applied sciences

MDPI

Article

Defect Detection on Rolling Element Surface Scans Using Neural Image Segmentation

Nico Prappacher [1], Markus Bullmann [2], Gunther Bohn [1,*], Frank Deinzer [2] and Andreas Linke [1]

[1] Faculty of Electrical Engineering, University of Applied Sciences, Würzburg-Schweinfurt, 97421 Schweinfurt, Germany; nico.prappacher@student.fhws.de (N.P.); andreas.linke@fhws.de (A.L.)
[2] Faculty of Computer Science and Business Information Systems, University of Applied Sciences, Würzburg-Schweinfurt, 97074 Würzburg, Germany; markus.bullmann@fhws.de (M.B.); frank.deinzer@fhws.de (F.D.)
* Correspondence: gunther.bohn@fhws.de

Received: 9 April 2020; Accepted: 6 May 2020; Published: 9 May 2020

Abstract: The surface inspection of steel parts like rolling elements for roller bearings is an essential component of the quality assurance process in their production. Existing inspection systems require high maintenance cost and allow little flexibility. In this paper, we propose the use of a rapidly retrainable convolutional neural network. Our approach reduces the development and maintenance cost compared to a manually programmed classification system for steel surface defect detection. One of the main disadvantages of neural network approaches is their high demand for labeled training data. To bypass this, we propose the use of simulated defects. In the production of rolling elements, real defects are a rarity. Collecting a balanced dataset thus costs a lot of time and resources. Simulating defects reduces the time required for data collection. It also allows us to automatically label the dataset. This further eases the data collection process compared to existing approaches. Combined, this allows us to train our system faster and cheaper than existing systems. We will show that our system can be retrained in a matter of minutes, minimizing production downtime, while still allowing high accuracy in defect detection.

Keywords: automated optical inspection; image segmentation; convolutional neural networks; surface defects; quality control

1. Introduction

Rolling elements present in, e.g., the automotive or aeronautics industry are required to satisfy high quality standards. A single failure can lead to enormous financial and fatal damages. To assure these quality standards, manufacturers employ surface inspection systems. These scan the roller bearings for defects. This quality assurance step sits between the manufacturing of the element and the bearing's final assembly.

Surface inspection is either done through human experts or with automated machine vision systems. The former exhibit human errors, especially after extended hours of working on the tedious inspection task. Most automated approaches implement a set of manually parameterized operations. This requires extensive domain knowledge. The resulting classification system is highly specialized for the task [1]. Even minor changes in the manufacturing process can make the system unusable. With changes in the production process, more expert knowledge is required to adapt the classification system to the new conditions. This leads to production downtime and lower profits.

Classical machine learning systems solve this problem only to a certain extent. Such systems combine feature extraction algorithms built on expert knowledge with learned classification

algorithms [2]. The latter can automatically find optimal classifiers for a given set of training data. No domain specific expert knowledge is needed for the classification algorithm. While this automates parts of the whole classification system, expert knowledge is still required for finding descriptive features. The feature extraction is not automatically learned from training data. Therefore, it can't be automatically adapted when the need arises.

Modern neural network approaches can fully automate the expert's task of injecting domain specific knowledge into the system [3]. Neural networks are self-learning systems and can automatically identify useful features in an example set of data. As will be discussed in Section 2 of this work, literature already proposes the use of convolutional neural networks (CNN) to reduce the need for expert knowledge. However, most of the proposed systems require extensive amounts of labeled data with a balance of defect and non-defect parts to train their CNNs [4]. Since the production of rolling elements is already a highly optimized process, defects are a rarity. This makes collecting the necessary amount of defect samples a time-consuming process. To avoid this, we train our network on simulated defects.

The rest of this paper will be structured as follows: Section 2 analyzes current proposals in the literature for the given problem. Section 3 explains the necessary background on CNNs required for the rest of this paper. In Section 4 we elaborate on the process of collecting the training data and simulating the defects. Section 5 describes the classification system, including the architecture of our neural network and its training. In the last section, we evaluate the proposed system on previously unseen training data.

2. Related Works

Detecting defects on steel surfaces is a known problem in literature. After the success of AlexNet in the ImageNet challenge [3], many proposed systems have relied on the use of CNNs for defect detection on such surfaces. While the literature refers to a variety of steel surfaces, their methods were found to be applicable to our problem area of rolling elements. Zhou et al. [5] use a CNN with a softmax classification to classify image segments. The segments have a fixed size of 200 × 200 px and are classified into one of eight classes. The network is trained on a labeled set of 300 examples per class before augmentation. To collect such a large dataset requires much manual labor and results in costly production downtime. Tao et al. [6] use a cascaded auto-encoder architecture for semantic segmentation. The defect regions found by this part of the classification system are then fed into a traditional CNN with a softmax layer at its end. This CNN classifies the defect into one of three classes. The authors note the systems need for labeled data as one of its main drawbacks. Ghoria et al. [2] propose a system that to our knowledge allows the finest classification, with 24 classes. The system slices the input image into non-overlapping windows. In our experiments, this can lead to misclassifications when small defects are cut in half by the windowing.

