Automation of nuclear material cladding coating measurement process

Emma Tekulová

April 9, 2025

Summary

The data used in this work are from researchers using scanning electron microscope images to investigate the properties of oxidation and coating layers on various materials essential to the nuclear industry. Accurately measuring the thickness of these layers is important for material characterization. The current manual measurement process is time-consuming, and automating it can significantly improve efficiency. Many algorithms can assist with automation, varying widely in complexity and input expectations. While many of these algorithms promise powerful and accurate results, real- world scenarios require substantial groundwork before these tools can be effectively deployed and automation established. This work discusses the challenges and solutions encountered, beginning with the collection and assessment of all available data. It then explores experiments with both conventional computer vision and machine learning algorithms, evaluating their performance. The overall parameters of different solutions are examined, and the most suitable approach is selected. This approach is then seamlessly integrated into the researchers' existing workflow.

Súhrn

Dáta použité v tejto práci pochádzajú od výskumníkov, ktorí využívajú snímky zo skenovacieho elektrónového mikroskopu na skúmanie vlastností oxidačných a povlakových vrstiev na rôznych materiáloch dôležitých pre jadrový priemysel. Presné meranie hrúbky týchto vrstiev je kľúčové pre charakterizáciu materiálov. Súčasný manuálny proces merania je časovo náročný a jeho automatizácia môže výrazne zvýšiť efektivitu.

Existuje mnoho algoritmov, ktoré môžu pomôcť s automatizáciou, pričom sa líšia svojou zložitosťou a požiadavkami na vstupné dáta. Hoci mnohé z nich ponúkajú presné a výkonné výsledky, v realite si ich nasadenie vyžaduje rozsiahlu prípravu, aby bolo možné efektívne dosiahnuť automatizáciu.

Táto práca sa zaoberá výzvami a riešeniami, ktoré boli pri vývoji automatizovaného riešenia identifikované. Začína zberom a analýzou dostupných dát, následne sa venuje experimentom s konvenčnými algoritmami počítačového videnia a strojového učenia, pričom hodnotí ich výkonnosť. Sú zhodnotené parametre rôznych riešení a vybraný najvhodnejší prístup, ktorý je následne integrovaný do existujúceho pracovného postupu.

Acknowledgements

I would like to express my gratitude to my advisor, **Ing. Petr Čech, Ph.D.**, and my consultant, **Mgr. Jaroslav Knotek**, for their guidance and support throughout this work.

The input data used was created with state support from the Technology Agency of the Czech Republic under the THÉTA Program as part of project No. TK04030082 and utilizing the CICRR infrastructure, which is financially supported by the Ministry of Education, Youth, and Sports – project LM2023041.

Contents

Introduction

The Research Institute Řež conducts numerous projects in the nuclear industry. Some of these involve electron microscopy and the analysis of materials subjected to high temperatures and aggressive environments, such as those found in nuclear reactors. A significant portion of these projects requires extensive manual image analysis, making automation a crucial challenge. Currently, image analysis is performed using various processing tools, including Fiji [15] by ImageJ. This thesis aims to automate one of these manual processes.

The primary focus of this work is the segmentation of coating layers in material samples. These images contain both coating and oxidation layers, with coating layers exhibiting greater uniformity, whereas oxidation layers present irregular structures, as illustrated in Figure 1.1. By initially addressing the segmentation of coating layers, this thesis evaluates the potential of automation in reducing analysis time while maintaining sufficient accuracy. If successful, future work could extend this approach to more complex cases, such as samples without coating layers and those with multiple oxidation types.

Three computer vision algorithms are explored to achieve segmentation, ranging from traditional clustering techniques to advanced deep-learning models. The first method explored is K-means clustering, an unsupervised algorithm that does not require labeled data. While K-means provides a lightweight approach, its effectiveness is hindered by the non-homogeneous nature of coating layers. Due to variations in layers, colors, and structures, different parameter settings must be adjusted for each image, limiting its practical usability. Further details on this method are discussed in Section 2.1.3.

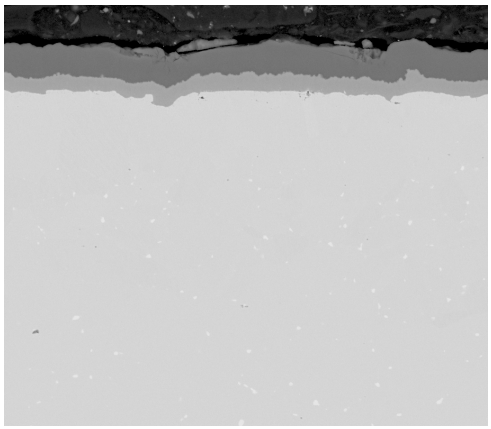
The second approach involves Convolutional Neural Networks (CNNs), which, unlike K-means, require labeled data and are computationally more demanding. However, CNNs offer good segmentation accuracy once trained. This thesis examines three levels of label precision, balancing annotation effort with segmentation performance. The original labeling come in the form of information about the thinkers of the coating layer at 10 places per picture. From this can be easily reconstructed polygon labels which are not that precise. On the other hand the creation of more precise labels requires extra time

of the scientist on top of clacical labeling ith 10 lines. Although the most precise labels yield the best results, less precise labels provide comparable outcomes while significantly reducing annotation time, as discussed in Section 2.2.

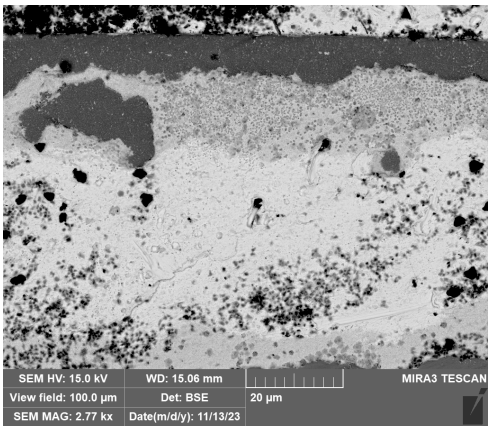
A major limitation of CNN-based methods is the computational cost of prediction. The available hardware lacks sufficient memory for complex image processing, leading to slow predictions, particularly when relying on CPU computation. Under the current setup, a single prediction requires approximately 14 seconds, compared to around one minute for manual segmentation. Thus, the trade-off between segmentation accuracy and computational efficiency remains a key consideration.

The "Segment Anything" (SAM) model [9] represents a state-of-the-art approach in image segmentation, leveraging transformers. While SAM holds significant potential to improve segmentation performance, it comes with considerably higher computational demands due to its robustness and the large number of parameters involved. Specifically, the number of parameters in the SAM model and U-Net in this thesis was measured, with the SAM model containing over 21 times more parameters than U-Net. This substantial increase in model complexity leads to a corresponding rise in hardware requirements during predictions. Consequently, the computational demands of the SAM model exceed the capabilities of the available hardware, particularly in the absence of GPU infrastructure. As a result, fine-tuning and evaluating the SAM model was not pursued in this thesis. However, as hardware resources continue to improve, this approach may become a viable option for future research.

