Assignment 3

Julia Putz, Dario Giovannini

June 2023

Contents

1 Introduction

For this assignment, we focused on a semantic segmentation project. To accomplish this, we relied on a tutorial available on github [5], which provides instructions on how to fine-tune MaskFormer for a dataset provided specifically for semantic segmentation tasks. Our goal was to not only fine-tune MaskFormer but also Mask2Former, using a dataset containing sidewalk images and compare its outcomes. Both, the models and the dataset are publicly available on Hugging Face.

1.1 Semantic Segmentation

Semantic segmentation involves the process of assigning labels to every pixel in an image, enabling us to distinguish different objects, e.g. a sidewalk or a person, within this image. As this labelling is very fine-grained, it provides a detailed understanding of the scene. This is crucial for tasks like autonomous driving or medical imaging.

1.2 Hugging Face

Hugging Face can be best described as a data science platform. It offers users to build, train and deploy machine learning models. Moreover, it offers several libraries of which 'transformers', 'evaluation' and 'datasets' are used to complete this assignment. 'transformers' enables us to access and use pre-trained models like MaskFormer and Mask2Former with only a few lines of code. Using the 'datasets' library we can load a dataset provided by Hugging Face with only one line of code by using the function load_datasets. However, further steps, like splitting it into test and train data or dividing it into batches has to be implemented separately.

1.3 MaskFormer vs. Mask2Former

MaskFormer is a model, introduced in the paper "Per-Pixel Classification is Not All You Need for Semantic Segmentation" by [1]. It offers a novel approach to semantic segmentation by utilizing mask classification, which, according to them, is capable of solving not only semantic- but also instance-level segmentation tasks. Especially with a large number of classes, they observed that MaskFormer even outperforms per-pixel classification baselines. The model simplifies the segmentation landscape by employing a mask classification model that predicts binary masks associated with global class label predictions [3].

Significant performance and efficiency improvements over MaskFormer are featured by Mask2Former, which is why we decided to explore this model primarily. These improvements are due to components like masked attention and localized feature extraction. For preprocessing and postprocessing the same steps are used [2].

1.4 Dataset

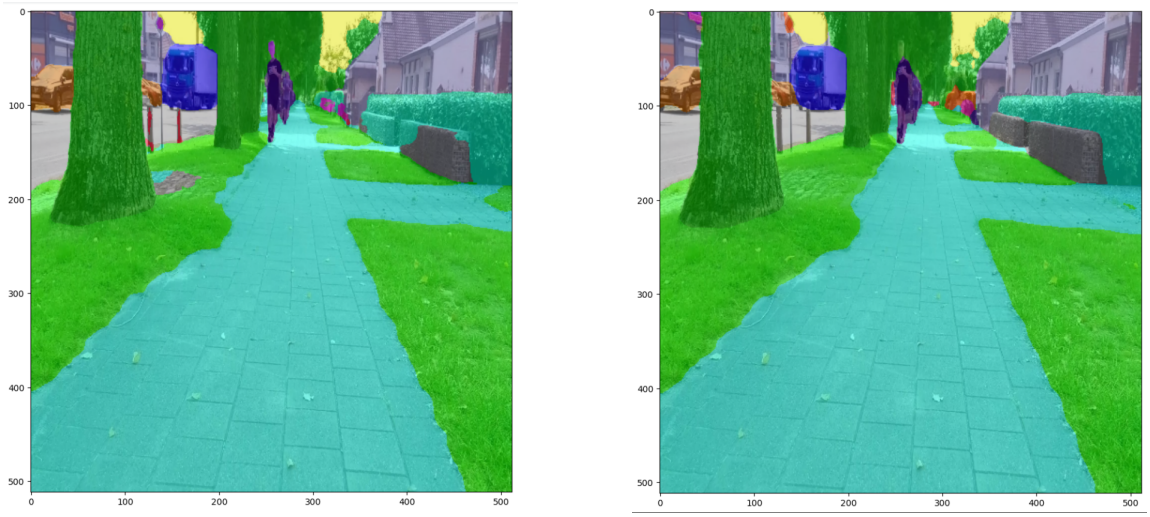
The dataset used in this assignment is the same as in the original code. It is called sidewalk semantic and is available on Hugging Face [4]. This dataset contains images of Belgian sidewalks captured in 2021. It encompasses a total of 1000 rows, with each row containing the pixel values of an image along with its corresponding labels per pixel, also called a segmentation map. Therefore, one image can be associated with multiple labels, which provide descriptions of various elements present in the image. For instance, labels such as 'flat-sidewalk', 'human-person', or 'vehicle-bicycle' may be assigned to an image. In total, there are 34 distinct labels available.

