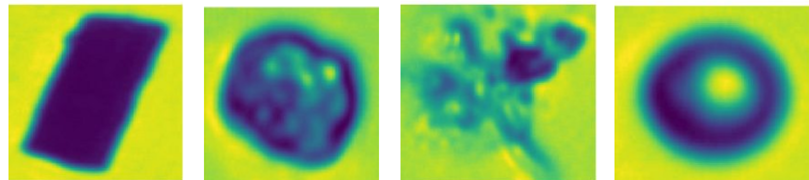
Peruvian ice core insolubles

Multi Classification using a CNN and unsupervised learning using an autoencoder and UMAP

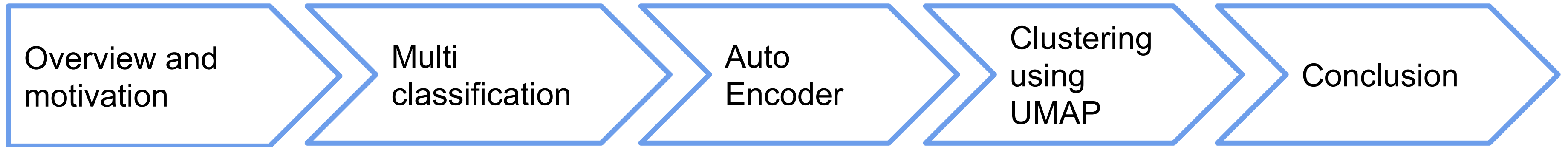
Applied Machine Learning - Final Project
Supervisor: Troels Christian Petersen

Project work done equally by:
Eva Lopez Rojo, Sara Schjødt Kjær
Ulrik Hvid, Andreas Mosgaard Jørgensen
& Peter Andresen

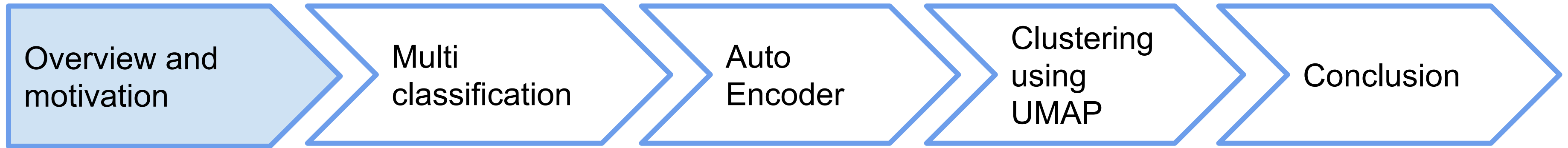
Source: Edubucher, Wikimedia Commons



Overview of presentation



Overview of presentation



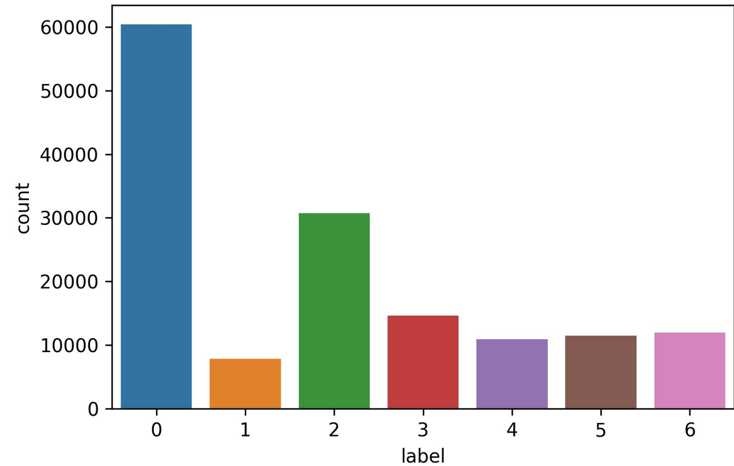
Data set, motivation and objective

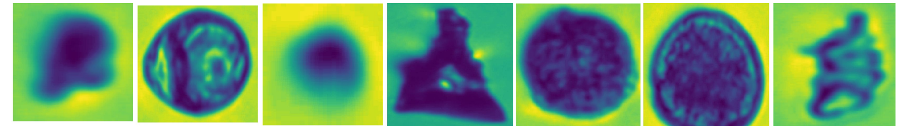
Data

- Two datasets: Fabricated training data (7 classes) and unclassified Peruvian data
- Pictures (128,128) pixels and metadata (36) of insolubles from ice cores
- Training dataset: 147960 samples (ash, dust, pollen, contamination)
- Peruvian dataset: 102764 samples
- Relatively flat training distribution → no re-weighting

Objective

- Classify the insolubles from Peru dataset
- Explore if there are other possible classes



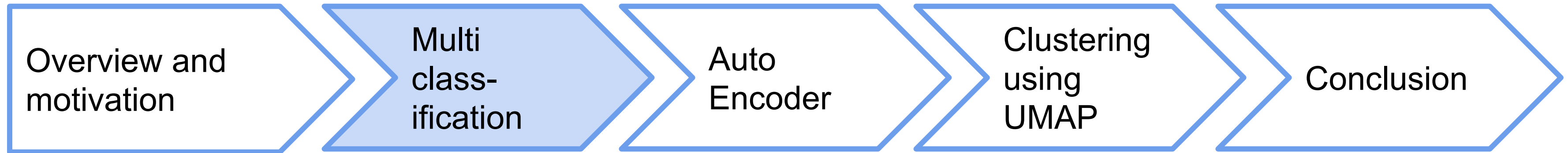


0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Approach

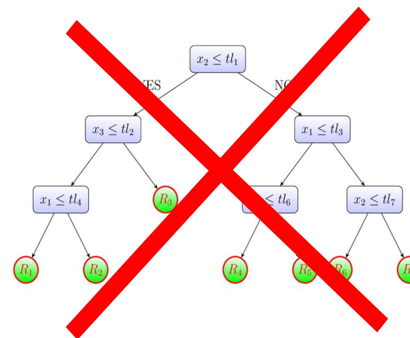
- Multi Classification using a BDT on metadata, or CNN on the pictures both with and without metadata
- Autoencoder to recreate photos and generate latent spaces
- UMAP either latent space or 2nd last layer in CNN to try to identify new clusters
- CNN is slow to train, so we used Google Colab Pro+ GPUs to speed up the training by 30-40 times (3 hours instead of 5 days)

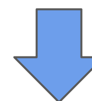
Multi Classification

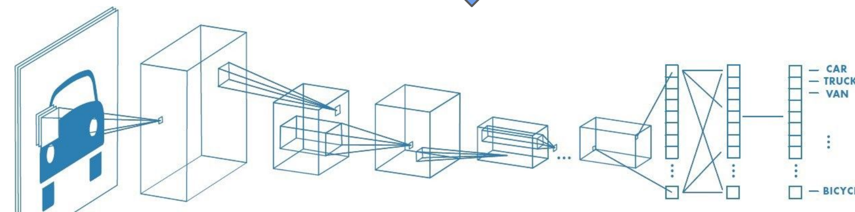


Multi Classification - what method to use?

- First attempt: BDT from XGBoost (optimized using randomized parameter search, CV), on metadata.
- Accuracy of 86% - 87%. Not very satisfying, most information is probably in the images
- Tried CNN instead, with out without metadata inspired by code provided by Amalie Regitze Faber Mygind







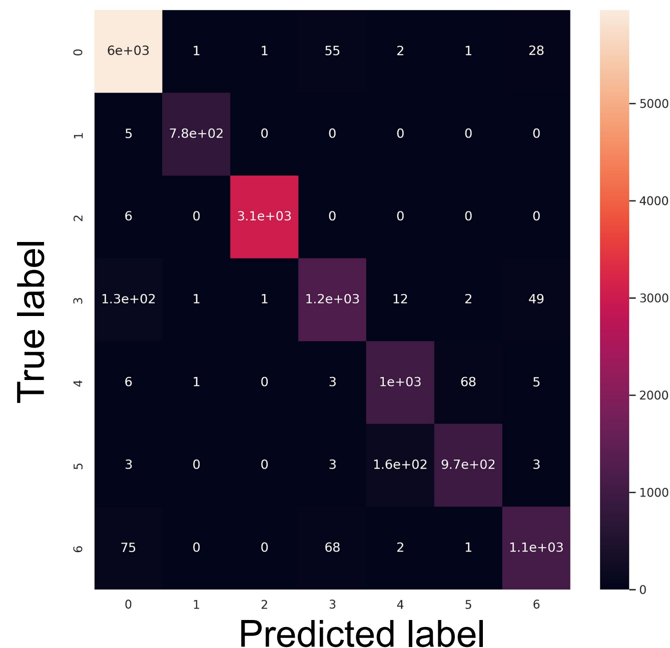
Multi Classification - Architecture & results on training data

Architecture

- Resnet18 → (512) → (64) → (40) →(7)
- Batchnorms, dropouts and RELU are applied in-between layers
- Loss: multiclass cross entropy
- Optimized by experimenting with learning rate

Results

- Accuracy on validation training data = 95-96%
- Nice! So what are the insolubles in the Peru ice cores?

