Anomaly Detection with Autoencoders using PLT Data

Alfredo Castaneda
Brenda Leyva
Universidad de Sonora

IML Machine Learning Working Group meeting, January 18th, 2022

Motivation

- Data quality monitoring is a fundamental task in the operation of every detector in high energy physics.

- In preparation for the Phase-2 (HL-LHC) it is more evident that new tools, including ML algorithms, are needed to optimize different tasks, including the identification of anomalous data that could point to system failures (dead pixels, error in address decoding, degradation due to radiation).

- Machine learning algorithms are being implemented in different subsystems in CMS showing a better performance respect to the standard algorithms and techniques.

Introduction

The use of autoencoders for anomaly detection can potentially improve the time and accuracy of the detection process.

Preliminary results show that the implementation can be constructed with basic data processing and tools.

This work is inspired by similar implementations done in muon chambers. In a study by Pol et al.* DT data is used and various algorithms were trained, including CNN autoencoders

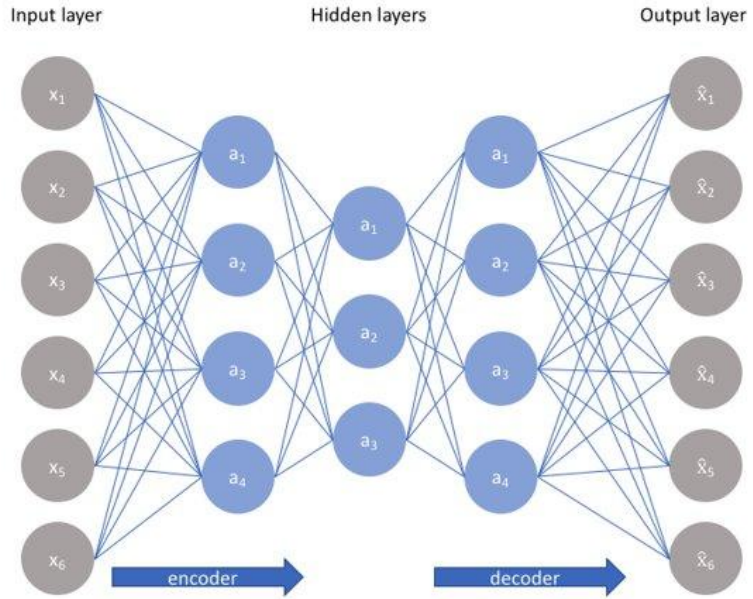
* Pol, A.A., Cerminara, G., Germain, C. et al. Detector Monitoring with Artificial Neural Networks at the CMS Experiment at the CERN Large Hadron Collider. Comput Softw Big Sci 3, 3 (2019). https://doi.org/10.1007/s41781-018-0020-1

Previous Work with PLT

- There is already an implementation for ML anomaly detection using PLT data (reported in DN-19-028).
- That method uses a clustering algorithm (K-means) with quite promising results.
- We want to collaborate in the same line of developments by working on additional methods.

Autoencoders

Autoencoders are an unsupervised learning technique in which we use neural networks for **representation learning**. The autoencoder *imposes a* bottleneck layer in the network that forces a **compressed knowledge representation** of the original input. We can take an unlabeled dataset and frame it as a supervised learning problem tasked with outputting a **reconstruction** of the original input.



Source: https://www.jeremyjordan.me/autoencoders/

The network is trained by minimizing the **reconstruction error** which measures the difference between the original input and the consequent reconstruction.

The top row shows the original input images. The second row shows the autoencoder's reconstruction from the compressed representation.

Source: https://blog.keras.io/building-autoencoders-in-keras.html

Autoencoders for Anomaly Detection

For anomaly detection, the algorithm is trained with information that represents expected or "normal" behavior.

When the model is used to predict on information that represents this normal behavior, the reconstruction **error is low**.

If the model is used to predict on information that deviates from the expected behavior, the reconstruction **error is high** and creates evident spikes. These error spikes can be easily identified using a threshold and labeled as **anomalies**.

In this example, the autoencoder was trained with the Keras dataset **mnist** (images of numbers). The reconstruction error on items of the same nature is low.

The dataset for prediction includes elements from the Keras dataset **Fashion mnist** (clothing items).

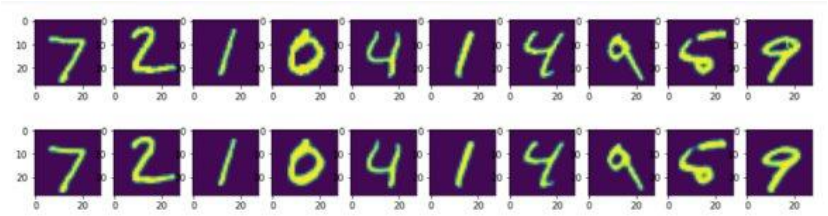
The reconstruction error for items of clothing is high and the items are labeled as anomalies because the autoencoder was trained to recognize numbers.

