technische universität dortmund

Institut für Produktionssysteme

Technical University Dortmund

Project Report:

Pseudo X-ray Image Recognition Utilizing Machine Learning

Examiners: Prof. Dr.-Ing. Jochen Deuse
Prof. Markus Pauly
Prof. Jens Teubner
Prof. Erich Schubert

Submitted by: Manouchehr Norouzi

Matriculation number: 231831

Member of Group 5, Use case3

Submitted on: 24.08.2023

Dortmund, August

Confidential Clause

This project entitled “Pseudo X-ray Image Recognition Utilizing Machine Learning” by Manouchehr Norouzi is based on internal, confidential data and information of the following enterprise: SIEMENS AG.

This work may only be available to the first and second reviewers and authorized members of the board of examiners. Any publication and duplication of this project – even in part – is prohibited.

An inspection of this work by third parties requires the expressed permission of the author and the company.

_____

Signature

Table of Contents

List of abbreviations

2D	Two-dimensional
CNN	Convolutional Neural Network
nio	Nicht in Ordnung (EN: Not Ok)
io	In Ordnung (EN: Ok)

List of Figures

Chapter 1: Introduction

Problem statement and objective

In today's world, machines are our reliable companions, helping us get things done faster and with fewer mistakes. They shine when dealing with complex data, which can be tricky for humans to handle due to its intricate nature. Often logical algorithms are used to sort through this data. However, these algorithms have a drawback: they struggle with even minor changes in input. Humans, on the other hand, are adaptable but slower. Here's where machine learning comes in. It's like teaching machines to learn, similar to how humans do. Just as examples are needed to learn and improve, machines can use labeled examples to learn the patterns and get smarter over time. This makes them good at handling data and making decisions. There are different machine learning techniques, such as Decision Trees and neural networks, that help machines mimic human thinking, giving better results in less time.

This report focuses on applying machine learning to inspect X-ray images. It will explore how a special kind of machine learning model, the Autoencoder model, can improve this process. The goal is to reduce mistakes, make inspections more efficient and quicker, and bring precision to the next level. In the inspection section of the production line, light in the non-visible spectrum enables the feature inspection of hidden regions of interest as solder connections. The model will use these X-ray images to detect the nio images from these solders, which have the following signs as common types:

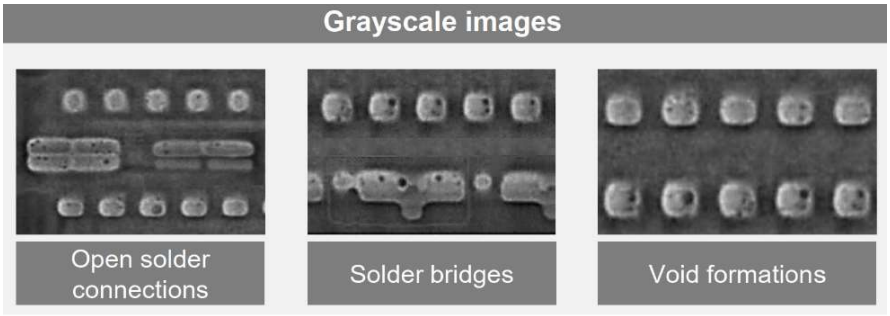

Figure 1. Frequent error patterns in the images

The model must capture these error types alongside other unidentified types and use them to detect the images being io or nio.

Structure of the Thesis

This report is dedicated to the binary image classification task for X-ray images, which will be done by an Autoencoder model as mentioned above. As an important step of an ML model, the images are preprocessed to be prepared to feed to the model, which will be explained in detail in Chapter 2.

In the next step, the model is structured due to the special circumstances of the task and the input data. Finally, predictions on new unseen data are done to check the output of the trained model on new images. The implementation of the model and prediction is completely explained in Chapter 3.

Methodology

This section outlines the methodology adopted for the development of an Autoencoder Binary classification model applied to X-ray images. The project adhered to the Cross-Industry Standard Process for Data Mining (CRISP-DM) methodology, which encompasses a structured approach to guide data mining projects through various stages. The methodology employed in this project involved iterative cycles through the CRISP-DM phases to achieve optimal results.

At the outset, the project aimed to comprehend the objectives and prerequisites. The focal point was designing an automated system capable of categorizing X-ray images as io or nio. This comprehension laid the groundwork for project scope and success criteria.

