ML-based decoding of dot-peen marked Data Matrix codes

Supervisors: Yucheng Lu (80%) & Veronika Cheplygina (20%)
In collaboration with MAN Energy Solutions

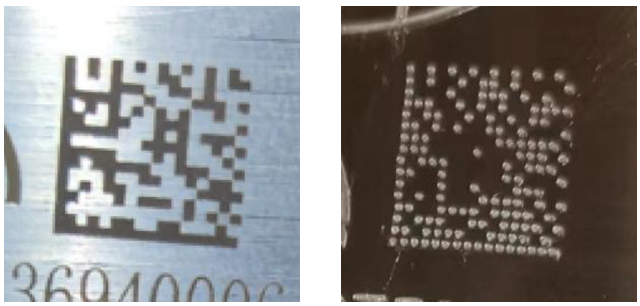
Aidan Stocks — aist@itu.dk

***Abstract*—Dot-peen marked data matrix codes are widely used at MAN Energy Solutions to serialize metal components. Traditionally, decoding these codes has relied on specialized handheld scanners, limiting scalability and automation. This study explores a vision-based alternative by developing a custom decoding pipeline that combines YOLOv11 for region detection with a modified U-Net architecture for segmentation. To improve robustness, a cascaded template matching module recovers missing segmentations, followed by a grid fitting method that maps detected dot-peen marks to their corresponding data matrix indices. Applied to image input, the system achieved a preliminary decode rate of 8.33% on a test dataset. While modest, these results highlight the complexity of the task and motivate the creation of a high-quality labeled dataset and refinement of the pipeline.**

I. Introduction

IN industrial engineering settings, Data Matrix codes (DMCs) are a type of 2-dimensional barcode often marked directly onto machined metal components for serialization. MAN Energy Solutions is a company that requires its manufacturers to serialize engine components, allowing purchasing customers to verify that a MAN-approved supplier produced the component.

Although commonly used DMC scanners like libdmtx[1] or ZXing[2] work in ideal conditions, they often fail when scanning from smartphone cameras inside ship engine rooms or in specific factory settings. Scanning failure can occur due to the use of poor-quality cameras, the DMCs being physically small, the metal component surface being highly reflective, or poor lighting conditions. Most scanners are also not designed to recognize dot-peen marked DMCs, a commonly used and cheaper alternative method of marking components seen in Figure 1b.



(a) Laser **(b)** Dot Peen

Fig. 1: Difference between a laser and dot peen marked DMC.

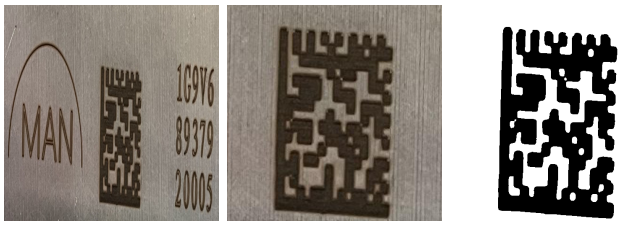
We propose a decoding pipeline that utilizes two machine learning models, followed by a custom decoding process to eliminate reliance on image processing methods from libdmtx. The proposed models are the real-time object detection model YOLO[3] to detect and crop to the DMC area in the image, followed by a U-Net[4] model to segment out dot-peen markings from the image. Thereafter, custom methods using template matching and grid fitting match the located dots to their corresponding underlying data matrix indices, allowing for standard decoding according to the ECC200 standard[5].

The code used in this paper and a README of the exploratory process can be found in this GitHub repository.

II. Related Work

A. Previous Research on Laser-marked DMCs

Previous work (seen in Appendix A) focusing on laser-marked DMCs found that preprocessing the images with YOLO and U-Net before decoding with libdmtx resulted in a 12% increase in decode rates when compared to feeding the raw input to libdmtx. To preprocess an input image, YOLO was used to detect and crop an image to its DMC, followed by a U-Net segmentation model to binarize the DMC modules from the rest of the image. An example of this preprocessing can be seen in Figure 2.



(a) Input Image **(b)** YOLO Crop **(c)** U-Net Binarization

Fig. 2: Result of preprocessing workflow, decodable with libdmtx.

While the paper proves that these machine learning models are applicable to decoding laser-marked DMCs, they failed when applied to images containing dot-peen-marked DMCs. While the U-Net model was still able to segment dot-peen marked DMCs, even accurately segmented dot-peen marked DMCs proved difficult for libdmtx to decode, successfully decoding only one example of a dot-peen marked DMC. This paper aims to address this issue by developing a custom decoding pipeline that leverages previously used model architectures, incorporating additional methods to remove reliance on libdmtx.

Libdmtx's decoding process can be broken down into the following steps:

1) Preprocessing. The input image is converted to grayscale, and noise reduction methods such as binarization are applied.
2) Localization. The image is scanned for edges and patterns that resemble DMCs to find the region of the image containing the DMC.
3) Grid Fitting. The region is divided into a grid of DMC modules, and the grid is adjusted to fit the actual grid of the DMC.
4) Bit Sampling. Each module is sampled to determine whether it is black or white to form the raw bit matrix representation of the DMC.
5) Error Correction. The bit stream is parsed according to the ECC200[5] standard, part of which applies error correction (Reed-Solomon).
6) Data Decoding. The corrected bits are converted into data codewords and interpreted to produce a decoded string.

In our previous work, YOLO and U-Net effectively solved the first two steps for libdmtx. Localization was performed using YOLO, which reduced the processing time for subsequent steps without affecting the decode rate. Preprocessing was performed using U-Net, which segmented and binarized the DMC modules more effectively than libdmtx, resulting in an increased decode rate at the expense of increased overall processing time. We argue that the previously successful use of these two models warranted further exploration of their use in decoding dot-peen marked DMCs.

B. ML-based Barcode Detection

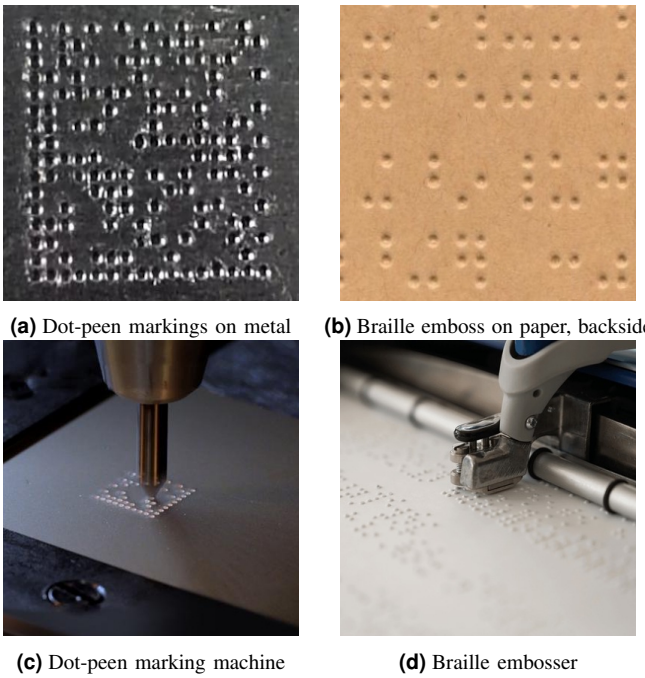
There are extensive amounts of papers that have explored machine learning based approaches to barcode detection.

Deep learning methods have been well-proven[6]–[9] to be effective in detecting classical 1D and 2D barcodes, often achieving state-of-the-art results on benchmark datasets with detection rates reaching nearly 100%. While our scenario differs slightly in that our barcodes are dot-peen marked, we argue that this difference is likely negligible for detecting the DMC area. A study comparing YOLO to similar methods[10] finds that versions of YOLO outperform other object detection models in both accuracy and speed, leading us to continue using YOLO in step 1 of the decoding pipeline.

However, the same research[10] also finds that a common challenge for YOLO and similar object detection models is that they struggle to detect relatively small objects. We therefore avoid using YOLO for detecting individual dot-peen markings and instead adhere to our previous segmentation approach, which is based on a U-Net architecture. Furthermore, other research has demonstrated that the U-Net architecture can be applied in similar industrial settings, such as segmenting defects on steel surfaces[11] or segmenting defects in welds[12].

C. Braille Detection

When examining examples of dot-peen markings, similar features can be observed in Braille dots. Figures 3a & 3b show how the dot-peen marks of a DMC have similar lighting in and around the dots when compared to the backside of Braille dots embossed on a sheet of paper. Furthermore, Figures 3c & 3d show that while the materials differ, the machines used to create the dots both use needle-like objects to press dots into the material.



(a) Dot-peen markings on metal **(b)** Braille emboss on paper, backside

(c) Dot-peen marking machine **(d)** Braille embosser

Fig. 3: Dot-peen marking and braille embossing methods.

Because of these similarities, we argue that previous research in braille detection is likely applicable in this domain. Previous work[13]–[17] in braille detection demonstrates that different model architectures are capable of accurately detecting braille dot locations or classifying braille characters. A literature review of Optical Braille Recognition (OBR) techniques[18] shows that non-model-based segmentation techniques have been commonly used in this task, using classic histogram analysis methods such as Otsu's method[19] for binarization, separating the dot areas from the rest of the image. However, from our previous research, global thresholding methods like Otsu's method typically fail in our domain due to differences in illumination across an image, which are present because of the reflective properties of the materials used. We therefore argue that, because deep learning architectures are applicable for braille detection, and Otsu's method is applicable for braille segmentation on non-reflective surfaces, a non-global thresholding deep learning method, such as U-Net, should be capable of segmenting dot-peen markings on reflective surfaces.

III. Decoding Pipeline

Our proposed decoding pipeline replaces the first four steps of libdmtx when decoding dot-peen marked DMCs:

1) DMC Localization. Using YOLO to detect a bounding box of the DMC in the input image, and crop down to the DMC area.

2) U-Net Dot-Peen Localization. Using U-Net to segment dot-peen marking locations from the DMC.
3) Cascaded Template Matching Dot-Peen Localization. Using a template matching algorithm to find dot marking locations missed by U-Net.
4) Grid Fitting. A custom grid fitting algorithm to map found dot-peen marking locations to their respective Data Matrix indexes.
5) Custom Error Correction. Error correction methods aimed at fixing errors that may occur from previous steps.
6) Libdmtx Decoding.* Rebuilding the DMC to an image and decoding with libdmtx to utilize its standard ECC200 parsing and Reed-Solomon error correction methods.

*Note that while the last step still uses libdmtx, because we fully rebuild a DMC, the earlier steps of libdmtx decoding are effectively bypassed and therefore not used.

Listing 1 shows a more detailed overview of the framework in Python code form.

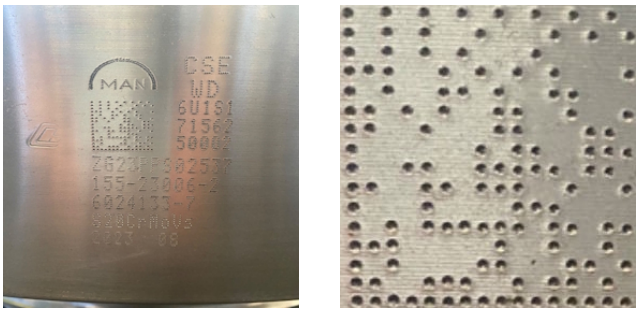
```
# Load to PyTorch Tensor on device
img = load_image_to_device(image_path)

# DMC Localization
img_cropped = yolo_crop(img)

# U-Net Dot-Peen Localization
img_reflect, templates = unet_get_templates(
    img_yolo)

match_thresh = 0.95
while True:
    # Template Matching Dot Localization
    matches = cascade_template_matching(
        img_reflect, templates, match_thresh)

    # Thresh too high or failure to decode
    if len(matches) <= 1:
        match_thresh -= 0.025
        continue
    if len(matches) >= 256:
        return None

    # Grid Fitting
    init_params = estimate_grid_params(matches
        )
    opt_params = optimize_grid(init_params,
        matches)
    grid_pts = generate_grid(opt_params)
    mapped_grid_pts = map_to_DMC(matches,
        grid_pts)

    # Transform to DMC indexes
    dmc_pts = inverse_grid_transform(
        mapped_grid_pts, opt_params)
    dmc_matrix = to_dmc_matrix(dmc_pts)

    # Custom Error Correction
    dmc_matrix = error_correct(dmc_matrix)

    # Attempt Decode
    decoded_data = pylibdmtx(dmc_matrix)

    # Retry with lower thresh on failure
    if not decoded_data:
        match_thresh -= 0.025
        continue
    else:
        return decoded_data
```

Listing 1: A simplified version of the decoding pipeline, outlining the structure of the decoding pipeline function written in Python.

Detailed descriptions of each step are provided in this section. The usage of the trained YOLO & U-Net models *are* described in this section, but information on their architectures and training processes is in the later Models section.

A. Crop to DMC

The first step in the decoding pipeline is to localize the DMC area within the image. By localizing the DMC, we can crop down to it, eliminating most of the pixels from the image. With accurate cropping, the processing time for subsequent steps will be significantly reduced due to the lower image size, and the removal of noise around the DMC area should also enhance the consistency of later steps. Figure 4 shows an example of ideal cropping to a DMC area. Ideally, the crop should be tight around the DMC to remove noise, but not so tight that information on the dots themselves is lost.



(a) Input (size 1280x1706) **(b)** Output (size 276x282)

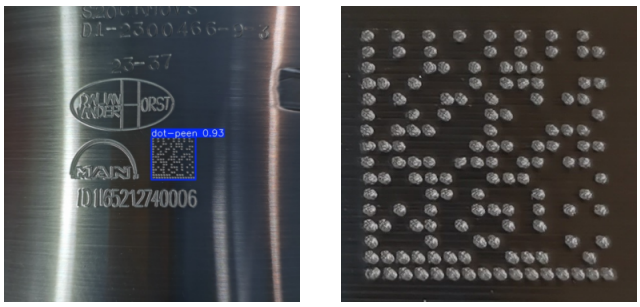
Fig. 4: Example of an ideal crop from input image. The resulting crop is $\sim 3.6\%$ the size of the original image.

Following the successful usage of YOLO in our previous work, the Ultralytics yolov11n[3] object detection model was chosen for this step of the pipeline.

YOLO is a real-time object detection model capable of predicting bounding boxes around specific objects within an image. By using a YOLO model trained for creating bounding boxes around DMCs, we can effectively take an input image containing a DMC, feed it to YOLO, and use the YOLO output to crop down to the DMC area. By utilizing a real-time object detection model and the lightest version available from Ultralytics, we remain closer to MAN's use case of employing a decoding pipeline on mobile phones.

