Insolubles in Ice Cores

By Cecilie, Flemming, Laura, Mads and Yannick

Hypothesis and outline

By training on images and metadata from insolubles that can be found in ice cores, we want to classify different classes of insolubles from a Peruvian ice core (supervised learning).

Outline:

- Data
- Training the model
- Predict on peruvian ice core

The data

Training data:

Data from Niccolò Maffezzoli

- 7 classes with labels
- 147960 data points
- images of varying resolution
- 56 metadata
- ‘Synthetic data’

Peruvian ice core data:

- Unknown amount of classes
- 102764 data points
- images and metadata

Metadata -contains 56 only 34 is applied

'Area (ABD)', 'Area (Filled)', 'Aspect Ratio', 'Biovolume (Cylinder)',

'Biovolume (P. Spheroid)', 'Circle Fit',

'Circularity', 'Circularity (Hu)', 'Compactness', 'Convex Perimeter',

'Convexity', 'Diameter (ABD)', 'Diameter (ESD)', 'Edge Gradient',

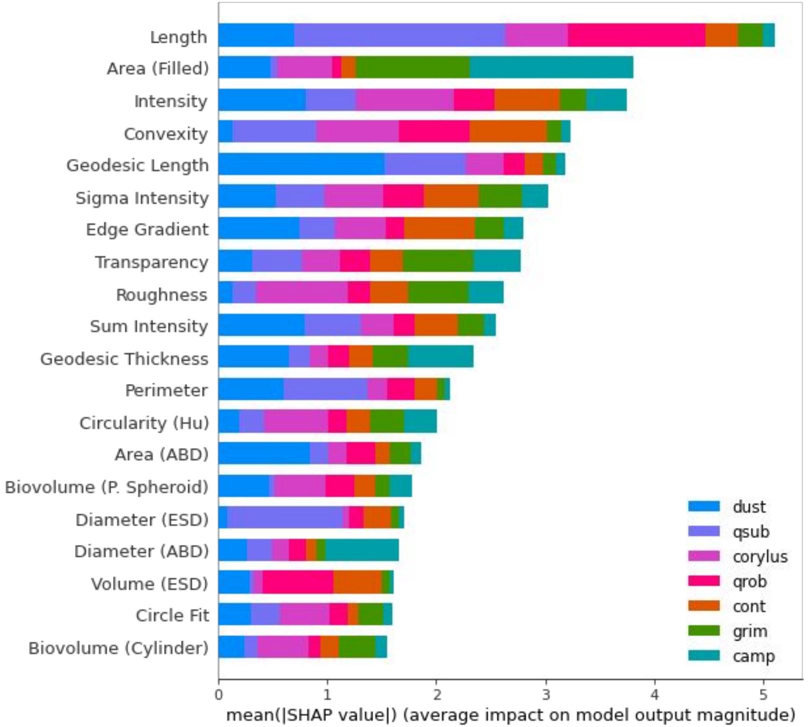
'Elongation', 'Feret Angle Max', 'Feret Angle Min', 'Fiber Curl',

'Fiber Straightness', 'Geodesic Aspect Ratio', 'Geodesic Length',

'Geodesic Thickness', 'Intensity', 'Length', 'Particles Per Chain',

'Perimeter', 'Roughness', 'Sigma Intensity', 'Sum Intensity',

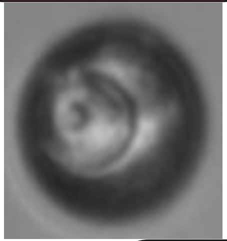
'Symmetry', 'Transparency', 'Volume (ABD)', 'Volume (ESD)', 'Width'



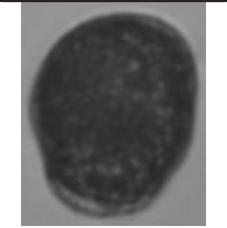
Images in training

Pollen

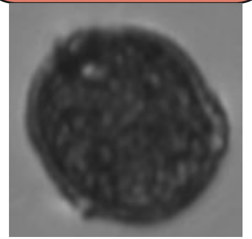
Corylus



Qsuper

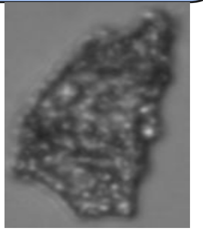


Q Robur

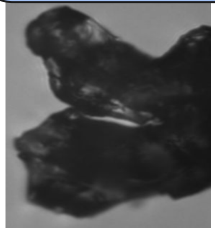


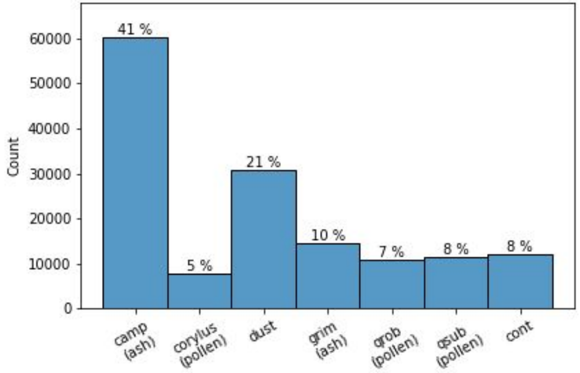
Ash

Campanien

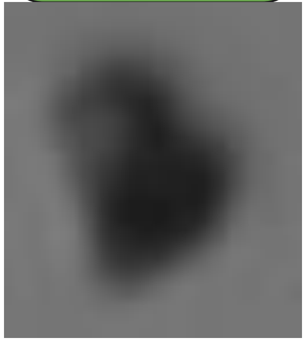


Grimsvotn

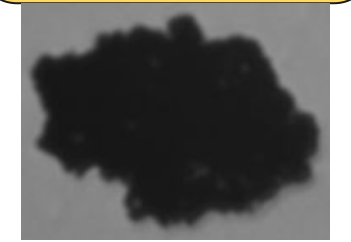




Dust



Contamination



Training the model

Preprocessing

Images:

- Reshape to 128x128 - so can be input into NNs
- Normalization - to have consistent scale
- Grayscale - to reduce size

Metadata:

- Also normalize metadata

Models and applications

Models:

- Auto encoder
- Making a NN only from metadata
- Making a CNN - without the metadata
- Combining images and metadata
  - With LightGBM
  - **NN from tenserflow**
  - Combining a CNN and NN with Resnet

Model and applications:

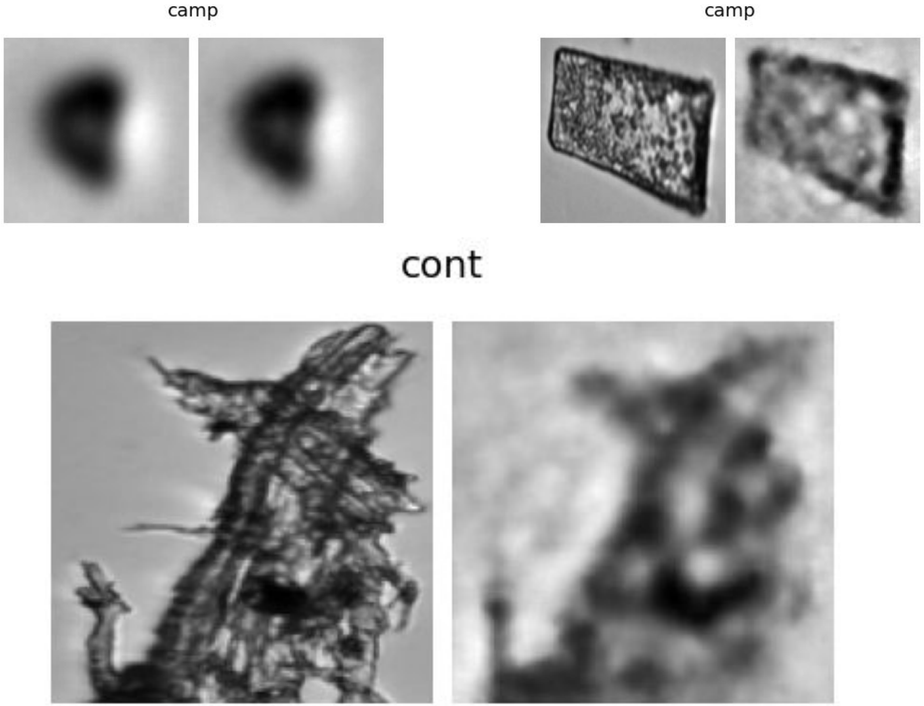
- **Shap to find best metadata**
- Train on 6 classes - umap to find 7
- **Use last hidden layer to classify**
- **Use auto encoder to remake images**
- Optimize latent space
- Does single classes have structure?
- Only look at pollen because they are hardest to classify
- **Classify into 4 classes then into 7**

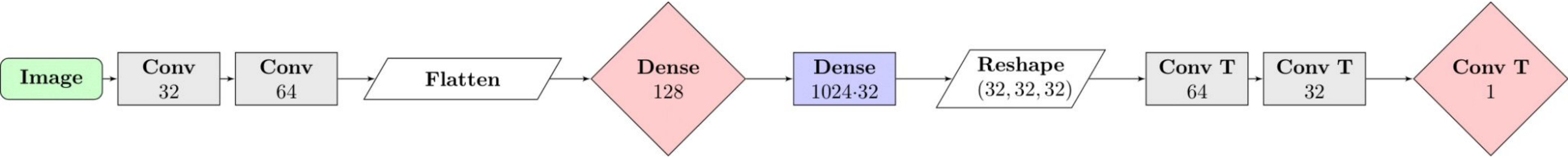
Auto encoder

Only encoder for the NN - auto encoder to validate

Optimized latent space and looked at how it improved the NNs predictions

Tried different auto encoders - more convolutions and included maxpools



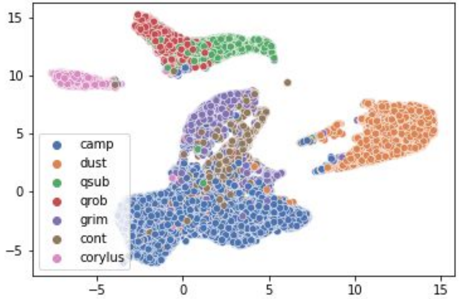


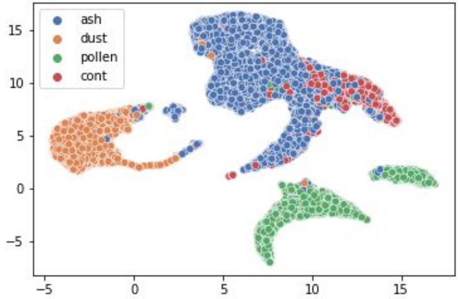
Performance and predictions

	Accuracy	Comment
LightGBM meta and encoded images	86%	Fast and fine
Only images (CNN)	81%	Bad
Only metadata (NN)	84%	Not good
Only images (NN on encoded images)	87%	Decent
Meta and encoded (NN)	90%	Much better than individual
Combined convolutional NN	86%	Difficult to train and not better than the above