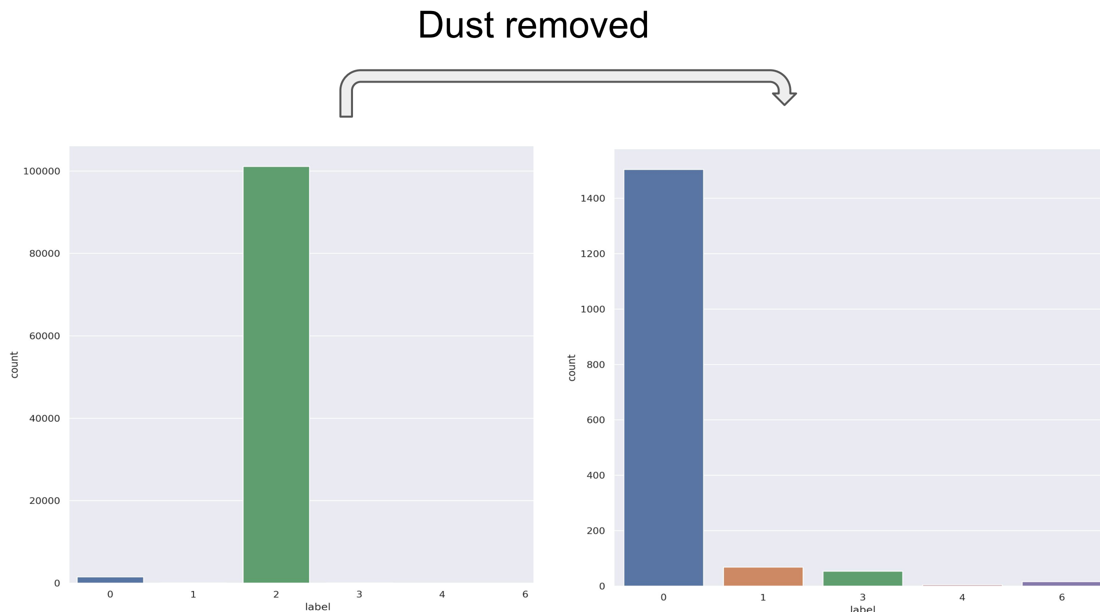


0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Multi Classification - Prediction on Peru data

How many of each class?

0:	1504	Ash #1
1:	68	Pollen #1
2:	101118	Dust
3:	54	Ash #2
4:	4	Pollen #2
5:	0	Pollen #3
6:	16	Contamination

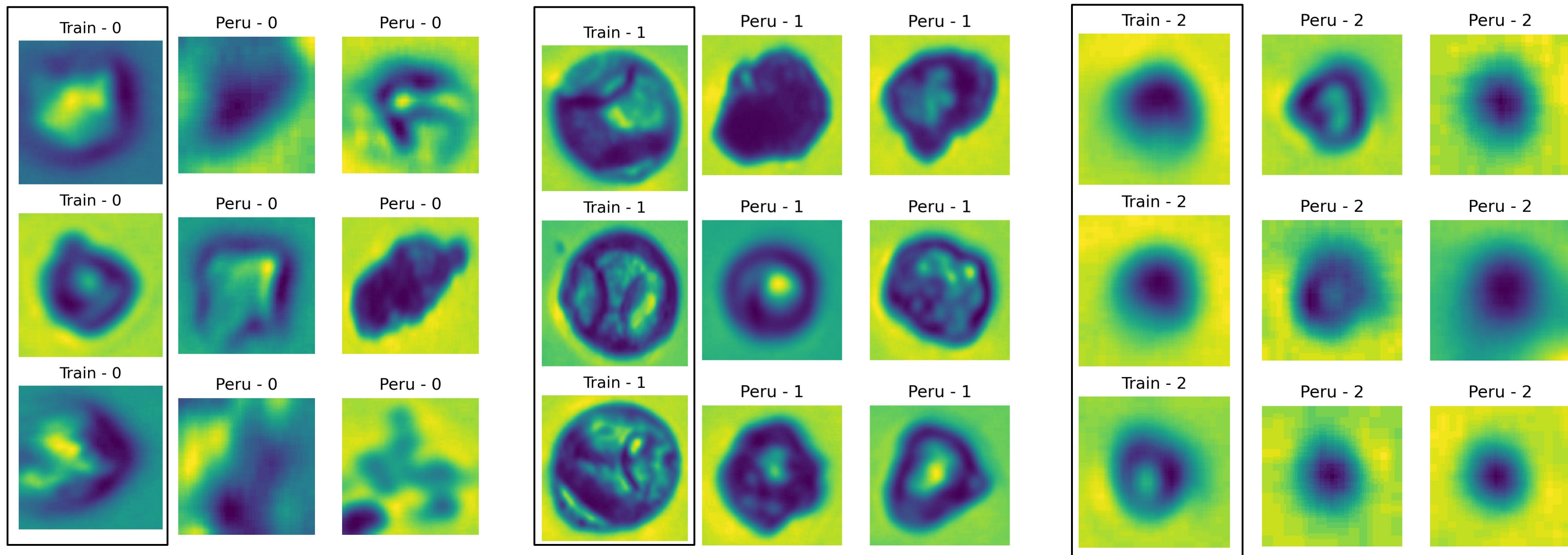


Mainly dust… but also quite a lot of class 0 (ash)

But, how does it actually look?

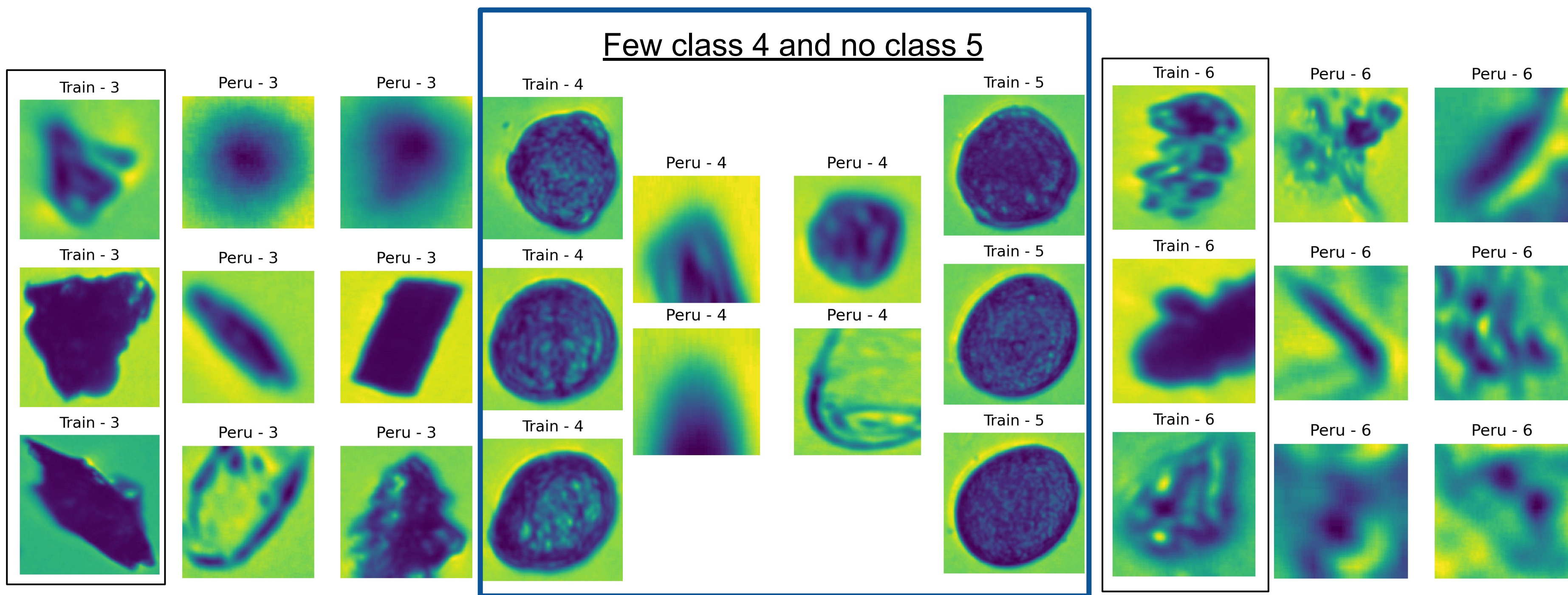
0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Multi Classification - Plots of the classes



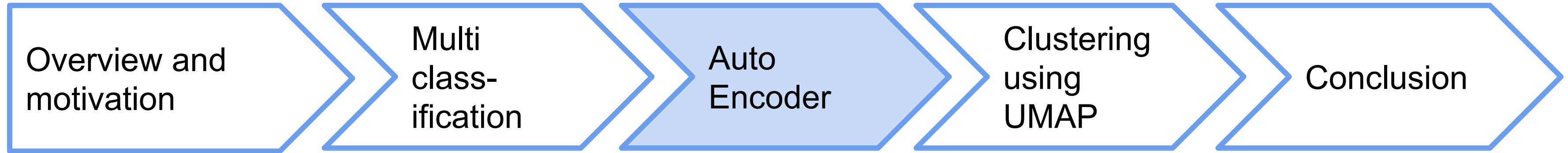
0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Multi Classification - Plots of the classes



