Machine Learning

Lab 7:

Final Project: Industrial Anomaly Detection

Load: 6-10 hours in the lab

80 hours of personal work

Programming language: Python

1. Objective

The objective of this lab is to begin the first phase of your final project, which involves addressing a real-world quality control problem. Specifically, we will analyze images from MILAN Stationary Company's automated imaging system, which is used to inspect the quality of manufactured erasers.

Figure 1: Erasers produced by MILAN in three different colors.

Each pack contains 15 erasers, and a fixed-angle camera captures images of the entire pack across multiple color channels. The goal is to develop methods for automatically evaluating two key quality aspects:

1. The printed logo on each eraser must be clear and free from defects

2. Each eraser must be correctly shaped: square, undamaged, and without physical defects.

This lab introduces the foundational steps for processing and analyzing these images to support automated quality assessment. This repository has listed all the useful articles on industrial anomaly detection. Use it for understanding the problem and finding effective solutions:

Here is a sample quality control project And here is the link to Pytorch basic tutorials with information that you'll need:

2. Environment Set Up

For this project, you should work in groups of two or three students. You should pick a name for your team and use the same name for saving your model.

To begin, create a new notebook using either Google Colab or Jupyter Notebook. At the top of your notebook, include a text cell with the name of your group and listing the full names and student ID numbers of all group members. This information should be clearly visible and formatted neatly. Only one member of the group needs to submit the completed notebook and the final model; duplicate submissions are not necessary. Make sure all code runs correctly within the notebook environment, and that any required libraries or dependencies (such as PyTorch, NumPy, Pandas, Matplotlib, or OpenCV) are properly installed and imported before running the project. Don't forget to define your device.

3. Importing Dataset and Necessary Libraries

The dataset provided by MILAN contained images with 15 erasers each as seen in figure 3, for each image captured, 6 channels were saved as their own grayscale image file, each denoted by a different suffix in the file name: AM, GRAY, G, R, W. An extra capture for another channel is also taken for each image with the suffix: NORMAL.

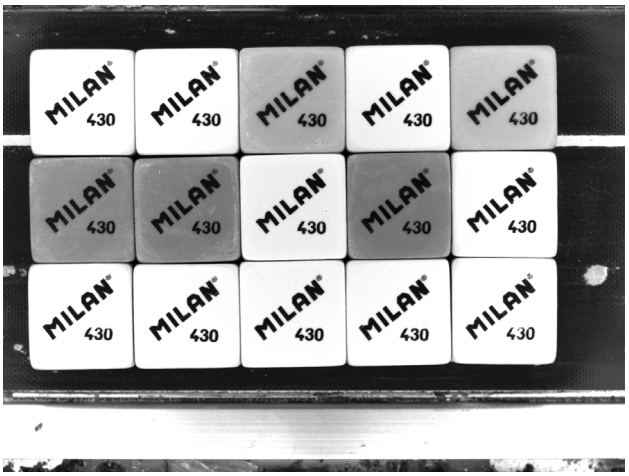

Figure 3: Grayscale photo of the erasers as they appear on the conveyor belt to check for any errors or defects.

To make the processing easier, all images have been cropped to generate a new dataset where each one contains a single eraser. This means that for each individual eraser, we have 6 different images. But since these images won't be in a standard image format, we created a new dataset by channel stacking. The new dataset (arrays.zip) is formed of .npy files with 310x310x6 arrays, each channel being GRAY, AM, G, R, W and NORMAL in that order. The original grayscale images have also been provided for you (images.zip) as png files in case you feel it's easier to use them. They share the same name as the .npy files as a prefix and have a different suffix indicating their respective channel.

The annotations have been provided in two csv formats: binary and multi class. In the first column of each csv file you have the ID of the eraser. By using these IDs and adding the correct suffixes (such as .npy, _AM.png, etc.) you can access the images. The second column in the multi-class annotations is a string in the form "X_Y", where X and Y are a binary classification of printing error and physical defect respectively. Some defect and printing error examples can be seen in figures 3, 4 and 5 below. In the binary annotations, the second column has a binary value indicating whether or not the eraser has a defect (1 being defected). You can load these annotations as a dataframe to accelerate your data loading pipeline.

Figure 3: Example of eraser defect where some kind of eraser residue is pressed on top of the eraser.

Figure 4: Example of eraser defect where a crack has appeared.

Figure 5: Example of printing error where the "N" is missing ink.

4. Data Analysis and Pre-processing

In this part of the lab, you should begin by analyzing your dataset to understand its structure and quality before building the model. Start by performing a statistical analysis using the provided annotations to examine the distribution of different defect types, such as logo defects and shape irregularities. Next, calculate descriptive statistics (e.g., mean, variance, and class balance) to identify any imbalances or biases in the dataset. You can also visualize the data using plots and graphs, such as histograms of pixel intensity, color channel distributions, and

sample image grids, to gain insight into the image quality and labelling consistency. In this step you should decide how you want to classify your data (binary or multi-class classifier) and what metrics you want to use.

