Ensemble Clustering in Reduced Latent Space

Deep Learning Project

Mathieu Pont and Lucas Rodrigues Pereira

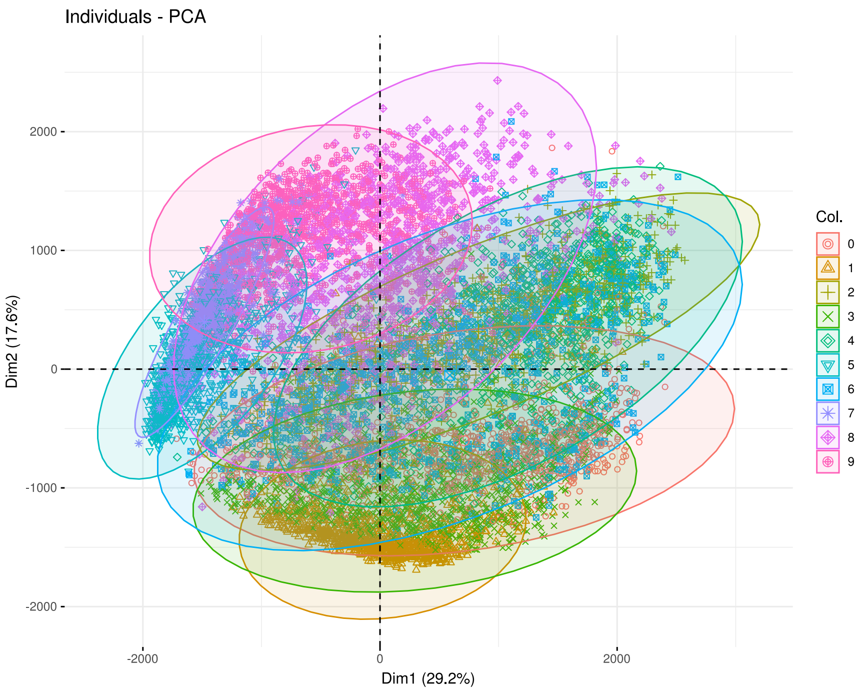
January 2020

Abstract

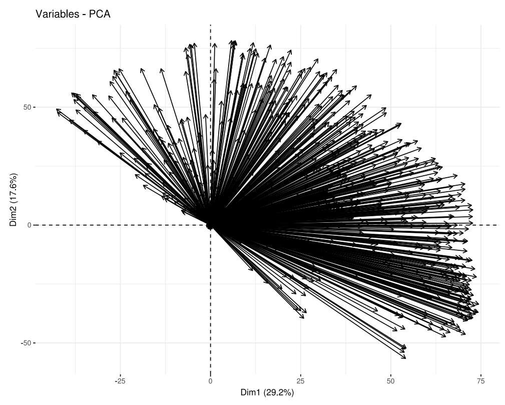
1 Introduction

In the context of the Master's Degree on Machine Learning for Data Science, at the University of Paris (Descartes), we have been given the task to discuss clustering in reduced latent space of many autoencoder, as part of the "Deep Learning" course. The famous Fashion-MNIST data set (representing clothes images) will be used in our experiments.

2 Dimensionality reduction with Principal Component Analysis (PCA)



(a) Individuals factor map.



(b) Variables factor map.

Figure 1: Factor maps of PCA.

The first two components explain 46.8% of the initial variance. We have decided to use a non-normed PCA since the variables have identical interval of values, with a normed PCA the variance explained was only 36.49%. The classes are not well separated when projected to the first factor map.

The variables factor map of Figure 1b is quite complicated to analyze because of the initial number of dimensions ($28 \times 28 = 784$). We can however see that no dimension are negatively correlated with both components at the same time, they tends to point in the same direction meaning that they bring the same kind of information.

Nevertheless, some dimensions are negatively correlated with each other, pixels associated to these dimensions are therefore never colored at the same time. It can be explained by the fact that the classes can be divided in two groups, those that their picture are more expanded from left to right (like the different shoes and bags), and the opposite group having pictures more expanded from top to bottom (like trousers and the different top clothes). Implying that pixels at the top and bottom and those at the left and right are rarely colored at the same time.

