metals

MDPI

Article

A Multi-Branch U-Net for Steel Surface Defect Type and Severity Segmentation

Robby Neven * and Toon Goedemé

PSI-EAVISE, KU Leuven, 2860 Sint-Katelijne-Waver, Belgium; toon.goedeme@kuleuven.be
* Correspondence: robby.neven@kuleuven.be

Abstract: Automating sheet steel visual inspection can improve quality and reduce costs during its production. While many manufacturers still rely on manual or traditional inspection methods, deep learning-based approaches have proven their efficiency. In this paper, we go beyond the state-of-the-art in this domain by proposing a multi-task model that performs both pixel-based defect segmentation and severity estimation of the defects in one two-branch network. Additionally, we show how incorporation of the production process parameters improves the model's performance. After manually constructing a real-life industrial dataset, we first implemented and trained two single-task models performing the defect segmentation and severity estimation tasks separately. Next, we compared this to a multi-task model that simultaneously performs the two tasks at hand. By combining the tasks into one model, both segmentation tasks improved by 2.5% and 3% mIoU, respectively. In the next step, we extended the multi-task model using sensor fusion with process parameters. We demonstrate that the incorporation of the process parameters resulted in a further mIoU increase of 6.8% and 2.9% for the defect segmentation and severity estimation tasks, respectively.

Keywords: steel surface defects; visual inspection; computer vision; deep learning; semantic segmentation

Citation: Neven, R.; Goedemé, T. A Multi-Branch U-Net for Steel Surface Defect Type and Severity Segmentation. *Metals* **2021**, *11*, 870. https://doi.org/10.3390/met11060870

Academic Editor: Diego Celentano

Received: 27 April 2021
Accepted: 22 May 2021
Published: 26 May 2021

1. Introduction

Producing sheet steel material involves a cumbersome process consisting of many delicate steps, which include heating, rolling, drying, and cutting. In each of these steps, different machines manipulate the sheet steel material, which can all induce surface defects on the product. To ensure a qualitative end product, visual inspection of the sheet steel material is a crucial part of the production process. With a rolling speed of multiple meters per second, this remains a challenging task.

Over the years, many researchers have studied this field. In [1], a broad overview of the most common methods that have been used to date can be found, ranging from statistical methods, e.g., thresholding, clustering, edge-based; spectral methods, i.e., Fourier transforms, Gabor filters, wavelet transforms; and model-based methods, i.e., Markov random fields, Weibull models, to the latest, more modern research on machine learning, which includes supervised, unsupervised, and reinforcement learning. While the classic approaches have proven their efficiency, these methods heavily depend on a fixed environment, i.e., no change in lighting, surface finish, change in defect occurrence, etc., which will impact the performance of the algorithm drastically. On the other hand, machine learning algorithms will learn to generalize and are more robust to these changes in the environment.

However, the downside of the machine learning methods is that these are very data-hungry. Since these algorithms learn by example, the performance of the model will increase with an increasing dataset that exists of manually labeled image examples. Constructing such a dataset is a very labor-intensive task performed by trained employees, which is why many manufacturers still rely on the conventional computer vision approaches. However, due to publicly released datasets such as the NEU (North Eastern

University) dataset [2] and the Severstal Kaggle challenge [3], researchers can investigate this problem and implement new methods.

However, from a real-life industrial point of view, we conclude that these academic datasets over-simplify the task. In real-life cases, the number of different defect classes to be detected is much higher than the three or six types in these datasets. Moreover, real quality inspection systems must also produce a severity grade for each surface defect.

In this paper, we tackled this advanced sheet steel inspection research question while focusing on a deep learning approach using convolutional neural networks (CNNs) ([4]). Since the rise of deep learning, many computer vision tasks such as image classification, object detection, and image segmentation have seen significant progress for many industrial applications, including autonomous driving [5], medical analysis [6], geosensing [7], topology optimization [8], etc. This has been due to the many advancements in model architectures and training improvements.

This paper brings three novelties in the field of visual inspection of steel sheet material:

- Inspired by the real-life industrial dataset that we gathered, we study the detection of 16 different types of steel surface defects, of which some are almost visually similar but have a different root cause. Moreover, for each detected surface defect, a severity grading is also to be produced by the inspection system in order to produce a fine-grained quality report. We demonstrate that the joint training of these two tasks simultaneously yields better results than when treating them separately.
- We strive towards pixel-perfect segmentation of the surface defects. We show that our network can produce these, although some of the annotations of our dataset are in the less precise but cheaper bounding boxes format instead of pixel-perfect segmentation maps.
- We demonstrate that fusing the image data with steel production process parameters during classification is very advantageous for the detection accuracy of the proposed system.

While recent research including [9–12] only focuses on one task, i.e., detecting or classifying defects into 1 to up to 6 classes, our industrially inspired goal is to perform three tasks: detect defects, classify them into 16 different types, and make an estimate of the severity level of each defect. While the first task enables the manufacturers to improve the quality of the end product by cutting out the detected defects, the second task provides an insight into the occurrence of different types of defects. This can be an indicator of process malfunction, which induces the defects. Moreover, some defect types have to be immediately detected to prevent damage to the machinery. The last task, i.e., estimating a severity level, is another process malfunction indicator, which can be responded to. In addition, the severity estimation can be a deciding factor to remove certain defects or not. This enables the manufacturer to indicate a quality label for the end product, which indicates if there are minor defects present or more severe ones.

Some of the more popular deep learning methods that can be used to tackle this problem are image classification and object detection. The first approach (e.g., [9,11]) classifies entire images without an indication of where the defects occur in that particular image. The second approach (e.g., [10,12]) draws a bounding box around each defect, which gives additional information on the location and size of the defect. However, the bounding box is a crude estimation of the defect and will contain a significant portion of defect-free material. Another approach is image segmentation, which segments the image into background/foreground pixels. An extension to this method is semantic segmentation ([13,14]), which additionally classifies the foreground pixels into different classes. This algorithm provides a pixel-perfect defect mask, in contrast to bounding boxes, where a large portion contains background pixels.

In our approach, we propose to use a semantic segmentation approach to segment each defect from the camera input which comes in at a high frame rate depending on the roller speed. The reason that we chose to segment the pixels of the images instead of using object detection is that, besides the crude bounding box approximations that object

detection provides, the surface defects that we target are not rigid objects. Whereas object detection methods would work better on rigid-objects datasets, our dataset consists of non-rigid objects and is, therefore, better suited to segmentation.

Since deep learning algorithms are very data-hungry, we first constructed a real-life industrial, manually labeled dataset containing thousands of images. Ideally, these should all be annotated with pixel-perfect segmentation maps. However, as bounding box labels around the defects are easier to annotate, our dataset partly contains these. In this paper, we show that we can treat these bounding-box annotations as crude rectangular segmentation maps and demonstrate that our network still learns to produce pixel-perfect maps.

We trained a multi-branch segmentation network that simultaneously performs two tasks: to segment different types of defects and to estimate their severity level. The network can segment 16 different defect types and estimates a suitable severity level between one and three.

Additionally, we extended the multi-branch network by incorporating process parameters to improve both the defect segmentation and severity estimation. Indeed, we know that, for human quality inspectors, the context is important in detecting some defects. They know by experience that some defects have a higher chance of occurring in certain circumstances: when the cylinder pressure is above a certain value, for a specific material alloy, when the temperature of a certain heater is too high or too low, etc. Because, for every image in our dataset, a large number of such process parameters are known, we indeed successfully were able to train our neural network in order to take such context as additional input.

Figure 1 shows an overview of our approach.