To optimize the decoding process, the YOLO model is loaded, its layers are fused, and it is switched to evaluation mode outside of the decoding pipeline. The fusing method, pre-written by Ultralytics, merges the convolution and batch normalization layers into a single layer to optimize inference time. For use in an application, this loading would typically occur during initialization.

In the decoding pipeline, YOLO inference is performed on a version of the input image resized to the standard YOLO size of 640x640. To preserve the original image quality for later pipeline steps, the bounding box outputs of YOLO are used to crop the images to their original sizes. Because later steps in the decoding pipeline rely on DMCs being visible in their entirety, the cropping process is done with a padding of 1% to reduce occurrences of YOLO predicting overly tight bounding boxes. Figure 5 shows an example of a YOLO crop from our trained YOLO model.



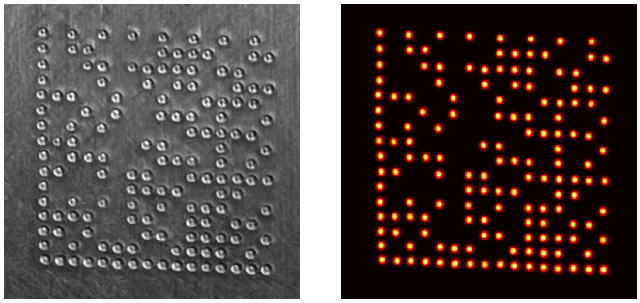
(a) YOLO Detection **(b)** YOLO Crop with 1% padding

Fig. 5: Example YOLO detection on input image with resulting crop

The cropped version of the original image is passed on to the next step.

B. U-Net Dot-Peen Localization

To decode the DMC, we need to know the locations of most dot-peen marks. This step in the decoding process utilizes the trained U-Net segmentation model to generate a heatmap of potential dot-peen locations. Figure 6 shows an example of the ideal heatmap for U-Net to produce from an input image. Notice how the heatmap dots are uniform in size and brightness, making them suitable for extraction.



(a) Input to U-Net **(b)** Resulting U-Net output

Fig. 6: Example of ideal U-Net segmentation from input image

Similarly to YOLO, the U-Net model is loaded and switched to evaluation mode outside of the decoding pipeline. Inside the decoding pipeline, the cropped image passed on from the YOLO step is prepared for input to U-Net by resizing to 384x384, converting to grayscale, and applying a Single-Scale Retinex (SSR)[20] to flatten illumination across the image.

Retinex theory attempts to model how humans perceive colors and brightness under varying lighting conditions. It aims to achieve more consistent colors in an image by separating the illumination from the reflectance, where illumination is defined as the lighting that hits a surface, and reflectance is defined as the inherent color and texture of the surface itself. In short, the goal of Retinex is to estimate the true color of a scene, independent of lighting changes.

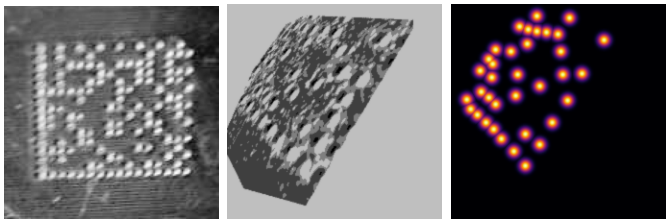
We employ SSR with a standard deviation of 50 used in the Gaussian blur, matching the SSR used during U-Net training as described in section IV-B.

From the U-Net output, a strict selection process is performed to ensure only valid dot-peen markings are extracted:

1) Heatmap pixel values at a threshold of 0.1 to separate most dots.
2) Watershed algorithm applied to separate any close heatmap blobs not separated in the previous step.
3) Contours of heatmap blobs analyzed in their area and circularity.
4) Blobs sorted by their area in descending order.
5) Blobs with areas higher than 1.5 median blob area are rejected.
6) Blobs with circularity below 0.5 are rejected.

We reject blobs with overly large areas and low circularity to reduce the chance of extracting false-positive U-Net detections.

In some of the images provided by MAN, it can be seen that the dot-peen marking process has produced overlapping holes in the metal. This overlap causes some ground-truth heatmaps generated for the U-Net training process to also overlap, which U-Net then replicates in its predictions. Figure 7 shows an example of a DMC with overlapping dot-peen marks with its accompanying ground-truth heatmap. To extract templates from DMCs with this overlap, algorithms such as the Watershed algorithm are required.



(a) Original image **(b)** Augmented patch **(c)** Ground Truth

Fig. 7: Example DMC with overlapping dots.

We use the Watershed function from scikit-image, which is based on the Watershed algorithm described by Vincent and Soille[21]. For our use case, we utilize the inverse of our heatmaps, treating the centers of blobs as valleys and using these centers as markers. By performing watershed flooding on the inverse heatmap and using these markers, we can estimate the boundaries between blobs where water from different valleys meet, allowing us to separate them.

While U-Net has the potential to produce a heatmap with all dots present, experimentation has shown that it often misses some and can sometimes predict false positives. To compensate for this, we only find the first three of these dots in this step, and the rest in a later step.

With perfect U-Net output, we could extract all dot-peen marks from the image in this step. However, through experimentation, it became clear that the trained U-Net model often misses some dots and sometimes produces false dot-peen detections from other dots present in the image. Because of this, we only rely on U-Net to find three dot locations. To extract three dot-peen marks from the U-Net output, the heatmap is analyzed to find valid dot areas, and the accompanying regions of the cropped image are extracted for use as templates in the next step of the pipeline.

The three largest blobs that meet the selection criteria are extracted with a 10% padding applied to account for overly tight blob boundaries. We select the largest blobs as they indicate strong predictions from the U-Net model. If no blobs meet the selection criteria, the decoding is considered a failure.

Once extracted, the templates are passed on for use as templates in the cascaded template matching process.

C. Cascaded Template Matching Dot-Peen Localization

After extracting some dot-peen markings from the U-Net output, the locations of the rest of the dot-peen markings are found using template matching.

Template matching is the process of finding where in a given image smaller images similar to a given template are. Figure 8 shows some typical applications of template matching.

(a) Mario coin template

(b) Mario coin detections

(c) Messi face template

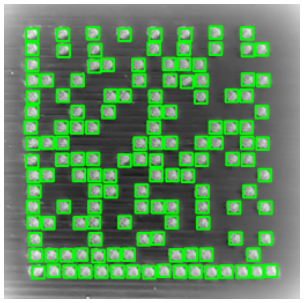
(d) Messi face detection

Fig. 8: Example uses of template matching.

In this step of the pipeline, our goal is to use our extracted dot-peen marking images as templates and run template matching to find other dot-peen markings. Figure 9 shows an example of an ideal template matching result on a DMC.

Template matching works by sliding the template across the image, and at each position calculating how similar the region is to the template. Because we are interested in finding

(a) Dot template



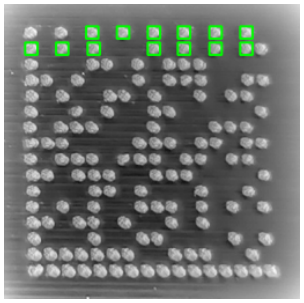
(b) Dot detections

Fig. 9: Example uses of template matching.

an unknown number of template matches, we cannot simply select some number of matches with the highest similarity scores. Instead, we can accept all matches above a predefined threshold. Due to the later steps used for decoding the DMC, it is critically important to avoid false positives from the template matching process. We therefore start with a strict matching threshold of 0.95. If a later step in the decoding process fails due to not enough dot-peen markings being found, this threshold is reduced by 0.025, and the decoding is repeated from this step.

However, dot-peen markings across a DMC can vary in similarity due to lighting conditions and deformations in the manufacturing process, causing a single run of template matching with a strict threshold to only find a few similar dot-peen markings. Figure 10 shows an example of how the ideal template matching shown in Figure 9 performs with the strict threshold of 0.95.

(a) Dot template



(b) Dot detections

Fig. 10: Template matching with threshold 0.95.

To find our desired matches while maintaining a high matching threshold, we propose a cascaded template matching process, which repeatedly runs template matching with a high threshold, utilizing new matches in subsequent template matching. By running this repeatedly until no more matches are found, we can potentially find enough of the dot-peen markings to decode the DMC, while avoiding false positives.

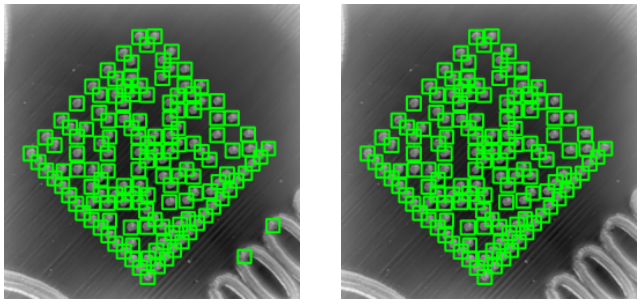
Concretely, the cascaded template matching is performed in these steps:

- Get initial template list from U-Net template extraction
- Initialize unused template list as a copy of initial templates
- Initialize used template list (empty)

- While there are unused templates:
  - Perform template matching with unused templates
  - Remove used templates from the unused templates list
  - Add used templates to the used template list
  - Add new matches not overlapping with existing templates to the template list
- Remove overlapping templates from the template list
- Remove outlier matches

Overlapping templates are removed using non-maximum suppression with an IOU threshold of 0.2. Non-maximum suppression is a method for removing bounding boxes that have overlap above a given threshold, where the bounding box with the lower score is rejected. In our case, if two template matches overlap above the threshold, the template match with the lower similarity score is rejected.

During the manufacturing process, MAN marks their logo and DMC serial numbers next to the DMCs in case the DMCs were to be damaged. If the orientation of the DMC in the input image is poor (e.g., 45 degrees; a diamond shape), even a perfect YOLO crop may still have some dot-peen markings present from the logo or serial number. An example of this false positive matching can be seen in Figure 11a. Because the dots look similar, the U-Net and template matching processes will fail to differentiate between DMC dot-peen markings and non-DMC dot-peen markings. To account for these false positives, the clustering method DBSCAN[22] is used to cluster the points belonging to the DMC together. DBSCAN is designed to cluster densely packed points together, treating points not part of any dense clusters as outliers. We argue that since DMCs are designed to be densely packed, the DBSCAN clustering algorithm is a good choice when compared to alternatives. Furthermore, some researchers[23] suggest that despite the age of DBSCAN, it still performs on par with more recently developed methods when appropriate parameters are chosen. Figure 11b shows how DBSCAN removes the aforementioned false positives.

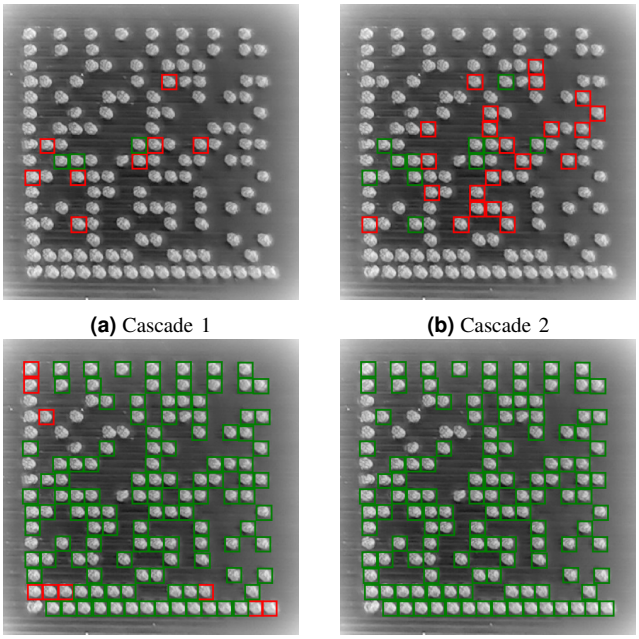


(a) Without DBSCAN **(b)** With DBSCAN

Fig. 11: A comparison of template matching with or without DBSCAN outlier removal.

The parameters chosen for DBSCAN are mostly the default scikit-learn parameters, except for ”eps” and ”min_samples”. ”eps” defines the maximum distance (in pixels) between two points for them to be considered part of the same neighborhood. Since all images at this stage of the pipeline are 384x384 pixels, we set the eps to 50 to ensure that only points that are relatively far apart (more than 50 pixels away) are treated as outliers. min_samples specifies the minimum number of points required to form a dense region. Based on visual inspection of the DMC structures, meaningful clusters typically contain at least 3 points, and we therefore set the min_samples to 3.

An example of the first and last couple cascades in the cascaded template matching process is shown in Figure 12. Note that in this example, the first cascaded template matching with a threshold of 0.95 failed.



(a) Cascade 1 **(b)** Cascade 2

(c) Cascade 6 **(d)** Cascade 7

Fig. 12: Example of cascaded template matching with threshold 0.925. For each iteration, green boundaries denote used templates, and red boundaries newly found unused templates. This example contained sufficient dot-peen markings to be successfully decoded.

On completion of the cascaded template matching process, the center coordinates of the matches are passed on for use in the grid fitting process.

D. Grid Fitting

With the dot-peen marked locations acquired, the remaining task for decoding the DMC is to map each point to its respective index in the underlying DMC matrix.

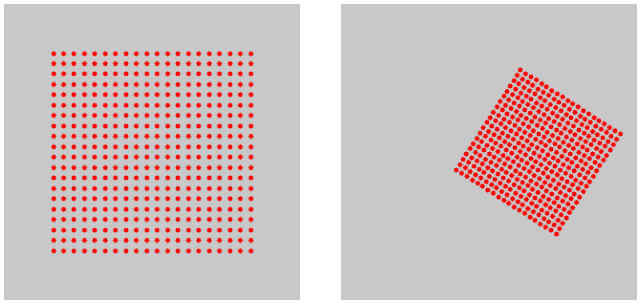
To accomplish this, we formulate the task into a minimization problem. From the MAN dataset, we know the points belong to either a 14x14 or 16x16 grid. We can therefore create a complete grid of a higher size and alter its parameters to fit over the matched templates acquired from cascaded template matching. The grid fitting process is written in a way where grid sizes of at least two dimensions higher than the DMC size are preferable. As higher grid sizes are more computationally expensive, we choose a balanced grid size of 20x20, but the grid size can be increased for use in DMCs of higher dimensions if desired.

