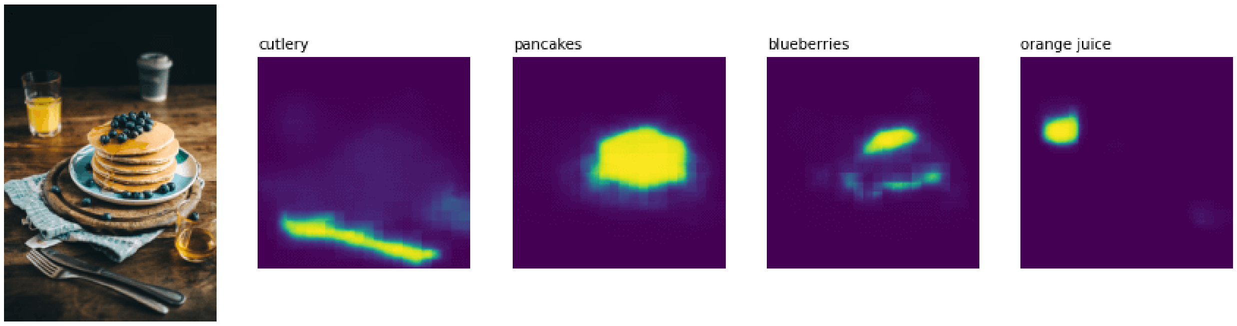
ZERO-SHOT IMAGE SEGMENTATION WITH CLIPSEG



Introduction

Image segmentation is a well-known task within the field of computer vision. It allows a computer to not only know what is in an image (classification), where objects are in the image (detection), but also what the outlines of those objects are. Knowing the outlines of objects is essential in fields such as robotics and autonomous driving.

One limitation of most image segmentation models is that they only work with a fixed list of categories. For example, you cannot simply use a segmentation model trained on oranges to segment apples. To teach the segmentation model an additional category, you have to label data of the new category and train a new model, which can be costly and time-consuming. But what if there was a model that can already segment almost any kind of object, without any further training? That's exactly what CLIPSeg, a zero-shot segmentation model, achieves.

CLIP: the magic model behind CLIPSeg

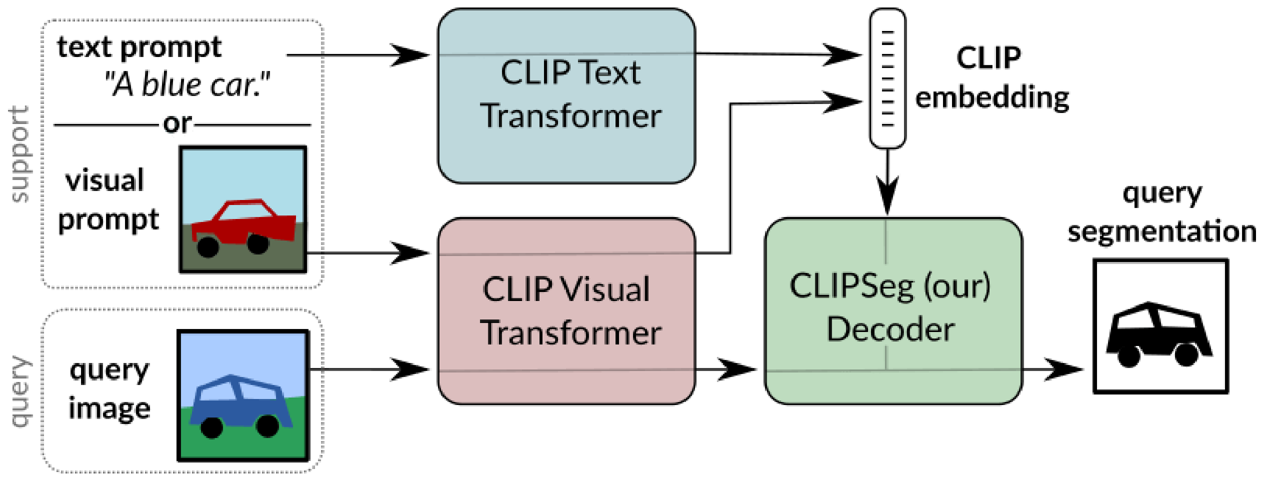
CLIP, which stands for Contrastive Language–Image Pre-training, is a model developed by OpenAI in 2021. You can give CLIP an image or a piece of text, and CLIP will output an abstract *representation* of your input. This abstract representation, also called an *embedding*, is really just a vector (a list of numbers). You can think of this vector as a point in high-dimensional space. CLIP is trained so that the representations of similar pictures and texts are similar as well. This means that if we input an image and a text description that fits that image, the representations of the image and the text will be similar (i.e., the high-dimensional points will be close together).