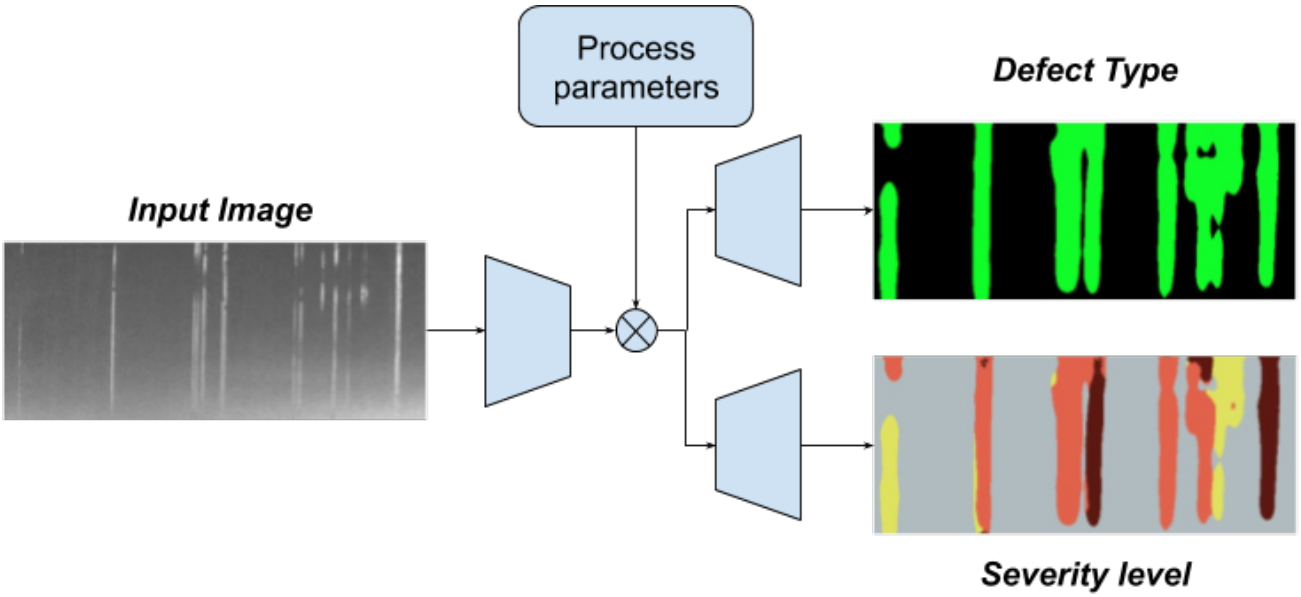

Figure 1. Overview of our approach. A multi-task model segments the input image into 16 different defect classes while also estimating a suitable pixel-based severity level. To enhance the segmentation performance, additional steel production process parameters are fed into the model.

2. Method

2.1. Dataset

To train our model, we gathered a dataset of around 10K images, of which 4K have a resolution of 280 × 780 and 6K have a varying resolution of 128 × 1920 to 1920 × 1920. As mentioned before, some of the images are annotated with pixel-perfect masks and the remainder with bounding box masks to speed up labeling. Figure 2a gives an overview of the amount of bounding-box and pixel-perfect labeled pixels per defect type, while Figure 2b shows how these defect type pixels are distributed among the different severity

levels. Remarkably, we noticed that we are able to treat the bounding-box labels as weakly labeled pixel-perfect labels. We use the filled-in bounding boxes as rectangular-shaped segmentation maps, and observe that the segmentation network after training produces maps that better shrink around the actual defect pixels. In the future, these bounding-box labels will be fine-tuned to pixel-perfect labels using model-assisted labeling. The influence of the bounding-box labels is slightly noticeable, with some bounding-box-like output predictions visible. However, since some classes are highly imbalanced, these bounding-box labels are needed in order to have extra examples for these underrepresented classes. Nevertheless, to reduce the influence of the false-positive pixels within the bounding-box targets, a weight is applied to the pixel-perfect labels to increase their influence during training.

Our dataset contains 16 types of defects, which, from now on, we will further refer to as A-P, and which can be seen in Figure 3. Some types can be easily distinguished. Other types share visual features and are harder to distinguish from each other. Moreover, each defect is annotated with a severity score of 1 to 3, indicating the severity of the defect. This severity level is given by a trained labeler and is based on pre-determined quality regulations. Some examples of different severity levels can be seen in Figure 4. While severity 3 defects are clearly visible, low-severity defects are harder to distinguish from the background, which makes them difficult to detect. Due to the difficulty of visually detecting and distinguishing defects from each other, we are aware that our dataset is not 100% error-free and may contain missing or erroneous annotations, which will impact the training of the model. Nevertheless, due to the size of our dataset, we will neglect these deficiencies for now and this will be investigated in future work.

Figure 2 shows that the dataset deals with two imbalance problems: both the occurrence of different defect types and the severity levels are highly imbalanced. The first imbalance is due to the fact that some defects occur on a regular basis, while other defects only occur on a monthly or yearly basis, which makes it difficult to gather many examples. The severity imbalance is due to the fact that some defect types are inherently extremely severe (e.g., holes, folds, etc., which can cause damage to the machinery) and will primarily appear as a level 3 severity defect (e.g., defect type 'G'). Other defect types, such as rough patches, stains, etc., are primarily classified as lower-level severity defects and will therefore contain less severity 3 defects (e.g., defect type 'A').

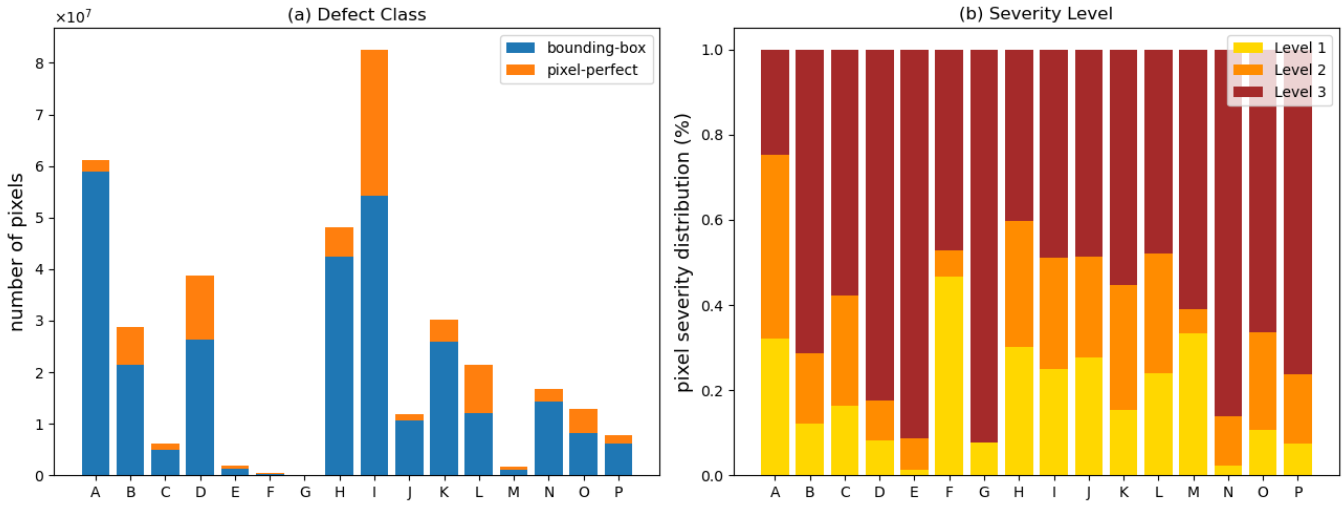

Figure 2. (**a**) Number of bounding-box and pixel-perfect labeled pixels per defect type. (**b**) Severity occurrence per defect type. It is clear that our dataset deals with two imbalance problems: both defect type (across dataset) and severity levels (per defect class) are imbalanced.

