# IFT 6759 - Horoma - Block 2

**Arlie Coles** **Jonathan Guymont** **Nicolas Laliberté** **Yishi Xu**

## 1 Introduction

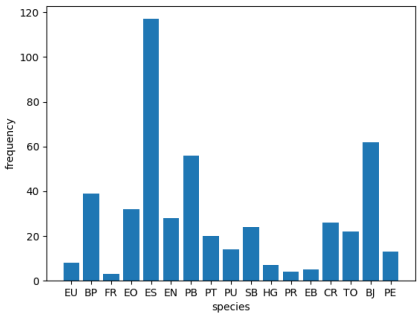
Horoma is a Montreal-based company that seeks to automatically characterize and segment large images of forest canopies in order to determine traits such as tree species. In Block 1, a set of characteristics of tree canopy images was predicted from an unlabeled data set by using a PCA dimensionality reduction followed by a K-Means clustering. In Block 2, the task is reduced to predicting the species of the trees in the unlabeled canopy images, but we seek to improve on the PCA dimensionality reduction by instead using autoencoder methods to "encode" the examples in a latent space, hopefully extracting meaningful features most useful for K-Means clustering.

## 2 Methodology

### 2.1 Dataset

The Horoma training dataset consists of 150,900 unlabeled "pixel patches", each of which is $32 \times 32$ pixels. Each pixel in these patches has an RGB value (from 0 to 255) associated with it. Also included is a labeled validation dataset, consisting of 480 pixel patches, each of which includes one of 21 possible species labels.

(a) Pixel patches of different species in the validation set; each row represents a different species




(b) Frequency of species present in the validation set

Figure 1: Data visualization

Figure 1 shows two challenging aspects of the species classification problem. The histogram in Figure 1b shows that the validation set is highly unbalanced; for instance, the species ES accounts for about one quarter of the validation data. Another challenging aspect is that some of the species are, at least to the human eye, difficult or impossible to tell apart. For instance, the images from the second and third row in Figure 1a are extremely similar. This makes it challenging to create with a classifier that does not overfit to noise, as which features are important for the classification task are not clear.

When testing our classification approach described in the following section, 336 examples of the labeled validation set were used to train the K-Means portion of the classifier, and the remaining 144 examples were held out, and used to validate the accuracy of the model. Given the unbalanced nature of the labeled validation set, we must be cautious when splitting it to create this held-out set, since

this may lead to high variance in the evaluation. We first split the validation set with respect to its proportion of classes. For example, if class $A$ represents $0.3$ of the validation set, then we ensure that the proportion of the class $A$ among the held-out set is also $0.3$. Furthermore, we also randomly split the validation set this way $10$ times, and carry out an evaluation on each of the $10$ splits; we report the arithmetic mean of the resulting accuracies as a metric to evaluate our model in order to give a more accurate idea of its performance in spite of high variance.

### 2.2 Approach

There are two main phases to this unlabeled species classification task:

1. The *encoding phase*. First, each training example is encoded into a lower-dimensional latent space. Ideally, this dimensionality reduction also represents a feature extraction, so that the most salient features of the example are represented while the less relevant features are reduced.
2. The *clustering phase*. Second, each of the encoded examples are clustered in an unsupervised manner, where each cluster represents a possible class (species in this case).

In Block 1, the encoding phase consisted of a simple PCA over 30 components, which is a linear dimensionality reduction. In Block 2, we explore two types of *autoencoders*, which are nonlinear dimensionality reduction methods that exploit neural network architecture to encode an example into a latent space, and improve on this encoding by training on an objective that minimizes the *reconstruction loss* when decoding the latent representation. (An example of a typical autoencoder architecture can be found in Figure 2.) Then, the encoded representations are fed into a K-Means clusterer in the clustering phase, which ultimately tags each example with a species label. A classifier thus consists of an encoder and a clusterer (which is always K-Means in this case).

