Project number: J7

Project name: Deep Clustering for Unsupervised Image Classification

Dataset: GTSRB - German Traffic Sign Recognition Benchmark

Team member: Besson Sacha; Capdeville Jérémie; Arriagada Silva Sebastián Ignacio; Mei Jiaojiao;

Individual work:

1. Hierarchical clustering (Sebastian Arriagada)
2. K-NN (Sacha BESSON)
3. - Transfering learning + clustering methods (Mei Jiaojiao)
   - kmeans
   - agglomerative
   - https://github.com/JIAOJIAOMEI/ADND-J7
4. Mean shift

try above mentioned clustering algorthms based deep learning neural networks and make a comparison of their performances

The whole experimental process is like:

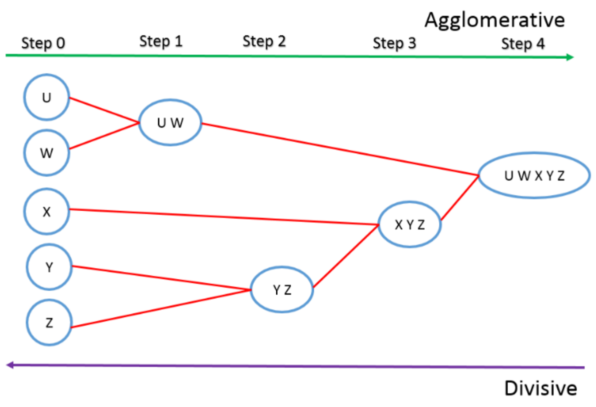
Each one finds a related neural network example code and implemente it with our imbalanced data, and evaluate your performance, at last we can merge our results and make a comparion.

- Decide what kind of clustering method you want to use
- find related neural network
- find a way to solve imbalanced data
- Implement the neural network you find and then make some modifications based on your appetite
- Training/Aplication
- Testing
- calculate accuracy, percision and AUC
- do some data visulization work

and we discuss our results together and write a report.

Hierarchical clustering

Hierarchical clustering is a type of unsupervised machine learning algorithm used to cluster similar data points together. Hierarchical clustering algorithms create a cluster tree or dendrogram, which is a way to represent the relationship between the clusters. The algorithm starts by treating each data point as a single cluster. It then iteratively merges the closest clusters until only a single cluster remains. It can be Agglomerative or Divisive depending on the approach.



Agglomerative clustering.

Input: Initialize the number of clusters to be formed.

Output: Clustered data.

Application: Consider each data point as a cluster. while clustering condition is not satisfied do perform merging of two clusters having minimum inter-cluster distance. End the while when you only have one cluster.

Divisive clustering

Input: Initialize the number of clusters to be formed.

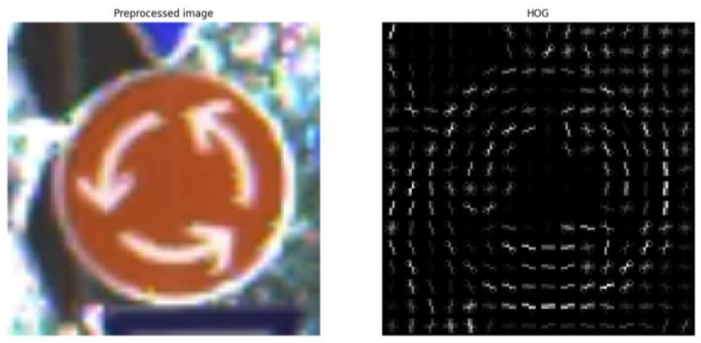
Output: Clustered data.

Application : Consider all data points as a single cluster. While clustering condition is not satisfied do Divide the cluster into two clusters resulting in the largest inter-cluster distance; end while

Application in the Dataset

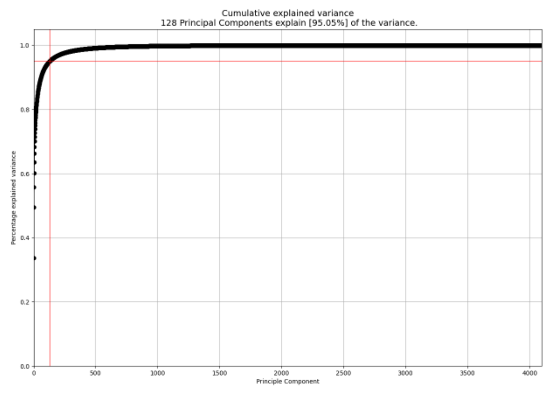
The first step is to extract the features from the image. In this case it is done through Principal component analysis (PCA) to capturate the 95% of the variance and Histogram of oriented gradients (HOG) to get the gradient orientation.