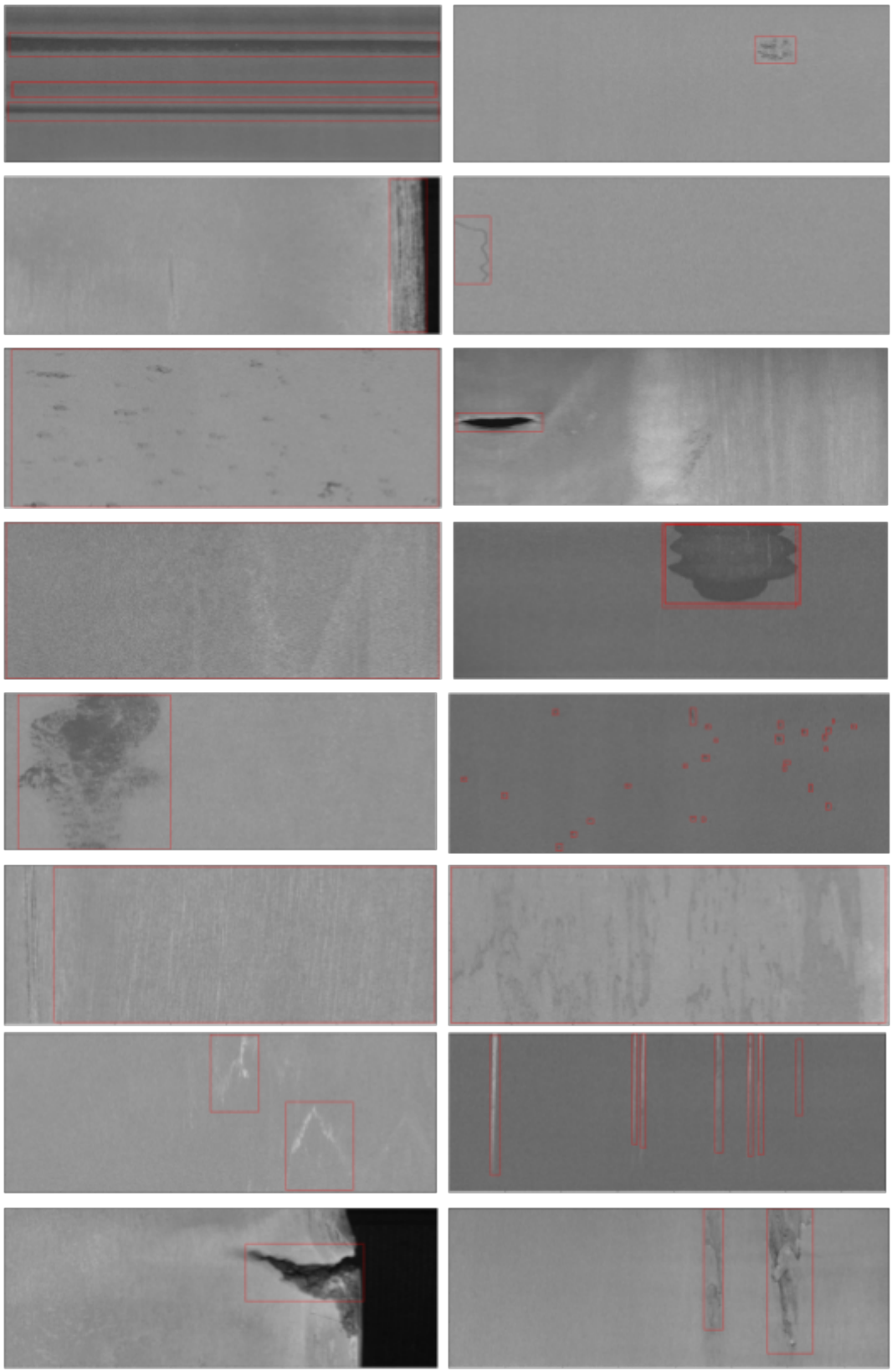

Figure 3. Defect types (first column: A–H, second column I–P). For illustration purposes only, bounding boxes are drawn around each defect. However, during training, the segmentation label consists of either a pixel-perfect or filled-in bounding-box mask as supervision.

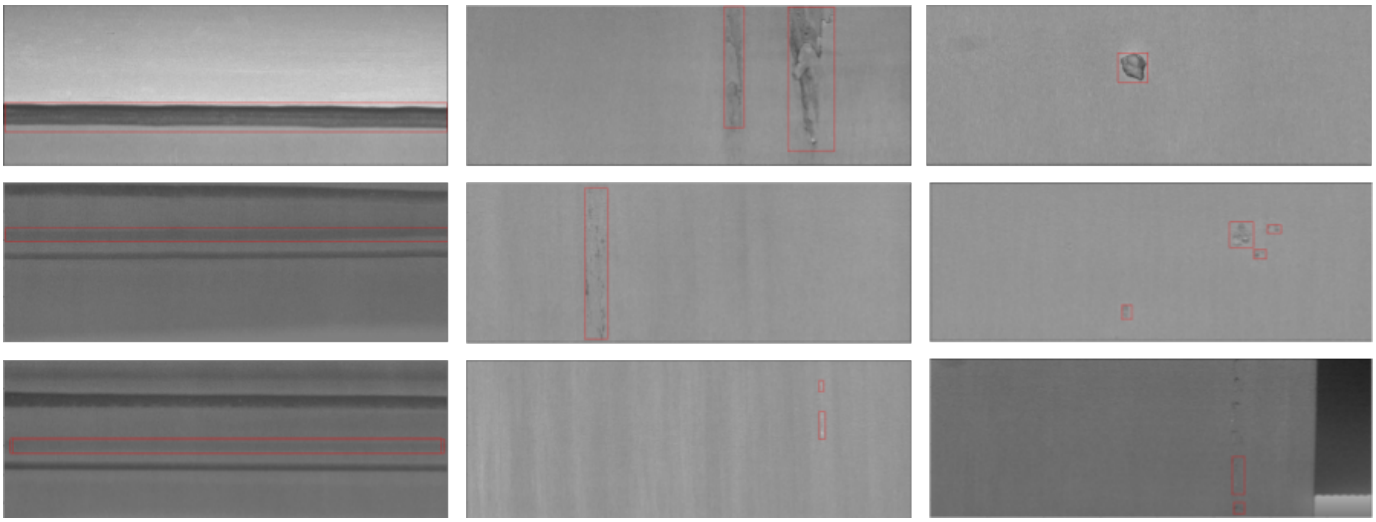

Figure 4. Defects with different severity levels. Each column represents a certain defect class with, for each row, a different severity. Top row shows severity level 3 defects, while the bottom shows level 1 defects. It is clear that level 3 defects are easier to detect than the lower-severity-level defects.

Additionally, all images have a set of parameters as extra supervision. These parameters include oven temperature, steel surface finish, and so on. Many works (e.g., [15–17]) have shown that, during the several processing steps needed to produce sheet steel material, the different process settings during the melting (e.g., oven temperature), hot rolling (e.g., rolling speed, furnace temperature, heating time, coiling temperature), and cold rolling steps (e.g., rolling speed, surface finish) have an influence on the defect occurrence rate.

Since we have these parameters available during inference, it is of interest to investigate whether there is any correlation between the presence of defects and the current parameter settings. By incorporating these parameters during training by using sensor-fusion-like approaches, the model might be able to learn these correlations and provide better predictions.

2.2. Semantic Segmentation Using U-Net

In our approach, we will use one of the popular semantic segmentation networks called U-Net [18] (Figure 5). The U-Net architecture follows the encoder–decoder structure. The goal of the encoder part is to capture the global context of the image, explaining 'what' is in the image, by gradually reducing the size of the image while increasing the depth of the image. This is done using only convolutional and max-pooling layers. The decoder's goal is to translate this information back to the original pixel location, by up-sampling the image using transposed convolution layers. These layers increase the size of the image while reducing the depth. Additionally, to improve the up-sampling of the image, at every step of the decoder, skip connections are used that concatenate the feature maps of the encoder to the output of the transposed convolution layers. These feature maps contain the original spatial information, which was lost during the encoder's compression and will help the decoder to construct a more precise segmentation result. It is easy to derive its name by looking at the symmetric U-shape of the network due to the skip connections. As only convolutional layers are used, this model is an end-to-end fully convolutional network and will enable us to train the model on any image resolution size.

To improve the model's performance, instead of using U-Net's standard encoder, we will use a ResNet model [19] as the encoder. Specifically, our experiments showed that a ResNet34 is a good compromise between both performance and execution time.

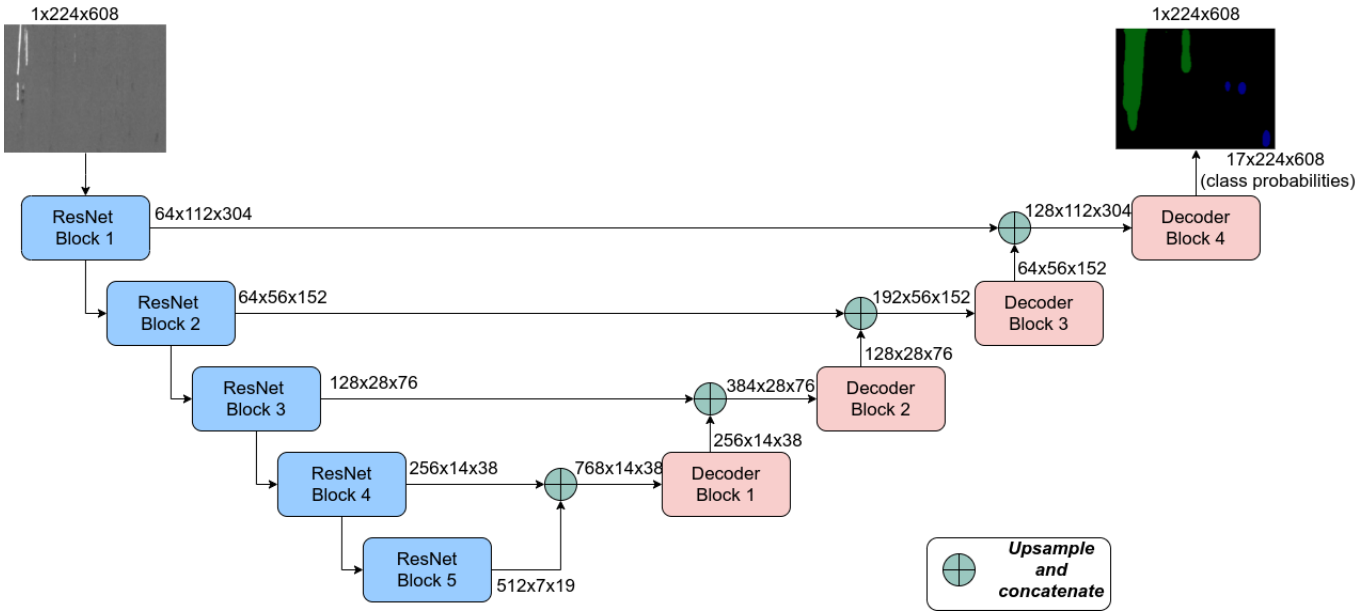

Figure 5. U-Net model [18] with ResNet [19] backbone.

