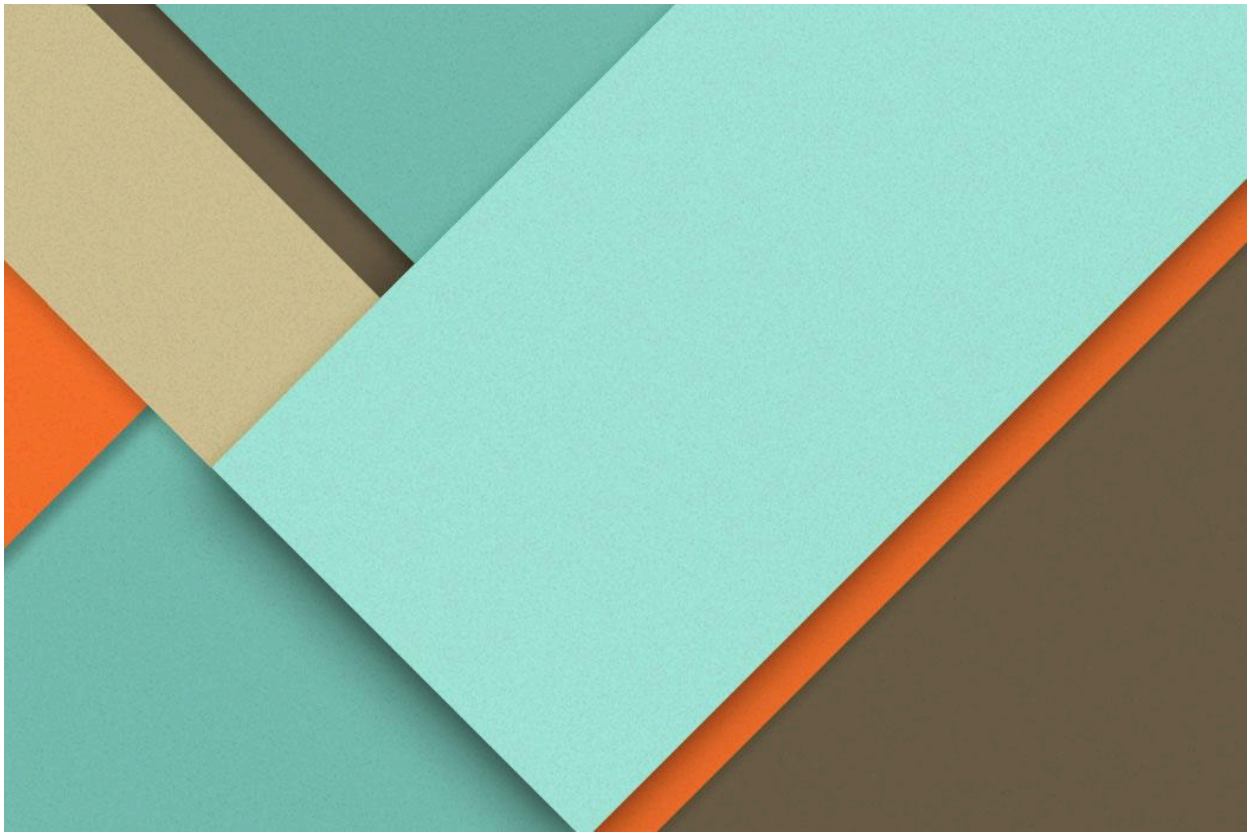

Reproducing “Unsupervised Semantic Segmentation by Distilling Feature Correspondences”

14/06/2024

—

Chris Overtveld
TU Delft - 4423621

Introduction

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 initial evaluation on the `cocostuff27` dataset and evaluating the model's performance on different datasets.

First, perhaps a brief recap on what semantic segmentation is and how it is most commonly achieved. Semantic segmentation involves classifying each pixel in an image into a predefined category, unlike more classical image classification methods which assigns a single label to an entire image. Semantic segmentation provides a more detailed understanding by being able to label specific parts of an image. This is also achieved through object classification tasks, but object classification is more general still as it only provides a bounding box around detected objects rather than highlighting pixel-by-pixel where that object is.

Traditional approaches to semantic segmentation involve thresholding and clustering which group similar pixels based on color or texture or region-based methods which group neighboring pixels with similar properties. These approaches pre-date the far more common deep learning-based approaches which leverage Convolutional Neural Networks (CNNs), Encoder-Decoder Architectures, and more recently Attention Mechanisms.

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.

Methodology

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.