We define the grid parameters as:

- x0, y0: center (x,y) coords of grid placement

- sx, sy: horizontal/vertical spacing between gridlines
- theta: rotation of grid (radians)

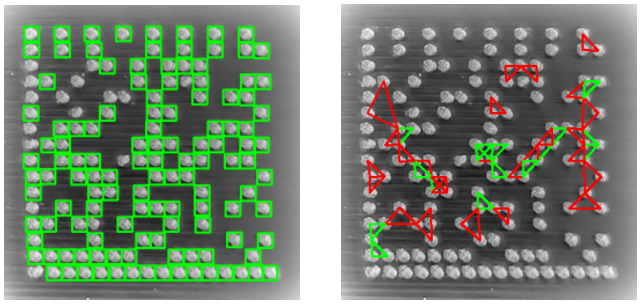
With these parameters, we can fit a grid over the template matches in any location of the image, with different horizontal and vertical spacing, and with any orientation. Note that the vertical and horizontal spacing applies to all rows and columns; there are not vertical and horizontal parameters between every row or every column. Figure 13 shows two examples of grids generated with different parameters.



(a) [192, 192, 13.5, 13.5, 0] **(b)** [256, 192, 8.1, 8.1, 1.0]

Fig. 13: Example grid generations with different [x0, y0, xs, ys, theta] parameters on blank image of size 384x384. Compared to the left grid, the right grid has been shifted to the right, rotated, and its grid spacing tightened.

Experimentation showed that using some standard starting parameters, such as those seen in Figure 13a, led to poor grid fitting. Instead, the template match coordinates calculated in the previous step of the pipeline can be used to estimate better starting parameters. To estimate the center coordinates of the grid, the centroid of the template matches was used. Estimating the spacing and theta parameters involves a process that searches for ”L shapes” in the template matches. Figure 14 shows the L shapes found for a DMC, illustrating which are desired for use in parameter estimation and which are not.



(a) Template Matches **(b)** L Shapes

Fig. 14: Example of desired L shapes from template matches for use in parameter estimation. Green is desirable, red is undesirable.

From the template matches, we perform the following steps to find valid L shapes:

- Initialize list to store L shapes
- For each point...
  - Calculate its 2 closest neighbors
  - Calculate the 3 possible L shapes
  - Reject L shapes that do not form 90-degree angles
  - Reject L shapes with distances that differ by over 5% of image size
  - Append remaining L shapes to the L shapes list
- From stored L shapes, calculate the modal horizontal distance from horizontal distances rounded to the nearest integer
- From stored L shapes, calculate the modal vertical distance from vertical distances rounded to the nearest integer
- Remove L shapes with horizontal or vertical distances higher than the respective modal distance
- From remaining L shapes, calculate the modal orientation away from pure vertical, rounded to the nearest degree
- Remove L shapes with orientations rounded to the nearest integer that differ from the modal orientation

With the rejection process, we ensure that the remaining L shapes must form close-to-90-degree angles and have equally short distances. We also ensure that we only keep the most commonly occurring L shapes by using the mode. We avoid using the mean or median due to the presence of many outliers. Because the distances and orientations are continuous variables, we round distances to their nearest integer (equal to pixel size) and round orientations to their nearest degree so that modal calculation is valid. Because the DMC structure belongs to a grid, the most commonly occurring remaining L shapes are our desired green Ls, which is why we use the modal distances and orientations to reject other L shapes. With the remaining L shapes, the average horizontal and vertical distances, along with the average orientation, are used as the initial starting parameters for the grid.

Figure 15a shows an example of a starting estimate for grid fitting calculated using the template matches from Figure 14a.

After a starting grid is found, the parameters are optimized based on a cost function using the summed distances between template matching coordinates mapped to grid points. The mapping process is covered later in this section.

During experimentation, it was found that optimizing the grid for different parameters in the following order performed better than optimizing for all at once:

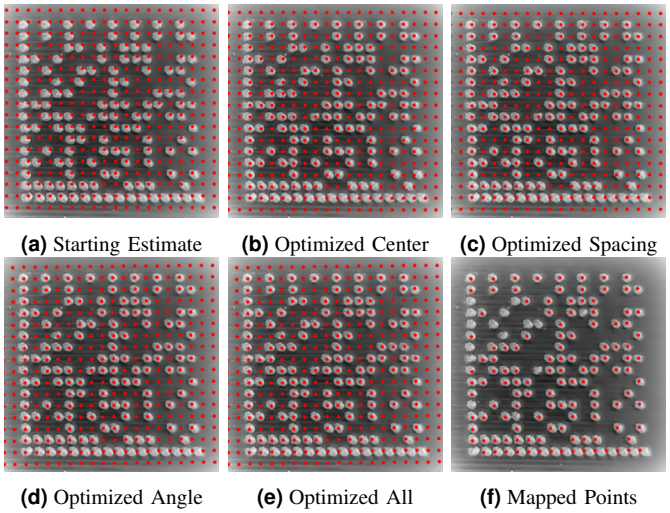
- Optimize for center (x0, y0)
- Optimize for spacing (xs, ys)
- Optimize for angle (theta)
- Optimize for all (x0, y0, xs, ys, theta)

Figures 15b to 15e show an example of the grid fitting process at different optimization steps.

With the grid fitted to the template matches, we can find which template match coordinates map to which fitted grid points. Specifically, we want to map every template match to a grid point, minimizing the summed distances between mappings. Template matches cannot share grid points, and unmapped grid points are discarded.

This problem is effectively the classical assignment problem[24], which can be solved in polynomial time using well-established algorithms. We use SciPy’s linear_sum_assignment function[25], which implements a modified version of the Jonker-Volgenant algorithm[26].

Figure 15f shows the resulting mapped points after the grid fitting process. Note that in this example, even though the



(a) Starting Estimate **(b)** Optimized Center **(c)** Optimized Spacing

(d) Optimized Angle **(e)** Optimized All **(f)** Mapped Points

Fig. 15: Example showing entire grid fitting process from initial estimate to mapping. Template matches used for fitting from Figure 14a

template matching from 14a missed some dot locations, the mapping process succeeds and the points are later decodable.

Once the template matches are mapped to their corresponding grid points, the optimized parameters can be used to transform the grid points back to their original DMC matrix space. Figure 16a shows how the mapped grid achieved in Figure 15f is transformed back to its DMC space.

However, note that the missing dot peen marks are still present in the rebuilt DMC. There are also empty rows and columns in the DMC due to how we generated the grid to be 20x20. The mapped points are passed to the error correction method to reduce some of these types of errors.

E. Custom Error Correction

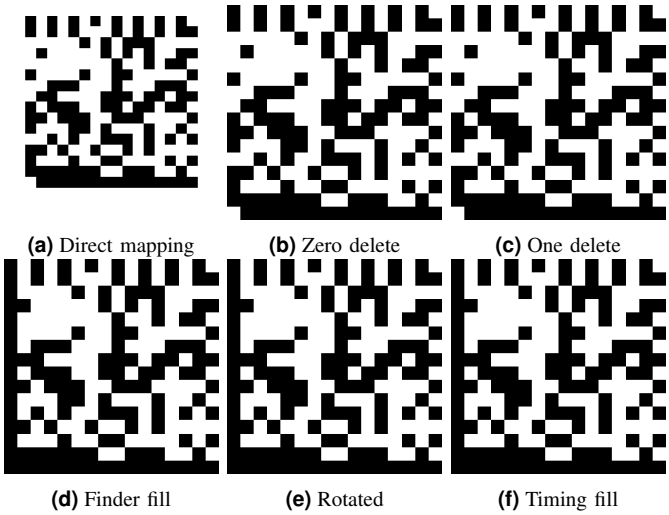
With the template matches mapped to DMC matrix indexes, it is possible to rebuild and decode the DMC.

However, many errors can occur during the pipeline due to uncaught edge cases in the implementation of different methods, poor image quality, or even due to a manufacturing process error in the DMC. While some mistakes are too significant to be resolved in this step, many minor errors can be addressed.

Below is every post-processing step done to the rebuilt DMC:

1) Remove empty rows & columns
2) Remove rows & columns containing single modules
3) Estimate the finder pattern from the DMC edge row/-column pair with the highest count of ones
4) Fill Finder pattern
5) DMC rotated to standard orientation according to finder pattern
6) Unexpected timing pattern modules shifted away from the edge
7) Fill timing pattern

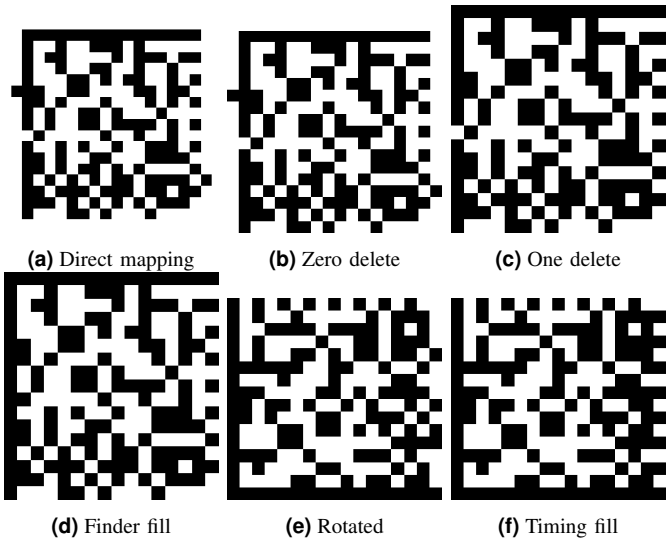
Figure 16 shows each post-processing step with the example mappings from Figure 15f. Figure 17 shows the error correction steps on a more difficult-to-decode DMC, where every error correction step provides some change to the DMC.



(a) Direct mapping **(b)** Zero delete **(c)** One delete

(d) Finder fill **(e)** Rotated **(f)** Timing fill

Fig. 16: Example showing entire custom error correction process using mapped points from 15f. This example is decodable.



(a) Direct mapping **(b)** Zero delete **(c)** One delete

(d) Finder fill **(e)** Rotated **(f)** Timing fill

Fig. 17: Example showing custom error correction on poorly mapped DMC points. This example is not decodable.

With the custom error correction complete, the matrix is passed on to the final decoding step of the pipeline.

F. libdmtx Decoding

Finally, with the underlying DMC matrix rebuilt, it is possible to decode. Because the Python implementation of libdmtx (pylibdmtx[27]) does not offer functionality to perform the decoding process directly from the DMC matrix, the rebuilt DMC is converted to an image and decoded as standard image input. A padding of 2 pixels is applied around the DMC before converting to an image following ECC200 specifications, which state that the quiet zone of a DMC should be larger than the size of individual modules.

Figure 18 shows a padded DMC image fed to libdmtx with the resulting decoded string.

Fig. 18: Example input to libdmtx. The dashed red boxes denote the added quiet zone; they are not present in the image fed to libdmtx. The resulting decoded string is "1D1I65212740006".

IV. Models

A. YOLO

YOLO (You Only Look Once) is a fast, single-stage object detector, typically used in real-time applications due to its high speed. At a high level, it works by dividing an image into a grid, and for each grid cell, it predicts bounding boxes and class probabilities of possible objects present in the cell. It performs both calculations simultaneously, which is why it is a single-stage process. Figure 19 shows these predictions.

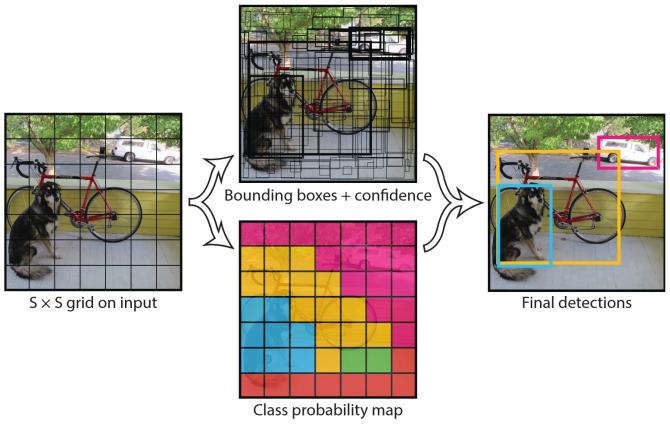


Fig. 19: YOLO from input to output. Image sourced from the original YOLO paper[28]. The original YOLO system divides the image into an SxS grid and for each grid cell predicts B bounding boxes, confidence scores for those boxes, and C class probabilities.

After predictions are made, non-maximum suppression is used to remove multiple detections of the same objects.

The original architecture of the YOLO model consists of 24 convolutional layers, followed by two fully connected layers. Figure 20 shows this architecture.

Subsequent YOLO versions build on the core version of YOLO, altering and optimizing it in various ways. YOLO11, used in this paper, is one of the most recent versions released by Ultralytics in 2024. Along with changes introduced in previous versions, v11 adds specialized feature extraction blocks (C3K2, C2F, C3K) and additional features to enhance small object detection while maintaining real-time speeds.

To prepare the images provided by MAN for YOLO training, the relevant x, y, w, and h bounding box values for DMC locations in the images were required. Roboflow is

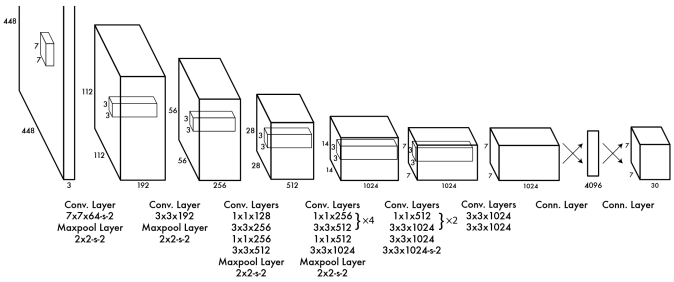


Fig. 20: The original YOLO architecture. Image sourced from the original YOLO paper[28].

a website offering various tools related to computer vision, one of which is their image annotation tool, which was used for annotating bounding boxes over DMCs in each image. Using Roboflow, we also train/val/test split the 206 images into splits of 126/30/50 (.61/.15/.24). The splits are the same as those used in our previous work, allowing for comparison of results, with some new images obtained from MAN added to the training split. The images are augmented using Roboflow to triple the size of the training and validation sets, with augmentations including rotation (± 45 degrees) and shearing (± 10 horizontal, ± 10 vertical).