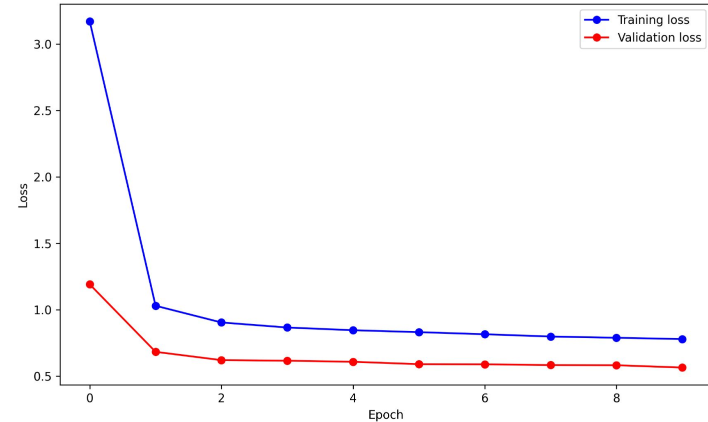
0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

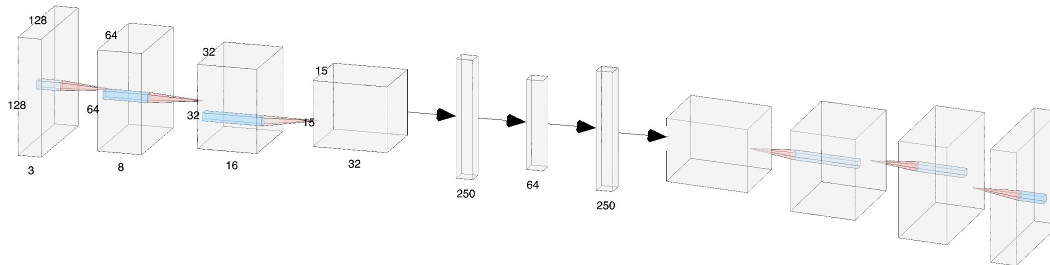
Autoencoder



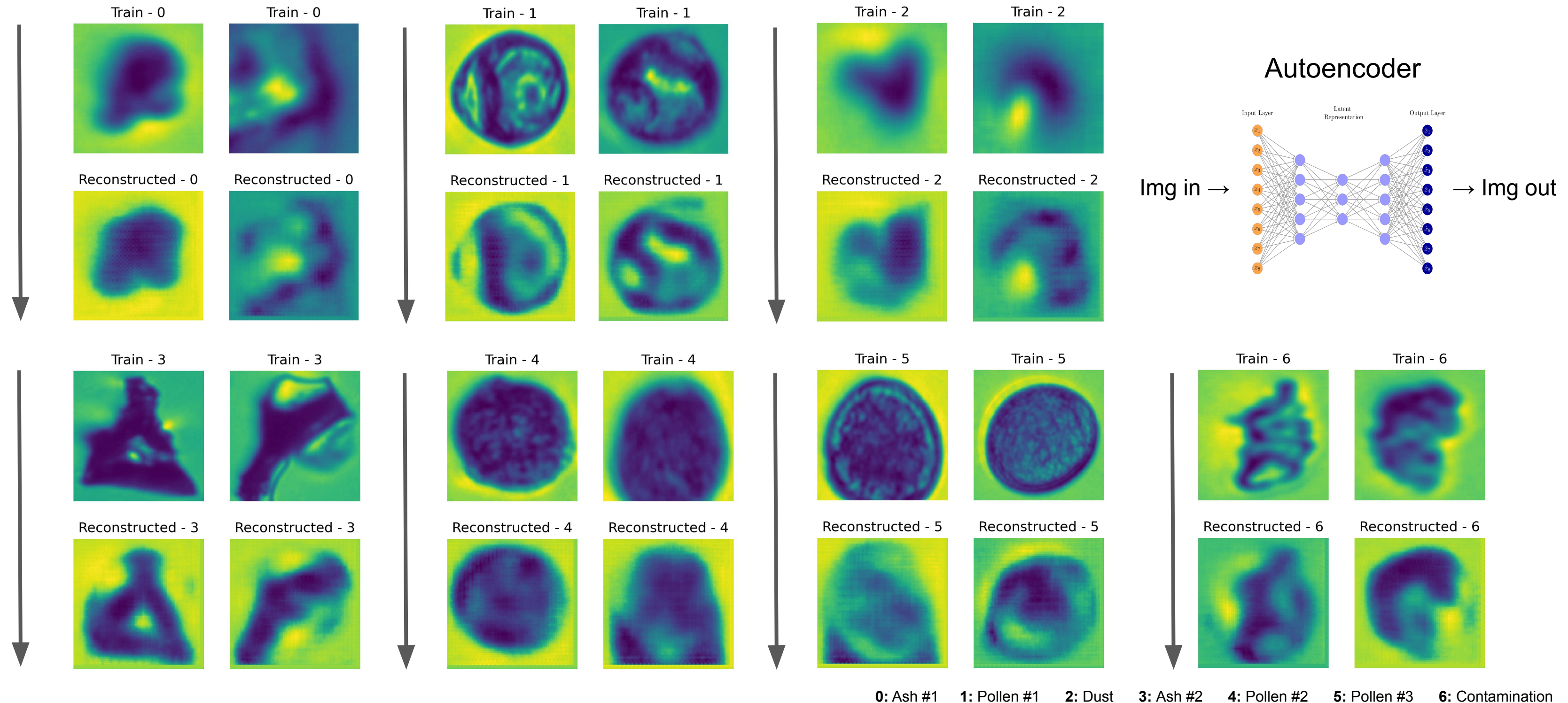
Autoencoder - Architecture and training epochs

- Layers: 3 convolutional, 2 linear
- Latent Space dimension: 32 or 64
- Loss: Mean squared error
- Let’s see how well it works



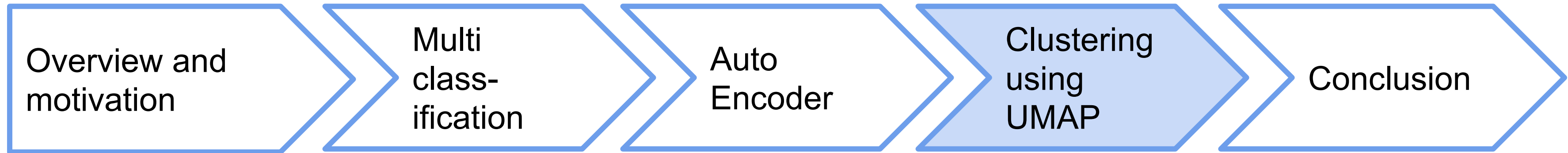


Autoencoder - Reconstructed images - Training



0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

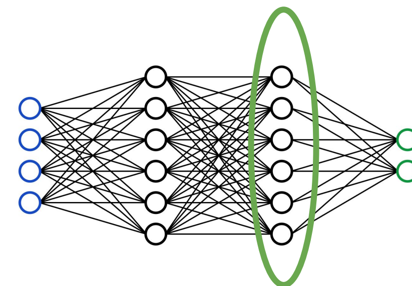
Finding new types of insolubles using UMAP



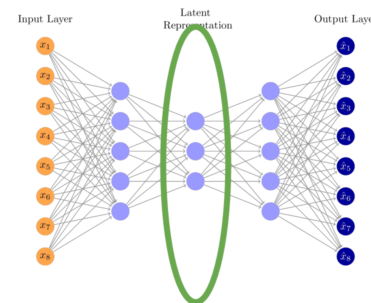
Could our algorithms discover an eighth type of insoluble?

<u>Two approaches</u>

- **Multi classification CNN**
  - Trained on either 6 or 7 classes
  - UMAP or parametrized UMAP on training data and Peru data



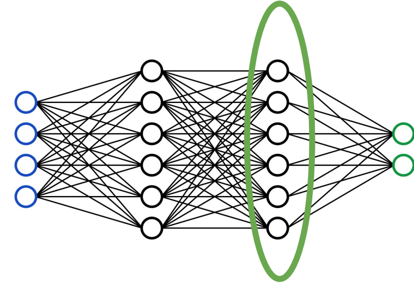
- **Latent space of the Auto encoder.**
  - Trained on either training data 6 or 7 classes or Peru data
  - UMAP or parametrized UMAP



Could our algorithms discover an eighth type of insoluble?