In the following chapters, the thesis will focus on the use of K-Means clustering and label creation to develop a cohesive dataset for CNNs with U-Net architecture. It will also address the training and optimization of the model's hyperparameters. Ultimately, the best-performing model will be integrated into the research workflow, and the results obtained by the researcher using this model will be evaluated.



(a) Material sample with coating.



(b) Material sample with oxidation.

Figure 1.1: Coating and oxidation layers in the material sample.

Practical Part

2.1 Data

The dataset consists of images captured using a scanning electron microscope (SEM). These images are part of a study focused on the thickness of coating cladding layers applied to various materials, both before and after being subjected to high temperatures and aggressive environments, to assess the properties of these layers. The layers of interest are marked within red rectangular boxes in Figure 2.1. These layers exhibit varied characteristics, such as differences in color, number of layers, and degrees of damage.

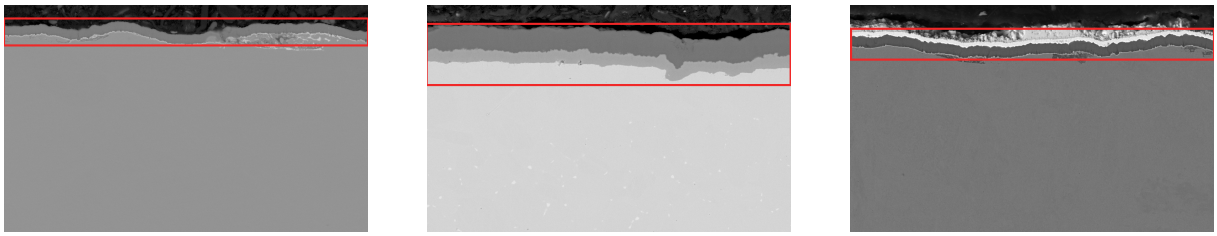


Figure 2.1: SEM images with highlighted coating layers in red boxes.

2.1.1 Manual Data Processing

The images are initially obtained in *.tif format after the measurements with the SEM. Then they are processed using Fiji, an open-source image processing software that is a distribution of ImageJ [15]. A predefined template consisting of 20 lines is used for measurements, each corresponding to a specific measurement region. The first ten lines, indexed 1 to 10, are employed to measure the thickness of the coating layer, while the second set of ten lines, located beneath the first, are used to measure oxidation. The lines are evenly distributed across the image's width and share a constant x-coordinate.

Subsequent to the initial measurements, each of the 20 lines is manually adjusted to align with either the oxidation or the coating layer. After all adjustments are completed, the measurements are exported to Excel for statistical analysis. If a layer is absent at the

location of a predefined line, the line is left in its position but is later marked as 0.0 in the corresponding Excel spreadsheet.

The images are grouped into batches of approximately 30, with each batch containing images of the same material, though different parts of the material are captured in each image, leading to a high degree of similarity among the images in each batch. The initial image of each batch typically takes longer to process since the line adjustments from the previous image are reused for the other images in the same batch. As the remaining images within a batch are more similar, they require less time for adjustment. The primary challenges arise in images where significant oxidation is present, which may destroy the coating layer.

Figure 2.2 illustrates the measurements performed in Fiji. Lines 1 to 10, used for measuring the coating layer, are shown. In this example, no oxidation layer is present.

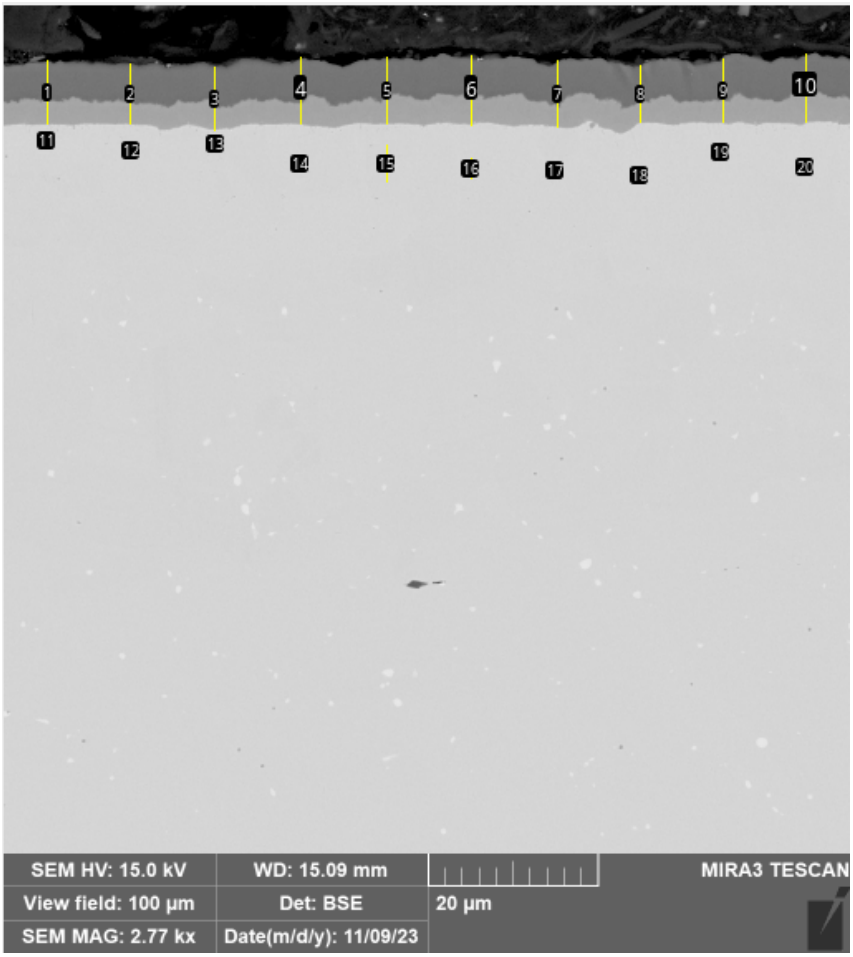


Figure 2.2: Measurement in Fiji software.

Although oxidation and coating layers are both critical for material characterization, this thesis focuses specifically on automating the measurement of coating layer thickness in SEM images. Coating layers prevent underlying materials from oxidizing, making oxidation layers less common in samples that feature coatings.

2.1.2 Fiji Adjustments

The first modification involved the customization of the ‘StartUpMacro‘ in Fiji, which runs automatically each time the program is launched. Three new buttons were incorporated into the interface: one for saving measurements, another for adjusting the scale, and a third for displaying the default lines.

The `Save Measurement` button was implemented to store data from the 20 lines required to create the dataset necessary for automation. This feature ensures that the required measurements are automatically saved.