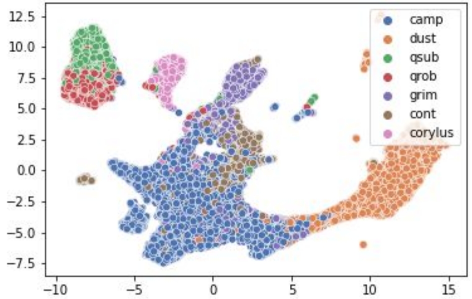
Umaps

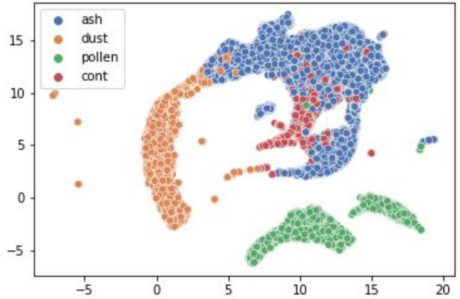
Only encoded images (NN)



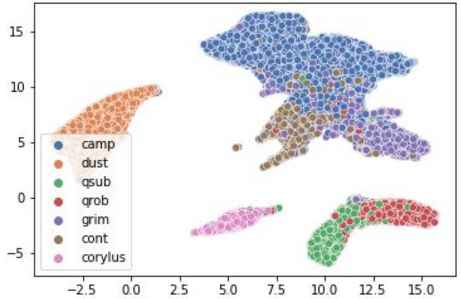


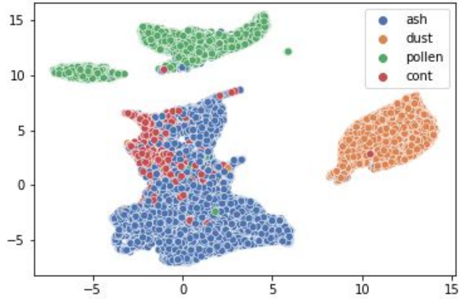
Only metadata (NN)



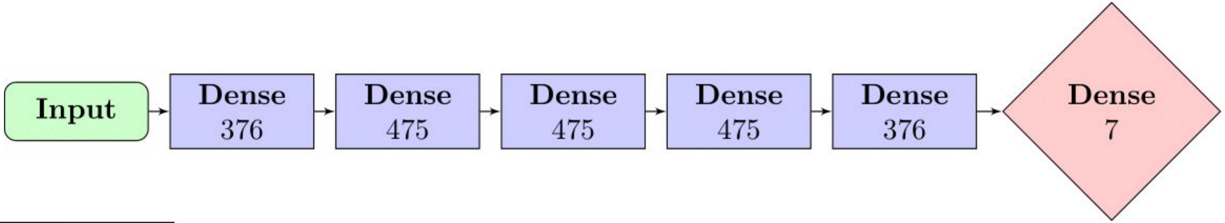


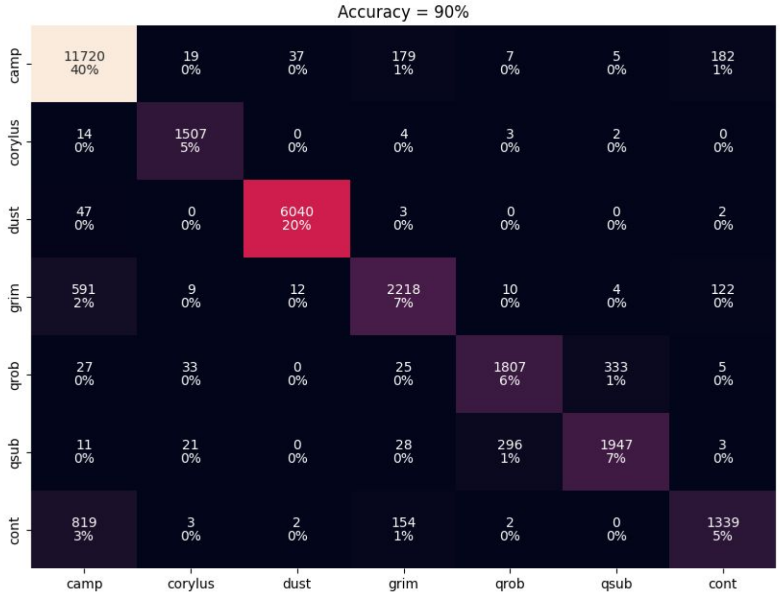
Meta and encoded (NN)



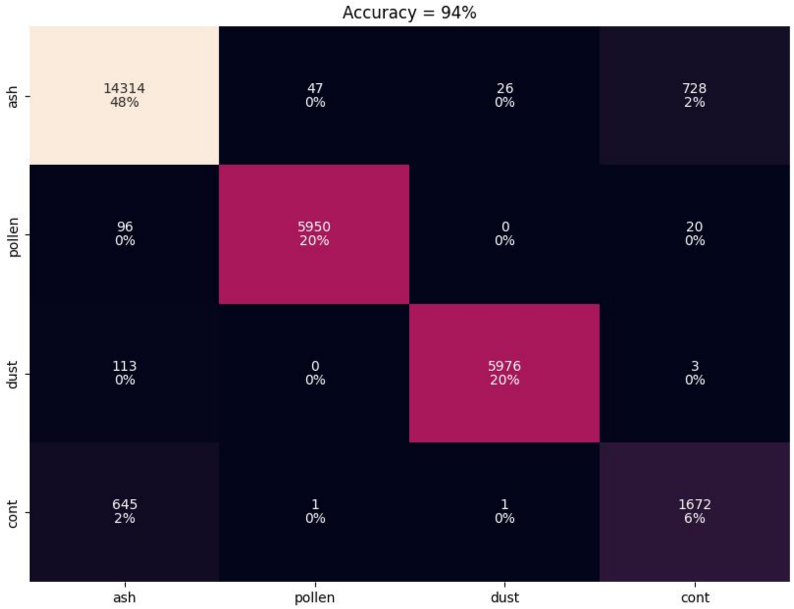


NN from encoded images and metadata





Combining classes into larger groups



Optimize

- Balancing of data types
  - smote, random undersampling and random oversampling
- Baysianoptimazation