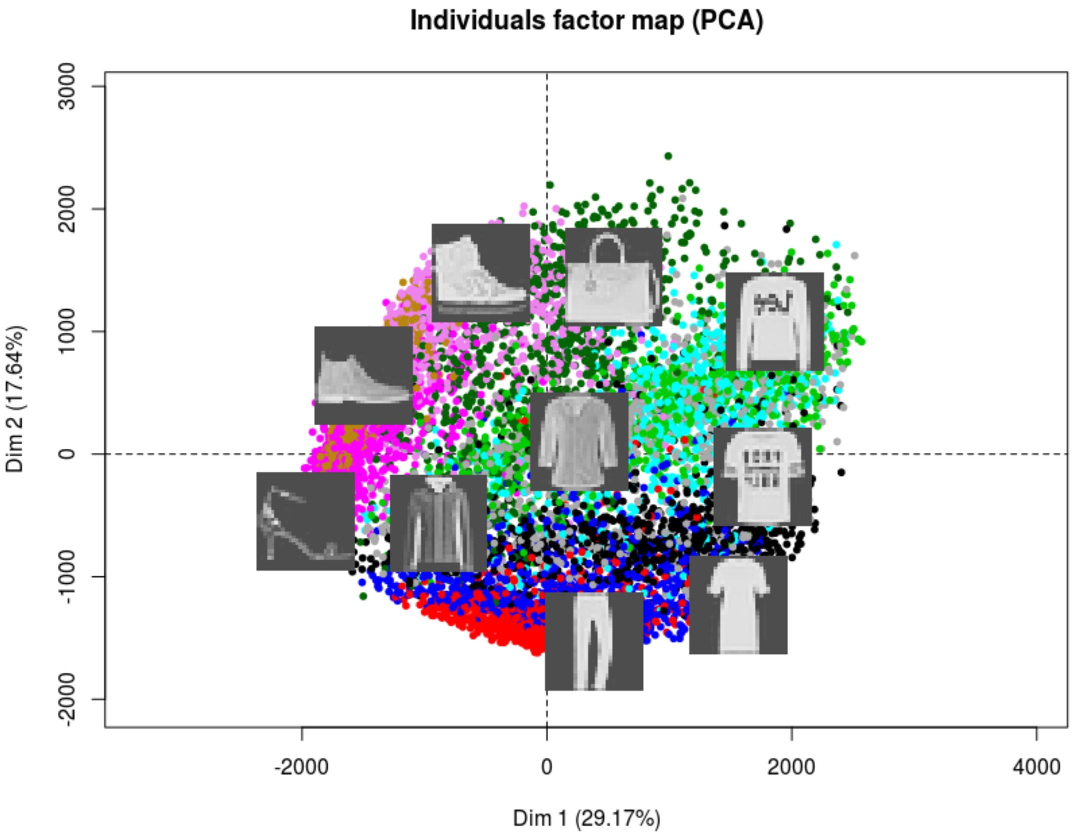


Figure 2: Individuals factor map of PCA with one representative for each class.

Figure 2 shows us that, when projected in the principal components, classes are well grouped together according to their kind of clothes. For example, we can see at the left that the 3 classes corresponding to shoes are close to each other, at the right we have top clothes like t-shirt, pullover etc. Finally, at the top and bottom we have the marginal classes corresponding to bags and trousers respectively.

We can relate with the variables factor map of Figure 1b and see that some pixels are more implicated with top clothes than shoes for example. Indeed, most of the time the top right corner of the picture of shoes is empty contrary to tops.

Reconstructed images are available in Appendix C. We choose 23 components in order to explain 80% of the variance. We see that some reconstructions are badly impacted by the trouser class, for example the reconstructed image of the dress and the bag look a lot like a trouser.

3 Dimensionality reduction with t-SNE

In this part of the study, we will experiment t-SNE to reduce the dimensionality of the Fasihon-MNIST. As described on the Scikit-learn site[1], this method converts similarities between data points to joint probabilities and tries to minimize its difference between the original vector space and the reduced latent space. Two other characteristics are worth mentioning, as they are required to interpret the results of this experiment:

- t-SNE has a cost function that is not convex, which means that different initializations can get different results.
- It is also recommended to use it with up to 50 features, to suppress noise and speed up computation of pairwise distances. For that purpose, we will later on this research use it with autoencoded data.

The result of the experiment is plotted below. Reducing 784 dimensions to only 2 seemed to have worked relatively well. Further on this study we will follow the recommendation and reduce dimensionality to a lower level before implementing t-SNE. In this case, we will use autoencoder.

