Practical Machine Learning, Unsupervised Task on Individual Dataset

The dataset chosen by me was a dataset of 7 classes of animals.

The 2 methods of clustering chosen were DBSCAN and Clustering by Unmasking.

The 2 methods of feature extraction were Histogram of Oriented Gradients and Bag of Visual Words.

HOG

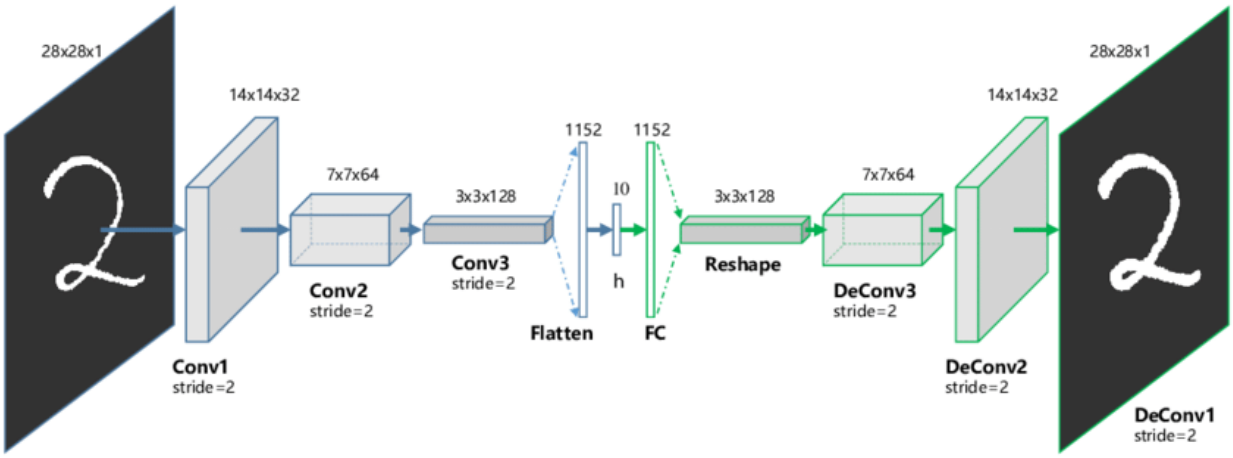
Firstly, I resized the images with the same width and height (of 128 pixels), in order to apply Convolutional Networks easier (explained further below).

I applied HOG on the resized images. The figure below is an example of a picture initially, resized version, and HOG version.

HOG does not keep any features regarding color, so the classifier/clustering method will have to do without it. However, it could help Convolutional Neural Networks with recognizing shapes easier, as it gets rid of everything besides the shapes and the orientation of the lines.

In order to be able to cluster the images, I need to transform then to a flat 1-D array. Just simply flattening the image by reshaping it might not keep all of the distinctive features of a class (a specific animal), so I decided to build my own Autoencoder.

An Autoencoder consists of 2 parts: the encoder, and the decoder. The encoder is a Convolutional Neural Network, meant to reduce the size of the image with each layer, and flatten it with the last fully connected layer. The decoder repeats the same steps, in reversed order. Its role is to reconstruct the image correctly. By training an autoencoder on my resized data, I can make sure that the flattened image (bottleneck) contains the most important information of the image, needed for its reconstruction.



After training the Autoencoder, I will save the weights and restore them for the sole reason of using them for the Encoder part. This is how I flattened the HOG images to a 1-D array of size (512,).

