A study on image segmentation using unsupervised learning

Pătrașcu Valentin

1 Data

1.1 The dataset

The dataset used for this project was the Cityscapes dataset. The dataset consists of semantic segmentation samples, and from it, a combination of image sample, colored segmentation and label index segmentation was used. The dataset has a number of 33 classes and in the label index segmentation images, the indexes of these classes are given as the intensity value of the pixels corresponding to a certain class.

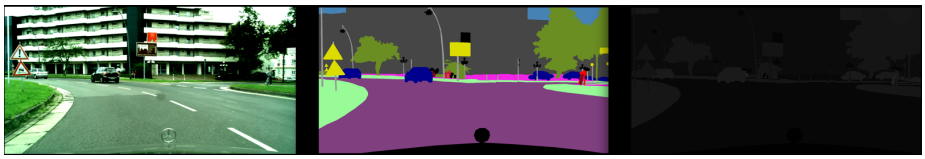

Figure 1: A sample from the dataset consisting of a) the original image, b) the colored segmentation image reference, c) the label index segmentation image reference

From the total number of 5000 images (that were distributed in train, valid and test subsets), a subset of 1046 images was extracted to be the used in this project.

1.2 Preprocessing the data

The first step was to prepare the data for the models. The original images are at a resolution of 2048x1024 pixels, and because the chosen task was going to use each image as a "mini-dataset", I had to lower the resolution of each image to 512x256.

After the resolution change, the images seemed to be a bit to dark and the color was quite faded. So, as the next step, two additional preprocessing steps were introduced to prepare the data for the models, a contrast and a brightness enhancements, both with a factor of two. These enhancements were implemented using PIL ImageEnhance module.

1.3 Modeling the features

Two different types of characteristics for the pixels in each image were used to train the unsupervised models. The first method to build the feature space was to extract the color of each pixel and to use this information to train the model. For each image, the dimension of the input into the unsupervised models was: (512x256, 3), the 3 came from the color channels, R.G.B. The second type of characteristic was the color-position. In addition to the color channels, a number of 2 other new arrays of values were introduced, the position described by the X-value and the Y-value of the pixel in the image. Thus, for this task, the input has the following shape (512x256, 5).

2 The algorithms

2.1 Unsupervised algorithms

The two unsupervised learning algorithms that were used were K-Means and DBSCAN. The implementation used for each of the algorithm was the one provided by sklearn.clusters. These methods were chosen because they are fast and well documented

2.1.1 Finding the optimal hyperparameters for the algorithms

The most important aspect of this analysis was the hyperparameter tunning and it was also the most time consuming one. The fact that each image has a different distribution of each cluster and that they are poorly balanced made this operation more difficult.

For the K-Means implementation, the most important parameter to tune was K, which directly controls the number of the final clusters. The approach to find the optimal K was to use the Elbow Method. The Elbow method takes into consideration the distortion metric that computes the sum of squared distances from each point to its corresponding cluster center. To implement the Elbow method the KElbowVisualizer module was used from yellowbrick.cluster. It takes the model as a parameter and a range for K on which the computing of distortion will be done. After randomly choosing a random number of samples (the Elbow running time was pretty long so I had to only take a few images to analyse), the results appeared as it follows.

As you can see from the Figure 2, for neither of the three images, there is no strong inflection point on the blue plot and that means that the Elbow method may not work for this particular taks. So, even if I tried to find an optimal K value, it was still to be found. The next approach was to use Silhouette but after a few attempts I gave up on this method because it was extremely slow. (after one hour, the first iteration was still not done).

The final approach that I used was that for each image, to generate the segmentation using k-means with K being in the range from 4 to 8. These numbers were chosen because from the observations, these were the numbers of clusters

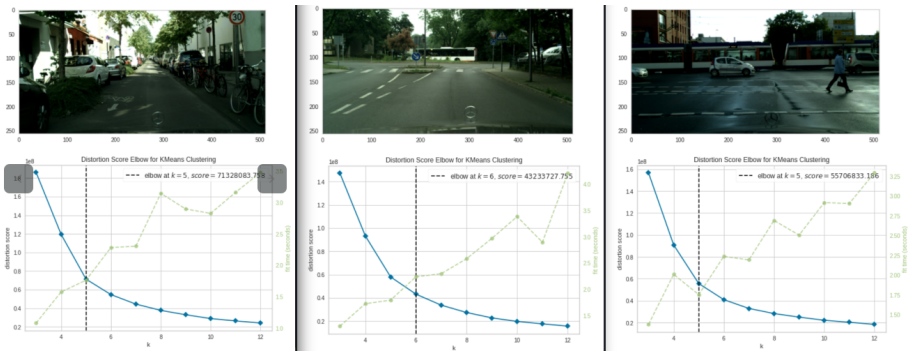

Figure 2: The example containing three visualization of the Elbow method for three different images.

with great semantic significance.