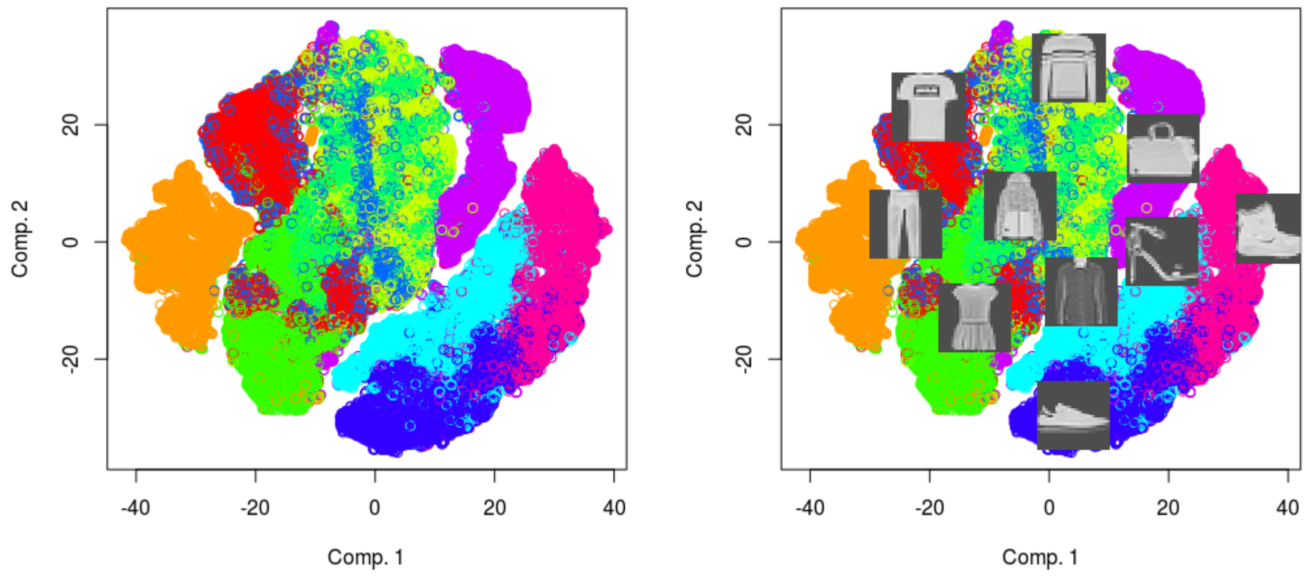


Figure 3: t-SNE on original images with default parameters with $KL = 3.08$.

Like for PCA the different classes are well grouped together according to their kind. Shoes at the right bottom, top in the middle, and marginal classes at the right and left for respectively bags and trousers.

We have tested two hyper-parameters: perplexity and learning rate. The former is related to the number of nearest neighbors. We have tried minimum and maximum values for each parameter and plotted it to analyse their impact on the output. We have chosen each time between the minimum and the maximum value for the specific parameter, keeping for all others the default.

[1]https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html

A small learning rate caused the model to either stop at a local minimum or reaching the maximum number of iterations before reaching the global minimum. As regards the perplexity, it appears that the greater the value, the more separated the clusters. The opposite is also true, the smaller the perplexity the more similar to a sphere (for two dimensions) the result should be.