While all the above papers classify the defect into several given categories, our system only detects the presence or absence of defects. In a multi-class setting, information on the concrete type of defect gives information on possible problems in manufacturing. However, this classification step requires additional training time, labeled data and computational resources. By forgoing a multi-class approach, our system outperforms most existing systems in inference and training time. The system is not required to learn to associate features with specific defect classes. This simplifies the classification problem. If a classification of the found defects into categories is needed later, an additional classification model can be appended to the proposed system. In the resulting two-stage model cascade, this second model would then classify the defects into multiple classes. These classifications would then again allow conclusions to be drawn on problems in the production process. Luo et al. [7] discuss the advantages of those two-stage systems and compare different approaches.

One of the downsides of supervised classification systems is their need for labeled training data. Outlier detection approaches circumvent this requirement by training on unlabeled data. Staar et al. [8] apply such an outlier detection approach to the problem of detecting defects on steel surfaces.

The authors of this paper use a CNN to extract features from input images. Outliers are then detected by comparing distances between samples in feature space in a nearest neighbor fashion. While the need for labeled data is a downside of supervised learning approaches, they are shown to perform better than outlier detection approaches in defect detection problems [9,10].

Our proposed approach uses simulated defects, avoiding the need for manual labeled data while preserving the advantages of a supervised approach.

Based on the aforementioned reasons, a CNN was trained for binary image classification. With simulated defects, data requirements in comparison to existing systems were reduced.

3. Convolutional Neural Networks

Traditional multi-layer perceptrons (MLP) are a kind of artificial neural network that is exclusively made up of fully connected layers. When high resolution images are used as input data, these fully connected layers result in a high number of parameters, called weights. This is due to all input pixels being connected with all neurons in the following layer. This makes them prone to overfitting and computationally intensive to train. To lower the parameter count of MPLs, preprocessing is applied to the input data. The preprocessing reduces the input data dimensionality. These preprocessing steps must be carefully chosen to not discard information from the data that is important to the classification task. As such, it relies heavily on expert knowledge to minimize dimensionality while retaining as much information as possible. This has similar downsides as the classical optical inspection and machine learning approaches, as outlined in the introduction.

CNNs are another kind of ANN. They reduce the parameter count of MLPs by only connecting neurons of ensuing layers locally. Furthermore, the weights are shared across the dimensions of the input data, usually images. The weight sharing allows translation invariant detection of, e.g., defects in the image. In practice, these features are implemented using a convolution operation. Learned filter kernels of predetermined size are slid across the image with a predetermined step-width, called stride. The network needs only to learn the parameters of the filter kernels. This reduces their parameter count compared to an MLP operating on equal sized input data. This allows CNNs to directly operate on high resolution input images, without preprocessing. At every position of the input image, the kernels are convolved with the underlying pixels. The resulting image is called an activation map. The activations of a whole layer are then stored together in a tensor and fed further downstream in the neural network. As is the case in traditional ANNs, non-linearities are applied to the resulting activations. This allows the network to approximate non-linear functions. Deep CNNs stack multiple layers of convolution operations together. Every subsequent layer extracts more and more high-level features from the input image.

Most classification systems that rely on CNNs directly classify the input image, based on these high-level features. More recently, however, image segmentation approaches have come to the fore. Here, the pixels of an input image are individually classified into a set of classes. For defect detection on images, this not only allows the classification of the input, but also the localization of the defects. The importance of defect localization is discussed in [10].

A decoder part is added to the network to achieve this segmentation. The decoder consists of further convolution layers. It maps the CNN's high-level features, also called latent representation, back into an image of the input image's size.

4. Data Collection

For data acquisition, the cylindrical rolling elements of length 13 mm and diameter 10 mm are rotated on two co-rotating roles. A light emitting diode (LED) array emits pulses of light in synchronization with an E2V UNIIQA+ Essential line scan camera's trigger. The camera is connected to a workstation computer via a high bandwidth camera-link cable and a Matrox Solius eCL/XCL-B framegrabber. The camera is capturing a 1395 px line. To reduce motion blur, the exposure time of the

camera is set to 50 µs. Accordingly, the LED light is of high luminance to enable the camera to capture a bright image. This renders the influence of ambient light insignificant by comparison.

A rotary encoder is attached to the co-rotating roles on which the steel parts lie. It continuously triggers the camera and lighting. During the element's 360° rotation, the camera captures 3335 lines of pixels, resulting in a 1395 px by 3335 px image. The resolution in both dimensions is ~107 ppcm, creating an undistorted scan. The whole scanning process takes about half a second. This sets the upper time limit on the processing time per image classification. Exceeding this limit results in additional latency in the inspection pipeline.

The steel surface of an ideal rolling element is of specular nature. As is known in physics, the angle of incidence of light is equal to the angle of reflection. Based on this, the camera is aligned to capture the light emitted by the LEDs and reflected by the rolling element, as visualized in Figure 1. Surface defects in the form of unevenness result in the light being reflected away from the camera. Similarly, corrosion and other diffuse surface defects also result in less light reaching the camera. The scanned images thus exhibit dark spots where the steel surface is less specular or not perfectly even. The camera captures a Red-Green-Blue (RGB) image with 8-bit depth per color channel. Since no relevant color information is present, the images were stored as grayscale images. This lowers their storage requirement and their loading time during training.

