Received: 12 July 2024 | Revised: 10 September 2024 | Accepted: 26 October 2024 | Published online: 31 October 2024

RESEARCH ARTICLE/REVIEW

Journal of Data Science and Intelligent Systems
2024, Vol. 00(00) 1–9
DOI: 10.47852/bonviewJDSIS42023833

BON VIEW PUBLISHING

Automated Defect Detection Using Image Recognition in Manufacturing

Sercan Dikici[1] and Rachel John Robinson[1],*

[1]*Department Computer Science, IU International University of Applied Science, Germany*

Abstract: This abstract examines the process of creating a binary classification project on defect detection in manufacturing, highlighting significant learnings and prospective areas that might be tackled differently if the project was to be redone. Obtaining data from Internet sources is the first critical stage in the procedure. This stage involves obtaining essential datasets from reputable websites. Acquiring high-quality and varied data is critical to the project's success since it serves as the foundation for further analysis and modeling. After acquiring the data, the investigation and comprehension of the dataset begin. This process entails extensive study and analysis to get insights into the data's structure, properties, and distribution. Visualization tools are used to comprehend the insights of the data. After understanding the data, the following step is to construct an efficient input pipeline. This entails preparing and processing the data in order to provide a streamlined and efficient pipeline for the model. The model is constructed using convolutional neural networks (CNNs) in TensorFlow using Python after the input data have been set up. CNNs are a good choice for this project since they do jobs involving images very effectively. To improve the model's performance and avoid overfitting, activation functions, optimization methods, and regularization approaches are carefully selected. The Early Stopping strategy is used with the patience parameter to optimize the training process. When using Early Stopping, the training process is stopped if the performance on the validation set does not increase after a predetermined number of epochs. The model architecture is effectively developed, trained, and optimized by utilizing TensorFlow and Python, enabling effective defect identification in the manufacturing process.

Keywords: convolutional neural networks, image recognition, Early Stopping strategy, defect identification

1. Introduction

Reducing processing mistakes in the small production process is critical for increasing profitability in the manufacturing business. To eliminate processing mistakes, a quality assurance budget must be established, manual inspection work must be implemented, and the production process must be reviewed. Many firms carry out the inspection process manually; however, there are issues such as inconsistent accuracy, reliance on inspection staff, and increasing labor expenses.

There are several types of defects that can occur in casting processes, and the most common defects can be identified are as follows:

- Casting Shape Defects
- Blowholes, Pinholes, Open Holes
- Metallurgical defects
- Swells
- Drops
- Shrinkage defects
- Pouring metal defects [1]

The application domain of visual inspection has been significantly and directly impacted by automated surface-anomaly identification using machine learning (ML), which has emerged as an intriguing and promising topic of research [1–3]. In this project, we will test if the problem of "manual inspection" can be removed by automating the inspection process in the casting product production process using ML. By using ML algorithms, we can train a model utilizing a dataset of tagged photos of both faulty and non-defective castings. The model will learn the distinct patterns associated with each sort of defect, allowing it to reliably categorize fresh photos.

The dataset comprises a substantial 10,000 collection of grayscale images. These images, measuring 300 × 300 pixels, have been augmented to diversify the dataset and improve the model's performance. The dataset is split into two folders for training and testing a classification model. The "train" folder contains a significant portion of defect impeller images, while the "test" folder includes a smaller percentage of defect and non-defect impeller images.

When seeking the former works, it has made significant strides in automating surface-anomaly detection using ML, but they possess several limitations. Major weakness is the reliance on insufficiently varied and often imbalanced datasets, which can result in overfitting and a lack of generalization when models are exposed to novel defect types or changes in environmental conditions. Also, few works may inadequately address the issue of data augmentation specifics and optimization techniques to enhance model robustness. Extending it, the integration of practical deployment aspects, such as real-time processing capabilities and user-friendly interfaces, that would facilitate seamless adoption in live manufacturing environments has not been thoroughly investigated in prior

***Corresponding author:** Rachel John Robinson, Department Computer Science, IU International University of Applied Science, Germany. Email: rachel.john-robinson@iu.org

research. The noted gaps prompt further inquiry to develop more comprehensive, resilient, and deployable solutions in the defect detection domain.