Understanding the dataset involved examining its content, size, and quality. The X-ray image dataset was probed to uncover insights. Initial data quality evaluation identified potential challenges such as resolution disparities, and class imbalances. Then, the dataset underwent preparation to make it suitable for modeling. Preprocessing included resizing images, changing them into gray scale (3-channel into 1-channel grey images), and normalizing pixel values. In a further refinement of the data preparation, due to the imbalanced nature of the data, the data augmentation process, duplicating minority class (nio images) was performed. The next step of the data preparation was to divide the images into train-validation and test data to get them ready for the model training. The initial model construction phase focused on the Autoencoder Binary classification model. This architecture was designed to compress input X-ray images, emphasizing relevant features for classification. Model training utilized the prepared dataset, with hyperparameters fine-tuned for optimal performance. Finally, the trained model was evaluated on the test data, and prediction on the raw data was done. The results from the test data are presented in the end.

The CRISP-DM method was performed iteratively, and subsequent modeling iterations aimed to refine the data preparation, augmentation, and overfitting prevention function (early sopping) of the initial model. Insights from the first iteration guided adjustments to model architecture and training methods.

Each iteration's model underwent evaluation similar to the initial iteration, tracking progress toward meeting business objectives, including the use and improvement of the value of Recall instead of Accuracy due to the imbalanced classes of the data.

Chapter 2: Data Preparation

Data preparation is a crucial cornerstone in the realm of machine learning tasks. Effective data preparation ensures that the input images are transformed into a suitable format for the machine-learning model. This involves tasks such as loading and resizing images to a consistent dimension, normalizing pixel values to a common scale, and partitioning the dataset into training, validation, and testing sets. Python libraries NumPy, and Torchvision-PyTorch facilitate these operations, allowing for image augmentation techniques like rotation and flipping that enhance the model's ability to generalize. Furthermore, data preprocessing steps like converting images to tensors (numerical format) and applying one-hot encoding to class labels are vital for model training. In sum, meticulous data preparation not only enhances model performance but also fosters robustness and efficiency throughout the image classification pipeline.

Reading the Data

Hyperparameters play a critical role in the success of a machine learning project. They are the parameters that are set before the actual training of the model and significantly influence its performance, convergence speed, and generalization. So, as an initial step, the hyperparameters are defined and set. However, these hyperparameter values were adjusted several times due to the objectives of the task to get the best possible result.

In a machine-learning task, the first step is to read and preprocess the images before they can be fed into a machine-learning model. This process involves loading images from the dataset, converting them into a consistent format, and preparing them for training and validation. Therefore, the images were saved in two separate folders named “io” and “nio” folders. An initial transformation using the “transforms” module of the “Compose” class from PyTorch is set to read the images into a variable. The images are changed into tensors and then resized to have a height and width of 256 pixels each. Resizing images to a consistent size is a common preprocessing step that ensures that all images in the dataset have the same dimensions. This is important because most machine learning models expect input data to have uniform dimensions. It makes it easier to process and train models on the data. Furthermore, larger images require more computation and memory resources to process. Resizing images to a reasonable size reduces the computational burden on the training process and allows to training of models more efficiently. By the way, resizing images can help improve the generalization of the model. By training on images of different sizes, the model might learn to focus on more prominent features and patterns rather than becoming overly dependent on specific image resolutions.

In this initial transformation for loading the images, antialiasing is set to True. Antialiasing helps to smooth out the edges of the image during resizing, reducing aliasing artifacts that can occur.

A small note to add, a selection of images is manually taken from both io and nio (15 io images and 11 nio images). These images will be used as input for the prediction stage. Before processing the data, these images are set aside in the initial stage. This ensures that the model has not seen them before and allows for appropriate input during the prediction assessments.

One sample of the images used in this task will be presented in the next pages, where the data transformations are explained.

Data Augmentation

Imbalanced data is a common challenge in machine learning, particularly in tasks like image classification. In the context of image classification, this means that one class may have a much larger number of images compared to other classes. This imbalance can have a significant impact on the performance of the model. Models trained on imbalanced data tend to be biased toward the majority class. Since there are more samples of the majority class, the model might prioritize predicting that class correctly at the expense of the minority classes. Moreover, imbalanced data can lead to poor generalization of new, unseen data. The model might struggle to accurately predict minority classes because it hasn't seen enough examples of those classes during training. When evaluating a model on imbalanced data using metrics like accuracy or recall, it can be misleading. Even a model that predicts the majority class for all instances might achieve high accuracy, but it's not performing well on the task, resulting in a lower recall or f1-score.