Two approaches

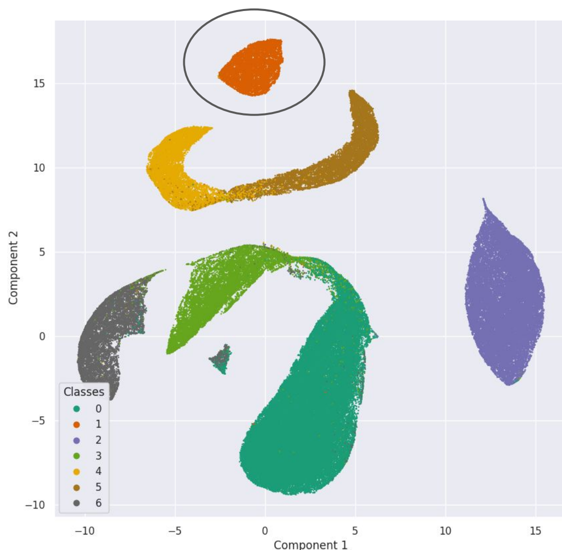
- **Multi classification CNN**
  - Trained on either 6 or 7 classes
  - UMAP or parametrized UMAP on training data and Peru data



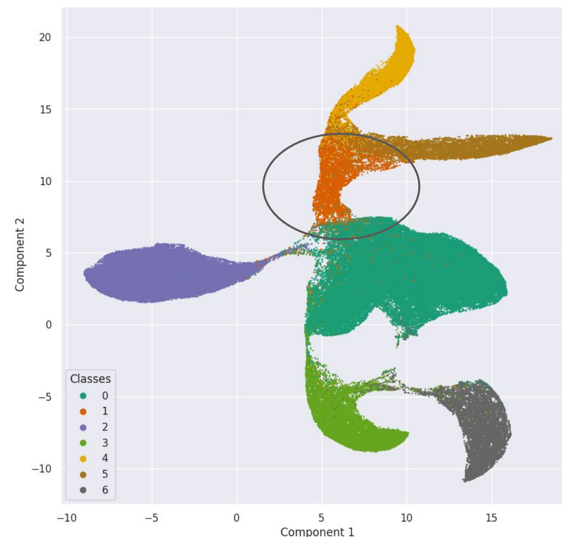
- **Latent space of the Auto encoder.**
  - Trained on either training data 6 or 7 classes or Peru data
  - UMAP or parametrized UMAP

UMAP 2nd last layer CNN - Trained on 6 or 7 classes

- UMAP identifies 7th class - though not as separated as when training on 7
- This shows that the method could work, i.e a new cluster could appear in UMAP on Peru data



CNN Trained on 7 classes



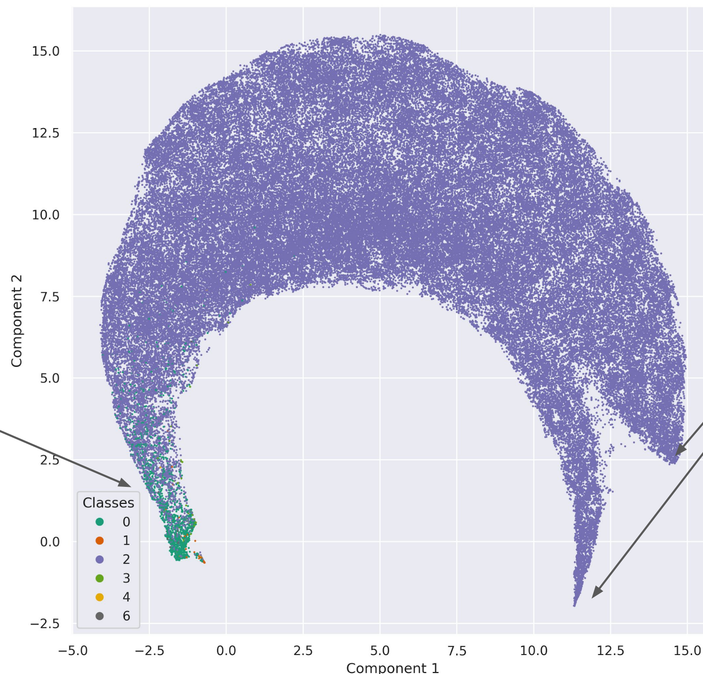
CNN Trained on 6 classes

0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

UMAP 2nd last layer of CNN on Peru data

- No new clusters in UMAP of Peru data - perhaps because mostly dust?
- Colored based on our best predictions

Non dust types are “separate” from the dust



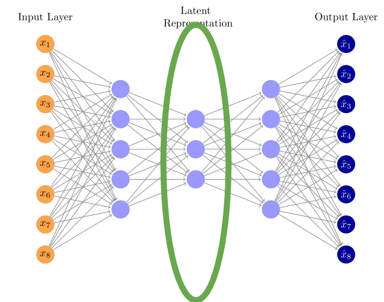
Two types of dust? We’ll get back to it!

0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Could our algorithms discover an eighth type of insoluble?

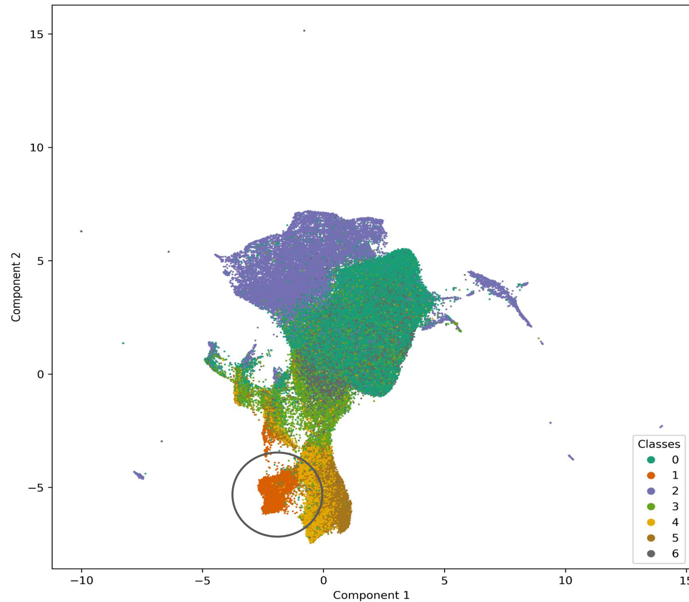
Two approaches

- **Multi classification CNN**
  - Trained on either 6 or 7 classes
  - UMAP or parametrized UMAP on training data and Peru data
- **Latent space of the Auto encoder.**
  - Trained on either training data 6 or 7 classes or Peru data
  - UMAP or parametrized UMAP

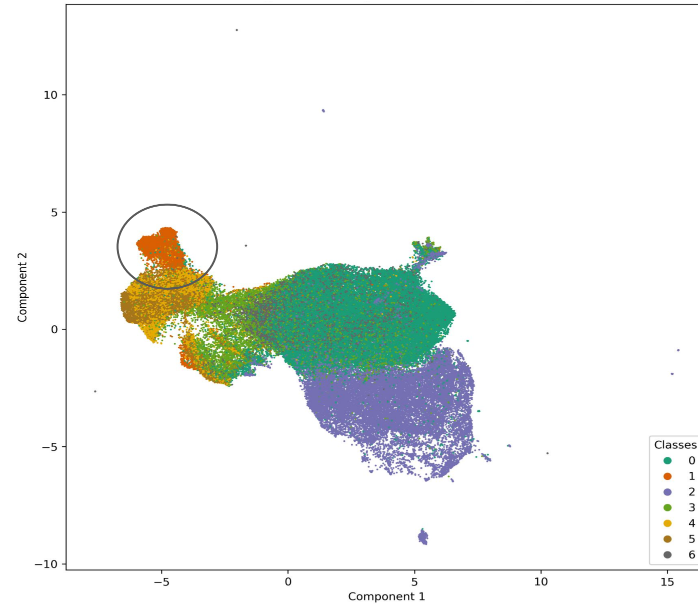


Autoencoder - Train on 6/7 classes, UMAP on 7 classes

Trained on 7 classes



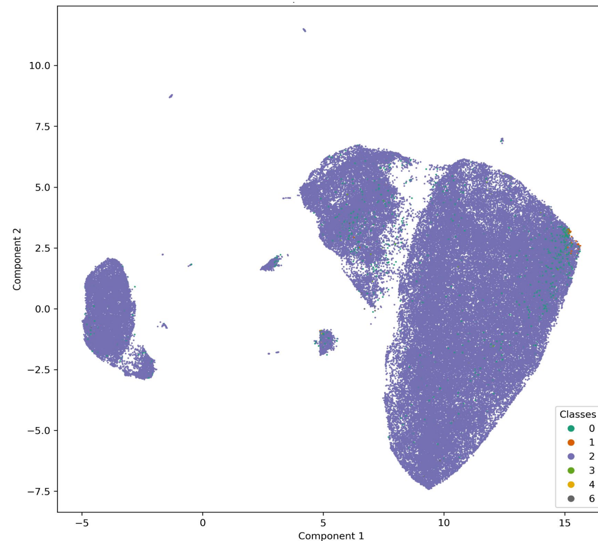
Trained on 6 classes (without class 1)

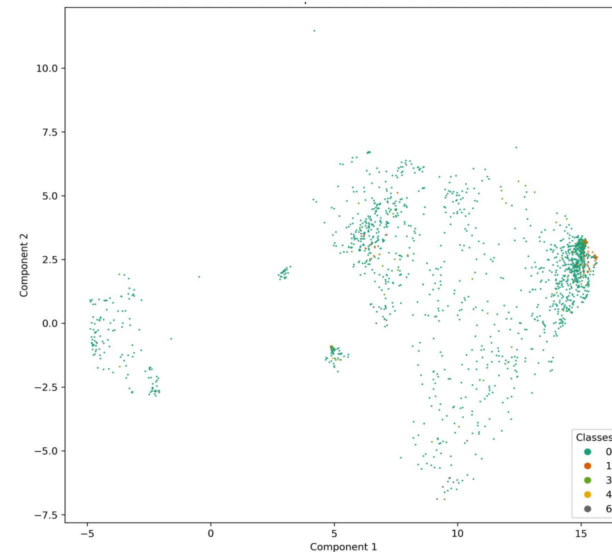


0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Autoencoder applied to Peruvian data