Multi-Branch U-Net

Since we have two tasks at hand, i.e., the prediction of the defect class as well as the defect severity, we decided to expand the U-Net segmentation model to simultaneously output two segmentation maps. By inserting an extra decoder branch after the encoder, the model can output both the defect class segmentation map and the defect severity map. Each branch decoder has the same bottleneck as its input, meaning the encoder is shared between the two tasks. We have evaluated different shared decoder levels (instead of only sharing the encoder, some of the decoder layers can also be shared), but concluded that splitting the branches right after the encoder gives the best results. Figure 6 gives an overview of our approach.

Since we have to distinguish defects between 16 different classes, the first branch will output 17 channels (background included), which will be evaluated with an argmax function to indicate the predicted class. Similarly, the second branch will output 3 channels, with each channel representing the probability for each severity level. Again, these channels are evaluated with an argmax to indicate the predicted severity per pixel.

As described above, the first branch, or defect class branch, will output a probability for the background class. This means that this branch inherently learns the binary segmentation between back- and foreground. Since the second branch, or severity branch, learns to predict a severity instead of a defect class, this branch could also include a probability for the background class. This, however, would mean that both branches would learn the binary segmentation task, which is redundant. Therefore, we let only the defect class branch learn the semantic segmentation task including the background, while the severity branch only learns to predict a severity for each defect pixel. To achieve this, the severity branch will only receive gradients for defect pixels, i.e., only the pixels for which the branch has to learn a severity, by masking out the loss function with the pixel-perfect ground truth mask. The final severity mask can then be computed by masking out the output of the severity decoder with the foreground–background map of the defect type decoder.

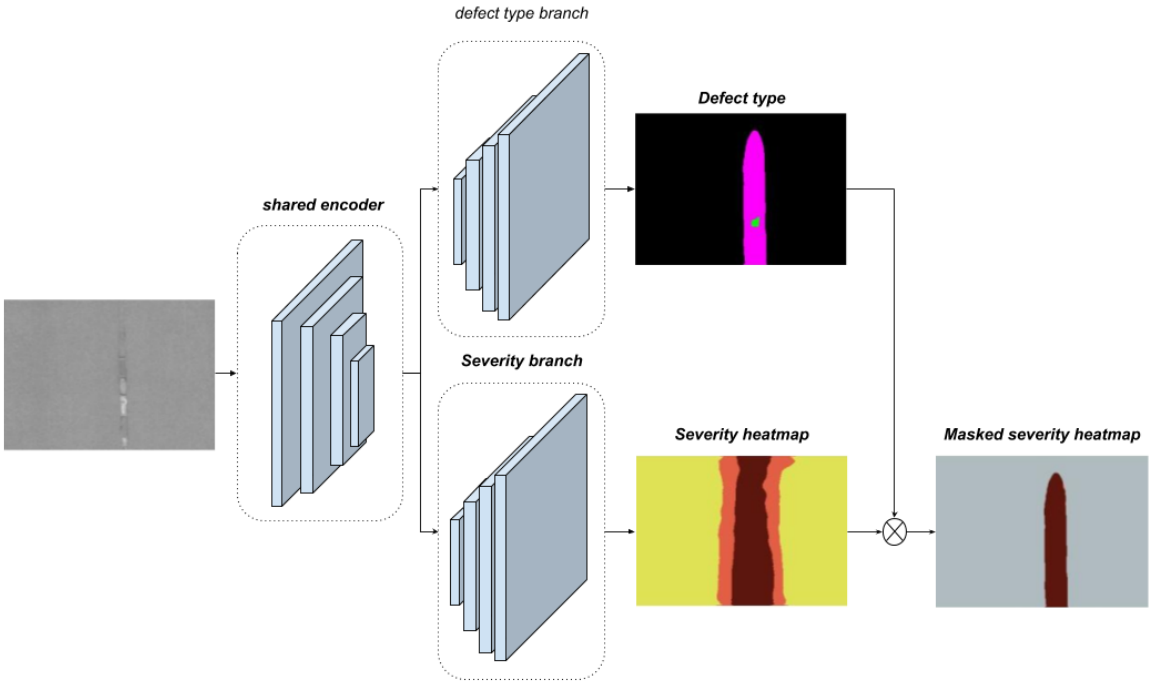

Figure 6. An overview of our multi-branch network. We expanded the U-Net architecture by adding an additional decoder branch. Each branch learns a specific segmentation task. The first branch (defect type branch) learns the segmentation of defect types. The second branch (severity branch) learns to generate a severity heatmap. The defect type mask generated by the first branch is then used to mask out the severity heatmap to generate the defect severity mask.

2.3. Loss Function

Two of the more popular loss functions for training a semantic segmentation network are the DICE (Sørensen–Dice) loss (Equation (1)) and the pixel-wise cross-entropy (CE) loss (Equation (2)).

$$L = 1 - \frac{2\sum_{pixels} y_{true} y_{pred}}{\sum_{pixels} y_{true}^2 + \sum_{pixels} y_{pred}^2} \tag{1}$$

$$L = -\sum_{classes} y_{true} log(y_{pred}) \tag{2}$$

After some initial experiments, we have established that the CE loss works slightly better than the DICE loss. Therefore, we will use the CE in all the subsequent sections. Because of the class imbalance in our dataset, we will weigh the CE loss using the class weights originally implemented by ENet [20], as seen in Equation (3). The main advantage of these weights is that they are bounded as the probability approaches zero, in contrast to the inverse class probability weighting. We adopt the same value as [20], namely 1.02, for the c hyper-parameter, meaning that the weights will be restricted to the interval of [1, 50].

$$w_{class} = \frac{1}{\ln(c + freq_{class})} \tag{3}$$

We will train each branch with its CE loss weighted with the branches' class weights. As the severity branch is only trained on foreground pixels, the frequencies for these severity classes are calculated with respect to the total amount of foreground pixels instead of the total amount of pixels.

3. Experiments and Results

In the following experiments, we will first train the two tasks (i.e., defect type and defect severity) separately. This way, we have a benchmark for the multi-task network when these are learned simultaneously using the multi-branch U-Net network (Figure 6).

In the last experiment, we will incorporate the process parameters during training and investigate the effect on the model performance. All experiments were executed using the PyTorch framework [21] with a 80/20 training/validation dataset split on an Nvidia V100.

3.1. Single Task Networks

For both tasks, we will use the (single-branch) U-Net model together with the cross-entropy loss (Equation (2)). The defect type model's output consists of 17 channels, while for the severity model, the output consists of 4 channels. Inherently, learning the background/foreground segmentation is exactly the same for both tasks. The difference is that the two models have to learn a different classification for the foreground pixels.

To compensate for the data imbalance present in both tasks as seen in Figure 2, we used the ENet class weighting as described in the previous section (Equation (3)). Each model was trained using the Adam optimizer [22] with a learning rate of 3×10^{-4}. To avoid over-fitting of the models, we experimented with different data augmentation techniques. Since our dataset consists of multiple resolutions, we already have to crop the images to provide the model with fixed resolutions during training. However, we have found that different crop sizes have significant effects on the segmentation output. In our experiments, we have found that, while a binary defect segmentation model can be trained with a small crop size, training a semantic segmentation network, which has to perform a dense prediction, requires a larger crop size. This can be explained by the fact that cropping a defect might make it similar to other defect types, resulting in confusion between different classes. However, while the majority of the images have a height of 280 pixels, some images have a height of 128. A crop size of height 128 showed a significant decrease in performance for both tasks. Therefore, the smaller images are padded using reflective padding to a height of 280 before taking a random crop of size 224×608.

Both models trained for 2500 epochs (3–4 min per epoch). An example of a training curve can be seen in Figure 7. We evaluated the model's performance on the validation set after each epoch. The model with the highest mIoU (mean Intersection over Union) validation score is chosen as the best model, since this metric is the most important for our application. The first row of Table 1 gives an overview of the performance on the validation set for both single-task models. Apart from the mIoU scores, the table also shows other metrics such as mean accuracy (mACC), frequency weighted IoU (fwIoU), and pixel accuracy (pACC) for comparison. Figures 8 and 9 show the pixel-wise confusion matrices for the different classes (validation data). For each task, we have plotted both the class matrices, i.e., the defect type and severity matrices and the background/foreground matrices. The first gives an insight into the confusion between the foreground pixels. The latter shows how many pixels were confused between background and foreground.