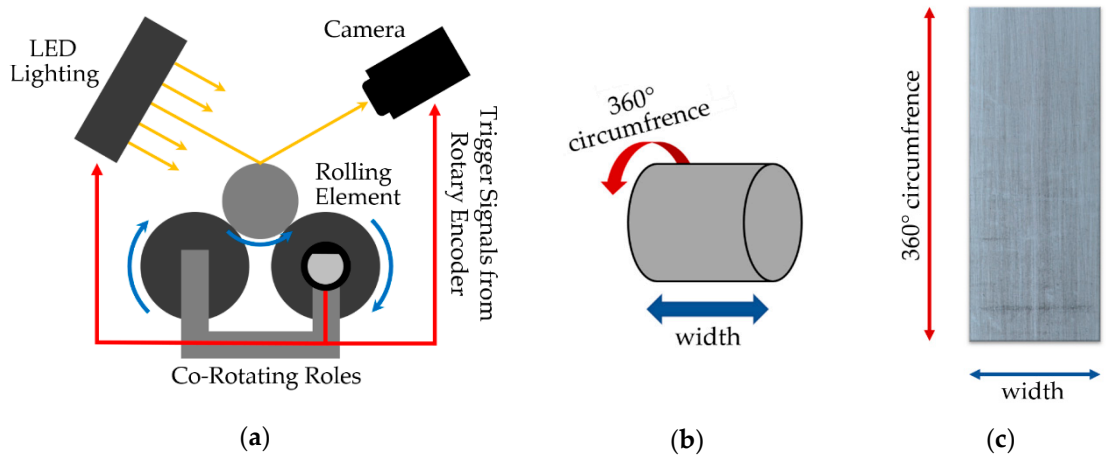


Figure 1. (**a**) A schematic side view of the hardware setup used for scanning the rolling elements. (**b**) and (**c**) indicate how the resulting surface scan images of size 1395 px by 3335 px correspond to the rolling element parts they originate from.

The network is trained using only defect free parts as source material. However, for evaluation and validation purposes, multiple elements with a variety of defects were scanned. Some of these defects are visualized in Figure 2.

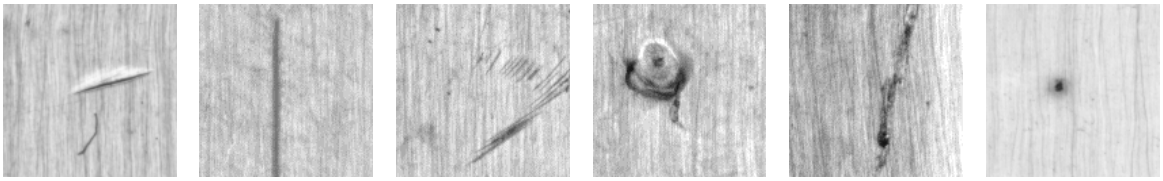


Figure 2. Various defects on the steel surface scans.

For the training of the neural network, 20 high-resolution scans of defect-free elements were captured. These were sliced into 4.000 windows of size 128 × 128 px. This allows the network to be trained on large batches of small windows instead of small batches of whole scans. The latter approach suffers from noisy gradients and thus longer convergence times.

4.1. Defect Simulation

In case of alterations in the production process, the system needs to be quickly retrainable to minimize production downtime. In order to make this possible, production-like conditions have been set for training. This means, that the neural network is given only few data samples. Most neural networks are trained on a balanced dataset to avoid bias in the trained system. However, this requires massive amounts of defective rolling elements. Since the production of rolling elements is already a highly optimized process, defective parts are a rarity. To avoid the need for real defective parts, we propose the use of simulated defects. These defects are digitally added to the scans of defect-free rolling elements. Furthermore, limiting the training dataset to be sourced from defect-free parts also greatly reduces the cost of manual data labeling. Most traditional segmentation approaches require manually created labels for their training. Our approach, however, only requires ensuring that the data samples include no defects. Scans of defective parts captured during the retraining's data collection process need to be taken out of the training set. These can still be used for evaluation purposes.

Defects on surface scans are simulated by masking the windows with ellipses. These are applied to the image at a randomized position, orientation and with a randomized brightness. Low amplitude noise is then added to texture the ellipse. The ellipses have a minimum size that escapes the tolerance limit set by quality control. Figure 3 showcases a few examples of such defect simulations.

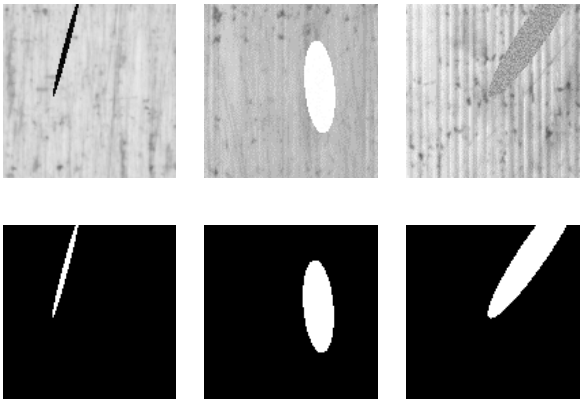


Figure 3. (**top**) Simulated defects on otherwise defect-free images. (**bottom**) Corresponding segmentation images used as labels during neural network training.