As the dataset contains both laser- and dot-peen-marked examples, during annotation, we also label the class as either laser- or dot-peen-marked DMCs to see if YOLO can correctly differentiate between the two. While not specifically useful for this task, it would be helpful if combining the methods used in this paper with the techniques used in previous work, to create a pipeline capable of detecting the type of DMC in step 1, and branching to use methods from either this paper or previous depending on the DMC class. Furthermore, we wish to maximize our available training data for the model by including both types of DMCs in the training data. While we will see the results of YOLO differentiating between the classes in the results section, in the actual pipeline, we do not use the class labels and choose the bounding box with the highest confidence.

Our previous work in training YOLO11n to recognize DMCs demonstrated that fine-tuning the Ultralytics pre-trained model yields better results compared to training from scratch; therefore, this process was repeated for the purposes of this paper. Mostly default Ultralytics training parameters were used, with only the patience value reduced from 100 to 10 to minimize unnecessarily long training times, as the best mAP50-95 scores were consistently achieved around the 27th epoch.

B. U-Net

We effectively implement a slightly modified version of U-Net in PyTorch[29]. Modifications to the original architecture include adding batch normalization layers after each convolutional layer, a sigmoid output activation layer, and using padding instead of cropping. The original U-Net architecture is illustrated in Figure 21.

To prepare the images provided by MAN for U-Net training, the relevant x, y coordinates and sizes of DMC dots were

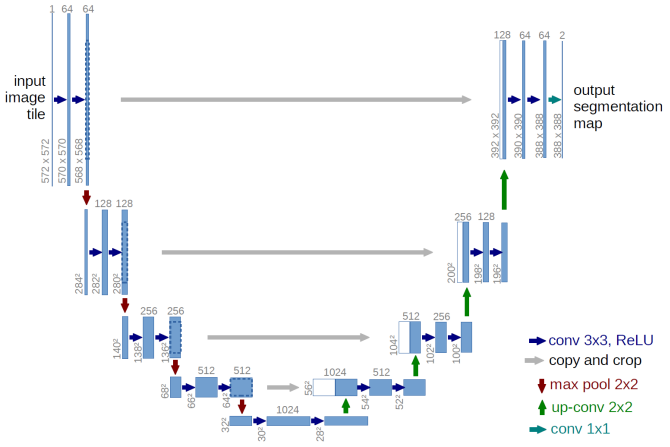


Fig. 21: Original U-Net architecture. Image sourced from the original U-Net paper[4]. The depicted architecture has been modified in this paper by adjusting the input image size to 384x384, adding batch normalization layers after each convolutional layer, and incorporating a final sigmoid head at the end.

required. Roboflows' annotation tool was used to annotate the x,y coords of dot centers, and manual crops of dot-peen marks were saved as separate images for use in calculating dot sizes. Once annotated, accompanying heatmaps for each image were generated using the x,y coords and manual crops of dot marks to generate differently sized Gaussian distributions at specified locations. To more closely match the training images to the expected outputs of YOLO, a crop of the DMC is done during dataset generation by cropping slightly outside of the boundaries of the annotated dot locations. Figure 22 shows an example of one of these crops with annotated dot locations.

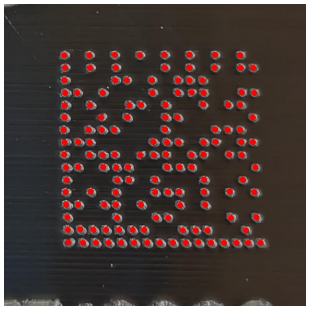
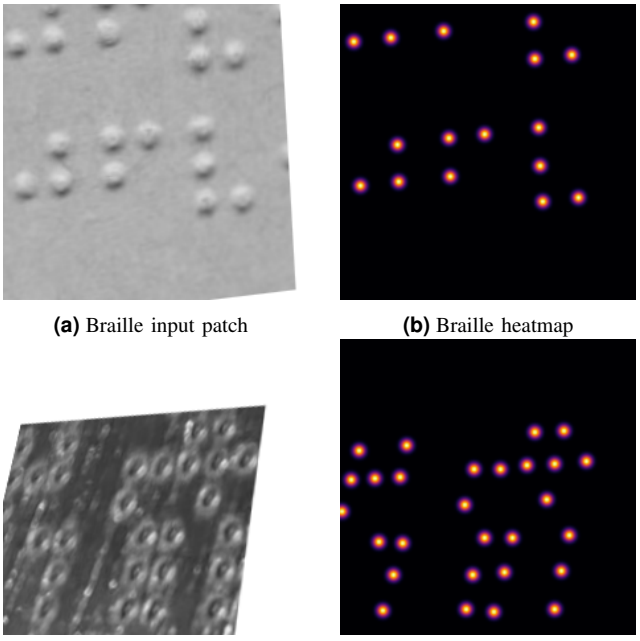


Fig. 22: Example crop generated for the U-Net training dataset. The dots (red) show locations of the annotated dot-peen centers.

Due to the similarity of their problems, the training data could be increased by including images from a dataset initially created for braille detection[13]. This dataset contains 114 images with annotations for bounding boxes of individual braille dots, which were used to generate the accompanying heatmap images required for U-Net training. As seen in Figure 3, when comparing dot-peen markings made on metal to braille dots made on paper, many similarities present themselves. Due to these similarities, we supplement the dataset provided by MAN with the Braille images.

To further increase the amount and variety of images used in training, both the Braille and MAN datasets were split into patches of size 512. Figure 23 shows both an example of Braille and MAN image patches with their accompanying heatmaps.



(a) Braille input patch **(b)** Braille heatmap

(c) MAN input patch **(d)** MAN heatmap

Fig. 23: Example of U-Net training patches with accompanying heatmaps.

After splitting the images into patches, we are left with 137 MAN patches and 2280 Braille patches. This significant difference occurs due to a significantly lower number of dot markings present in a single DMC compared to a page of Braille text. To ensure equal contribution from both datasets, the MAN patches were oversampled to match the number of training images in the braille dataset. We choose to oversample rather than undersample to maximize the overall amount of training data available. Equal contribution is preferred so that the Braille dataset aids in training by providing much more data, without dominating the dataset and possibly overfitting the U-Net model to the wrong domain.

After patch generation, a dataloader was established to perform various augmentations on patches during training. The applied augmentations can be seen in Table I.

Type	Augmentation	Prob.	Parameters
Shape	Horizontal Flip	50%	
	Vertical Flip	50%	
	Random Zoom	100%	scale (0.8 - 1.2)
	Random Crop	100%	to patch size 512
	Random Affine	100%	angle (-45 to +45), translate (0.9 - 1.1), scale (0.9 - 1.1), shear (-10 to +10)
	Random Perspective	50%	distortion scale 0.2
Color	Color Jitter	100%	brightness 0.5, hue 0.3
	Random Invert	50%	
	Random Posterize	50%	bits 2
	Random Solarize	50%	threshold 192/255
Other	Gaussian Blur	100%	kernel size (3, 3), sigma (0.1 - 2)

TABLE I: Augmentations applied to patches for U-Net training, with probabilities and parameters used.

To avoid distorting the heatmaps, heatmap generation occurred post-augmentation. As a last step before U-Net inference, a retinex algorithm is applied to flatten illumination across the image, with a standard deviation of 50 in the Gaussian blur used to estimate the illumination component.

A relatively simple training loop was performed with a train/val ratio of 0.8, a batch size of 8, a learning rate of 0.001, and 30 epochs. The PyTorch Adam optimizer was used, and losses were calculated with the PyTorch L1 loss criterion.

V. Results

A. YOLO

Table II shows performance metrics of the trained YOLO model for different splits of the MAN dataset.

Split	Class	Count	Precision	Recall	mAP50-95
Train	all	378	0.963	0.941	0.786
	dot-peen	**207**	**0.946**	**0.934**	**0.789**
	laser	171	0.979	0.947	0.784
Val	all	30	0.838	0.762	0.609
	dot-peen	**21**	**0.888**	**0.857**	**0.78**
	laser	9	0.789	0.667	0.438
Test	all	50	0.82	0.64	0.688
	dot-peen	**36**	**0.958**	**0.637**	**0.717**
	laser	14	0.681	0.643	0.66

TABLE II: Performance of YOLO on each split from the annotated MAN dataset. Count denotes the number of samples for specific splits and type of DMC.

B. U-Net

We compare the heatmaps generated by U-Net for the non-augmented training images to their corresponding ground-truth heatmaps. Here, we evaluate the U-Net model directly. For the performance of the U-Net model in the pipeline, refer to the Pipeline section.

Due to the high time requirement for annotating individual dot locations in images, we only annotated the training split from MAN. We can therefore only calculate our metrics on the training set. Table III shows metrics calculated on the training set.

MSE	MAE	SSIM	PCC
0.0115	0.0272	0.7825	0.4065

TABLE III: Performance of U-Net on dot-peen marked DMCs from the training split of the MAN dataset. MSE = Mean Squared Error. MAE = Mean Absolute Error. SSIM = Structural Similarity Index. PCC = Pearson Correlation Coefficient.

The metrics are calculated by comparing the pixels of heatmaps generated by U-Net with the ground-truth heatmaps. To avoid flattening the images for PCC calculation, PCC is calculated for each row and column and averaged. This helps preserve some spatial structure for the PCC calculation, as opposed to calculating PCC on the flattened versions of the heatmaps.

C. Pipeline

We establish two baselines for comparison with the proposed decoding pipeline. The first baseline involves feeding the input image directly to libdmtx. The second baseline consists of cropping the image using YOLO and then feeding the cropped image to libdmtx. The second baseline is a common improvement[30] to DMC decoding. Decode rate of the pipeline and baselines in different dataset splits can be seen in Table IV

Split	Count	Libdmtx	YOLO + libdmtx	Pipeline
Train	66	0%	0%	**21.21%**
Val	21	0%	0%	**9.52%**
Test	36	0%	0%	**8.33%**
All	123	0%	0%	**15.45%**

TABLE IV: Decode rate of pipeline compared to baseline decode rates. Count denotes the number of samples present in each split.

The failure rate of each step of the pipeline for each split can be seen in Table V.

Split	Pipeline Step	Rate
Train	DMC Localization	15.15%
	U-Net Dot-Peen Localization	53.03%
	Cascaded Template Matching Dot-Peen Localization	6.06%
	Grid Fitting	4.55%
	Custom Error Correction	N/A
	Libdmtx Decoding	0%
Val	DMC Localization	23.81%
	U-Net Dot-Peen Localization	38.1%
	Cascaded Template Matching Dot-Peen Localization	14.29%
	Grid Fitting	14.29%
	Custom Error Correction	N/A
	Libdmtx Decoding	0%
Test	DMC Localization	25%
	U-Net Dot-Peen Localization	38.89%
	Cascaded Template Matching Dot-Peen Localization	13.89%
	Grid Fitting	11.11%
	Custom Error Correction	N/A
	Libdmtx Decoding	0%

TABLE V: Failure rate of specific pipeline steps. The rate calculation excludes successfully decoded images. Custom error correction method is excluded due to its nature of reducing errors from other methods.

The failure rate statistics are based on manual analysis of which step in the pipeline is the first point of failure for a given image. For example, if the DMC localization step for a given image partially cuts off the DMC, that step is considered the point of failure, and no others. If the cascaded template matching process received good templates from U-Net but produced false-positive detections, the grid fitting process is not considered a failure, but rather the cascaded template matching process.

VI. Discussion

A. Overall

Examining the results in Table IV, it becomes clear that the baseline methods, which have decoded no examples, are inadequate for decoding dot-peen marked DMCs. The proposed pipeline, on the other hand, achieves a low decode rate of 15.45% on all dot-peen marked DMCs provided by MAN.

While the decode rate of the proposed pipeline is low, considering that the alternative is a decode rate of zero, we believe the methods used are a step in the right direction. However, we still feel that the low decode rate is too unreliable for use as is.

For the remainder of the discussion, we cover why the current decode rate is too low for use as-is, analyze the reasons for the low decode rate with recommendations for improvements and/or alternative approaches, and discuss some possible applications for the methods in other areas.

B. Decode-Rate & Speed Tradeoff

For MANs' use case, an ideal DMC decoding pipeline should have high speed and a high decode rate, so that users attempting to scan with their smartphones can obtain the encoded data faster and more reliably than by manually typing the serial number imprinted beside the DMC.

In practice, when decoding from a smartphone camera feed, achieving such a decoder relies on a balance between decode rate and speed. This is due to the nature of images acquired from smartphones, where a frame may be blurry or out of focus in one moment compared to another. We found in our previous work that many of the images provided by MAN contain out-of-focus DMCs, or DMCs that are blurry due to the motion of the camera. By having a high processing speed, the likelihood of capturing a high-quality image increases, potentially leading to an overall improvement in the reliability of decoding compared to more complex methods with lower processing times.

Research into the various applications of all barcode types reveals that different balances of decode rate and speed are considered, depending on the scanner's application. For example, in a paper focusing on the scenario of automatic warehouse sorting[31], scanning QR codes located on boxes moving on high-speed conveyor belts naturally requires a higher focus on speed, as boxes will eventually move out of view of the still scanning camera. Another paper focusing on creating visually appealing QR codes[32] naturally places less emphasis on decoding speed, ensuring that the code remains reliably decodable at near real-time speeds. A paper focusing on decoding multi-colored QR codes for mobile phones[33] measures the average time to process each frame, but does not go into depth on the trade-off between decode-rate and speed.

To better understand the importance of this balance and enable data-driven decisions on decoder complexity versus speed, we propose constructing a dataset from smartphone camera feeds. With such a dataset, frames can be extracted along with timestamps, allowing, for example, the measurement of processing time per frame and the simulation of decoding pipelines on sections of video, thereby further bridging the gap between theoretical use and practical use.

While the DMC localization method used in this paper runs at real-time speed, both the U-Net and cascaded template matching methods do not. In future research, when real-time dot-peen marking localization methods are established, an investigation into establishing a balance between decode rate and processing time should be made.

C. Points of Failure - U-Net

To investigate which methods need improvements, we refer to the failure rates of each step shown in Table V. From the table, it is evident that the U-Net dot-peen localization method is the most significant bottleneck in the pipeline.

By looking at the U-Net metrics in Table III, we can see that while some metrics are relatively ok (MSE, MAE, and SSIM), PCC only shows a slight correlation of ~ 0.4. From visually inspecting the produced heatmaps from U-Net, it becomes clear that the U-Net model produces very clear heatmaps for certain types of DMC markings, and fails to produce heatmaps for other types. The model likely performs well for examples of DMCs that are more similar to braille dots, because half of the images the model has been trained on include braille dots. Furthermore, upon inspecting the ground truth heatmaps presented to U-Net, it can be observed that the method used to specify the size of the heatmaps does not produce accurately sized dots for all examples.