Keras dataset:
Fashion mnist

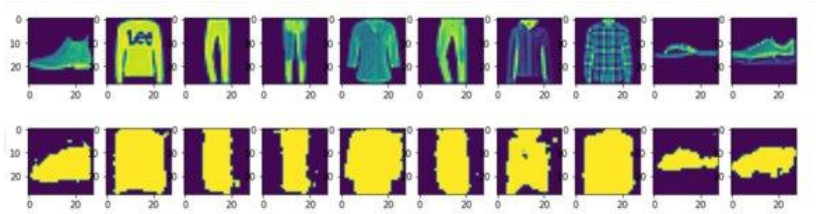
Keras dataset:
mnist

	MSE0	mse_outlier
0	6456.843262	1
1	29026.750000	1
2	14435.411133	1
3	7868.586914	1
4	10548.414062	1
...	...	...
19995	0.002142	0
19996	0.001315	0
19997	0.001373	0
19998	0.003728	0
19999	0.003721	0

20000 rows × 2 columns



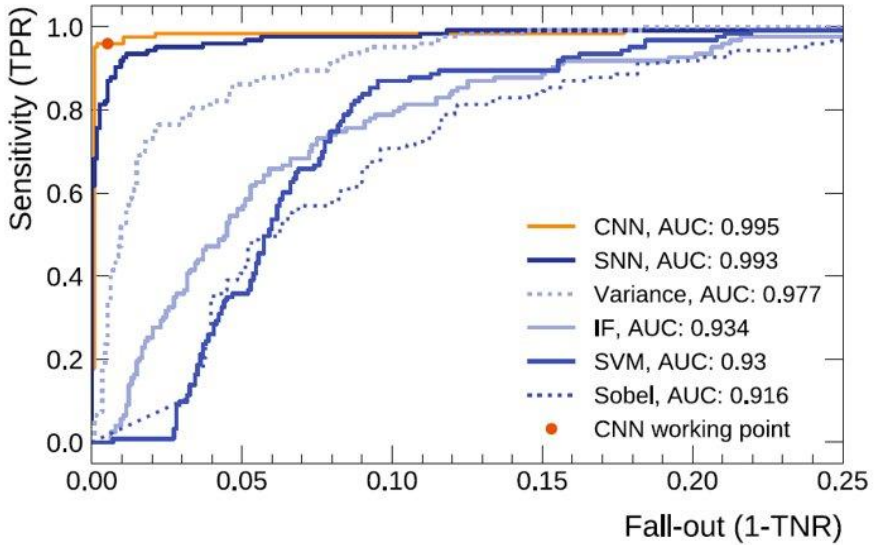
Autoencoder’s reconstruction of numbers



Autoencoder’s reconstruction of clothing items

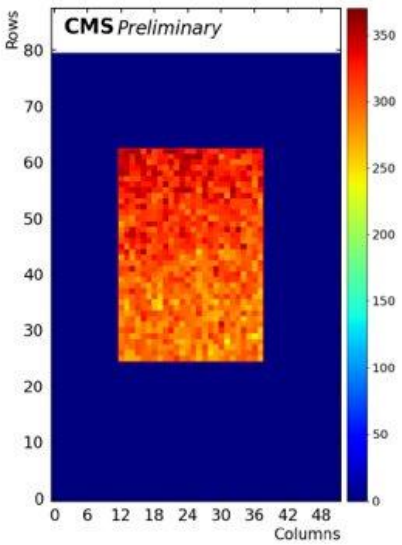
CNN Autoencoders

- Based on various sources, CNN autoencoders outperform compared with other algorithms.
- The performance depends on the system details (the plot on the right is for the DT system).
- The objective is to use CNN autoencoders as the main method and later compare with other algorithms.