The `Adjust Scale` button was added to streamline user interaction with Fiji, addressing the need to manually adjust the scale at the start of each session. Since the scale, visible at the bottom of the image, remains constant across all images, this functionality simplifies the process.

The `Set Default ROI` button was introduced to allow the researcher to quickly access the prepared default 20 lines for new batch.

Furthermore, the extraction of the final measured lengths into an Excel file was simplified.

2.1.3 Dataset

As previously mentioned, the first step in dataset creation was gathering data. Prior to this, results were stored exclusively in Excel tables, which included the lengths of the measurement lines but lacked positional information, rendering them insufficient for generating a comprehensive dataset. After collecting enough data thanks to the Fiji adjustments 2.1.2, stored in a zip folder containing 20 *.roi files (each representing one measurement line), the dataset was created.

Polygon Labels

The initial set of labels was generated using measured data from ten measurement lines. The upper ten points of each line were connected to form a boundary curve, and the same process was applied to the lower points. The space between these two curves was then filled to generate the mask. However, due to missing information at the edges of the image, cropping of both the images and the masks was needed. Figures 2.3a and 2.4a illustrate that the masks do not extend to the edges of the images.

This labeling process was automated using a Python script. The script loads each image along with the corresponding zip folder containing the line measurements. The x and y coordinates of these lines are stored in arrays, which are then mapped onto an Excel sheet. Any lines marked as missing (labeled as 0) in Excel are excluded, even if they are present in the folder.

To interpolate the boundary points, cubic spline interpolation [4] was employed. The upper and lower boundary points, stored in arrays, were used to create a spline model, which was then applied to generate the boundary curve. The interpolation function was used with a smoothing parameter of s=0 to ensure that the interpolation closely followed the measured points. The space between the curves was filled, resulting in a binary mask. The mask was resized to match the original image dimensions and saved as a binary mask.

This approach enabled the automated generation of masks from the measured oxidation lines, ensuring that the labels were consistent with the scientific data.

K-means Labels

The K-means algorithm [10] is a clustering method used in unsupervised machine learning, designed to minimize a function that measures the quality of cluster formation. The algorithm selects k random points as cluster centers from the dataset, assigning each data point to the nearest center based on distance. The geometric center of each cluster is then recalculated, and the process repeats until a convergence criterion is met.

K-means is particularly effective for images with distinct pixel colors. One of the key advantages of K-means is that it does not require prior labels, but the value of k (the number of clusters) must be chosen carefully. Thus, the optimal k was determined through testing for each batch of images.

In most cases, further processing was necessary, particularly to isolate the coating layer

while eliminating noise. As illustrated in Figures 2.3b and 2.4b, K-means encountered difficulty distinguishing between oxidation and the coating layer, as their pixel values were similar and even formed a continuous cluster. Consequently, the algorithm alone was insufficient for accurately measuring the coating layer.

K-means Refined

To improve the accuracy of the K-means labels, the entire dataset was reviewed collaboratively with the researcher, and the masks were manually refined using image editing software. These manually refined masks, considered the most accurate, were validated by the researcher. The final version of the masks was then used as a test dataset for further automation.

(a) Polygon mask (b) K-Means mask (c) Refined K-Means

Figure 2.3: Masks on SEM image

(a) Polygon mask (b) K-Means mask (c) Refined K-Means

Figure 2.4: Binary masks

2.2 Convolutional Neural Networks

As discussed in the previous section, K-Means clustering struggled to distinguish between the oxidation and coating layers in complex samples. Additionally, K-Means has the limitation of requiring a fixed set of hyperparameters and postprocessing strategies, which cannot be generalized across different batches of images. This makes it unsuitable for automating the measurement process. In contrast, Convolutional Neural Networks (CNNs)[12] offer a promising solution to these challenges . CNNs are supervised machine learning models, well-suited for tasks involving spatial hierarchies in images, where the relationship between pixels and their positions is crucial.

For this task, the U-Net architecture[13] is particularly effective. U-Net has shown strong performance in various segmentation problems, especially in biomedical imaging. Unlike typical CNNs used for classification, U-Net is designed for pixel-wise classification, making it ideal for tasks requiring precise localization of objects, such as in microscopy images.

2.2.1 U-Net Architecture

The U-Net architecture consists of two main components:

- **Contracting path (Encoder)**: The left side of the U-Net includes a series of convolutional, activation, and pooling layers that progressively reduce the image's dimensions. This part extracts features from the input image.
- **Expanding path (Decoder)**: The right side of the U-Net gradually upsamples the feature maps back to the original image size. This step recovers spatial resolution lost during downsampling and ensures precise pixel-wise segmentation.

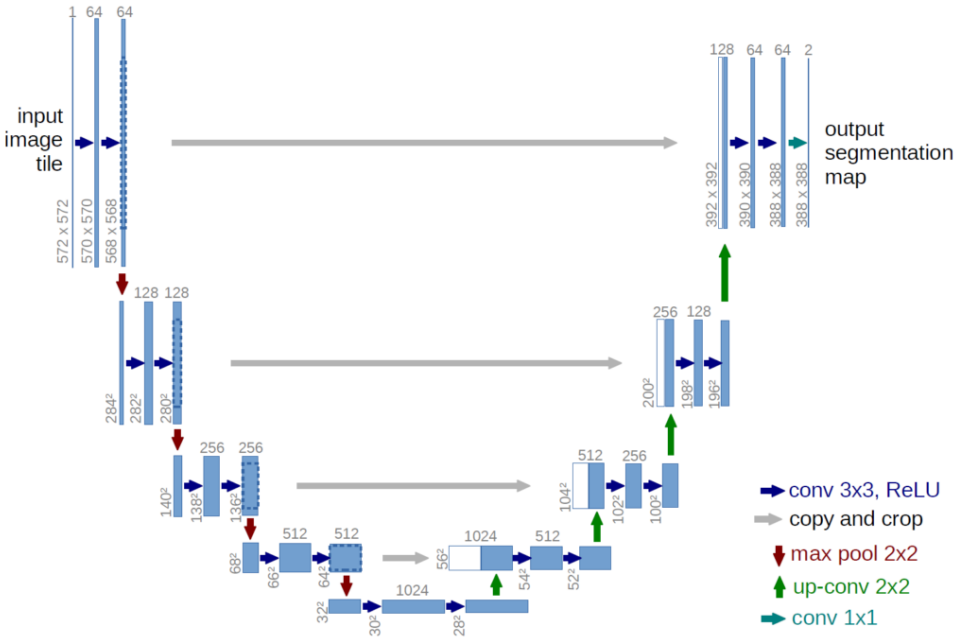


Figure 2.5: U-Net architecture [14]

The input to a U-Net model is typically an image of size $H \times W \times C$, where H is the height, W is the width, and C is the number of input channels. In Figure 2.5, the input consists of grayscale images, where $C = 1$. For RGB images, $C = 3$. The output is an image of size $H \times W \times N$, where N is the number of segmentation classes. In this case, there are two classes (binary segmentation), so $N = 2$.