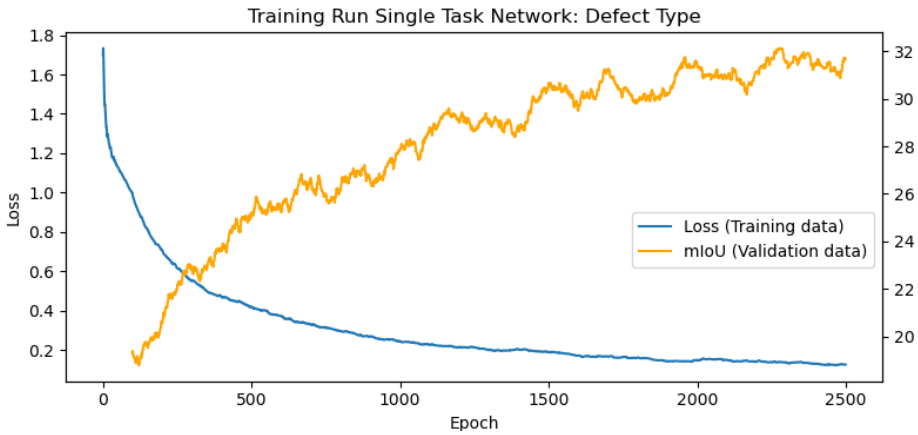

Figure 7. Training curve for the single-task defect type model. The chart shows both the loss curve on the training set and the mIoU score on the validation set for each epoch. The model with the highest mIoU validation score is chosen as the best model.

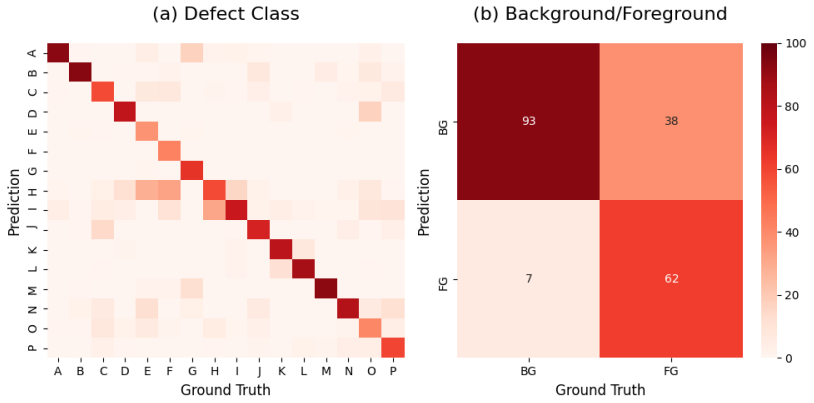

Figure 8. Confusion matrices for the defect type segmentation network (validation set). The first chart shows the confusion matrix for the foreground pixels. Most of the defect classes (A to P) show no or very low confusion with other classes. However, it is clear that for some classes, e.g., class E or P, the segmentation can still improve. This is mainly due to the fact that these classes are less represented in the dataset or that these are more difficult to detect. The second matrix shows the background/foreground confusion matrix.

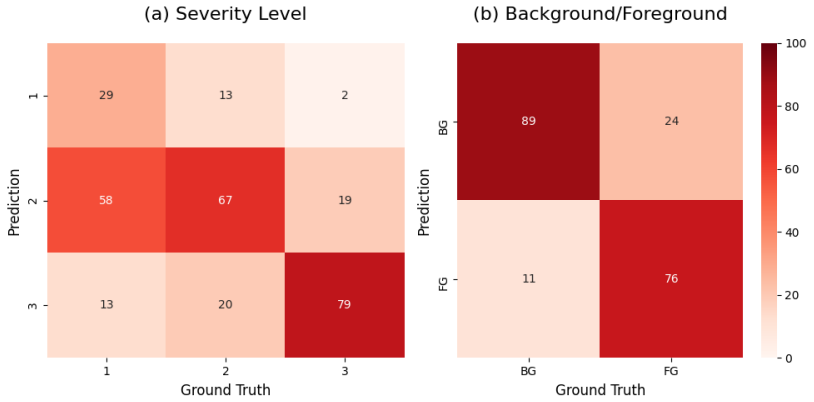

Figure 9. Confusion matrices for the severity segmentation network (validation set). The first chart shows the confusion matrix for the foreground pixels. Severity level 3 has the highest accuracy, with some confusion with level 2 defects. Severity 1 and 2 clearly show confusion with the other severity levels, especially severity 1, which are mostly segmented as level 2 defects. The second chart shows the background/foreground confusion matrix. Here, we can see that this network performed better on the back- and foreground task than the defect type network shown in Figure 8.

Table 1. This table summarizes the performance metrics of the single-task, multi-branch, and multi-branch model extended with process parameters on the validation set. The table shows the following metrics: mean IoU, mean accuracy, frequency weighted IoU, and pixel accuracy. For both tasks, the performance increased when combining the two tasks into one multi-task model (second row). Moreover, after extending the multi-branch model with process parameters, the performance of both tasks increased significantly (third row).

	Defect Type				**Severity**			
Model	**mIoU**	**mACC**	**fwIoU**	**pACC**	**mIoU**	**mACC**	**fwIoU**	**pACC**
Single-Task	31.9	49.8	83.2	88.6	37.8	55.3	81.2	85.6
Multi-Branch	33.4	51.7	84.1	89.2	40.8	59.5	82.3	86.1
Multi-Branch with Process Params	40.2	73.9	85.4	90.1	43.7	66.4	83.4	88.2

Some example output predictions from the validation set of the defect type and severity models can be found in Figures 10 and 11, respectively.

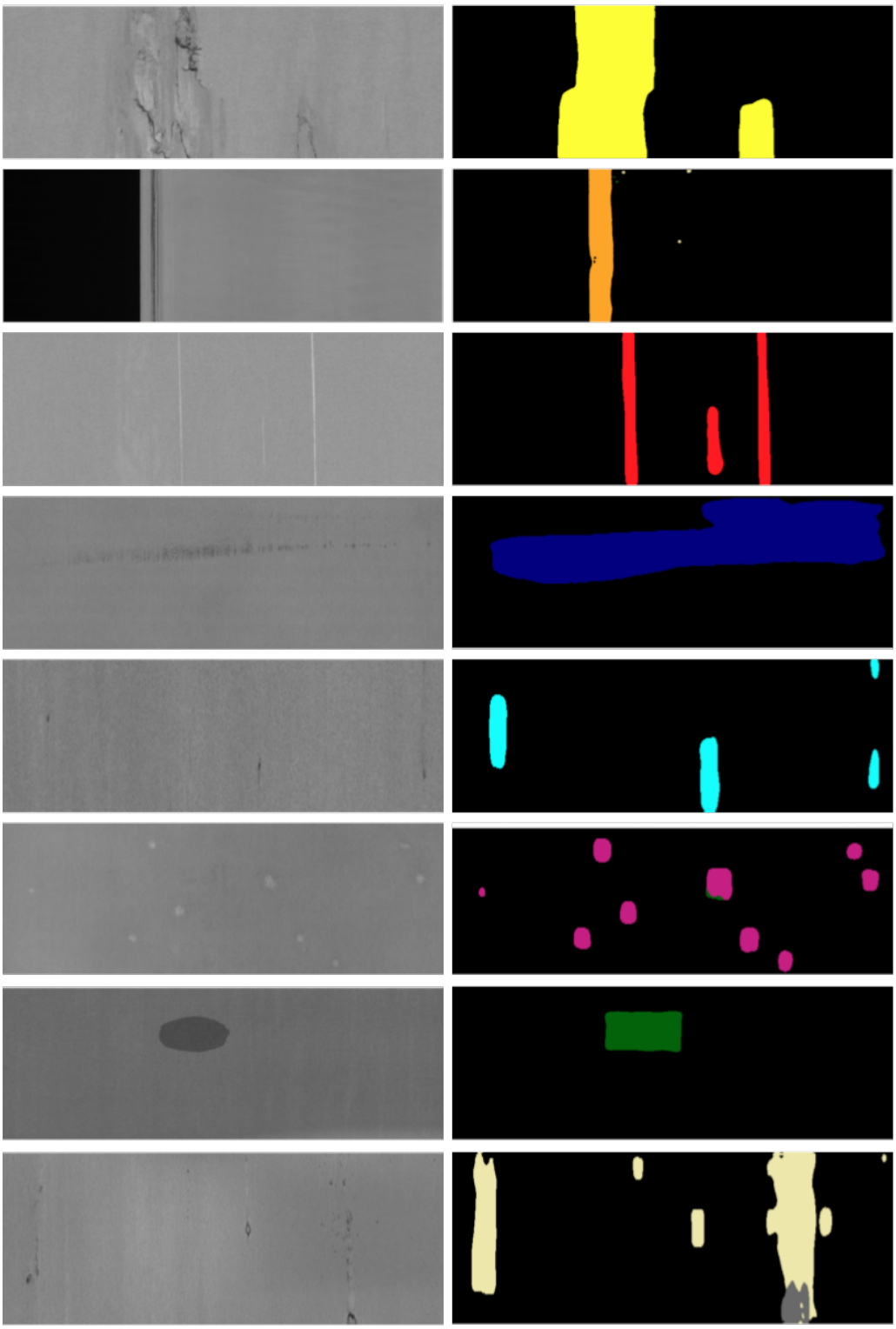

Figure 10. Example predictions of the defect type segmentation model (validation data). It is clearly visible that, although a large portion of the dataset includes rectangular target masks, the segmentation of the defects has a more confined border. However, sometimes, the segmentation can have a rectangular shape as in the example image in the second last row. Nevertheless, it is clear that the model is capable of segmenting most defects while also predicting the right class.