Feature Extraction

The core of the method involves extracting features from images using a pre-trained Vision Transformer (ViT). The ViT model is fine-tuned on the target dataset to ensure that the features are well-aligned with the segmentation task. The paper formes the feature correspondence tensor:

$$F_{hw_{ij}} := \sum_c \frac{f_{chw}}{|f_{hw}|} \frac{g_{cij}}{|g_{ij}|}$$

The entries of which represent the similarity between two spatial positions (h, w) of feature tensor f and (i, j) of feature tensor g. So this tensor allows us to visualize image correspondences and also correlate with the true label co-occurrence tensor. From here, the ground truth label co-occurrence tensor given segmentation class labels k and l, can be defined:

$$L_{hw_{ij}} := \begin{cases} 1, & \text{if } l_{hw} = k_{ij} \\ 0, & \text{if } l_{hw} \neq k_{ij} \end{cases}$$

The compatibility of features with semantic segmentation labels can then be measured by looking at how well the feature correspondences, F, predict the ground-truth label co-occurences, L. The authors appropriately note how the ground truth labels are not used to tune any parameters of STEGO.

Correspondence Distillation

This is where the key innovation of the method comes in, distilling correspondences between features from different views of the same scene. As shown in the high-level architecture, the work done in this paper keeps the deep network backbone (mapping image x to feature f) frozen and focuses on training a light-weight segmentation head which maps their feature space to a code space. The ultimate goal for this segmentation head being to learn a nonlinear projection that "forms compact clusters and amplifies the correlation patterns of f".

The final loss function needed for this distillation is as follows:

$$\mathcal{L}_{\text{corr}}(x, y, b) := -\sum_{hw_{ij}} \left(F^{\text{SC}}_{hw_{ij}} - b \right) \max(S_{hw_{ij}}, 0)$$

Where b is a hyper-parameter.

Clustering

Once the feature correspondences are distilled, the features are clustered to form the final segmentation. This paper applies a cosine distance based minibatch K-Means algorithm to

extract these clusters and compute class assignments. Finally, the labels are refined with a Conditional Random Field (CRF) to further improve their spatial resolution.

STEGO High Level Architecture

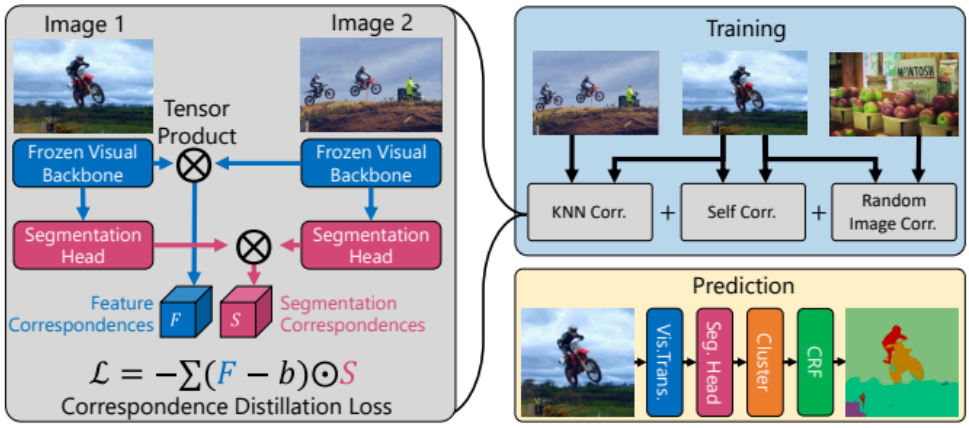

Figure 1.

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:

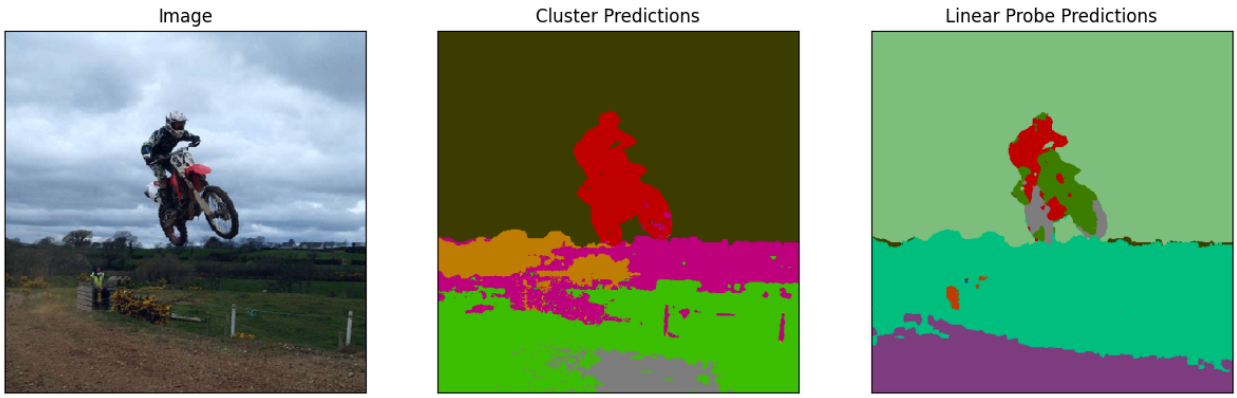

Figure 2. Moto1.jpg clustering and segmentation.

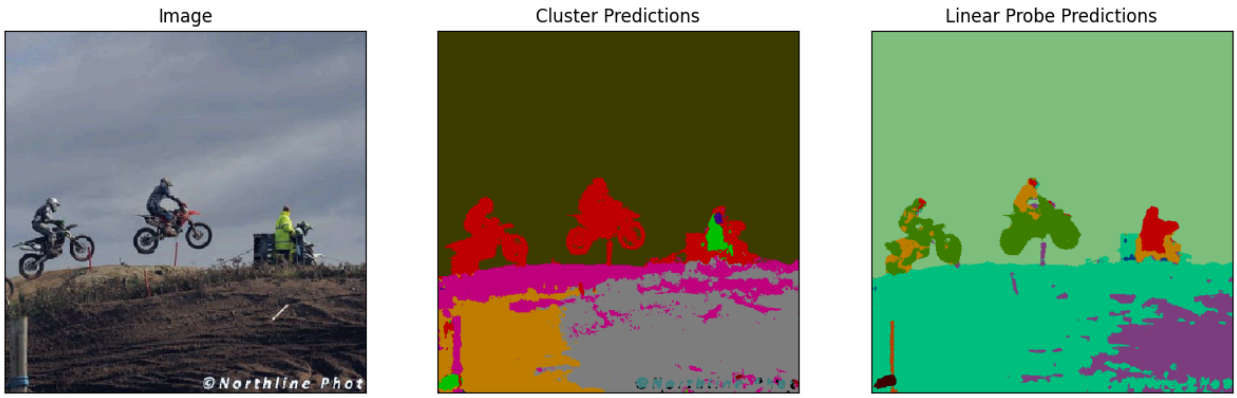

Figure 3. Moto2.jpg clustering and segmentation.

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 4. (Replicated Figure 1) Unsupervised semantic segmentation predictions on the CocoStuff (Caesar et al., 2018) 27 class segmentation challenge. STEGO does not use labels to discover and segment consistent objects. Unlike the prior state of the art, PiCIE (Cho et al., 2021), STEGO's predictions are consistent, detailed, and do not omit key objects.

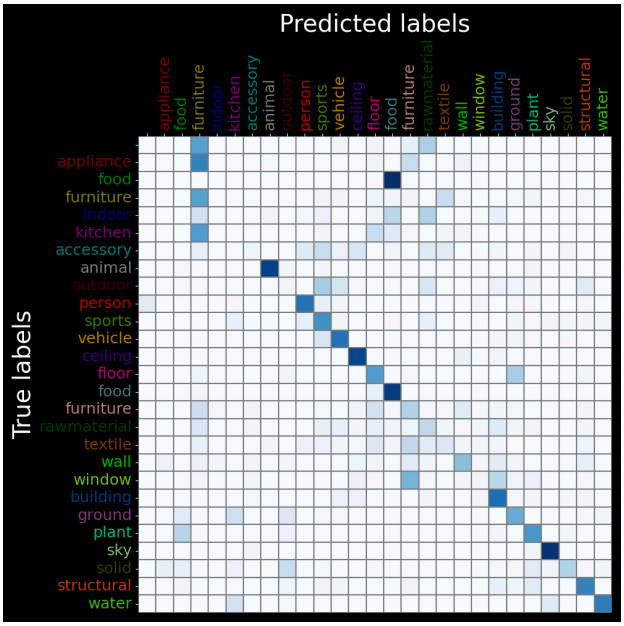

Figure 5. (Replicated Figure 6) Confusion matrix of STEGO cluster probe predictions on CocoStuff. Classes after the "vehicle" class are "stuff" and classes before are "things". Rows are normalized to sum to 1.

Figure 4 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

Table 1.

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 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.

Github:

https://github.com/COvert96/STEGO-Reproduced.git