Horoma Project - Block 2

Team 1 (b2phot1)

Basile Dura
Université de Montréal
basile.dura@umontreal.ca

Philippe Marcotte
Université de Montréal
philippe.marcotte@umontreal.ca

Saber Benchalel
Université de Montréal
saber.benchalel@umontreal.ca

Sahar Bahrami
Université de Montréal
sahar.bahrami@umontreal.ca

Abstract

Our task is to analyze images of forest canopy to determine the tree species. The goal is to develop an unsupervised machine learning pipeline and apply advanced clustering algorithms to predict forest canopy properties with little to no labeled data. We use Vanilla, Convolutional, Variational and Wasserstein Autoencoders for dimensionality reduction. We use K-Means and Gaussian Mixture Models for clustering.

1 Introduction

In order to have a better understanding of our environment, the Horoma project aims to design algorithms able to identify and segment aerial photographs. During this project, we focused on a subtask whose goal was to determine the tree species from aerial pictures of forest canopy.

Given the extremely low number of labelled samples, using traditional supervised techniques was not an option, and we had to resort to unsupervized methods in order to harness the information contained in the large amount of unlabelled data.

Autoencoders play a fundamental role in unsupervised learning and in deep architectures for transfer learning and other tasks [1]. In this project, we have implemented Vanilla, Convolutional, Variational and Wasserstein Autoencoder algorithms.

This report is organized as follows. Section 2 includes data description and data selection, processing and augmentation. Section 3: the methodology and algorithms used in this study, list of our hyper-parameters and the model architecture. Sections 4 and 5 present the results we obtained and an analysis of the best models. Finally we summarize the project in section 6.

2 The Horoma dataset

2.1 Data description and data selection

The dataset (be it labelled or not) contains aerial images of forest canopy in the form of three-channel 32×32 tensors. Each pixel roughly correspond to a 30 cm by 30 cm patch of canopy.

Since the pictures were taken in a spectrum that does not correspond exactly to visible light, the images seem shifted towards low frequency colors, hence the red tint. Note that contrary to many computer vision application, the task seems very hard for a human, despite our supposed ability to learn with very few shots.

Figure 1: An example of pixel patches

In order to augment the size of the dataset, we have the possibility to use a dataset with 50 % overlap. To make sure that we could split the images into non-overlapping groups, we kept a trace of the region from which the images were taken within the forest. By choosing different regions during the split, we can guarantee that there will not be any overlap.

In the labelled dataset for validation, we also get the tree species represented in the patch.

2.1.1 Input format

The dataset is provided in the form of four files containing a subset of the $3 \times 32 \times 32$ pixel patches.

- `train_x.dat`: 150 900 examples
- `train_overlapped_x.dat`: 544 749 examples (pixel patches with 50% overlap).
- `valid_x.dat`: 480 examples.
- `valid_overlapped_x.dat`: 1 331 examples (pixel patches with 50% overlap).

Only the `valid` and `valid_overlapped` datasets contained labels, hence the necessity of using unsupervised learning methods to tackle the task.

2.1.2 Label format

The labels were provided in the form of two text files (one for each dataset) that can be loaded as `numpy` arrays. The ith value in those files is associated to the ith pixel patch of the corresponding data file.

- `valid_y.txt`: contains 480 tree species (2 characters).
- `valid_overlapped_y.dat`: contains 1331 tree species (2 characters).

2.1.3 Data processing and augmentation

Given the supposed very low semantic content of the images, we chose to be conservative in the way we dealt with data processing and augmentation.

Hence, we made minimal alterations to the images. In order for them to comply to the image tensor format and to avoid conversion bugs, we transform them to PIL image (`Python`'s de facto standard image library) and then to tensor format after some minor alterations including :

- Horizontal or vertical flip ;
- Rotation by a multiple of a quarter-turn.

By doing so, we wanted to make sure that we did not change the semantics of the images. In the case of the quarter-turn for instance, this modification is strictly equivalent to a quarter turn of the direction taken by the plane that took the picture, and should thus not alter the semantics of the image in any way[1].

