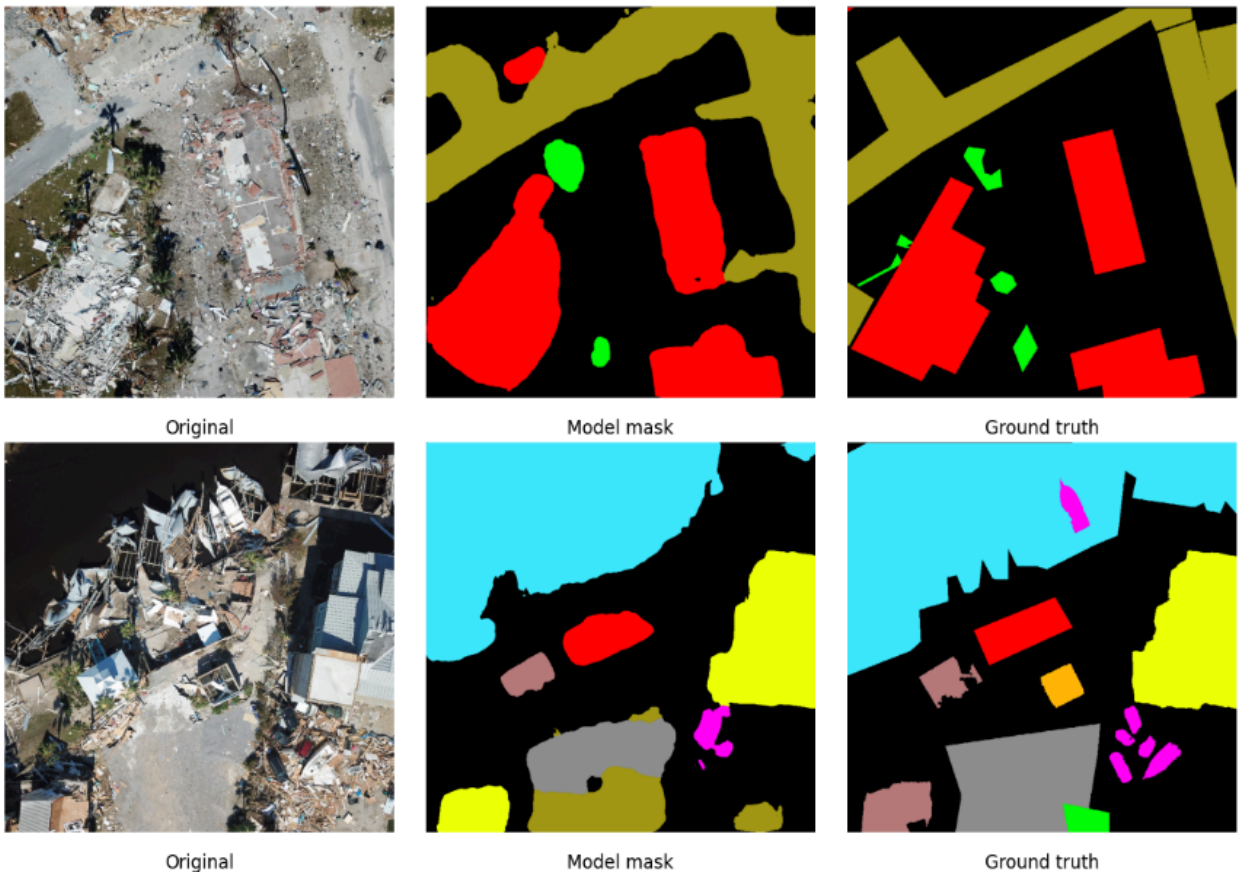
Training Details and Results

I used an A100 GPU and a resnet50 backbone for deelabv3+. AdamW was the optimizer, learning rate was 1e-4, weight decay was 1e-4, gradient clipping max_norm was 1.0, and learning rate scheduler was cosine annealing with 1-epoch warm up.

The result is a **color-coded mask** overlayed on the original image, showing the full landscape of destruction at pixel-level precision. Below is the classes based on pixel color, and below that are the results:

```
COLOR_MAP = {
   (0, 0, 0): 0,  # background
   (61, 230, 250): 1,  # water
   (180, 120, 120): 2, # building-no-damage
   (235, 255, 7): 3,   # building-medium-damage
   (255, 184, 6): 4,   # building-major-damage
   (255, 0, 0): 5,     # building-total-destruction
   (255, 0, 245): 6,   # vehicle
   (140, 140, 140): 7, # road-clear
   (160, 150, 20): 8,  # road-blocked
   (4, 250, 7): 9,     # tree
   (0, 101, 140): 10,  # pool
}
```

Results:



Examples of the semantic segmentation of original images are shown above.