Simultaneously, a segmentation mask is created for every sample. The mask features the same ellipse with white color on a black background. These segmentation masks are used as targets during the training of the neural network. Additionally, the original defect-free windows are also added to the training set. The segmentation masks corresponding to these windows are all black. For every defect-free window in the original dataset, one simulated defect with exactly one ellipse is created. This doubles the size of the training dataset to 8.000 samples.

4.2. Augmentation and Preprocessing

The size of the dataset is further multiplied through augmentation. Augmentation generates new data from existing data by applying label-preserving transformations to the image. While the convolution operations in the network are invariant to translations, they are not invariant to flipping or rotations. These additional data samples acts regularizing during training to achieve lower generalization error.

For the presented system, every existing window was quadrupled. Its new copies were rotated by 180° and/or horizontally flipped. Further augmentation by 90° rotations was waived, since this would alter the fine, vertical structures in the background textures apparent in Figure 2. This results in a final dataset size of 32.000 samples.

Before the data samples are passed to the neural network for training, further augmentation is applied online. These augmentations include slight cropping, rotations and brightness variations. Importantly, every transformation applied to the input image must also be applied to the targeted segmentation mask. With every epoch of training the neural network, these additional augmentations differ. The network thus never sees the exact same image twice.

5. Classification

5.1. Neural Network Architecture

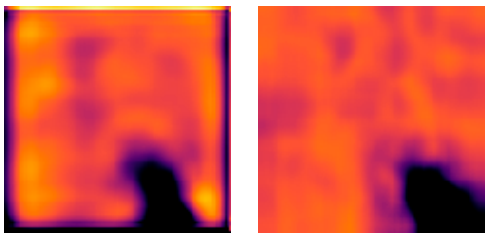
As stated in Section 3, we propose the use of a neural segmentation network to localize defects on the input images. Image segmentation problems are solved in the literature with convolutional encoder-decoder architectures [11]. These remap the features extracted by a traditional CNN trunk, back into an image of the input size. The use of an existing pre-trained network was foregone. Architectures like SegNet [11] are usually trained on segmentation tasks for scene understanding problems. The structures found on the steel surface scans of rolling elements bear little resemblance to the structures found in these scenes. This results in a suboptimal feature extraction, when such pre-trained networks are used. Additionally, existing architectures are usually quite large with 26 and 23 layers of convolutions for SegNet and U-Net [12], respectively. This depth is required for their complex task of segmenting images into dozens of classes or finding segmentations in complex biomedical images. Instead, a custom designed architecture as listed in Table 1 was used. It is of smaller depth, as the given defect detection problem is a more specific and therefore less complex problem. Furthermore, a smaller architecture allows for shorter training time and results in faster inference time during deployment.

Table 1. Architecture of the utilized neural segmentation network.

	Layers	Kernel Size	Filters	Stride
Encoder	Conv1	7×7	16	/2
	Conv2	7×7	32	/2
	Conv3	7×7	64	/2
	Resize1	-	-	$\times 2$
Decoder	Conv4	7×7	32	1
	Resize2	-	-	$\times 2$
	Conv5	7×7	16	1
	Resize3	-	-	$\times 2$
	Conv6	7×7	8	1
	Conv7	1×1	1	1

The encoder CNN consists of three stacked layers of convolutions. Each layer uses filter kernels of size 7×7 pixels. After all convolution layers, Leaky-ReLU activations are used as non-linear activation functions. The first layer convolves the image with 16 filters. Every subsequent layer in the encoder doubles the filter count. A stride of two in all three convolution layers reduces the size of the activation images in both dimensions by a total factor of eight. A reduction of the activation map sizes through the more common max pooling operation was forgone, based on the conclusions drawn in [13]. The bottleneck resulting from the size reductions reduces the representational capability of the network and acts regularizing. Since every pixel in a segmentation problem is equally important, the convolution operations also act on edge pixels of the image. To allow this, reflection padding extends the size of the image before convolutions are applied. Most CNN architectures use zero-padding to fill the space around the image. In our experiments, this led to artefacts on the edges of activation maps. The padded zeros can be interpreted as defect image regions and result in erroneous behavior. To combat this problem, reflection padding extends the image by reflecting inner image pixels outwards.

These padded pixel values are more plausible and reduce unwanted detection in the activation map, as is apparent in Figure 4.



(**a**) (**b**)

Figure 4. Comparison between a decoder activation map with zero-padding (**a**) and reflection padding (**b**). Reflection padding reduces artefacts at the edge of the activation map.

To regularize the network, spatial dropout instead of regular dropout is added after every convolution layer in the encoder. Regular dropout is less effective in convolution networks, as is discussed in [14]. The authors of [14] use a 10% drop rate in their CNN. Our experiments showed that further regularization at 30% drop rate results in best performance, given the low amount of training data.

