Classification of data from Charge Coupled Devices

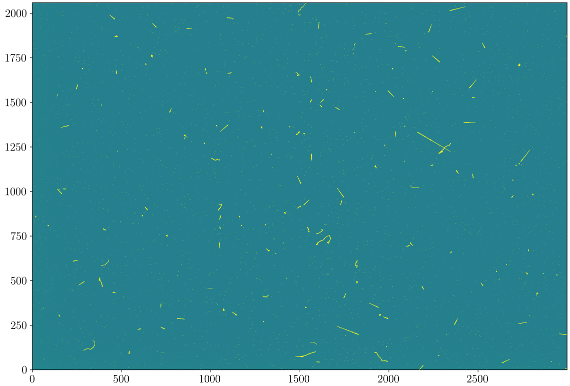
Asbjørn Bonefeld Preuss
Daniel Lomholt Christensen
Jens Kinch
Sejr Sebastian Haahr Bergmann

KØBENHAVNS UNIVERSITET

Outline

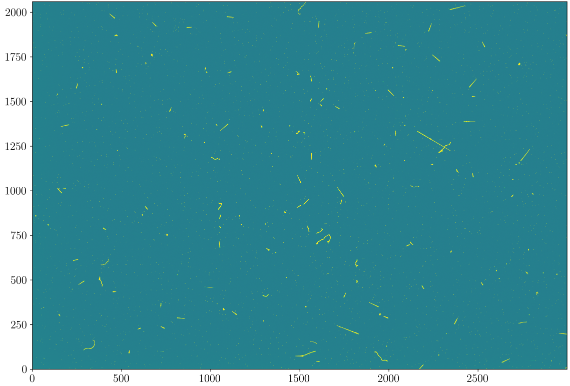
Data

Data are energy pictures, in 2D showing traces of particles from space:



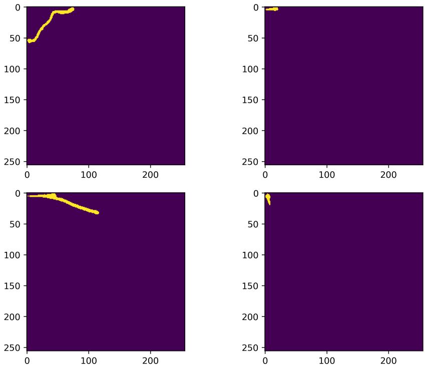
Data

Data are energy pictures, in 2D showing traces of particles from space: Project Purpose: To classify the particles faster than humans can, and possibly classify more difficult particles than we as humans can.



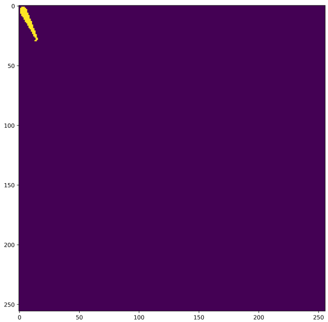
Data

We can perform binary clustering and padding, to get all clusters as binary 256 x 256 images:



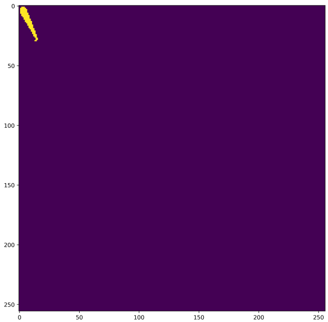
Thoughts on data processing:

- Images are quite large (256 X 256 pixels) where most images are nothing



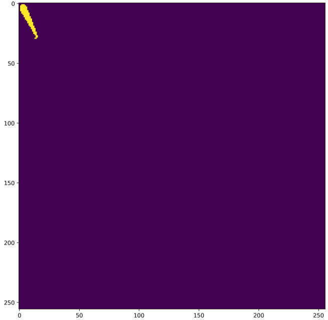
Thoughts on data processing:

- Images are quite large (256 X 256 pixels) where most images are nothing
- Images are binary



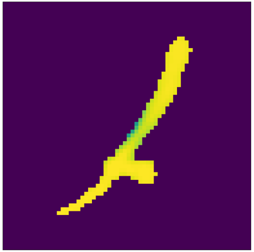
Thoughts on data processing:

- Images are quite large (256 X 256 pixels) where most images are nothing
- Images are binary
- The labels we can give, are (perhaps) quite poor



Thoughts on data processing:

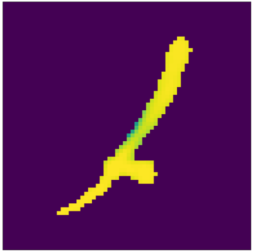
- Images are quite large (256 X 256 pixels) where most images are nothing
- Images are binary
- The labels we can give, are (perhaps) quite poor



Solutions:

Thoughts on data processing:

- Images are quite large (256 X 256 pixels) where most images are nothing
- Images are binary
- The labels we can give, are (perhaps) quite poor



Solutions:

- Downscale the large images to 64 X 64 pixels[a], pad the small images up to 64 X 64
- Re-insert energy
- Use self-supervised learning?