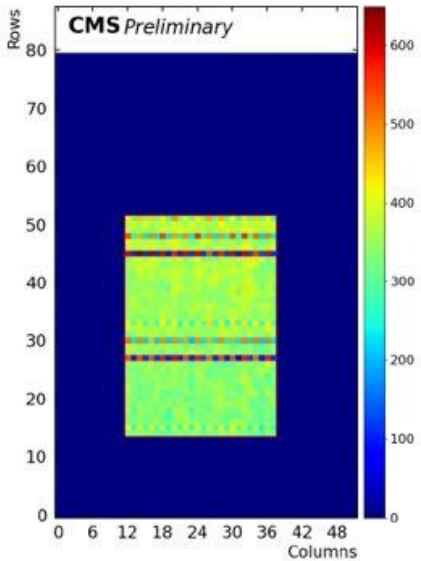


PLT Data

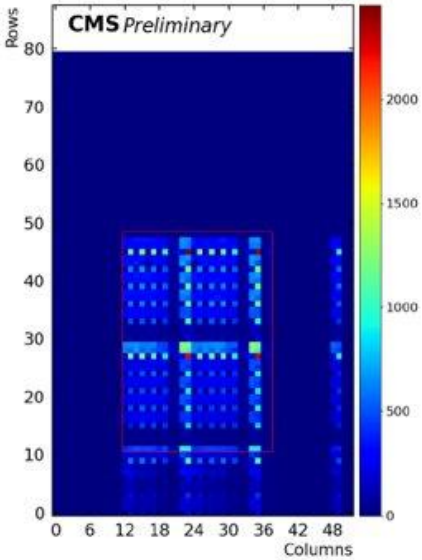
“Good” data



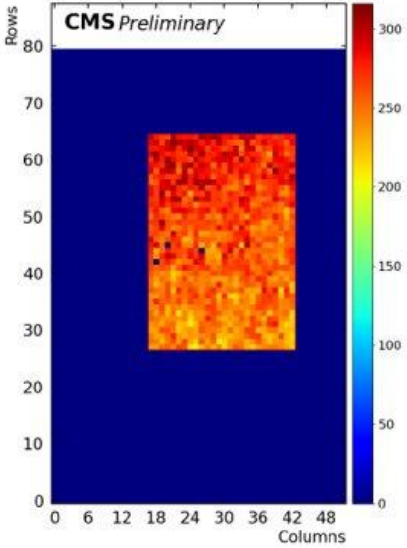
Anomaly



Anomaly



Anomaly

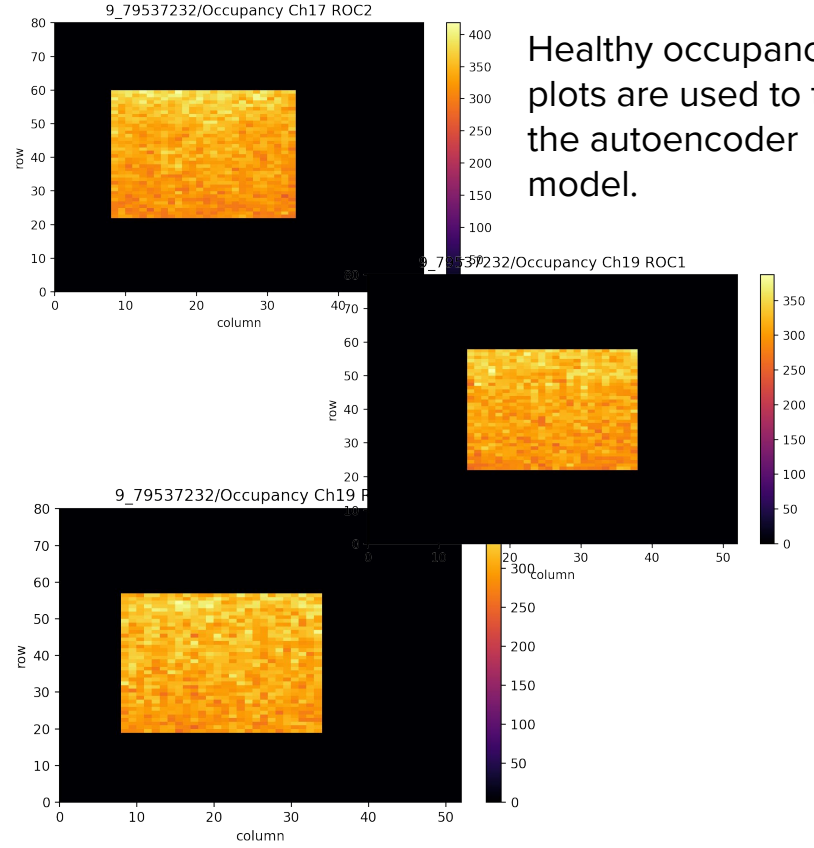


Examples taken from DN-19-028

Data Processing

- The data for this exercise is available through PLT Offline.
- The `.root` files contain SLINK data ran with five minute intervals over three years (2016, 2017, and 2018).
- The `.root` files are used to create occupancy plots with a simple **python** script using **uproot** to extract the numpy arrays.
- For this preliminary phase, the tools are used with a couple `.root` files. The generated occupancy plots are manually separated into healthy and anomalous plots.

