Project J7 : Unsupervied Deep Image Classification

The goal of this project is to research and experiment with Unsupervised Classification models on the German traffic sign dataset.
https://www.kaggle.com/datasets/jerems69/german-signs

For my part, I was interested in several models. First, I wanted to see how well a supervised model performed in order to have a performance benchmark.
Then I did some research and tried a model using pre-trained neural networks to extract features from images. I then applied a Kmeans algorithm on these features with or without Principal Component Analysis to predict in an unsupervised way.
In the third part I could test a more advanced model which is the Invariant Information Clustering. I first tested it on the MNIST dataset which is the one on which it performs the best. I then wanted to train it on our dataset but I ran into a non-learning problem that I could not solve.
Finally I did some research on other advanced models with Semantic Clustering. The goal is to transform the images to do Self Supervized as in the previous model and then apply a Nearest Neighbors algorithm. I did not implement it because Sacha was already working on a similar subject.

Supervised Neural Network

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.

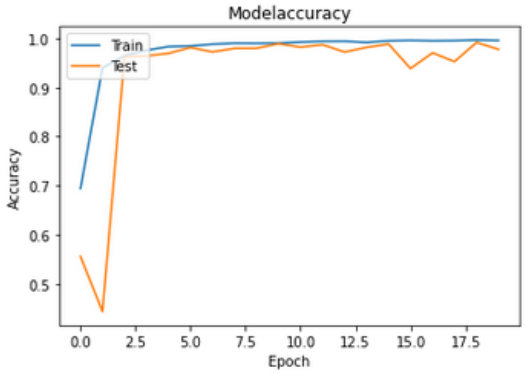
So I created a somewhat arbitrary CNN model and tried to train it on the traffic sign data of the project. This allowed me to familiarize myself with the data but also with how to import and train a model on images because it is something I had not done before.

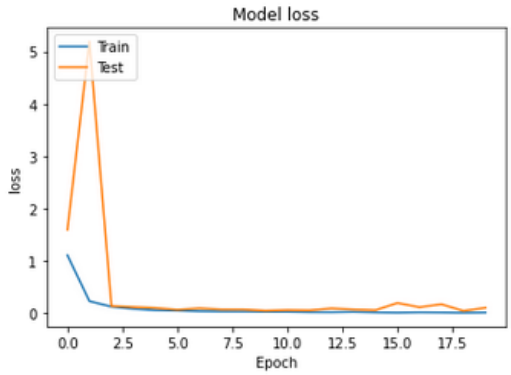
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 16, 16, 32)        0
_________________________________________________________________
batch_normalization_2 (Batch (None, 16, 16, 32)        128
_________________________________________________________________
dropout_2 (Dropout)          (None, 16, 16, 32)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_3 (Dense)              (None, 1000)              8193000
_________________________________________________________________
dense_4 (Dense)              (None, 43)                43043
=================================================================
Total params: 8,237,067
Trainable params: 8,237,003
Non-trainable params: 64
_________________________________________________________________
```

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 the 43 available in our dataset. And the training time was very fast using Kaggle GPU for 20 epoch.

I had previously tested this same model on Google Clolab GPUs but it was much too long. The model took several hours to perform the first epoch. So I chose to use Kaggle and its very powerful free GPU afterwards.
We can see that for a basic supervised CNN model, the problem is not a problem. We reach very quickly a more than honorable accuracy.
The challenge lies in the unsupervised nature of the problem.





CNN pretrained models + PCA + Kmeans

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.

I think it is 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.

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

So we now want to apply it to our dataset of german sign that is quite more complexe than the pets breed. We will use theses 3 pre-trained models :

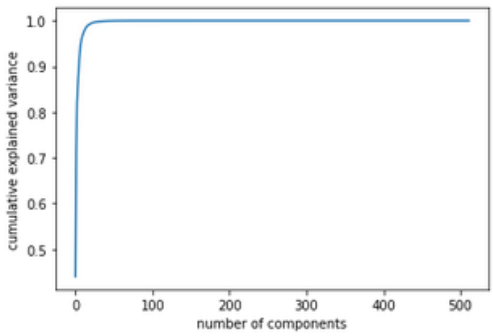
```
VGG16 flattened output has 512 features
VGG19 flattened output has 512 features
ResNet50 flattened output has 2048 features
```

The above cell shows us the number of features each covnet gives to a single image. When we compare these to the original size of the image 32 x 32 x 3 = 3072 pixels/features, we can see that this is a large reduction in what the clustering algorithms will have to work with.

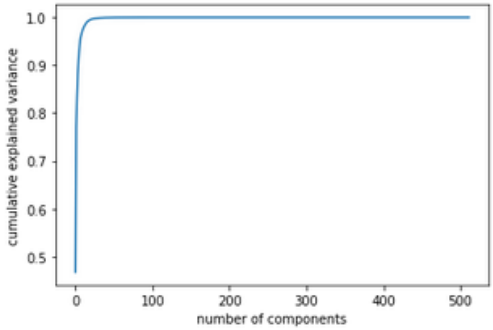
Hopefully these reduces number of feature are represent more meaningful features in the image structure.

After that we will try both PCA and not PCA to see if the results are better with ou without.

We can see on the right the explained variance as a function of the number of components. We can see here a fast convergence so with a small number of components we can explain almost all the variance.



We will then apply our Kmeans algorithm on these features to perform our clustering. We can see below that the algorithm converges much faster on data that are not transformed by the PCA.

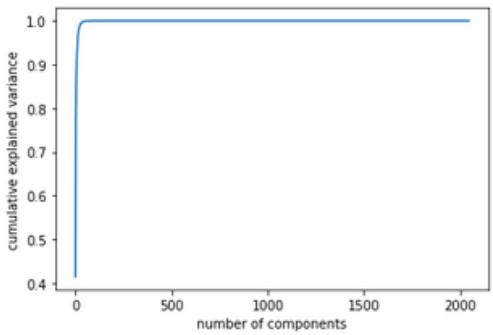



