Foundations of Deep Learning Final Project

Yasmina Hobeika, Firas Abo Mrad

Introduction

Hurricane Harvey was a powerful Category 4 storm that struck Texas and Louisiana in August 2017, resulting in severe flooding and over 100 fatalities. The National Hurricane Center estimated that the storm caused $125 billion in damage, making it one of the costliest natural disasters in U.S. history, second only to Hurricane Katrina.

As the world increasingly faces threats from climate change-related natural disasters, it is crucial to find ways to better track and assess the impact of these events. In this exercise, we will work both independently and together to create effective machine vision models that can identify residential properties and community assets after a major flood.

Problem Statement

The frequency of natural disasters and human-induced conflicts has brought attention to the importance of Earth Observation (EO) data in designing effective Humanitarian Assistance and Disaster Relief (HADR) interventions. The use of remotely sensed data is revolutionizing our understanding of these situations and improving the speed and effectiveness of our response. However, one major challenge in this area is the timely availability of accurate post-disaster information. Aerial post-flood maps can provide localized information about the extent of flood damage and how it has affected communities' access to essentials such as shelter, clean water, and communication. However, these insights are often only available hours or days after a flooding event has occurred. Additionally, the limited availability of historical data makes it difficult to develop practical machine learning-based methods.

In this challenge, we will work on developing an accurate dense segmentation model for aerial post-flood imagery. We will focus on identifying property attributes in the Houston area following Hurricane Harvey in order to generate an autonomous post-flood map. High-resolution image data of Houston, Texas, USA is provided to us via GeoEngine. We will use this drone image dataset to train segmentation models to better understand ground conditions after hurricanes. Our goal is to accurately identify and segment common large and small assets in residential Houston that may or may not have been damaged by Hurricane Harvey. In this way, we hope to make the process of generating post-disaster maps more efficient and accurate, ultimately improving the speed and effectiveness of HADR interventions.

Pre-processing and Transformations

We started by moving all the files into a data folder containing 3 folders: train images, train masks and test images (by storing the images id in 3 lists). Then we used the glob module to find all files in the directory that match a certain pattern. The glob.glob() function returns a list of file paths that match the given pattern, and the sorted() function is then used to sort that list alphabetically. We then created a dataframe to store all the ids of the train images in order to split efficiently the images between train and validation sets. We proceeded to split the data for training and validation.

Moreover, we applied random data augmentation to the image and mask, including darkening, brightening, horizontal flipping, vertical flipping and rotating. By augmenting the training data, the model is exposed to a wider variety of images, which can help to improve its generalization and reduce overfitting. These transformations help simulate different conditions that the model may encounter in real-world scenarios. For example, resizing the image can help the model learn to handle different scales of objects, while flipping the image horizontally or vertically can help the model learn to handle symmetry and orientation. The grid distortion and random brightness/contrast changes can help the model learn to handle variations in lighting and perspective, while Gaussian noise can help the model learn to handle image noise.
It's important to note that the validation set is not augmented with these transformations, as this set is used to evaluate the performance of the model under real-world conditions. This is why the validation set is only composed of a subset of the transformations used for training set.
Moving on to the dataset, it takes in several parameters in its constructor, including the paths to image and mask files, a list of image file names, means and standard deviations for normalizing the image data, a flag to indicate whether or not the data is for training, and an optional transform to apply to the images and masks.
The image and mask are read from file, converted to RGB color space, and transformed (if a transform is provided). The image is then converted to a PyTorch tensor and normalized using the provided means and standard deviations, and the mask is converted to a PyTorch tensor of long data type. If the data is for training, random data augmentation is applied to the image and mask.
If the patches flag is set to True, the image and mask are divided into smaller patches.
Indeed, we took the image and mask and divided them into smaller patches of size 512x768. This way we are able to process small pieces of the image in order to detect features (edges, etc). This also has a nice regularization property, since we're estimating a smaller number of parameters, and those parameters must be "good" across many regions of each image, as well as many regions of all other training images.

All models and Adjustments

Unet from scratch:

The U-net architecture is a convolutional network designed for fast and accurate image segmentation. We created an original U-net architecture from scratch and trained it. The architecture has two main components, the contraction path (also known as the encoder) and the symmetric expanding path (also known as the decoder). The encoder is composed of a stack of convolutional and max pooling layers, it is used to capture context in the image. The decoder uses transposed convolutions to enable precise localization. Despite various augmentation techniques, the network showed overfitting after 300 Epochs with a learning rate of 0.001. We stopped when the loss was almost not changing and the validation accuracy was 0.7. When we submitted this result as our baseline submission, we got a score of 0.635.

Hyperparameter tuning function: One of the most important hyperparameters of the optimizer is the learning rate, which determines the step size at which the optimizer adjusts the parameters of the model. After the first implementation, we decided to implement a function that takes an optimizer object as an input and iterates through a dictionary that contains information about the hyperparameters of the optimizer, including the learning rate. In this way, we optimize the model by choosing the best learning rate at each epoch.