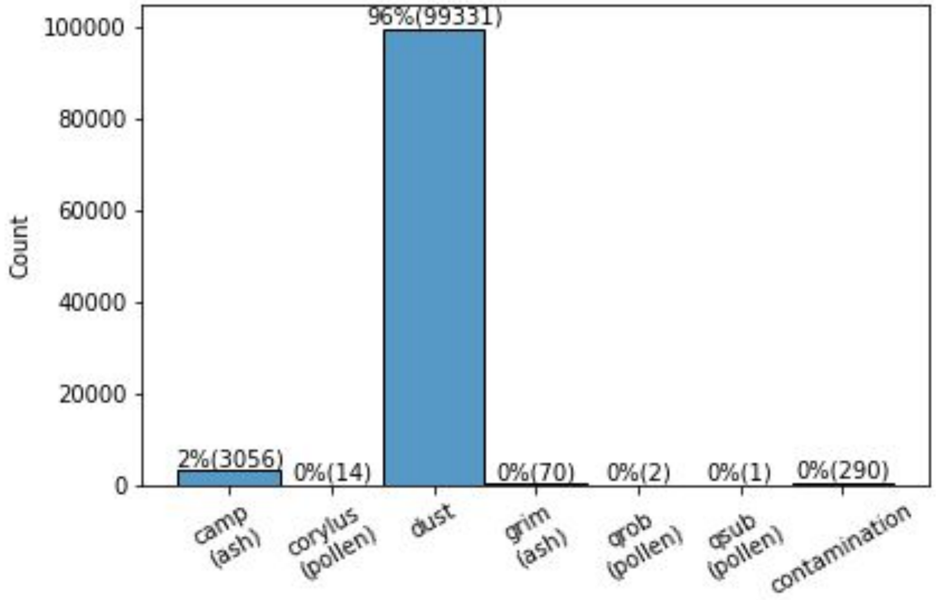
Troubles with training

- Models that took images as input were troublesome in google colab.
- Limited GPU time on colab.