The decoder part of the segmentation network decodes the latent representation into a segmentation mask. It stacks three resize convolutions, starting with 32 kernels. Every subsequent layer halves the filter count. Resize convolutions were used instead of the more common transpose convolutions to avoid checkerboard artefacts, as discussed in [15]. The resize layers of these convolutions use bilinear up-sampling to double the image dimensions along both axes. An additional, final convolution layer merges the activations the last eight filters into a grayscale image. A sigmoid activation function in the last layer limits the images to a zero to one range.

The general architecture was modeled to reflect common practices in the field of deep learning. The kernel counts are a power of two, doubling with every encoder layer, similar to SegNet and U-Net. Likewise, the decoder is roughly mirroring the encoder part of the network. The filter kernels were chosen to be relatively big at 7×7, to give the network a big receptive field of r = 127 px, as shown in the following calculation:

$$\mathrm{r} = \mathrm{k}_1 + \sum_{n=2}^{N} \left[\left(\prod_{m=1}^{n-1} s_m \right) \cdot (k_n - 1) \right] \tag{1}$$

where N is the total number of layers, s_m is the stride of the m-th layer and k_n is the kernel size of the n-th layer. The large receptive field allows the network to put individual output pixel into the context of a large input image region. This allows the network to accurately detect defects of low severity but big expanse.

The hyper-parameter configuration of the model was empirically arrived at, after the model was continuously fine-tuned over the course of several training runs. It was chosen to maximize accuracy while keeping inference time acceptable.

5.2. Training

The 128×128 px windows created during data collection are randomly grouped into batches of size 128 and fed into the network. For the entire training set, this results in 250 batches. These batches are iterated over during every training epoch. Due to its fast convergence, the Adam optimizer with a learning rate of 0.001 was chosen for training the network [16]. The optimizer applies gradient updates to the network weights to lower a binary cross-entropy loss function. The graph in Figure 5 displays training and validation loss of the network during training. A 20% subset of the training data is withheld from training and used for validation. Since this validation data also consists of unrealistic simulated defects, it is not wholly representative of the network's final generalization error on real

data. Therefore, a smaller validation dataset of real defect data was generated. These samples were manually labeled to measure the network's generalization error to real defects. After every epoch during training, the errors for both validation sets are recorded. Every epoch of training took around one minute to complete on a Nvidia RTX 2070 Super GPU.

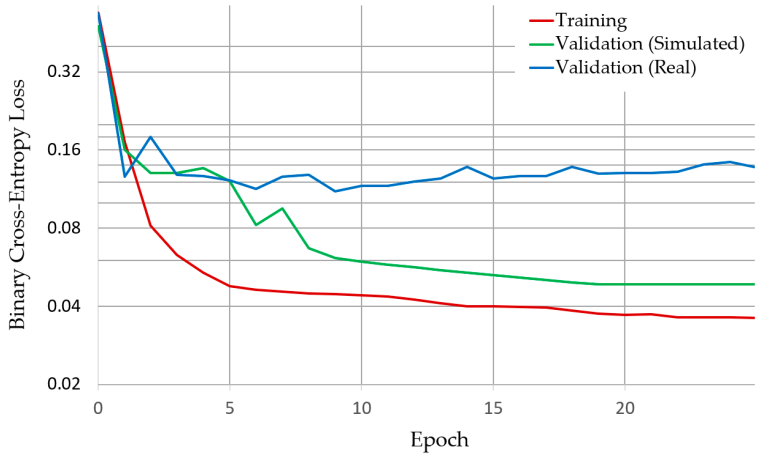


Figure 5. Training graph and two validation graphs for a validation dataset consisting of real and simulated defects respectively, plotted over the course of training.

As apparent in Figure 5, the blue validation curve reaches its minimum after the ninth epoch, which took around nine minutes to train. The continuing downward trend in the other two curves indicates that further training would reduce their respective errors. However, the network would then over-fit on simulated defects and perform worse on real defects. This fast over-fitting on simulated defects can be attributed to their dissimilar appearance to real defects. Stopping the training when the validation error on real data is smallest is necessary as a regularizing measure. This so-called early stopping allows generalization from defect simulations to real defects. In repeated trials, the lowest generalization error on real defects reliably lay between the fifth and tenth epoch. When the system is retrained and no validation data is available, the above-mentioned early stopping point can be used to avoid overfitting. The complete data collection and training process is summarized in Figure 6.

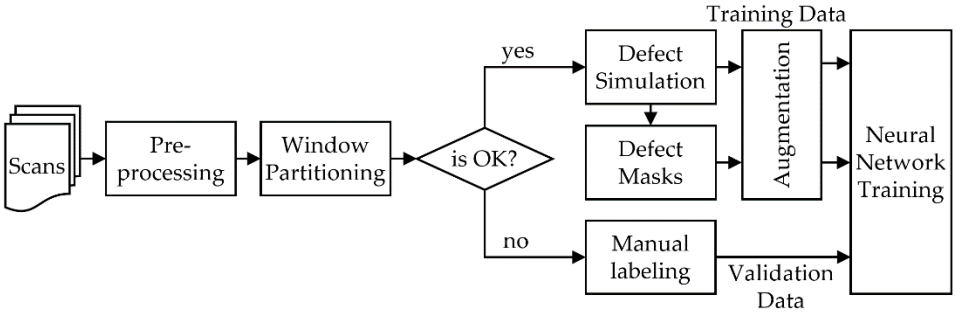


Figure 6. A flowchart of the data collection and training process. A dataset of simulated defects is generated for training. Defect samples in the scanned parts are labeled and used as validation data.