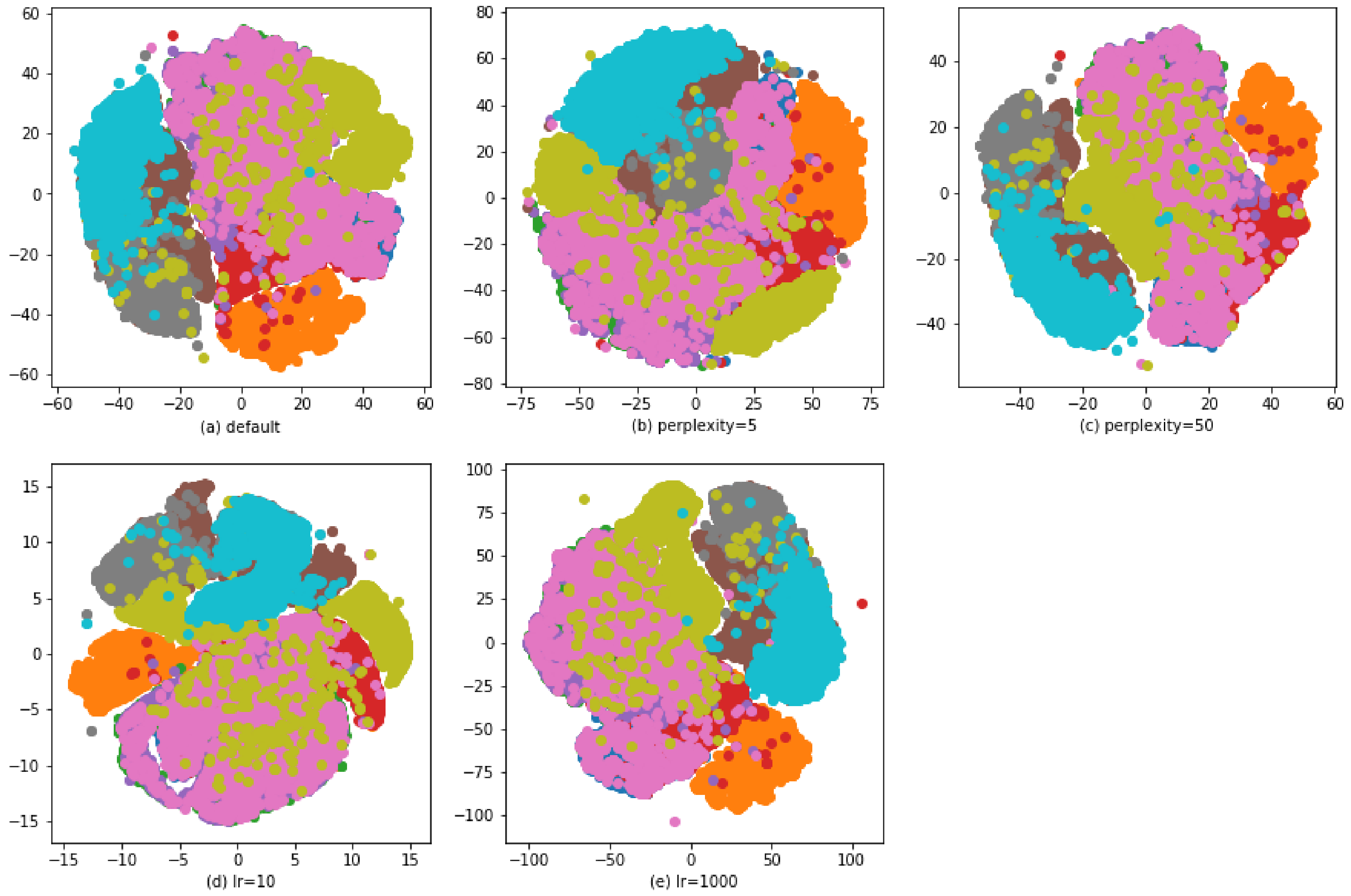


Figure 4: t-SNE on original images with different parameters.

4 Dimensionality reduction with Autoencoder

In order to continue in our analysis through dimensionality reduction, we can use an autoencoder. Several variants of this algorithm exist such as Convolutional Autoencoder (where the layers are no more dense but convolutional), Variationnal Autoencoder (where a gaussian distribution is used instead of the "code") etc.

We take many different models to evaluate their dimensionality reduction performance through a clustering in the reduced latent space provided by each of them. The models selected were DAE (Deep AutoEncoder) and DCAE (Deep Convolutional AutoEncoder).

Since we have many models we can also use the encoded representation of all of them to do an ensemble clustering.

For the deep autoencoder we have chosen a basic architecture whose hyper-parameters are available in

Appendix A. For the deep convolutional autoencoder, at first we have selected an existing and common architecture for this kind of task. Then, guided by the reconstruction loss error and the clustering metrics in the reduced latent space, we iteratively modify the model to improve it. Hyper-parameters are available in Appendix B.1. We have decided to choose an encoding dimension of 49 because it can be represented as an image of 7×7 and therefore make easier the use of convolutional layer.