Unet imported from smp:

We used a bunch of architectures using the library segmentation_models_pytorch which we imported as smp and included Unet resnet34, resnext50_32x4d with and without softmax and resnext101_32x4d with and without softmax. Moreover, the U-Net model is a convolutional neural network designed for image segmentation tasks, which uses a "U" shaped architecture with a contracting path to capture context and a symmetric expanding path to enable precise localization.

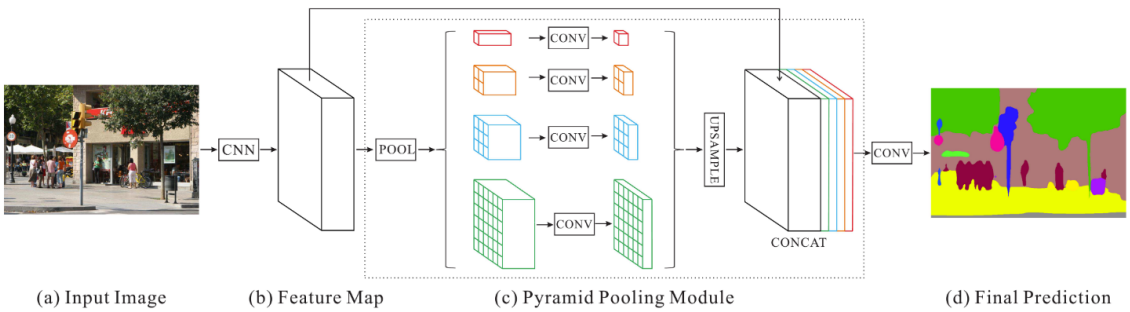
The "resnet34" version of this model utilizes a ResNet-34 pre-trained model as the encoder part of the U-Net. ResNeXt (Residual Networks with Extreme Inflation) models are a variation of ResNet models that use multiple branches with grouped convolutions in order to improve performance. "resnext50_32x4d" and "resnext101_32x4d" are versions of the ResNeXt model with 50 and 101 layers respectively, and 32 groups of 4 convolutional filters.The "with softmax" versions of these models include a softmax activation function in their final layer, which is often used for classification tasks to produce probability values for each class.

The "without softmax" versions do not include this function and may be used for tasks such as object detection or semantic segmentation.

Similarly to the Unet we developed from scratch, the network showed overfitting after 250 Epochs with a learning rate of 0.001. We also stopped when the loss became more constant, and the validation accuracy approached 0.7. When we submitted this result as our baseline submission, we got a score of 64.62 which was slightly better than our first model. Concerning the hyperparameters, similarly to the first UNET we tried a multitude of learning rates which yielded different results, the best one was 0.001 which got the best result.

PSP net:

PSPNet, or Pyramid Scene Parsing Network, is a semantic segmentation model that utilizes a pyramid parsing module that exploits global context information by different region based context aggregation. The local and global clues together make the final prediction more reliable. Given an input image, PSPNet uses a pretrained CNN with the dilated network strategy to extract the feature map. The final feature map size is 1/8 of the input image. On top of the map, we use the pyramid pooling module to gather context information. Using our 4-level pyramid, the pooling kernels cover the whole, half of, and small portions of the image. They are fused as the global prior. Then we concatenate the prior with the original feature map in the final part of. It is followed by a convolution layer to generate the final prediction map.

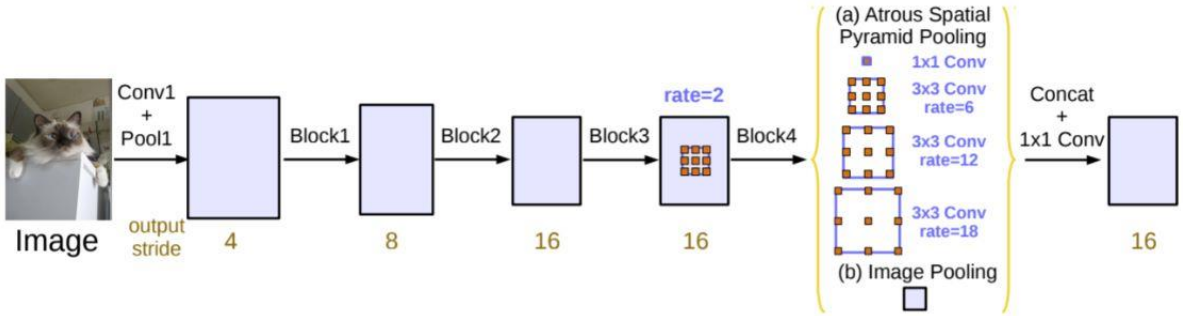


Moving on to the training of this model, it showed signs of overfitting after 250 Epochs. To address this, the training was stopped when the loss became more constant and the validation accuracy reached a satisfactory level of 0.7. We then submitted this model and got a score of 71.55 which was also slightly better than the previous model.