The concept of this work is to develop an accurate and efficient system for detection in manufacturing using supervised learning methods and image recognition techniques. The main goal is to improve productivity, reduce costs, and enhance product quality in the casting industry through automated defect detection.

The structure of the paper starts with studying the theoretical base coupled with the underlying methodology for this research in turn getting into the test and results to frame conclusions.

2. Literature Review

Prior studies have shown that automated flaw identification in manufacturing processes has the potential to improve product quality and save costs. According to research by [4], the ML techniques can perform better than traditional machine-vision algorithms that use support vector machines and hand-engineered features to classify surface defects. They proved this by using a convolutional neural network (CNN) with five layers to classify images of various steel defect kinds. They got outstanding results; nevertheless, their work was limited since they did not employ ReLU as an activation function or batch normalization.

Similar deep learning method is used by [5, 6], and their study achieved a high accuracy rate of 92% in identifying rail surface defects. They used ReLU and Tanh as activation function and compared and presented the performance results. One of their important findings was although network training takes longer, the big DCNN model outperforms the small and medium DCNN models for classification.

Ref. [7] presented a more efficient network for explicitly executing defect segmentation. In order to segregate the flaws, they used a fully convolutional network with 10 layers and both ReLU and batch normalization. They achieved very good results (close to 99%) in both anomaly segmentation and classification. The suggested network has a low parameter count, is stable over a wide range of surface textures, and gives visual localization and categorization explanation.

In order to find flaws in metal casting products, [7] used computer vision techniques and ML algorithms. They used a segmentation network that locates the surface fault pixel-by-pixel and image processing methods to extract features. Their research has a high accuracy rate for a particular goal (crack detection) and on a particular kind of surface, although the network's architecture was not built just for this domain.

In comparison to these related methods, the approach used in this project follows a similar architecture proposed by [7]. We plan to employ a binary classification network that locates the surface defect in pixels and classifies the image as defect or non-defect.

The technical underpinning of the problem involves several key points. By leveraging these technologies and techniques, we were able to develop an effective CNN model for automated defect detection in manufacturing. Within this section, the paper is going to discuss some of the technical concepts such as a) choosing the ML model, training, and evaluation, b) software and imported libraries, c) data collection and preparation.

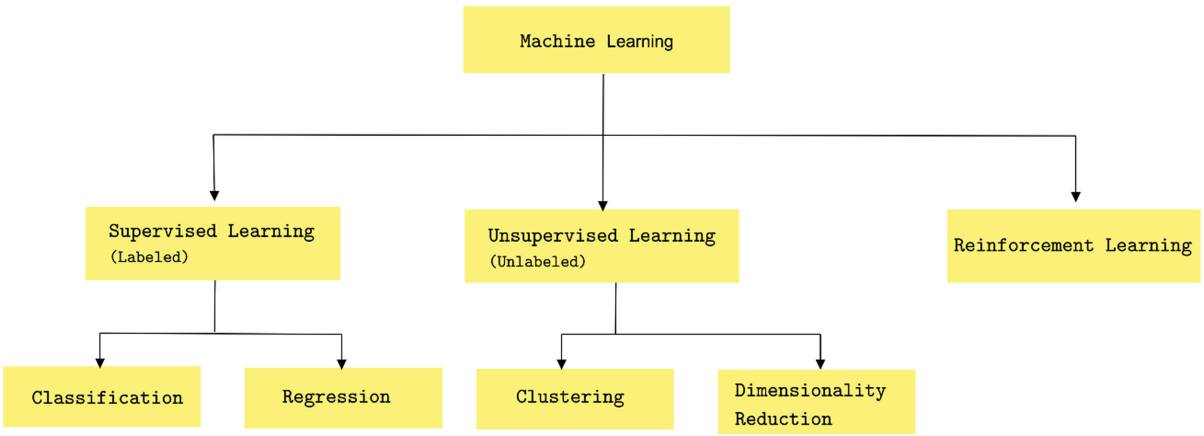
2.1. Choosing the ML model, training, and evaluation

ML features have lately been used on a range of computer vision and classification issues, with success in several domains [6, 8]. The primary benefit of employing ML is that once an algorithm knows what to do with data, it can do it automatically. There are a variety of ML algorithms, and the type of method used is determined by the type of problem to be solved, the amount of variables, the appropriate model for it, and so on. Here's a representation of some of the most often used algorithms in ML [6, 8].

