Foundation of Deep Learning: Hurricane Harvey Challenge

Leonardo Basili, Léopold Granger

February 7, 2024

Abstract

We worked on the Hurricane Harvey Challenge for the Foundation of Deep Learning. To do so, we use dense segmentation models on aerial imagery. After multiple iterations, we achieve a 73% accuracy score.

1 Introduction

Dense segmentation models, which we will also refer to as aerial classification models, typically perform pixel-level classification of aerial images. Essentially, they assign a label to each pixel in an image. Typically, we use convolutional neural networks (CNNs) to work on these tasks, CNNs are trained on relatively large datasets of labeled aerial images. In this challenge, the use of dense segmentation models on aerial images can help government agencies quickly understand the damages made my future hurricanes, and distribute emergency care to those most in need.

2 Data preprocessing

We were given 374 ”raw” images to work with, out of these, 299 were images with a mask provided. Masks were multi-class images that reflect for every pixel, which label it belongs to. Our first approach was to identify which images were our testing set, we did so by iterating through both the raw images folder, and the masks folder. Once the raw images were stored in one place, we created a variable to iterate through them. To do so, we generated a list, sorted the list, and then shuffled the list. This helps us ensure that there is no entry bias. We also checked the amount of pictures we have, to make sure that no mistakes were made. We then created a validation set, to ensure that we were not over-fitting. Afterwards, we created a dataset class inherited from the PyTorch dataset class. The class has a constructor ”init” which takes in three arguments: ”imagesfilenames”, ”imagesdirectory”, ”masksdirectory” and ”transform”. ”imagesfilenames” is a list of image file names, ”imagesdirectory” is the directory where the images are located, ”masksdirectory” is the directory where the masks are located, and ”transform” is an optional argument that can be used to apply data augmentation to the images and masks. The ”len” and ”getitem” methods are required by PyTorch’s dataset framework. ”len” returns the number of images in the dataset and ”getitem” is used to access a specific image and its corresponding mask. The ”getitem” method takes the index of the image and reads the image using OpenCV library. It then converts the image from BGR to RGB color space. The reason why we apply this transformation is that most deep learning frameworks expect the input images to be in RGB format. We then also read the corresponding mask file using OpenCV and convert it from 8-bit integer to float32. One reason why we converted the masks to float32 is that deep learning algorithms often convert images to float32, therefore, it is convenient to keep the same data type between masks and images. Finally, our function applies transformations to the image and mask, if there is a mask provided.

2.1 Transformations

In computer vision, data augmentations, also known as transformations, are essential in PyTorch as they increase the diversity of the training data, which can lead to more robust and generalizable models. The reason for this is that by applying different transformations to the training data, the model is exposed to a wider range of variations in the input, leading to a more robust representation of the underlying pattern in the data.

Our code defines two sets of image preprocessing steps, one for training data (traintransform) and one for validation data (valtransform). We included several image augmentation techniques including but not limited to: resizing, shifting, rotating, color shifting, and adjusting brightness and contrast. These techniques were applied randomly with a probability of 0.5 for each step. The final step in the training preprocessing was normalizing the image by subtracting mean values and dividing by standard deviations and converting the image to a PyTorch tensor using ToTensorV2.

A big part of this assignment was to decide on the appropriate size of the images.

The validation data preprocessing is less extensive, it only includes resizing and normalizing the image by subtracting mean values and dividing by standard deviations and converting the image to a PyTorch tensor using ToTensorV2.

3 Model architecture

Setting up a semantic segmentation model includes several key steps. To start with, we had to pick an overall structure for the model, this defines how the input image is processed to produce a segmentation mask. The common architectures that applied here were U-net, DeepLab and Mask R-CNN. We chose DeepLabV3 because after iterations, it was the model that achieved the highest accuracy. We also had to choose an encoder, the encoder extracts the features from the images. Among the encoders we could choose from, we tried both VGG, ResNet and mobilenetv2. mobilenetv2 achieved the highest accuracy. Our code used preinitalized encoder weights to save time, we use ImaegeNet. Imagenet is a large dataset of images and associated labels that is widely used for training and evaluation computer vision models. We used a default amount of 5 for the encoder depth. Another decision we had to make was the choice of optimizer. We started with Adam and eventually tried stochastic gradient descent. Adam outperform stochastic gradient descent. We also had to choose a loss function. The loss function measures the difference between the predicted segmentation mask and the ground truth mask. Common loss functions for semantic segmentation include CrossEntropyLoss and DiceLoss. We used CrossEntropyLoss because CEL is a good choice for multi-class classification problem as it is able to handle categorical data, where the target labels are one-hot encoded and the output is a probability distribution over the classes. This makes it a natural choice for multi-class classification problems. A last decision we had to make was to choose a learning rate. The intuition was that a higher learning rate can cause themodel to converge quickly but may also overshoot the optimal weights. We also had to choose the number of epochs, the idea is that the more epochs we have, the more time the model has to learn from the data. A more complex task or a dataset with more noise may require more epochs to learn, while a simpler task or a dataset with higher quality data may require fewer epochs to learn. After many iterations, we found that 20 epochs was a good middle ground.

4 Results

The following table shows the different scores obtained with the models used. As mentioned before, we tried three different models one from the library DeepLabV3 and the other two from the library PSPNet. We chosen these models since they are well known deep learning architectures for semantic segmentation tasks. When using mobilenet_v2 and resnet101 we were able to achieve higher scores increasing the number of epochs instead resnet34 did not lead us to high scores with an higher number of epochs probably due to the low learning rate set. We used Adam W optimizer for almost all the trials since we consider it the most proper for the task, indeed we also tried to use Stochastic Gradient Descent on mobilenet_v2 obtaining unsatisfatctory result. It is worth to mention that the we achieved the best score (75.58) adding horizontal flip and vertical flip transformations on the data loaders which probably increased furthermore the performance compared to the other trials. Overall the mobilenet_v2 and resnet101 allowed us to achieve the best 3 scores and in general an higher number of epochs lead us to the best scores (as commonly assumed).

[1] the score was achieved adding horizontal flip and vertical flip transformations to the data.

mobilenet_v2 from DeepLabV3	learning rate	0.001	0.001	0.0001
	batch size	4	4	4
	epochs	5	5	20
	criterion	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss
	optimizer	Adam W	SGD	Adam W
	score	72.24	66.53	73.51
resnet34 from PSPNet	learning rate	0.001	0.000001	0.001
	batch size	4	4	4
	epochs	5	25	10
	criterion	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss
	optimizer	Adam W	Adam W	Adam W
	score	70.25	66.06	70.63
resnet101 from PSPNet	learning rate	0.001	0.0001	0.0001
	batch size	4	4	4
	epochs	10	20	25
	criterion	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss
	optimizer	Adam W	Adam W	Adam W
	score	70.32	14.39	75.58[1]

Table 1: Score and parameters for the three models implemented

model	mobilenet_v2	mobilenet_v2	resnet101
learning rate	0.001	0.0001	0.0001
batch size	4	4	4
epochs	5	20	25
criterion	Cross Entropy Loss	Cross Entropy Loss	Cross Entropy Loss
optimizer	Adam W	Adam W	Adam W
score	72.24	73.51	75.58

Table 2: models and parameters for the best 3 scores