Upsampling with Transpose Convolutions

The expanding path uses transpose convolutions (also called deconvolutions). Convolution reduces the image to a feature map, while transpose convolution upsamples the feature map back into an image.

Skip Connections

During the downsampling process, the network learns various features. However, as the feature maps shrink, details are lost. Skip connections allow the network to preserve these details by carrying them over from the earlier layers in the contracting path to the corresponding layers in the expanding path. This process helps retain important spatial information, improving segmentation accuracy. These connections are represented by the horizontal arrows in Figure 2.5.

Key Steps in U-Net Processing

1. In the contracting path, feature maps are progressively downsampled until the lowest resolution is reached.

2. Before each downsampling step, feature maps are saved.

3. During the upsampling process, saved feature maps are combined with the upsampled maps to recover spatial details [14].

2.2.2 U-net and Microscopy

A 2023 survey [16] highlights U-Net as an effective model for microscopic image segmentation. This success is attributed to its ability to perform well with limited training data and its relatively quick training process. The survey also emphasizes the use of the original, unmodified U-Net, which has been widely applied in fields such as cytology and geology [3] (e.g., using scanning electron microscopy). Given that the images in this dataset share similar characteristics, the U-Net architecture is considered suitable for this task.

2.2.3 Data Preprocessing – Augmentation

The dataset consists of 150 training images, which are divided into training and validation subsets, along with 47 test images. To enhance the model's ability to generalize to unseen data, data augmentation is applied to train data.

The first step involves cropping the images to a predefined $n \times n$ size, where n represents the patch size, a model hyperparameter. Cropping is done using the *CropNonEmptyMaskIfExists*[2] function. This function ensures that the cropped region always contains parts of the coating, preserving important features. The cropping is applied with a probability of $p = 1.0$, ensuring its occurrence.

Additional augmentations are applied with a probability of $p = 0.3$ to increase variability. These include:

- **Brightness and contrast adjustment:**This simulates lighting variations.

- **Sharpening:** This enhances the details in the image.

- **Horizontal flipping:** The image is mirrored along the horizontal axis to reduce positional bias.
- **Elastic transformation:** A non-linear deformation is applied to simulate distortions and improve generalization. It is applied with a probability of $p = 0.5$.

The effects of these augmentations are illustrated in Figure 2.6, where different transformations are applied to a sample image.

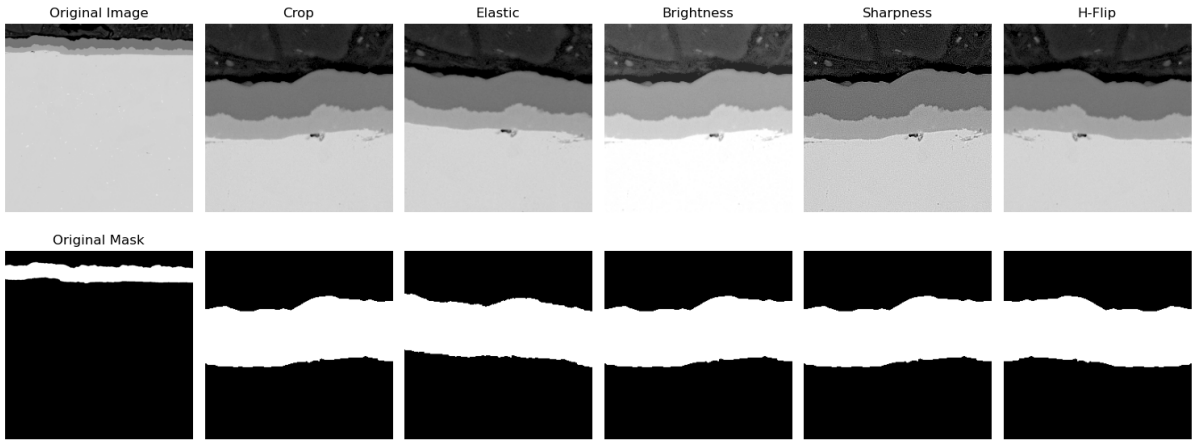


Figure 2.6: Illustration of various augmentation techniques applied to the dataset.

2.2.4 Technical Details

The `segmentation_models` library [5] was used for the U-Net implementation. Documentation for this implementation can be found here [6]. After initializing the training functions and other helper functions, hyperparameter tuning can begin. The main hyperparameters include depth, patch_size, filters, and learning rates, all of which will be tested.

2.2.5 Depth

The depth of a U-Net is determined by the number of encoder and decoder blocks. This influences the model's ability to capture features within the input image. For instance, in Figure 2.5, the depth is 5.

A deeper network (higher depth) can handle complex data with high variability, but may suffer from excessive downsampling and overfitting. In contrast, a shallower net-

work may struggle to extract relevant features and differentiate between noise and useful information.

2.2.6 Patch Size

Patch size plays a crucial role in how well the model learns different features from the image.

Smaller patches allow the model to focus on fine details but might miss the broader context, making it harder to understand relationships between different parts of the image.

Larger patches capture more context and larger structures, but may lose fine details.

To ensure that the region of interest, specifically the coating layer, is always included, the images are cropped in a way that retains this key feature.

2.2.7 Filters

The filters parameter determines the number of filters used in each encoder and decoder layer, which affects the number of feature maps generated by the model. In the code, the number of filters in the decoder is calculated as follows:

```
decoder_channels = [filters * 2**i for i in range(depth, 0, -1)]
```

This calculation, inspired by the original U-Net architecture, doubles the number of filters at each downsampling step [1]. For example, with `filters = 8` and `depth = 3`, the number of filters would be:

Encoder: 8, 16, 32

Decoder: 32, 16, 8

Using more filters allows the model to learn more complex patterns. However, it can also increase the risk of overfitting when there is insufficient training data. On the other hand, using fewer filters might prevent the model from capturing important features, reducing its performance.

2.2.8 Hyperparameter values

In this project, all combinations of the following hyperparameters were tested:

- **Patch size:** 128, 256
- **Depth:** 3, 4, 5
- **Filters:** 8, 16, 32

2.2.9 Learning Rate and Schedulers

During training, the model's parameters are adjusted to find the optimal solution. The **Adam optimizer** [8] is used for this purpose. Adam (Adaptive Moment Estimation) calculates adaptive learning rates for each parameter.

Since Adam adapts the learning rate, manually modifying it can have a small impact but still affect performance. The learning rate is chosen from a range between 0 and the set value.

- **Linear Scheduler:** The learning rate starts at the set value and decays linearly. The **end factor** multiplies the learning rate to finish at **end factor * lr**.
- **Warmup Cosine Scheduler:** The learning rate starts at a predefined value and decreases over `T_0` epochs, following a cosine annealing schedule. After reaching the minimum, the learning rate resets and begins from the previous value, restarting the cycle.

