SimCLR training and evaluation

To be noted: there are two different trainings in this implementation (although you can choose not to do the second one). First, a SimCLR model is trained using 100% of the image data. Second, a classifier is trained on image embeddings using 80% of the data to train and 20% of the data to validate the classifier. You can find a more thorough explanation of the general process below, and then details about each step.

Table of content

1. General process

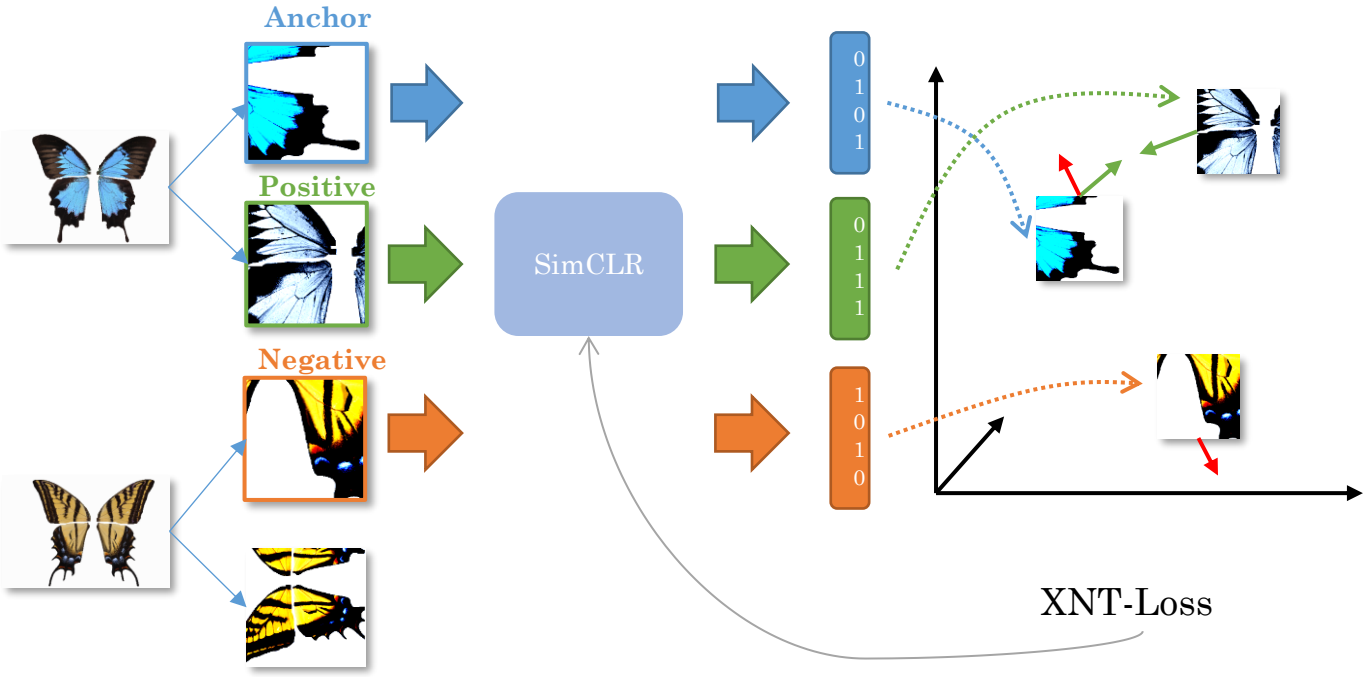
Network training

SimCLR is trained using image augmentations (modifications from original images). Augmentations that originate from a same image are labelled as "positive pairs" and augmentations that do not originate from the same image are labelled as "negative pairs". The network is then trained to bring closer embeddings of positive pairs in the embedding space and move away embeddings of negative pairs in the embedding space. Eventually, the network will have learned what discriminant image features make a positive pair similar and what discriminant image features make a negative pair dissimilar.

