



# Reproducing “Unsupervised Semantic Segmentation by Distilling Feature Correspondences”

14/06/2024

---

Chris Overtveld  
TU Delft - 4423621

## Overview

In the 2022 paper "Unsupervised Semantic Segmentation by Distilling Feature Correspondences," the authors propose a novel approach to semantic segmentation without relying on annotated data. This blog post aims to reproduce their results with a particular focus on the sensitivity to hyperparameters and evaluating the model's performance on different datasets.

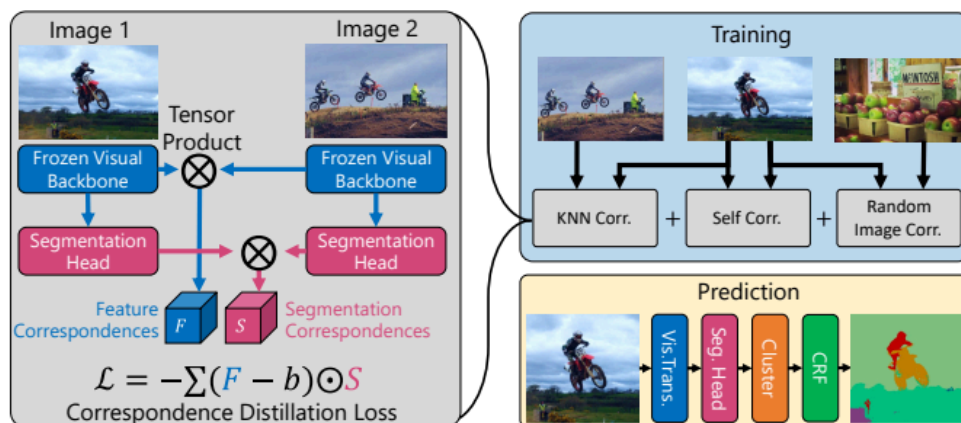
## Understanding the Paper

The core idea of the paper is to leverage feature correspondences within images to perform semantic segmentation. By distilling these correspondences, the method aims to cluster pixels into semantically meaningful regions without supervised training data.

## Key Components

1. **Feature Extraction:** Utilizing a pre-trained backbone to extract feature maps from images.
2. **Feature Correspondence:** Identifying correspondences between features within an image.
3. **Clustering:** Clustering these features to form the segmentation mask.
4. **Distillation:** Refining these clusters to enhance segmentation accuracy.

