Semantic segmentation applied in the context of natural disasters

Trying to accurately identify and segment common large and small assets in residential areas of Houston following Hurricane Harvey.

Nicolo' Giacopelli
Master in DSBA
ESSEC & CentraleSupélec
Paris, France
B00794705@essec.edu

Pauline Michel
Master in DSBA
ESSEC & CentraleSupélec
Paris, France
b00730198@essec.edu

1 Motivation and Problem Definition

In August 2017, Hurricane Harvey, a devastating Category 4 hurricane, hit Texas and Louisiana, causing significant damage and loss of life. Resulting in $125 billion in damages, it is one of the most costly natural disasters in US history. In an era when humanity is facing more climate-related natural disasters, there is a clear need for better ways to monitor and assess ground conditions before, during and after such events. The objective of the exercise is to develop efficient segmentation models to identify residential property features and neighborhood assets after a major flooding event.

First of all, it is worth recalling what semantic segmentation is. It refers to a technique in computer vision that involves assigning a semantic label, such as 'car' or 'road', to each pixel in an image. This allows the computer to understand the structure and meaning of an image at a pixel-level, as opposed to just recognizing an object in the image as a whole. This difference is illustrated in Figure 1 below. The output of semantic segmentation is therefore a label map, where each pixel is labeled with the class of the object it belongs to.

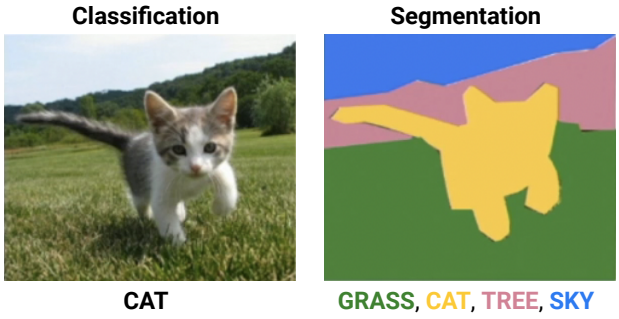

Figure 1. An example from the provided dataset demonstrating the desired dense mask

In the context of natural disasters such as Hurricane Harvey, the objective of semantic segmentation is therefore to develop an accurate and dense segmentation model for post-flood aerial imagery, in order to identify property attributes in the Houston area and autonomously generate a post-flood map (Figure 2 displays the desired output). This information can assist in a variety of ways, such as assessing the extent of damage, identifying areas most in need of assistance, and guiding the deployment of resources for recovery and rebuilding efforts. In addition, this information can be used to improve the understanding of ground conditions before, during and after such events, thereby improving preparedness, response and recovery mechanisms.

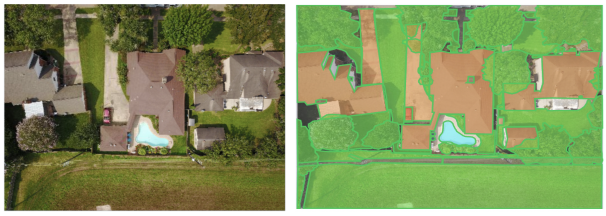

Figure 2. An example from the provided dataset demonstrating the expected dense mask

2 Methodology

The whole process can be broken down into 5 main stages: Data exploration, Preprocessing, Development of architectures, Model training, and Mask generation. In this Methodology section, we will focus on the first three steps. We will discuss the next two stages in the Evaluation section.

1) Data exploration

First, it is essential to explore and understand the data, in order to best figure out the pre-processing of the data. The provided Hurricane Harvey dataset contains 374 drone images of residential neighborhoods in Houston. For each, a dense segmentation mask has been generated for the classes that we'll detail in a moment. Each mask was further vectorized to provide additional representations for modeling, visualization, and review. Of the 374 drone images, 299 serve as train sets and 75 as test sets. This

means that the semantic segmentation of these 75 images is not communicated to us: it is the masks that we must predict. Note that the 299 training data should be divided into two parts: training and validation, to avoid overfitting.

There is also a second dataset named Hurricane Harvey Synthetic. which consists of a synthetic dataset with 2,093 images and 156,933 annotations. This dataset can be used to pre-train a large model and then fine-tune on Hurricane Harvey dataset.