The following schedulers and values were tested:

- **LR values**: 0.01, 0.001, 0.0001
- **Linear scheduler**: With end factors of 0.01 and 0.001
- **Warmup Cosine scheduler**: With `T_0`= 25 and `T_0`= 50

Figure 2.7 shows the learning rate schedulers. On the left, the linear scheduler spans 200 epochs, starting at 0.01 and ending at 0.001. On the right, the warmup cosine scheduler starts at 0.01 and ends at 0.00, with a warmup phase during the first 50 epochs.

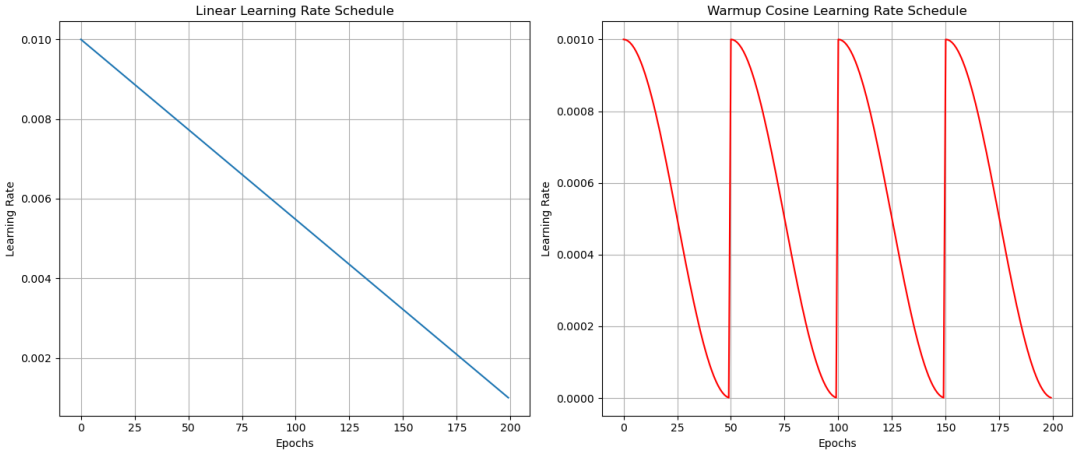


Figure 2.7: Learning Rate Schedulers (LRS)

2.2.10 Loss Function

The loss function $L(Y, \hat{Y})$ is used to evaluate prediction errors. Since the model's performance on test data is assessed using the Intersection over Union (IoU) metric, IoU is also used as the loss function during training and validation.

The IoU is defined as:

$$\text{IoU} = \frac{\text{intersection} + \text{smooth}}{\text{union} + \text{smooth}}$$

Where:

- The intersection is the number of pixels correctly predicted as the foreground (True Positives).
- The union consists of all pixels predicted or labeled as foreground. It equals the sum of True Positives (TP), False Positives (FP), and False Negatives (FN).

Thus, the IoU formula becomes:

$$\text{IoU} = \frac{\text{TP} + \text{smooth}}{\text{TP} + \text{FP} + \text{FN} + \text{smooth}}$$

The smoothing constant (smooth = 1) prevents division by zero. The loss function returns $1 - \text{IoU}$, ensuring that lower values indicate better model performance.

2.2.11 Training

After defining the key components, model training begins. The dataset is divided into 98 training images and 32 validation images. The training, validation, and testing labels come from the K-means refined dataset, with the validation set remaining fixed as a single batch.

During training, just the training data are shuffled each epoch to reduce the risk of overfitting. The shuffling function selects 32 random images for each batch. Training proceeds for either 200 or 150 epochs, with each epoch consisting of both training and validation phases.

Training Phase

In each epoch, the training dataset is shuffled, and three batches of 32 images are formed. The model predicts the binary mask for each batch and computes the training loss. Back-propagation adjusts the model's parameters using the optimizer based on the computed loss.

Validation Phase

At the end of each epoch, the validation set (a single batch) is evaluated using the loss function. The model's parameters are not updated during this phase. If the validation loss is lower than the previously recorded minimum, the model and its parameters are saved, and the minimum validation loss is updated.

2.2.12 Optimization Strategy

1. Three optimization runs were performed to determine the optimal learning rate (LR) and learning rate scheduler. Details about the hyperparameters tested are provided in section 2.2.9.

2. The main optimization focused on the three key hyperparameters, using the best LR and LR scheduler from the initial optimization. The specific hyperparameters optimized are discussed in section 2.2.8.

3. After identifying the best hyperparameters, two additional runs were conducted. One used polygon labels, while the other used non-refined K-means labels.

2.2.13 Optimization results

The optimization of the learning rate and learning rate schedulers is discussed first. The schedulers were tested with the hyperparameters **depth = 3**, **patch size = 128**, and **filters = 8**. These values were chosen as they offered the best computational efficiency among the tested ones. Initially, the warmup cosine scheduler was optimized. The contour plot below illustrates the relationship between T_0 and the learning rate. The optimal point occurs at $T_0 = 50$ and lr = 0.01, resulting in a minimum objective function value (IoU validation loss) of 0.03306.

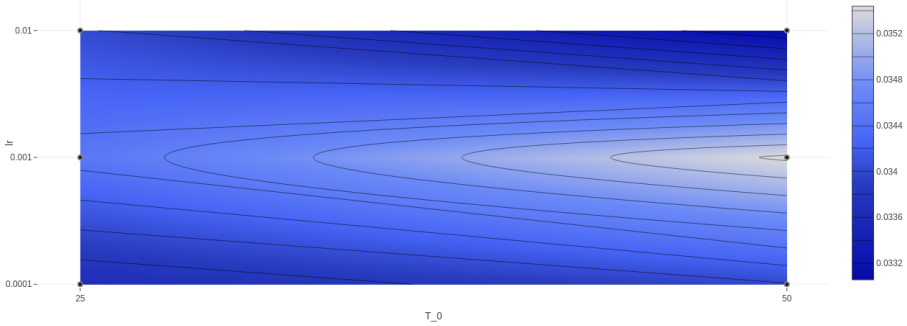


Figure 2.8: Objective function contour: effect of T_0 and Learning Rate

The best results for different learning rate schedulers are summarized in Table 2.1. The linear scheduler achieved the lowest objective function value of 0.03075 with a learning rate of 0.001 and an end factor of 0.001. The optimization without a scheduler resulted in a minimum objective function value of 0.03133 at a learning rate of 0.01.

Schedule	Learning Rate (lr)	End Factor / T_0	Val Loss
Warmup Cosine	0.01	50	0.03306
Linear	0.001	0.001	0.03075
None	0.01	—	0.03133

Table 2.1: Best results for different Learning Rate Schedulers.

Next, the main optimization was conducted using the linear scheduler and the best values from the previous step. The hyperparameters discussed in section 2.2.8 were optimized, resulting in 18 combinations (2 * 3 * 3 = 18). The complete results are shown in Table 2.2, with the best result found at index 9.