HOG example:



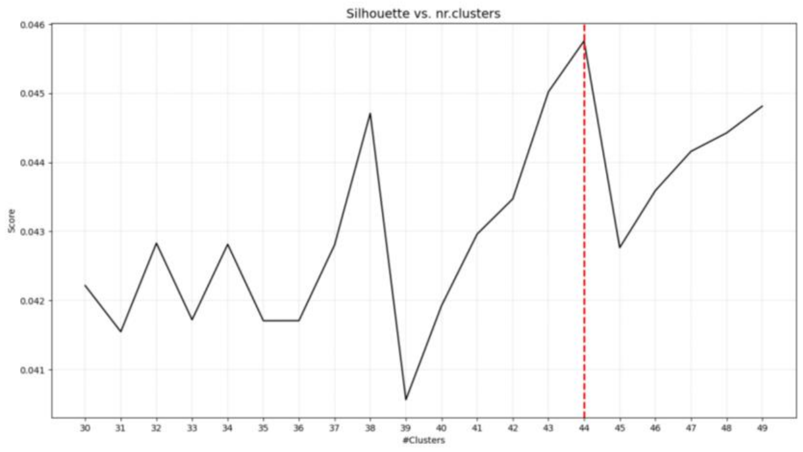
PCA example:

For a 64x64 image, 128 principal components explain the 95.05% of the variance.