## High-Level Architecture

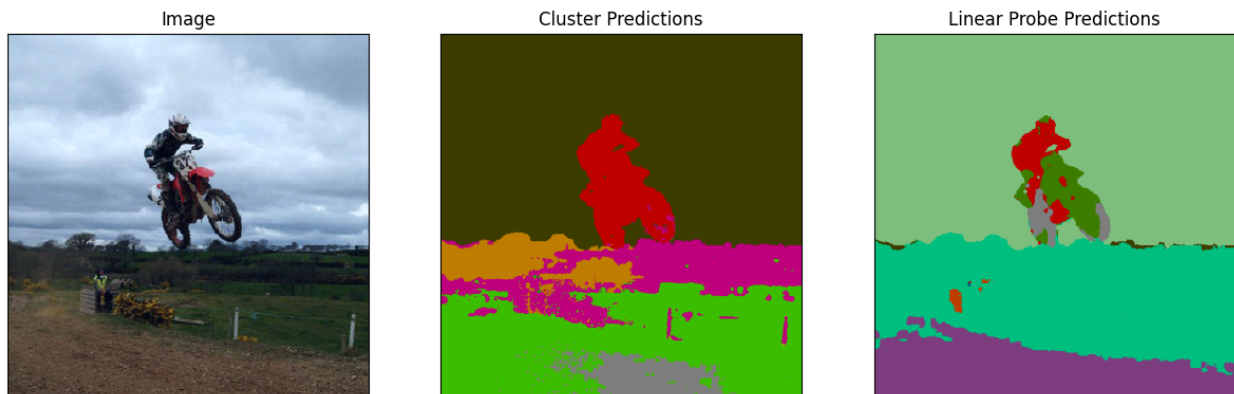


## Reproduction

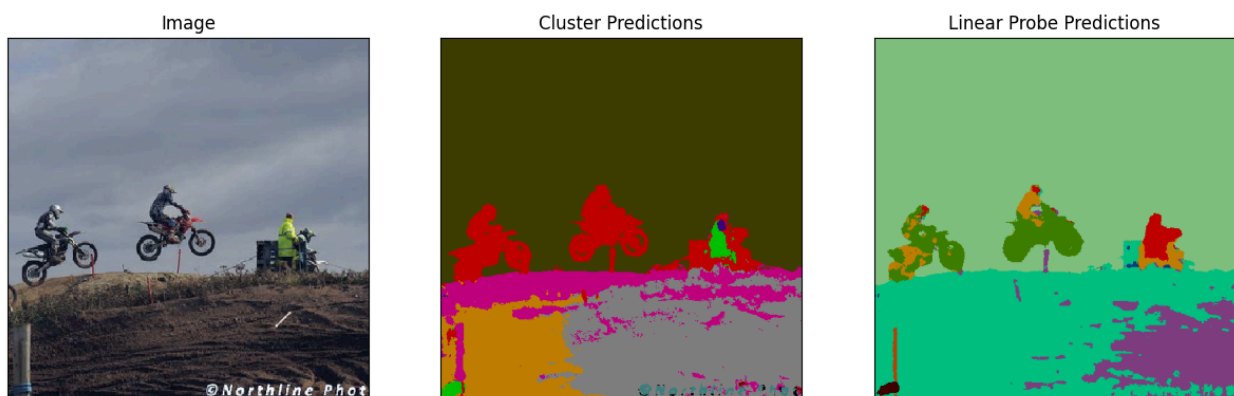
### 1. Initial Reproduction

Getting a Google Colab environment running just to test the paper implementation with pre-trained weights was the first step in reproducing the paper's results. The author's provided Colab environment was slightly out of date so required removing some deprecated imports (`torch._six`) and downgrading `pytorch-lightning` to version 1.8.4.

Both motocross sample images were used to see how the segmentation and predictions worked:



*Figure 1. Moto1.jpg clustering and segmentation.*



*Figure 2. Moto2.jpg clustering and segmentation.*

After running the colab, I setup the local environment to further evaluate the model, train from scratch, and fine tune the model on new data if desired. The results from evaluating the `cocostuff27` dataset are shown below (running eval\_segmentation.py):