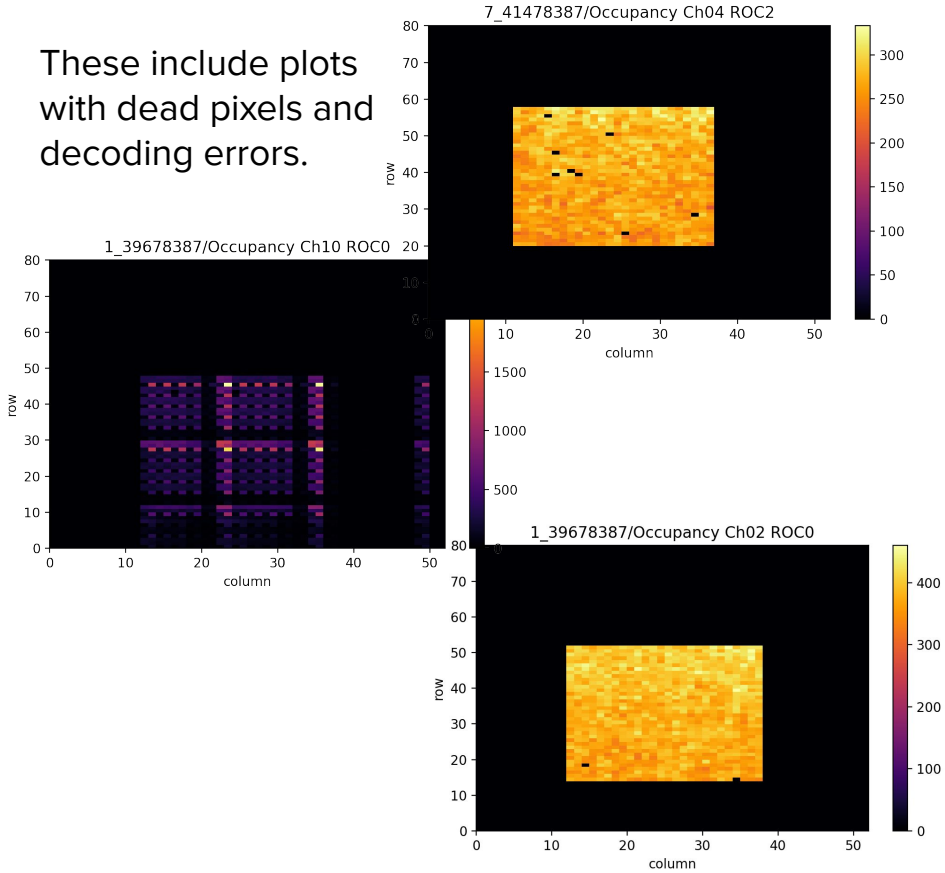
Examples of healthy occupancy plots



Healthy occupancy plots are used to train the autoencoder model.

Examples of anomalous occupancy plots

These include plots with dead pixels and decoding errors.



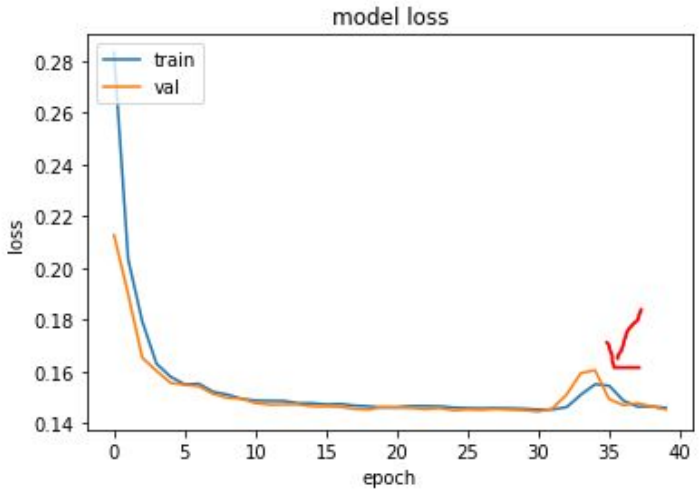
Simple Autoencoder Model (Keras)

```
input_img = keras.Input(shape=(4160,))
encoded = layers.Dense(2080, activation='relu')(input_img)
encoded = layers.Dense(800, activation='relu')(encoded)
encoded = layers.Dense(200, activation='relu')(encoded)

decoded = layers.Dense(800, activation='relu')(encoded)
decoded = layers.Dense(2080, activation='relu')(decoded)
decoded = layers.Dense(4160, activation='sigmoid')(decoded)
```

```
autoencoder = keras.Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')

history = autoencoder.fit(x_train, x_train,
                epochs=40,
                batch_size=15,
                shuffle=True,
                validation_data=(x_val, x_val)).history
```



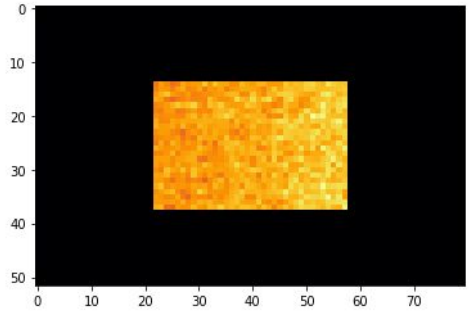
A basic arrangement of Dense layers with Keras

Shows an error “bump” towards the end of the training (same on various configurations)

Good at reconstructing the original images

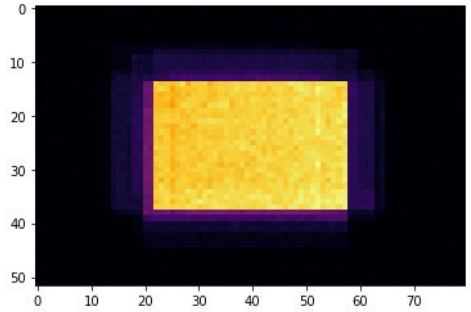
```
plt.imshow(x_test[1].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x25d826b9198>
```

