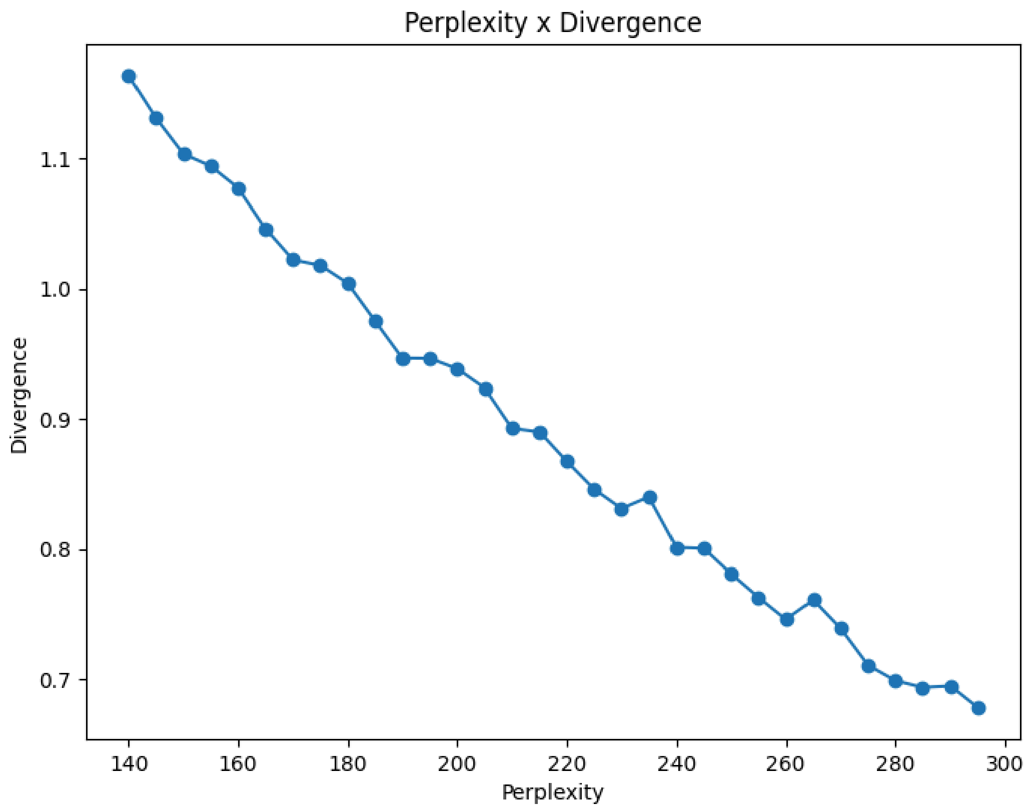
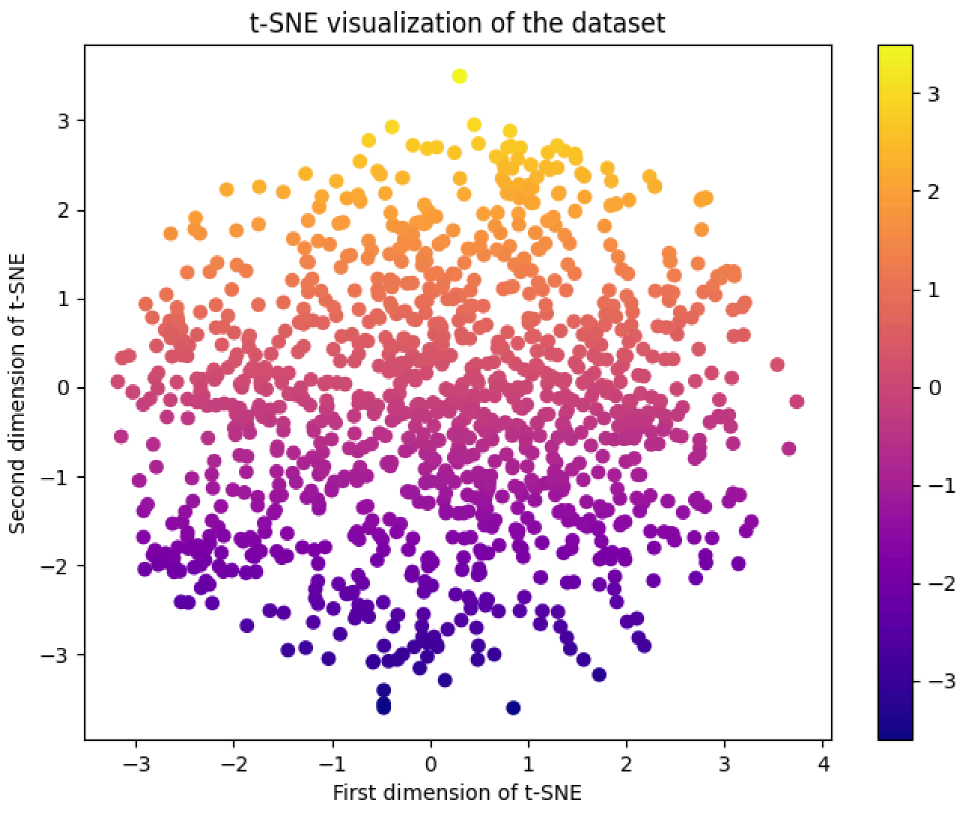
Data analysis and transformations