3 Methodology and algorithms

Our task can be subdivided into two sub-tasks:

1. Reducing the dimensionality of the inputs

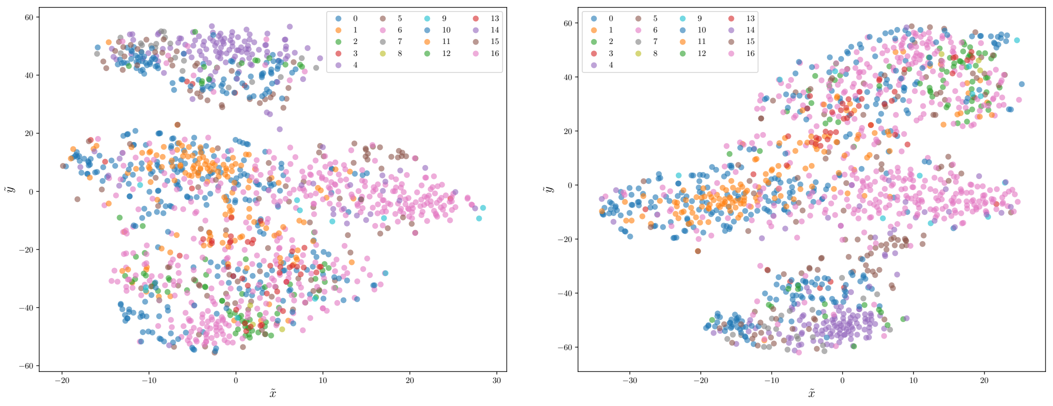
[1]By contrast, this kind of modification would probably be catastrophic in a CIFAR-100 context

2. Clustering of the latent representations

For the former task, we use autoencoder algorithms. For the latter, we have experimented with k-Means and Gaussian Mixture Models. During the block, we performed hyper-parameter tuning to try to reach the best performance on each model.

3.1 Dimensionality reduction

In order to leverage the information contained in the overwhelming fraction of data that is unlabelled, we use dimensionality reduction (DR) techniques to learn a meaningful low-dimensional latent representation of the pixel patches.



(a) Using the full $3 \times 32 \times 32$ images (b) PCA (27 components, 90% variance)

Figure 2: t-SNE representation of the latent space

During block 1 of this project, the DR sub-task was done using a principal component analysis (PCA) approach, which aims to create a linear mapping from the input space into a latent space, by keeping the directions that account for most of the variance. Because of the linearity of the model, PCA is often not expressive enough to keep meaningful details on the data.

Note that in figure 2 the t-SNE representation does not shift in a major way from the full dimensionality to PCA. Although we can see some structure, the data seems to be distributed quite uniformly and there is much overlap.

During this block, however, we focused on a non-linear and more powerful method, and resorted to autoencoders. Autoencoders are deep networks that contain two parts : an encoder, that drastically reduces the number of dimensions of the input to a manageable latent representation, and a decoder, that decodes the latent representation back into a reconstruction of the input. Since the latent representation has far lower dimensionality than the input and reconstruction, it behaves as a bottleneck and some information is lost.

By training the network to minimize the reconstruction error, we can effectively create a non-linear mapping that generates a meaningful representation of the inputs inside a latent space. While conceptually simple, autoencoders play an important role in machine learning [1].

We can make the network fit some other objective (for example, matching a prior distribution of the latent space in the case of the Wasserstein auto-encoder), by adding constraints in the form of penalization of the objective. Throughout the learning process, we must make sure that the representation is not only meaningful for the reconstruction task, but also for the main objective.

3.1.1 Vanilla AutoEncoder

We implemented a Vanilla AutoEncoder. This simple model is used as a baseline for other encoders. For this implementation, we only used linear layers for both encoding and decoding. All but one activation functions are ReLU. The last activation used is a sigmoid to generate pixel intensity values bound between 0 and 1.

3.1.2 Convolutional AutoEncoder

The input data being multi-channel square images, we expect a convolutional model to outperfom a simple model using deep linear layers since the former preserves the 2D structure of imges. To test this hypothesis, we implemented a Convolutional AutoEncoder (CAE). The use of 2D convolution and maxpool layers should create a translation agnostic model that better fits the available data.