- UMAP of Peru data in latent space gives different clustering

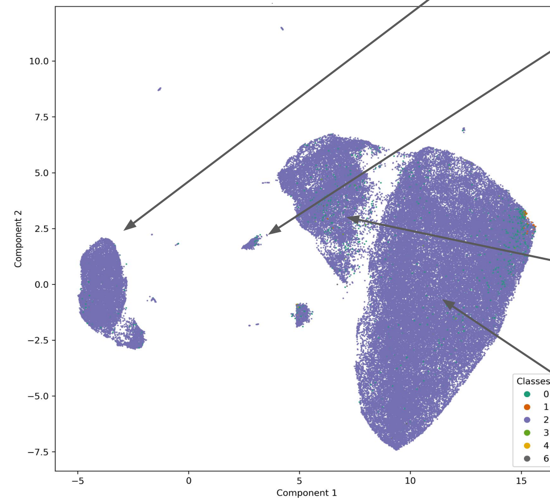


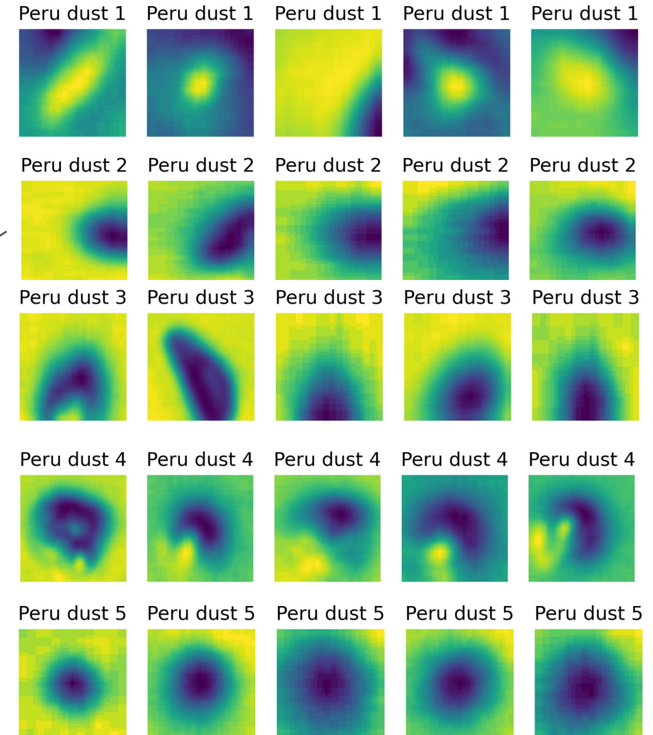


0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

A look at the dust subtypes

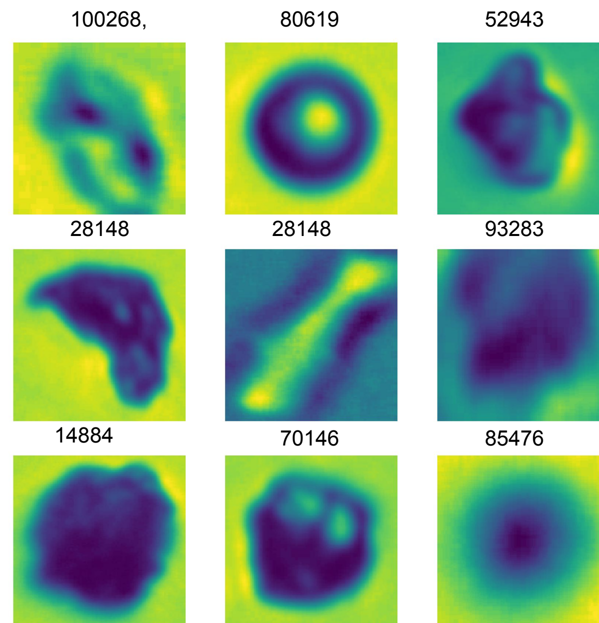
- Clusters mainly based on cropping, lighting or position in frame
  - not exciting!





Images that confused the classifier: New types?

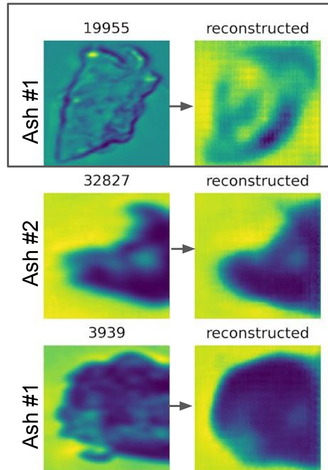
- Use median on multiclassification-scores as measure of classifier uncertainty
- Look for at images with highest median

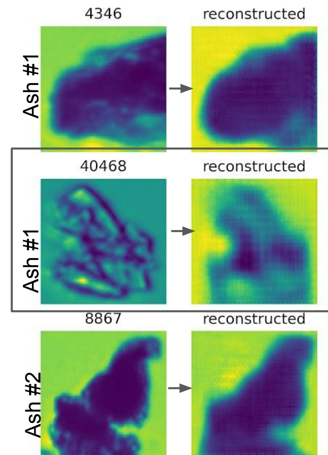


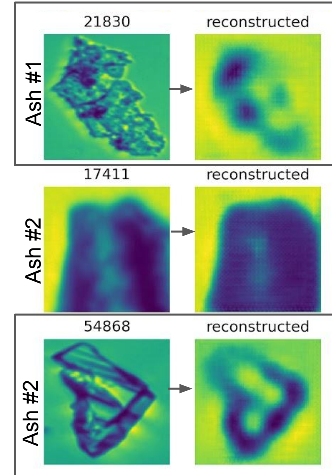
0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Best method: Look for highest loss in autoencoder

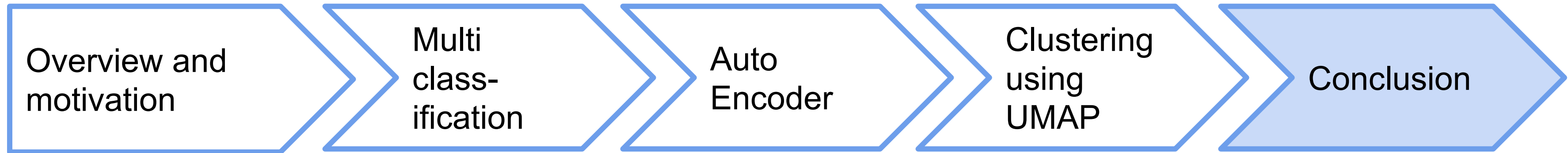
- AE trained on training data, ie. knows only the 7 training classes
- Yields our best candidates for interesting/new insolubles







Conclusion

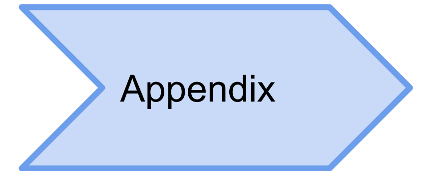


Conclusion

- Multi classification using CNN works best, disregarding the metadata
- Mainly dust in Peruvian samples, although ash is also quite prevalent. Though perhaps not the same type?
- Auto encoder is able to replicate images quite well with latent space dimension of 64.
- No apparent new classes in Peruvian samples
- But some interesting samples were found using three methods
- Given more time, we would optimize NN's further and explore the not-dust types more

Thanks for listening!

Appendix



Names of classes

0: Campanian	**1:** Corylus	**2:** Dust	**3:** Grimsvotn	**4:** Qrobur	**5:** Qsuber	**6:** Contamination
0: Ash #1	**1:** Pollen #1	**2:** Dust	**3:** Ash #2	**4:** Pollen #2	**5:** Pollen #3	**6:** Contamination

Running on Google Colab GPUs

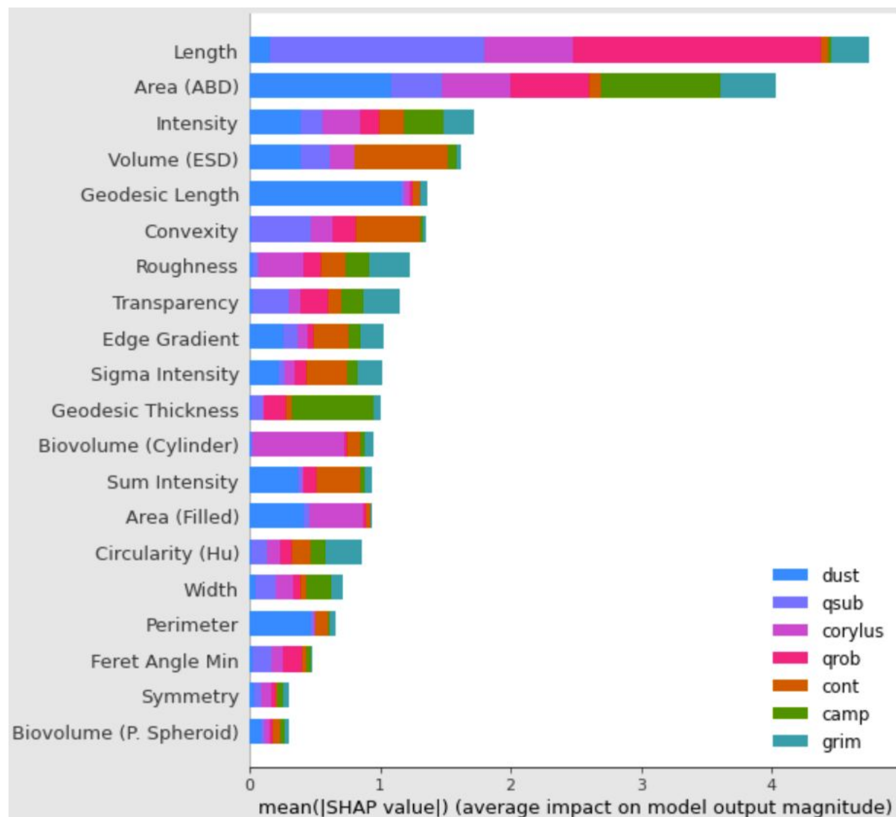
At first we had trouble getting a speedup from using google colab, since we had saved the pictures in google drive, and for each batch, we had to import them from drive, which takes so long it cancels any speedup.