To ensure proper training and testing of the model, the dataset must be divided into training (80%) and testing (20%) sets. Additionally, to enhance the model's robustness, data augmentation techniques, including resizing, random cropping, and horizontal flipping of the images, are employed.

It is necessary to reformat the images and their corresponding segmentation maps into a format that aligns with the model's expectations. This is because, as mentioned in section 1.3, the selected models anticipate masks instead of per-pixel annotations. Furthermore, to reduce overfitting, the training data is split into batches and reshuffled every epoch. On the other hand, the test data is solely split into batches without shuffling. For this purpose, PyTorch provides a convenient function called 'DataLoader', which can be utilized to handle the batching process in combination with a custom function taking care of the reformatting.

The dataset was found to be used in several projects on Hugging Face. One creative potential use-case we encountered was a pizza delivery robot that needs to find its way to the customer without any collisions.

To get a first glimpse, semantic segmentation is performed and processed on the initial pre-trained models prior to fine-tuning. The resulting segmentation overlay, when using MaskFormer and Mask2Former, is visualized in Figure 1.



(a) MaskFormer (b) Mask2Former

Figure 1: Segmentation overlay when using initial pre-trained model

2 Code

As already mentioned in Section 1, the code we were working from is publicly available on github [5]. The code is provided in form of a jupyter notebook, which we moved to 3 python scripts and wrapped code blocks into functions in order to get a better overview. Therefore, we had to make minor adjustments to certain parts, e.g. to ensure the accessibility of variables where necessary.

- One instance where this was required was when invoking 'collate_fn_custom' within 'create_batch' (because of what was described in Section 1.4). In this case, it was necessary to pass the processor to this function to ensure its accessibility.

- Visualizing the image and its segmentation mask is not only done in the beginning to check if loading and reformatting the data worked, but also in the end when comparing the ground truth to the segmentation map predicted by the fine-tuned model. Therefore, the function 'show_overlay' is introduced and called whenever needed to reduce code duplication.

Finally, a jupyter notebook was used to perform the fine-tuning and experiments for ease of use.

3 Problems

When initially experimenting with the code presented in the tutorial, there was some confusion regarding the datatypes expected at various steps - numpy arrays, pytorch tensors, PIL images being the most prominent - and the correct order of dimensions. Some trial-and-error was required to get the dataset in the correct shape, specifically skipping a transpose step from the tutorial that we found to produce incorrect instance shapes.

While fine-tuning the model using a new classification head for the specific dataset we were working with, the use of cuda to utilize GPU processing power led to some hassle with making sure the various tensors were sent to the correct device.

In the tutorial itself, the model is evaluated using the metric 'mean_iou' provided by the 'evaluation' package after every epoch. In our testing we found this step to be extremely time consuming, even moreso than the training itself. A training epoch with 800 images took about 5 minutes using cuda, but the evaluation of only 200 images took almost 25 minutes. The documentation on this function is somewhat lacking and the source code difficult to read, so we can only assume that it is in some way not quite suitable to our task. We then implemented our own "mean_iou" metric using cuda-compatible, torch-native tensor methods, which brought evaluation of the entire test set down to about 90 seconds. The results are in a similar ballpark but not identical to the ones from the provided metric, which we cannot explain satisfactorily.

Our initial training and testing was done using the more modern mask2former model. When attempting to perform the same training regimen with a maskformer model, we quickly discovered that the training time is significantly longer (several hours for an epoch, even using cuda). One possible explanation for this is the efficiency gain thanks to the improvements of the newer model, especially in regards to memory. It was observed that during training with the mask2former model, the 8GB GPU memory was not fully utilized with a batch size of 2, but it was fully utilized by the maskformer model with the same batches. It thus seems likely that memory issues contributed to the observed efficiency issue. Training of the maskformer model was not completed, as it did not seem possible in a sensible time.

4 Further Experiments / Tests

The principal experiment performed was finetuning a mask2former model on a new dataset. This was accomplished by extracting the labels from the dataset and passing the resulting dictionary to the model upon loading, which defines a new classification head. Using a training loop with an Adam optimizer, the model is trained for a defined number of epochs.

Initially, a model was trained with only two epochs to test the effectiveness of the training loop. The loss was observed to fall from more than 50 to about 26. The evaluation using the custom implementation of the "mean_iou" metric for this model after fine-tuning is 0.466 across all classes and images.