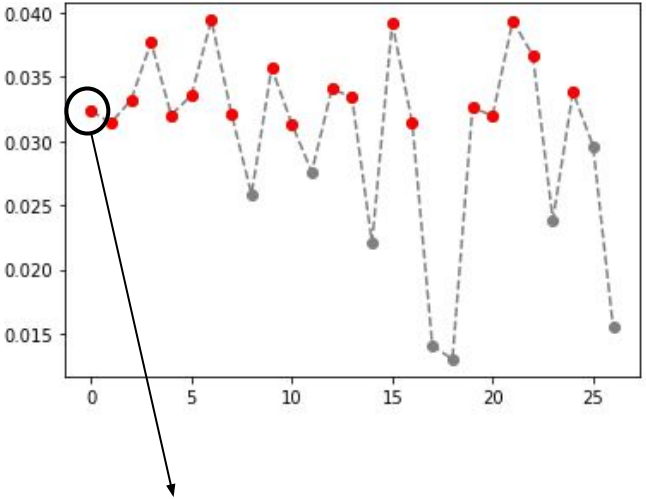


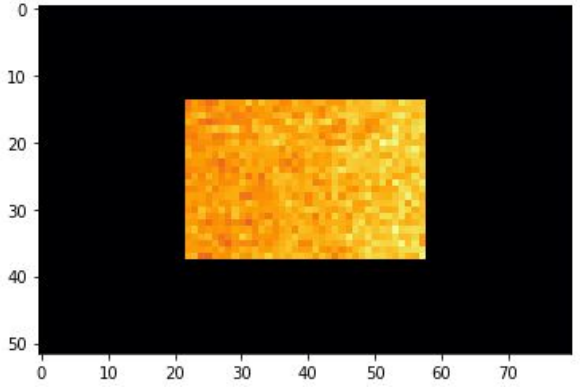

```
plt.imshow(out_images[1].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x25d82ae8a90>
```



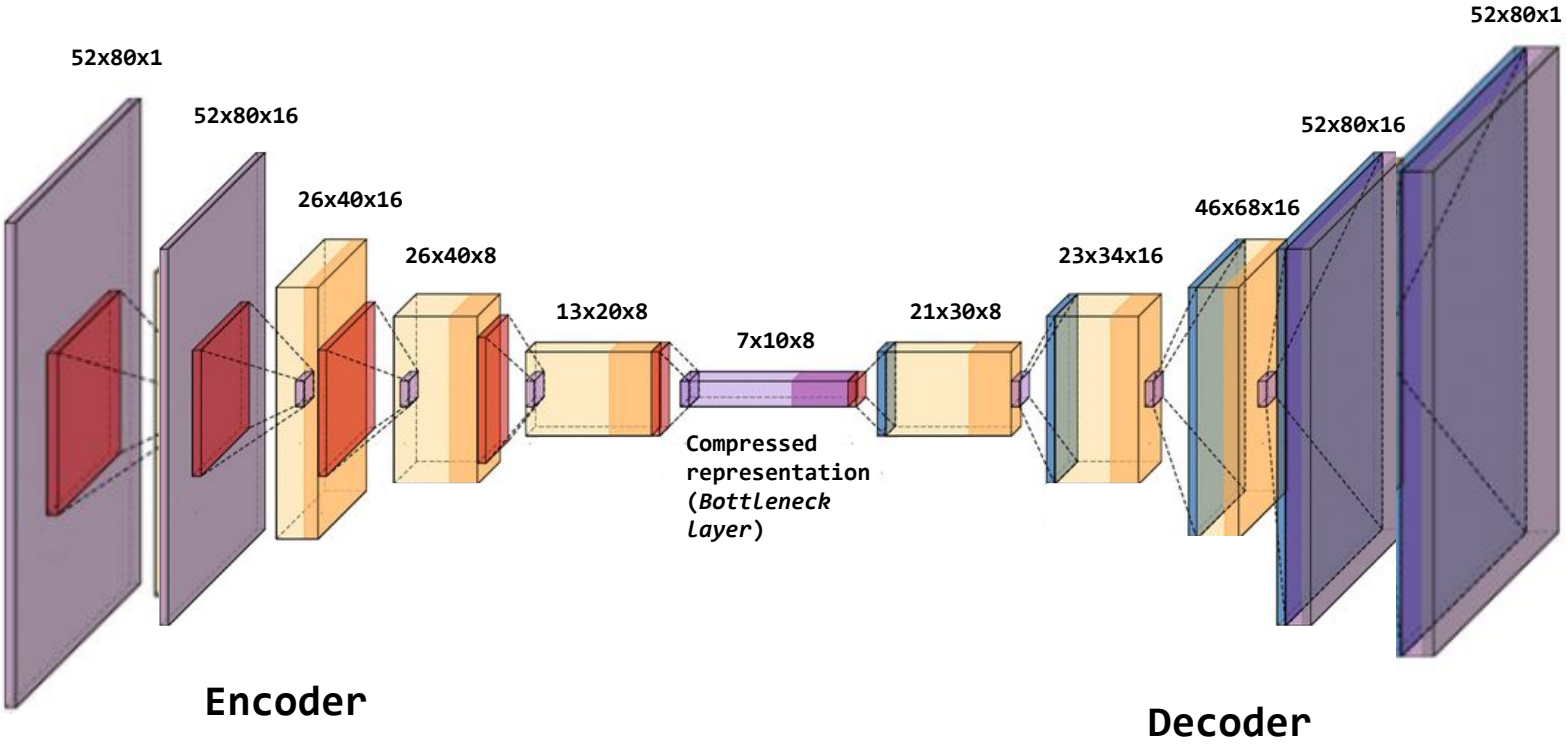
Not so good at detecting anomalous images