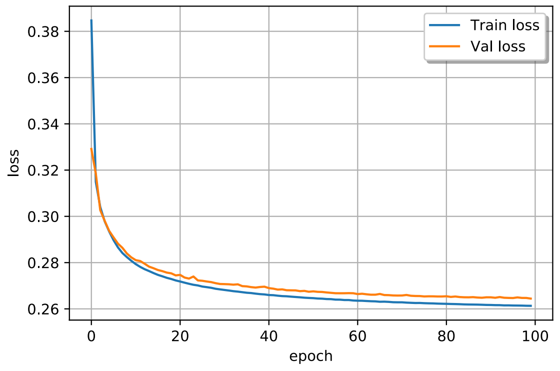


Figure 5: Evolution of the train and validation (test) loss.

The loss behaves correctly with no sign of overfitting or underfitting.

Reconstructed images with the autoencoder are available in Appendix D. We see that the reconstructed images are much more accurate than those of PCA, so much so that we can't tell the difference between real and reconstructed images for some of them. However we see that the autoencoder struggle for some details, for example, for the last image it did not manages to reconstruct the logo of the shirt correctly.

In order to visualize the reduced latent space we have applied a t-SNE on encoded images.

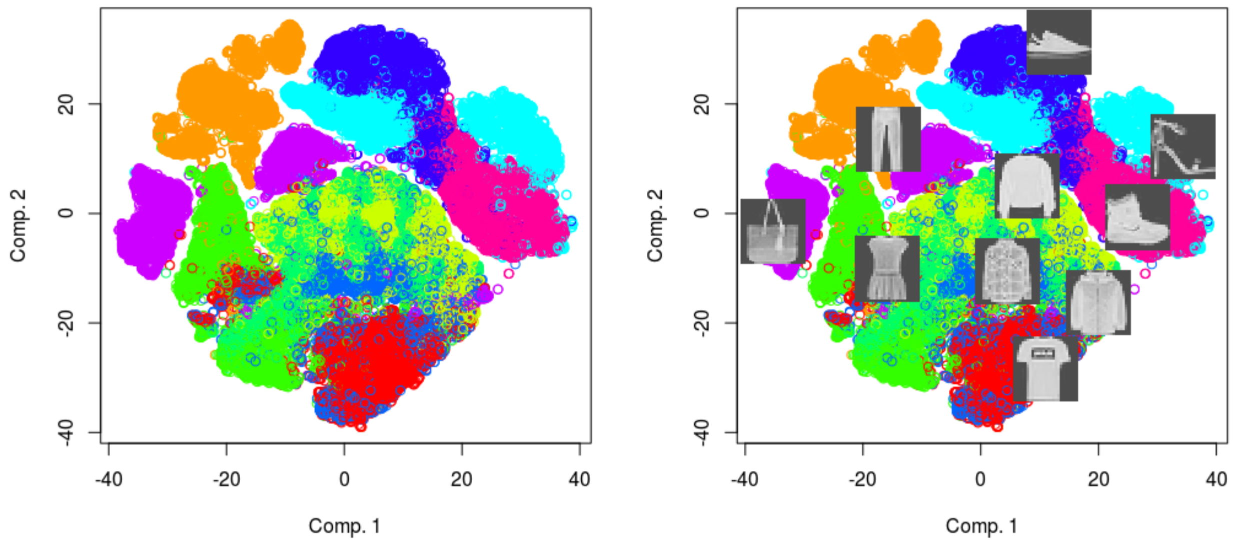


Figure 6: t-SNE on encoded images with default parameters, $KL = 3.14$.

The results of t-SNE on encoded images and on original ones are more or less the same, the KL divergence is almost the same for each. Moreover the same comment that was done can be done here, classes are well regrouped by their kind, the only difference with previous plot is that, here, bags and trousers are side by side whereas they were at the opposite.

Since components of t-SNE does not have meaning like for PCA we can't really explain this rapprochement. However, since the results look a lot like each other we can say that the encoding of the autoencoder has preserved the important information about data, this will be confirmed with the clustering metrics here after (in 4.1).

4.1 Clustering and ensemble/consensus in reduced latent space

In order to compare architecture and be able to make an ensemble clustering we have trained many models. DAE has juste one hidden layer (with 256 neurons) and an encoding dimension of 49, DAE 2 is deeper whose architecture is in Appendix A, DAE 3 is the same but with an encoding dimension of 100. DCAE has convolutional layers and is described in Apppendix B.1. DCAE 2 is the same architecture but where the convolutional layers are duplicated one time (see Appendix B.2).

In order to use all encoding we have used some ensemble and consensus methods.