The data was loaded from the available pickle file, standardized and then splitted into training and testing parts, using the *train_test_split* scikit-learn function. With a simple first analysis, there were found 10 different classes of syndromes, 941 subjects and 1116 images, creating a dataset with X being the encoded images and the labels the ID of the syndrome.

As part of the requirements of the assignment, the T-Distributed Stochastic Neighbor Embedding tool was to reduce the dimensions of the dataset for visualization. For this part, upon further analysis of the reduction of the *divergence* increasing the value for the *perplexity* parameter, I've decided to stop trying to reduce its value once it got below 1.0, because the values for the perplexity were getting to high and I have considered that it was already a good result, so, on the image visualization of the t-SNE, the chosen *perplexity* value was of 280, giving a Kullback-Leibler divergence of approximately 0.84.



Searching for the best value for *perplexity*, decreasing the *divergence*.



t-SNE visualization for *perplexity* of 280.

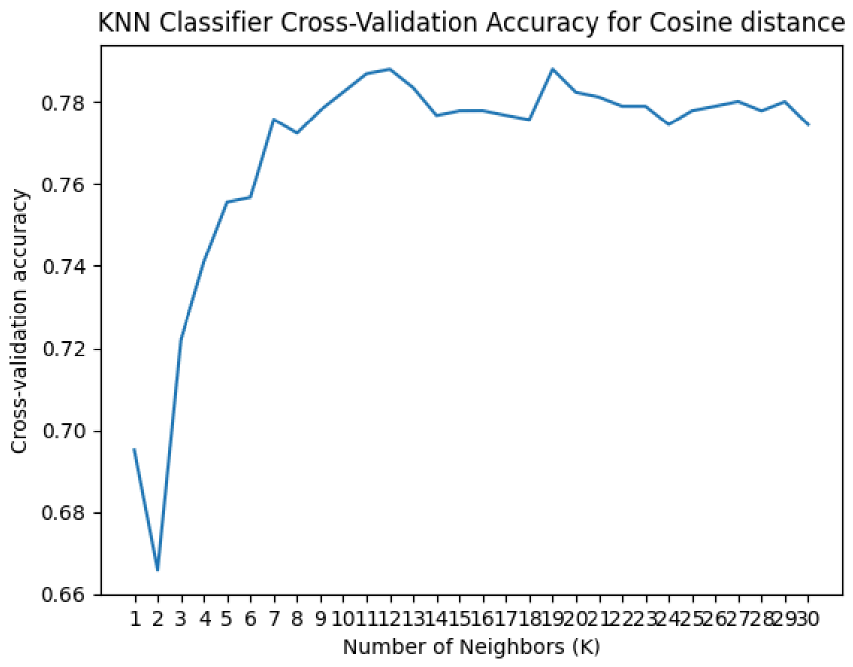
It is possible to see some patterns in the data, with a few clear clusters, suggesting that some images have common characteristics. Despite the cases of clustering, there are regions with scattered points, showing a larger diversity of features within the images. Regarding its shape, most of the points are concentrated within the center of an ellipse, appearing to have a uniform distribution, without large concentrations in any parts.