[a] See appendix for explanation of downscaling

Approaches

Supervised approach

- Use CNN to classify the data

Self-Supervised approach

Approaches

Supervised approach

- Use CNN to classify the data
- This requires labels. These are made manually by us ($\approx$500)

Self-Supervised approach

Approaches

Supervised approach

- Use CNN to classify the data
- This requires labels. These are made manually by us ($\approx$500)
- Will have a hard time classifying better than us.

Self-Supervised approach

Approaches

Supervised approach

- Use CNN to classify the data
- This requires labels. These are made manually by us ($\approx$500)
- Will have a hard time classifying better than us.

Self-Supervised approach

- Use variational auto-encoder and clustering on latent space

Approaches

Supervised approach

- Use CNN to classify the data
- This requires labels. These are made manually by us ($\approx$500)
- Will have a hard time classifying better than us.

Self-Supervised approach

- Use variational auto-encoder and clustering on latent space
- Could show us more than we can predict

Approaches

Supervised approach

- Use CNN to classify the data
- This requires labels. These are made manually by us ($\approx$500)
- Will have a hard time classifying better than us.

Self-Supervised approach

- Use variational auto-encoder and clustering on latent space
- Could show us more than we can predict
- Does not need labels

Approaches

Supervised approach

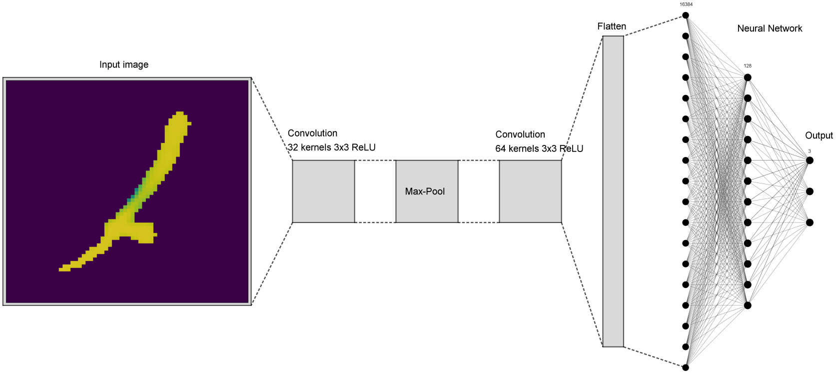
- Use CNN to classify the data
- This requires labels. These are made manually by us ($\approx$500)
- Will have a hard time classifying better than us.

Self-Supervised approach

- Use variational auto-encoder and clustering on latent space
- Could show us more than we can predict
- Does not need labels
- Experimental $\Leftrightarrow$ Might not work

Supervised classification

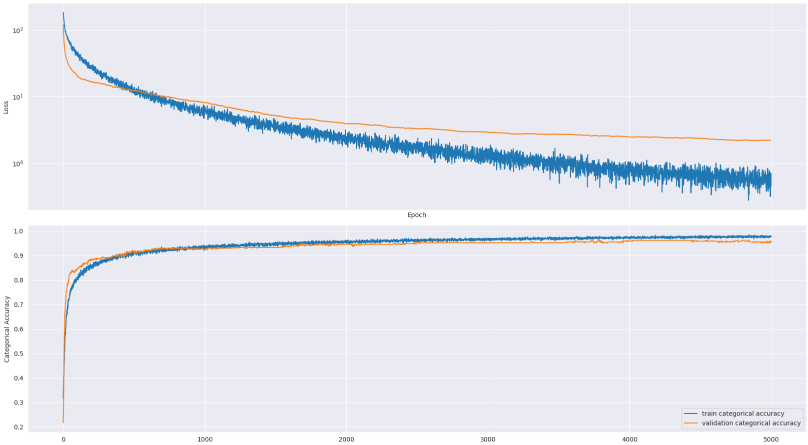
Use Convolutional Neural Network, with a categorical cross entropy loss function.
Quick run down of CNNs:



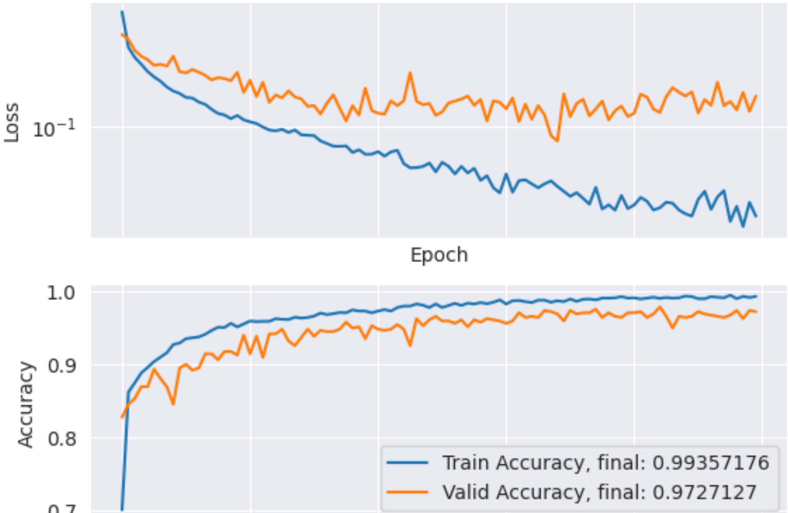
Our CNN

- Built in Tensorflow
- Hyper parameters optimized with optuna (Bayesian optimization)

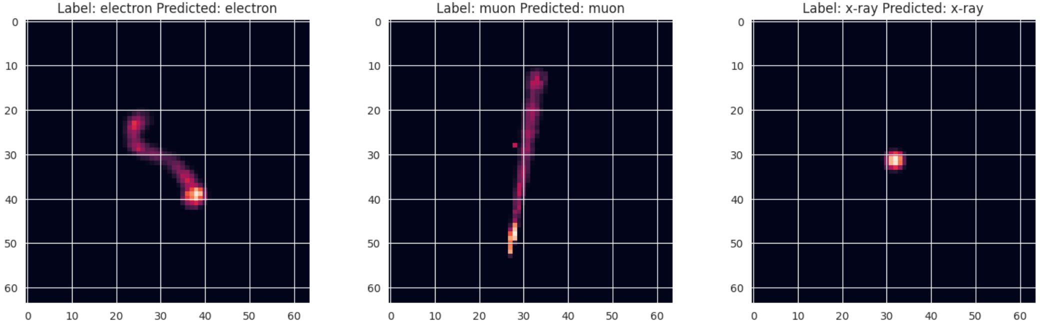
Results from Supervised Classification



Results from Supervised Classification



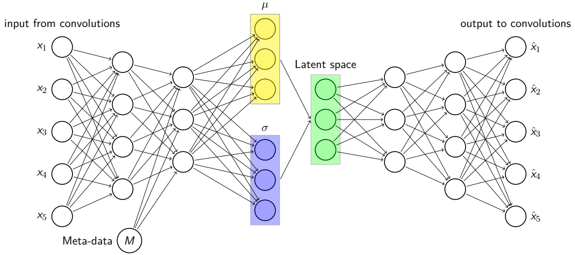
Results from Supervised Classification



Self-supervised classification

Use variational auto-encoder via a convolutional neural network and finally umap cluster on the latent space:

- Parameters for convolutions initially based on promising results from the CNN
- Meta data (Width, Height, Number of pixels, sum of those pixels) fed into the second layer of the neural network



[1]

[1] adapted from: `https://tikz.net/vae/`

Variational layers

Pseudo-code:

- Encode data to latent space
- Pass latent space to linear layer that returns array of means
- pass latent space to linear layer that array of variances
- return an array of means with Gaussian noise based on variances

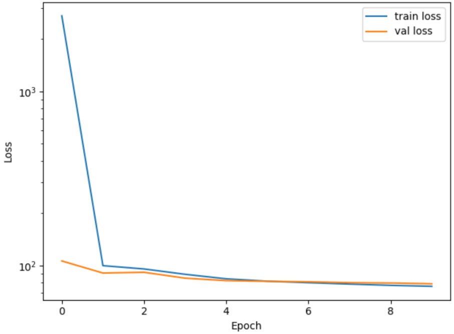
Code (PyTorch):

```
    ### Variational part
    self.variational_mean = nn.Linear(encoded_space_dim, encoded_space_dim)
    self.variational_var = nn.Linear(encoded_space_dim, encoded_space_dim)

def reparameterization(self, mean, var):  # Stolen from MNIST example, device
    epsilon = torch.randn_like(var).cpu()
    z = mean + var * epsilon
    return z

def forward(self, x):
    #CNN and removal of meta data:
    meta = x[-4:]
    x = x[:-4].unflatten(0, (1, 1, 64, 64))
    x, indices = self.encoder_cnn(x)  # Capture indices from MaxPool2d
    x = self.flatten(x)
    x = self.encoder_lin_pic(x)

    #Add meta data and encode:
    y = torch.cat((x, meta), dim=0)
    x = self.encoder_lin_w_meta(y)
    mean = self.variational_mean(x)
    log_var = self.variational_var(x)
    z = self.reparameterization(mean, torch.exp(0.5 * log_var))
    return z, mean, log_var, indices
```