At first, this might not seem very useful, but it is actually very powerful. As an example, let's take a quick look at how CLIP can be used to classify images without ever having been trained on that task. To classify an image, we input the image and the different categories we want to choose from to CLIP (e.g. we input an image and the words "apple", "orange", …). CLIP then gives us back an embedding of the image and of each category. Now, we simply have to check which category embedding is closest to the embedding of the image, et voilà! Feels like magic, doesn't it?

The reason why CLIP works so well is that the model was trained on a huge dataset of images with text captions. The dataset contained a whopping 400 million image-text pairs taken from the internet. These images contain a wide variety of objects and concepts, and CLIP is great at creating a representation for each of them.

CLIPSeg: image segmentation with CLIP

CLIPSeg is a model that uses CLIP representations to create image segmentation masks. It was published by Timo Lüddecke and Alexander Ecker. They achieved zero-shot image segmentation by training a Transformer-based decoder on top of the CLIP model, which is kept frozen. The decoder takes in the CLIP representation of an image, and the CLIP representation of the thing you want to segment. Using these two inputs, the CLIPSeg decoder creates a binary segmentation mask. To be more precise, the decoder doesn't only use the final CLIP representation of the image we want to segment, but it also uses the outputs of some of the layers of CLIP.



The decoder is trained on the PhraseCut dataset, which contains over 340,000 phrases with corresponding image segmentation masks. The authors also experimented with various augmentations to expand the size of the dataset. The goal here is not only to be able to segment the categories that are present in the dataset, but also to segment unseen categories. Experiments indeed show that the decoder can generalize to unseen categories.

One interesting feature of CLIPSeg is that both the query (the image we want to segment) and the prompt (the thing we want to segment in the image) are input as CLIP embeddings. The CLIP embedding for the prompt can either come from a piece of text (the category name), or from another image. This means you can segment oranges in a photo by giving CLIPSeg an example image of an orange.