So instead, we imported the zipped folders into the google colab environment instead, allowing much faster import of the images.

This is important, since we cannot import all images in the program at once, instead we just have a column in the meta data which has the location of the images, which we then import for each epoch.

In the end, running on the GPUs resulted in a large speedup.

SHAP values for metadata BDT



- By using SHAP values we gain insights into, which variables have the largest impact - useful for future experiments
- Hyper parameters (optimized):
  - eta = 0.296
  - eval_metric = 'merror'
  - max_depth = 14
  - n_estimators = 200
  - objective = 'multi:softprob'

Appendix

CNN on images

```python
class CNN_IMG(nn.Module):
    def __init__(self, pretrained=True):

        super().__init__()
        self.base = models.resnet18(pretrained=pretrained) # the CNN is based on pretrained ResNet18
        n_features = self.base.fc.in_features #512
        self.base.fc = nn.Linear(n_features, 64)
        self.bn1 = nn.BatchNorm1d(64)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.2)
        self.fc3 = nn.Linear(64, 40)
        self.bn3 = nn.BatchNorm1d(40)
        self.layer_out = nn.Linear(40, 7)
    def forward(self, imgs):
        cnn1 = self.base(imgs)
        x = self.bn1(cnn1)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        x = self.bn3(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.layer_out(x)
        #x = self.sigmoid(x)
        return x
```

- code provided by Amalie Regitze Faber Mygind

CNN on images with metadata

```
class CNN_BOTH(nn.Module):
    def __init__(self, pretrained=True):
        super().__init__()
        self.base = models.resnet18(pretrained=pretrained) # the CNN is based on pretrained ResNet18
        n_features = self.base.fc.in_features #512
        self.base.fc = nn.Linear(n_features, 64)
        self.bn1 = nn.BatchNorm1d(64)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.2)
        self.meta_net = nn.Sequential(nn.Linear(34, 128),
                                      nn.BatchNorm1d(128),
                                      nn.ReLU(),
                                      nn.Dropout(p=0.2),
                                      nn.Linear(128, 64),
                                      nn.BatchNorm1d(64),
                                      nn.ReLU(),
                                      nn.Dropout(p=0.2),
                                      nn.Linear(64, 32)
                                     )
        self.fc3 = nn.Linear(96, 40)
        self.bn3 = nn.BatchNorm1d(40)
        self.layer_out = nn.Linear(40, 7)

    def forward(self, imgs, metas):
        cnn1 = self.base(imgs)
        x = self.bn1(cnn1)
        x = self.relu(x)
        x = self.dropout(x)
        meta_ = self.meta_net(metas)
        x = torch.cat((x, meta_), 1)
        x = self.fc3(x)
        x = self.bn3(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.layer_out(x)
        #x = self.sigmoid(x)
        return x
```

- code provided by Amalie Regitze Faber Mygind

Appendix

NN on metadata

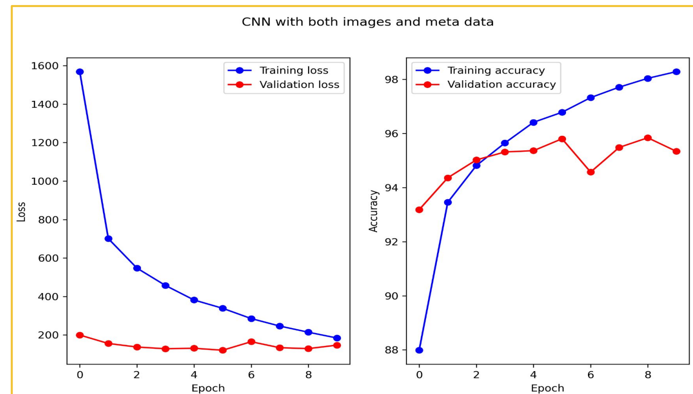
```
class NN_META(nn.Module):
    def __init__(self):
        super().__init__()
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.2)
        self.meta_net = nn.Sequential(nn.Linear(34, 128),
                                      nn.BatchNorm1d(128),
                                      nn.ReLU(),
                                      nn.Dropout(p=0.2),
                                      nn.Linear(128, 64),
                                      nn.BatchNorm1d(64),
                                      nn.ReLU(),
                                      nn.Dropout(p=0.2),
                                      nn.Linear(64, 32)
                                     )
        self.fc3 = nn.Linear(32, 40)
        self.bn3 = nn.BatchNorm1d(40)
        self.layer_out = nn.Linear(40, 7)
        self.sigmoid = nn.Sigmoid()
    def forward(self, metas):
        x = self.meta_net(metas)
        x = self.fc3(x)
        x = self.bn3(x)
        x = self.relu(x)
        x = self.dropout(x)
        x = self.layer_out(x)
        #x = self.sigmoid(x)
        return x
```

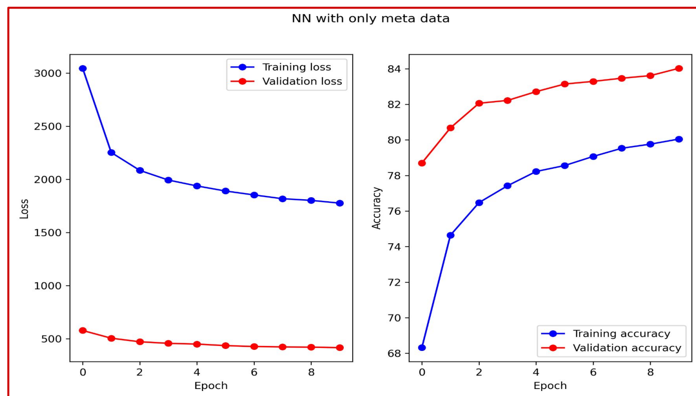
- code provided by Amalie Regitze Faber Mygind

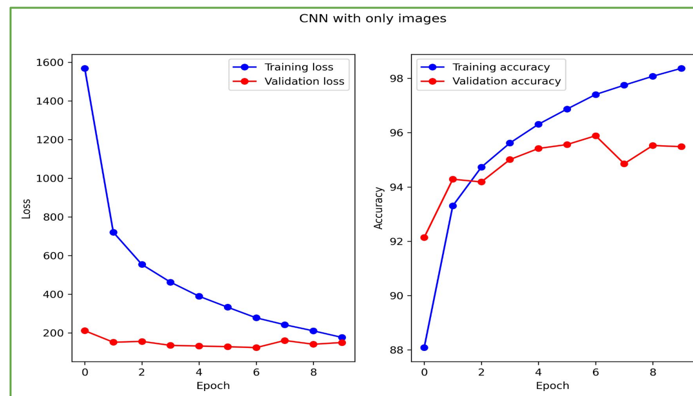
Appendix

Multi Classification - Images, metadata or both?

- Tried CNN only on pictures, in combination with metadata and a normal NN on metadata
- Most information contained in images, including metadata does not improve the classification
- Going forward, we omit metadata







Appendix

CNN output to predictions and optimization

- As mentioned the final layer of the CNN for classification outputs 7 values. The loss is evaluated by comparing these 7 values, to the label which has an integer value from 0-6, which corresponds to the 7 classes.
- To get predictions, we use nn.softmax() on the output, which converts the 7 values to “probabilities” of being the 7 classes, which add up to 1.
- The class with highest probability is then the one we predict the sample to be. One could have selected a cleaner sample of “not dust” by requiring a certain percentage of for instance 90%>, and not just selecting the highest. We did not have time to experiment with it.
- We optimized the CNN by tweaking the learning rate, but did not try many configurations of layers.

Getting the 2nd last layer of the CNN

- To get the 2nd last layer values out of the CNN, we split it into two. The first part does exactly as the original CNN, but outputs 30 values, which is then fed into a second NN to get it further down to 7 values. By optimising these NN's together, we get the same accuracy, but are able to just use the first, to get the values of the 2nd. last layer.