Idx	**Val Loss**	**Depth**	**Filters**	**Patch Size**
0	0.03315	5	8	256
1	0.03902	5	8	128
2	0.03204	3	8	256
3	0.03365	5	16	256
4	0.02999	5	32	256
5	0.03192	4	8	128
6	0.03047	4	16	256
7	0.03307	3	8	128
8	0.03406	4	8	256
9	**0.02892**	**5**	**16**	**128**
10	0.03502	3	32	128
11	0.03136	5	32	128
12	0.03001	4	16	128
13	0.02910	4	32	128
14	0.03108	3	16	256
15	0.03036	4	32	256
16	0.03271	3	32	256
17	0.03289	3	16	128

Table 2.2: Validation loss values with Depth, Filters, and Patch Size

Figure 2.9 presents the validation loss values over 150 epochs. Initially, both curves decrease rapidly, then oscillate around similar values, suggesting that the parameters no longer change significantly. The yellow line represents the best optimization, while the purple line represents the second-best optimization. From epoch 100 onward, the results stabilize. The optimization at index 9 provides the best performance. This model will be evaluated on the test dataset and implemented into the researcher's workflow.

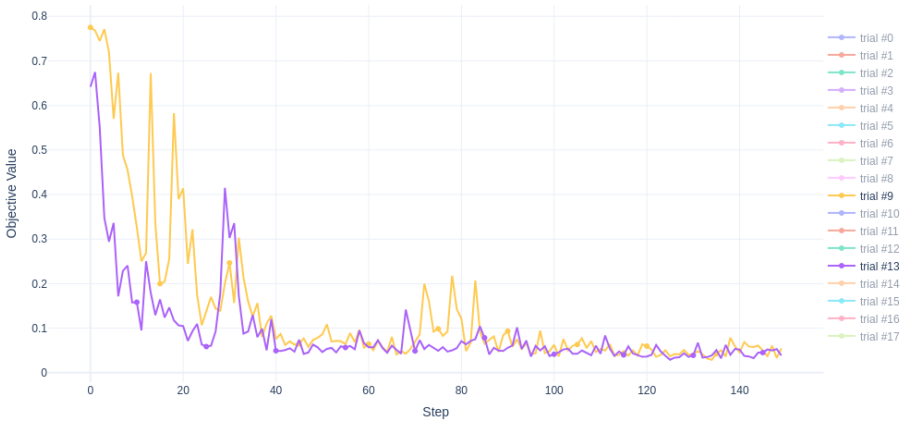


Figure 2.9: Validation loss over 150 epochs

Additionally, Figure 2.10 shows the feature importances, indicating that patch size has the greatest impact on the results. This insight can be used in future work.

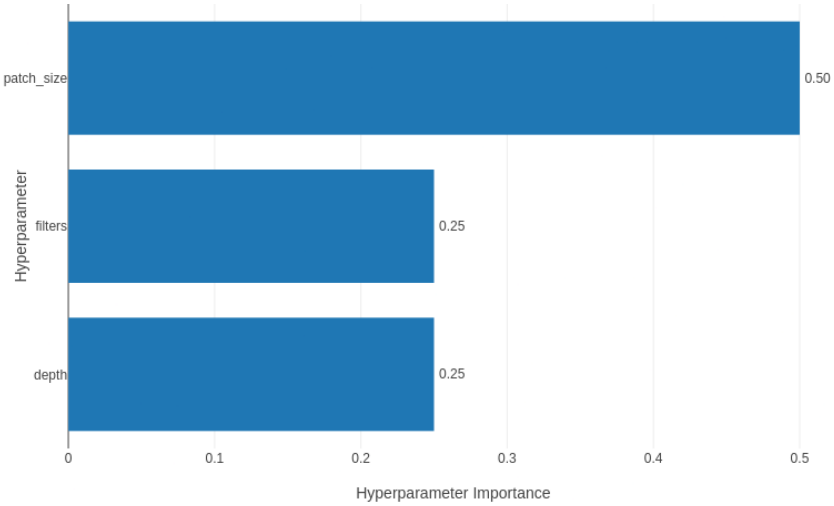


Figure 2.10: Hyperparameter importance

The final two runs were conducted using the best hyperparameters, but with datasets that included labels generated by polygons and K-means clustering. The corresponding validation loss values are presented in Table 2.3.

Metric	**K-means Refined**	**K-means**	**Polygon**
Validation Loss	0.02892	0.03118	0.09698

Table 2.3: Validation loss values for different labeling methods

Since both validation sets use labels from their respective training sets, these values may not be fully reliable. It is better to assess the model's performance on unseen images using test labels generated from K-means refined clustering. Although the K-means and

polygon labels are not intended for use in the final workflow, evaluating their performance on real test data could still provide useful insights.

2.2.14 Evaluation on Test Data

Two primary evaluation metrics were used:

- **Mean IoU**: The intersection over union (IoU) is calculated and averaged across all test data.
- **Min IoU**: The minimum IoU value is used to assess the worst-case scenario.

The evaluation results are presented in Table 2.4:

Metric	**K-means Refined**	**K-means**	**Polygon**
Mean IoU	0.95473	0.83111	0.92177
Min IoU	0.83872	0.17518	0.78456

Table 2.4: Evaluation Results on Test Data

The results reveal some key patterns. Although the "polygon" model was trained on validation data with polygon labels, it still performed well. This suggests that polygon labels could be used instead of K-means refined labels when applying the model to other parts of the images, such as various oxidation layers. They can also be used when re-training the model once more data is gathered. In such cases, K-means refinement may not be necessary, as polygon labels can be efficiently generated with a single script. On the other hand, the K-means labels without refinement performed poorly. This likely occurred because the model struggled to differentiate between oxidation and coating layers, which were not well-separated in the training data. In contrast, the K-means refined labels provided the best results on unseen data.

Figure 2.11: Original image, ground truth, prediction (K-means refined model)

Figure 2.12: Original image, ground truth, prediction (K-means model)

Figure 2.13: Original image, ground truth, prediction (polygon model)