```
KMeans (PCA):

VGG16
Training took 182.82956957817078 seconds

VGG19
Training took 108.3705050945282 seconds

ResNet50
Training took 885.5556704998016 seconds
```



```
KMeans:

VGG16:
Training took 95.48613357543945 seconds

VGG19:
Training took 83.96388053894043 seconds

ResNet50:
Training took 254.85358929634094 seconds
```

The clustering algorith does not detect which images are which in term how labels, it only groups images that look alike together and assigns them a number arbitrarily.

We now need to count how many of each label are in each cluster, this way we can take a look and if sufficient eperation has happened we can quicly see which cluster is which label. So we write a function that does that.

After that we just have to find the clusters and calculate the accuracy of our model. We can see that our accuracy is not good even on training data so we will move to a more advanced model.

```
KMeans VGG16:
0.15069497577148686
KMeans VGG16 (PCA):
0.15299030859474624

 KMeans VGG19:
0.14527543993879113
KMeans VGG16:
0.1483040040805917

 KMeans Resnet50:
0.07593726090283091
Kmeans Resnet50 (PCA):
0.16606095383830655
```

Invariant Information Clustering

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.

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 the 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.

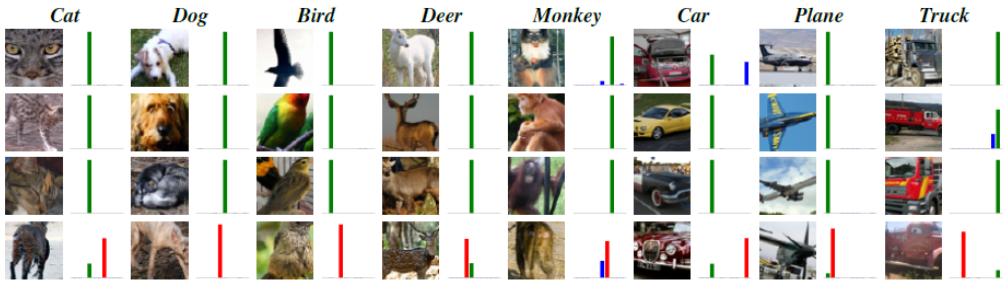


Figure 5: **Unsupervised image clustering (IIC) results on STL10.** Predicted cluster probabilities from the best performing head are shown as bars. Prediction corresponds to tallest, ground truth is green, incorrectly predicted classes are red, and all others are blue. The bottom row shows failure cases.

	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.

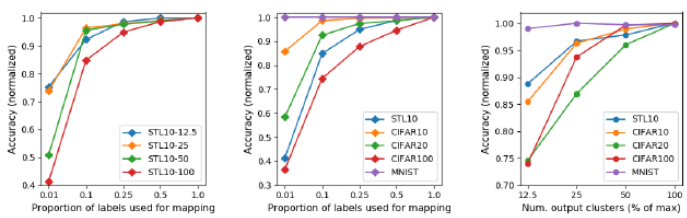


Figure 6: **Semi-supervised overclustering.** Training with IIC loss to overcluster ($k > k_{gt}$) and using labels for evaluation mapping only. Performance is robust even with 90%-75% of labels discarded (left and center). STL10-r denotes networks with output $k = \lceil 1.4r \rceil$. Overall accuracy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