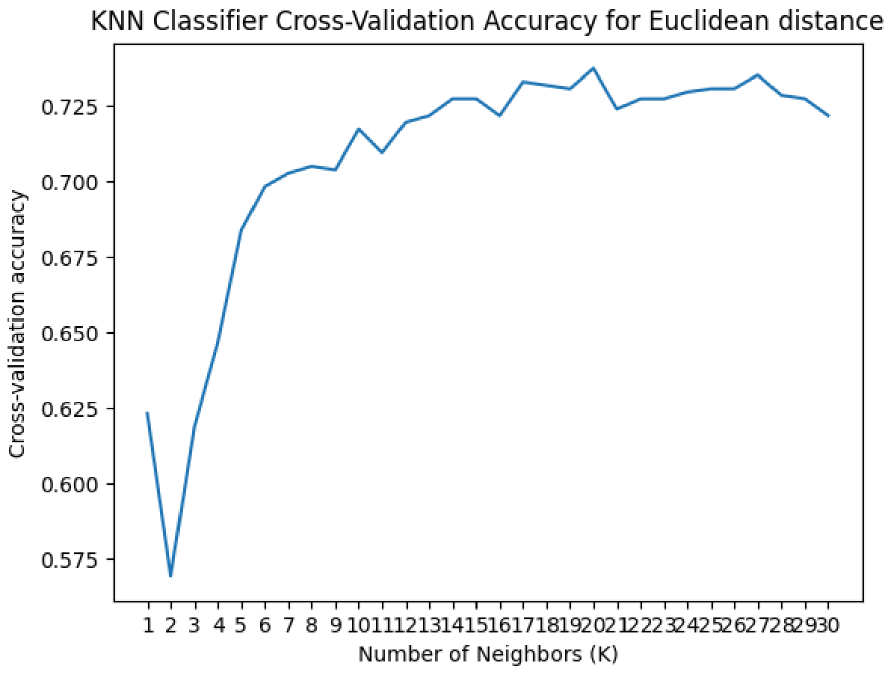
Classification

Using a 10 fold cross validation and following the requested calculations, the below results were shown (also are in a separated file, generated by the script in the *app.py* file):

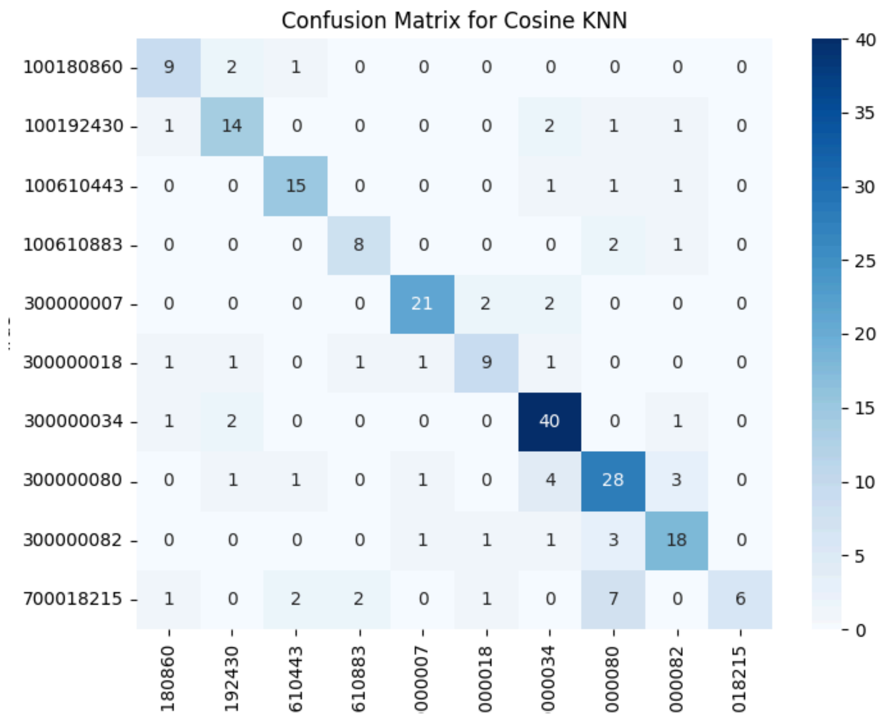
Algorithm	Best K	F1 score	Precision	Recall	Avg. distance	Avg. max. distance
Cosine	19	0.750	0.750	0.750	0.916	1.397
Euclidean	20	0.728	0.728	0.728	29.675	45.722