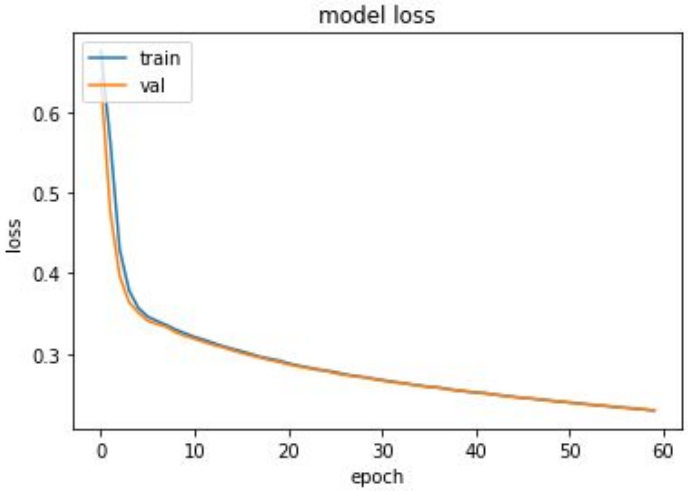
This normal plot shows a high MSE (mean squared error)

Convolutional Autoencoder Model (Keras)



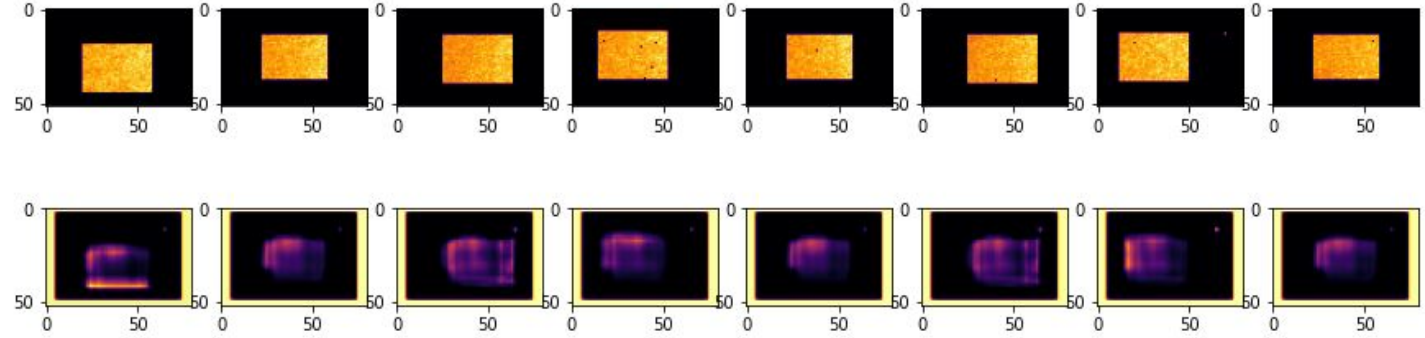
Model training

`x_train` (300 healthy occupancy plots)
`epochs` = 60
`Batch_size` = 15



Final values:
loss: 0.2294 - val_loss: 0.2295

Reconstruction results



The reconstruction of healthy plots needs improvement.

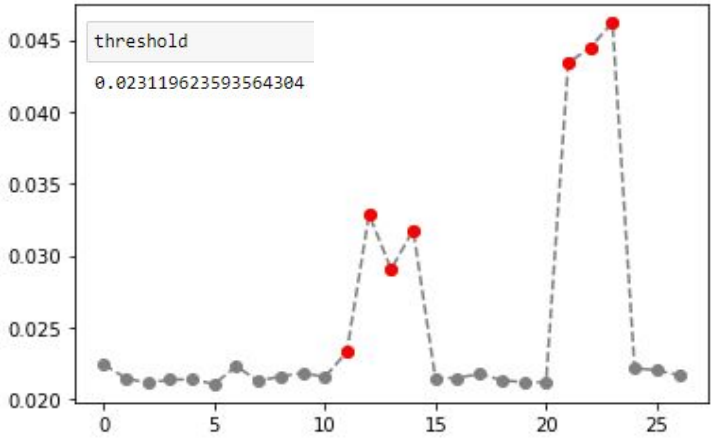
Preliminary Results

The model is used to predict on 27 items that include 6 occupancy plots with decoding errors (considered as extreme anomalies). The other 21 are healthy occupancy plots and occupancy plots with dead pixels.

Based on the results of the model training the MSE **threshold** is set as the **last value of training loss** that we obtained. The instances with an MSE over 0.03 are flagged as anomalies.

The results show that the autoencoder successfully detected the 6 instances of decoding errors.