The last encoding layers are linear to allow for a tunable latent representation size independent of input and kernel size. Dropout layers are used to prevent overfitting [5]. We use Scaled Exponential Linear Units or *SELU* activation functions instead of ReLU to avoid vanishing and exploding gradients [4].

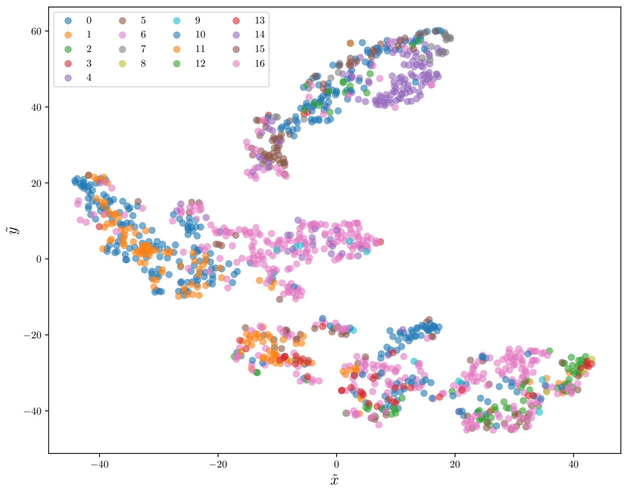


Figure 3: t-SNE representation of the latent space for the CAE

Figure 3 shows the t-SNE 2D-embedding for latent representation of the labelled set. Notice that compared to using the full dimensionality or PCA (see figure 2), the structure is much more salient —although the different classes are still intertwined. We do not expect a clustering algorithm to perform in the range of 90% accuracy from this representation.

3.1.3 Variational AutoEncoder

The Variational AutoEncoder[2] (VAE) is a variant where we add a stochastic component and a penalisation on the latent space. Instead of computing the embedding directly, the encoder returns a tuple containing the mean and diagonal variance of a normal distribution from which the latent code is sampled. Thanks to this stochasticity, we make sure that the model does not learn a sparse latent space and keeps similar examples together. This enables us to make interpolation between samples for example.

During training, we penalise the latent representation so that the encoded distribution remains close to a standard gaussian (by adding the KL divergence to the optimized loss) [3]. In this setting, the KL divergence has a closed-form solution. Excellent higher-level presentations on VAEs can also be found in various blog posts, eg here or there.

During our experiments however, it seemed that the benefits obtained by the more meaningful and evenly distributed latent representation were counteracted by the clusters coming together inside the gaussian prior. A possible fix could have included using a different weight on the penalisation of the prior, but we did not find a good candidate during our experimentation.

3.1.4 Wasserstein AutoEncoder

Similar to the VAE, the Wasserstein autoencoder (WAE) penalises the latent space as well as the reconstruction error. It does so by minimising a penalized form of the Wasserstein distance between

[2]Since using convolution layers is almost guaranteed to outperform a simple MLP, we only implemented convolutional versions of VAE and WAE.

the model distribution and the target distribution, which leads to a different regularizer than the one used by the VAE. This regularizer encourages the encoded training distribution to match the prior [6]. We implemented the maximum-mean-discrepancy (MMD) variant.

Much like the VAE, we found that the drawbacks outweighed the benefits in terms of final accuracy, and decided early on not to pursue in this direction.

3.2 Clustering

The given baseline uses PCA to capture 90% of the variance and *100*-means for the clustering part to obtain 26.51% test accuracy. However, the clustering algorithm is trained on `train` but the resulting clusters are labelled with `valid_overlapped` and tested on `valid`, even though the two share a great number of pixel patches. This seems to cause overfitting, as the validation accuracy is 50.94%.

We noticed that the F_1 score and the accuracy varied a lot depending on the seed used to randomly split the validation dataset into a labelling and a testing sets. The very low number of samples seems to cause high variance in results.