First, we propose a baseline "vanilla" autoencoder, which consists of an MLP encoder that takes flattened pixel patches as input and a decoder that reverses the operations of the encoder in reconstructing the original flattened pixel patch. Due to the non-linearity of these operations, the features learned at the bottleneck may be able to model more particularities of the data useful for clustering than the linear PCA.

We also propose a convolutional autoencoder, which consists of a convolutional encoder and decoder. This has the same advantage of the vanilla autoencoder in its non-linearity, but also is able to exploit the substructure of the images, since it need not flatten them at the input, and can learn non-flattened representations at the bottleneck that may also be good features for clustering. Figure 2 shows the architecture of a typical convolutional autoencoder.

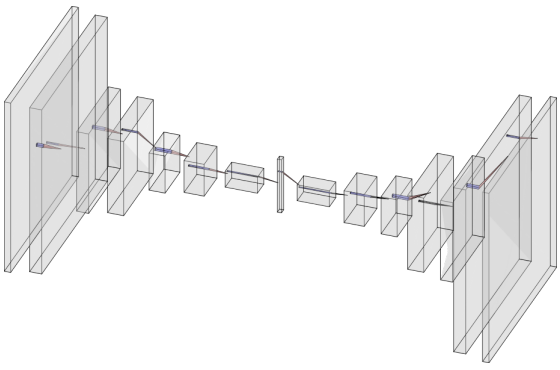


Figure 2: Example of a convolutional autoencoder with 3 convolution layers all directly followed by a max pool layer. The network is trained by minimizing the mean square error between the input (left) and output (right) after passing through the reduced-dimension bottleneck (middle). (Figure generated with [2].)

After we get the latent representations of unlabeled images from either PCA or the bottlenecks of a vanilla or convolutional autoencoder, we use these latent representations to train a K-means clusterer. In our experiments, we choose to use 21 clusters as it is the number of tree species existing in the

dataset. We then use a subset of labeled images to assign labels to these clusters. More specifically, we extract the latent representations of the labeled images and then predict which cluster they fall into the K-means clustering. The label of each cluster is determined by the majority vote of the labeled image in that cluster. The classification accuracy is then calculated using the remaining subset of labeled images which were not used in assigning labels to clusters.

### 2.3 Models

We propose a vanilla autoencoder that we call *Vanilla-AE*. Vanilla-AE consists of a two-layer MLP encoder that takes flattened pixel patches as input and reduces them into a 32-dimensional latent representation. The decoder then reverses these operations, reconstructing the input. Vanilla-AE trains using an Adam optimizer, an MSE loss, and a learning rate of 0.001 over 20 epochs.

We also propose a convolutional autoencoder that we call *CAE*. CAE uses a 4-layer convolutional network, with max-pooling of size 2. Likewise, the decoder inverses (or approximate the inverses) of these operations. To be more precise, for each convolutional layer in the encoder, we associate a transposed convolutional layer (also called deconvolutional layer) that acts as an inverse. In order to reverse the max pool application, for which the inverse does not exist, we apply the max unpool function, setting every lost scalars to zero. The convolutional network uses ReLU non-linearities, ensuring the features maps are positive. To assure valid reconstruction in the decoder, we apply ReLU at each layer where it should be positive. We also apply batch norm in the decoder, where the inputs should be normalized according to the encoder. A fully-connected layer separates the convolutional encoder and the deconvolutional decoder. The whole network was train using an Adam optimizer with batch size equal to 256 and a learning rate of 0.01 over 15 epochs.