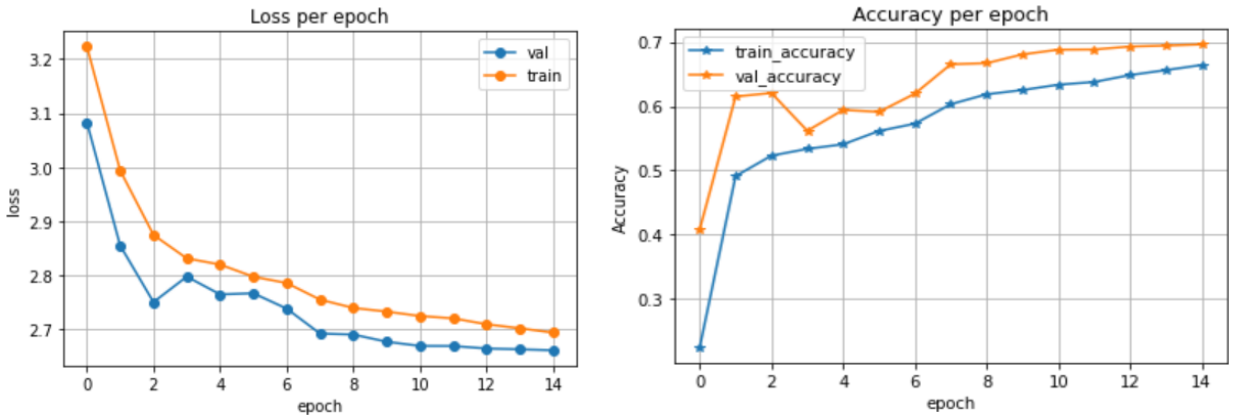
Hyperparameters, such as the learning rate, play a crucial role in the performance of a neural network. For this particular model, a range of learning rates were experimented with to find the optimal value. The best result was obtained with a learning rate of 0.001 which seemed to be consistently the best learning rate for all our models. Indeed, this indicates that the model was able to learn from the data at a suitable pace, which is crucial for a good performance.

DeepLabV3:

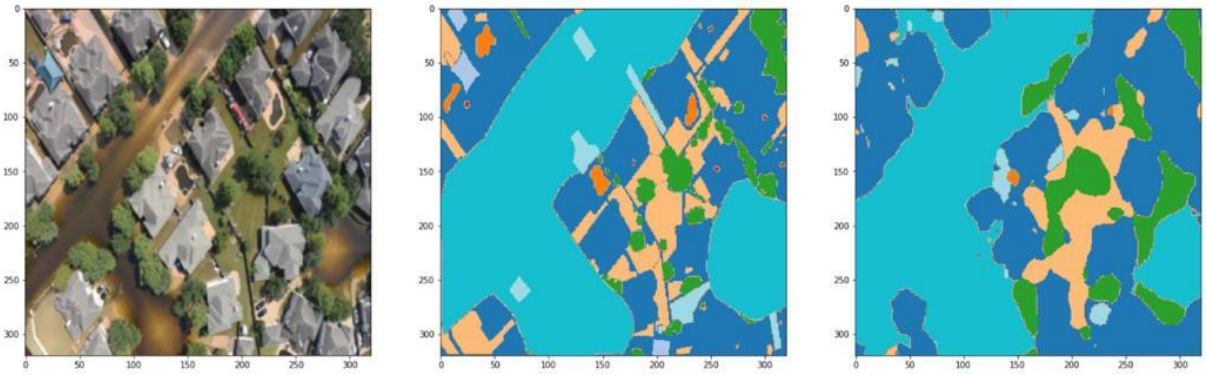
In the DeepLabV3 model features are extracted from the backbone network (VGG, DenseNet, ResNet). To control the size of the feature map, Atrous convolution is used in the last few blocks of the backbone. On top of extracted features from the backbone, an ASPP network is added to classify each pixel corresponding to their classes.
The output from the ASPP network is passed through a 1 x 1 convolution to get the actual size of the image which will be the final segmented mask for the image.



This model performed the best and yielded a score of 72.62 when we trained it for 15 epochs. We did not see the necessity to train it more because the loss became constant after the 9th epoch and same for the accuracy at around 70% shown in the graphs below.



We tuned the hyperparameters in a similar fashion to the previous models and the learning rate of 0.001 also gave the best results. Below is a representation of the image on the left vs the true mask in the middle and our prediction on the right, noting that this prediction was computed with our best performing model which got us our best submission of 72.62



Post-processing steps

To properly evaluate how accurate our model is we calculated the “pixel accuracy” which takes in the model's output and the ground truth mask as input and calculates the proportion of pixels that have been correctly classified. It does this by first taking the argmax of the output across the channel dimension, to obtain a predicted mask, and then compares it to the ground truth mask. It then returns the proportion of pixels that have the same label in the predicted and ground truth masks.
We also calculated the mean Intersection over Union which takes in the model's output and the ground truth mask as input. It first applies softmax function to the output to obtain class probabilities. Then it takes argmax of the class probabilities to obtain predicted mask. The function then loops over each class and calculates the intersection over union (IoU) score for each class and returns the mean of IoU score across all classes.
We also plot the training and validation loss as well as the accuracy to better visualize our results.
We finally create a dataframe to store all the ids of the test images, in addition to storing their sizes then calculate the output for each image in test set using the best model previously trained and save the prediction in a new folder which is then looped over to output the content in the tar file that was submitted on Granular.ai.