For the second unsupervised learning algorithm, DBSCAN, the most important parameters to tune were eps and min_samples. The eps parameter controls the radius of the eps-neighborhood and the min_samples controls the number of points within the eps radius.

To determine the optimal values for these parameters I used the heuristic presented in the Practical Machine Learning course that computes the distance for each point p_i to its k-th nearest neighbor, sort the points by the distance and plot the corresponding curve. Set the eps to the value of the distance where the largest change in slope is encountered.

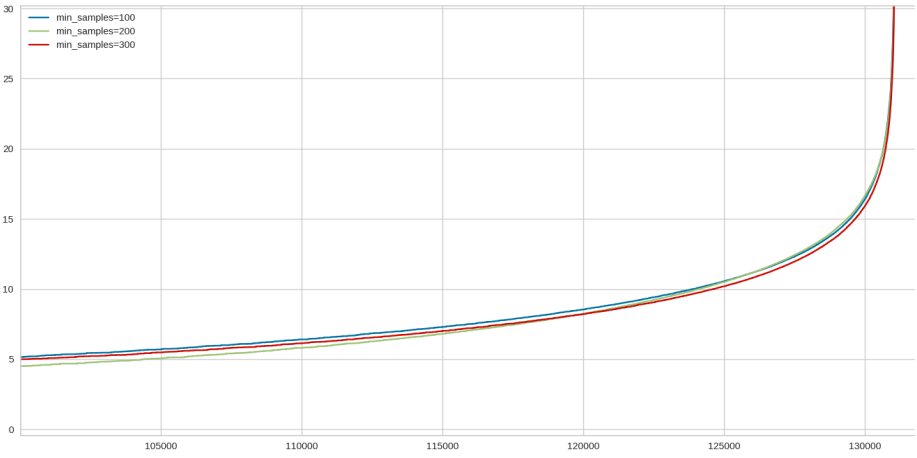

Figure 3: Figure 3: The heuristic of choosing the DBSCAN hyperparameters in the case of color-position features for three diferent min_samples. Around 15-20, the curve has the larges change in slope.

Figure 3 shows that the largest change in slope is encountered around 20. The min_samples value was set to 250. Because I wanted to have a bigger neighborhood I set the eps parameter to 28 for the color-position features. For color features, the hyperparameters were chosen in a similar form, with eps set to 4.2 and min_samples to 60.

2.2 Supervised algorithm

For the supervised algorithm I chose to use a package for PyTorch that implements different networks. The architecture used was FPN, with the resnext50_32x4d encoder and the encoder weights from imagenet.

3 Results

The metric that was used to determine the performance of the segmentation task was Adjusted Random Index. For this metric, I used the implementation from sklearn. For K-means I chose to present the results for all Ks (from 4 to 8).

Also, an important aspect was that the DBSCAN algorithms generated a number of outliers. From the observations, the percentage of outliers pixels from the total number of varied from 10 to 20 percent. To deal with the outliers, I implemented a function that takes the nearest nonzero valued pixel from the column of the zero valued pixel (I chose to search only on the column due to time efficiency issues).

Figure 4: The output of DBSCAN algorithm. The left images are before the outliers removal, where the black portions of the image are the outliers and in the right side are the corresponding modified images, each pixel having its own class (based on the neighbors classes)

	ARI
supervised	0.7549
kmeans-col-pos-k8	**0.2304**
kmeans-col-pos-k7	0.2266
kmeans-col-pos-k6	0.2191
dbscan-col-pos	0.2129
kmeans-col-pos-k5	0.2019
kmeans-col-k4	0.1914
kmeans-col-k5	0.1819
kmeans-col-k6	0.1727
kmeans-col-pos-k4	0.169
kmeans-col-k7	0.1649
kmeans-col-k8	0.1583
dbscan-col	0.1043
random_chance	-1.00E-07

Table 1: Comparison results

So, the final results and adjusted random index metric for each of the methods are as it follows.