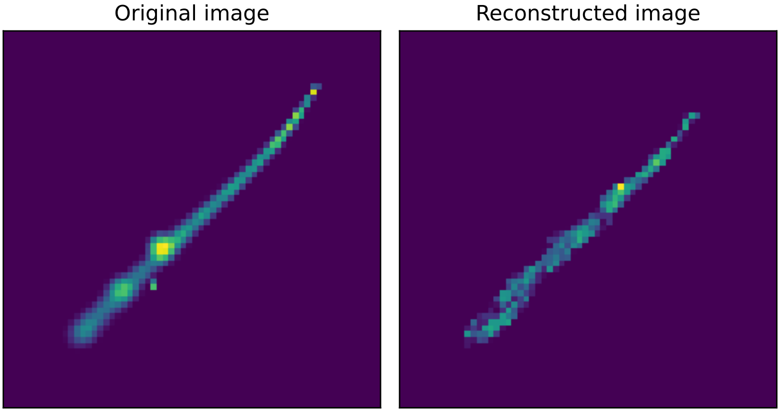
Loss

$$\mathcal{L}_{\mathsf{VAE}} = \mathcal{L}_{\mathsf{BCE}} + \lambda \mathcal{L}_{\mathsf{KL}}, \qquad \mathcal{L}_{\mathsf{KL}} = -\frac{1}{2}\sum_{n}\left(1 + \log \sigma_n^2 - \mu_n^2 - \sigma_n^2\right)$$

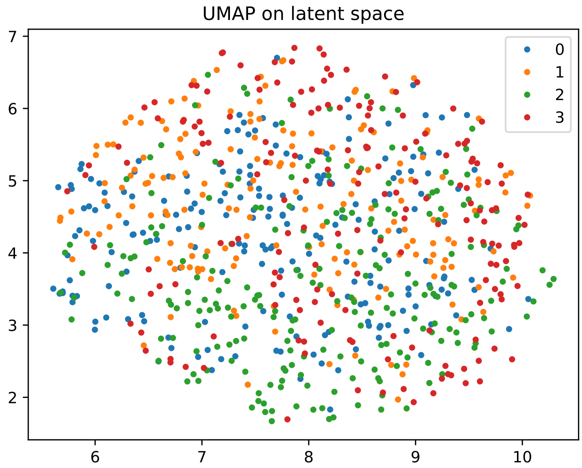
second term is the Kullback-Leibler (KL) divergence, that enforces a Gaussian latent space



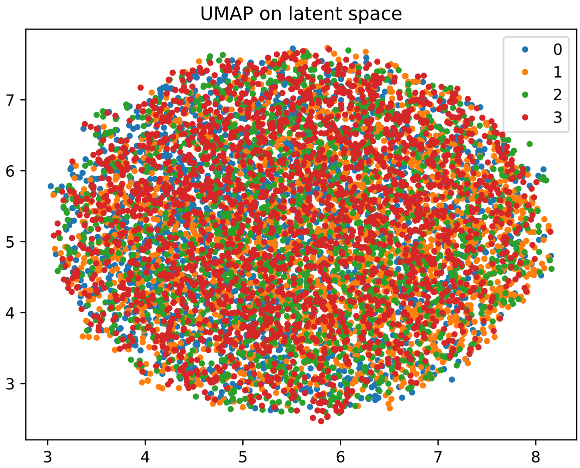
Results from our variational auto-encoder



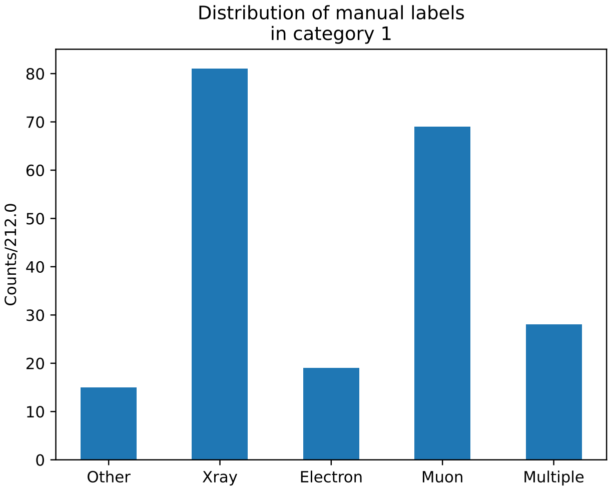
Dimensionality reduction and clustering of the latent space



Dimensionality reduction and clustering of the latent space



Result from single cluster



Conclusion

CNN accurately categorises the samples, but only as well as we could.