Regarding the semantic segmentation, it is broken down into 27 given classes: *Boat, Bridge, Chimney, Dense Vegetation / Forest, Flooded, Garbage Bins, Grass, Industrial Site, Parking Area, Parking Area - Commercial, Power Lines & Cables, Property Roof, Road / Highway, Satellite Antenna, Secondary Structure, Solar Panels, Sports Complex / Arena, Street Light, Swimming Pool, Trampoline, Trees / Shrubs, Under Construction / In Progress Status, Unlabelled, Vehicle, Water Body, Water Tank / Oil Tank, Window.* It should be noted that the classes are highly imbalanced, as shown in Figure 3.

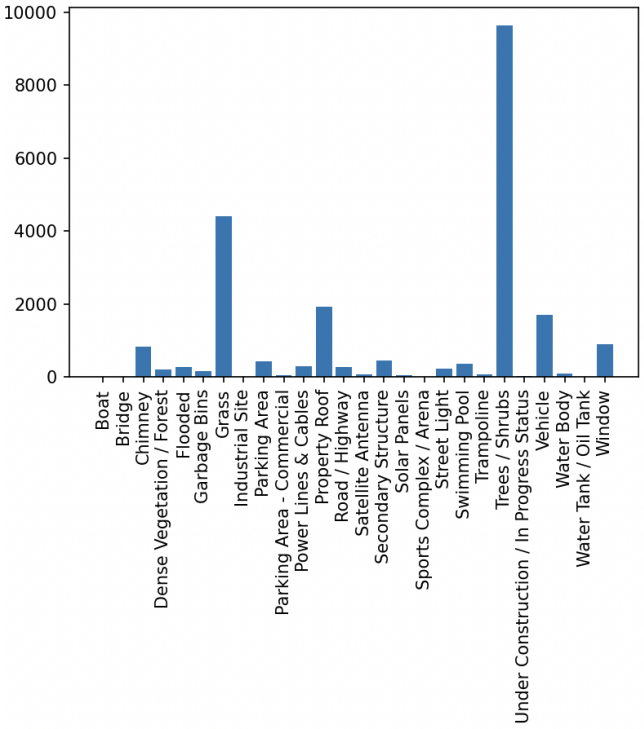

Figure 3. Data repartition among classes

We can already guess the difficulties we will face with this task. One of the main obstacles is the high variability in the appearance of assets and structures in an affected area, which can be caused by factors such as different types of damage, lighting conditions, and the presence of debris. Also, the data used for training the models is quite limited. This makes it challenging to develop models that can generalize well to new images, and handle unseen variations. However, we should use the Hurricane Harvey Synthetic dataset for this purpose. Another difficulty is the high resolution of the images. Indeed, these images contain a large number of pixels, which requires a lot of computational resources to process. The task of semantic segmentation is a highly computational task, requiring large amounts of data and computational resources, which can be a challenge. Also, high resolution images increase the complexity of the task, as it requires the model to be able to identify small objects and structures. Finally, the task of semantic segmentation is highly dependent on the quality of the annotation data, in this case the information about which pixels belong to which asset and structures, and this is again a parameter that we do not control.

2) Preprocessing

The pre-processing part of the code deals with loading and preparing the dataset for training and testing. It involves various steps such as loading the data, normalizing the images and creating a data loader. Let's detail the different steps of our preprocessing:

- Images and masks are loaded into a list and transformed into PyTorch tensors.
- The images are then put through data augmentation techniques such as random horizontal flipping and random rotation. This is done to increase the diversity of the training data and improve the generalization of the model.
- The images are then normalized using the mean and standard deviation of the images. This is a common pre-processing step to ensure that the input data has zero mean and unit variance.
- The data is divided into training and validation sets, with the validation set being used to evaluate the performance of the model during training.
- Finally, images and masks are converted into PyTorch datasets, and a data loader is created to batch load the data during training.

This part is the longest and most complex, because the result of our model depends largely on the preprocessing of the data. Therefore, before starting the development of the model, it is important to check that the images and masks are loaded correctly, that the transformations are applied correctly and that the images and masks are aligned and have the right dimensions. Figure 4 shows an instance of the output of a function that allows us to check the images and masks in the training data loader.