Supervised learning is distinguished by the use of labeled datasets to train algorithms that properly categorize data or predict outcomes (What Is Supervised Learning?) Deep learning is a subset of ML, which uses numerous layers of neural networks to process vast amounts of data and perform computations on it. CNN is a supervised form of deep learning that is most frequently employed in computer vision and image recognition [9]. Derivatives play a crucial role in optimizing Neural Networks, allowing to fine-tune the model's performance.

There are 3 layers used in neural networks, and these are the input layer, the hidden layer(s), and the output layer, and apart from the input layer, activation functions need to be employed while creating these layers to perform calculations. The input layer only stores the incoming data, and there are no calculations

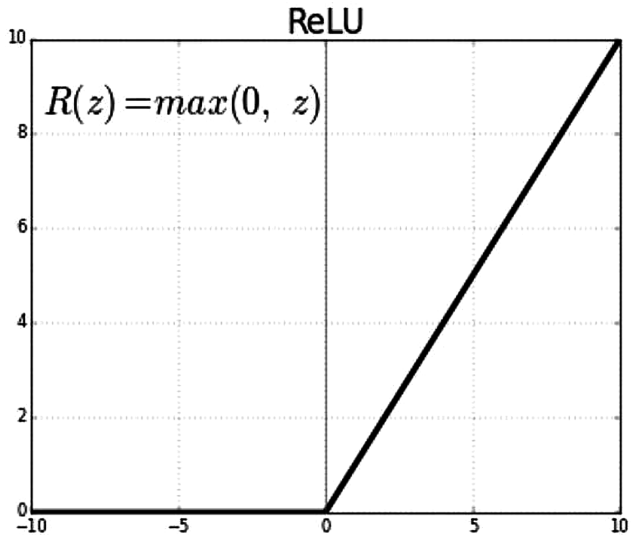
Figure 1
Machine learning algorithms



Source: Adapted from [9]

made here; therefore, there is also no activation function used. There are some activation functions that are commonly used, and they are responsible for introducing non-linearity into the inherently linear operations of neural networks [2, 10]. When building a binary classification model under supervised learning, the sigmoid function needs to be utilized at the output layer, where the output is interpreted as a class label [2]. In the hidden layers of the model, we utilized the widely adopted activation function known as Rectified Linear Unit (ReLU), depicted in Figure 2. ReLU is a commonly employed activation function in Neural Networks for hidden layers and works well with supervised and unsupervised learning models, enabling effective modeling of complex relationships between variables. It works especially well when the precise variables and their connections are not well stated. ReLU aids in identifying and modeling complex patterns and relationships within the data by introducing non-linearity and allowing the network to learn from the data [3]. In essence, it maintains positive input values while setting negative input values to zero.

Figure 2
ReLU activation function



Source: Adapted from [3]

Following this information above, we now can suggest that CNN would be the best method for finding solution to the business case problem. To progress and achieve better result in training the model, the dataset was divided into training, validation, and testing sets. We made model parameter improvements as the training process progressed.

2.2. Software and imported libraries

We made use of a variety of software tools, coding languages, and libraries specific to the issue. This might have included a source-code editor called Visual Studio Code and well-known programming language, Python.

Popular libraries like pandas, NumPy, and Matplotlib were used for data processing and visualization. NumPy allowed for efficient numerical computations, while Pandas offered efficient data analysis and manipulation capabilities. Data visualization and the creation of instructive graphs were accomplished using Matplotlib. We used Keras pre-processing's Image Data Generator to manage the images in the dataset. The library offered features for manipulating, loading, and saving images. This made it possible to prepare and pre-process the photos for additional analysis. TensorFlow in combination with Keras is used as library, a high-level neural networks API, to model the CNN. Callbacks like Model Checkpoint and Early Stopping were added to save the best model weights during training and prevent overfitting. I used the sklearn library's confusion matrix and Confusion Matrix Display methods to assess the performance of the model. We were able to construct a confusion matrix and illustrate the categorization results as a consequence of this. We also used the classification report function to obtain classification metrics such as precision, recall, and F1-score. These technological components supplied the essential tools and procedures for pre-processing the data, building the neural network, and properly evaluating its performance.