The idea of consensus is to apply many different clustering methods and combine the results to achieve better classification. It involves two steps: Generation and Consensus. First, partitions are generated using multiple approaches; then, these labels compose a new matrix in which a new method is applied (directly or indirectly on this matrix) to form a consensus over the different individual results. Many methods exist and a review can be found in [Vega-Pons and Ruiz-Shulcloper, 2011]. Some of them will be briefly explained here.

We have used different methods:

- **Ensemble** is the result of k-Means on the concatenation of each encoding (each sample have therefore a dimension of $49 + 49 + 100 + 49 + 49 = 296$).
- **Consensus MMM** (for Multinomial Mixture Model) is a consensus method. In the generation step we generate partitions of each encoding using k-Means, all these partitions can be concatenate in a new categorical data set, then, in the consensus step we use a Multinomial Mixture Model to make a clustering on this new categorical data set of partitions.
- **Consensus k-Modes** is the same as *Consensus MMM* but using k-Modes for the consensus step.
- **Consensus co-assoc.** uses the co-association matrix to do the consensus. The latter is a $n \times n$ square matrix (with n the number of samples) that indicates how many times a sample appears in same cluster than another sample among the different partitions. Then, we can use a clustering algorithm on this matrix to make the consensus. Here, the same method (k-Means) is used to generate the partitions and do the clustering on the co-association matrix.
- **Consensus hypergraph** is a method proposed by [Strehl and Ghosh, 2003] consisting on finding a new partition that share the higher NMI with all the proposed partitions. The problem is defined as a (hyper)graph partitioning problem.

The results are show in Table 1 and are the mean and standard deviation over 20 experiments. For the consensus methods we have generated 5 partitions using k-Means for each encoding leading to a total number of 25 partitions.

For *Consensus co-assoc.* we have only used 1/3 of the full data set because the co-association matrix is a big square matrix with dimension the number of samples (60000×60000 if we take the full data set that is too large to be stored at once even in sparse format). The samples were selected randomly for each experiment with the constraint of having a balanced final data set.

	NMI	**ARI**
Original	0.51	0.35
DAE	0.59 ± 0.01	0.41 ± 4e-3
DAE 2	0.60 ± 0.01	0.41 ± 3e-3
DAE 3	0.60 ± 0.02	0.43 ± 0.03
DCAE	0.59 ± 2e-3	0.40 ± 3e-3
DCAE 2	0.60 ± 0.01	0.42 ± 0.01
Ensemble	0.60	0.42
Consensus MMM	0.58 ± 0.02	0.42 ± 0.02
Consensus k-Modes	0.56 ± 0.03	0.41 ± 0.03
Consensus co-assoc.	**0.61 ± 3e-3**	0.43 ± 4e-3
Consensus hypergraph	**0.61**	**0.46**

Table 1: Metrics of k-Means on different representations.

Interestingly, the results of clustering on original data are lower than those on the different encoded data, meaning that the encoding make the clusters more appropriate to classify, at least for k-Means.

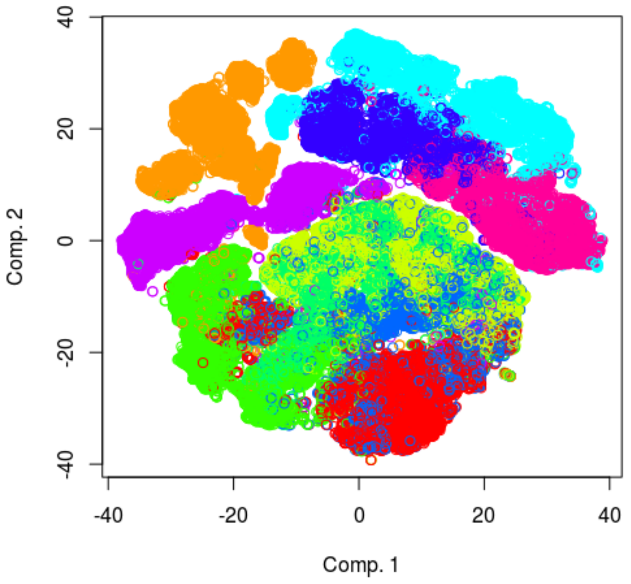
Among the different neural network architectures, that is the DCAE that has the lowest metrics, it can be explained by the fact that this is the architecture with the smallest number of trainable parameters. The one having the highest is DAE 3 whose have the best metrics for the clustering in the reduced latent space. DCAE 2 did very well, very close of DAE 3, but the latter has almost twice the number of trainable parameters. It shows us the importance of convolutional layer, architecture with this kind of layers can give very good results while reducing the number of trainable parameters. Among all these architectures the difference in results is minimal even though it seems that, here, bigger architecture get better results. It makes sense since we add more expression power to the neural network when we add trainable parameters.