Autoencoder - Encoder

```python
class CNN_encoder(nn.Module): #Original name CNN_IMG
    def __init__(self, latent_space_dim,pretrained=True):
        super().__init__()
        self.encoder_conv = nn.Sequential(
            nn.Conv2d(3,8,ks_e[0],stride = strides_e[0], padding = pads_e[0]),
            nn.ReLU(True),
            nn.Conv2d(8,16,ks_e[1],stride = strides_e[1],padding = pads_e[1]),
            nn.BatchNorm2d(16),
            nn.ReLU(True),
            nn.Conv2d(16,32,ks_e[2],stride = strides_e[2], padding = pads_e[2]),
            nn.ReLU(True)
        )
        self.flatten = nn.Flatten(start_dim = 1)
        self.fc1 = nn.Linear(int(dim**2*32), 250)
        self.dropout = nn.Dropout(p=0.2)
        self.bn1 = nn.BatchNorm1d(250)
        self.relu = nn.ReLU(True)
        self.fc2 = nn.Linear(250, latent_space_dim)
        #self.sigmoid = nn.Sigmoid()
```

```python
def forward(self, imgs):
    cnn1 = self.encoder_conv(imgs)
    x = self.flatten(cnn1)
    x = self.fc1(x)
    x = self.dropout(x)
    x = self.bn1(x)
    x = self.relu(x)
    x = self.fc2(x)
    #x = self.relu(x)
    #x = self.dropout(x)
    #x = self.layer_out(x)
    #x = self.sigmoid(x)
    return x
```

Autoencoder - Decoder

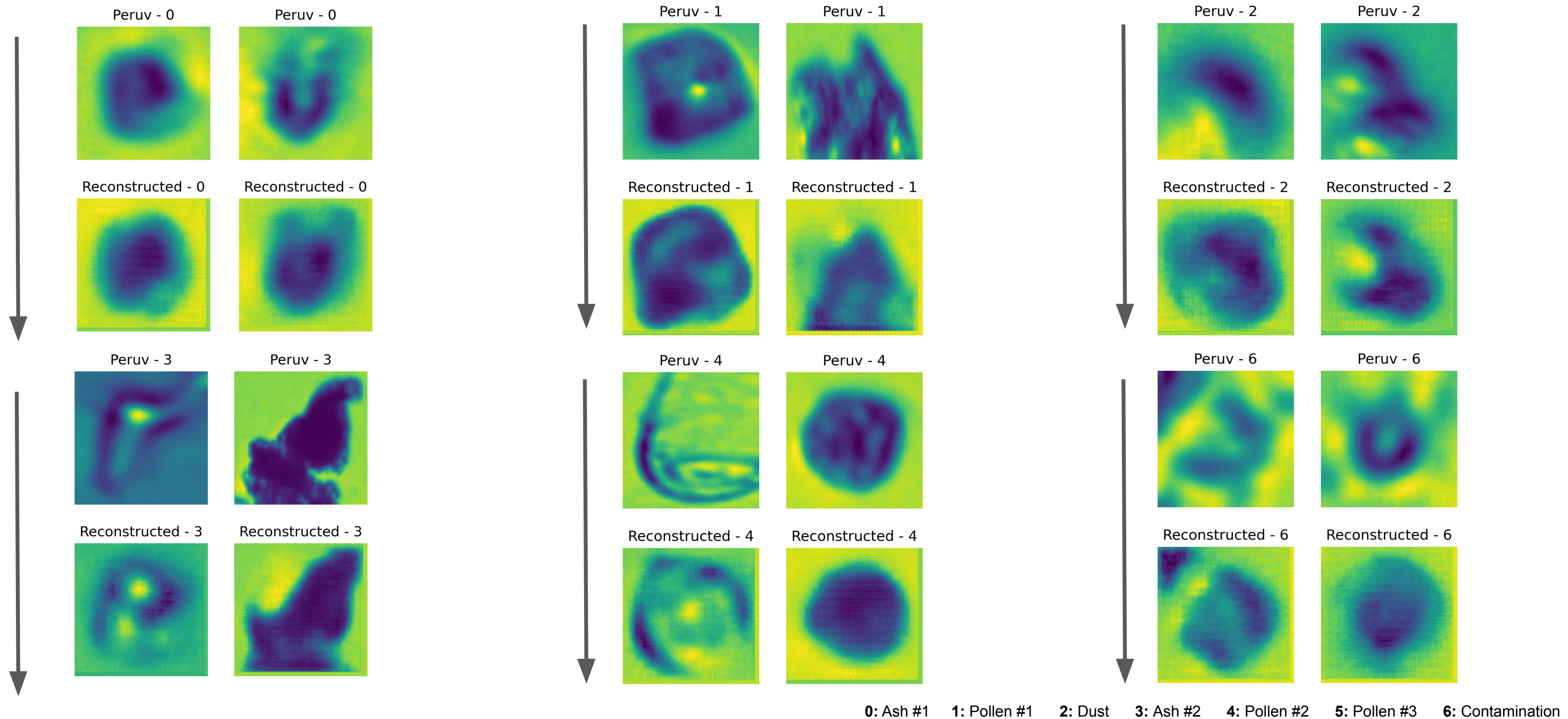
```
class CNN_decoder(nn.Module):
    def __init__(self, latent_space_dim,pretrained=True):
        super().__init__()
        #self.layer_out_rev = nn.Linear(latent_space_dim,100)
        #self.bn3_rev = nn.BatchNorm1d(100)
        self.fc2_rev = nn.Linear(latent_space_dim,250)
        self.dropout = nn.Dropout(p = 0.2)
        self.bn1_rev = nn.BatchNorm1d(250)
        self.relu = nn.ReLU(True)
        self.last_linear = nn.Linear(250,7200)
        self.unflatten = nn.Unflatten(dim=1,
            unflattened_size=(32, 15, 15))
        self.decoder_conv = nn.Sequential(
            nn.ConvTranspose2d(32, 16, ks_d[0], stride=strides_d[0], output_padding=pads_out_d[0]), #Output 21x21
            nn.BatchNorm2d(16),
            nn.ReLU(True),
            nn.ConvTranspose2d(16, 8, ks_d[1], stride=strides_d[1], padding=pads_d[1], output_padding=pads_out_d[1]), #Output 64x64
            nn.BatchNorm2d(8),
            nn.ReLU(True),
            nn.ConvTranspose2d(8, 3, ks_d[2], stride=strides_d[2], padding=pads_d[2], output_padding=pads_out_d[2]) #Output 128x128
            )
```

```
def forward(self,latent): #Latent is the picture expressed in the latent space
    #lin1 = self.layer_out_rev(latent)
    #x = self.bn3_rev(lin1)
    x = self.fc2_rev(latent)
    x = self.dropout(x)
    x = self.bn1_rev(x)
    x = self.relu(x)
    x = self.last_linear(x)#(x)
    x = self.unflatten(x)
    x = self.decoder_conv(x)
    #x = torch.sigmoid(x)
    return x
```

Optimisation of autoencoder

- We built two different versions of the autoencoder.
- One used Resnet18 in the encoder while the decoder was built manually.
- The other was built from scratch and with symmetric encoder and decoder.
- We went with the latter option, as this gave more flexibility and, after manually optimizing the architecture (eg. kernel-size=3 in all layers) significantly lower losses.
- A latent-space of 32 dimensions gave losses about 10% larger, so we stuck to 64 dimensions.

Autoencoder - Reconstructed images - Peruvian



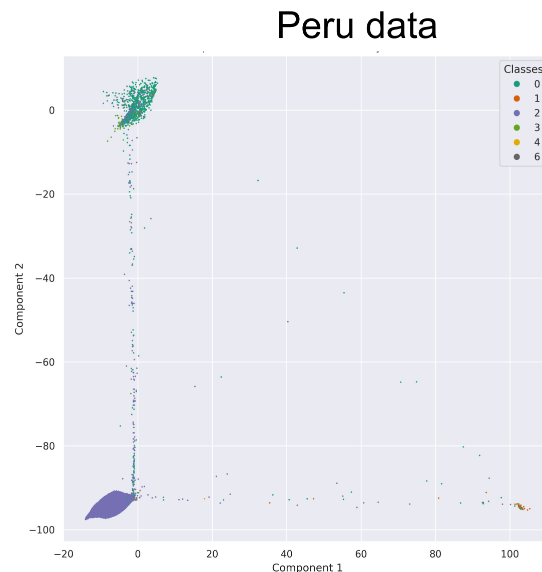
Appendix

UMAP parametrized 2nd last layer multi classification CNN

- UMAP Parametrized is NN trained to separate like UMAP, but can be applied to new data
- UMAP Parametrized shows agreement with our classification, but no new clusters in Peru