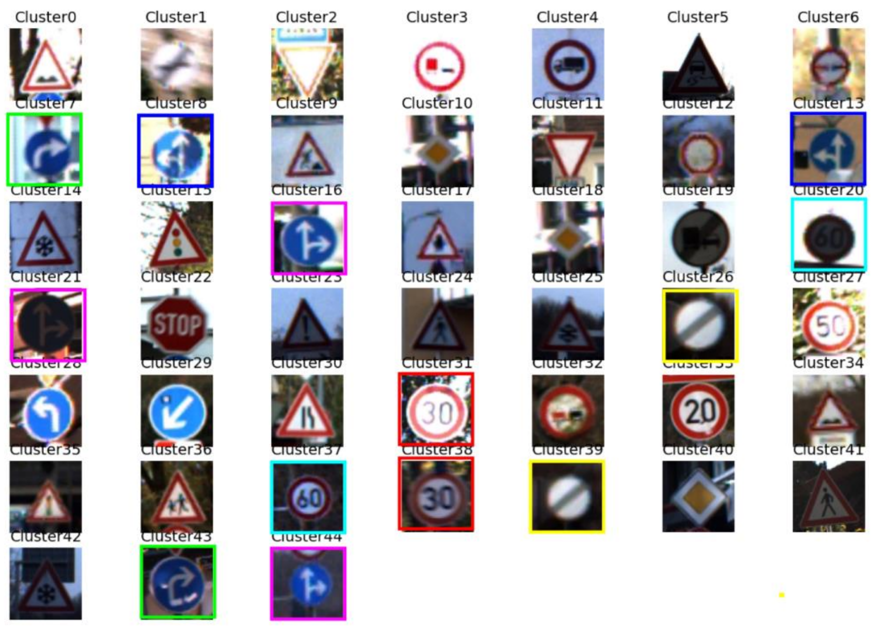


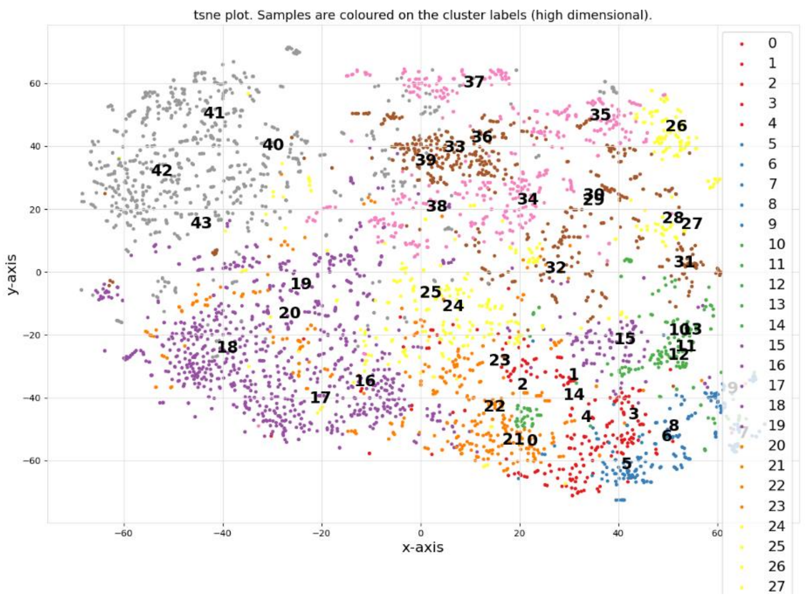
To test the algorithm, first only 100 images of each label are taken. As a result, it was obtained that, when the algorithm is left totally free to define the number of clusters, it tends to generate only 3 to 4 clusters. This may be due to the similarity within the dataset, since the main difference between labels is found only in the drawing inside the sign.

However, by forcing the generation of at least 30 clusters, 44 are recognized.



Among them, 6 are repeated. Between the repeated clusters it can be differentiated that it usually divides the darkest and the lightest versions of the label.





Pre-conclusion: Because classification through this unsupervised method cannot compete against supervised methods, which easily reach 96% yields (kaggle examples), this method may be better for estimating what the variation is within the data set. In the same way it can be useful to search for mislabeled data within a label.

Transer Learning, k-means and agglomerative

Link to code repository and details: https://github.com/JIAOJIAOMEI/ADND-J7

conclusion until now:

1. I tried kmeans with sklean library, but the accuray is low, accuracy = 18%. 200 pictures for each class are used.
2. then I tried to extract features by pretained model resnet50, it is better, but almost very little improvement, 18.3%.
3. I tried pretained model resnet50 + hdbscan, the result is very bad.
4. I tried kmeans with clustimage library, it is much better than kmeans with sklearn lrbrary. But it seems like there are no improvment if grey pictures are used.
5. I tried agglomerative with different linkages, like "single", "complete", "centroid" and "ward". It turns out "ward" is the best.
6. all in all, clustimage library can recognize basic shapes such as circles, triangles and rectangles, but more complex shapes cannot be accurately extracted from features, and it can recognize colours such as blue, red, dark blue and black, which are more distinctive, but more similar colours cannot be recognized.
7. **I think the key is feature extraction, as long as this step is done well, the accuracy rate will be much better later on. However, if you really want a high accuracy rate, pertained models are not enough and you need to train specifically for the gtsrb dataset.**

Invariant Information Clustering (Jérémie Capdeville)

The Invariant Information Clustering model is one of the best model at the moment for the Image Classification and Segmentation. This neural network learn from scratch, only with unlabelled data samples. It is very important because today, getting an huge amount of labelled datas cost a lot while unlabelled datas are massive. So we can say developping Usupervized models is the futur of Deep Learning. In our project, we have german traffic signs and we want to cluster them “automatically” so this kind of model is exactly what we should use.

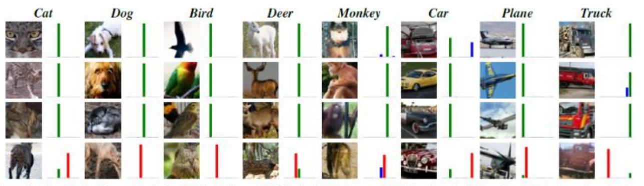


Figure 5: **Unsupervised image clustering (IIC) results on STL10.** Predicted cluster probabilities from the best performing head are shown as bars. Prediction corresponds to tallest, ground truth is green, incorrectly predicted classes are red, and all others are blue. The bottom row shows failure cases.