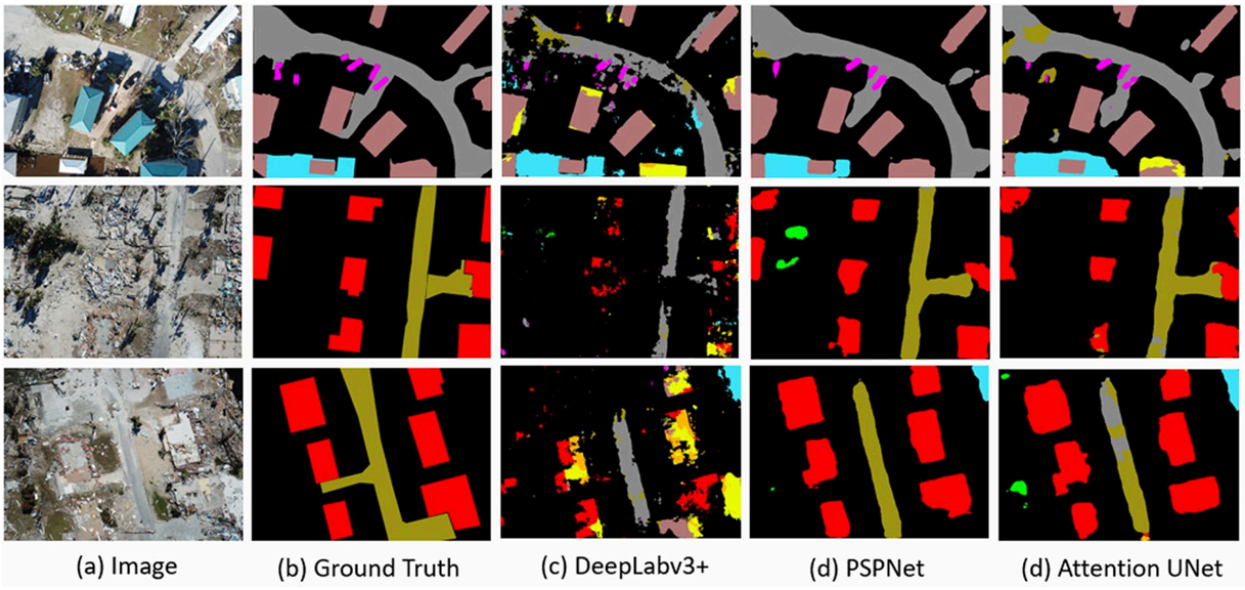
Pixel Accuracy	≈ 90 %
Validation mIoU	≈ 72.4 %
Strong classes	Water & Buildings (no/medium damage ≈ 0.82 IoU)
Weaker classes	Vehicles/Road-blocked ≈ 0.45–0.55 IoU

Note that while the overall accuracy is around 90%, this model still needs improvement especially around fine granularity classification, e.g., identifying small vehicles and assessing blockage on the road. This leaves room for me to explore in future works.

Future work

DeeplabV3+'s true performance?

In Rahnemoonfar, et.al's RescueNet paper, the combo of DeepLabv3+ and ResNet100 had a low performance and barely captured large objects' boundaries.



This observation seems to be different from my test results shown earlier. Such a difference is especially interesting since my training uses ResNet50 instead of ResNet100. It is possible that the performance improvement is from my other optimization efforts from sampling to loss function composition.

I plan to replace ResNet50 with ResNet100 in my training while keeping all other setups intact. This can be a good exercise to see how different parts of deep learning can play in the final results.

Attention Mechanism

While the UNet performs less effectively in my training compared with DeepLabv3+, the Attention UNet performs significantly better in Rahnemoonfar, et.al's study.

Attention, in the context of image segmentation, is a way to focus on the relevant activations during training. This reduces the computational resources wasted on irrelevant activations, providing the network with better generalization power. The model acts like it can pay "attention" to certain parts of the image.

Down to the implementation, the interaction of different layers to adjust the “Soft Attention” is intriguing.

After I switch to ResNet100 as the backbone in my training, I plan to try the Attention UNet and see how powerful the “Attention” Mechanism is in deep learning.