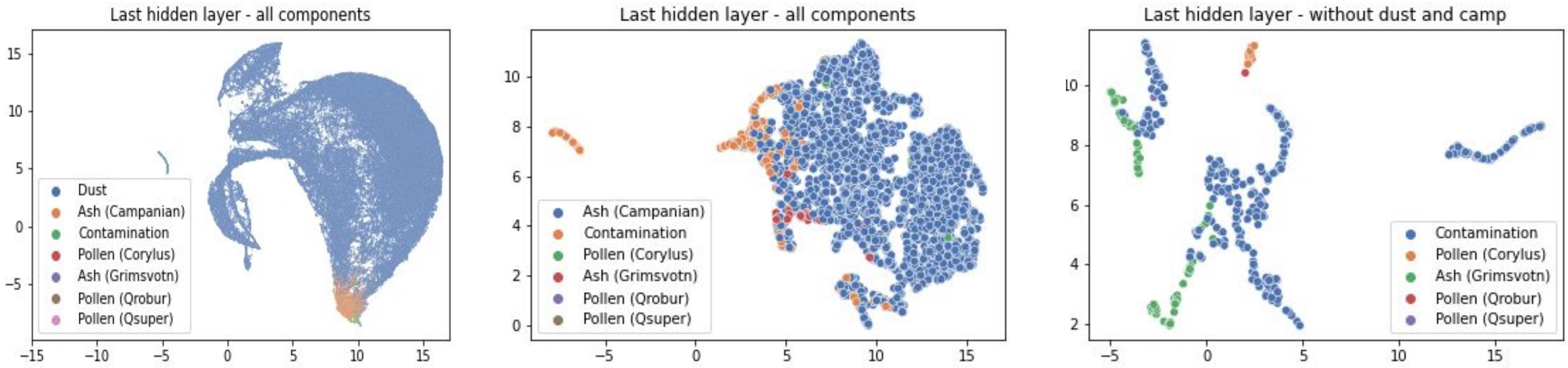
PERUVIAN ICE CORE

Predictions from test and train

Found a lot of dust and a bit of ash and not so much pollen

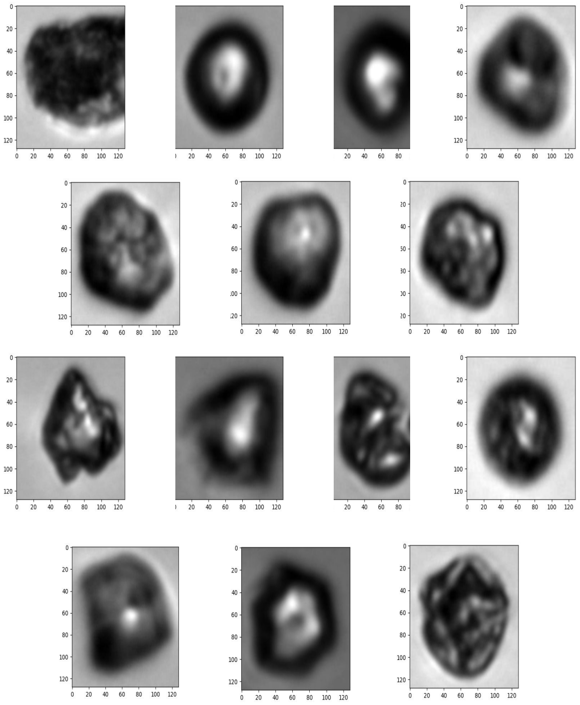


Umap on last hidden layer, colored by predictions from NN

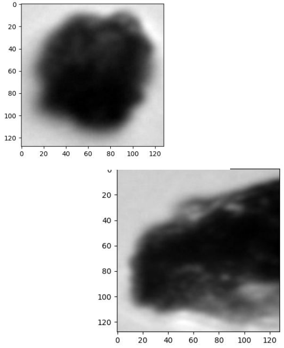


20 interesting images

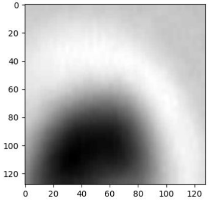
Class 1 - corylus



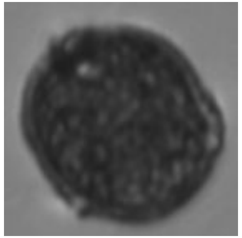
Class 4 - qrobur



Class 5 - qsuber

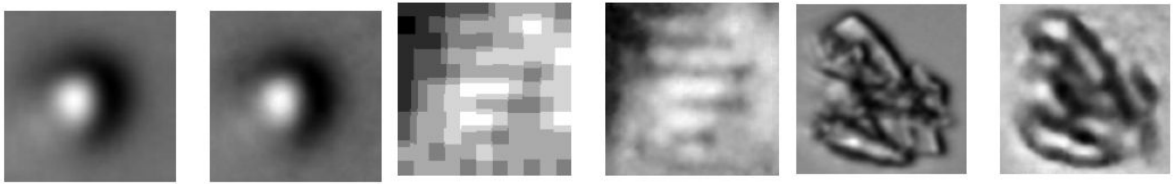


True pollen



From training set - to compare to pollen

Encoder best/worst:



Conclusions and perspective

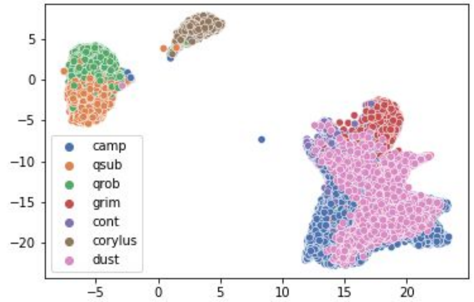
- Our model is pretty good at separating major groups (pollen, ash, dust, and contamination)
- Difficult to separate inside these groups
- There is mostly dust in peruvian ice cores, little ash and not much pollen at all

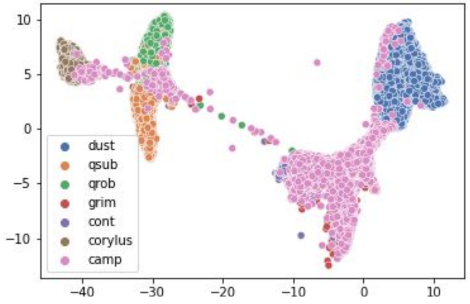
- Perspective:
  - optimize our models
  - find a way to do better training

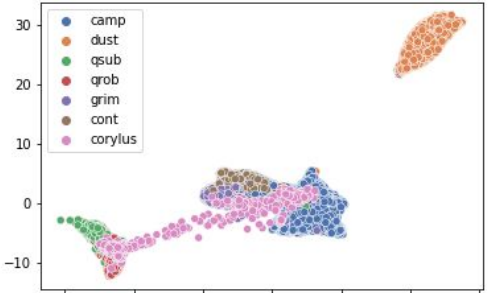
Appendix

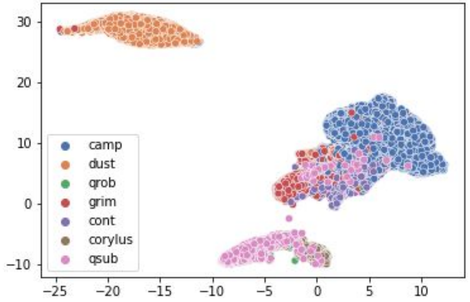
train on 6 umap on 7

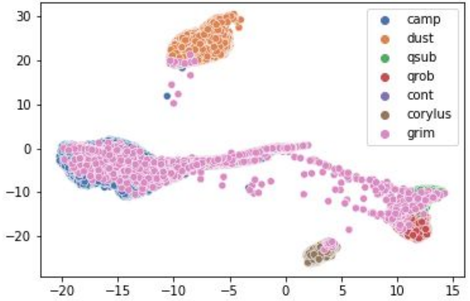
Idea is to only train with 6 of the labels. Then use umap on the last layer of NN to see if the 7th makes a distinct group.