For more about SimCLR, check out the article "*Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020, November). A simple framework for contrastive learning of visual representations. In International conference on machine learning (pp. 1597-1607). PMLR.*"

Or the blog articles: https://sthalles.github.io/simple-self-supervised-learning/, https://sh-tsang.medium.com/review-simclr-a-simple-framework-for-contrastive-learning-of-visual-representations-5de42ba0bc66

Or the lightly example https://docs.lightly.ai/self-supervised-learning/tutorials/package/tutorial_simclr_clothing.html



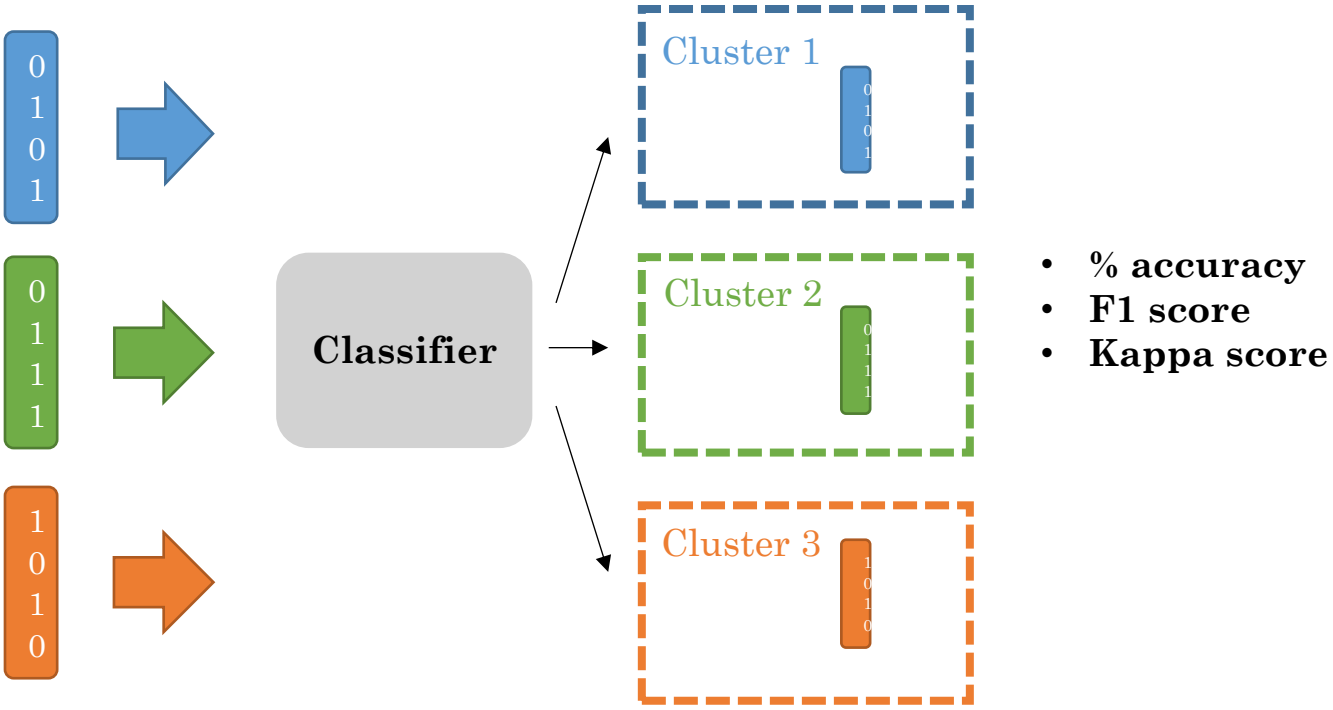
Evaluation by training a classifier

One way to evaluate the embeddings quality is to use a linear evaluation protocol.

From https://sthalles.github.io/simple-self-supervised-learning/:

"The idea is to train linear classifiers on fixed representations from the SimCLR encoder. To do that, we take the training data, pass it through the pre-trained SimCLR model, and store the output representations. […] These fixed representations are then used to train a Logistic Regression model using the training labels as targets. Then, we can measure the testing accuracy, and use it as a measure of feature quality."