Label Uncertainty: the annotations for these images were generated by an automated system. Therefore, some of the labels might not be correct and some erasers might have defects but were missed by the annotation system. You should visually analyse this in your dataset and report it in your deliverable. Later one, you can investigate the its impact on your model performance.

After completing the analysis, define a data pre-processing and augmentation pipeline. All images should be resized to square dimensions of equal size and converted into tensors suitable for model training. In addition, apply various data augmentation techniques such as random rotations, flips, brightness and contrast adjustments, and small affine transformations(you can add any augmentation algorithm that you feel would be good for this task). This will help increase the variability of the dataset and improve the model's ability to generalize to new images. Because we are using custom data, you must build a custom Pytorch dataset class that can wrap your data for your data loader. In this class you will also apply your transform function.

In the next step, split your data into train and validation. You must make sure that your splits are stratified (see train_test_split library documentation on scikit-learn). After that, define your train and validation loaders.

---

5. Build and Compile the Model

In this section, you should build a Convolutional Neural Network (CNN) model to train and validate on your processed dataset. Design your network architecture according to the complexity of your data (simpler datasets may require fewer layers, while more complex data might benefit from a deeper network). Each convolutional layer should be followed by a batch normalization layer and an activation function (such as ReLU or Swish) to help stabilize and improve training performance. You can also include pooling layers where appropriate to reduce spatial dimensions and computational cost and dropouts to reduce overfitting.

After the convolutional feature extraction layers, flatten the output and connect it to a fully connected (dense) classifier network. The final layer of your model should contain as many output neurons as the number of classes you want to predict, using an appropriate activation function (e.g., softmax for multi-class classification or sigmoid for binary classification).

Once the architecture is defined, compile your model using an appropriate optimizer (such as Adam or SGD), a suitable loss function based on your prediction task (e.g., cross entropy for multi-class), and relevant performance metrics (e.g., accuracy, precision, recall).

Here is a list of public repositories on building CNN models with Pytorch:

https://github.com/Bengal1/Simple-CNN-Guide
https://github.com/dandiws/CNN-MNIST-pytorch/blob/master/cnn_mnist.py
https://github.com/OpenGenus/CNN-pytorch/blob/main/simple_cnn_model.py

---

6. Train and Validate Your Model

Build your train and validation function (similar to what you had in Lab 5 and 6). Remember to use your metrics in each function to measure the performance of your model at every epoch. Then, plot your train and validation metrics and your train and validation loss. Then build your main function in which you will create the train and validation loop. In your main function you can store your best metric value and save the best model checkpoint. You can also define an early stopping pipeline here.

7. Model Tuning

In this step, you should focus on tuning your model hyperparameters and architecture to improve its performance. Experiment with different configurations such as learning rate, batch size, number of convolutional layers, filter sizes, activation functions, and dropout rates. You may also adjust the optimizer or loss function to see how they affect the training results.

After each modification, retrain and validate your model, then compare the performance metrics (e.g., accuracy, precision, recall, or F1-score) to determine which settings produce the best results. Repeat this process as many times as necessary until you achieve performance metrics that you consider satisfactory. Once you are confident in your model's performance, save your model with your group name.

8. Deliverables

➡ **Lab 7 assignment (deadline 19-11-2025):** only submit your notebook with your base code completed. It is not important if you still haven't reached a good metric value. The goal of this phase is for you to finish writing your entire based code and train and validate your model at least once.

➡ **For your final project (deadline 15-12-2025):** you are required to submit your best model (.pth file) and a written report of no more than six pages summarizing your project work. You do not need to submit your notebook in this final stage.

The report should be structured as a formal manuscript and must include the following sections:

Title: choose a title for your work!

Author names: complete names of your group members.

Introduction: Briefly explain the overall project, its objectives, and the real-world problem you are addressing.

Data Analysis and Pre-processing: Describe the dataset used in your project, including details of the data source, annotations, and structure. Summarize the results of your data analysis and explain your data pre-processing and augmentation methods.

Methodology: Describe the architecture of your model, outlining the key components and layers. Justify your design choices and explain why this architecture is suitable for your task.

Experimental Setup: Provide details about your training configuration, including the loss function, optimizer, number of epochs, learning rate, evaluation metrics, and any other hyperparameters you used. Clearly justify the reasoning behind each choice.

Results: Present your validation results and report your final performance score from the challenge leaderboard. Use tables or figures where appropriate to support your findings.

Discussion: Reflect on your results and discuss why you obtained them. Consider potential reasons for your model's performance, possible improvements, and any limitations of your current approach.

Conclusion: Summarize the key takeaways from your project, highlighting what was achieved and any insights gained from the process.

Make sure your report is well-organized, concise, and professionally formatted. Use clear figures, tables, and captions where necessary, and ensure that all group members' names and student IDs are included on the title page.

Your final project evaluation will be based on your report and the metric your final model achieves on a private test set that we have not shared with you.