Approach

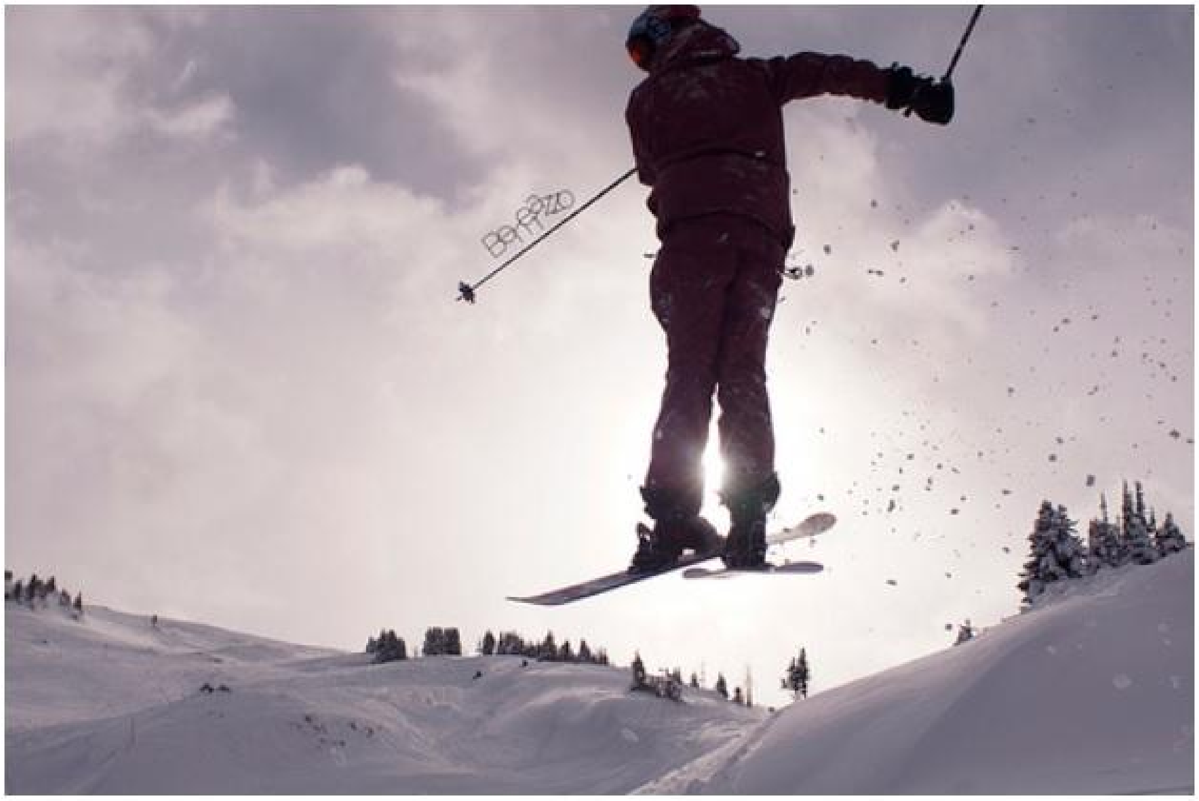
We start by installing the transformers.

Then we download a pre trained CLIPSeg model, by simply instantiating it.

Download the refer_train.json file, huge file, to the local computer, by using ctrl+click on the link; then upload it to the user's https://drive.google.com/drive/my-drive. Mount the User's google drive

Load the necessary json files from the user's google drive

Next we load a random image from the Phrasecut dataset to try out segmentation.

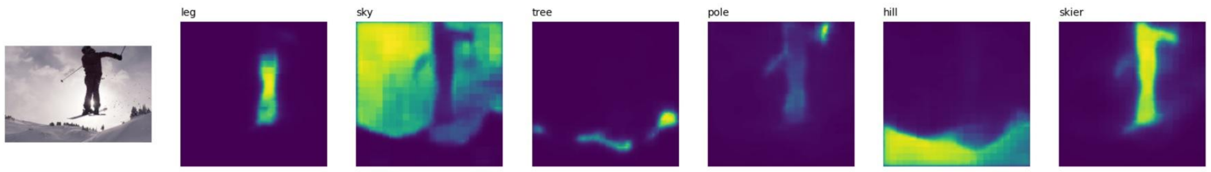


Then we load the corresponding text prompts from the Phrasecut dataset for defining the text categories that we want to segment the image

```
text prompts : ['leg', 'sky', 'tree', 'pole', 'hill', 'skier']
```

Now that we have our inputs in the form of image and text prompts, we can process them and input them to our model

Finally we visualize the output using matplotlib.pyplots



Next we calculate and display the metrics like IOU and Dice coefficient for each of the text prompts for the chosen image

```
Phrase: leg                       IOU = 81.92628420536721        Dice
Coefficient = 90.0653630817687

Phrase: sky                       IOU = 87.27646813721542        Dice
Coefficient = 93.2060167572883

Phrase: tree                      IOU = 14.316180256984227       Dice
Coefficient = 25.046638585721

Phrase: pole                      IOU = No Overlap               Dice
Coefficient = No Overlap

Phrase: hill                      IOU = 88.07142643246392        Dice
Coefficient = 93.6574237810550

Phrase: skier                     IOU = 59.78921039375981        Dice
Coefficient = 74.835103379539

Mean IOU :55.22992823763177

Mean Dice Coefficient :62.801757597562094
```