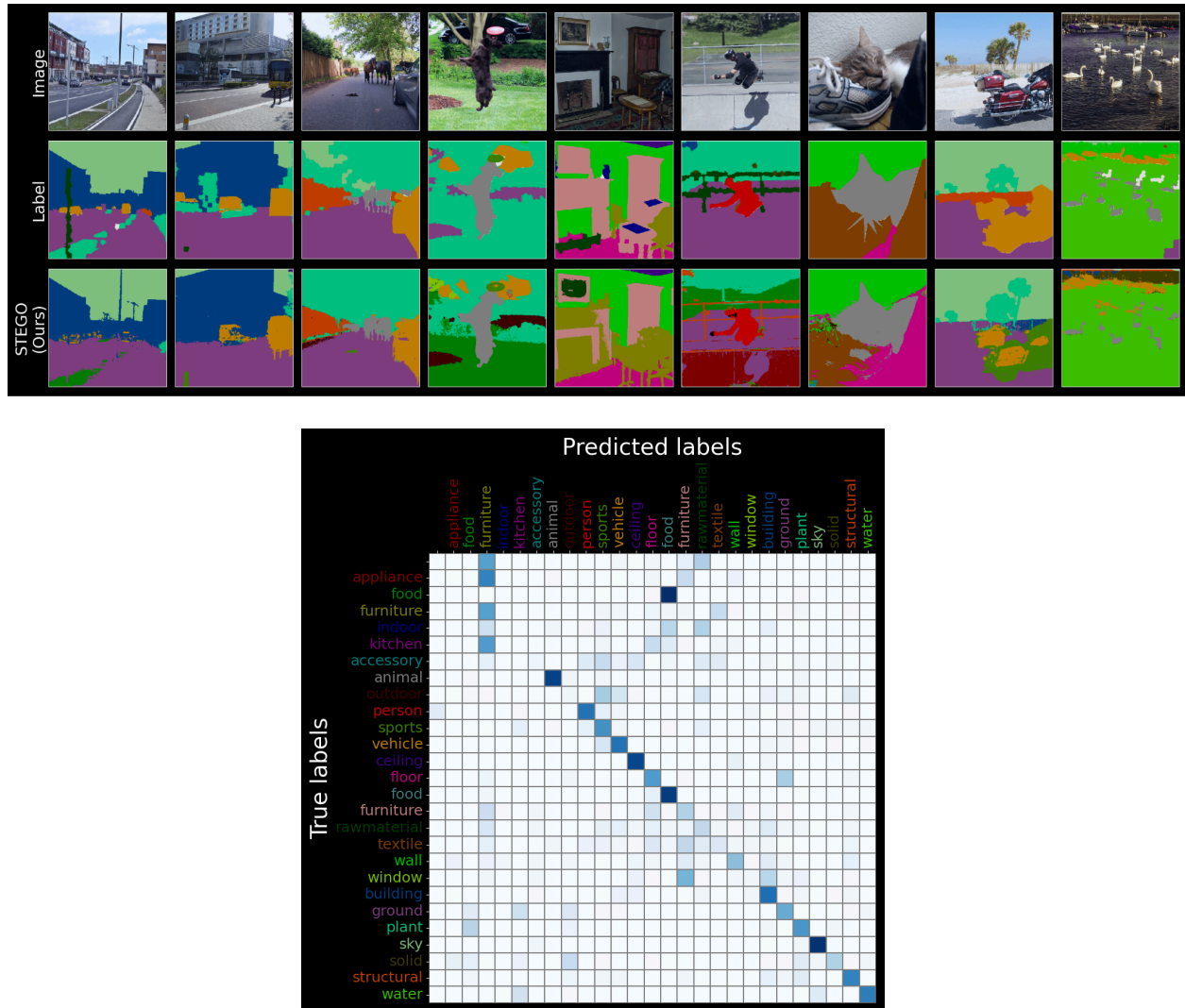


Figure 3 matches the results shown in Figure 1 of the paper nicely, the evaluation metrics were as follows:

| Metric                | Value     |
|-----------------------|-----------|
| final/linear/mIoU     | 41.074637 |
| final/linear/Accuracy | 76.128519 |

|                        |           |
|------------------------|-----------|
| final/cluster/mIoU     | 28.187278 |
| final/cluster/Accuracy | 56.929111 |

## 2. Attempting to train from scratch

To train the model from scratch, possibly with different hyperparameters, should be simple enough. In order to test the process, I wrote a small script to effectively get a subset of the original data (>98000 images) and put this into a new `cocostuff\_small` dataset which I then pointed to in the Coco dataset class.

Unfortunately there were still some limiting factors, when running the `train\_segmentation.py` script to train the model it was showing the the first tensor input was a scalar value. This raised an Exception in the plotting function, but outlined an issue with reading the data since this should not be a scalar tensor.

## 3. New Dataset: Evaluation of PASCAL Visual Object Classes (2012)

In this section, we'll experiment with a different dataset to understand how well the model can generalize to different classes and to various types of scenes.

### Experimental Setup

We'll use the PASCAL VOC 2012 dataset for the experiment, as it provides varied object classes and scenes. The steps include:

1. Preprocessing the datasets to fit the input requirements of our model.
2. Running the model on these datasets without any additional fine-tuning.
3. Comparing the results with the baseline performance on the cocostuff27 dataset.


### Method

In order for the images to be correctly read by the model, some preprocessing is required. All the images must be placed in the directory where the PyTorch datasets are read, with a specific structure:

```
'''
```

```
| dataset_name
```

```
| -- imgs
```



```
|---- train
|----- 000001.jpg
|---- val
|----- 000003.jpg
|-- labels
|---- train
|----- 000001.jpg
|---- val
|----- 000003.jpg
'''
```

Since the model should be able to train on datasets without labels, the corresponding directory will be omitted so that only the images are supplied. After some minor debugging, the `crop\_datasets` and `preprocess\_knns` scripts ran without issue but once again the issue arose in `train\_segmentation`. These appear to be ongoing issues in the author's repository, which will hopefully be remedied as I could not get the model to train successfully.