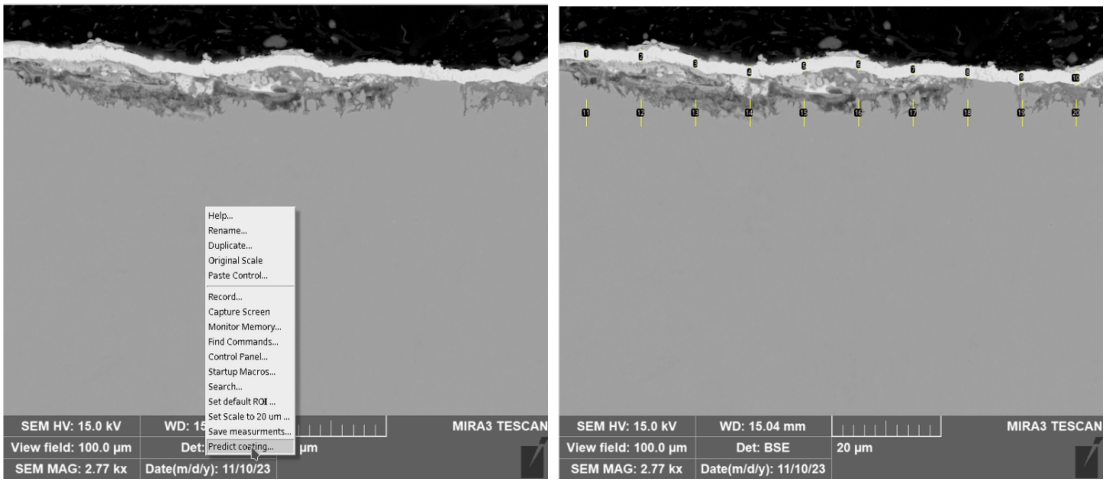
Figure 2.12 illustrates that the model struggles to recognize the oxidation and coating layers, resulting in poor performance. In contrast, Figures 2.13 and 2.11 demonstrate that the models can accurately predict coating layers, even though the images are complex and distinguishing between the layers is difficult.

2.3 Integration of the Model

After training the model, it was crucial to integrate it smoothly into the existing workflow to avoid requiring the researcher to learn new software. The model was integrated into the Fiji software, which was already in use, instead of creating a new application.

The first step involved adding a new button to the Fiji interface, as shown in Figure 2.14a. Pressing this button triggers a Python script from the 'StartupMacro', with the path to the image being analyzed passed as an input argument. The script, stored in the Fiji application folder along with the macros and model parameters, loads the image and crops the lower part to remove irrelevant information. It then predicts a binary mask, processes it, and extracts the ten most relevant lines.

By default, another set of ten lines is added beneath the first set. The resulting data, containing the twenty lines, are saved in a zip folder. The path to this zip folder is then returned, and Fiji opens the folder to display the predicted lines to the researcher. If the researcher is dissatisfied with the predictions, they can modify the lines as needed.



(a) Menu with *Predict coating...* button (b) Predicted lines

Figure 2.14: Example workflow in Fiji

2.4 Results - Time Experiment

Following the model integration and the update to the Fiji setup, a time measurement study was conducted to assess time savings. Eight images were selected from four batches, with two images chosen from each batch. The experiment was designed as follows:

The first image in the set is measured manually, while the second is analyzed using the prediction model. In the first image from the first set, only the predefined default lines were used, simulating the first image in each batch where the researcher does not have lines similar to the coating. The first image from the second set used predefined lines from another image in the same batch. Since the images in each batch are similar, the researcher typically spends less time adjusting these lines. The third set followed the same approach as the second, but with an image where the model did not predict all the lines correctly. The fourth set included a challenging image with complex oxidation, where the model's predictions were less accurate, though the same approach as the second set was used on the first image.

The "Model" column in Table 2.5 indicates whether the model was used for line prediction. The model was not used for the first image in each set but was used for the second image.

The "Predict" column shows the time from pressing the "Predict coating" button until the predicted lines were displayed, which took approximately 14 seconds, depending on the computer hardware.

The "Measure" column represents the total time from starting the prediction (or manual measurement with predefined lines) to when the researcher finished editing.

The "Add_Excel" column reflects the time the researcher took from the start until adding the measurements to the Excel file. When using the model, the researcher worked with an updated version of Fiji, which included the improvements described in *Fiji adjustments* (2.1.2). Consequently, Fiji, with the model, was also able to export to Excel faster.

Type	Model	Predict	Measure	Add_Excel
First Batch	N	/	01:12.00	01:30.00
First Batch	Y	00:14.00	00:28.00	00:38.44
Normal	N	/	00:57.32	01:26.00
Normal	Y	00:14.00	00:25.20	00:38.34
Model Wrong	N	/	01:09.00	01:32.00
Model Wrong	Y	00:14.00	00:57.32	01:10.00
Hard Oxidation	N	/	01:05.00	01:26.00
Hard Oxidation	Y	00:14.00	00:45.24	00:59.00

Table 2.5: Time measurement results for four sets of images. All times are in min:s:ms.

2.4.1 Time per one batch

A theoretical batch of 30 images was considered for this experiment. It was assumed that the first image in the batch would take the longest to process, as is explained in 2.1.1. Additionally, two images were expected to experience significant model errors, while two others presented challenging oxidation conditions. The time spent on measurement with the model was approximated as:

$$\text{Time with model} = 28+(25\times25)+(57\times2)+(45\times2) = 857 \text{ seconds} = \frac{857}{60} \approx 14.28 \text{ minutes.}$$

The results of the time calculations are summarized in Table 2.6.

Scenario	Time (seconds)	Time (minutes)
Model - Measure	857	$\frac{857}{60} \approx 14.28$
Manual - Measure	1765	$\frac{1765}{60} \approx 29.42$
Model - Total	2596	$\frac{2596}{60} \approx 43.27$
Manual - Total	1246	$\frac{1246}{60} \approx 20.77$

Table 2.6: Comparison of time taken in different scenarios

The percentage reductions in time are presented in Table 2.7:

Scenario	Percentage reduction
Reduction in measurement time	$\frac{29.42-14.28}{29.42} \times 100 \approx 51.5\%$
Reduction in total time	$\frac{43.27-20.77}{43.27} \times 100 \approx 52.0\%$

Table 2.7: Percentage reduction in measurement and total time

Thus, using the model results in significantly faster processing times for measurement and saving results. Although these measurements are approximations, the results clearly demonstrate that the model improves the speed of the process.

2.5 Results - Precision

After the model was integrated into the workflow, data were collected from the research process. A total of 90 images were analyzed using the model. For each image, two folders were generated: one containing the predicted lines and another for the lines that could be manually adjusted by the researcher. The results from three separate batches are summarized below.

The first and second batches showed high accuracy, requiring minimal manual adjustments. However, the third batch exhibited reduced performance. This decline was attributed to the introduction of different types of layers. Notably, the model also detected the white oxidation layer above the coating layer, as shown in Figure 2.15. This layer is visually similar to other structural layers, as seen in Figure 2.1 (right image).

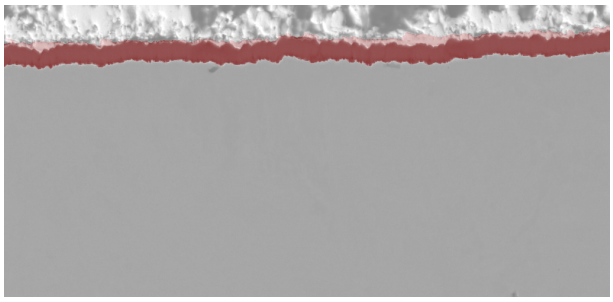


Figure 2.15: Image from the third batch, highlighting a poor prediction.