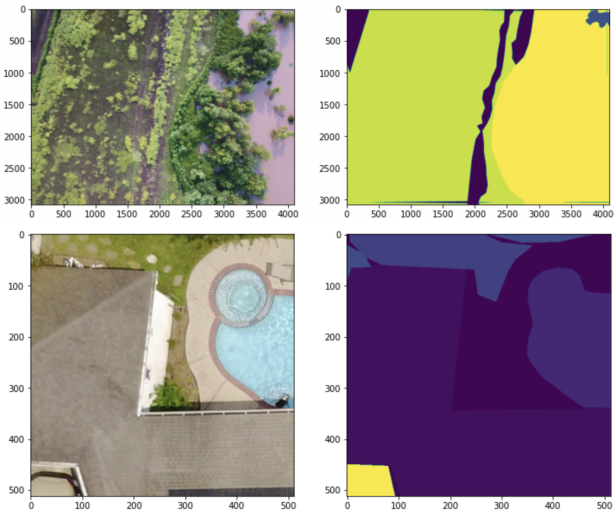

Figure 4. An example of correctly preprocessed data

Since everything appears to be correctly preprocessed, we can proceed to train our model using these images and masks.

3) Development of architectures

The model architecture we use is a U-Net model. This is a type of fully convolutional neural network (FCNN) particularly well-suited for semantic segmentation tasks. It was first introduced in 2015 by Olaf Ronneberger et al. in their paper "U-Net: Convolutional Networks for Biomedical Image Segmentation." The architecture is called U-Net because it is symmetrical and has a "U" shape, with the encoder and decoder parts of the network mirroring each other. This architecture is observable in Figure 5.

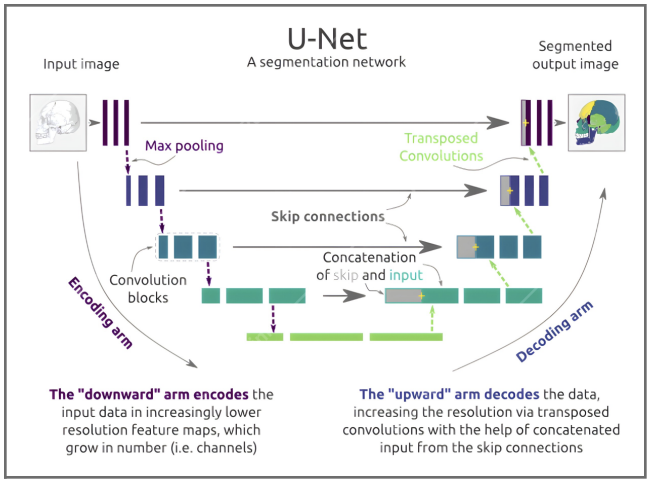

Figure 5. U-Net architecture

It is an encoder-decoder architecture, which means that it first compresses the input image into a feature map representation, then uses this feature map to generate a segmentation mask. The encoder part of the U-Net is a traditional CNN, consisting of multiple layers of convolutional and max pooling layers. These layers extract increasingly complex features from the input image, with the final layer producing a feature map that represents the most important information from the image. The decoder part of the U-Net then uses this feature map to generate a segmentation mask. It does this by using multiple layers of up-sampling and convolutional layers, which gradually restore the spatial resolution of the image.

One of the key features of U-Net is the use of skip connections between the encoder and decoder parts of the network. These connections allow the decoder to access the detailed feature maps from the encoder, which helps to preserve fine details in the segmentation mask. The skip connections also help to combat the problem of vanishing gradients, which can occur when using deep networks.

The U-Net architecture also uses a technique called "upsampling" to increase the spatial resolution of the feature maps in the decoder. Upsampling can be done using various methods, such as transposed convolutions or interpolation. In this project, we used the interpolation techniques provided by PyTorch.

Overall, U-Net is particularly powerful for segmentation tasks, as it can handle small objects and fine details. This model already achieves good results, but in addition to it, we also used a PSPNet (Pyramid Scene Parsing Network) which allowed us to achieve our best score on Kaggle (74.75% of accuracy). PSPNet is actually an extension of the popular U-Net architecture. The main difference between the two is that PSPNet uses a pyramid pooling module in addition to the encoder-decoder structure of U-Net. This module helps to capture context information at multiple scales and improve the overall performance of the model.

The pyramid pooling module consists of several pooling layers with different kernel sizes, which are applied to the feature maps of the encoder part of the network. A close-up on this part of the architecture can be seen in Figure 6 below. These pooling layers are then concatenated with the feature maps of the decoder part, before the final prediction is made. The pyramid pooling module allows the network to learn features at different scales, which improves the ability of the network to handle objects of different sizes.