I’ve found it interesting that the F1 score, precision and recall all had the same values found for when using cosine and euclidean distances, but couldn’t conclude much about the reasons for that. Overall, due to the shape of the embedding and the nature of the data, the cosine distance showed a better result as a metric for the KNN classifier. Below, are the images that show the search for the best value of K (amount of neighbors) for each kind of distance:

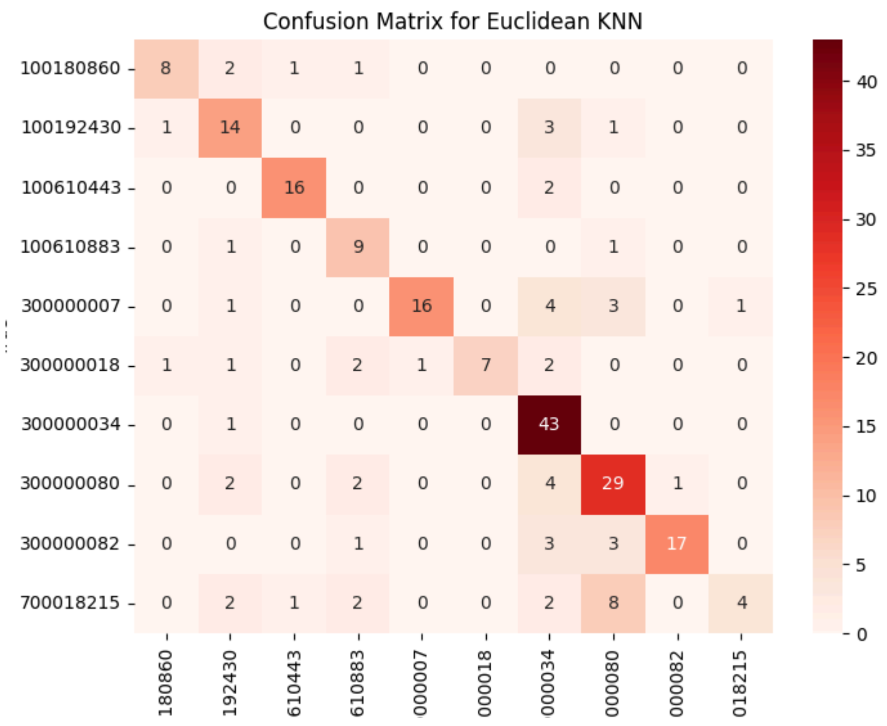




So using the best value of K for each metric algorithm, I’ve plotted the confusion matrices for both, which overall show a good result, with the models correctly classifying the images in the majority of the cases of the test set of images.