In the context of this task, a significant class imbalanced data, with approximately 1800 io images and only 200 nio images is encountered. Addressing this imbalance is crucial to ensure that the classifier can generalize effectively and make accurate predictions on both classes.

To mitigate this issue, and enhance the robustness of the model, data augmentation techniques can be specifically tailored to each class, including random rotations, random flips, adding noise, color jittering, random crops, and so on. In this task, for images classified defective (nio), a random vertical is applied to introduce variability in the orientation of these images. This augmentation was performed with a probability of 0.6, ensuring that most of the nio images would undergo this transformation. The resulting augmented images were integrated with the original dataset to form the final dataset for training and evaluation. The outcome of this augmentation process was a more balanced dataset for training, consisting of both original and augmented images. This dataset is expected to lead to improved model performance in predicting both classes more accurately.

Moreover, another method is used to tackle the imbalance data, called “weighted loss function, which takes into account the weight of the classes and adds its effect on the model training. This will be explained in the next chapter.

There is also the possibility to augment the io images and add them to the main data, which is included in the code file, as a note. In this case, there will be more data to be trained, and this can result in slightly better results. It is also possible to add more data augmentation for the nio images to reduce the distance of the class counts, and one can it to have more data, but naturally, as a rule in machine learning, this requires a higher system, and also will result in more training time, which due to the requirements of the task, less training time is more preferable. Therefore, these image augmentations are included as a note cell.

The final result of the image augmentation in this task is that the total number of images increased to exactly 2162 images including 378 nio and 1784 io images.

In the end of this section, the labels are extracted and saved into a variable named “labels”, which will be used in visualizing sample image.

Data Split and Transformers

Splitting data into train, validation, and test sets is a common practice in machine learning tasks and serves several important purposes including model training with the train dataset and then evaluating the train procedure on each epoch with the validation set. Then the final trained model will be tested on unseen new data to check the suitable metrics for the task. Furthermore, during the training process, machine learning models often have hyperparameters that need to be tuned for optimal performance. The validation set is used to fine-tune these hyperparameters and choose the best configuration without overfitting to the training data. By evaluating a model's performance on the validation set, one can compare different models or algorithms and choose the one that performs best on unseen data.

Moreover, an existing validation set apart from the train and test sets deals with the overfitting issue. Overfitting occurs when a model becomes too specialized in learning the training data's noise and peculiarities, resulting in poor generalization to new, unseen data. The validation set helps monitor the model's performance on data it hasn't seen before and prevents overfitting by stopping training when performance on the validation set plateaus or worsens, which is done by setting an early stopping criterion in the validation loss.

In the split procedure, it must be ensured that the data division is being done randomly so that no intentional bias enters the data and according to the model training. To do this, a random split from "torch.utils" with a generator of a fixed seed value is used to ensure the reproducibility of the random operations.

For this dataset, the final images are divided into the train-validation and test datasets with a rate of 80-20 percent. Then the train-validation set is divided with the rate of 80-20 to the sets of train and validation. This ensures that there are enough images to input into the model as train validation and test procedures. Other rates could be used such as 70-15-15, but the selected rates have provided suitable results for the task.

The next step is to define the data transformations. Image transformations, through techniques like data augmentation, provide multiple advantages for training machine learning models on images. They enhance model performance, robustness, and generalization by exposing the model to a wider range of variations and scenarios, ultimately improving its ability to better handle the data. To define the image transformations, the mean and standard deviation of the images in the train-validation set are calculated. These values are used in the normalization of the images in both train and test sets.

Three different transformations are defined in a dictionary and the data will use each of them accordingly: train, test, and prediction transformations. The train and test transformations are set to be identical because the raw data in the test set must get the same changes as the train data. The transformations for the train and the test set include changing the image into grayscale, transforming images into grayscale mode to make the learning process faster, random rotation, setting color jitter, and normalizing images with parameters of the train-validation set. Moreover, the prediction transformation includes transforming the images into tensors, resizing the images, changing them into grayscale, setting color jitter, and finally normalizing them with the same parameters of the train-validation set. The reason for using the mean and std of the train-validation set for the test set is rooted in the principle of ensuring that the model's training and testing data are treated consistently and are representative of the same underlying data distribution. This is done to avoid data leakage.