2.3. Data collection and preparation

Kaggle is a prominent site for sharing and researching datasets, providing with a dataset linked to the issue domain, especially manufacturing faults or product quality. Industry professionals, researchers, or data lovers may have contributed to the dataset. The dataset is made up of top-view photos of submersible pump impellers; the source can be reached using this link by [8, 11]. The collection contains 7,348 photos, all of which are grayscale and 300x300 pixels in size. These photographs have previously been augmented to increase the variety of the dataset and improve the model's performance. Additionally, the dataset was divided into two folders for training and testing a classification model. There are 3,758 photographs of "defect" impellers and 2,875 images of "non-defect" impellers in the "train" folder. Similarly, 453 photographs of defect and 262 images of okay status are included in the "test" folder.

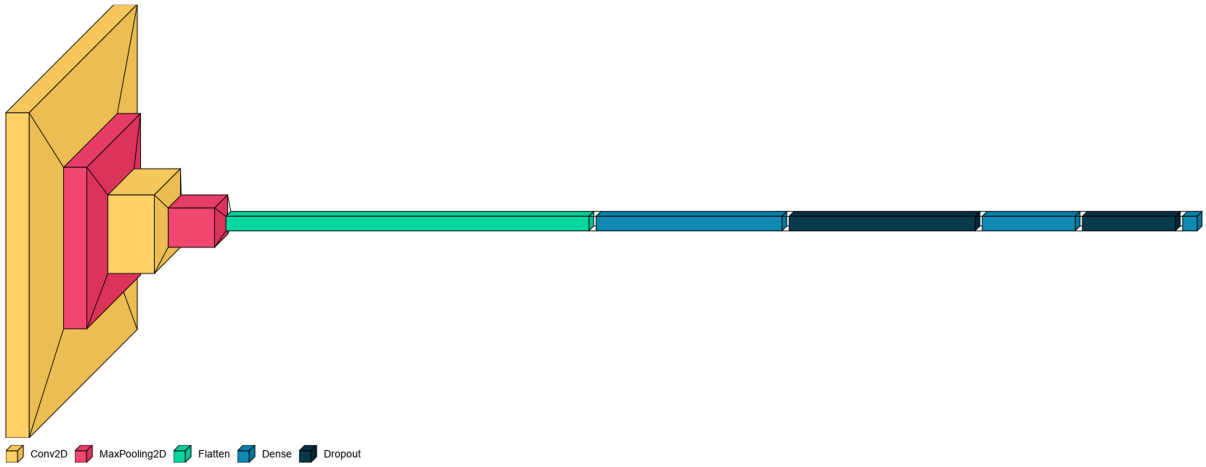
3. Research Method

We constructed a ML model for the solution to the problem and an ML model usually follows this workflow below [12].

- Examine and understand data: This step involves exploring and analyzing the dataset to gain insights into its structure, features, and distribution.
- Build an input pipeline: In this step, the data are prepared and processed to create an efficient input pipeline for the model. This includes tasks such as data pre-processing, feature extraction, data augmentation, and data splitting into training, validation, and testing sets.
- Build the model: This step involves designing and constructing the model architecture. Depending on the problem and data characteristics, suitable ML or deep learning models are chosen.
- Train the model: In this step, the model is trained on the training data using an optimization algorithm and a defined loss function. The model learns to adjust its parameters by iteratively making predictions.
- Test the model: After training, the model is evaluated on the testing data to assess its performance and generalization ability.
- Improve the model and repeat the process: Based on the evaluation results, the model can be further refined and improved. The steps 2 to 5 are repeated until satisfactory results are achieved.