When analyzing the U-Net heatmap outputs and subsequent extracted templates, it is seen that two main points of failure occur. Firstly, as previously discussed, for some images, the U-Net model fails to segment any dots accurately. In other images, while U-Net segments some dots accurately, the template extraction method fails to extract three dot-peen marks in their entirety, resulting in one or more extracted templates containing partial areas of the dot-peen mark. When U-Net fails to segment any dots accurately, no templates can be extracted, so the decoding fails. Templates extracted from U-Net that contain partial dot-peen marks result in the cascaded template matching method failing, where later cascades will begin to falsely match with background areas of the image, leading to a decode failure.

Both types of failures from U-Net likely occur due to limitations of the dataset. Firstly, while the braille dataset is similar to dot-peen marked DMCs, it will not contain many types of dot markings that can be produced through the dot-peen marking process, which may lead to an overrepresentation of dot markings similar to braille dots in the dataset. Secondly, the low number of annotated DMC images (81) results in further underrepresentation of certain types of dot-peen marked DMCs. This becomes especially clear when looking at examples of dot-peen markings where overlapping dots occur, such as those seen in Figure 7, which do not appear in braille characters. A study on the reliability of dot-peen marking[34] explains that this overlap occurs if the marking process uses a combination of high-impact force on low material hardness and small code size. While it may not be achievable in all cases, we recommend using a lower-impact force when marking DMCs on softer materials to reduce this overlap. Thirdly, minor inaccuracies during annotation and the method for estimating the size of the generated dot heatmaps result in some ground truth heatmaps having overly small dots, which leads to poor learning from U-Net.

Overall, the lack of gold-standard annotated images contributes heavily to the poor performance of the U-Net segmentation.

D. Points of Failure - YOLO

From Table V, the second most significant point of failure is the DMC localization step, which relies entirely on YOLO.

It can be seen that the failure rate is highest in the test split, followed by the validation split, with the train split having the lowest failure rate from YOLO. This is expected behavior from learned models and is explainable, as models are likely to have higher failure rates when predicting on images they have not seen before. From visual inspections, failure occurs either due to no detections being made or due to failed detections where the resulting crop only contains a partial section of the DMC. Due to the nature of the failure cases, we argue that an increase in the dataset likely has high potential to significantly improve the performance of the YOLO model.

Furthermore, instead of using the default training hyper-parameters provided by Ultralytics, a cross-validation [35] could be performed to potentially find better ones and further enhance performance.

When comparing the results of the YOLO model on the test set with those of the YOLO model from previous work, it can be seen that the recall and mAP50-95 scores have decreased. Due to very minor differences in the training processes, we attribute this to a negative impact from the images added to the MAN dataset more recently. This could be due to many factors of the new images, but from visual inspection, it could possibly be due to differences in the background present in the images.

E. Points of Failure - Cascaded Template Matching

The third most common point of failure in the pipeline is the cascaded template matching step.

From visual inspection, all failures in this step appear to occur due to false-positive matches being generated. While relatively uncommon due to the strict matching threshold used, because a good matching threshold for an image is different for all examples, the best threshold for an image is sometimes skipped over in the threshold reduction.

While the problem could be reduced by reducing the threshold by less than 0.025 on each reduction, the increase in computation is unjustified. It is possible that a better method for finding a suitable matching threshold could be achieved by first attempting to match with both high and low thresholds, and then estimating a threshold based on the number of matches found in both. Then, the threshold can be effectively determined by further adjustments.

For a suggestion on further optimization, we know from the dataset that DMCs used by MAN are either 14x14 or 16x16. From this, it is possible to calculate the range of potential matches according to the ECC200 standard. With that estimate, it would be reasonable to make significant adjustments to the threshold until the number of matches is within the range, and then make more minor adjustments until false positives are avoided.

Overall, the current cascaded template matching process is prone to errors when incorrect thresholds are used. Either a method for estimating better thresholds should be established, or a different method entirely should be used.

F. Points of Failure - Grid Fitting

The fourth common point of failure in the pipeline is the grid fitting step.

Upon visually inspecting each of the produced grid predictions, it becomes apparent that failures occur due to poor starting angles. The optimization method used finds the local minimum cost for the grid. It will therefore be unable to find the globally optimal theta parameter if the starting angle is too different from the optimal angle. To reduce these errors, either an optimization method capable of finding the global minimum cost should be established, or a different method for estimating the starting angle should be used.

Alternatively, a different grid fitting method not reliant on optimization could be used. If a method for accurately estimating the orientation of the DMC could be established, the coordinates of the grid points could be de-skewed, and a simpler grid fitting method could be used, such as one based on K-Means clustering.

G. Applications in Other Fields

There are possibly other applications for the grid fitting method developed in this paper. For example, in the domain of braille detection, if images of braille dots are warped due to perspective or angle, fitting a grid over the points after localizing them could enable a de-skewing transformation. Furthermore, in the field of robotics, it is often crucial to be able to localize a robot's current position within a given environment. A common approach[36], [37] for localizing a robot's location involves placing recognizable landmarks throughout the environment, typically using 2D barcodes or tiles. It may be possible to set distinct dots in a grid-like pattern in the environment and use a similar grid-fitting method to localize a robot's location.

The cascaded template matching process could be beneficial in other scenarios where it is desired to locate objects within an image, provided there is a lack of training data to train modern object detection models or if current object detection models poorly handle the scenario.

VII. Limitations & Future Work

A. Limitations

The most significant limitation discussed in this paper is the lack of quantity and quality of the dataset. Firstly, while a total image count of 206 initially seems reasonable, it is worth noting that 123 are of dot-peen marked DMCs, and only 81 of those are annotated for the U-Net dot localization method. This, along with the high visual variety of dot-peen marked dots, results in the lack of data, causing the trained U-Net model to underperform for certain types of dot-peen marked DMCs. This lack of data also contributes to the YOLO model underperforming on unseen data. Due to time constraints, the quality of some of the annotations made on the U-Net training images is also not perfect. All of these points compound together, contributing to the high failure rates of the models seen in Table V.

A more general limitation on the models is the lack of exploration in training optimization. No cross-validation has

been performed to optimize hyperparameters, and the U-Net model has a low training time instead of a longer training time stopped by early stopping[38].

Finally, we must acknowledge two unintentional mistakes made in the U-Net training process. Firstly, in our previous work, it was found that a loss calculation combining both Binary Cross Entropy and Dice loss resulted in the U-Net segmentation providing a higher decode rate to the pipeline. Ideally, this loss calculation would have been retried to see if the higher performance would carry over. Secondly, during the template extraction process from U-Net output, the heatmaps are binarized, and other methods are used to clean and extract blob locations as potential templates. These processes could have been included as a head to the U-Net output, allowing the training process of U-Net to match the application of it more closely.

Furthermore, the current methods are not ready for use on mobile phones, as the dot-peen localization methods used are currently too computationally expensive.

B. Future Work

To better develop and test future DMC decoding methods, we recommend the creation of a gold-standard dataset. Due to the specific application of decoding DMCs on machined metal components, a dataset using real-world data matrix markings would enable the development of decoding methods to a provable high standard. Furthermore, to bridge the gap between theory and practical use, the dataset should utilize video footage captured from different mobile phone cameras that record both laser- and dot-peen-marked DMCs, allowing benchmarks to effectively simulate how real users would experience using the decoding methods. Furthermore, using video footage could speed up annotation time by enabling interpolation and tracking methods in between labeled frames. Annotations should include oriented bounding boxes of the DMC itself, and non-oriented bounding boxes for individual dot-peen markings. We recommend oriented bounding boxes, as they effectively allow the training of both oriented and non-oriented object detection methods. Bounding boxes for individual dot-peen locations have the advantage of being more consistent for the generation of heatmap dots, as center dot locations will be more consistent, and the size of heatmap dots can be directly calculated from the size of the bounding boxes.

With a gold-standard dataset established, the methods used in this paper can be revisited and benchmarked for comparison with other approaches. Furthermore, the training processes of both YOLO and U-Net can be optimized via the implementation of cross-validation for hyperparameter tuning and early stopping for U-Net.

It may be possible to entirely avoid cascaded template matching with an accurate enough U-Net model, which would save processing time. With the aforementioned dataset construction and U-Net retraining, it may be possible that preferring lower processing time by avoiding cascade template matching could lead to higher reliability by retrying U-Net segmentation at a higher rate.

Alternatively, an exploration into using object detection methods for dot-peen detection could be made. YOLO may be applicable here, but previous work[39] has indicated that YOLO struggles with densely packed objects, which we also experienced during the exploration phase of this paper.

There is also room for improvement and optimization of the grid fitting methods used in this paper. Firstly, the heavy reliance on accurate starting estimation leaves room for improvement in either the initial angle estimation or a switch to an optimization method more robust to poor starting parameters. Secondly, there is room for optimization of the grid fitting algorithm to reduce processing time. Furthermore, a more complex grid fitting method based on a model that outputs grid parameters could prove more robust. Still, such an exploration should be wary of increases in computation time.

Additionally, the created dataset should be analyzed to find optimal parameters for use in DBSCAN. Two of the current parameters were chosen based on some intuition of the current data, but most of the parameters were set to their default scikit-learn values.

Lastly, the methods used by MAN-approved suppliers for marking the dot-peen DMCs should be investigated to create standardized instructions on how to perform the dot-peen marking process in a way that reduces the overlap between dot-peen markings.

VIII. Conclusion

This paper combines the pre-established model architectures YOLO11 and U-Net with custom decoding methods to create a dot-peen data matrix code decoding pipeline with partial success. While the current implementation has high processing time, the pipeline successfully decodes 15.45% of the dot-peen marked images in the dataset provided by MAN, providing the grounds for future research in improving the methods in both speed and accuracy.

Acknowledgment

The author would like to thank MAN Energy Solutions for providing the dataset of real-world images containing Data Matrix codes, and their supervisors, Yucheng Lu & Veronika Cheplygina from the IT University of Copenhagen research group DASYA for guiding the entire process.

References

[1] M. Laughton and M. Straight, *Libdmtx*, Accessed: 2025-05-26. [Online]. Available: https://github.com/dmtx.

[2] GitHub contributors, *Zxing*, Accessed: 2025-05-26. [Online]. Available: https://github.com/zxing.

[3] G. Jocher and J. Qiu, *Ultralytics yolo11*, version 11.0.0, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics.

[4] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, Springer, 2015, pp. 234–241.

[5] KEYENCE, *Basic practice of 2d codes*, Available online as PDF, 2024. [Online]. Available: https://www.keyence.eu/ss/products/auto_id/codereader/basic_2d/datamatrix.jsp.

[6] D. K. Hansen, K. Nasrollahi, C. B. Rasmussen, and T. B. Moeslund, "Real-time barcode detection and classification using deep learning," in *International Joint Conference on Computational Intelligence*, SciTePress, 2017, pp. 321–327.

[7] A. Zharkov and I. Zagaynov, "Universal barcode detector via semantic segmentation," in *2019 International Conference on Document Analysis and Recognition (ICDAR)*, IEEE, 2019, pp. 837–843.

[8] J. Jia, G. Zhai, P. Ren, *et al.*, "Tiny-bdn: An efficient and compact barcode detection network," *IEEE Journal of Selected Topics in Signal Processing*, vol. 14, no. 4, pp. 688–699, 2020.

[9] R. Wudhikarn, P. Charoenkwan, and K. Malang, "Deep learning in barcode recognition: A systematic literature review," *IEEE Access*, vol. 10, pp. 8049–8072, 2022.

[10] T. Diwan, G. Anirudh, and J. V. Tembhurne, "Object detection using yolo: Challenges, architectural successors, datasets and applications," *multimedia Tools and Applications*, vol. 82, no. 6, pp. 9243–9275, 2023.

[11] R. Neven and T. Goedemé, "A multi-branch u-net for steel surface defect type and severity segmentation," *Metals*, vol. 11, no. 6, p. 870, 2021.

[12] S. Zhang and Y. Zhang, "Automated weld defect segmentation from phased array ultrasonic data based on u-net architecture," *NDT & E International*, vol. 146, p. 103 165, 2024.

[13] R. Li, H. Liu, X. Wang, and Y. Qian, "Dsbi: Double-sided braille image dataset and algorithm evaluation for braille dots detection," in *Proceedings of the 2018 2nd International Conference on Video and Image Processing*, 2018, pp. 65–69.

[14] I. G. Ovodov, "Optical braille recognition using object detection neural network," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct. 2021, pp. 1741–1748.

[15] I. G. Ovodov, "Optical braille recognition using object detection cnn," *arXiv preprint arXiv:2012.12412*, 2020.

[16] C. Li and W. Yan, "Braille recognition using deep learning," in *Proceedings of the 2021 4th International Conference on Control and Computer Vision (ICCCV)*, ser. ICCCV '21, Macau, China: Association for Computing Machinery, 2021, pp. 30–35, ISBN: 9781450390477. DOI: 10.1145/3484274.3484280. [Online]. Available: https://doi.org/10.1145/3484274.3484280.

[17] Y. J. Ahn, *Dotneuralnet: Light-weight neural network for optical braille recognition in the wild*, 2023.

[18] M. Hanumanthappa and V. V. Murthy, "Optical braille recognition and its correspondence in the conversion of braille script to text–a literature review," in *2016 International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*, IEEE, 2016, pp. 297–301.

[19] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. DOI: 10.1109/TSMC.1979.4310076.

[20] A. B. Petro, C. Sbert, and J.-M. Morel, "Multiscale retinex," *Image processing on line*, pp. 71–88, 2014.

[21] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based on immersion simulations," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 13, no. 06, pp. 583–598, 1991.

[22] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *kdd*, vol. 96, 1996, pp. 226–231.

[23] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, "Dbscan revisited, revisited: Why and how you should (still) use dbscan," *ACM Trans. Database Syst.*, vol. 42, no. 3, Jul. 2017, ISSN: 0362-5915. DOI: 10.1145/3068335. [Online]. Available: https://doi.org/10.1145/3068335.

[24] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval research logistics quarterly*, vol. 2, no. 1-2, pp. 83–97, 1955.

[25] P. Virtanen, R. Gommers, T. E. Oliphant, *et al.*, "Scipy 1.0: Fundamental algorithms for scientific computing in python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, 2020.

[26] R. Jonker and A. Volgenant, "A shortest augmenting path algorithm for dense and sparse linear assignment problems," *Computing*, vol. 38, no. 4, pp. 325–340, 1987.