Data Loaders

To create the data loaders to feed into the model, the data first needs to be read and then the labels and images extracted, and then the transformations introduced in the previous section be applied. For this reason, a class named "MyDataset" is defined which allows to creation of instances of the dataset with specific transformations. It takes the data and the transformation dictionary, defined above, as input, separates the label and the image, applies the transformations, and finally returns the labels and images as output. In this class, the default value for the transform variable is set to be "transform=

None", so that it can be used for the cases without transformation and return the raw images. Using the MyDataset class, the instances of the train, validation, and test datasets with appropriate transformations are created and then fed into the "DataLoader" from "torch.utils.data" to create the data batches which will be fed as input to the model. These data loaders will be used to efficiently load and manage the data during the training and evaluation of the network. A generator is also used in the data loaders to ensure to prevent differences in the loader result for each run of the code. Using Data loaders and subsequently data batches have different positive reasons including memory efficiency, parallelism, generalization, training stability, regularization, and some other reasons.

A point to add on loading the data, the batch size as a hyperparameter is set to be equal to 128. Different values of the batch size including 32, 64, 128, and 256 are tried in this task several times with other hyperparameters and the result was that higher batch sizes led to faster train but poor generalization, and also there was a memory issue. On the other hand, with lower batch sizes, the convergence was faster and had better results, smaller batch sizes allow the model to "start learning before having to see all the data., but the time efficiency was not desired and it took more time to model to reach to the stopping point.

The shuffle argument is set to be True. This indicates that the data will be shuffled randomly before being divided into batches for each epoch. Shuffling the data helps prevent the model from learning the order of the data and reduces the chances of overfitting.

Now that the data loading and data transformations are explained, one sample of the raw image and its transformed result is presented as:

Figure 2. Raw image and its transformation

Chapter 3: Model train and evaluation

Neural networks are particularly successful in image classification tasks due to their ability to learn complex patterns and hierarchical features from pixel data. Convolutional Neural Networks (CNNs) are one of the most common types of neural networks used for image classification tasks. Another very well-known type of neural network for image classification tasks is Autoencoder models. An autoencoder is a neural network model that seeks to learn a compressed representation of an input. It is a neural network that is trained to attempt to copy its input to its output. An autoencoder model is a particularly good choice for imbalanced data due to its capability to feature extraction representation learning and anomaly detection.

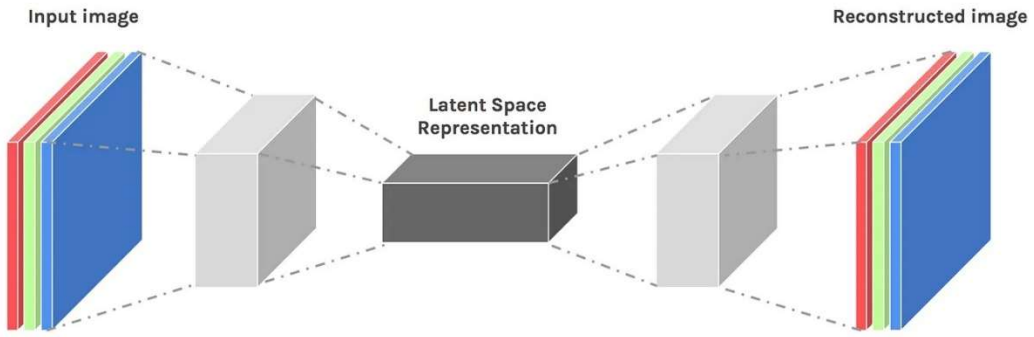

Figure 3. Autoencoder model Structure

Autoencoder models consist of two different sections.: Encoder and Decoder. The encoder is responsible for mapping the input data into a lower-dimensional representation, often referred to as the "latent space" or "encoding." This representation captures the essential features of the input data. The goal of the encoder is to compress the input data while preserving important information. The decoder takes the encoded representation from the encoder and aims to reconstruct the original input data from this representation. It's responsible for transforming the compressed representation back into a format that closely resembles the original input data. Different layers can be used in the encoder section including convolution layers, activation functions, pooling layers, and dropout layers. Furthermore, in the decoder section, different kinds of layers for different purposes can be used, such as transposed convolution layers, activation functions, and output layers.

This architecture is designed to perform dimensionality reduction and feature learning on grayscale image data. The encoder extracts important features from the input images, and the decoder attempts to reconstruct the original images from the learned features.

Model Structure

To our task, due to the imbalanced nature of the data, a supervised Autoencoder model is used. They are effective for image classification tasks, especially when only a small amount of labeled data is available. They are used to classify images of different objects and X-ray images. Supervised autoencoders can learn more complex representations of images, be used for semi-supervised learning, and extract features from unlabeled data that are relevant to the classification task.