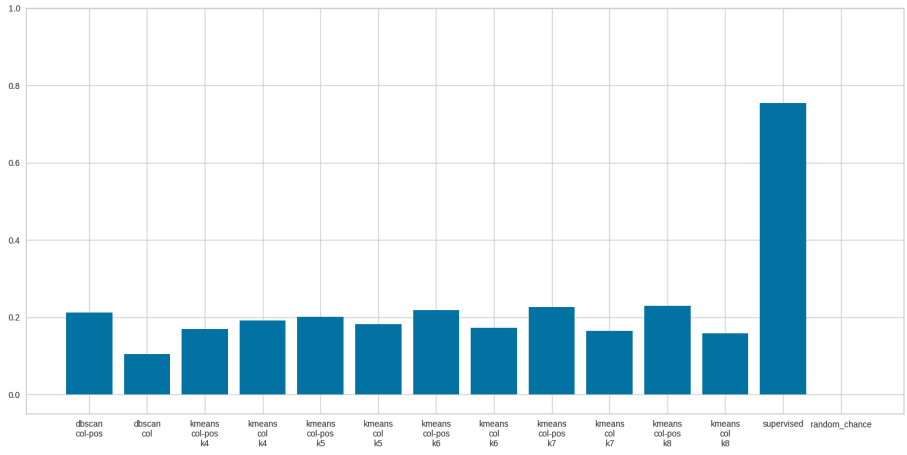

Figure 5: The final results with Adjusted Random Index metric.)

From the Table 1 we can see that the supervised method worked really well, as we would have expected, and that the kmeans with the color position method and the maximum number of posible clusters were the best unsupervised methods. I think that the reason why the DBSCAN methods have really bad results (especially the color features method) is that the number of the outliers was a little to high. It is also possible that a wider range of ks for Kmeans would give some better results.

4 Conclusions

This project proposed a real challenge in terms of understanding, resource management and efficiency. The results that the described methods proposed are not too far from the best performance that these approaches can have. A much better and on-point hyperparameter tunning can increase the performance, but the very nature of the dataset limits a lot the posibilities of much better performance because. I think that these methods can be efficient in some cases (where the images have balanced clusters, with simplier scenes), but they are not the best approach that can be used when dealing with complex scenes and with a big number of image instances.

Finally, I think that the objective was reached by having much better results than the random chance clustering, but there are a lot of improvements to be done in order to get closer to the supervised methods results.

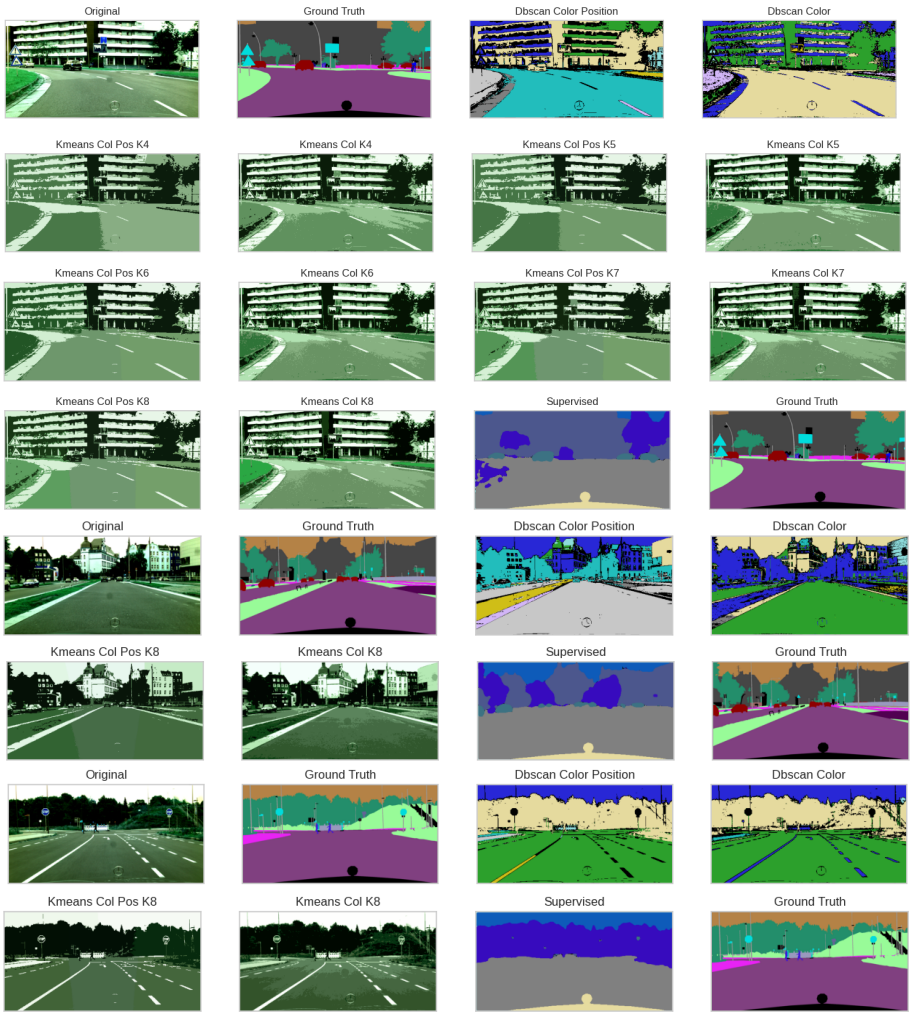

Figure 6: Visual examples of the results.