5.3. Post-Processing

The segmentation image created by the neural network marks defect image regions and their intensity. Further processing is required to translate the segmentation into a binary classification for the whole rolling element. If any of the defect segments found by the network escapes the tolerance threshold and is of significant intensity, the whole part is rejected. Algorithmically, the image is convolved with a Gaussian kernel. This reduces noise in the image and lowers the amplitude of small defects. Pixels of larger defect segments are only averaged with neighboring defect pixel, thus retaining

their peak amplitude. The resulting image only leaves big enough defects that escape the tolerance threshold. In a final step before classification, the image is binarized by rounding its pixels to either 0 or 1, whichever is nearest. If any pixel in the binarized image differs from zero, the whole rolling element is discarded.

5.4. Inference

During its application, the system runs on a workstation desktop PC with a Nvidia RTX 2070 Super GPU. The camera captures one 1395 × 3335 px image per 0.5 s. During this time, the system preprocess, segments and post-processes the previous image. The preprocessing part of the system is added to the system after training. It consists of converting the images from RGB scans to gray images and normalizing them. This is necessary, since the network was trained on gray images, as was discussed in Section 4. The conversion is done with a preset convolutional layer. This layer has a single filter kernel with a 1 × 1 size. Its weights correspond to the channel weights as defined in the CIE 1931 linear luminance and shown in Equation (2). Implementing the conversion as a convolution layer allows us to add it to the neural network's computational graph.

$$\mathrm{Y} = 0.2126 \cdot \mathrm{R} + 0.7152 \cdot \mathrm{G} + 0.0722 \cdot \mathrm{B} \tag{2}$$

The normalization of the images deducts their mean and divides every pixel by the standard deviation of the image. This makes the whole system invariant to differences in brightness as they can arise when the camera's aperture is changed or the LED lighting ages.

Both preprocessing steps, as well as both post-processing steps described in the previous section were added to the computational graph and run on the GPU. This allows these operations to be accelerated though parallelization. Furthermore, this means the entire system from input image to binary classification is encapsulated in a single graph definition. Updates to the classification system only require a single file to be swapped during production. This further reduces maintenance time.

Since the segmentation network consists solely of size invariant operations, it can process images of any given size. While the network was trained on windows of size 128 × 128 px, the inference is done on the whole image of size 1395 × 3335 px. The batch size during inference is one, resulting in memory requirements of similar magnitude. Segmenting the whole image at once removes the necessity to slice up the image for inference. This removes artefacts that would occur at the cutting edges. At the edge of the scan itself, reflection padding further reduces artefacts, creating a clean segmentation. Figure 7 summarizes the inference process.

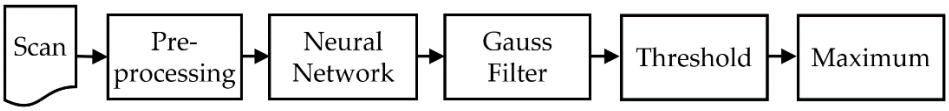


Figure 7. A flowchart of the inference procedure. A single scan at a time is preprocessed and fed through the neural network. The resulting segmentation is filtered and binarized via a threshold. The maximum in the resulting binary image, either one or zero, is the final classification.

6. Evaluation

As mentioned in the introduction, defect elements are a rarity. Due to this, 10 non-defect rolling elements were manually damaged to create a test dataset. These parts were not used during training of the neural network. To test the system on numerous defects, each rolling element was damaged with an average of 5 defects. After scanning the parts, this resulted in a total test-set of 50 defect image regions. The system was then evaluated on how many of these defects were correctly found on the 10 test images.

If the system's performance would be rated on a per part basis, it would suffice to only detect one of multiple defects on a given part. Even though the other defects could be missed, the resulting

classification is correct. This would inflate the system's accuracy, as these false negatives are not factored into the resulting score. Instead, measuring the system's accuracy on individual defects within an image prevents this. Since there are more defects than parts, this also allows for a much finer accuracy measurement. In practice, the predicted binary segmentations are evaluated instead of the final binary classification of the part. The former highlights individual defects, while the latter only classifies the whole part. However, during the system's later deployment only the binary classifications of the individual images are of interest.

To measure the accuracy of the network predictions, they were compared to manually created segmentations. The system was evaluated using an intersection over union (IoU) score between the labels and the network predictions, as is standard in image segmentation problems in literature [11]. IoU is a normalized measure of the overlap between two segmentation masks. It is defined as:

$$\mathrm{IoU}(y,\ \hat{y}) = \frac{\left|y \cap\ \hat{y}\right|}{\left|y \cup\ \hat{y}\right|} \tag{3}$$

where $\hat{y}$ is the network prediction and y is the manually created target segmentation.

Figure 8 visualizes the measures used in the IoU score. The pixels highlighted in green in the rightmost image are in the intersection of both binary masks to the left. Together with the pixels highlighted in red and yellow, they form the union of both masks.