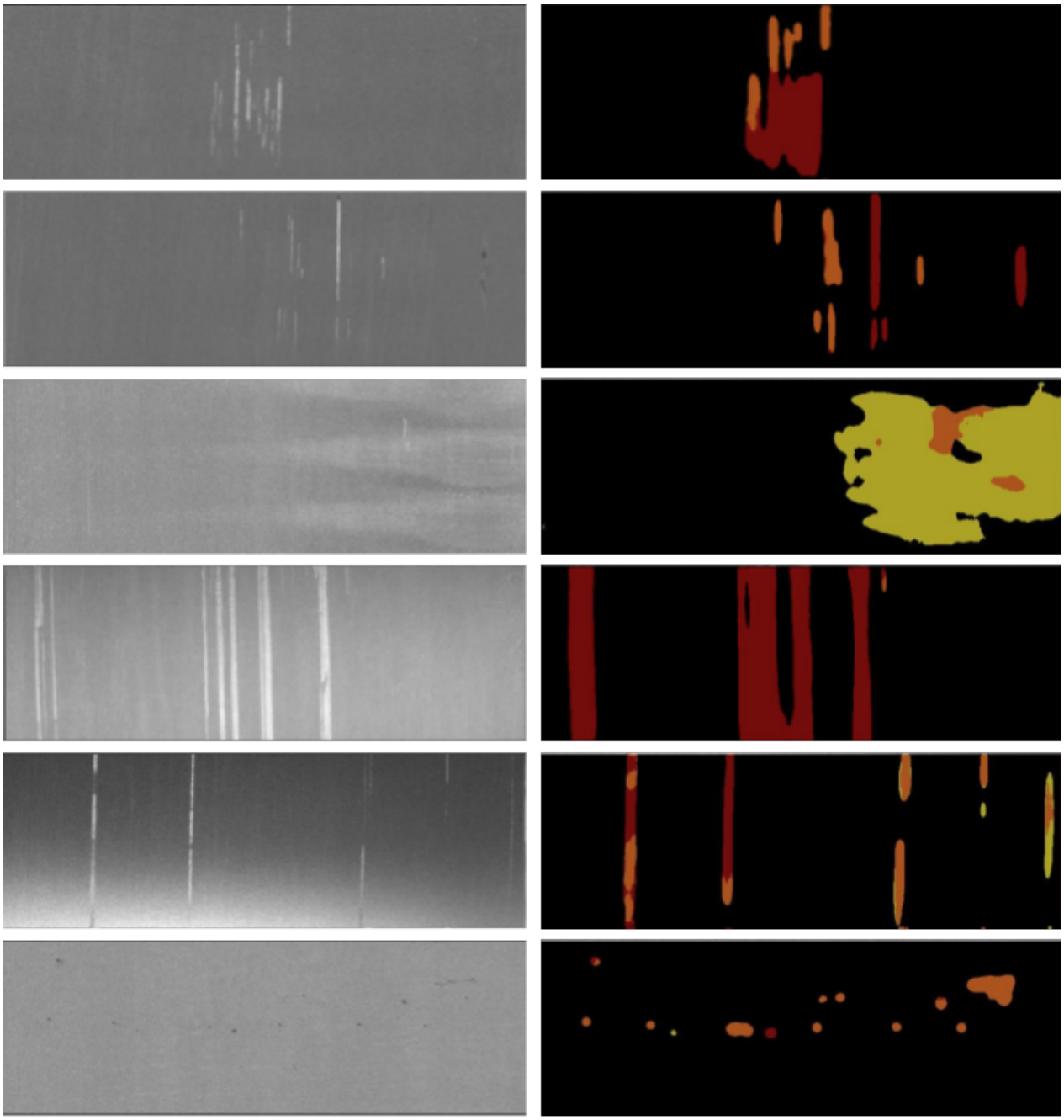

Figure 11. Example predictions of the severity segmentation model (validation data). The model succeeds in detecting most defects while accurately estimating the severity. However, the performance on level 1 severity defects still needs improvement due to erroneous labels, i.e., unlabeled or wrongly labeled defects.

3.2. Multi-Branch Network Combing Defect Type and Severity

In the previous section, we trained a model to perform the defect type segmentation as well as the defect severity segmentation tasks separately. This way, we had an idea of how the model performs when training on only one task. We already achieved good results using the single-task networks. However, we aim to perform a real-time quality inspection. Having two models running simultaneously is a huge computational burden and will slow down the pipeline. Therefore, by combining the two tasks into one model, the computational burden can be lowered drastically. Additionally, recent works on multi-branch learning, e.g., [23,24], have shown that if two tasks are learned together, each task might learn from the other, resulting in increased performance.

Multi-Branch U-Net

The U-Net model uses the encoder–decoder structure with skip connections between the encoder and decoder, transferring spatial information from the encoder to the decoder to improve the up-sampling. To achieve a multi-branch network, we will insert a second decoder into the network. This decoder receives the same encoder's output and skip connections. Each decoder branch will be responsible for one task and receives gradients from its own loss function. Since the cross-entropy loss has proven its efficiency in the single-task setups, we will adopt the same loss for each branch of the multi-branch model. A visual representation of the setup can be seen in Figure 6.

Having an extra decoder also introduces extra computation. This might be reduced by, instead of only sharing the encoder's layers between the tasks, also sharing some early layers of the decoder. However, for our application, initial experiments have shown that the best segmentation results are achieved by using completely separate decoders while having a shared encoder.

The training of the network is straightforward and is similar to the single-task setup. However, as described above, both the defect type and severity single-task models inherently learn the binary segmentation task. Therefore, it is only reasonable to learn this task in one branch only. In our experiments, the defect class branch will learn the binary segmentation task. With the background class discarded from the severity branch, this branch now outputs a probability for only three classes, i.e., the three severity levels. Since this is only required for foreground pixels, during training, we only provide gradients to the foreground pixels by masking the loss out with the target label. This means that, during inference, the output from the severity branch is only valid for foreground pixels, which are predicted by the defect class branch. Therefore, the severity output map has to be masked out by the background/foreground prediction of the defect type branch as seen in Figure 6.

Again, we train the model with the Adam optimizer and a learning rate of 3×10^{-4} for both branches. Both branch losses use the ENet class weighting. However, the severity weights are now relative to the foreground pixels only, instead of the total amount of pixels, since the loss is only applied to the foreground. The model trained for 2500 epochs (4–5 min per epoch) with a batch size of 16. We applied the same data augmentations as in the single-task setups, i.e., the random scaling between 0.8 and 1.2, and the random cropping of size 224×608 after conditional reflective padding if the height of the image is smaller than the crop size. The confusion matrices of the background/foreground, defect type and severity segmentation tasks of the trained multi-branch model can be found in Figure 12, Figure 13 and Figure 14 respectively. The second row of Table 1 gives an overview of the model's performance. It is clear that both the defect type and severity tasks have been improved by using the multi-branch setup, while reducing the computational burden of having two separate models.

Figure 12. Confusion matrices for the multi-branch model. (**a**) Confusion matrix of the multi-branch model for the background/foreground task. (**b**,**c**) Confusion matrix difference between the current confusion matrix and the matrices from the single task defect type and severity models displayed in Figures 8 and 9. It is clear that the multi-branch model performs better than the two single-task models on the background/foreground task.

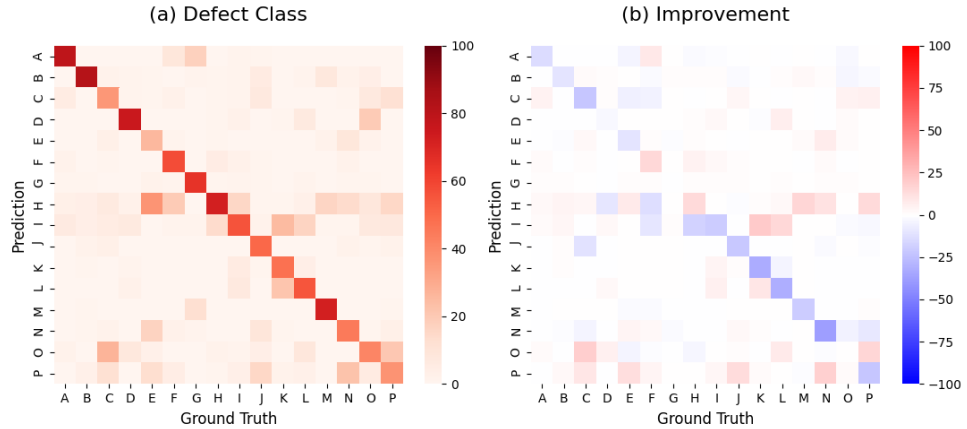

Figure 13. (**a**) Confusion matrix for the defect type task of the multi-branch model. (**b**) Accuracy improvement of the multi-branch model compared to the single-task model (defect type) from Figure 8. The model shows both a decrease and increase in accuracy for different classes. However, the overall performance of the task increased, as shown in Table 1.