[27] N. Sobolev *et al.*, *Pylibdmtx: A python wrapper for libdmtx*, Accessed: 2025-05-26, 2024. [Online]. Available: https://github.com/NaturalHistoryMuseum/pylibdmtx.

[28] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1506.02640.

[29] A. Paszke, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[30] T. Almeida, V. Santos, O. M. Mozos, and B. Lourenco, "Comparative analysis of deep neural networks for the detection and decoding of data matrix landmarks in cluttered indoor environments," *Journal of Intelligent & Robotic Systems*, vol. 103, no. 1, p. 13, 2021.

[31] R. Chen, Y. Yu, X. Xu, L. Wang, H. Zhao, and H.-Z. Tan, "Adaptive binarization of qr code images for fast automatic sorting in warehouse systems," *Sensors*, vol. 19, no. 24, p. 5466, 2019.

[32] S.-S. Lin, M.-C. Hu, C.-H. Lee, and T.-Y. Lee, "Efficient qr code beautification with high quality visual content," *IEEE Transactions on Multimedia*, vol. 17, no. 9, pp. 1515–1524, 2015.

[33] Z. Yang, H. Xu, J. Deng, C. C. Loy, and W. C. Lau, "Robust and fast decoding of high-capacity color qr codes for mobile applications," *IEEE Transactions on Image Processing*, vol. 27, no. 12, pp. 6093–6108, 2018.

[34] D. Dragičević, S. Tegeltija, G. Ostojić, S. Stankovski, and M. Lazarević, "Reliability of dot peen marking in product traceability," *International Journal of Industrial Engineering and Management*, vol. 8, no. 2, p. 71, 2017.
[35] D. Berrar *et al.*, *Cross-validation.* 2019.
[36] Y. Alkendi, L. Seneviratne, and Y. Zweiri, "State of the art in vision-based localization techniques for autonomous navigation systems," *IEEE Access*, vol. 9, pp. 76 847–76 874, 2021. DOI: 10.1109/ACCESS.2021.3082778.
[37] M. Betke and L. Gurvits, "Mobile robot localization using landmarks," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 2, pp. 251–263, 1997. DOI: 10.1109/70.563647.
[38] L. Prechelt, "Early stopping-but when?" In *Neural Networks: Tricks of the trade*, Springer, 2002, pp. 55–69.
[39] M. L. Ali and Z. Zhang, "The yolo framework: A comprehensive review of evolution, applications, and benchmarks in object detection," *Computers*, vol. 13, no. 12, 2024, ISSN: 2073-431X. DOI: 10 . 3390 / computers13120336. [Online]. Available: https://www.mdpi.com/2073-431X/13/12/336.

APPENDIX

ML-based Data Matrix image localization and segmentation for standard scanners

Supervisors: Yucheng Lu (80%) & Veronika Cheplygina (20%)
In collaboration with MAN Energy Solutions

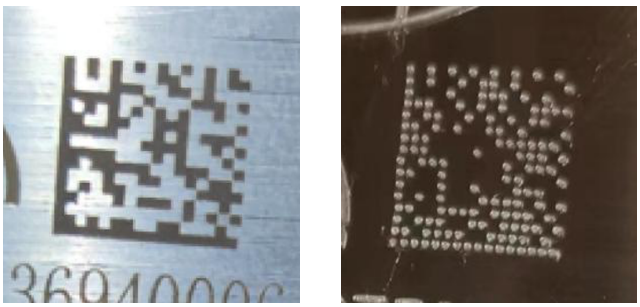
Aidan Stocks — aist@itu.dk

***Abstract*—Data Matrix codes are used at MAN Energy Solutions to serialize various machined metal components. Historically, decoding these codes has been done using algorithmic approaches without machine learning, such as in the case of the open-source library libdmtx. While libdmtx performs well under typical conditions, it often fails in industrial environments such as factories and ships, where poor lighting and damaged codes are common. To address these challenges, this research leverages recent advances in object detection and image segmentation, employing a cascaded model integrating YOLOv11 and a modified U-Net architecture to improve the performance of libdmtx. Experimental results demonstrate a 12% increase in decode rate under these industrial conditions, though at the expense of processing time, highlighting opportunities for further optimizing these methods for real-time applications.**

***Index Terms*—Data Matrix, YOLO, Object Detection, U-Net, Image Binarization, Segmentation**

I. Introduction

IN industrial engineering settings, Data Matrix codes (DMCs) are often marked directly on machined metal components to serialize them. MAN Energy Solutions requires manufacturers to use DMCs to serialize engine components so that they are verifiably from a MAN-approved supplier. Although DMC scanners work in ideal conditions, they often fail when scanning from smartphone cameras while on board ships or in specific factory settings. This is usually due to poor quality cameras, the DMCs being physically small, on reflective (metal) surfaces, or due to poor lighting conditions. Most scanners are also not designed to recognize dot-peen marked Data Matrix codes, a commonly used and cheaper alternative method for marking components seen in Figure 1b. This paper investigates how machine learning methods can improve the speed and decode rate of the existing open-source DMC scanner library libdmtx[1] in this industrial setting.



(a) Laser **(b)** Dot Peen

Fig. 1: Difference between a laser and dot peen marked DMC.

A two-step cascaded model implementing YOLO[2] object detection to localize the DMC, and U-Net[3] segmentation for image binarization to remove noise, successfully improves the decode rate of the standard scanner at the cost of increased runtime.

By preprocessing the image with YOLO cropping and a U-Net-based image binarization, some steps of libdmtx are effectively done for it: YOLO for efficient localization and U-Net for robust image binarization.

This research paper focused on using models that solve the tasks well and fast, ideally in real-time, where an input image can be processed at speeds greater than one frame per second (> 1 FPS). Previous works [4]–[6] show that object detection models such as YOLO[2] can perform object detection tasks accurately and in real-time speed. The lightest architecture of one of the most recent YOLO models developed by Ultralytics, yolo11n[7], was chosen for the localization task due to its improved accuracy and speed over previous versions.

Other works[8], [9] show that adaptive threshold binarization is an image-processing technique effective in preprocessing QR codes for the decoding process. One paper[10] shows that a more complex image processing model such as U-Net can binarize images better than methods that calculate global binarization thresholds[11] in highly noisy images with adverse lighting conditions. Therefore, we propose using the U-Net[3] architecture with an attached sigmoid and binarization head thresholded at 0.5.

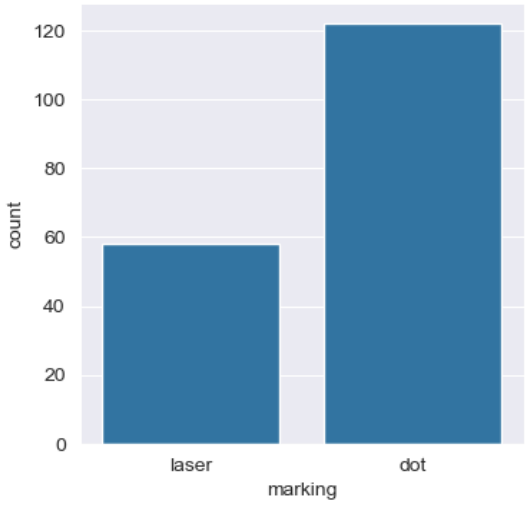
The code used to produce these results, along with a README of the exploratory process, is available at this GitHub repository.

II. Data Gathering

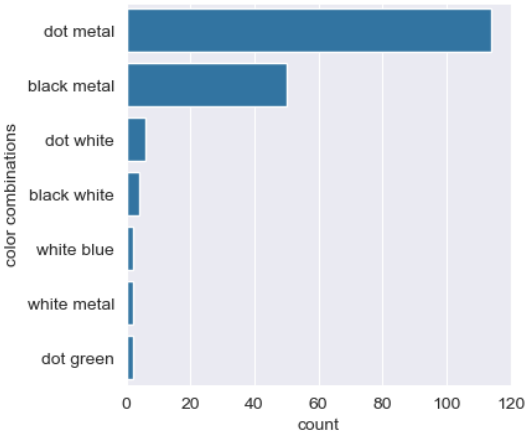
For training and testing the models, a dataset containing images of both laser and dot-peen-marked metal components was provided by MAN Energy Solutions. It contains 180 images, the distributions of which can be seen in Figure 2.

Most images contain DMCs made by dot-peen marking, and most have different types of distortions on their DMCs. A sample of the images manually cropped to their DMCs is shown in Figure 3 for convenience.

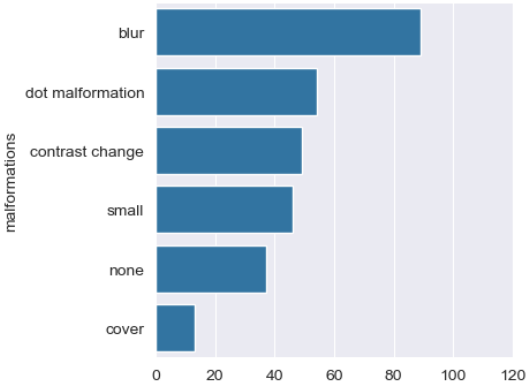
Only 50 of the 180 images were kept as a testing dataset to allow room for fine-tuning models.



(a) Counts of Laser and Dot Peen marked DMCs. Note the high amount of dot-peen marked components.

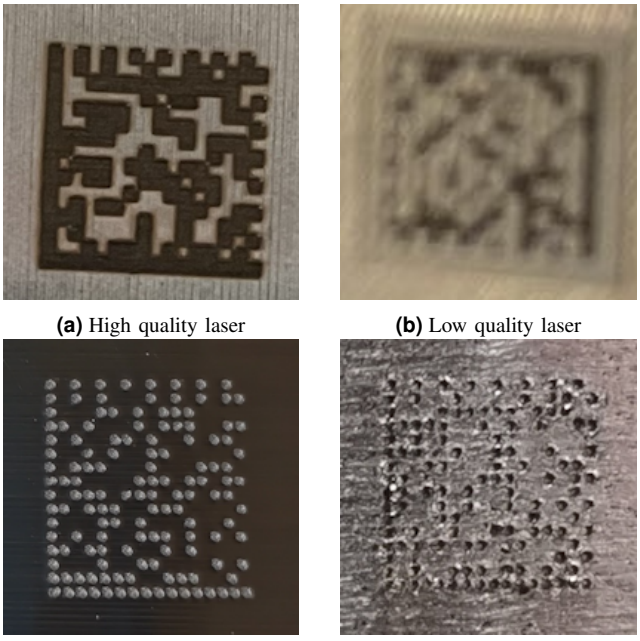


(b) Counts of combinations of DMC (left) and background (right) colors. Dot-peen marked components are marked "dot". Most laser-marked components are black on metal (lasered directly onto the metal), but cases of markings on white stickers and other materials also exist. In most cases, the background color is metallic, while the DMC is either a hole from dot-peen marking or blackened metal from a laser.



(c) Counts of malformation type present

Fig. 2: Difference between a laser and dot peen marked DMC. Note the high amount of blur and contrast change in all images and the high amount of malformations in the dot-peen marked components. "Contrast change" denotes a variation in background color due to light reflections, causing a difference in contrasting colors between DMC and background.



(a) High quality laser **(b)** Low quality laser

(c) High quality dot peen **(d)** Low quality dot peen

Fig. 3: Examples of differing quality between DMCs.

Roboflow[12] was used for train validation test splitting into sizes of 100, 30, and 50 respectively. Roboflow was also used to augment the training set and increase its size. The following pre-processing and augmentation steps were performed:

1) Auto orientation.
2) Resize to 640x640 (input size for yolo11n)
3) Rotations of 90°, 180°, and 270°.
4) Shearing of ± 5 in both horizontal and vertical dimensions.

The augmented dataset can be found on Roboflow here.

III. Models

A. Framework Overview

Condensed, libdmtx uses the following steps to decode a given image containing a DMC:

1) Image Preprocessing. The input image is grayscaled and binarized with a specified or calculated threshold to help reduce noise.
2) Localization. The DMC finder pattern ("L" shape) is localized, giving an approximate area of the image that contains the DMC.
3) Geometric Alignment. The DMC orientation is corrected, and a grid is overlayed on the DMC.
4) Symbol Decoding. The grid is iterated over, and the contrast between the black pixels of the DMC and the white pixels of the background is used to convert the DMC into encoded bits. Reed-Solomon error correction is used.
5) Data Extraction. The bits are finally decoded into the desired string or binary data.

The first two steps of the libdmtx decoder, image binarization & DMC area localization, are effectively replaced by

the two cascaded models. However, we perform the two steps in reverse order because YOLO is a computationally cheaper model architecture to run than U-Net. By doing the steps in reverse, YOLO's speed can be utilized to detect and crop to the DMC area relatively fast, leading to fewer pixels for the slower U-Net model to process for binarization.

Concretely, we end up with a framework where YOLO is used to crop an image down its DMC, U-Net is used to segment and binarize the DMC from the rest of the cropped image, and libdmtx is used to decode the binarized DMC.

B. DMC Localization

During the early exploration phase, the Ultralytics yolo11n pre-trained model was tested on all MAN images to see if any of the 80 classes from the dataset it was trained on (COCO[13]) would identify the DMCs. Only two images produced correct detections, proving that the YOLO model needed to be retrained or finetuned for the domain. The two detections can be seen in Figure 4.

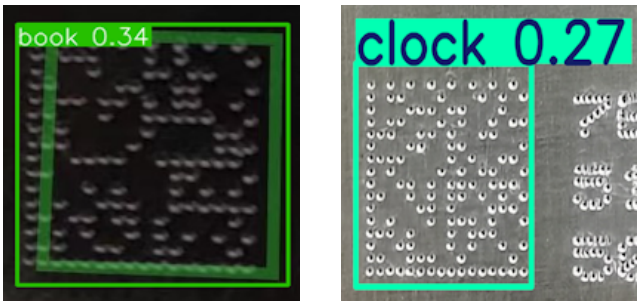


Fig. 4: Valid DMC detections from pretrained yolo11n.

Roboflow[12] was used for manual annotation of the dataset, drawing the bounding boxes to calculate and save the required x, y, w, and h of them. The (x, y) coordinates denote the center of the predicted bounding box, while the (w, h) denote the width and height of the box. YOLO models for object detection tasks predict these values.

Three different YOLO models were trained and evaluated.

1) YOLO from Scratch: A dataset publicly available on Kaggle[14] was used to train a YOLO model from scratch. The dataset contains images of QR codes both on plain white backgrounds and surrounded by text. The dataset also supplies the associated YOLO bounding box annotations for each image. Figure 5 shows samples of both types of images present in the Kaggle dataset.