(**a**) (**b**) (**c**) (**d**)

Figure 8. (**a**) An excerpt of a rolling element scan with a defect. (**b**) The binarized network segmentation prediction corresponding to the excerpt in (**a**). (**c**) The manually created label corresponding to the excerpt in (**a**). (**d**) Both binary masks in the left images are overlaid on top of the original input image. The intersection of both masks is highlighted green, while the false positive predictions of the network are highlighted in yellow and the false negatives in red.

Evaluated on the test set, the system achieved an IoU score of 98.0%. Neither the manually drawn masks, nor the predicted masks can be expected to pixel perfectly overlay the defect. Achieving higher scores through further parameter tweaking thus seems unlikely. Big defect areas that are correctly classified contribute the most to a high IoU score. At the same time, these defects are easiest to detect. Due to this, we further measured the detection rates of defects. If the system detected at least one NOK pixel in a defect, the defect is counted as detected. It is crucial to detect all defects on a part in the given setting. Over the whole test dataset, the evaluation resulted in the scores listed in Table 2 for detection of single defects. Here, false negatives (FN) count the number of defects missed by the system. False positives (FP) count the number of wrong detection of defects, where the surface is actually intact. The true positives (TP) count the number of correctly detected defects. Additionally, the listed counts are combined into an F_1 score, according to the following equation:

$$F_1 = \frac{2TP}{2TP + FP + FN}, \tag{4}$$

Table 2. True positive, false positive and false negative rate of the classification system on the test set and a combined F_1 score for the three rates.

TP	*FP*	*FN*	F_1
50	2	0	98%

The system correctly found all 50 defects. No defects were missed. Only two false positives were recorded, where the system predicted a defect where none was present on the manually created mask. Both false positives were close to each other. Figure 9 shows that they are a result of the same underlying image texture. The texture features small defects that are below the tolerance threshold. As such, they should not be detected by the system.

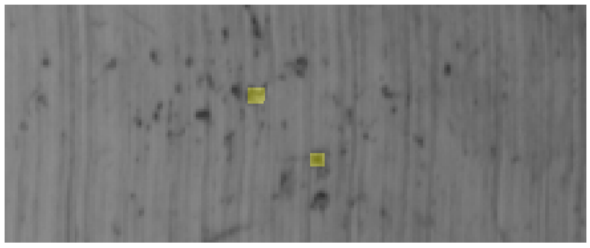


Figure 9. The two false negatives (yellow) that were predicted by the system. The underlying scan exhibits small defects, but below the tolerance threshold.

The complete evaluation process is summarized in Figure 10. Further testing on a larger test set is required to confirm the above-mentioned measurements. Given the risk associated with false negatives, trading off a lower false positive rate with a higher false negative rate is inadvisable. Due to the limited test-set size, the given results serve only as preliminary insights. It is possible that the limited test-set does not capture the full variance of defects encountered in production. The measured accuracy during later deployment could therefore be lower.

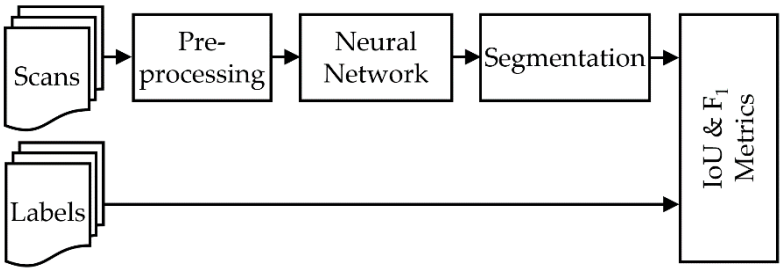
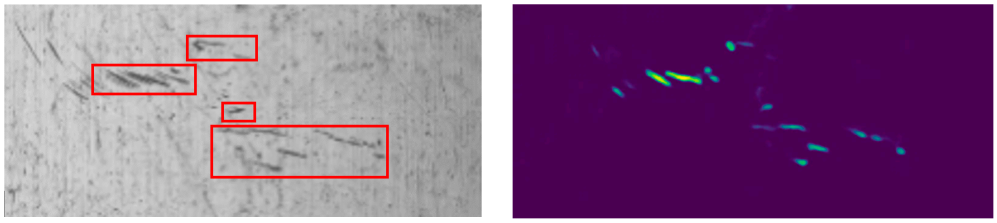


Figure 10. A flowchart depicting the evaluation procedure. The test data flows through a pipeline, similar to the inference pipeline, but without the post-processing steps. The predicted segmentations are then compared with manually created label masks, according to the two mentioned metrics.

Averaged over 25 runs, the system takes 153 ms to process one 1395 × 3335 px surface scan. This allows later scalability of the system to more or bigger scans during one scan period. Figure 11 shows an exemplary output of the network for an input image with a defect.



(**a**) (**b**)

Figure 11. (**a**) Excerpt of a rolling element. Defects were manually highlighted with bounding boxes. (**b**) Segmentation image of the neural network before post-processing, displayed with viridis color scheme.