3.2.1 Algorithms

For the clustering task, we experimented with k-Means and a Gaussian Mixture Model (GMM). We expected GMM to outperform k-Means, since it relaxes the assumption made by k-Means that the data is distributed according to a flat geometry : using GMM, clusters can each have there own variance matrix. Our experiments show that this was not the case, as k-Means seemed to perform consistently better than GMM (by a slight but robust margin).

Moreover, since the final task is to classify the trees represented in the pixel patches, we used the accuracy and F_1 score of the latter to monitor the validation performance.

3.2.2 The clustering metric

Since the final clustering task is evaluated using the accuracy and F_1 score of the prediction returned by our clustering model, we decided to use those scores as validation metrics during training (for early stopping for instance).

However, we found that the split of the labelled dataset had a tremendous influence on the final result (which ranged from 15% to 50% depending on the split). In order to curb this phenomenon, we decided to use an ensemble method, in which we trained multiple clustering algorithms and labelled them using different splits. At prediction time, the output of the model will use the majority vote from those algorithms.

This method enables us to be more robust to ill-defined splits, and use the entire training set while still monitoring the performance of every single model from the ensemble.

3.2.3 k-Means-friendly embeddings

In order to make the most of our dimensionality reduction technique in the context of clustering, we can add another term to our objective function that aims to pack the samples around their nearest centroid. Indeed, most learning approaches treat dimensionality reduction and clustering sequentially, but recent research has shown that optimizing the two tasks jointly can substantially improve the performance of both [7].

We implemented a new trainer class that takes into account the regular loss of the autoencoder and the divergence of the data points relative to their centroid to obtain a new loss function.

3.3 Hyper-parameter tuning

In order to select the best set of parameters, we ran a hyper-parameter search using Scikit-Optimize's Bayesian optimisation capabilities.

For each model, we tried various values for the latent dimension, the dropout factor and the size of the dense layers surrounding the latent representation layer. We performed the hyper-parameter search using the validation score with 20, 80 and 300 clusters. For better performance, one should do the hyper-parameter search with the final metric (in this case the F_1 score). Unfortunately we ran out of time and using such metric greatly increases the time per epoch.

4 Reconstruction Accuracy and F_1 score

The best results for the aforementioned AutoEncoders are presented in table 1. They cannot be directly compared to the other losses. The CAE is the model with the highest score and lowest losses. Note that the losses of the VAE contains the KL divergence.

Table 1: Comparison of the F_1 score, loss and accuracy for different Models

Model	**F_1**	**Accuracy**	**Train Loss**	**Valid Loss**
Vanilla AutoEncoder			0.0109	0.0100
Convolutional AutoEncoder	**37.9%**	**38.8%**	0.0088	0.0084
Variational AutoEncoder	13.5%	27.8%	0.02762	0.05315

For the hyperparameter search of the CAE, we searched for 1^{st} dense layer output values in the range $[50; 120]$ and 2^{nd} dense layer output (latent dimension) sizes in the range $[30; 100]$. Table 2 shows the final hyperparameters for the CAE model that produced the highest F_1 score for the clustering.

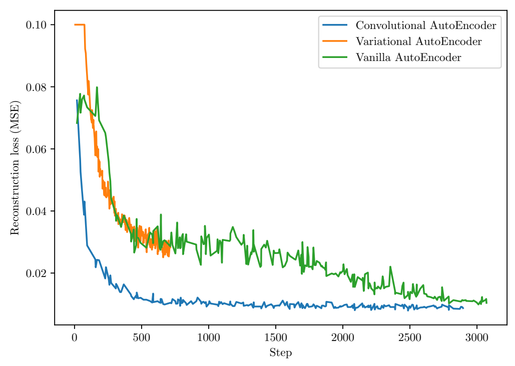
Table 2: Best Convolutional AutoEncoder model