A YOLO model trained from scratch on this dataset was initially considered because of the visual similarities between the two code types, potentially allowing for effective transfer learning. Figure 6 from another paper[15] shows the similarities and differences between the two code types.

2) Kaggle - Finetuned: Kaggle Scratch was finetuned on the MAN training and validation sets to produce another model for comparison. By fine-tuning to the real-world images, the model's performance should improve.

3) YOLO - Finetuned: Following Ultralytics recommendations, a final model for comparison is produced by fine-tuning their pre-trained yolo11n model on the MAN training and validation sets.

(a) Simple **(b)** Complex

Fig. 5: Examples from Kaggle dataset.

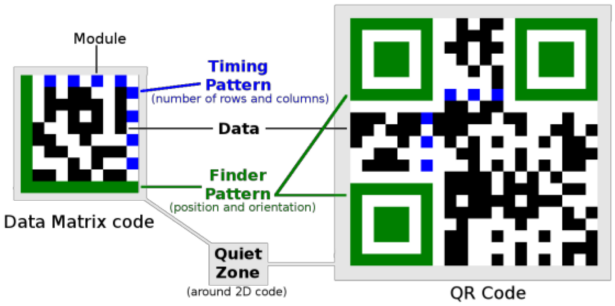


Fig. 6: Comparison between Data Matrix code and QR code. Note the main difference in finder and timing patterns.

C. DMC Binarization

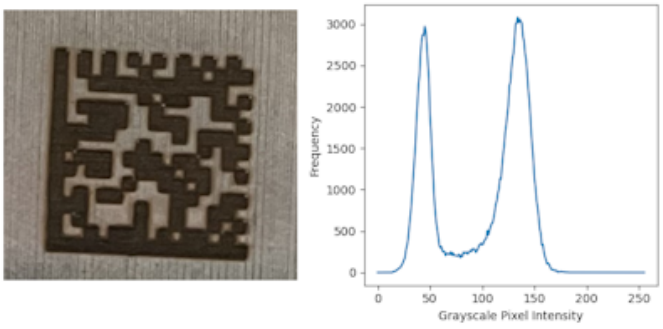
During the exploration phase of this paper, methods other than U-Net were tried and failed.

First, we trained a ResNet-18[16] model on some synthesized data, but the trained model produced poorly binarized images. Next, we performed a more straightforward histogram analysis of the MAN images to see if there was a global binarization threshold across the images. Histogram analysis of an image involves visually inspecting the distribution of grayscaled pixel intensities.

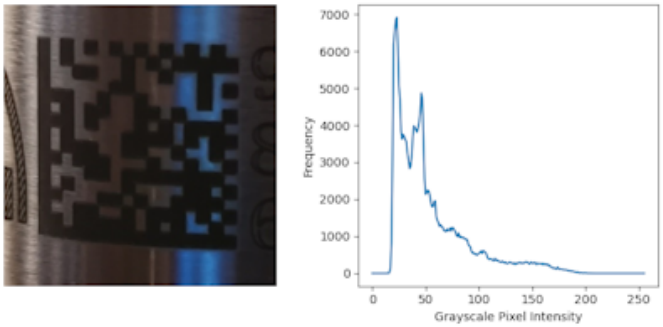
For an image binarization task, if an image contains a separation in peak grayscale pixel intensities, then binarizing the image with a threshold between the peaks will effectively separate the two areas in the image. Figure 7a highlights this calculated threshold, where there is a clear separation between two peaks of grayscale pixel intensities. In contrast, we see in Figure 7b a more minor separation between the peaks due to the poor lighting conditions, resulting in the failed binarization seen in 7c.

Unfortunately, while many images contain clear separations of grayscale value peaks in their histograms, there are still many failure cases similar to Figure 7b. These failure cases occur when there are light reflections and other related distortions within the area of a DMC. For these cases, no global threshold will produce a successful binarization. Because of this, other similar methods for automatic threshold selection, such as Otsu's method[11], will not work either. Therefore, a more complex method for effective binarization is required.

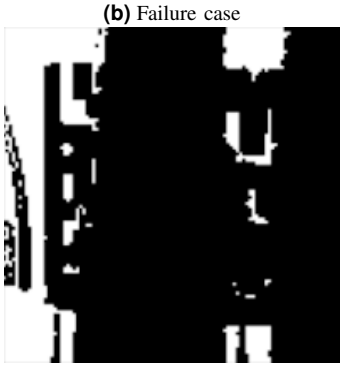
U-Net is an image segmentation model architecture developed for use in biomedical images. However, the application



(a) Success case



(b) Failure case



(c) Failure case when binarized with calculated threshold

Fig. 7: Binarization with histogram analysis. Note the more distinct separation of peaks in 7a compared to 7b.

of U-Net is not only limited to the medical industry, and its use cases are continually expanding. Since U-Nets development, the architecture has been used in image segmentation tasks for disease detection in leaves[17], [18], aerial image analysis for urban planning[19], and more. Because of its usage outside of medical applications, it stood to reason that it could be used to segment DMCs.

To train the U-Net model, we define that the model should, given an input image containing a DMC:

- Transform all pixels of the DMC completely black (0). These pixels would be present if the DMC were created with standard grid squares and without imperfections.
- Transform all pixels not of the DMC completely white (1). These are pixels belonging to the background surface the DMC is marked on or imperfections resulting from blur, which may extend the edges of the DMC grid squares.

The model should deal with all "noise" related to coloring, e.g., the DMC area not being perfectly black due to rust, oil, blur, light reflections, or other factors. The model should not deal with noise related to the shape of the DMC, such as rotation and warping.

To achieve this, a sigmoid layer followed by a binarization with a threshold of 0.5 was added as a head to the U-Net architecture. Figure 8 shows the complete architecture of the U-Net model used.

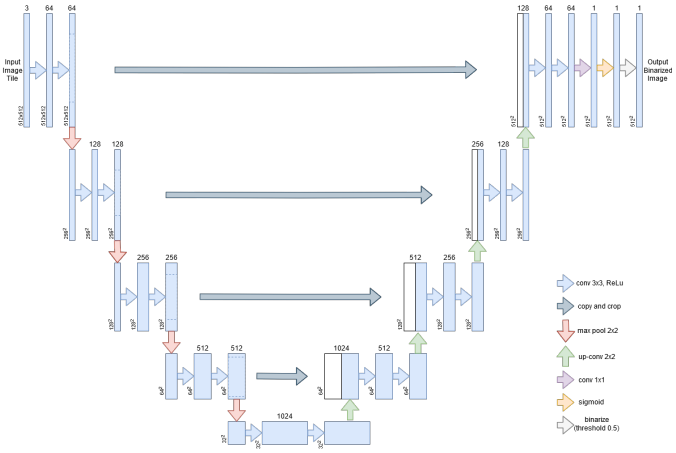


Fig. 8: Modified U-Net Architecture. The architecture differs from the original U-Net in the extra sigmoid and binarization layers at the end, and a slightly different padding method was used in the convolutional layers.

With the model well defined, the requirements of the dataset can be defined. Keeping in mind that the U-Net model will take the output of a YOLO model as input, the ideal input images for training are defined as the following:

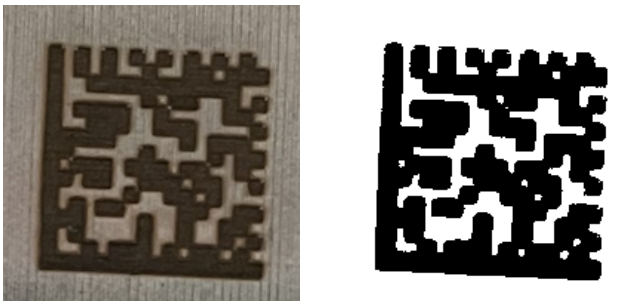
- The DMC within the image must cover a large area to simulate a cropped image from the YOLO model.
- The background of the DMC must be metallic, as Figure 2b shows this is the dominant background in the MAN-supplied images.
- The DMC *shape* should vary in rotation, shear, affine transformation, and perspective warping to simulate the examples seen in the MAN dataset.
- Due to the high amounts of blur distortion seen in Figure 2c, a random amount of blur should be applied to the images.
- Various random *color* variations (especially lighting distortion/noise) should be added to encourage model generalization. The model should learn to handle the main issue that globally thresholded binarization methods fail to handle.

Ground truth images are required for the model to learn how to binarize these images. The goal is to have the model binarize the images without rectification, so we define the ground truth images as counterparts to the training images in the following ways:

- Ground truth images should contain the same shape transformed DMC as its training image counterpart.
- Ground truth images are binarized images, where the black pixel values of the DMC are 0 ([0, 0, 0] in RGB

terms), and white background pixel values (in and around the DMC) are 1 ([255, 255, 255] in RGB terms).

An example of an ideal training and ground truth image pair is shown in Figure 9.



(a) Ideal "noisy" image **(b)** Ideal "ground truth" image

Fig. 9: Example of ideal noisy image and a (close to) ideal ground truth counterpart. Edited from a MAN-supplied image. More adverse lighting than seen here should be present in the synthetic images.

Because no public dataset like this existed, one needed to be synthesized. The synthesis process used libdmtx for DMC generation and PyTorch transforms v2[20] for applying the appropriate shape and color transformations. Metal textures were sourced from various free online sources and randomly cropped for use as backgrounds for the DMCs.

One thousand unique DMCs were generated, and a dataloader for augmenting the dataset appropriately during training was created. Figure 10 shows the synthesis process with an example training and ground truth image pair generated during the training process.

Of the 1000 generated DMCs, 900 were used for training, and the other 100 were used as a validation set. During training, on completion of each epoch, a model was evaluated on the validation set, and its loss was calculated. If the reduction in validation loss from the previous epoch was below a specified amount, the training would stop, and the model trained from the previous epoch would be saved. This form of early stopping helped reduce the model's overfitting to the synthetic training data.

We use the Python library "pillow" to load and compare images. This library also allows us to represent them as grayscale images with pixel values ranging from 0 (black) to 1 (white), the same range produced by the U-Net model architecture with its sigmoid layer head.

We trained three different U-Net binarizers, each optimizing for different loss functions. One for binary cross-entropy (BCE) loss, one for dice loss, and a final for an average of the two (BCEDice loss). In this case, BCE loss is calculated as the average of the differences between the predicted and ground-truth pixel values 11a. Dice loss here is calculated as the intersection between the predicted and ground-truth pixel values 11b. Theoretically, both can work as practical loss functions for this image binarization task, but to see which method (or a mix of the two 11c) produces the best model, we try all three options. Each model was trained and saved for later evaluation.

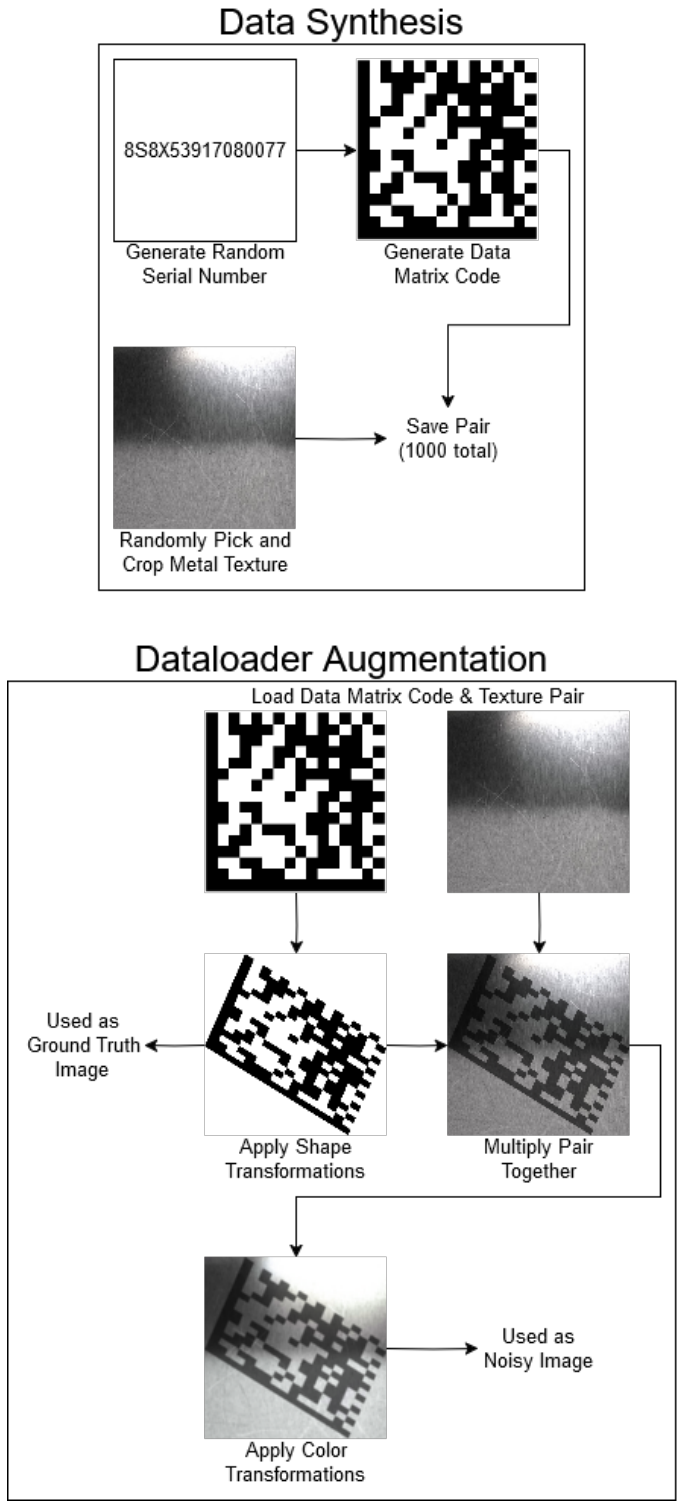


Fig. 10: The data synthesis and dataloader augmentation process. Note that while each epoch uses the same image pairs from the data synthesis, there is randomness introduced by the shape and color transformations.

$$BCELoss(x, y) = \frac{\{l_1, ..., l_N\}^\top}{N},$$
$$l_n = [y_n \cdot \log(x_n) + (1 - y_n) \cdot \log(1 - x_n)]$$

(a) BCE loss formula. $BCELoss(x, y)$ is the average batch loss comparing predicted images x with their ground truths y across all N images in a batch (batch size $N = 4$ in our case). l_n is the loss for a single n-th sample.

$$DiceLoss(x, y) = 1 - \frac{(2 \cdot \sum_{n=1}^{N} x_n y_n + \epsilon)}{(\sum_{n=1}^{N} x_n + \sum_{n=1}^{N} y_n + \epsilon)}$$

(b) Dice loss formula. $DiceLoss(x, y)$ is the average batch loss comparing predicted images x with their ground truths y across all N images in a batch (batch size $N = 4$ in our case). ϵ is the smoothing factor (standard of 1 in our case) to avoid $0/0$ cases.

$$BCEDiceLoss(x, y) = BCELoss_N + DiceLoss_N$$

(c) BCEDice loss formula. $BCEDiceLoss(x, y)$ is the summation of $BCELoss$ and $DiceLoss$ for batch N.