Building and training the model is one of the most important steps, and these are the places where we implement the solution to

Figure 3
CNN architecture of the model



Source: Own representation using visualkeras [13]

the problem. The process of dividing a set of elements into two groups based on a classification rule is known as binary classification [13]. Therefore, the problem of whether to classify the surface of the product as a defect or a non-defect is a binary classification problem. The method we use in steps 3 and 4 is a CNN which outperforms other neural networks when given inputs such as images, voice, or audio. CNNs are made up of several layers, including pooling, convolutional, flatten, and fully linked layers. These layers are coupled to one another and have activation mechanisms including sigmoid and ReLU. The proposed network architecture of CNN for the models' training can be visualized as below.

Convolutional Layer (Conv2D): The central component of a CNN is the convolutional layer, which is also where the majority of computation takes place. It needs input data, a filter, and a feature map, among other things (What Are CNN?). ReLU modification is applied to the feature map following each convolution operation by a CNN, adding non-linearity to the model. The model takes grayscale images of size 300 × 300 pixels as input and extracts features through convolutional layers.

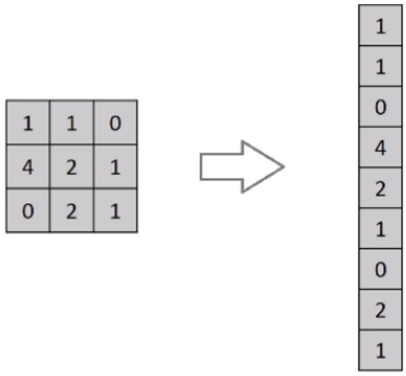
Pooling Layer (MaxPooling2D): The convolved features spatial size is decreased by the pooling layer. By lowering the dimensions, this will lower the amount of CPU power needed to process the data [14]. We proposed the Max Pooling layer with a pool size of 2 × 2 and a stride of 2. It performs downsampling, reducing the spatial dimensions of the feature maps.

Flatten: In essence, flattening is the conversion of a matrix obtained by convolutional and pooling methods into a one-dimensional array. This is significant because a one-dimensional array is the requirement for the input of fully connected layers [10, 15].

Fully Connected Layer (Dense): Based on the features that were gathered by the preceding layers' various filters, this layer conducts the classification operation (What Are CNN?). As the visualization below, each dense layer has specific number of neurons which are connected to neurons in the next layer. We proposed 2 layers of fully connected layer (hidden layer) with a size of 128 nodes each. Both of them apply a ReLU activation function, introducing non-linearity.

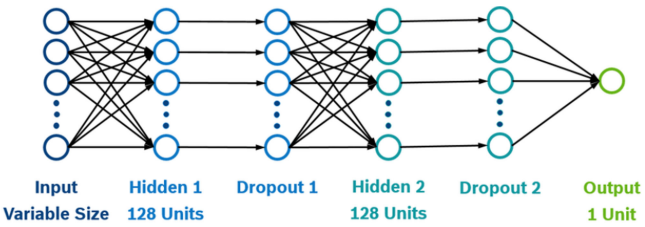
Dropout: In order to avoid overfitting, the Dropout layer randomly sets input units to 0 with a frequency of rate at each step during training [16]. To prevent overfitting, a Dropout layer

Figure 4
Flatten visualization



Source: Adapted from [10, 15]