Here we use clustering result as training labels for the classifier. We use clustering both for balancing the dataset during SimCLR training, and to evaluate the model during the linear evaluation protocol.



2. Step by step explanation

A. Segmentation

The segmentation code here is specialize to segment butterfly wings only (apart from the thorax and abdomen parts which are considered to be in the background). There are two main steps to the segmentation algorithm: (1) Segmentation of the wings based on a trained Mask-RCNN segmentation model, which is a machine learning based method that identifies pixel from each wing. This segmentation is not perfect, especially around the border of the wings, and thus there is step (2), a refinement step in which the background of the entire image is removed using the rembg package, leaving the wings, body and antennae, and then the resulting image is combined with the raw segmentation masks from the machine learning to produce refined masks with a clean border of the wings.

Figure 1: Original image

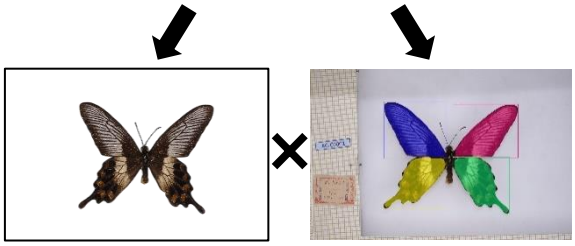


Figure 2: image with background removed

Figure 3: image with raw segmentation masks

Figure 4: refined mask for posterior right wing

Figure 5: segmented wings (refined)

B. Clustering

This step creates cluster of similar phenotypes from another ImageNet trained neural network. These clusters are used for two reasons: (1) to balance the dataset if some phenotypes are over or underrepresented. (2) to later evaluate classification into these clusters from the embeddings obtained from the unsupervised training and thus have an idea of embedding space quality regarding these clusters.

If you wish to rely on your own self-made clusters, you should create a clustering folder within which there is a subfolder for each cluster containing corresponding pictures, and skip “1-VGG16_clustering”, as such (folders/images names are not important):

clustering_folder

|__**cluster_0**
|__image_from_group_0
|__ ...
|__**cluster_1**
|__image_from_group_0
|__ ...
|__ ...

When running "1-VGG16-Clustering", first all picture are run through a pretrained VGG16 network, and then the resulting embeddings are projected onto 3 dimension using a UMAP projection. For the HDBScan clustering, you will be asked to choose two parameters (check HDBScan documentation for more information). Then the 3D UMAP projection will be plotted along with clustering results as a colored scatterplot, as well as the clusters silhouette scores. Close the window, and then you can either keep this clustering result (type 1) or try with new parameters (type 0).
(Code for the plots was inspired from this page: *https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html*)