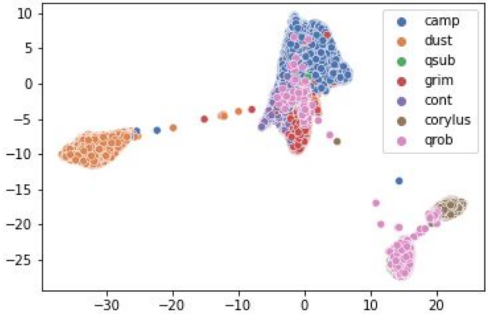


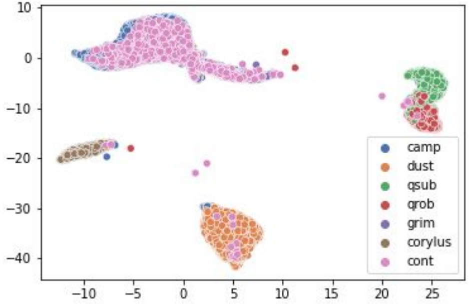










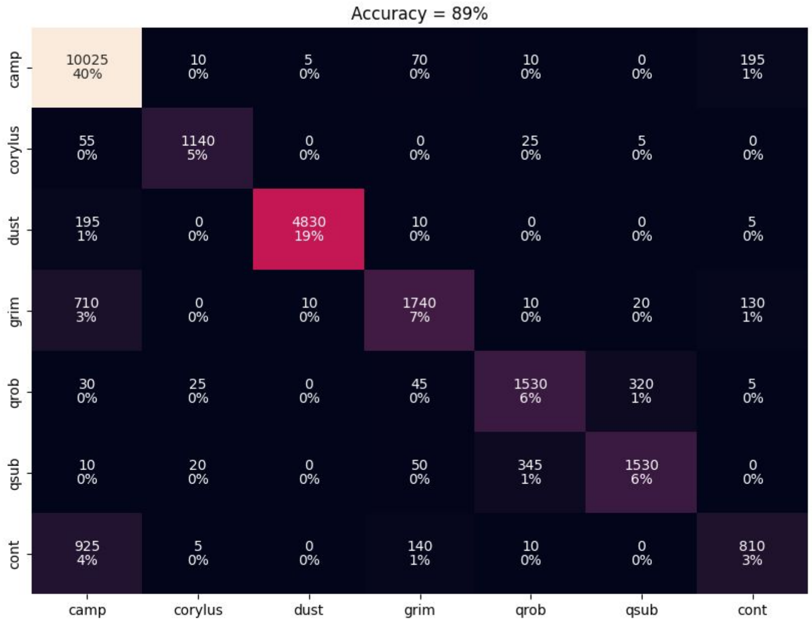


Dedicated NN to see differences between groups

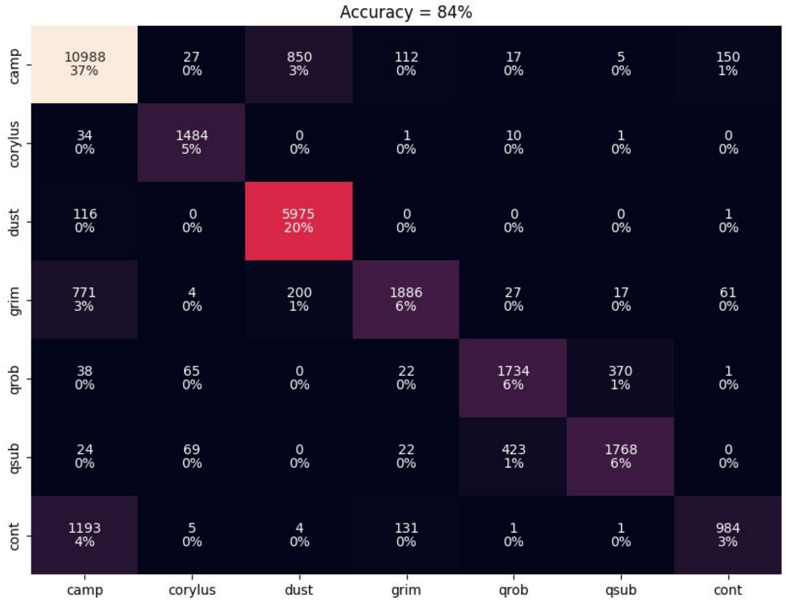
We also attempted to make dedicated NN to see the differences between the groups it at a bad time at seperating.

No increase in performance was made.

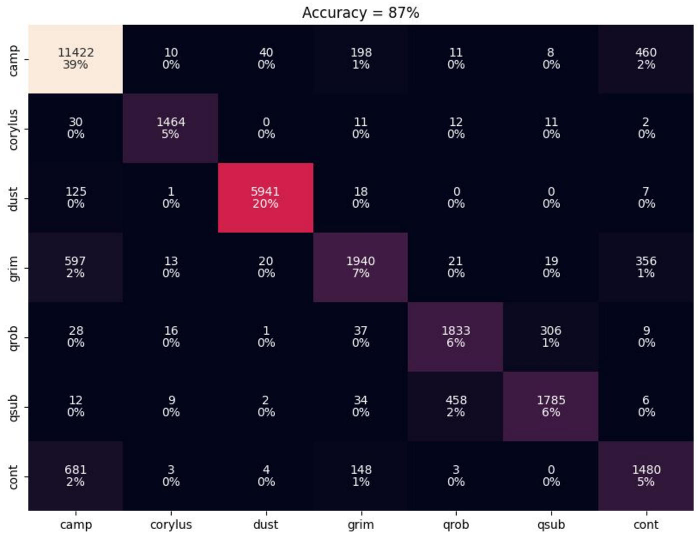
LightGBM


Accuracy = 89%
camp
corylus
dust
grim
qrob
qsub
cont
10025 40%
10 0%
5 0%
70 0%
10 0%
0 0%
195 1%
55 0%
1140 5%
0 0%
0 0%
25 0%
5 0%
0 0%
195 1%
0 0%
4830 19%
10 0%
0 0%
0 0%
5 0%
710 3%
0 0%
10 0%
1740 7%
10 0%
20 0%
130 1%
30 0%
25 0%
0 0%
45 0%
1530 6%
320 1%
5 0%
10 0%
20 0%
0 0%
50 0%
345 1%
1530 6%
0 0%
925 4%
5 0%
0 0%
140 1%
10 0%
0 0%
810 3%