Conclusion

CNN accurately categorises the samples, but only as well as we could.

VAE via CNN can cluster the latent space, and finds clusters that are not that great.

Evaluation of models

CNN:
Started working quite fast
Reached a high accuracy
Does not provide any new information/ can only predict what we saw

VAE via CNN:
Took a long time to get working (i.e many issues with preprocessing and long training time)
Clustered based on directions at best.
Might get better with more training.

Appendix

Project statement: All authors contributed equally.
Github: `https://github.com/AsbjornPreuss/AppMLFinalProject`

Hyperparameters CNN

Hyperparameter name	Parameter space	Final value
N_conv_layers	1-3	2
N_dense_layers	1-3	3
dropout_rate	0-0.5	0.4981
conv_filter_exp0	3-7	6
conv_kernel_size0	2-5	2
conv_filter_exp1	3-7	4
conv_kernel_size1	2-5	5
dense_units_exp0	3-5	4
dense_units_exp1	3-5	3
adam_learning_rate	1e-7-1e-1	0.0003676
adam_decay_steps	100-10000	3866
adam_decay_rate	0.8-1.0	0.9122

Hyperparameters CNN

N_conv_layers	3
N_dense_layers	3
dropout_rate	0.427
filters_layer0	41
filter_size_layer0	6
stride_layer0	1
padding_layer0	3
filters_layer1	49
filter_size_layer1	3
stride_layer1	1
padding_layer1	2
filters_layer2	9
filter_size_layer2	7
stride_layer2	2
padding_layer2	2

Hyperparameters VAE via CNN

No hyperparameter optimization was reached. This is the hyperparameters we manually chose.

Encoder:

Hyperparameter name		Final value
Learning rate		0.001
Latent space dim		50
Conv_1		12X12
Conv_2		6X6
Dense_layer_1		43264X200
Dense_layer_2		204X50

Decoder:

Hyperparameter name		Final value
Learning rate		0.001
Latent space dim		50
Conv_2		12X12
Conv_1		6X6
Dense_layer_2		200X43264
Dense_layer_1		50X200

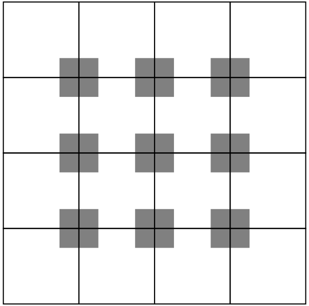
Preprocessing algorithms

Depth first algorithm:

1. Make data binary by threshold
2. Scan for values, starting from upper left corner
3. When a value is found, search neighbours to see if they have a value (larger than background noise)
4. If the neighbour has a value, add them to the cluster
5. Go to each neighbour in the cluster, and scan their neighbours
6. Repeat until no more neighbours with values are found
7. Go back and keep on scanning the image, whilst ignoring the ones that are already assigned a cluster.
8. Save height, width, size (in pixels) and summed value of the pixels in the original image (an estimate for total energy) for meta-data

Downscaling

The downscaling of images was done by averaging four nearest neighbours, thus reducing the image sizes with 1 along both axes.
This is done recursively, allowing for the images to be scaled to the required size.



Some thoughts on Convolutions in PyTorch

Debugging within training is difficult in PyTorch. The functions within an encoder can be accessed any time, this is hopefully a helpful snippet for debugging:

```
# encoder:
A = nn.Conv2d(1, 32, 3, stride=1, padding=1)
B = nn.Conv2d(32, 64 , 3, stride=1, padding=1)
C = nn.MaxPool2d(2)

# flatten
D = nn.Flatten()

# linear
E = nn.Linear(65536, 128)
F = nn.Linear(128, 10)

print(f'original data: {x.shape}')

print('encoder:')

print(f'Conv1: {A(x).shape}')
print(f'Conv2: {B(A(x)).shape}')
print(f'MaxPool: {C(B(A(x))).shape}')
print(f'Flatten: {D(C(B(A(x)))).shape}')
print(f'Linear1: {E(D(C(B(A(x))))).shape}')
print(f'Linear2: {F(E(D(C(B(A(x)))))).shape}')
```