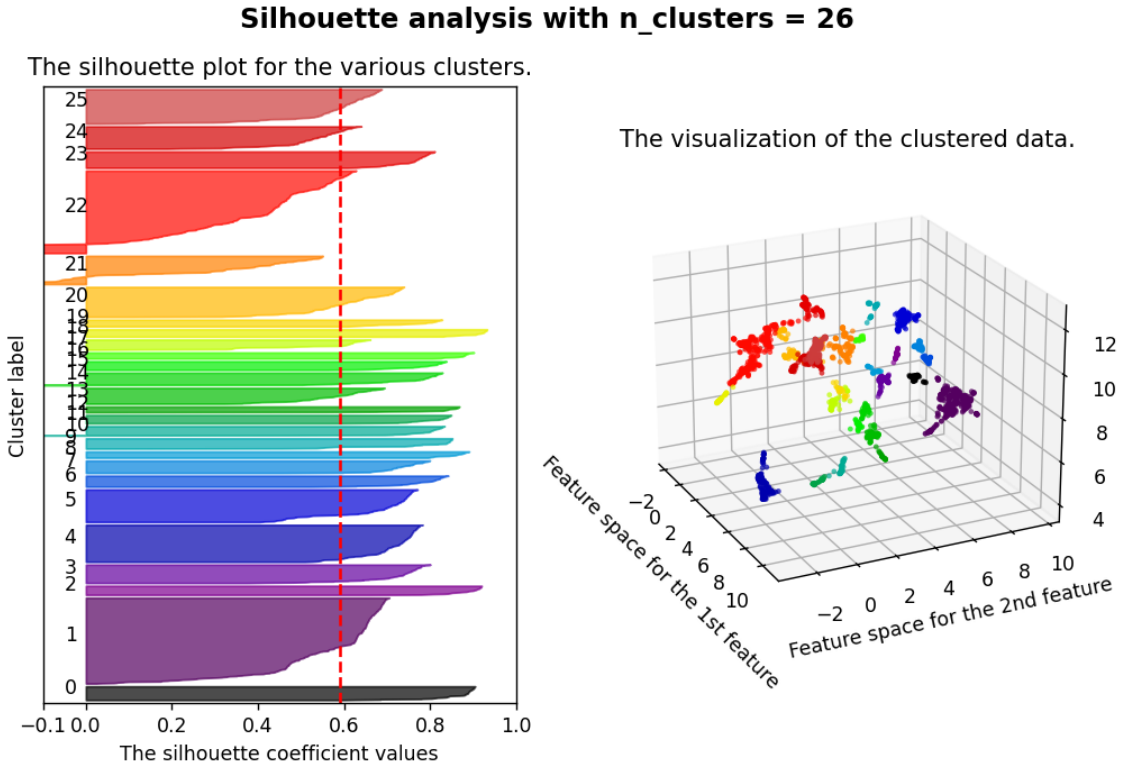


Figure 6: Silhouette and 3D scatter plot of pretrained-VGG16 image embeddings

```
Choose min_cluster_size: 20
Choose cluster_selection_epsilon: 0.1
Number of clusters: 25
Number of outliers: 599
For n_clusters = 26 The average silhouette_score is : 0.5898127
```

Figure 7: Terminal output for the clustering

After clustering, “2-create_dataset” uses these folders to create a dataset folder within which you will find: the training set that will be used to train SimCLR, which contains all of your picture data, in the folder “unlabeled” ; the training set that will be used to train the classifier for evaluation of the model, containing 80% of your data, in the folder “train” ; the validation set that will be used to train the classifier for evaluation of the model, containing 20% of your data, in the folder “validation”.

dataset_for_SimCLR_training_and_evaluation

|__**unlabeled**
|__cluster_0
|__ images
|__cluster_1
|__ images
|__ …
|__**train**
|__cluster_0
|__ images
|__cluster_1
|__ images
|__ …
|__**validation**
|__cluster_0
|__ images
|__cluster_1
|__ images
|__ …

C. Training

To change the hyperparameters of the network, change the lines 47 to 51 of “1-simCLR_training” (bold font values):

```
#To use pre-determined parameter values
if param == -1 :
  batch_size=256
  max_epochs = 300
  temp = 0.5
```

Or use the param_i argument (value 1 to 27) to explore the parameter grid, which is made of combinations of the following parameters:

```
batch_size = [64, 128, 256]
max_epochs = [50,100,300]
temp = [0.1, 0.5, 0.8]
```

To find out more about those hyperparameters and their recommended values, I recommend to read the SimCLR article: *"Chen, T., Kornblith, S., Norouzi, M., & Hinton, G. (2020, November). A simple framework for contrastive learning of visual representations. In International conference on machine learning (pp. 1597-1607). PMLR. »*

After training, the trained model will be saved as:
simclr_bs_[parameter]_nepochs_[parameter]_temp_[parameter].ckpt

D. Embed

From the saved trained model, you can generate embeddings for your images using "1-generate_embeddings". This code will save two files: (1) a csv file containing raw 2048-dimensional coordinates, with images in rows, and each dimension in columns, and (2) a csv file with PCA projection result, keeping axes that explain more than 0.5% of the total variance, still with images in rows and dimensions in columns.