Clustering with K-Means and DBSCAN

This project consists of implementing two of the most popular clustering methods. If is clustering it means it is about unsupervised algorithms. DBSCAN stands for **Density-based spatial clustering of applications with noise** and on the other hand the **"means"** in the K-Means refers to averaging of the data which is finding the centroids.

The hyperparameter I needed to set for K-Means was the number of clusters in which to split the images and for DBSCAN algorithm I needed to set was the epsilon value – which is the maximum distance of a circle that should contain a minimum number of elements – and the number of elements in the radius.

There is no algorithm that fits for all purposes. This means, that there are situations where DBSCAN is highly performant, while sometimes its performance is pretty bad.

Density clustering (for example DBSCAN) seem to correspond more to human intuitions of clustering, rather than distance from a central clustering point (for example K-Means).

Density clustering algorithms use the concept of reachability i.e. how many neighbors has a point within a radius. DBSCAN is different than K-Means because it doesn't need parameter, k, which is the number of clusters that it is trying to find, which K-Means needs. When the number of clusters is not known and there's no way to visualize your dataset, it's a good decision to use DBSCAN. DBSCAN produces a varying number of clusters, based on the input data.

Advantages of KMeans and DBSCAN:

- KMeans is much faster than DBSCAN
- DBSCAN doesn't need number of clusters

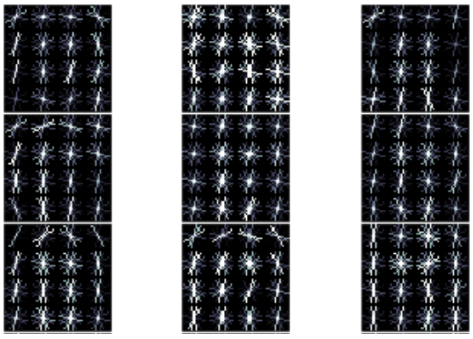
Disadvantages of KMeans and DBSCAN:

- K-means need the number of clusters hidden in the dataset
- DBSCAN doesn't work well over clusters with different densities
- DBSCAN needs a careful selection of its parameters

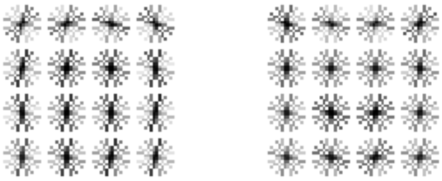
The dataset I have used consists of 160 balanced images with cats and dogs. Each image had only one-color channel on 8-bits. The dataset has been previously labeled due to the fact it was initially used for supervised learning so I had also the possibility to display the confusion matrix.

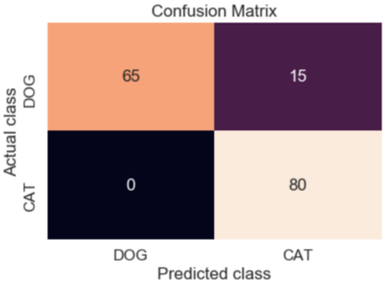
1. K-Means implementation

For clustering images using K-Means I previously extracted the **Histogram of Oriented Gradients** for each of the images. HOG had used 8 orientations in order to depict the image so a part of the plotted HOG is attached below.

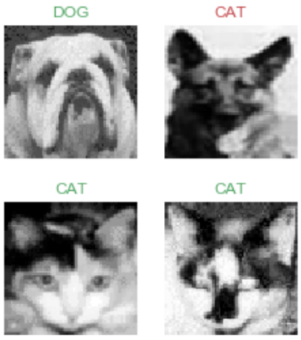


After getting a HOG matrix with all the images I have reshaped the matrix to (160, 4096) and I fitted the data with K-Means algorithm obtaining 2 centroids that look like the image attached.



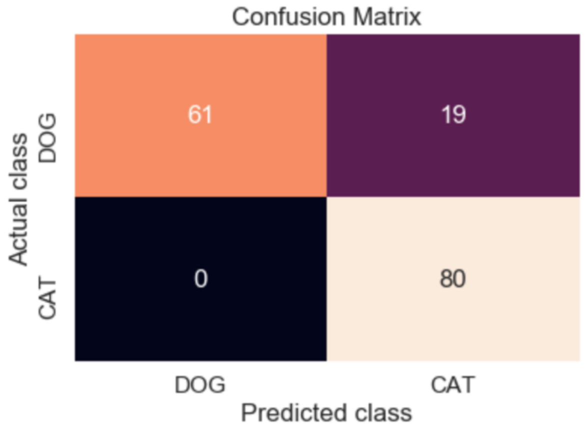


The following step was to reveal all the labels and compare them with the ground-truth using the confusion matrix. On the horizontal are the actual classes and on the vertical are the clustered classes. It seems a very good performance on clustering the cats, none of them was missed. This could be due to the fact that all the cats in dataset had the ears oriented straight up and that could make the difference, also those 15 dogs clustered as cats they had the ears in a triangle shape and oriented upward as cats' images. The overall accuracy of the model is 0.9.



The second implementation of K-Means algorithm uses the feature vector generated by the VGG16.

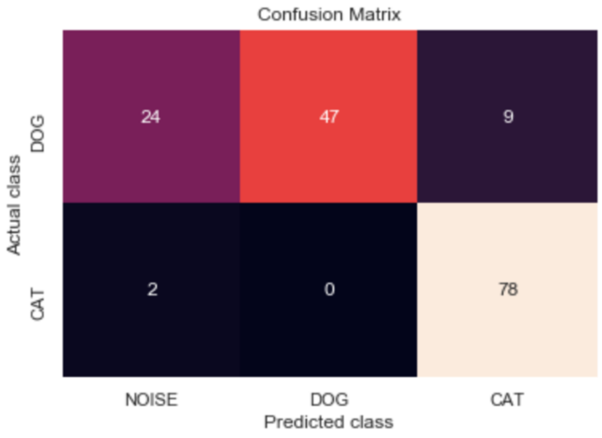
The confusion matrix for this implementation looks similar with the one for using HOG descriptor, except 4 dogs that was clustered in the same group as cats. The accuracy of the model now, got only 0.88.



2. DBSCAN Implementation

This time HOG descriptor didn't help and I have used a pretrained neural network in order to get a feature vector. I have used VGG16 from keras and I got a matrix reshaped at (160, 2048).

In order to find best parameters for DBSCAN (epsilon and number of elements) I used a for loop and I found that an epsilon of 825 and a minimum number of samples of 9 which got 2 clusters and 26 noise elements. 26 misclassified images already show that this model is not as accurate as K-Means. The following image represent the confusion matrix for this model.



If we won’t take into consideration the noises, then the accuracy would be 0.93 and if we would treat noise elements as being in the wrong cluster, then the accuracy of the model would hit only 0.78.

3. Comparison table

	Accuracy	Silhouette Coef.	Inertia	B(C)	W(C)	No. of features
Random Chance	50%					
K-Means (HOG)	90.63%	0.126	10469.8	80,328.5	74,855.6	4096
K-Means (VGG16)	88.13%	0.290	84982146.6	8698328.8	6701181.1	2048
DBSCAN (VGG16)	*78.13%**	0.211		8,105,426.4	4,506,739.3	2048

**accuracy for dbscan is calculated as stated in the end of 2nd section*