BOVW

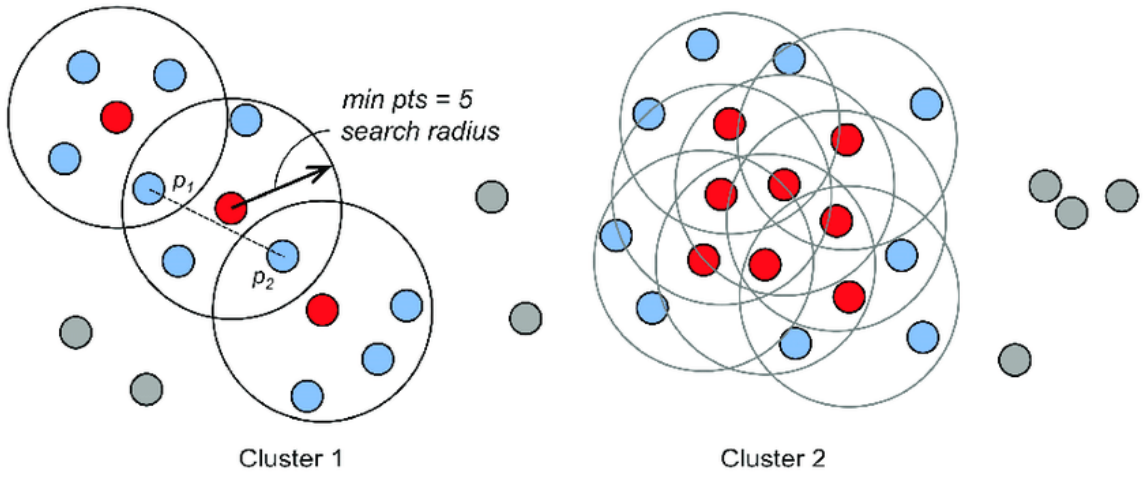
For the second method of feature extraction, BOVW, I had to extract a certain number of patches of a chosen size from my images. There is an example of 5 patches from the same image below:

After that, I got the value of the histograms by using LBP (Local Binary Pattern), and clustered the histograms with each clustering method.

When using BOVW with a classifier, I believe a dictionary of similar patches is required. While processing every image, the class can be decided by the patches of increased similarity with most appearances.

DBSCAN

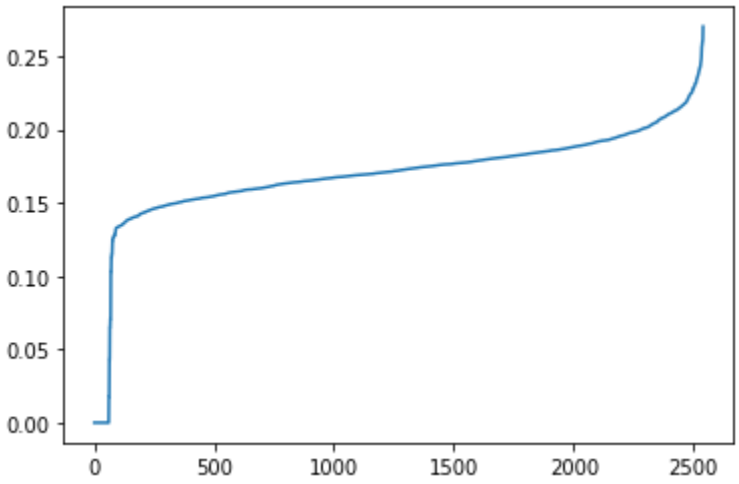
I used DBSCAN as a preprogrammed algorithm from sklearn. DBSCAN works by assigning a new data point to a cluster by judging the distance between it and other points already assigned. It starts with separating the Core Points from the Non-Core Points. Part of the Non-Core Points are outliers.



In the image above, the red points are Core-Points, the blue ones are Non-Core, and the rest are outliers.

The maximum distance between 2 neighboring points is epsilon, which needs to be determined depending on the training data.