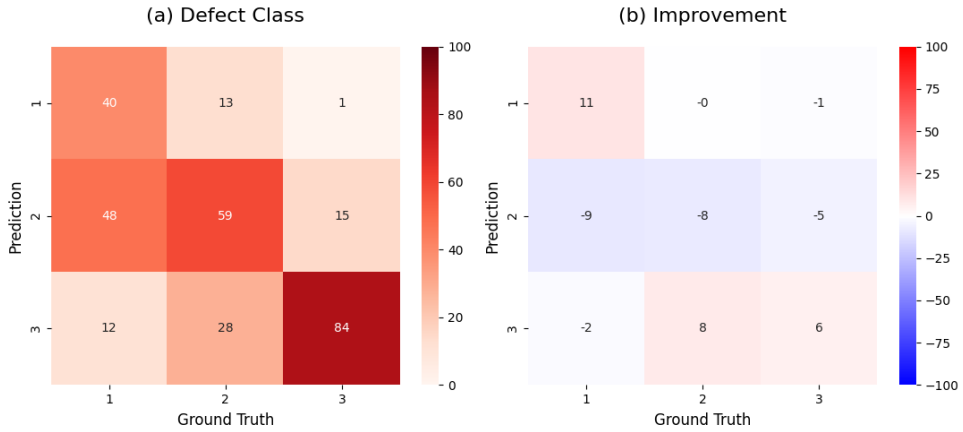

Figure 14. (**a**) Confusion matrix for the severity estimation task of the multi-branch model. (**b**) Accuracy improvement of the multi-branch model compared to the single-task model (severity) from Figure 9. The model shows both a decrease and increase in accuracy for different classes. However, the overall performance of the task increased, as shown in Table 1.

3.3. Using Process Parameters as Extra Supervision

Training the model with the annotated images supervises the model to determine which pixels contain defects and to predict the correct type and severity. However, due to different process parameters such as oven temperature or surface finish, the occurrence rate of specific defects and severity levels can differ. Therefore, we believe that there is a correlation between these parameters and the presence of certain defects which could help to improve the semantic segmentation. Without manually deriving these statistics, the model can learn the correlation between these parameters and the defect occurrence by incorporating them during training as extra supervision. Since these parameters are also available during inference, we do not have to treat them as target labels, as with the segmentation labels, but as inputs. Therefore, this can be seen as a form of sensor fusion, where we have both our camera images and the process parameters as inputs to our model.

There are many ways of incorporating the parameters into our model [25]. More specifically, we can insert them at the earlier layers, mid-layers, or the last layers of our model. Since the parameters are all integers, we have decided to insert them at the bottleneck of the network, i.e., the output of the encoder. Here, the activation tensor codes a very condensed, global representation of the image, in which it is logical to add other global data such as these process parameters. This is also referred to as 'mid-fusion', where the sensor's data are fused with the output of one of the middle layers of the model. This can be seen in Figure 15.

The parameters are a mix of discrete and analog values. However, since the analog values are integers in the range of 0–1000, we treat them the same as the discrete parameters and feed them directly into the model as is. As described above, the U-Net architecture is a fully convolutional network, meaning that the network can be trained on different input resolutions. However, the shape of the bottleneck, i.e., the output of the encoder, where we will feed in the process parameters, will change based on the input resolution. In the case of our U-Net model, the output of the encoder when training on 224×608 crops is a tensor of size $512 \times 7 \times 19$, meaning 512 channels of 7×19 size feature maps. To incorporate the parameters, we will expand (copy) our 11 parameters into a tensor of size $11 \times 7 \times 9$, each of the 11 feature maps containing the corresponding parameter value at each pixel location, and concatenate it to the bottleneck. The fused tensor will then be fed into both the defect type and severity branch as seen in Figure 15.

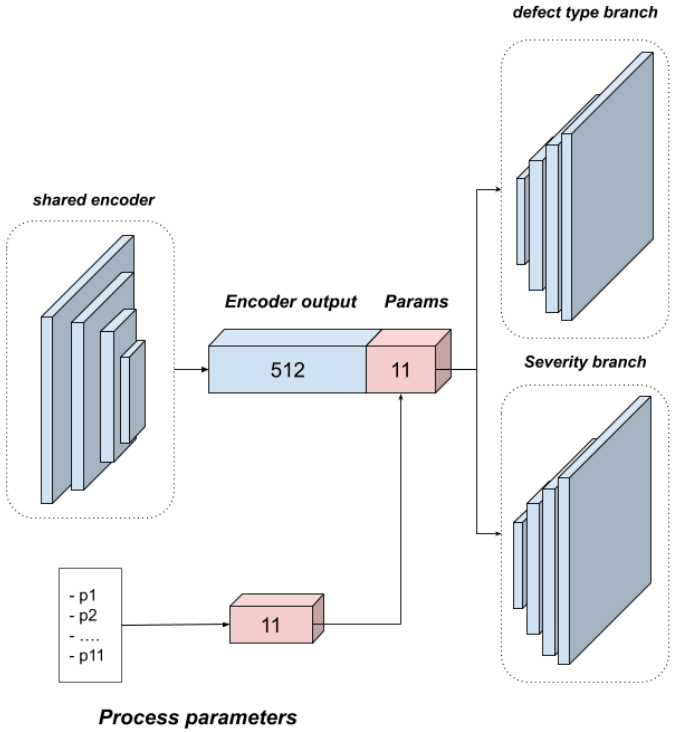

Figure 15. Incorporation of process parameters to improve model performance. After expanding the process parameter vector into a three-dimensional tensor, it is concatenated with the encoder output. The fused tensor then goes through both branches.

We trained the model for 2000 epochs (4–5 min per epoch). The same setup as in the previous experiment (using the multi-branch network) was used: Adam optimizer with a learning rate of 3×10^{-4}, cross-entropy loss with ENet class weighing for each branch, equally weighed. The same data augmentations were used: random scaling between 0.8 and 1.2 and a random crop of 224×608 with conditional reflective padding for images smaller than the crop size. Figures 16 and 17 compare the confusion matrices for the networks trained with and without the process parameters. The last row of Table 1 summarizes the performance metrics for the model.

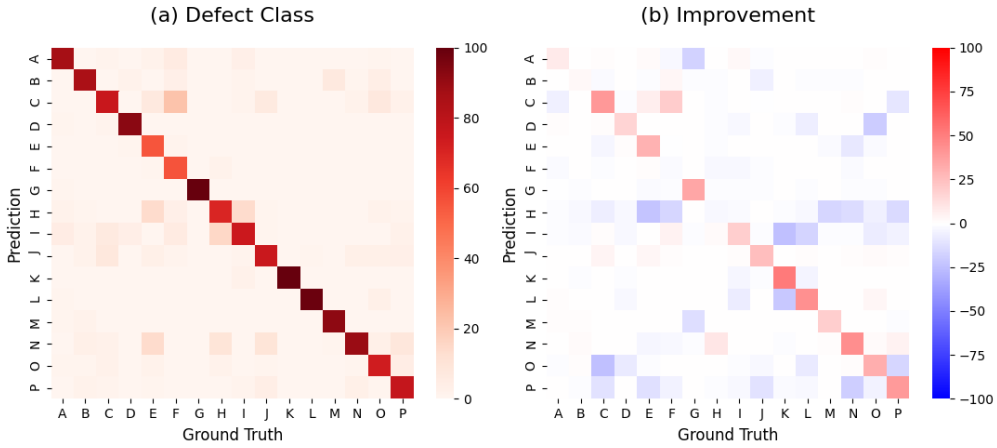

Figure 16. (**a**) Confusion matrix for the defect type segmentation task using the multi-branch model with process parameters. The first chart shows that the accuracy has improved over the model without the process parameters in Figure 13. (**b**) Improvement compared to the model without the process parameters. Most of the classes' accuracy has been significantly improved.