For the used model, the encoder part gradually reduces the spatial dimensions and increases the depth (number of channels) of the input image. A convolutional layer(nn.Conv2d) that takes a single-channel (grayscale) image as input. Then a ReLU activation function element-wise is applied. The third layer performs a max-pooling operation to reduce the spatial dimensions. Then a dropout regularization (nn.Dropout) is set to reduce the chance of overfitting. More layers like the first one (convolution, ReLU, max-pooling, dropout) are repeated to gradually reduce the dimensions and increase the depth of the features.

For the decoder section of the model, transpose Convolutional layers and then RELU activation function are used. The final layer of the decoder uses a sigmoid activation function to ensure the pixel values are scaled between 0 and 1.

In the end, a class of classifier is defined which represents a binary classifier that uses the features learned by the autoencoder's encoder to classify input images.

Training the Model

Before initiating of the training process, the optimizer of the model is set to be Adam (Adaptive Moment Estimation). Adam is a popular optimization algorithm used for training neural networks and other machine learning models. It is an extension of the stochastic gradient descent (SGD) optimization method that incorporates adaptive learning rates and momentum terms. Furthermore, the cross-entropy loss (nn.CrossEntropyLoss) is used as the loss function with an argument of weights, which is set to be the weights of the classes of the images to address the issue of class imbalance. The purpose of class weights assignment is to give more importance to minority classes and less importance to majority classes during the optimization process.

In the end, an early stopping function (EarlyStopping) is defined and set in the training process to monitor the fluctuations of the validation loss value and stop the training procedure if the count of these fluctuations exceeds a fixed predefined value. This will particularly prevent the model from overfitting.

The model in the validation section of the training process is set to be in evaluation mode(model. eval()). When a model is in evaluation mode, it behaves differently than during training. This mode is typically used when making predictions, performing inference, or evaluating the model's performance.

Evaluating the Model

After setting all parameters and required functions and variables, the model training was done on the training dataset and simultaneously, at each epoch, the model was evaluated by the validation data. The model stops after a while due to the early stopping patience parameter and the following result was achieved:

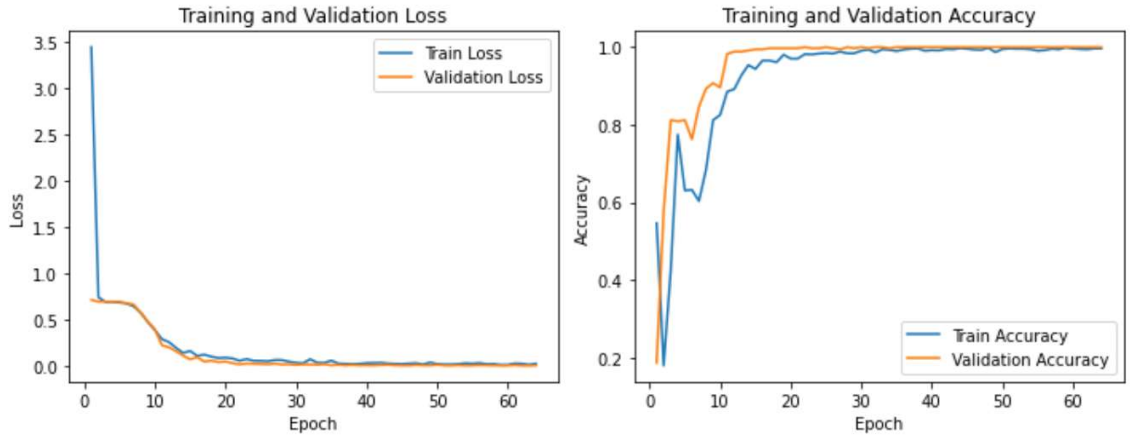

Figure 4. Train and Validation Loss and Accuracy

The reason the validation accuracy is higher than the training accuracy is that in the model structure, dropout layers are used. These layers drop randomly some of the data in the training phase but use the whole data for the evaluation phase.

It must be mentioned that if the early stopping parameter is chosen to be higher, the model more tends towards overfitting, and if it is chosen to be low value, then the training process will stop in fewer epochs and the model might not learn optimally.