Fig. 11: Loss functions used to produce three differently optimized U-Net binarizers.

IV. Results & Discussion

A. DMC Localization Results

The YOLO models were each evaluated on the MAN test set along with the baseline libdmtx decoder. The full evaluation pipeline for each YOLO model was done as follows. For each image:

1) Use the given YOLO model to predict bounding boxes for possible DMC locations in the image.
2) Crop the image down to the dimensions of the highest likelihood bounding box.
3) Add 45 pixels of padding around the crop to ensure the model prediction does not accidentally cut off the DMC.
4) Decode with baseline decoder.

Table I shows how each model performed on the test set. Figure 12 shows all successfully decoded test image crops produced by the models.

Measure	Baseline	Kaggle Scratch	Kaggle Finetuned	Ultralytics Finetuned
Precision	-	0.25	0.91	**0.96**
Recall	-	0.24	0.84	**0.89**
F1	-	0.25	0.87	**0.92**
mAP50-95	-	0.07	0.75	**0.75**
DMC decode rate	**0.12**	0.04	**0.12**	**0.12**
FPS	1.07	**2.56**	2.02	1.96

TABLE I: Performance Comparison of YOLO Models on MAN test set. FPS is calculated on a laptop and should not be read directly; it is used to compare the speeds between methods.

All YOLO models reduce the overall runtime of the decoding pipeline. However, note that every subsequent YOLO model after the Kaggle from scratch model performs slower than the previous. Both fine-tuned models have the same decode rate as the baseline.

(a) Baseline **(b)** Kaggle Finetuned **(c)** Ultralytics Finetuned

Fig. 12: Crops from successful decoding from baseline decoder and YOLO model outputs. Note that Kaggle Scratch failed to detect and crop to this example.

B. DMC Localization Discussion

Comparing the resulting cropped images with the baseline, it becomes clear why the overall decoder speed increases even with an extra preprocessing step. The average sizes of the cropped images are much smaller than the baseline, meaning the decoder pipeline has fewer pixels to process.

However, cropping the images does not increase the decode rate of the baseline decoder. This is likely because cropping the image only improves step 2 (localizing the DMC) of the libdmtx pipeline, and the decode rate of libdmtx may not depend on the DMC size but rather the DMC's quality. The Kaggle from scratch model decreases the decode rate due to the YOLO model failing to detect and crop to some of the DMCs, and as a result, it is not used in further comparisons. The slower decode rate of the finetuned models is likely due to a higher number of false positives in predicted bounding boxes compared to the Kaggle from scratch model.

The Ultralytics Finetuned model performs best in all measures except speed and is therefore selected as the YOLO model to use in step 2 with the U-Net binarizer.

Overall, we consider step 1 a success, as performing DMC localization with yolo11n has successfully reduced the baseline decoder's overall decode time without sacrificing the rate of decoding.

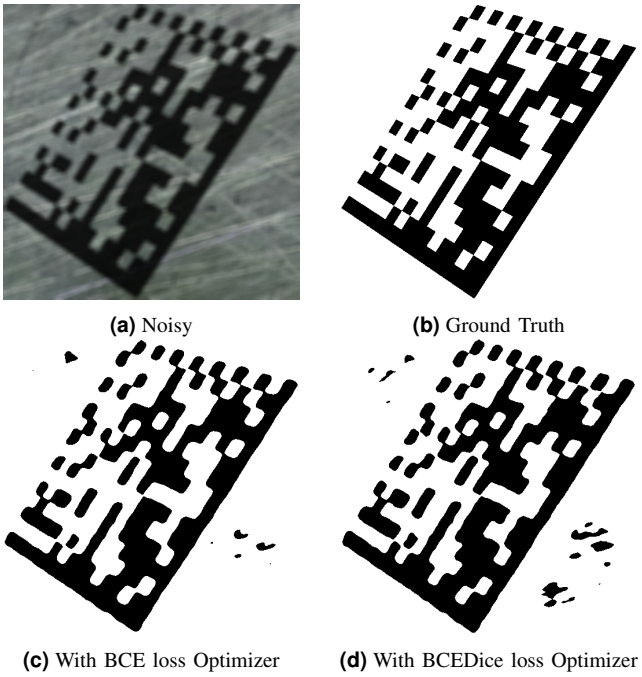
C. DMC Binarizer Results

Each trained U-Net binarizer was evaluated on the validation set and YOLO croppings of all MAN-supplied images containing laser markings (58 images). Because the binarizers were not finetuned to the real-world MAN data, bias is only present in the YOLO cropping step and not in the binarizer step.

A sample of the different model binarizations can be seen below in Figure 13. The model optimized purely for Dice loss produced only white images on the validation set and was therefore excluded from all evaluations.

The models trained for BCE loss and BCEDice loss produced similarly binarized images. The results of the two models on all MAN images containing laser-marked DMCs, along with samples of the model binarizations, are shown in Table II and Figure 14 below.

Both models sacrifice overall runtime to increase the decode rate of the baseline decoder. The model optimizing for both BCE and Dice loss outperforms the BCE loss model at an overall decode rate of $\sim 55\%$.



(a) Noisy **(b)** Ground Truth

(c) With BCE loss Optimizer **(d)** With BCEDice loss Optimizer

Fig. 13: Sample of Validation Binarizations with two successful models.

Measure	Baseline	BCE Loss	BCEDice Loss
DMC decode rate	0.43	0.48	**0.55**
FPS	1.23	0.32	0.32

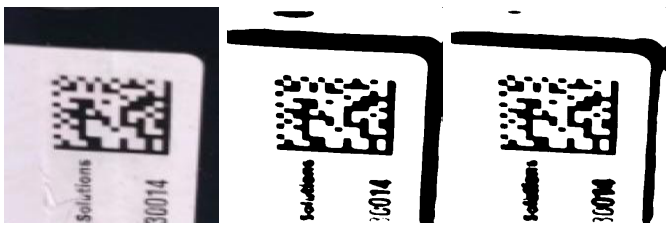
TABLE II: Performance Comparison of U-Net Binarizer Models on all MAN images containing laser-marked DMCs.

D. DMC Binarizer Discussion

Despite the failure of the model optimizing purely for Dice loss, the model optimizing for both BCE and Dice loss performs the best in practice. The model optimizing for Dice loss may have failed due to the lack of class balancing between black and white pixels. We could likely reduce the class imbalance problem by weighing the classes evenly across batches, potentially utilizing dice loss better. However, the dice loss combined with BCE loss leads to an even better decode rate than only optimizing for BCE loss.

Overall, the U-Net architecture can effectively be utilized for binarization tasks, as seen by the improvement in decode rate. However, the increase in runtime results in a decoding pipeline that runs in less than real-time, taking an average of $3s$ to process each image.

The model's high runtime is partially due to the high dimensionality of the input and output images. The size of the input images is three channels (RGB) 512x512, while the output is one channel (Grayscale) 512x512 images. It should be explored if having the input grayscale and scaling down the size could increase the model's forward pass speed without sacrificing the decode rate. A good first step for experimenting with this could be replicating the original U-Net architecture more closely by inputting grayscale images of sizes 572x572. Still, depending on how vital the color information was for achieving good binarization, it may produce lower decode rates.



(a) Input Image **(b)** BCE Loss **(c)** BCEDice Loss

Fig. 14: Comparison of Binarization between optimizer methods.

V. Final Discussion & Conclusion

We have shown that machine learning methods can effectively be utilized to improve the decode rate of standard scanners. However, with the techniques used in this paper, the enhanced decode rate comes at the cost of increased runtime. With more research into this area, it may be possible to increase the decode rate further and optimize the methods used to reduce the runtime for real-time usage.

A. Localization with YOLO

Localizing the DMC area with a finetuned version of Ultralytics pretrained yolo11n model successfully speeds up the baseline decoder without sacrificing its decode rate.

B. Binarization with U-Net

Using U-Net to segment the DMCs from the rest of the images via image binarization successfully improved the decode rate of the baseline decoder. However, the U-Net model architecture proved more computationally expensive than the baseline decoder, leading to a slower than real-time processing speed of around $3s$.

C. Limitations

The most significant limitation of this project was access to real-world data. It would be best to have thousands of real-world images of the DMCs marked on metal components in various conditions. However, obtaining these images requires requesting workers from across the globe to take them, and they would need to be manually annotated to get their YOLO bounding boxes and U-Net ground-truth labels. While technically possible, this process would be expensive and require backing from MAN.

Another limitation was time. More time to develop and optimize the methods, particularly the U-Net architecture, could improve the models' final runtime.

D. Future Work

Research into rectification of the DMCs, perhaps after the binarization step, could result in a decoding pipeline able to generalize for differently shaped DMCs, such as the dot-peen marked DMCs.

Furthermore, the U-Net binarization speed could be improved by reducing the dimensionality of the input and output

images. Depending on how relevant the color information was for binarizing the images, the inputs could be significantly reduced by providing grayscale images and inputs instead of RGB.

Experimentation with balancing the class weights during Dice loss calculation could lead to an improved BCEDice loss optimizer.

More versions of yolo11n exist that could be utilized as alternatives to the models used in this paper. For example, yolo11n-pose is a YOLO object detection model that predicts oriented bounding boxes, which could improve the decode rate of libdmtx even further by effectively performing part of step 3 (Geometric Alignment). Alternatively to U-Net, there also exists yolo11n-seg, an Ultralytics YOLO model designed for segmentation tasks. yollo11n-seg may outperform U-Net's speed and performance.

It was seen through earlier experimentation that care was necessary when defining the steps for data synthesis. For example, some previously used metal textures were too different from the real-world examples, leading to training models that did not generalize well to the MAN-supplied data. A more thorough investigation into which data synthesis methods produce good models for this domain could lead to a better understanding of relevant factors to synthesize, possibly producing better-trained models. The data synthesis could even be taken a step further by simulating the data in 3 dimensions. Three-dimensional CAD models (Computer-aided design) available from MAN could also be used, with randomized parameters to simulate pointing a phone camera at a DMC pasted onto it.

ACKNOWLEDGMENT

The author would like to thank MAN Energy Solutions for providing the dataset of real-world images containing Data Matrix codes, and their supervisors, Yucheng Lu & Veronika Cheplygina from the IT University of Copenhagen research group DASYA for guiding the entire process.

REFERENCES

[1] M. Laughton and M. Straight, *Datamatrix reading and writing library, utils and wrappers*. [Online]. Available: https://github.com/dmtx.

[2] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, 2016. arXiv: 1506.02640 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1506.02640.

[3] O. Ronneberger, P. Fischer, and T. Brox, *U-net: Convolutional networks for biomedical image segmentation*, 2015. arXiv: 1505.04597 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1505.04597.

[4] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023, ISSN: 2504-4990. DOI: 10.3390/make5040083. [Online]. Available: https://www.mdpi.com/2504-4990/5/4/83.

[5] T. Diwan, G. Anirudh, and J. V. Tembhurne, "Object detection using yolo: Challenges, architectural successors, datasets and applications," *multimedia Tools and Applications*, vol. 82, no. 6, pp. 9243–9275, 2023.

[6] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, "A review of yolo algorithm developments," *Procedia computer science*, vol. 199, pp. 1066–1073, 2022.

[7] G. Jocher and J. Qiu, *Ultralytics yolo11*, version 11.0.0, 2024. [Online]. Available: https://github.com/ultralytics/ultralytics.

[8] R. Chen, Y. Yu, X. Xu, L. Wang, H. Zhao, and H.-Z. Tan, "Adaptive binarization of qr code images for fast automatic sorting in warehouse systems," *Sensors*, vol. 19, no. 24, p. 5466, 2019.

[9] R. Chen, W. Li, K. Lan, J. Xiao, L. Wang, and X. Lu, "Fast adaptive binarization of qr code images for automatic sorting in logistics systems," *Electronics*, vol. 12, no. 2, p. 286, 2023.

[10] P. V. Bezmaternykh, D. A. Ilin, and D. P. Nikolaev, "U-net-bin: Hacking the document image binarization contest,", vol. 43, no. 5, pp. 825–832, 2019.

[11] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, 1979. DOI: 10.1109/TSMC.1979.4310076.

[12] *Everything you need to build and deploy computer vision applications*, 2024. [Online]. Available: https://roboflow.com.

[13] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, *Microsoft coco: Common objects in context*, 2015. arXiv: 1405.0312 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1405.0312.

[14] Hamid, *Yolo-qr-labeled*, version 1, 2024. [Online]. Available: https://www.kaggle.com/datasets/hamidl/yoloqrlabeled.

[15] L. Karrach and E. Pivarciova, "Comparative study of data matrix codes localization and recognition methods," *Journal of Imaging*, vol. 7, p. 163, Aug. 2021. DOI: 10.3390/jimaging7090163.

[16] K. He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition*, 2015. arXiv: 1512.03385 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1512.03385.

[17] M. E. Chowdhury, T. Rahman, A. Khandakar, *et al.*, "Automatic and reliable leaf disease detection using deep learning techniques," *AgriEngineering*, vol. 3, no. 2, pp. 294–312, 2021.

[18] S. Zhang and C. Zhang, "Modified u-net for plant diseased leaf image segmentation," *Computers and Electronics in Agriculture*, vol. 204, p. 107 511, 2023.

[19] Z. Pan, J. Xu, Y. Guo, Y. Hu, and G. Wang, "Deep learning segmentation and classification for urban village using a worldview satellite image based on u-net," *Remote Sensing*, vol. 12, no. 10, p. 1574, 2020.

[20] A. Paszke, S. Gross, F. Massa, *et al.*, *Pytorch transforms v2 documentation*, Accessed: 2024-12-14, 2024. [Online]. Available: https://pytorch.org/vision/0.20/transforms.html.