These results suggest that the model performs well overall, requiring only minor adjustments in most of the cases. However, the findings also highlight the need to obtain a bigger dataset. The current dataset is still relatively small. These new data can be

incorporated into the dataset using polygon labels, which can be efficiently created using automated scripts. After this, the model can be re-trained to make it more robust. This will create a feedback loop where labels can be generated faster, leading to a larger dataset, which in turn, after re-training, improves overall precision and results in time savings.

The precision of the model's predictions was evaluated by counting the number of points that required manual adjustment after the predictions were made. Table 2.8 presents the summary statistics for the three analyzed batches.

Batch	**Mean difference (pixel)**	**Unchanged Points (%)**
1	1.11	93.3% (560/600)
2	0.57	97.3% (584/600)
3	5.58	22.7% (136/600)

Table 2.8: Analysis of model predictions across three batches.

As shown in Table 2.8, the first two batches display a high percentage of unchanged points, indicating that the model's predictions are highly accurate. In contrast, the third batch shows a significantly higher mean difference and a lower percentage of unchanged points. This highlights the need for additional data to improve the model's performance.

Conclusion

This thesis successfully addresses the challenge of automating a time-consuming manual process in the nuclear industry. Researchers are often required to perform repetitive, labor-intensive measurements. The primary goal was to develop a solution that integrates seamlessly into the researcher's workflow without causing disruptions. The thesis focused on a specific category of coating-layer images with relatively homogeneous properties, in contrast to other images in the dataset. It was shown that segmentation of more complex, heterogeneous images is a viable direction for future work.

Three approaches were evaluated. First, the classical K-means algorithm was found to be computationally efficient, but it lacked universal applicability due to the variability of coating layers. This made it unsuitable for a streamlined workflow without extensive post-processing. Second, the Segment Anything Model (SAM) was considered, but hardware limitations made it infeasible for the given problem. Finally, Convolutional Neural Networks (CNNs) were explored. This solution required labeled data. Three types of labels were developed. The model was trained on the most precise set, but it was found that even the least manually intensive labels were effective for training, despite being less accurate. This insight suggests that efficient model retraining is also possible with this kind of labels and that the model could be adapted to handle more complex images in the future.

The optimization and testing of the model were thoroughly discussed, with results measured using the Intersection over Union (IoU) metric. The model achieved a mean IoU of 95.47%, demonstrating its high accuracy in segmenting coating layers.

After integrating the model into the researcher's workflow, the practical impact was clear. The time required for measurement tasks was reduced by half. For images similar to the training dataset, over 90% of predicted measurements required no further manual adjustments. However, for images from batches with significantly different characteristics, performance declined, highlighting the need for further re-training.

With the code infrastructure in place, future improvements can focus on streamlining

the training process and enhancing prediction accuracy. The current model was trained on fewer than 100 images from a limited number of batches, which is insufficient for creating a fully robust model. Nevertheless, even when incorrect, the model's average deviation was approximately five pixels, resulting in substantial time savings. This efficiency will accelerate the generation of additional labeled data, creating a feedback loop for continuous improvement. The iterative workflow is illustrated in figure 3.1.

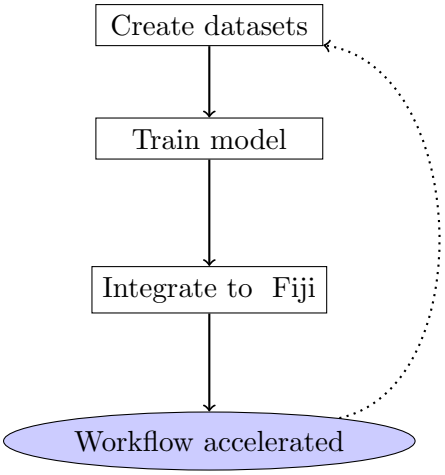


Figure 3.1: Workflow of model integration and performance improvement.

In conclusion, this thesis automates a portion of the researcher's measurement process and establishes a pipeline for continuous model retraining and improvement. It also demonstrates that polygon masks, generated directly from researcher annotations, are sufficient for training segmentation models. These results suggest that similar models can be developed for other types of images within the researcher's domain. The improvements brought by this work will lead to time savings and improved efficiency. This work highlights the practical impact of deep learning in automating image analysis tasks in the nuclear industry.

Appendix

The complete implementation of this thesis, including preprocessing, label generation, model training, and evaluation scripts, is available at [GitHub Repository Link (will be added in future when it has documentation, proper README,...)].

This project benefited from various open-source libraries that facilitated image processing, deep learning model training, and evaluation. A detailed list of dependencies and libraries used can be found in the repository's requirements.txt file.

Additionally, this text was refined and grammar-checked using ChatGPT [11], and Grammarly [7].

Bibliography

[1] A. Buslaev, A. Parinov, E. Khvedchenya, V. I. Iglovikov, and A. A. Kalinin. Albumentations: fast and flexible image augmentations. *ArXiv e-prints*, 2018.

[2] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020.

[3] Zhuoheng Chen, Xiaojun Liu, Jijin Yang, Edward Little, and Yu Zhou. Deep learning-based method for SEM image segmentation in mineral characterization, an example from Duvernay Shale samples in Western Canada Sedimentary Basin. *Computers & Geosciences*, 138:104450, February 2020.

[4] F. N. Fritsch and R. E. Carlson. Monotone piecewise cubic interpolation. *SIAM Journal on Numerical Analysis*, 17(2):238–246, 1980.

[5] Pavel Iakubovskii. Segmentation models pytorch. `https://github.com/qubvel/segmentation_models.pytorch`, 2019.

[6] Pavel Iakubovskii. Segmentation models pytorch, 2019. Accessed: 2025-02-04.

[7] Grammarly Inc. Grammarly: Ai-powered writing assistant, 2025. Accessed: 2025-02-19.

[8] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[9] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[10] Stuart Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

[11] OpenAI. Chatgpt: Language model for text generation and assistance, 2025. Accessed: 2025-02-19.

[12] Keiron O'Shea and Ryan Nash. An Introduction to Convolutional Neural Networks, December 2015.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation, May 2015.

[15] Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, et al. Fiji: an open-source platform for biological-image analysis. *Nature methods*, 9(7):676–682, 2012.

[16] Jian Wu, Wanli Liu, Chen Li, Tao Jiang, Islam Mohammad Shariful, Yudong Yao, Hongzan Sun, Xiaoqi Li, Xintong Li, Xinyu Huang, and Marcin Grzegorzek. A state-of-the-art survey of U-Net in microscopic image analysis: from simple usage to structure mortification. *Neural Computing and Applications*, 36(7):3317–3346, March 2024.

List of Figures

List of Tables

List of Abbreviations

- **TP** - True Positive
- **TN** - True Negative
- **FP** - False Positive
- **FN** - False Negative
- **CNN** - Convolutional Neural Network
- **LR** - Learning Rate
- **LRS** - Learning Rate Scheduler
- **SEM** - Scanning electron microscope
- **IoU** - Intersection over union
- **SAM** - Segment Anything Model