Lastly we perform the Welch T Test for the segmented image for each of the text prompts, taking the following as inputs:

- Two sets of pixel populations both inside of the segmented object

- One set of pixel population fully inside the segmented object and another set which is both inside and outside the segmented object

```
Phrase : leg

Welch T Test for two sets of pixel populations both inside of the
segmented object : Ttest_indResult(statistic=-1.7764823576180733,
pvalue=0.07719019393086203)

Welch T Test for two sets of pixel populations inside and outside of the
segmented object : Ttest_indResult(statistic=-169.15117167132033,
pvalue=3.326984122519277e-134)



Phrase : sky

Welch T Test for two sets of pixel populations both inside of the
segmented object : Ttest_indResult(statistic=0.4441738649321615,
pvalue=0.6574257251641284)

Welch T Test for two sets of pixel populations inside and outside of the
segmented object : Ttest_indResult(statistic=-7.639924145282616,
pvalue=8.305993843651495e-12)



Phrase : tree

Welch T Test for two sets of pixel populations both inside of the
segmented object : Ttest_indResult(statistic=0.296025882922104,
pvalue=0.7675246309346118)

Welch T Test for two sets of pixel populations inside and outside of the
segmented object : Ttest_indResult(statistic=-141.6842921788023,
pvalue=1.2807441171788013e-140)
```

```
Phrase : pole

Welch T Test for two sets of pixel populations both inside of the
segmented object : Ttest_indResult(statistic=0.0, pvalue=1.0)

Welch T Test for two sets of pixel populations inside and outside of the
segmented object : Ttest_indResult(statistic=-8.420982836984088,
pvalue=1.201352517400679e-14)




Phrase : hill

Welch T Test for two sets of pixel populations both inside of the
segmented object : Ttest_indResult(statistic=-13.677546605732324,
pvalue=1.39315776043328e-26)

Welch T Test for two sets of pixel populations inside and outside of the
segmented object : Ttest_indResult(statistic=-111.30824835905237,
pvalue=8.425693358212383e-112)




Phrase : skier

Welch T Test for two sets of pixel populations both inside of the
segmented object : Ttest_indResult(statistic=-3.613028271443276,
pvalue=0.0003950413342629438)

Welch T Test for two sets of pixel populations inside and outside of the
segmented object : Ttest_indResult(statistic=-94.63973994309983,
pvalue=6.7342398670810655e-136)
```

Conclusion

CLIPSeg is a zero-shot segmentation model that works with both text and image prompts. The model adds a decoder to CLIP and can segment almost anything. However, the output segmentation masks are still very low-res for now, so you'll probably still want to fine-tune a different segmentation model if accuracy is important.

Welch t test is a 2 sample location test which is used to test the null hypothesis that 2 given samples of unequal variance have the same means. In our particular case, we perform this test on samples consisting of differences in pixel intensities between pixels only located within a bounding box that encloses an object that is detected, and difference in pixel intensity between pixels in the bounding box and just outside it.

Our null hypothesis states that the 2 groups we have taken have the same mean and as such pixel intensities vary smoothly across object boundaries as well as within objects. Therefore, our alternate hypothesis states that the 2 groups have different means and hence pixel intensities within an object vary smoothly within an object but not across object boundaries.

For our purpose we test the null hypothesis with a significance level of 0.001 and find that for every image tested, the p value for each image segment(segmentation performed on the basis of text prompts from the phrasecut dataset) comes out to be lesser than the significance level. This indicates that the alternate hypothesis is true and the means of both samples are not identical. As a result, we observe that pixel intensities vary smoothly within an object as opposed to intensities that cross object boundaries or are from different objects.