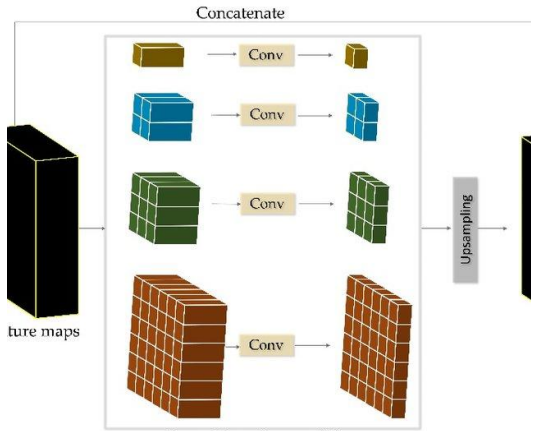

Figure 6. Pyramid pooling module

Additionally, it uses a spatial pyramid pooling module, which can adapt to the resolution of the feature maps, to achieve better performance. PSPNet is also equipped with a deep supervision mechanism, which can improve the accuracy of the model by supervising the performance of the model at multiple scales.

In summary, PSPNet is a powerful semantic segmentation model that improves upon the U-Net architecture. It incorporates a pyramid pooling module to capture context information at multiple scales and improve the overall performance of the model.

3 Model Evaluation

All the components needed to train our model are now defined: the model architecture, the optimiser, the loss function and the data loaders. We have also defined functions to evaluate the performance of your model, and an EarlyStopping class that allows us to save model weights when the validation loss does not improve over a number of patience epochs. We can now train our model and evaluate its performance.

In this section, we will cover the last two steps of the process, namely: Model training and Mask generation.

4) Model training

The objective of this step is to refine the model by iterating on a training process, to generate the best-in-class model to segment property and neighborhood features.

In addition to the U-Net architecture, our post-processing steps include:

- Intersection over Union (IoU) to calculate the similarity between the predicted mask and the ground truth mask.
- Interpolation to resize the predicted mask back to the original resolution.

These steps help to improve the accuracy of the model and produce more refined results.

Furthermore, it should be noted that the choice of loss function is highly important when designing complex segmentation architectures, as they drive the learning process of the algorithm. Slightly unbalanced datasets for example, which is our case, can work with a smoothed or generalized Dice coefficient (Jadon, 2020).

Thus, during the training loop, various metrics such as loss, pixel accuracy, mean IoU and Dice score, are calculated at each epoch and batch to monitor model performance. They are then logged in to WandB for visualization. The training loop also includes the use of a OneCycleLR scheduler, which adjusts the learning rate during training based on model performance.

5) Mask generation: results

Let's look at the scores we obtained with the Kaggle submissions. Note that when the score is high on the validation set, but lower on the test set (the one on which the model is applied when we submit to Kaggle), this is a sign of overfitting: the model does not generalize well to new, unseen data. Our different scores are summarized in Figure 7.

	Model parameters	Score on test set (Kaggle)
1	Deep LabV3 with default weights for ResNet50 encoder	17.16%
2	LinkNet with MobileNet as encoder	8.32%
3	PSPNet - ResNet101 encoder trained on ImageNet	**74.75%**
4	Unet, with ResNet34 encoder trained on Imagenet	62.20%

Figure 7. Summary score table

Let's discuss our results. First, we can mention the fact that we explored the idea of patch-wise training. Indeed, the last two models were trained on the full images, but we did integrate this technique into the training of our first two models, which did not seem to be performing. Patch-wise training is a training method in which the input image is divided into smaller overlapping patches, and each patch is individually passed through the model for the training. The model then makes a prediction for each patch and the predictions are stitched back together to form the final

output. This approach allows the model to learn from smaller and more local features, rather than trying to learn from the entire image at once. It allows the model to focus on specific regions of an image, which can be particularly useful for handling large images or images with complex structures. Additionally, by making predictions on overlapping patches, the model can make more accurate predictions for the boundaries of the objects in the image. Last but not least, patch-wise training helps to reduce the memory requirements of the model, since it is not necessary to store the entire image in memory at once.

In this report we are not going to talk about the first model, which has a bad Kaggle score (around 20% of accuracy). If you want more details, please refer to the Notebook, where all the models are commented on.

The second model scores even worse, as we got less than 10% accuracy on Kaggle. However, here it is interesting to note that the reason why the model did not perform well is due to a lack of complexity. The framework consists of a MobileNet as encoder (fairly lightweight, without too many parameters) and a Linknet as decoder, which has a similar operation to U-Net but with summation instead of concatenation. As a result of its lack of complexity, the model was not able to accurately learn the features and patterns in the input data. This led to poor performance on the semantic segmentation. It made really bad predictions, resulting in a high error rate, low accuracy, and poor performance on metrics such as Mean IoU and Pixel Accuracy.