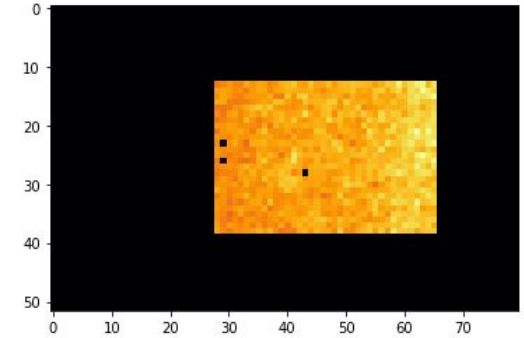
	MSE0	mse_outlier
0	0.022420	0
1	0.021478	0
2	0.021116	0
3	0.021367	0
4	0.021374	0
5	0.021054	0
6	0.022287	0
7	0.021281	0
8	0.021573	0
9	0.021841	0
10	0.021534	0
11	0.023287	1
12	0.032834	1
13	0.029054	1
14	0.031760	1
15	0.021436	0
16	0.021485	0
17	0.021762	0
18	0.021320	0
19	0.021185	0
20	0.021185	0
21	0.043376	1
22	0.044474	1
23	0.046177	1
24	0.022165	0
25	0.022013	0
26	0.021684	0



MSE: 0.023

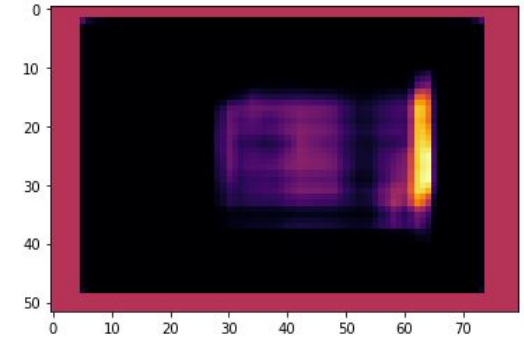
```
plt.imshow(x_test[11].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x178ee37aeb8>
```



```
plt.imshow(out_images[11].reshape(52,80), cmap='inferno')
```

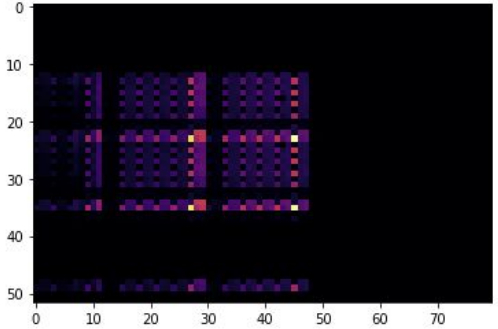
```
<matplotlib.image.AxesImage at 0x178eccd9518>
```



MSE: 0.033

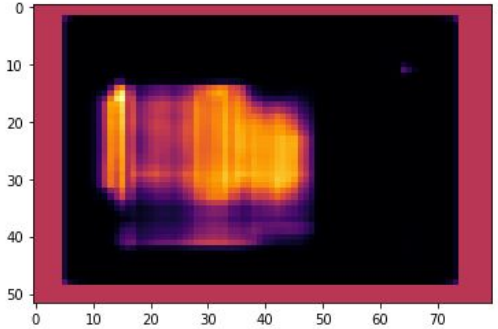
```
plt.imshow(x_test[12].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x21a7c790d68>
```



```
plt.imshow(out_images[12].reshape(52,80), cmap='inferno')
```

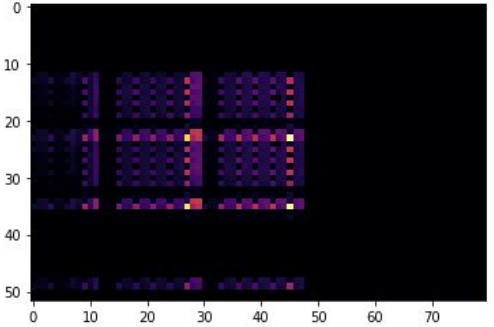
```
<matplotlib.image.AxesImage at 0x21a7d1425c0>
```



MSE: 0.029

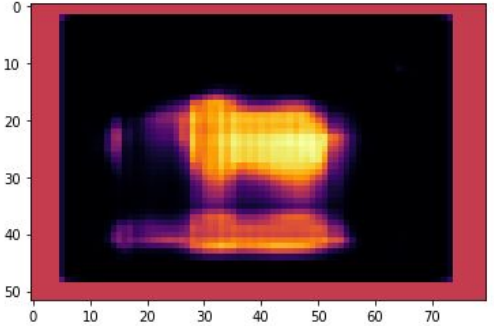
```
plt.imshow(x_test[13].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x21a7c790d68>
```



```
plt.imshow(out_images[13].reshape(52,80), cmap='inferno')
```

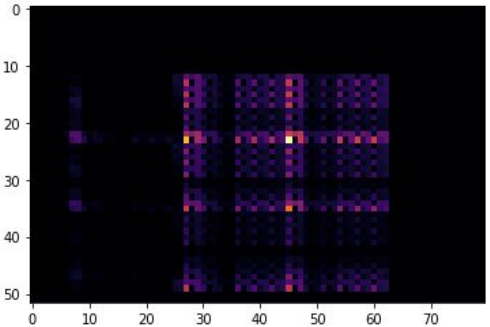
```
<matplotlib.image.AxesImage at 0x21a791b4cf8>
```