Figure 5
Fully connected layer

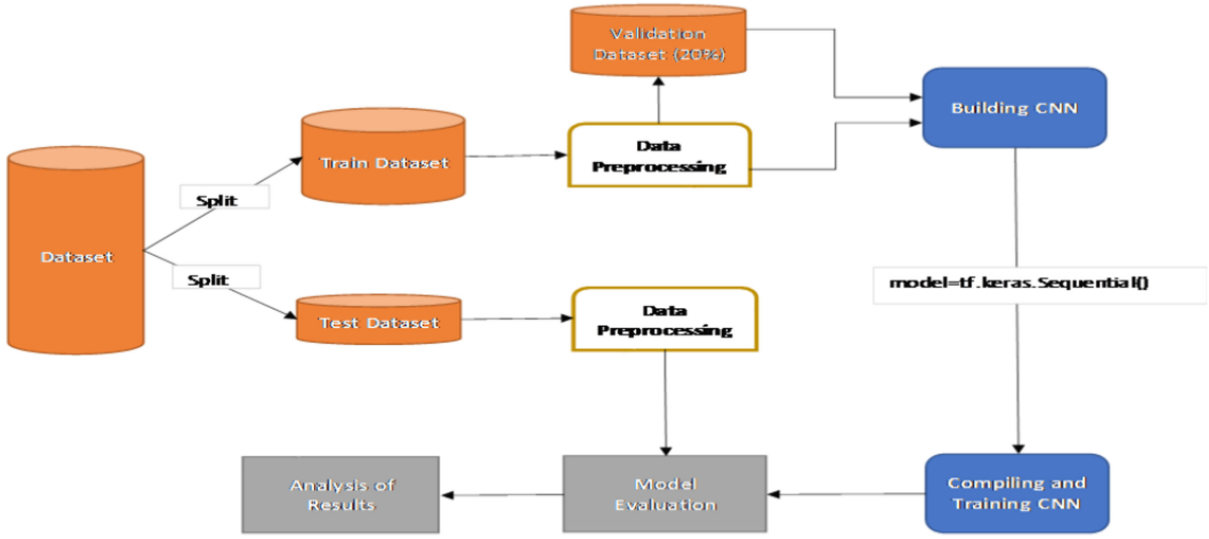


Source: Adapted from [16]

is added with a rate of 0.3. It implies that every epoch, 30% of the neurons in this layer will be dropped at random.

Finally, a single node representing the binary classification is used to generate the output layer. It makes use of a sigmoid activation function, which generates a number between 0 and 1 that represents the likelihood that the input image belongs to the positive class.

Figure 6
CNN model framework



Source: Own representation

4. Implementation and Testing

The implementation involves the use of a CNN model to classify images into defect and non-defect categories. Figure 6 below shows the model's framework in general and method of building.

After gathering the required dataset from the sources, the first step is importing the essential libraries and dependencies for data analysis, picture preparation, and model construction in the implementation. The libraries imported are:

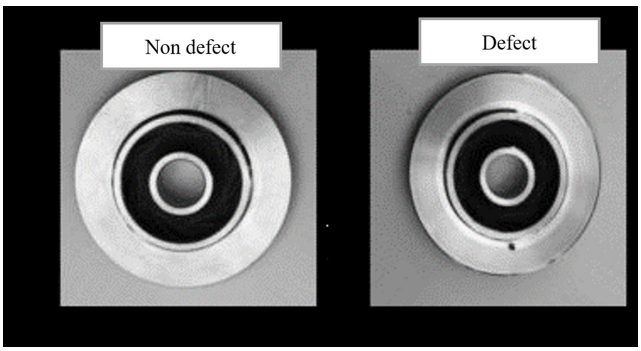
- Pandas, NumPy, Matplotlib, OS and image (from PIL) for data analysis and visualization
- Image Data Generator from Keras is for pre-processing.
- Tensor flow and Keras along with Model Checkpoint and Early Stopping for building and training the model.
- Confusion matrix, Confusion Matrix Display, and classification report from sklearn for model evaluation [17].

Importing the dataset from the local directory is the next step. Images from both defective and non-defective objects are imported from the dataset. A few illustrations of defective and non-defective goods are included to help the viewer understand the images. The figure below shows sample images from both categories.

The next step after importing the dataset from the local directory is showing the quantity of data instances in each class in order to gain insight into the dataset. This analysis helps in understanding the class distribution and potential data imbalances. To show the results, we used a pandas data-frame and counted the images in each directory with the Python's len() function, which gets the length of an array. The quantity of images in each directory is shown as below.

The step prior creating and developing the CNN model is the image pre-processing. This step in the example includes multiple sub-steps such as resizing images to a consistent size, converting them into grayscale images, rescaling pixel values to a range of 0 to 1 by dividing each pixel value by 255 and applying validation split. All of these steps can be done via Python code in seconds for whole data of over 7000 images, it is not necessary to perform a single step manually. It is common to train the model

Figure 7
Defective and non-defective products



Source: Own representation using Matplotlib

Figure 8
The quantity of images

	TRAIN DATA	*TEST DATA*	*Total*
OK	2875	262	3137
DEFECT	3758	453	4211
Total	6633	715	7348

Source: Own representation using pandas

with the train data and evaluate the performance during the training with validation dataset. The training and validation datasets are typically divided 80:20 [18, 19]. Therefore, 20% of the data are reserved for validation. This is one of the most important steps which may affect the results of training.