Training data

Peru data

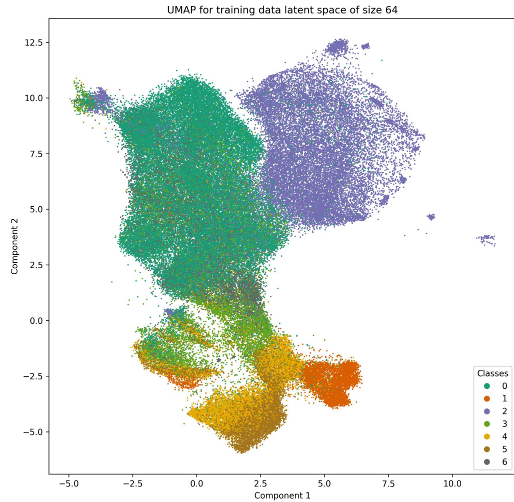


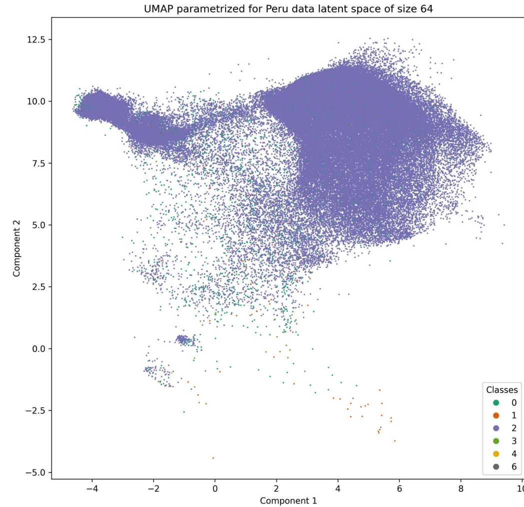
0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

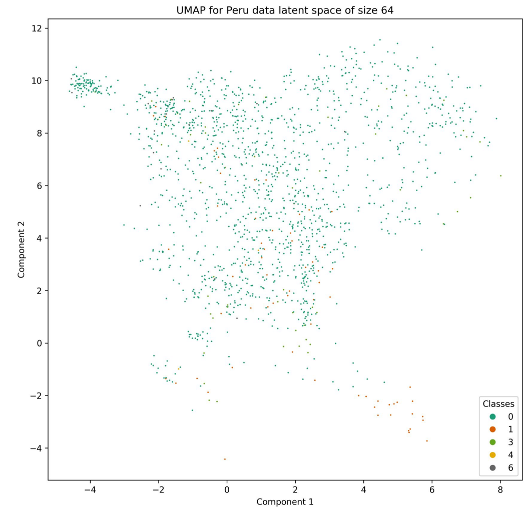
Appendix

Autoencoder - Trained on 7 classes, Parametrized UMAP

- Did not really give something new





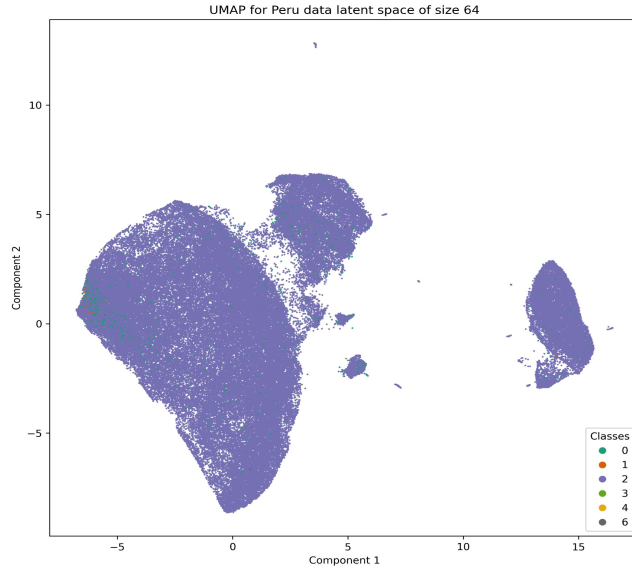


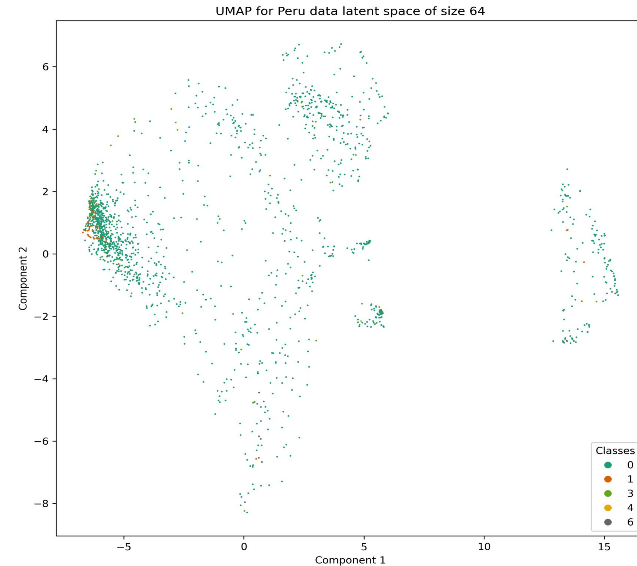
0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Appendix

Autoencoder - Training on Peru UMAP on peru

- Shows the same as when autoencoder is trained on training and Umapped on Peru!



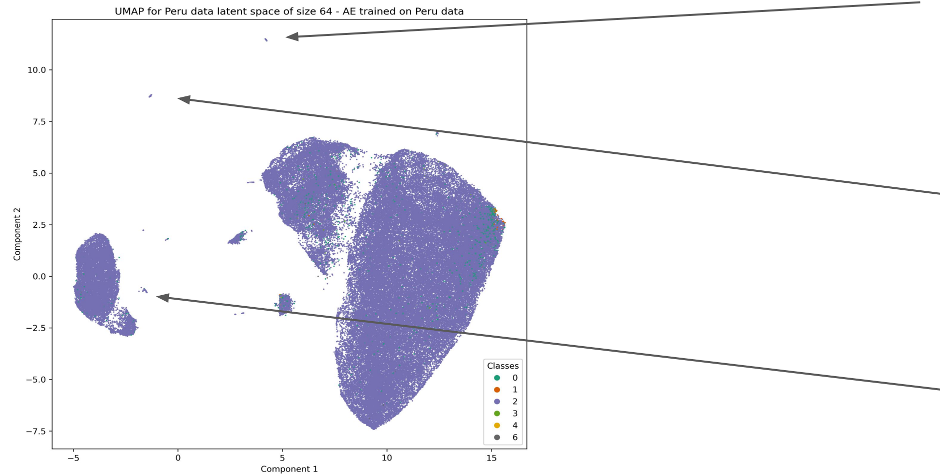


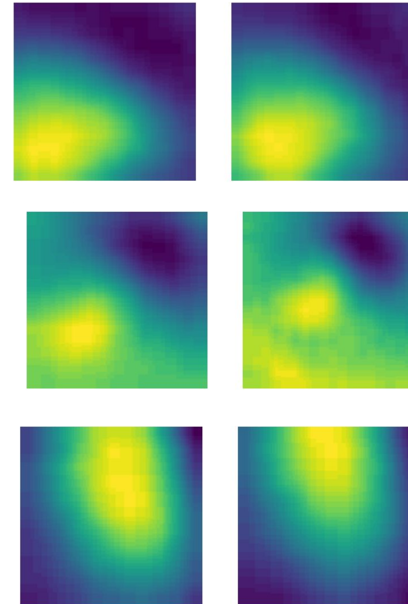
0: Ash #1 **1:** Pollen #1 **2:** Dust **3:** Ash #2 **4:** Pollen #2 **5:** Pollen #3 **6:** Contamination

Appendix

3rd way of looking for interesting/new insolubles

UMAP-outliers did not turn out to be interesting.

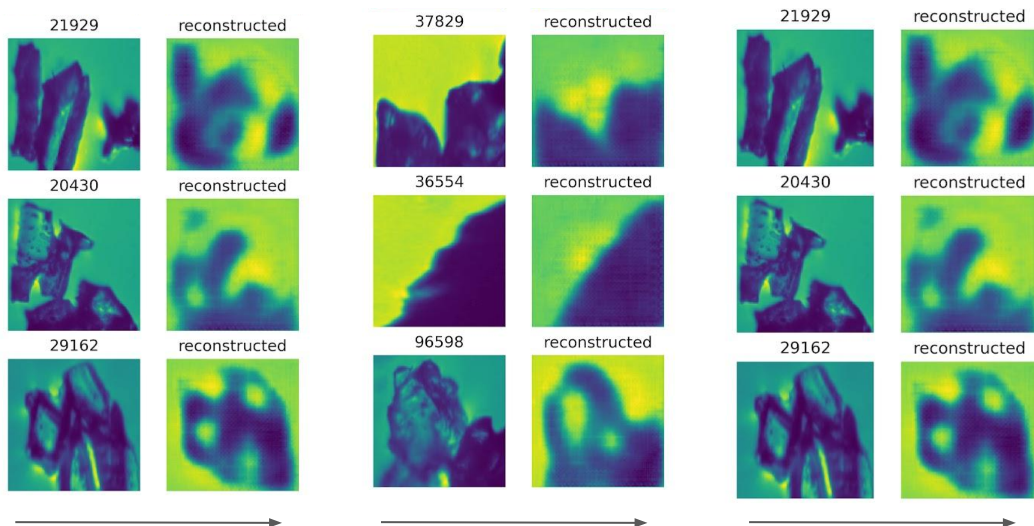




Appendix

Training images with highest loss for autoencoder

- Training images that are the hardest to replicate also have complex structures, suggesting that the most difficult to replicate from the Peru data, could just be complex, not a new type of insoluble



These are all mainly Ash#2

Appendix