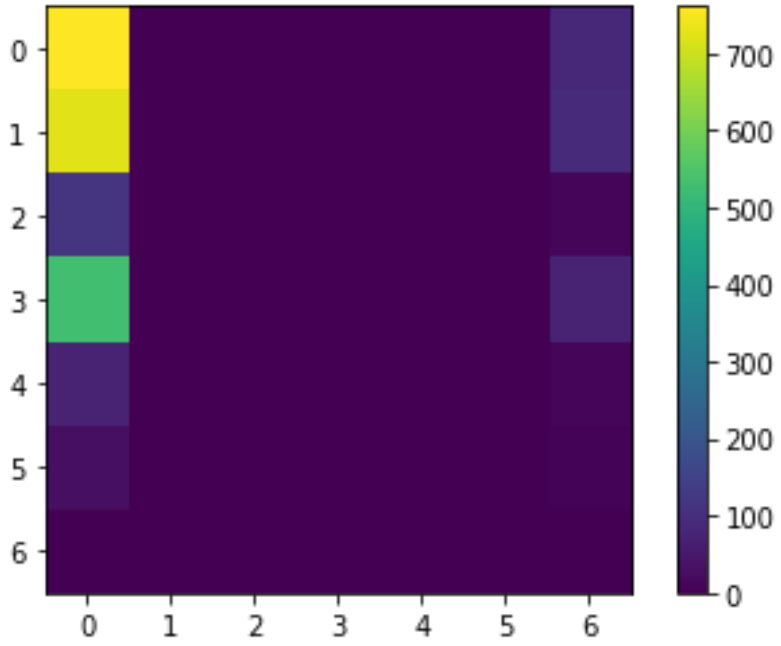
I consulted [TowardsDataScience,] which gave me an insight on how I can calculate epsilon. They used K-Nearest Neighbours and considered 3 clusters: Core Points, Non-Core Points and Outliers. After fitting the algorithm on the training data, they got the distances and plotted them. On the y-axis, there are different values that would work for epsilon, but the most optimal value designates the area of maximum curvature. For example:



Is the figure I got for DBSCAN on HOG. Most optimal epsilon seemed to be 0.1375, and second best – 0.22.

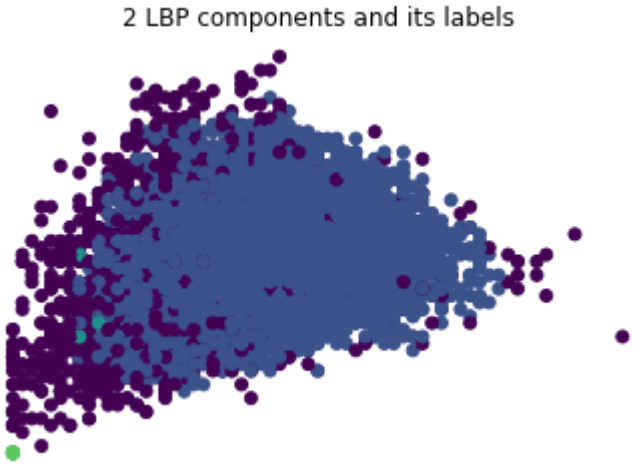
However, DBSCAN didn't work well on my dataset, as I decided to reduce the number of examples for a faster execution. Most of the time, DBSCAN chooses to split the data into 2-4 clusters, instead of 7.

An example of the confusion matrix:



On the y-axis are the actual labels, and on the x-axis – the predictions.

DBSCAN clustering with BOVW and LBP:



Clustering by unmasking

I have manually implemented this method by reading the pseudo-code. I thought the method was interesting, because by dropping the features with higher weight value, the model is forced to rely on the remaining features, which don't bring as high of a result and I might even call them "underdeveloped". Two data points of the same class will have the remaining features very

similar to each other, and the dropped features are what mostly separates them. This algorithm reminded me of Drop-Out Regularization, even though the context and usage is different.

In order to calculate the similarity between 2 images, I decided to compare the values of the encoded images, either the lbp values. I considered 2 values of 2 arrays similar, if their difference is smaller than a threshold. For the Encoded images, the difference is a maximum of ~0.1, so I chose the threshold = 0.1/7 (number of classes). The similarity of 2 clusters is equal to the sum of similarities of 2 different images of 2 clusters, divided by the number of combinations of 2 out of len(cluster1) + len(cluster2).