	STL10
Dosovitskiy 2015 [18]†	74.2
SWWAE 2015 [54]†	74.3
Dundar 2015 [19]	74.1
Cutout* 2017 [15]	87.3
Oyallon* 2017 [42]†	76.0
Oyallon* 2017 [42]	87.6
DeepCluster 2018 [7]	73.4⋆
ADC 2018 [24]	56.7⋆
DeepINFOMAX 2018 [27]	77.0
IIC plus finetune†	**79.2**
IIC plus finetune	**88.8**

Table 3: **Fully and semi-supervised classification.** Legend: *Fully supervised method. ⋆Our experiments with authors' code. †Multi-fold evaluation.

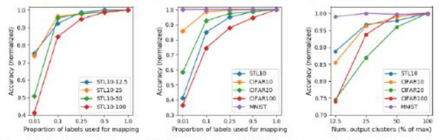


Figure 6: **Semi-supervised overclustering.** Training with IIC loss to overcluster ($k > k_{gt}$) and using labels for evaluation mapping only. Performance is robust even with 90%-75% of labels discarded (left and center). STL10-r denotes networks with output $k = \lceil 1.4r \rceil$. Overall accuracy improves with the number of output clusters k (right). For further details see supplementary material.

Lets talk about its accuracy. The IIC model perfroms very well. It was tested on STL10 , an unsupervised variant of ImageNet (51.6% +/- 0.844), CIFAR10 (61.7% +/- 5%), and also MNIST (99.2% +/- 0.652) which are benchmark references in this fild. The model beat the accuracy of their closest competitors by 6.6 and 9.5% respectively. Also, it was tested with semi-supervised learning and it has 88.8% accuracy on STL10 classification. Their developpers says that it's more than all existing methods (supervised, semi-supervised or unsupervised).

For the principle of the model, the trained network directly output semantics labels in order to maximise mutual information between the class. We pick one image and we apply a "perturbation" to it. After that we want the network to learn about the object perturbed and the original object → mutual information. This pertubation could be scaling, skewing, rotation or flipping (geometric), changing contrast and colour saturation (photometric), or any other perturbation that is likely to leave the content of th e image intact. For training the model use Adam optimiser with a learning rate of 0.001. The evaluation is based on accuracy (true positive / sample size). The model also avoid degenerate solutions by being rigorously grounded in the information theory.

The developpers also say that it's easy to implement but maybe not for me because I'm very new in this field but maybe i'll try if you consider it possible.

Conclusion : IIC have shown that it's possible to do unsupervised clustering with neural network. It shows a great level of accuracy and it's quite simple to understand as far as I have read. I will also explore other state-of-art model in this field to compare but i'm quite interested into this one for now. I've also understand that creating an unsupervised deep learning model can be done by creating paires of images artificialy with a transformation. So an interesting point could be trying different transformation to find the best. But i'll have to get some imagination there.

https://paperswithcode.com/paper/invariant-information-distillation-for github.com/xu-ji/IIC

CNN pretrained models + PCA + Kmeans (Jérémie Capdeville)

I've found a simplier way of doing unsupervised clustering. It's just a different use of the wellknown Kmeans Machine Learning model. The first idea would be using Kmean directly on the images but I assume we will get a poor result. To get better results I have found that we can use pretrained model from Tensorflow.Keras to transoform images. As theses models where trained on the ImageNet Dataset so I don't know if it's considered as an unsupervised learning or not afterall. But anyway if we can use it what we will be doing is encoding the images through a trained convolutional network, and then apply a clustering algorithm to the encoded features. We can then check the clusters and see if it worked.

As the convolutional network outputs multiples features, (for 244 x 244 x 3 images we get back 25k+ features as i've seen so far) we can apply a Principal Component Analysis to get rid of this and then apply the Kmeans without running out of memory. PCA also helps us getting the feature that are the most meaningful but I don't know if there is any possible interpretation as it's comming out of a Convolutional Neural Network.