Next, a model was trained for 20 epochs, with the goal of finding a suitable stopping point beyond which the gain in training loss is minimal. This was observed to be around 6 epochs, at which point the training loss had fallen below 20; after 20 epochs it only reached a final value

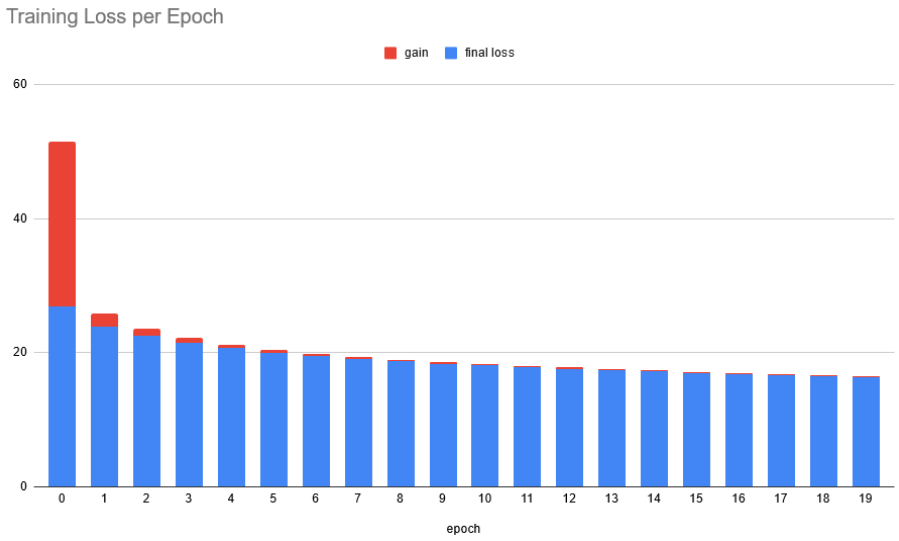


Figure 2: Training loss development per epoch of fine-tuning

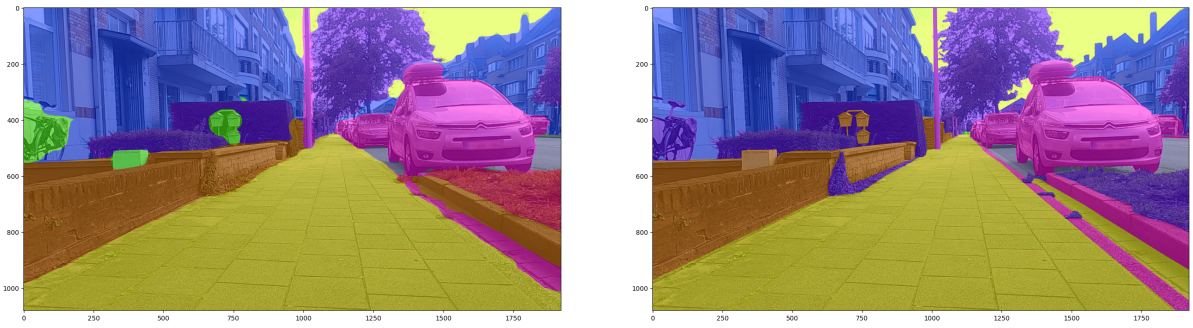
of 16.31. The evaluation score with this model was 0.54 average IoU, a considerable but not breathtaking improvement over the model that only trained for two epochs.

An attempt was made to fine-tune a maskformer model, but as mentioned above the training time turned out to be prohibitive, and so the training was aborted before even one epoch could be completed, as that would have taken several hours.

5 Results

The development of the training loss with the epochs is illustrated in figure 2. It shows the previously discussed steep decline in training effectiveness as training goes through more epochs; training past 5-6 epochs with this dataset only gave quite small improvements.

A comparison between a prediction using the fine-tuned mask2former model and the ground truth is shown in figure 3. It clearly shows that there is broad agreement for the most notable areas of the picture, but details are mislabeled or missed, and areas with complex shapes can lead to errors, such as around the tree in the background.



(a) Prediction (b) Ground Truth

Figure 3: Segmentation overlay with the fine-tuned model

References

[1] Bowen Cheng, Alexander G. Schwing, and Alexander Kirillov. Per-pixel classification is not all you need for semantic segmentation, 2021. arXiv:2107.06278.

[2] Huggingface Documentation. Mask2former. URL: https://huggingface.co/docs/transformers/model_doc/mask2former.

[3] Huggingface Documentation. Maskformer. URL: https://huggingface.co/docs/transformers/model_doc/maskformer.

[4] Hugging Face. Sidewalk semantic. URL: https://huggingface.co/datasets/segments/sidewalk-semantic.

[5] Niels Rogge. Fine-tune maskformer for semantic segmentation. URL: https://github.com/NielsRogge/Transformers-Tutorials/blob/master/MaskFormer/Fine-tuning/Fine_tuning_MaskFormerForInstanceSegmentation_on_semantic_sidewalk.ipynb.