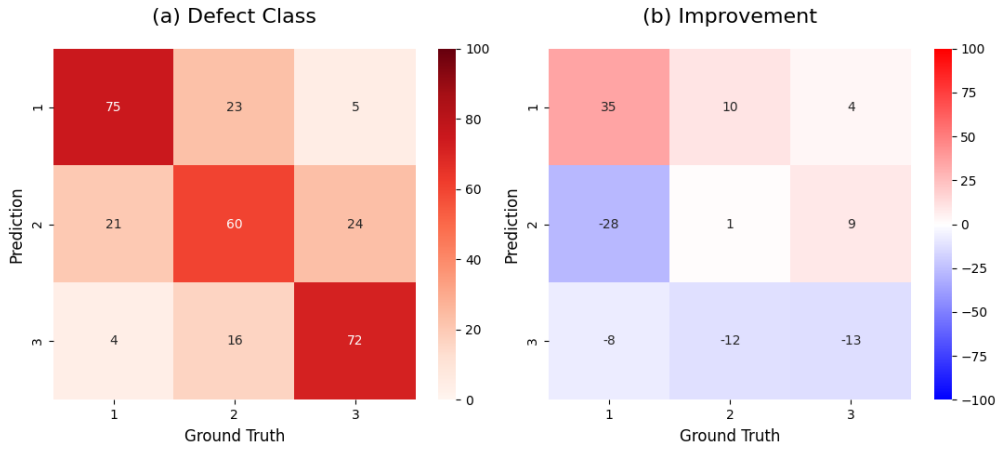

Figure 17. (**a**) Confusion matrix for the severity segmentation task using the multi-branch model with process parameters. The first chart shows that the accuracy has significantly improved over the model without the process parameters in Figure 14. (**b**) Improvement compared to the model without the process parameters. The chart shows a significant increase for severity level 1, while severity level 3 has a slight accuracy drop, showing an increase in confusion with level 2 severity defects.

Table 1 shows that the incorporation of the parameters drastically improved the performance on both tasks, with an increase of around 7% mIoU on the defect type task and a 3% increase on the severity task. If we look at the mean accuracy, the defect type task increased by 22% while the severity task increased by 7%.

4. Discussion

In this work, we have shown that the multi-branch network is able to detect the defects, classify the pixels, and estimate severity better than two single-task networks. However, the results show that the mean IoU is still beneath 50% and that a large portion of foreground pixels are segmented as background. Nevertheless, visual analysis of the results shows that nearly all of the defects are detected and accurately classified. The severity estimation is nearly as accurate as the defect type; however, there seems to be some confusion between the lower severity levels. This will be further investigated in future work, but we have to indicate that, due to the difficult distinction between background and level 1 severity defects, these are easily missed by both the labeling team and the trained model.

One other important point to make is the fact that our dataset consists of a large portion of rectangular target masks, resulting from the coarse bounding-box annotations. This adds confusion during the training phase, but also during the evaluation. Because the bounding boxes encompass also a large amount of defectless pixels, a portion of the false-negative pixels might be subjected to the wrong interpretation. Since the model is able to segment the defects with a finer mask than a rectangular shape, a portion of the false-negative pixels are correctly segmented as background, since these are the background pixels in the rectangular target masks. Moreover, due to missing labels in our dataset, some of the false-positive pixels are unlabeled defects, which the model is able to detect. Hence, we argue that the achieved mIoU metrics are actually an underestimation of the true accuracy of the model.

However, we have already stated that we will investigate this further and will improve our in-house dataset using model-assisted labeling, building further on this work. This will enable the semi-manual conversion of all bounding-box labels to full pixel-perfect segmentation maps.

5. Conclusions

In this work, we have successfully implemented and trained a multi-task U-Net segmentation network to detect steel sheet surface defects. The model is able to detect and recognize up to 16 different defect classes and also estimates a defect severity level between 1 and 3. We showed that training both tasks simultaneously yields substantially better accuracy results as compared to separately trained distinct models. We also demonstrated that incorporating extra context information in the form of steel production process parameters is also very advantageous. The resulting model achieves a mean IoU score of 40% for the defect class task and around 43% mean IoU for the defect severity task, despite the partially coarse training and test annotations.

Author Contributions: Methodology, R.N. and T.G.; Software, R.N.; Supervision, T.G.; Writing—original draft, R.N.; Writing—review and editing, T.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research recieved funding from the VLAIO project HBC.2017.1002 and the Flemish Government (AI Research Program).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Acknowledgments: We would like to thank Robbert Camps for collecting and managing the labeling process of the training data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Luo, Q.; Fang, X.; Liu, L.; Yang, C.; Sun, Y. Automated Visual Defect Detection for Flat Steel Surface: A Survey. *IEEE Trans. Instrum. Meas.* **2020**, *69*, 626–644. [CrossRef]
2. Song, K.; Yan, Y. A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. *Appl. Surf. Sci.* **2013**, *285*, 858–864. [CrossRef]
3. Severstal. *Severstal: Steel Defect Detection*; Severstal: Cherepovets, Russia, 2019.
4. O'Shea, K.; Nash, R. An Introduction to Convolutional Neural Networks. *arXiv* **2015**, arXiv:1511.08458.
5. Tao, A.; Sapra, K.; Catanzaro, B. Hierarchical Multi-Scale Attention for Semantic Segmentation. *arXiv* **2020**, arXiv:2005.10821.
6. Vakanski, A.; Xian, M.; Freer, P.E. Attention-Enriched Deep Learning Model for Breast Tumor Segmentation in Ultrasound Images. *Ultrasound Med. Biol.* **2020**, *46*, 2819–2833. [CrossRef] [PubMed]
7. Azimi, S.M.; Henry, C.; Sommer, L.; Schumann, A.; Vig, E. SkyScapes Fine-Grained Semantic Understanding of Aerial Scenes. In Proceedings of the 2019 IEEE/CVF International Conference on Computer Vision (ICCV), Seoul, Korea, 27 October–2 November 2019; pp. 7392–7402. [CrossRef]
8. Kollmann, H.T.; Abueidda, D.W.; Koric, S.; Guleryuz, E.; Sobh, N.A. Deep learning for topology optimization of 2D metamaterials. *Mater. Des.* **2020**, *196*, 109098. [CrossRef]
9. Konovalenko, I.; Maruschak, P.; Brezinová, J.; Viňáš, J.; Brezina, J. Steel Surface Defect Classification Using Deep Residual Neural Network. *Metals* **2020**, *10*, 846. [CrossRef]
10. Damacharla, P.; Achuth, R.M.V.; Ringenberg, J.; Javaid, A.Y. TLU-Net: A Deep Learning Approach for Automatic Steel Surface Defect Detection. *arXiv* **2021**, arXiv:2101.06915.
11. Fu, J.; Zhu, X.; Li, Y. Recognition Of Surface Defects On Steel Sheet Using Transfer Learning. *arXiv* **2019**, arXiv:1909.03258.
12. Wang, S.; Xia, X.; Ye, L.; Yang, B. Automatic Detection and Classification of Steel Surface Defect Using Deep Convolutional Neural Networks. *Metals* **2021**, *11*, 388. [CrossRef]
13. Minaee, S.; Boykov, Y.; Porikli, F.; Plaza, A.; Kehtarnavaz, N.; Terzopoulos, D. Image Segmentation Using Deep Learning: A Survey. *arXiv* **2020**, arXiv:2001.05566.
14. Takos, G. A Survey on Deep Learning Methods for Semantic Image Segmentation in Real-Time. *arXiv* **2020**, arXiv:2009.12942.
15. Naumenko, V.; Muntin, A.; Danilenko, A.; Baranova, O. Study of the Surface Defect Nature of Hot-Rolled Products in the Edge Zone. *Steel Transl.* **2020**, *50*, 46–52. [CrossRef]
16. Yu, H.; Tieu, K.; Lu, C.; Deng, G.y.; Liu, X.h. Occurrence of surface defects on strips during hot rolling process by FEM. *Int. J. Adv. Manuf. Technol.* **2012**, *67*. [CrossRef]
17. Dhua, S. Metallurgical Analyses of Surface Defects in Cold-Rolled Steel Sheets. *J. Fail. Anal. Prev.* **2019**, *19*, 1–11. [CrossRef]

18. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *Medical Image Computing and Computer-Assisted Intervention—MICCAI 2015*; Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F., Eds.; Springer: Cham, Switzerland, 2015; pp. 234–241.
19. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep Residual Learning for Image Recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016.
20. Paszke, A.; Chaurasia, A.; Kim, S.; Culurciello, E. ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation. *arXiv* **2016**, arXiv:1606.02147.
21. Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*; Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R., Eds.; Curran Associates, Inc.: Red Hook, NY, USA, 2019; pp. 8024–8035.
22. Kingma, D.P.; Ba, J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, 7–9 May 2015.
23. Vandenhende, S.; Georgoulis, S.; Van Gansbeke, W.; Proesmans, M.; Dai, D.; Van Gool, L. Multi-Task Learning for Dense Prediction Tasks: A Survey. *IEEE Trans. Pattern Anal. Mach. Intell.* **2021**. [CrossRef] [PubMed]
24. Crawshaw, M. Multi-Task Learning with Deep Neural Networks: A Survey. *arXiv* **2020**, arXiv:2009.09796.
25. Zhang, Y.; Sidibé, D.; Morel, O.; Mériaudeau, F. Deep multimodal fusion for semantic image segmentation: A survey. *Image Vis. Comput.* **2021**, *105*, 104042. [CrossRef]