```
def IIC(z, zt, C=10):
  P = (z.unsqueeze(2) * zt.unsqueeze(1)).sum(dim=0)
  P = ((P + P.t()) / 2) / P.sum()
  P[(P < EPS).data] = EPS
  Pi = P.sum(dim=1).view(C, 1).expand(C, C)
  Pj = P.sum(dim=0).view(1, C).expand(C, C)
  return (P * (log(Pi) + log(Pj) - log(P))).sum()
```

Figure 4: IIC objective in PyTorch. Inputs `z` and `zt` are $n \times C$ matrices, with C predicted cluster probabilities for n sampled pairs (i.e. CNN softmaxed predictions). For example, the prediction for each image in a dataset and its transformed version (e.g. using standard data augmentation).

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://github.com/astirn/IIC/blob/master/models_iic.py

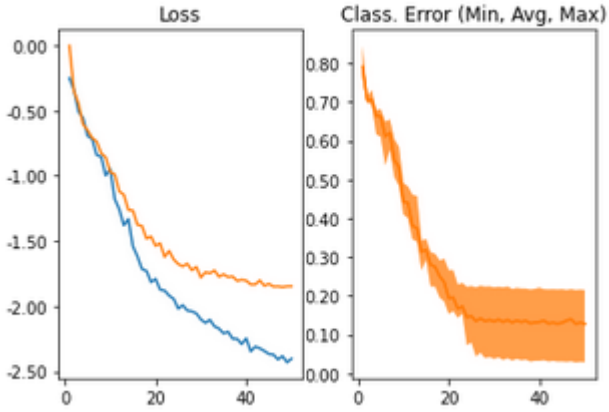
> https://paperswithcode.com/paper/invariant-information-distillation-for

> github.com/xu-ji/IIC

Implementation :

Now that we understand how the IIC model works, we will try to apply it.
First we will implement it on the MNIST dataset for which it is very efficient. MNIST has only 10 classes and this makes it much simpler than our dataset.

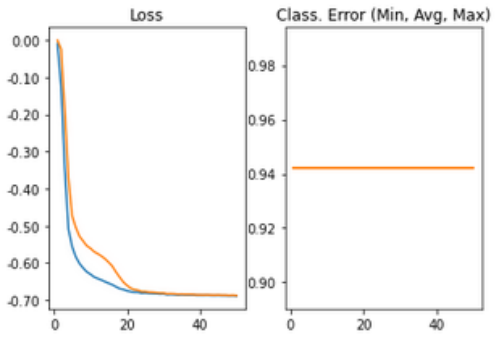
After adapting and correcting the compatibility issues between Tensorflow 1 and 2 we were finally able to run the IIC model on MNIST and we got this performance which is quite good.



Now we have to adapt it to our dataset. This poses some problems because MNIST is a tensorflow dataset with information about this dataset. After making these changes, the model worked but it doesn't learn at all and I haven't found any solutions to date to fix this issue. My hypothesis is that the number of classes is more important than for MNIST (42 against 10) and that it creates problems.
There was a warning that could suggest a data size problem. I tried to downscale the dataset without success.
Maybe some parameters were not well modified to fit our dataset.
As the implementation of this code is not the official one but a tensorflow version, it is possible that it has some errors.
I haven't found a solution to this problem yet.

```
2022-12-09 16:09:47.355680: I tensorflow/core/common_runtime/process_util.cc:146] Creating
new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads f
or best performance.
2022-12-09 16:09:51.186829: I tensorflow/core/common_runtime/process_util.cc:146] Creating
new thread pool with default inter op setting: 2. Tune using inter_op_parallelism_threads f
or best performance.
2022-12-09 16:09:52.397223: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:185]
None of the MLIR Optimization Passes are enabled (registered 2)


Epoch 1, Loss = -0.0003
Evaluating classification accuracy...
Classification error = 0.9350
Classification error = 0.9350
Classification error = 0.9350
Classification error = 0.9350
Classification error = 0.9350
Done
```



Semantic Clustering by Adopting Nearest neighbors

Finally, I explored another model, or rather another approach that combines self supervised learning and Nearest Neighbors.

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.

Algorithm 1 Semantic Clustering by Adopting Nearest neighbors (SCAN)

Input: Dataset $\mathcal{D}$, Clusters $\mathcal{C}$, Task τ, Neural Nets Φ_θ and Φ_η, Neighbors $\mathcal{N}_\mathcal{D} = \{\}$.
Optimize Φ_θ with task τ. ▷ Pretext Task Step, Sec. 2.1
for $X_i \in \mathcal{D}$ **do**
  $\mathcal{N}_\mathcal{D} \leftarrow \mathcal{N}_\mathcal{D} \cup \mathcal{N}_{X_i}$, with $\mathcal{N}_{X_i} = K$ neighboring samples of $\Phi_\theta(X_i)$.
end for
while SCAN-loss decreases **do** ▷ Clustering Step, Sec. 2.2
  Update Φ_η with SCAN-loss, i.e. $\Lambda(\Phi_\eta(\mathcal{D}), \mathcal{N}_\mathcal{D}, C)$ in Eq. 2
end while
while $Len(Y)$ increases **do** ▷ Self-Labeling Step, Sec. 2.3
  $Y \leftarrow (\Phi_\eta(\mathcal{D}) >$ threshold)
  Update Φ_η with cross-entropy loss, i.e. $H(\Phi_\eta(\mathcal{D}), Y)$
end while
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. [61]	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

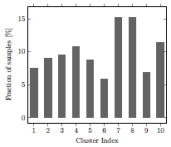


Fig. 3: K-means cluster assignments are imbalanced.

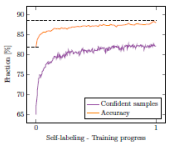


Fig. 4: Acc. and the number of confident samples during self-labeling.

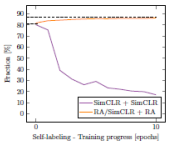


Fig. 5: Self-labeling with SimCLR or RandAugment augmentations.

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.

For ImageNet we can see that the performances really depends on the number of classes.

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