Only metadata

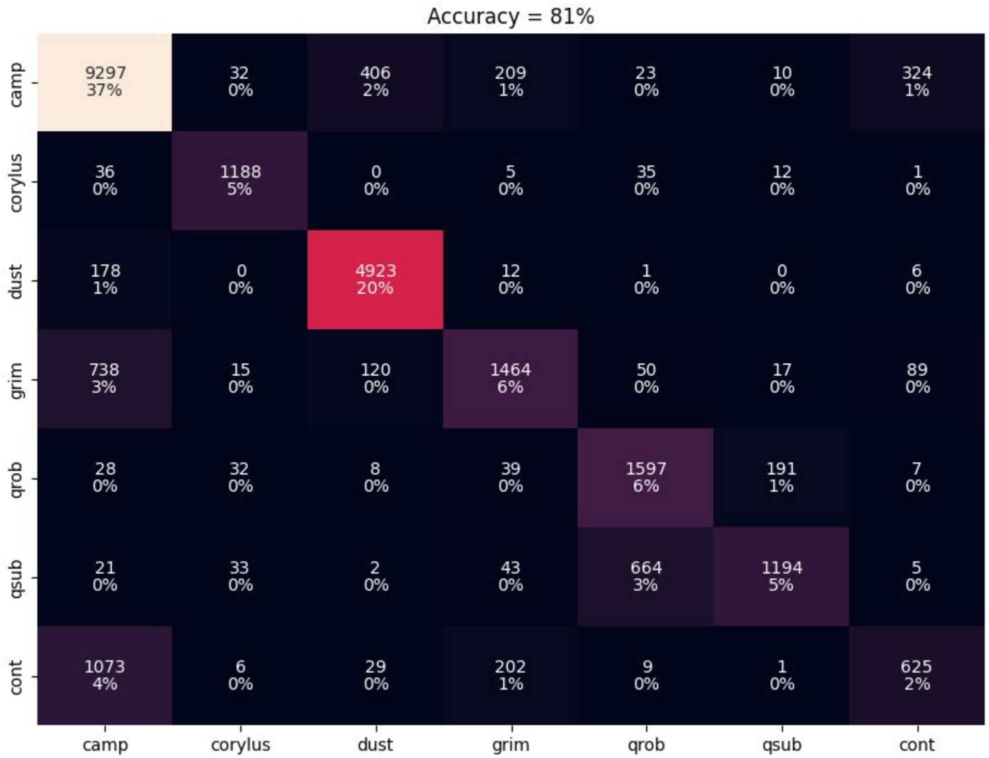


Only images



Accuracy = 87%
camp
corylus
dust
grim
qrob
qsub
cont
11422 39%
10 0%
40 0%
198 1%
11 0%
8 0%
460 2%
30 0%
1464 5%
0 0%
11 0%
12 0%
11 0%
2 0%
125 0%
1 0%
5941 20%
18 0%
0 0%
0 0%
7 0%
597 2%
13 0%
20 0%
1940 7%
21 0%
19 0%
356 1%
28 0%
16 0%
1 0%
37 0%
1833 6%
306 1%
9 0%
12 0%
9 0%
2 0%
34 0%
458 2%
1785 6%
6 0%
681 2%
3 0%
4 0%
148 1%
3 0%
0 0%
1480 5%
camp
corylus
dust
grim
qrob
qsub
cont