	Layer	**Kernel**	**Output**	**Dropout**
Encoder	1st Convolution	5×5	$10 \times 28 \times 28$	–
	Max Pooling	2×2	$10 \times 14 \times 14$	–
	2nd Convolution	5×5	$20 \times 10 \times 10$	0.1
	Max Pooling	2×2	$20 \times 5 \times 5$	–
	1st Dense Layer	–	50	0.1
	2nd Dense Layer	–	40	–
Decoder	1st Dense Layer	–	500	0.1
	2nd Dense Layer	–	3072	–

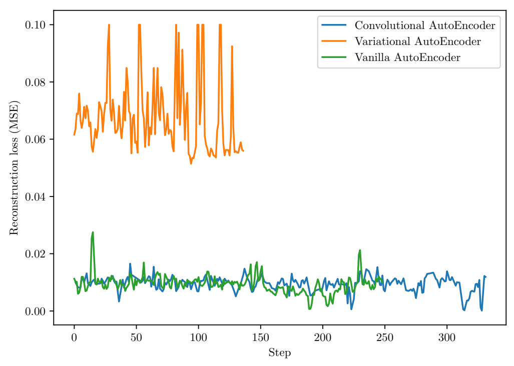
Figure 4 shows the evolution of the reconstruction loss during training for three different models. Note that as expected, the CAE yields the best results.

5 Analysis

Our early experiments showed that VAE models were showing better results when using GMM for the clustering task, while CAE models were showing better results with k-means. However, GMM clustering tasks were taking on average 90 minutes to run 3 clustering models as opposed to 12 minutes on average for 3 k-means. A single hyperparameter search would take over 12 days for 20 experiments of 10 epochs with GMM. A similar task with k-means would take around 40 hours. Time being a valuable resource, we decided to focus our study and computation time on CAE models with k-means. This explains the lower step count for VAE losses in figure 4.



(a) Training loss



(b) Validation loss

Figure 4: Reconstruction loss (MSE)

As expected and exemplified in figure 4, the reconstruction error is much lower when using convolutional layers, since they preserve the two-dimensional structure of the image. As such, two images shifted by a few pixel should remain extremely close in the latent space using a CAE, whereas the vanilla AutoEncoder will send them in very different parts of the space.

6 Conclusion and further directions

Classification with very few labelled samples is a very arduous task, and the relatively low performance of our models embodies this difficulty.

During the project, we have tried various methods aiming to use the large amount of unlabelled data in the hope of helping the final clustering class. These methods enabled us to outperform the results from block one, by leveraging the expressivity of AutoEncoders in order to generate a somewhat meaningful low-dimensional and non-linear manifold. As expected, adding convolutional layers to the AutoEncoders improved their results on image inputs.

However, we reckon that we did not reach the asymptote. In order to push the boundaries further, we believe that some methods that we did not have time to implement are worth the try. For example, we could have used GAN-like[3] penalization of the latent space in order to address the two sub-tasks at once.

Another approach could come from the subfield of few-shot learning. Methods such as *Centroid Networks* by Mila's very own Gabriel Huang [2] are almost tailored to our specific task, but we did not have time to implement such techniques.

References

[1] Pierre Baldi. Autoencoders, unsupervised learning and deep architectures. In *Proceedings of the 2011 International Conference on Unsupervised and Transfer Learning Workshop - Volume 27*, UTLW'11, pages 37–50. JMLR.org, 2011.

[2] Gabriel Huang, Hugo Larochelle, and Simon Lacoste-Julien. Centroid networks for few-shot clustering and unsupervised few-shot classification. *arXiv preprint arXiv:1902.08605*, 2019.

[3] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013. cite arxiv:1312.6114.

[4] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *NIPS*, 2017.

[3]Generative Adversarial Network

[5] Nitish Srivastava, Geoffrey Hinto, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15*, 2014.

[6] Ilya Tolstikhin, Olivier Bousquet, Sylvain Gelly, and Bernhard Schoelkopf. Wasserstein Auto-Encoders. *arXiv e-prints*, page arXiv:1711.01558, Nov 2017.

[7] Bo Yang, Xiao Fu, Nicholas D. Sidiropoulos, and Mingyi Hong. Towards k-means-friendly spaces: Simultaneous deep learning and clustering. *CoRR*, abs/1610.04794, 2016.