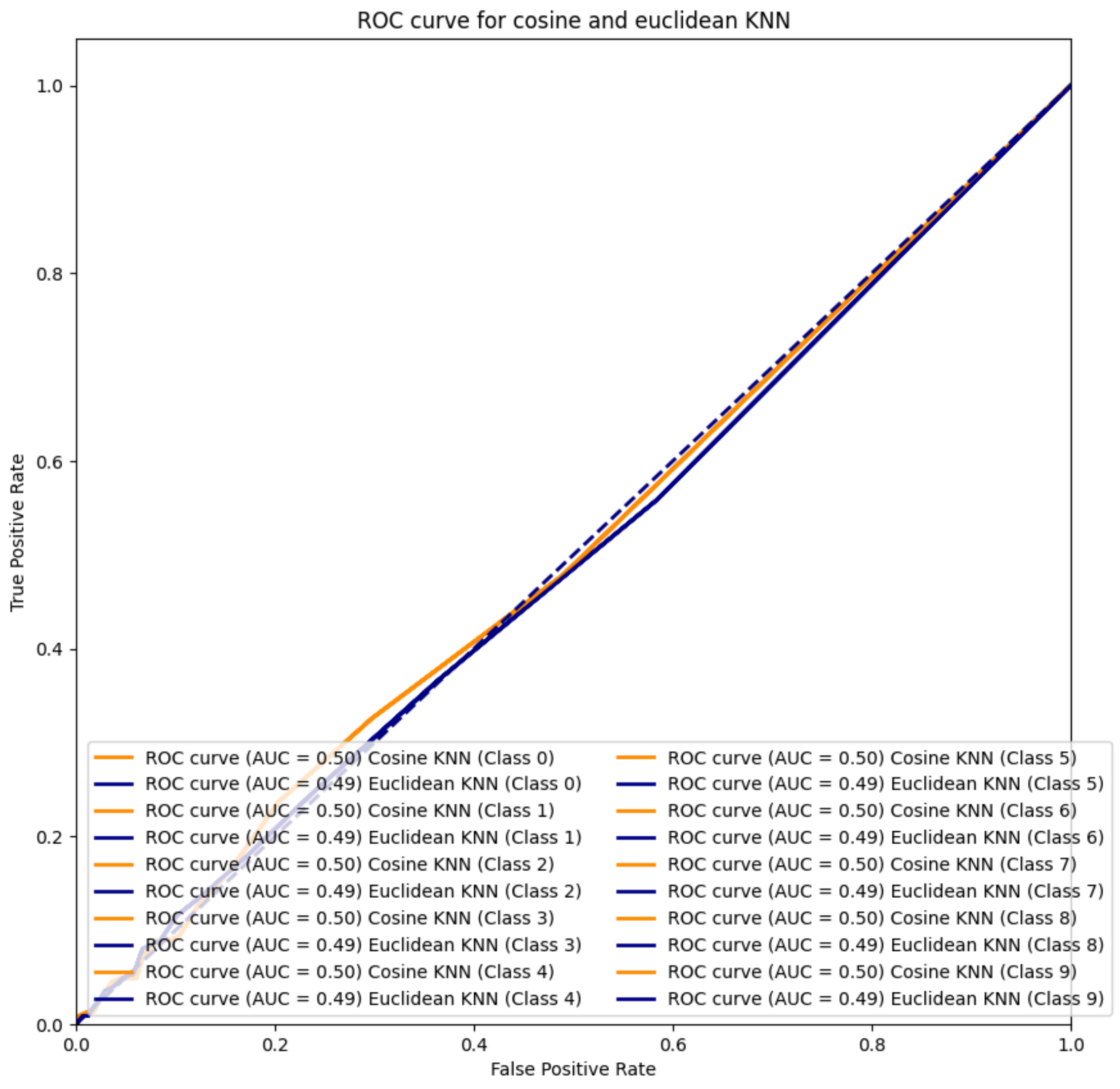


	180860	192430	610443	610883	000007	000018	000034	000080	000082	018215
100180860	9	2	1	0	0	0	0	0	0	0
100192430	1	14	0	0	0	0	2	1	1	0
100610443	0	0	15	0	0	0	1	1	1	0
100610883	0	0	0	8	0	0	0	2	1	0
300000007	0	0	0	0	21	2	2	0	0	0
300000018	1	1	0	1	1	9	1	0	0	0
300000034	1	2	0	0	0	0	40	0	1	0
300000080	0	1	1	0	1	0	4	28	3	0
300000082	0	0	0	0	1	1	1	3	18	0
700018215	1	0	2	2	0	1	0	7	0	6



	180860	192430	610443	610883	000007	000018	000034	000080	000082	018215
100180860	8	2	1	1	0	0	0	0	0	0
100192430	1	14	0	0	0	0	3	1	0	0
100610443	0	0	16	0	0	0	2	0	0	0
100610883	0	1	0	9	0	0	0	1	0	0
300000007	0	1	0	0	16	0	4	3	0	1
300000018	1	1	0	2	1	7	2	0	0	0
300000034	0	1	0	0	0	0	43	0	0	0
300000080	0	2	0	2	0	0	4	29	1	0
300000082	0	0	0	1	0	0	3	3	17	0
700018215	0	2	1	2	0	0	2	8	0	4

To finish the analysis of the models and the requested assignment, here is the image with an ROC AUC graph for comparing both algorithms.



Steps for reproducing

- The required libraries are all present in the *requirements.txt* file inside the project folder and can be installed using *pip*
    - *python3 -m pip install -r requirements.txt*
- Inside the *src* folder there are two main files:
    - *notebook.ipynb*: this is a step-by-step notebook, with some explanations and conclusions about the images being presented and allowing for experimenting with the models and other parameters;
    - *app.py*: it's the same models and steps in the notebook, but now as an executable script, that trains the models and shows information every time that is runned. It's also responsible for generating the images used in this report and the automatic one (with metrics about the two KNN models presented)