However, this is some of the ensemble and consensus methods that give the best results. The concatenation of each encoding (*Ensemble*) seems to not bring much to the clustering and has a slightly lower ARI than the best autoencoder (DAE 3) and a same NMI. It can be explained by the fact that encoding related to good metrics (like those of DAE 3 and DCAE 2) are bring down by encoding related to models having not performed well (like DAE or DCAE).

Consensus MMM and its equivalent with k-Modes seem to not work well in our case. Like it could be expected, the variant with k-Modes has lower results than with MMM since the latter can be seen as a generalization of the former.

Consensus co-assoc has a slightly better NMI than the best autoencoder but the difference in results is only really seen with *consensus hypergraph* with a increase of ARI.

Finally we have decided to run t-SNE on the *Ensemble* encoded images (concatenation of each encoding) and the results is quite similar to t-SNE on original images and encoded images. The KL divergence is the lower for this representation. To compare these three execution of t-SNE we have decided to keep the default hyper-parameters in order to not tune them because it could promote one execution if the other are not tuned correctly.



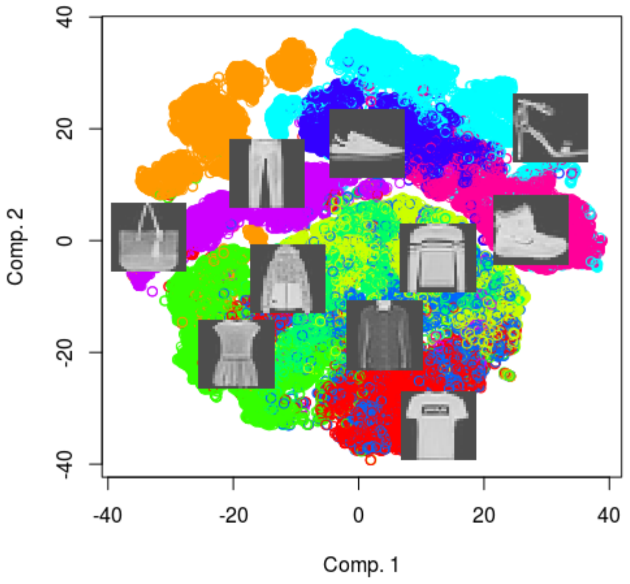


Figure 7: t-SNE on ensemble encoded images with default parameters, $KL = 3.05$.

5 Conclusion

We see through this work that autoencoder can provide meaningful encoded representation to work on. It can reduce the dimension and even manages to improve the clustering metrics compared to original data. Due to the big infatuation of Deep Learning, autoencoder was and will be improved for a long time allowing us to have better representations.

Moreover, we saw that an ensemble/consensus method on different partitions can provide better results that each of them separately.

One future work that could be interesting would be to construct many small neural network architectures, to train them and combine their prediction in a ensemble/consensus way. This could reduce a lot the number of trainable parameters and therefore the training time, and perhaps give a good way to make prediction. This future work is inspired by ExtraTrees [Geurts et al., 2006] where many small decision trees are made more or less randomly (according a hyper-parameter) and their prediction combined to make a better model.

Finally, our github repository is open source and free to use[2].

References

[Geurts et al., 2006] Geurts, P., Ernst, D., and Wehenkel, L. (2006). Extremely randomized trees. *Mach. Learn.*, 63(1):3–42.

[Strehl and Ghosh, 2003] Strehl, A. and Ghosh, J. (2003). Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3:583–617.

[Vega-Pons and Ruiz-Shulcloper, 2011] Vega-Pons, S. and Ruiz-Shulcloper, J. (2011). A survey of clustering ensemble algorithms. *IJPRAI*, 25:337–372.