MSE: 0.032

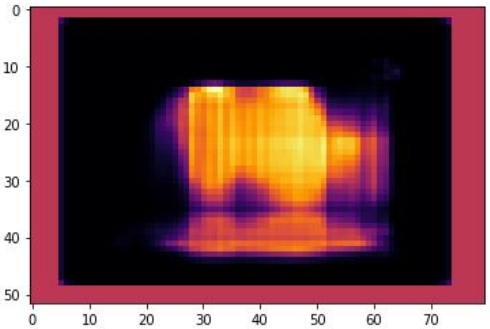
```
plt.imshow(x_test[14].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x21a7d3e45c0>
```



```
plt.imshow(out_images[14].reshape(52,80), cmap='inferno')
```

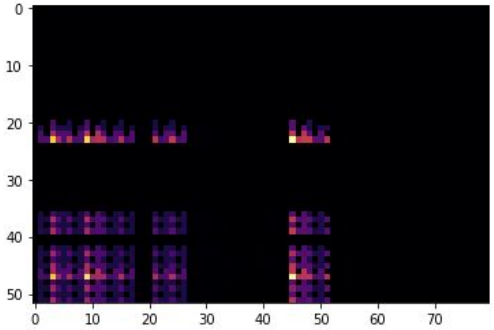
```
<matplotlib.image.AxesImage at 0x21a00ffce10>
```



MSE: 0.043

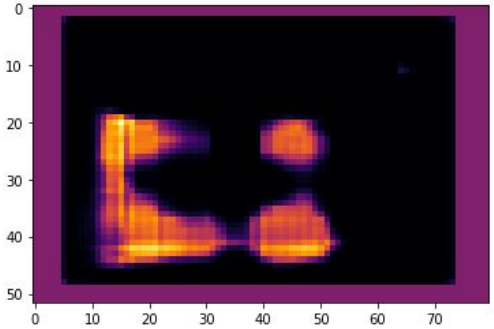
```
plt.imshow(x_test[21].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x21a0106a470>
```



```
plt.imshow(out_images[21].reshape(52,80), cmap='inferno')
```

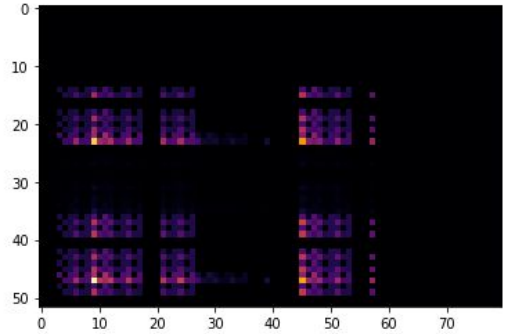
```
<matplotlib.image.AxesImage at 0x21a010c8dd8>
```



MSE: 0.044

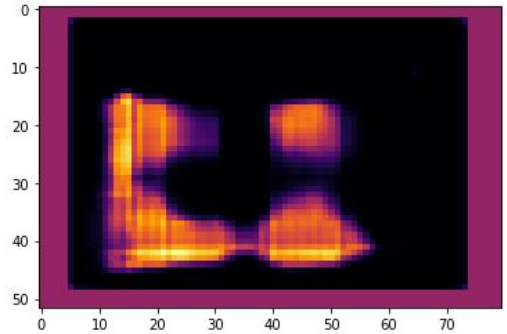
```
plt.imshow(x_test[22].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x21a01137400>
```



```
plt.imshow(out_images[22].reshape(52,80), cmap='inferno')
```

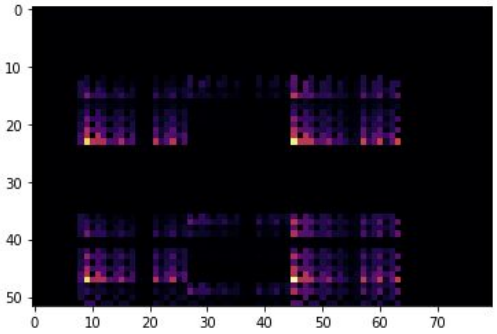
```
<matplotlib.image.AxesImage at 0x21a01196cc0>
```



MSE: 0.046

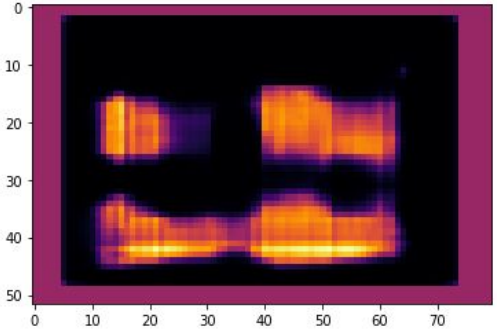
```
plt.imshow(x_test[23].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x21a01201550>
```



```
plt.imshow(out_images[23].reshape(52,80), cmap='inferno')
```

```
<matplotlib.image.AxesImage at 0x21a01260b38>
```



Next steps

- Increase the number of healthy occupancy plots for model training.
- Improve the reconstruction of healthy plots.
- Train variations of the model and the model's hyperparameters to optimize reconstruction of healthy plots (implement something like **Grid Search**).
- Improve accuracy to be able to detect plots with dead pixels (less severe anomalies).
- Extract and relate the date and time detail for each instance of the prediction.

Special thanks

To **Andrés G. Delannoy**, for his help with setting up and using PLT Offline, and for sharing scripts for occupancy plots.

To **Peter Major**, for reaching out and providing valuable resources for data and plots.