Using this approach, it's possible to test multiple Pretrained CNN models without much work so maybe I'll also try this approach and compare it to the others. As I have seen, this approach result in an accuracy up to 97% on the a dataset of pets breed depending on the CNN model you are using, so it's not that bad. But i didn't found any metric for famous datasets like STL10, CIFAR10 or MNIST so I can't compare.

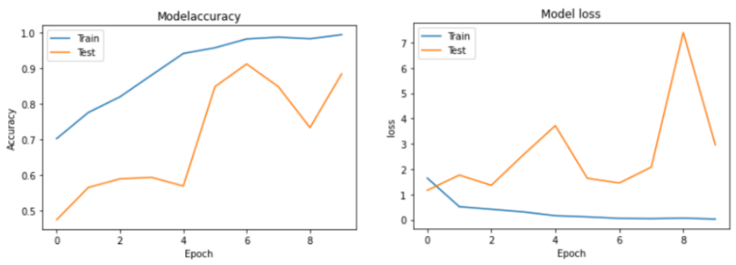
https://github.com/beleidy/unsupervised-image-clustering/blob/master/capstone.ipynb

Supervised Neural Network for performance testing (Jérémie Capdeville)

I also tried to get an idea of the performance and training times on these images using Supervised Learning models. I thought it was important to realize how much information a neural network needed to process and then do the same with an unsupervised version.

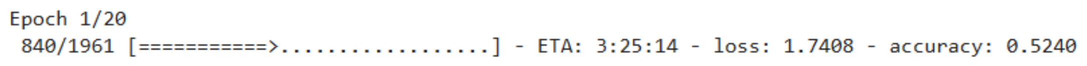
So I took a model made by someone on the internet and tried to train it on the traffic sign data of the project. This allowed me to familiarize myself with the data but also with how to import and train a model on images because it is something I had not done before.

Here are the results I obtained below.



Supervised models are quite efficient on this kind of data but it is extremely long to train which is a problem. Here I trained the neural network with only 5 classes out of the 43 available in our dataset. And the training time less than an hour for 10 epoch with the virtual machines provided for free by google colab which is ok I think. The first epoch is always very long.

I think it is more interesting to look at pre-trained models in order to encode our images and get more information out of them. Then it will be possible to apply clustering techniques such as Kmeans or Gausian Mixture model. The goal will be to determine which pre-trained models lead to better performances. I also tried with all the the categories and it was way to long for colab : (it stops after 3 hours of running time)



```
Epoch 1/20
 840/1961 [===========>..................] - ETA: 3:25:14 - loss: 1.7408 - accuracy: 0.5240
```

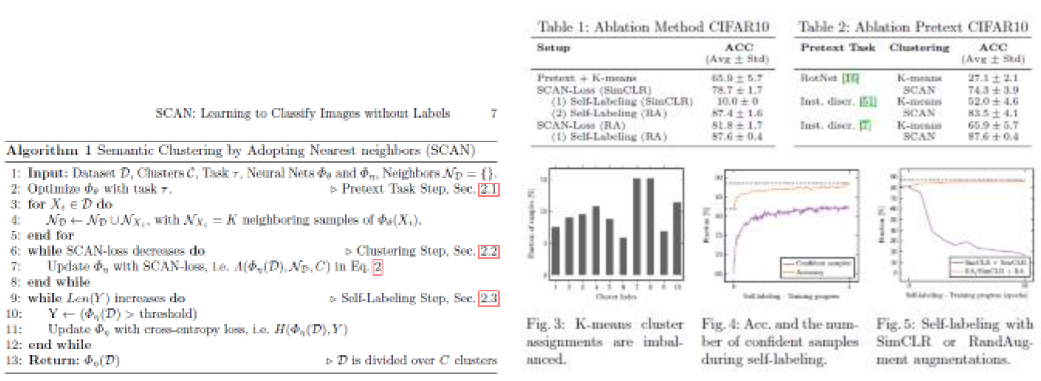
Maybe i'll try with Colab Pro because i'm not familiar at all with GPU from Azure or AWS. I can also try to use a simplier model because the one I used was quite sofisticated.

Semantic Clustering by Adopting Nearest neighbors (Jérémie Capdeville)

It's SSL + Clustering + Fine-tunning. First we want to transform de images to do self supervized learning. But here we have to "guess" the number of clusters. Then we can apply clustering to the images with Kmeans, or Nearest Neighbourg.