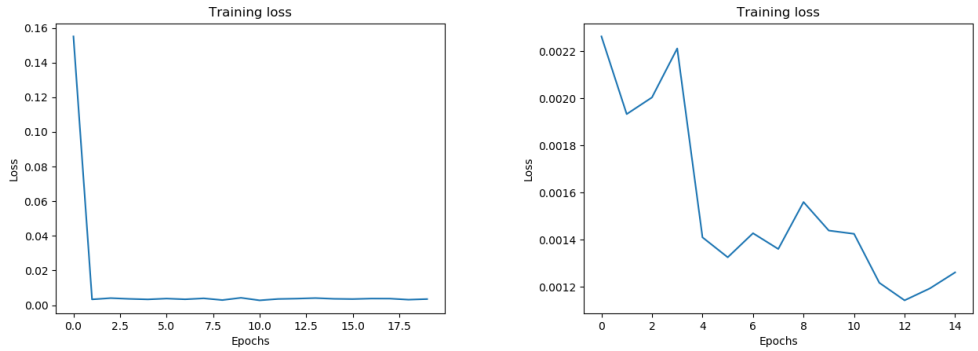
## 3 Results

We therefore test three classification methods (one for each encoding strategy), each of which uses K-Means clustering on the encoded representations: *PCA-KMeans*, as constructed by Block 1; *Vanilla-AE-Kmeans*; and *CAE-KMeans*. Table 1 shows the classification accuracy of each method on the held-out labeled validation set.

| Method | Classification accuracy (%) |
|---|---|
| PCA-KMeans (Block 1) | 29.03 |
| Vanilla-AE-KMeans | 32.43 |
| CAE-KMeans | 39.79 |

Table 1: Classification accuracies of each method

Figure 3 shows the training loss curves for Vanilla-AE and CAE over time, tracked by TensorboardX.




(a) Vanilla-AE loss

(b) CAE loss

Figure 3: Autoencoder training loss curves

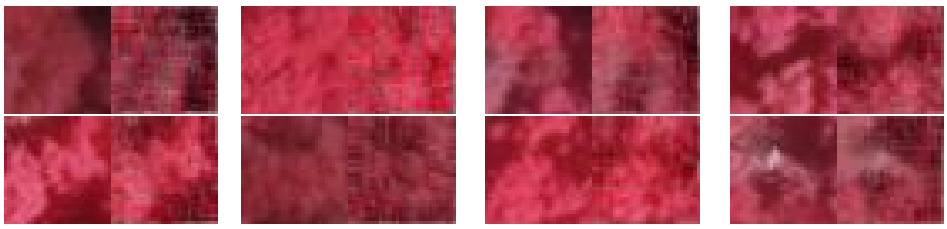


Figure 4: Reconstruction over several epochs using CAE

## 4 Discussion

From the experiments we see that CAE-KMeans method performs significantly better than PCA-KMeans or Vanilla-AE-KMeans. This illustrates that CAE could extract more meaningful latent representations of images than the other two methods. This result is not a surprise, as the convolutional neural network usually works well with images, since it can preserve the local connectivity and overall substructure of pixels in the images.

We also remark that Vanilla-AE slightly outperforms PCA-KMeans; we suspect that even better performance might be observed if we were to exhaustively tune the hyperparameters of Vanilla-AE.

As we can see in Figure 4, the embedding generated by the CAE contains enough information for a valid reconstruction of the initial image. On the other hand, the embedding does not seem to capture enough features that give information on the distinction of species useful for clustering. We can see in Figure 5 that some clusters are formed, but as a whole the distribution is somewhat messy. This explains the overall poor results on classification (though indeed CAE is the best-performing encoding method of the three).

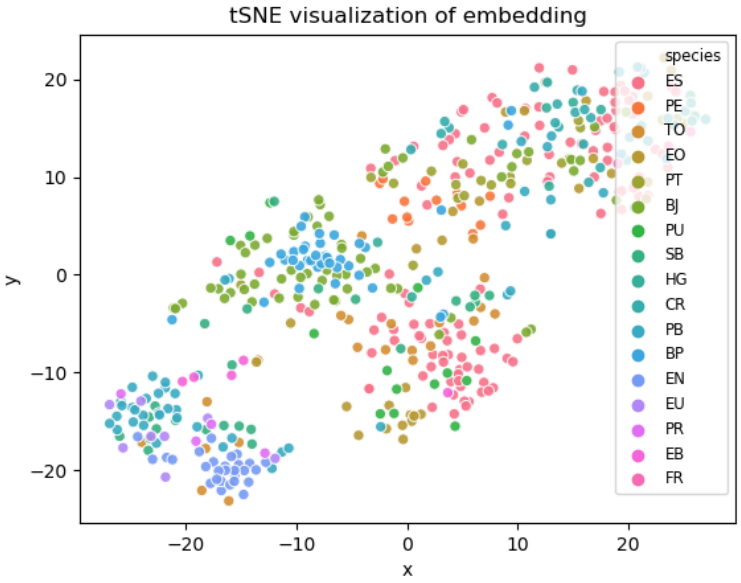



Figure 5: Visualization of encoding generated by CAE in 2-D space.