The TensorFlow-Keras framework is used to implement the CNN model. A sequential model, as its name indicates, enables layer-by-layer, step-by-step model construction. The model architecture is made up of a number of layers, including convolutional, pooling, flatten, and dense layers. The purpose of

each layer is explained in section 4. The model code contains information on the parameters, such as the number of filters, kernel size, activation functions (ReLU), and others. The model.summary() function at the end provides detailed information about the layers, their output shapes, and the number of trainable parameters in the model.

After defining the steps of the model, the compile technique is used to create the model, which prepares it for training. The optimizer is set to "adam", which is a popular optimization algorithm for neural networks. Given that the problem is one of binary classification, the loss function is designated as "binary crossentropy" [15, 20]. The "accuracy" statistic is defined as well to assess how well the model performed during training. The weights of the model are saved during training using the Model Checkpoint callback. As a result, we are able to get the top-performing model at the conclusion of training. If the monitored metric (validation loss in the model) does not improve for a predetermined amount of epochs, the Early Stopping callback is built to end the training process early. It saves computing resources and aids in avoiding overfitting. The "patience" parameter is set to 3, which means that training will end if the validation loss does not improve for three successive epochs.

The model is trained using the fit approach. Both the validation data (validation scaled) and the training data (train scaled) are offered. The callbacks argument is used to specify the callbacks to be applied during training, including the Model Checkpoint and Early Stopping callbacks defined earlier. The model_fit variable will hold the training history, which contains details about the loss and accuracy values for each epoch, once the model.fit function has been run.

Model fit function is:

```
model_fit = model.fit(train_scaled, epochs=num_epochs,validation_data=valida-
tion_scaled,verbose=2,callbacks = [check, e_stop])
```

The loss and accuracy curves are presented to evaluate how well the trained model performed. These curves show how the model's performance develops across the training epochs and may be used to spot problems like overfitting or underfitting.

We can evaluate the model's generalizability and categorize fresh photos with accuracy by testing it with unseen data. The efficacy and dependability of the model are validated in this stage using examples from the actual world.

The trained model is tested against the test dataset in the provided code below to measure its performance. The test loss and test accuracy are computed using the model.evaluate() method using the pre-processed test photos, and their related labels are included in the test scaled dataset. The outcome of this piece of code shows the loss and accuracy values. Here the loss is 0.0131 (≈ 1%) and the accuracy level is 0.9944 (≈ 99%).

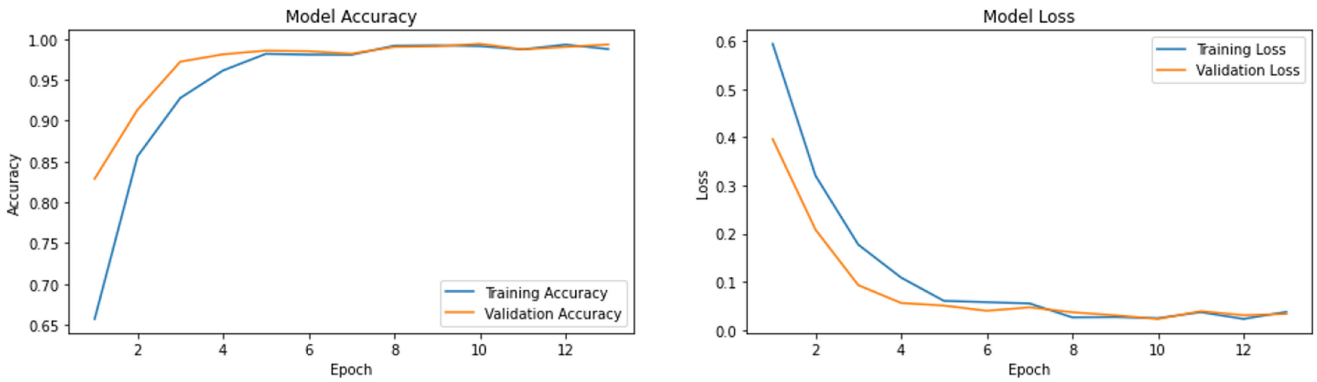
```
test_loss, test_accuracy = model.evaluate(test_scaled) #evaluate the model on the
test dataset
23/23 [==============================] - 3s 110ms/step - loss: 0.0131 - accuracy:
0.9944
```