[2]https://github.com/MatPont/Fashion-MNIST2

Appendices

A Autoencoder Hyper-Parameters

Layer Type	**#neurons**	**Activation**
Input	784	Leaky ReLU
Dense	512	Leaky ReLU
Dense	128	Leaky ReLU
Dense	$\{49, 100\}$	Sigmoid
Dense	128	Leaky ReLU
Dense	512	Leaky ReLU
Output	784	Sigmoid

Parameter	**Value**
Loss	Binary Cross-Entropy
Optimizer	Adam
Batch Size	256

encodingDim	**#trainable parameters**
49	948,545
100	961,652

B Convolutional Autoencoder Hyper-Parameters

B.1 Architecture 1

Layer Type	**#neurons**	**#filters**	**Kernel**	**Strides**	**Padding**	**Activation**
Input	28×28	-	-	-	-	Leaky ReLU
Conv2D	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Valid	Leaky ReLU
Dense	7×7	-	-	-	-	Sigmoid
Conv2DTranspose	-	128	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	64	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
Output (Conv2D)	-	1	(3, 3)	(1, 1)	Same	Sigmoid

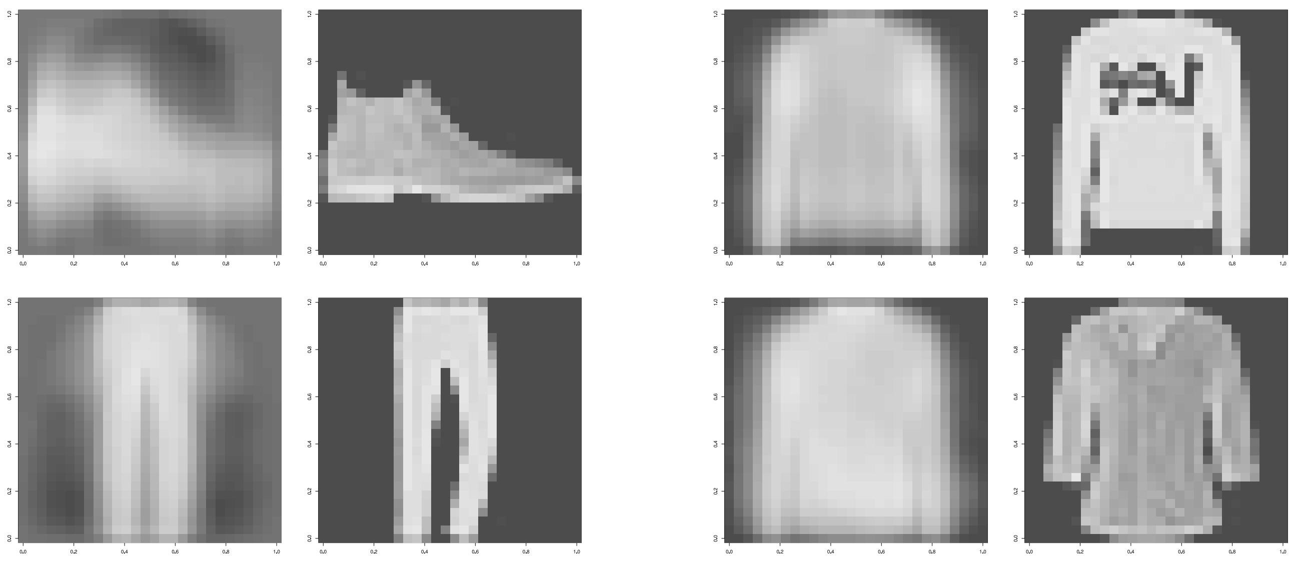
Number of trainable parameters: 228,018.

B.2 Architecture 2

Layer Type	#neurons	#filters	Kernel	Strides	Padding	Activation
Input	28×28	-	-	-	-	Leaky ReLU
Conv2D	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2D	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
MaxPooling2D	-	-	-	-	-	-
Conv2D	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2D	-	64	(3, 3)	(1, 1)	Valid	Leaky ReLU
Dense	7×7	-	-	-	-	Sigmoid
Conv2DTranspose	-	128	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	128	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	64	(3, 3)	(2, 2)	Same	Leaky ReLU
Conv2DTranspose	-	64	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
Conv2DTranspose	-	32	(3, 3)	(1, 1)	Same	Leaky ReLU
Output (Conv2D)	-	1	(3, 3)	(1, 1)	Same	Sigmoid

Number of trainable parameters: 504,882.

C Reconstructed images with PCA



D Reconstructed images with a Deep Convolutional Autoencoder