After that we can already get some good results but if we want to increase the accuracy we will do Fine Tunning through self labeling. So that means that we learn on our self labeled datas from the previous step. This new classifier will use the point (or images) that we are more certains about to learn a new classifier so I understand that it will result in a better accuracy.

The algorithm looks like this with the 3 steps. And what about the performances ? The metrics are quite good but here it's only with 10 clusters and it's not tested with ImageNet here.



SCAN: Learning to Classify Images without Labels 7

Algorithm 1 Semantic Clustering by Adopting Nearest neighbors (SCAN)

```
1: Input: Dataset $\mathcal{D}$, Clusters $\mathcal{C}$, Task $\tau$, Neural Nets $\Phi_\theta$ and $\Phi_\eta$, Neighbors $\mathcal{N}_\mathcal{D} = \{\}$.
2: Optimize $\Phi_\theta$ with task $\tau$.                       ▷ Pretext Task Step, Sec. 2.1
3: for $X_i \in \mathcal{D}$ do
4:     $\mathcal{N}_\mathcal{D} \leftarrow \mathcal{N}_\mathcal{D} \cup \mathcal{N}_{X_i}$, with $\mathcal{N}_{X_i} = K$ neighboring samples of $\Phi_\theta(X_i)$.
5: end for
6: while SCAN-loss decreases do                                  ▷ Clustering Step, Sec. 2.2
7:     Update $\Phi_\eta$ with SCAN-loss, i.e. $\Lambda(\Phi_\eta(\mathcal{D}), \mathcal{N}_\mathcal{D}, C)$ in Eq. 2
8: end while
9: while $Len(Y)$ increases do                                   ▷ Self-Labeling Step, Sec. 2.3
10:    $Y \leftarrow (\Phi_\eta(\mathcal{D}) > \text{threshold})$
11:    Update $\Phi_\eta$ with cross-entropy loss, i.e. $H(\Phi_\eta(\mathcal{D}), Y)$
12: end while
13: Return: $\Phi_\eta(\mathcal{D})$                            ▷ $\mathcal{D}$ is divided over $C$ clusters
```

Table 1: Ablation Method CIFAR10

Setup	ACC (Avg ± Std)
Pretext + K-means	65.9 ± 5.7
SCAN-Loss (SimCLR)	78.7 ± 1.7
(1) Self-Labeling (SimCLR)	10.0 ± 0
(2) Self-Labeling (RA)	87.4 ± 1.6
SCAN-Loss (RA)	81.8 ± 1.7
(1) Self-Labeling (RA)	87.6 ± 0.4

Table 2: Ablation Pretext CIFAR10

Pretext Task	Clustering	ACC (Avg ± Std)
RotNet [16]	K-means	27.1 ± 2.1
	SCAN	74.3 ± 3.9
Inst. discr. [51]	K-means	52.0 ± 4.6
	SCAN	83.5 ± 4.1
Inst. discr. [7]	K-means	65.9 ± 5.7
	SCAN	87.6 ± 0.4

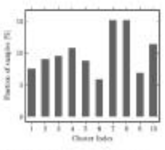


Fig. 3: K-means cluster assignments are imbalanced.

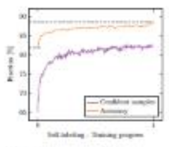


Fig. 4: Acc. and the number of confident samples during self-labeling.

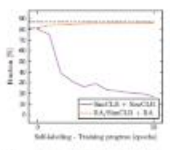


Fig. 5: Self-labeling with SimCLR or RandAugment augmentations.

For Imagenet we can see that the performances really depend on the number of classes. So maybe with 43 classes that we have it will be okay.