7. Conclusions

Based on only a limited, unlabeled, and unbalanced dataset, we developed a classification system that reliably detects defects on the surface scans of rolling elements. Using heavy augmentation and windowing, the size of the dataset could be multiplied until sufficient for training of a neural network. The chosen convolutional encoder-decoder architecture was trained to segment its input data. Simulated defects bypassed the need for rare, real defective parts and lowered the manual labeling efforts required for training. While simplistic, experiments showed that the simulated defects were viable in training the network to perform well on real data. It is unlikely however, that the presented approach would work on a task where defects are too complex to reliably simulate. This limits the proposed system's application to domains, where defects are well understood, simple and clearly defined.

The data required and the low training times of the neural network allow the system to be quickly retrained. This enables it to adapt to changes in the production process and different classification circumstances. Thanks to a GPU implementation of all necessary pre- and post-processing operations, the system allows rapid inference during its working phase. Preliminary laboratory tests resulted in satisfactory classification rates. In the future, a bigger test in a production environment is required to confirm these results.

Author Contributions: Conceptualization, N.P. and M.B.; data curation, A.L.; methodology, N.P. and M.B.; project administration, G.B.; software, N.P.; supervision, G.B. and F.D.; validation, N.P.; visualization, N.P.; writing—original draft, N.P.; writing—review and editing, M.B. and G.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Free State of Bavaria and Fertigungsgerätebau Adolf Steinbach GmbH & Co. KG, 97616 Salz, Germany, grant number IUK514/002.

Acknowledgments: The authors would like to thank Fertigungsgerätebau Adolf Steinbach GmbH & Co. KG for providing the steel rolling elements for the above-mentioned data collection process and providing feedback and input for the hardware design.

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

References

1. Yang, S.; He, Y.; Wang, Z.; Zhao, W. A Method of Steel Strip Image Segmentation Based on Local Gray Information. In Proceedings of the IEEE International Conference on Industrial Technology, Chengdu, China, 21–24 April 2008.
2. Ghoria, S.; Mukherjee, A.; Gangadaran, M.; Dutta, P.K. Automatic Defect Detection on Hot-Rolled Flat Steel Products. *IEEE Trans. Instrum. Meas.* **2013**, *62*, 612–621. [CrossRef]
3. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. *Adv. Neural Inf. Process. Syst.* **2012**, *25*, 1097–1105. [CrossRef]
4. Wen, S.; Chen, Z.; Li, C. Vision-Based Surface Inspection System for Bearing Rollers Using Convolutional Neural Networks. *Appl. Sci.* **2018**, *8*, 2565. [CrossRef]
5. Zhou, S.; Chen, Y.; Zhang, D.; Xie, J.; Zhou, Y. Classification of Surface Defects on Steel Sheet using Convolutional Neural Networks. *Mater. Technol.* **2017**, *8*, 123–131.

6. Tao, X.; Zhang, D.; Ma, W.; Liu, X.; Xu, D. Automatic Metallic Surface Defect Detection and Recognition with Convolutional Neural Networks. *Appl. Sci.* **2018**, *8*, 1575. [CrossRef]
7. 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]
8. Staar, B.; Lütjen, M.; Freitag, M. Anomaly detection with convolutional neural networks for industrial surface inspection. In Proceedings of the 12th Conference on Intelligent Computation in Manufacturing Engineering, Gulf of Naples, Italy, 18–20 July 2018.
9. Czimmermann, T.; Ciuti, G.; Milazzo, M.; Chiurazzi, M.; Roccella, S.; Oddo, C.M.; Dario, P. Visual-Based Defect Detection and Classification Approaches for Industrial Applications—A SURVEY. *Sensors* **2020**, *20*, 1459. [CrossRef] [PubMed]
10. Xie, X. A review of recent advances in surface defect detection using texture analysis techniques. *ELCVIA Electron. Lett. Comput. Vision Image Anal.* **2008**, *7*, 1–22. [CrossRef]
11. Badrinarayanan, V.; Kendall, A.; Cipolla, R. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *39*, 2481–2495. [CrossRef] [PubMed]
12. Ronneberger, O.; Fischer, P.; Brox, T. U-Net: Convolutional networks for biomedical image segmentation. In Proceedings of the 18th International Conference on Medical Image Computing and Computer-Assisted Intervention, Munich, Germany, 5–9 October 2015.
13. Springenberg, J.T.; Dosovitskiy, A.; Brox, T.; Riedmiller, M. Striving for Simplicity: The All Convolutional Net. In Proceedings of the International Conference for Learning Representations Workshop, San Diego, CA, USA, 7–9 May 2015.
14. Ghiasi, G.; Lin, T.Y.; Le, Q.V. DropBlock: A regularization method for convolutional networks. In Proceedings of the 32nd Annual Conference on Neural Information Processing Systems, Montreal, QC, Canada, 3–8 December 2018.
15. Odena, A.; Dumoulin, V.; Olah, C. Deconvolution and Checkerboard Artifacts. *Distill* **2016**. [CrossRef]
16. Kingma, D.P.; Ba, L.J. Adam: A Method for Stochastic Optimization. In Proceedings of the 3rd International Conference for Learning Representations, San Diego, CA, USA, 7–9 May 2015.