The main evaluation of the model is done on the test data to check if the results on the new unseen data are desirable. The important point in this result is that due to the imbalanced data, accuracy will not provide realistic results, as it will not reveal the bias of the model towards the majority class. For this reason, the recall and the f1-score measures will be considered. The evaluation output for the test data is as follows:

```
Test Accuracy: 100.00%, Test Loss: 0.0036, Test Recall: 100.00%, Test f1: 100.00%

 Confusion Matrix:

 [[364   0]
 [  0  68]]

 Classification Report:

              precision    recall  f1-score   support

           0       1.00      1.00      1.00       364
           1       1.00      1.00      1.00        68

    accuracy                           1.00       432
   macro avg       1.00      1.00      1.00       432
weighted avg       1.00      1.00      1.00       432
```

The desired recall value, as asked by the SIMENS, is 99%, and the result for the model on the test data is 100%. From the confusion matrix, it can be seen that almost all images are classified correctly, which indicates the good performance of the presented model. This also shows the high robustness of the model. In the end, the trained model's parameters ("state_dict") were saved and then can be easily loaded to further uses, without rerunning the time-consuming training step.

Prediction on New Images

The next step in the modeling is to use the trained model to predict new images, which were extracted before loading the initial data. Here, there are 15 io images and 11 nio images which are used to check the prediction capability of the fitted model. These images are completely new and unseen to the model and are placed in a folder to read from. A transformation almost the same as the train data is applied for these images and then fed to the model. The results show that the model was able to completely predict the correct labels for the images. Moreover, to check whether the model captures the labels from the names, the images were renamed differently, and again the model was able to predict completely correct labels. The predicted images will be finally saved to the corresponding io/nio folders.

Class activation map

Activation maps, also known as feature maps, are a crucial concept in the field of deep learning and play a significant role in understanding the behavior and information flow within these networks. Activation maps are essentially 2D grids of numerical values that represent the output of specific neurons or units within a convolutional layer of a neural network. These maps capture the responses

of these neurons to particular features or patterns in the input data. Activation maps help visualize and understand what features the network is detecting as it processes an input image.

Two different activation maps are implemented in the code for this task: Classical Method, and State-of-the-art Gradient-based Methods.

For each of the visualization methods, the flexibility to configure the parameter "onlyLastConv" is added to allow to visualize of the last layer or all convolutional layers.

The outputs of the gradient-based (SmoothGradCampp XGradCAM) for the last layer of the model are:

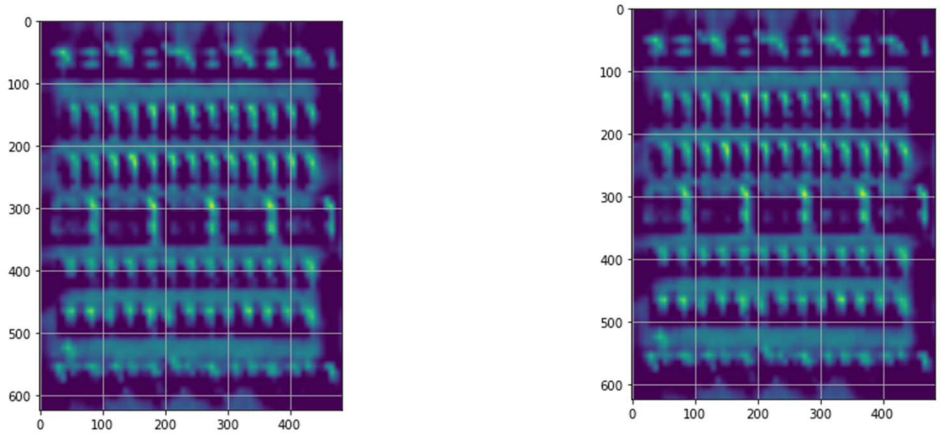

Figure 5. Class activation map representations

The results of both gradient-based methods are almost identical and show the same impact of the model on the image in the classification task for both functions.

Conclusion and Outlook

In conclusion, this report has presented a comprehensive exploration of the binary image classification task, showcasing significant advancements achieved through the application of a supervised autoencoder model. Through a series of carefully orchestrated steps, it is demonstrated how the performance of the classification task has been greatly enhanced, underscoring the effectiveness of this innovative approach.

However, the introduction of this model marked a turning point in the approach. By combining the capabilities of both autoencoders and classification layers, the power of unsupervised pretraining and supervised fine-tuning was harnessed. This hybrid approach not only enabled the extraction of meaningful features from raw image data but also facilitated the learning of highly discriminative representations specific to the classification task.

List of References

- LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.
- Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep learning. MIT Press.
- https://towardsdatascience.com/autoencoders-bits-and-bytes-of-deep-learning-eaba376f23ad
- https://frgfm.github.io/torch-cam/methods.html#torchcam.methods