In order to address this problem, one could train the autoencoder using the reconstruction loss as well as the clustering loss. Several clustering losses could be used, potentially including a K-Means loss which computes the sum of the distance between a representation and its centroid [1]. Using this loss, we ensure that the representations are evenly distributed around cluster centers. We could alternatively consider the following loss:

$$L(\Theta) = \alpha L_c(\Theta) + (1 - \alpha) L_r(\Theta)$$

where $L_r$ and $L_c$ are the reconstruction and clustering losses respectively. The constant $\alpha$ will be an additional hyperparemeter for training. It could be set at a fixed constant or could vary on a chosen schedule. In our setting, $\alpha = 0$. Note that this method will require a split to the validation set which is already small.

Another method we tried is to use a sort of bagging method when assigning labels to K-Means clusters. Due to the fact that we only have 480 labeled images, and that the distribution of the species are very unbalanced, the way we split the validation sets to subsets for assigning labels can lead to high variance in terms of cluster labeling and final accuracy. We tried to address this issue by splitting the validation set multiple times and each time use a subset of validation set to assign labels to clusters. At the end, we do a majority vote to determine the labels of clusters. We did not merge these functions to the final code due to a lack of time, but they are present in the `bagging` branch if future teams would like to explore them.

# 5 Conclusion

In Block 2, we kept a similar pipeline to that of Block 1. We divided the problem into two major blocks, the encoding phase and the clustering phase. For the encoding phase, we added two new methods, Vanilla-AE and CAE, to compress images to latent representations. For the clustering phase, we used the same K-Means method as the previous team did in Block 1.

We achieved a classification accuracy of $39.79\%$ with the CAE-KMeans method, which is higher than the accuracy of PCA-KMeans, which was used in Block 1, by about $10.76\%$. We believe this shows that a convolutional autoencoder has a better capability of compressing image into meaningful latent representations than PCA.

## 5.1 Further future work

Many of the issues in this block are due to the limited number of labeled images. One possible simple future direction could be toward data augmentation. Future teams could try to increase the size of the validation set by augmentation techniques such as flipping and distortion.

The data is also highly unbalanced, which is a challenging aspect for a classification task (see Figure 1b). Another possible future task could be balancing data by data augmentation on classes that are less represented, though we note that doing so might give worse accuracy. Considering that the $5$ most represented classes in the data accounts for about $80\%$ of the data, a random classifier (which predicts randomly between those $5$ classes) would score a little less than $20\%$. In the other case, with the data balanced the random classifier would score little less than $5\%$. Viewed this way, balancing the data will probably gives worse result but better generalization among all classes. Yet other methods could be explored, such as applying weights based on the proportion of the class to the loss function. If the class is not well represented, a heavier weight could be applied to the loss function.

Other potentially fruitful avenues to explore could include using a variational autoencoder for the encoding phase, and/or a Gaussian Mixture Model (GMM) for the clustering phase.

# References


[1] Elie Aljalbout et al. “Clustering with Deep Learning: Taxonomy and New Methods”. In: *arXiv e-prints*, arXiv:1801.07648 (Jan. 2018), arXiv:1801.07648. arXiv: `1801.07648 [cs.LG]`.

[2] Alexander Lenail. “LeNail, (2019). NN-SVG: Publication-Ready Neural Network Architecture”. In: *Schematics. Journal of Open Source Software, 4(33), 747, https://doi.org/10.21105/joss.00747* (2019).