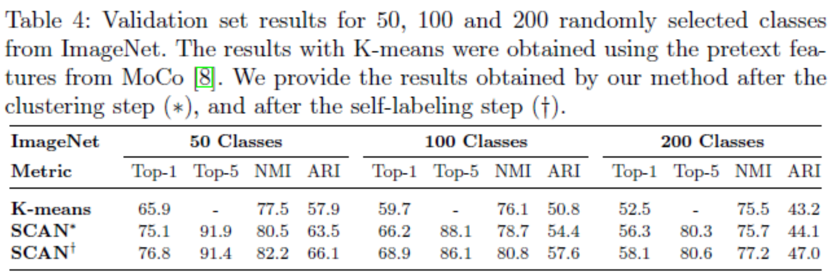


Table 4: Validation set results for 50, 100 and 200 randomly selected classes from ImageNet. The results with K-means were obtained using the pretext features from MoCo [8]. We provide the results obtained by our method after the clustering step (∗), and after the self-labeling step (†).

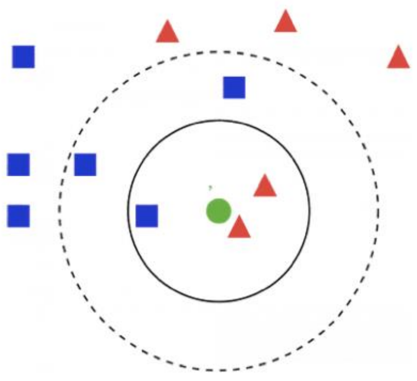
ImageNet	50 Classes				100 Classes				200 Classes			
Metric	Top-1	Top-5	NMI	ARI	Top-1	Top-5	NMI	ARI	Top-1	Top-5	NMI	ARI
K-means	65.9	-	77.5	57.9	59.7	-	76.1	50.8	52.5	-	75.5	43.2
SCAN*	75.1	91.9	80.5	63.5	66.2	88.1	78.7	54.4	56.3	80.3	75.7	44.1
SCAN†	76.8	91.4	82.2	66.1	68.9	86.1	80.8	57.6	58.1	80.6	77.2	47.0

https://arxiv.org/abs/2005.12320 https://github.com/wvangansbeke/Unsupervised-Classification
https://www.youtube.com/watch?v=hQEnzdLkPj4

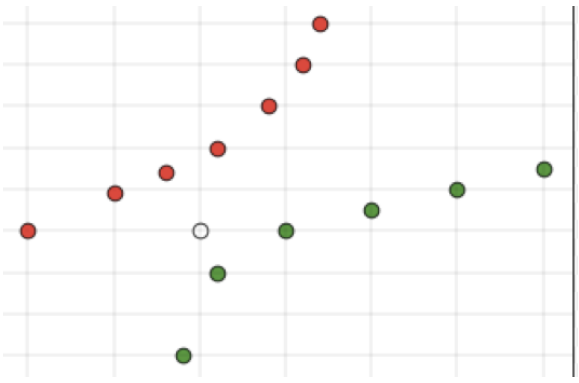
K-Nearest-Neighbors :

This is a **supervised** way to classify our data. The goal is pretty simple : when we visualise our data class by class (for example here, the blue and the red one), what will happen if we try to guess the class of a whole new data point ? (Here, the green point). What will his class be ? A very "simple" way to solve this problem is to use the K-NN classification.

If we try to take a look on our data, it would be logic to say that our green point is in the red class. Why ? Because it's very close to red data points, simple as that. This is what is based on the K-NN algorithm. It counts the K closest points of our green point (based on the dimension's norm), and try to guess the class by determining which class has the most points close to it.



What I tried was a little bit different. Actually, it's good to count the closest neighbors, but it's not always working. Look at the following graph : according to a KNN algorithm, it would output the red class. But, on the plot, our point is closer to the g reen class than the red one. It's just that there is one red point more. To avoid this kind of "miss learning", we can weight our classification, using the distance between each point. Then, the class with the best score (sum of the inverse of the distanc es) is the output of our algorithm.



I wrote the K-NN classification function by myself, but of course sklearn models are also very performing. This is what I tried to do. Firstly, I only used a little part of data, because it would be too long to run on 45.000 train images and 12.000 test images.

- With 2000 "random" images, the accuracy is quite poor : around 20%
- With a selection of classes and only 2000 train images and 700 test images, the accuracy is better , 40%, but still "bad".
- With all the data, it's still running

NB : This a supervised model, what means that we already know the data characteristics before guessing the class. But it was interesting to compare unsupervised and supervised ways to do.