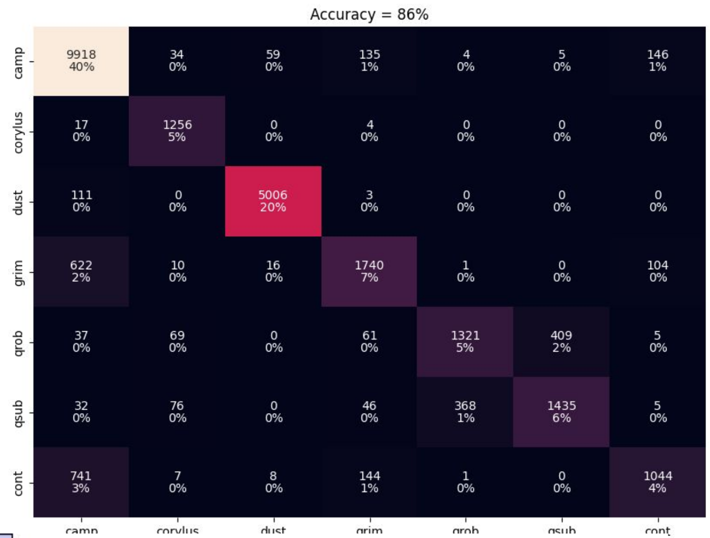
CNN

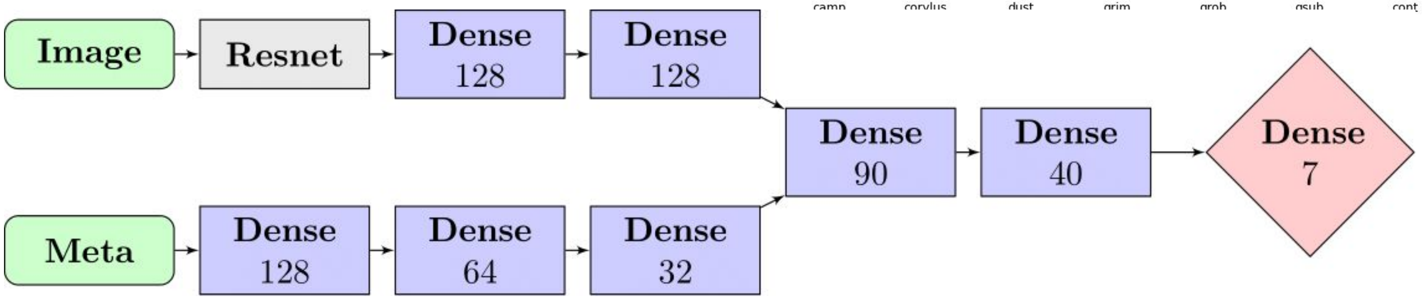

Accuracy = 81%
camp
corylus
dust
grim
qrob
qsub
cont
9297 37%
32 0%
406 2%
209 1%
23 0%
10 0%
324 1%
36 0%
1188 5%
0 0%
5 0%
35 0%
12 0%
1 0%
178 1%
0 0%
4923 20%
12 0%
1 0%
0 0%
6 0%
738 3%
15 0%
120 0%
1464 6%
50 0%
17 0%
89 0%
28 0%
32 0%
8 0%
39 0%
1597 6%
191 1%
7 0%
21 0%
33 0%
2 0%
43 0%
664 3%
1194 5%
5 0%
1073 4%
6 0%
29 0%
202 1%
9 0%
1 0%
625 2%
camp
corylus
dust
grim
qrob
qsub
cont

CNN

```
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 128, 128, 1)]     0

 conv2d_7 (Conv2D)           (None, 63, 63, 64)        640

 dropout_16 (Dropout)        (None, 63, 63, 64)        0

 batch_normalization_16 (Bat  (None, 63, 63, 64)       256
 chNormalization)

 conv2d_8 (Conv2D)           (None, 31, 31, 128)       73856

 dropout_17 (Dropout)        (None, 31, 31, 128)       0

 batch_normalization_17 (Bat  (None, 31, 31, 128)      512
 chNormalization)

 flatten_3 (Flatten)         (None, 123008)            0

 dense_12 (Dense)            (None, 256)               31490304

 dropout_18 (Dropout)        (None, 256)               0

 batch_normalization_18 (Bat  (None, 256)              1024
 chNormalization)

 dense_13 (Dense)            (None, 128)               32896

 dropout_19 (Dropout)        (None, 128)               0

 batch_normalization_19 (Bat  (None, 128)              512
 chNormalization)

 dense_14 (Dense)            (None, 7)                 903

=================================================================
Total params: 31,600,903
Trainable params: 31,599,751
Non-trainable params: 1,152
```

Combined convolutional





Combined convolutional

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 128, 128, 3)]	0	[]
resnet50 (Functional)	(None, 2048)	23587712	['input_1[0][0]']
input_3 (InputLayer)	[(None, 34)]	0	[]
flatten (Flatten)	(None, 2048)	0	['resnet50[0][0]']
dense_2 (Dense)	(None, 128)	4480	['input_3[0][0]']
dropout (Dropout)	(None, 2048)	0	['flatten[0][0]']
dropout_2 (Dropout)	(None, 128)	0	['dense_2[0][0]']
batch_normalization (BatchNormalization)	(None, 2048)	8192	['dropout[0][0]']
batch_normalization_2 (BatchNormalization)	(None, 128)	512	['dropout_2[0][0]']
dense (Dense)	(None, 128)	262272	['batch_normalization[0][0]']
dense_3 (Dense)	(None, 128)	16512	['batch_normalization_2[0][0]']
dropout_1 (Dropout)	(None, 128)	0	['dense[0][0]']
dropout_3 (Dropout)	(None, 128)	0	['dense_3[0][0]']
batch_normalization_1 (BatchNormalization)	(None, 128)	512	['dropout_1[0][0]']
batch_normalization_3 (BatchNormalization)	(None, 128)	512	['dropout_3[0][0]']
dense_1 (Dense)	(None, 128)	16512	['batch_normalization_1[0][0]']
dense_4 (Dense)	(None, 32)	4128	['batch_normalization_3[0][0]']
concatenate (Concatenate)	(None, 160)	0	['dense_1[0][0]', 'dense_4[0][0]']
dense_5 (Dense)	(None, 128)	20608	['concatenate[0][0]']
dropout_4 (Dropout)	(None, 128)	0	['dense_5[0][0]']
batch_normalization_4 (BatchNormalization)	(None, 128)	512	['dropout_4[0][0]']
dense_6 (Dense)	(None, 64)	8256	['batch_normalization_4[0][0]']
dropout_5 (Dropout)	(None, 64)	0	['dense_6[0][0]']
batch_normalization_5 (BatchNormalization)	(None, 64)	256	['dropout_5[0][0]']
dropout_6 (Dropout)	(None, 64)	0	['batch_normalization_5[0][0]']
batch_normalization_6 (BatchNormalization)	(None, 64)	256	['dropout_6[0][0]']
dense_7 (Dense)	(None, 7)	455	['batch_normalization_6[0][0]']

Total params: 23,931,687
Trainable params: 23,873,191
Non-trainable params: 58,496

Substructures of classes in Peruvian