Now let's talk about the third model, the one with which we got the highest score. PSPNet used as a decoder with an hyper-parametrized encoder like ResNet proved indeed to be the most competitive model among all. Figure 8 shows an example of the predicted masks next to the “true” masks. It can be seen that the prediction is quite close to reality.

Figure 8. An example of PSPNet predictions

Surprisingly, the structure of this model was quite simple, and without patches. It was already a powerful model performing very well, but we boosted it by a change in the setup. We simply resized the full image to a given size, a 480×640 rectangular without padding. We used learning rates around 0.1, lowered less abruptly by the learning rate scheduler, and the patience of the EarlyStopping was slightly raised to allow the model to run (while saving the checkpoints). As you can see on Figure 9, the EarlyStopping mechanism worked nicely, since the iterations are stopped at the right moment (when the validation loss is starting to increase). The best performance model was indeed coming from the checkpoint corresponding to an iteration around the 30th, and it provided a 74.75% score in the competition.

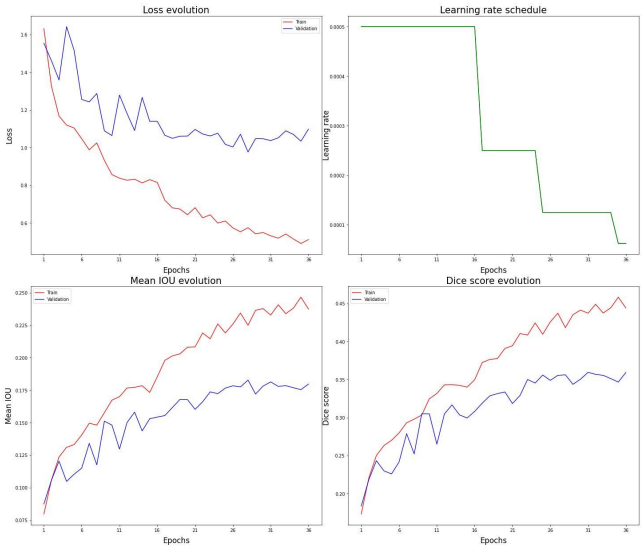

Figure 9. PSPNet metrics on the evaluation set

For the fourth model, it was a U-Net model, more classical than PSPNet, with which we obtained a decent but not outstanding score (around 60% of accuracy). We implemented a setup based on the resizing of entire images (similar to PSPNet). Then, the U-Net decoder was used, with pretrained weights coming from a library called *segmentation models pytorch*. The Decoder channels, representing the dimensions of channels used during the oversampling of the feature embedding, were made quite deep. Also, an additional head of a fully connected layer was called through the API, which allowed inserting Dropout.

4 Discussion

We achieved a score that is very acceptable considering that this was the first time we had ever built such a complex deep learning system, and that we were constrained by time and computing power. There are obviously additional techniques to those we used that could have improved our results. For example, we could have used Non-Maximum Suppression (NMS), a technique used in object detection to remove overlapping or redundant predictions. If we want to include it, we would need to implement it separately after generating the predictions.

REFERENCES

AspramG (no date) ASPRAMG/unet_image_segmentation: Aerial imagery segmentation task using UNET, GitHub. Available at: https://github.com/AspramG/UNET_Image_Segmentation.

Castro, P.byS. (2021) Object detection and instance segmentation with Detectron2 - Robotic Sea Bass, Robotic Sea Bass - Learn assorted topics in robotics, AI, programming, and more. Available at: https://roboticseabass.com/2020/11/22/object-detection-and-instance-segmentation-with-detectron2/.

Hurricane Harvey Challenge [DSBA 2022-2023]. *Notion*. Available at: https://granular.notion.site/granular/Hurricane-Harvey-Challenge-DSBA-2022-2023-80de810d1f6c4b729c1ee0419cc66943.

Jadon, S. (2020, October). A survey of loss functions for semantic segmentation. In 2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB) (pp. 1-7). IEEE.

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 3431-3440).

Ronneberger, O., Fischer, P., & Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical image computing and computer-assisted intervention (pp. 234-241). Springer, Cham.

Zhao, H., Shi, J., Qi, X., Wang, X., & Jia, J. (2017). Pyramid scene parsing network. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2881-2890).