Moving forward, the trained model is used to make predictions on the test dataset, and in order to gain insight into these predictions, I have created a confusion matrix. Confusion matrix table is used to describe how well a classification system performs [21–23]. We can learn more about the model's performance in accurately categorizing the test pictures by examining the confusion matrix, and we can also spot any misclassifications between the "OK" and "Defect" classes. Moreover, for assessing recall, precision, specificity, and accuracy, it is quite helpful. The visualization of confusion matrix shows that 4 images are misclassified by the model and 711 (449+262) test images are classified correctly. When we test this model on train dataset, we can find that we have even better predictions, all the data cases are predicted better by the model because the model is trained using these data points. In this case, we have only 15 misclassified points over 5307 images in total. The confusion matrix for calculated for test dataset and training dataset is computed and visualized as below.

In the confusion matrixes above, TP: True Positives, TN: True Negatives, FP: False Positives, and FN: False Negatives can be identified as follows:

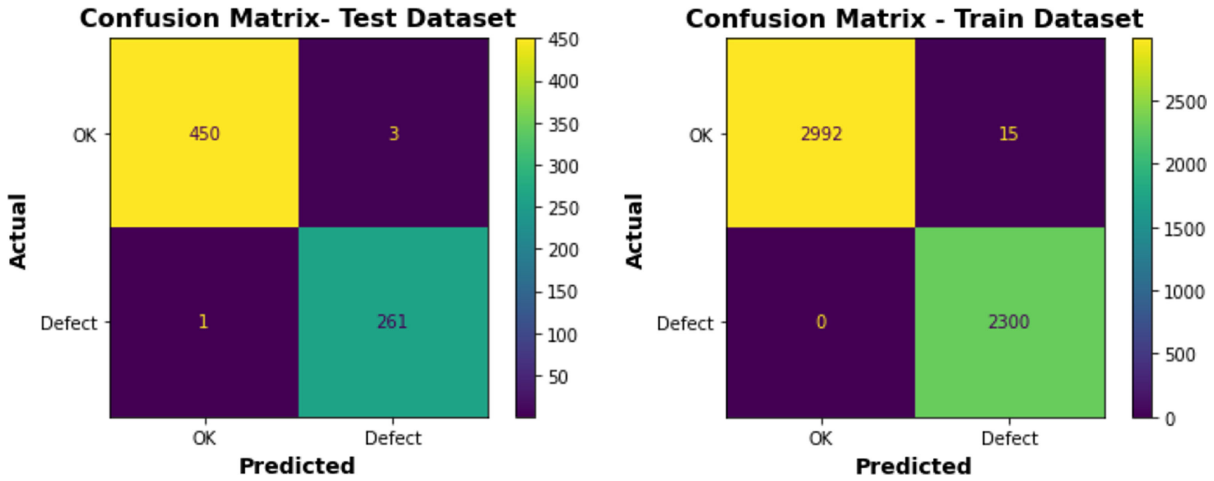
- TP: The cell where Actual OK and Predicted OK matches, 450 observations for test.
- TN: The cell where Actual Defect and Predicted Defect matches, 261 observations, for test.

Figure 9
The loss and accuracy curves



Source: Own representation using Matplotlib

Figure 10
Confusion Matrix



Source: Own representation using Matplotlib

- FP: The cell where Actual OK and Predicted Defect matches, 1 observation for test.
- FN: The cell where Actual Defect and Predicted OK matches, 3 observations for test. [11, 24]

Using confusion matrix, we can calculate accuracy, precision, and recall applying the results we received from the matrix to the formulas below.

$$Accuracy = \frac{TP + TN}{TP+TN+FP+FN} = \frac{450 + 261}{450 + 261 + 3 + 1} = \frac{711}{715} = 0.9944$$

$$Precision = \frac{TP}{TP+FP} = \frac{450}{450 + 1} = 0.9978$$

$$Recall = \frac{TP}{TP+FN} = \frac{450}{450 + 3} = 0.9934$$

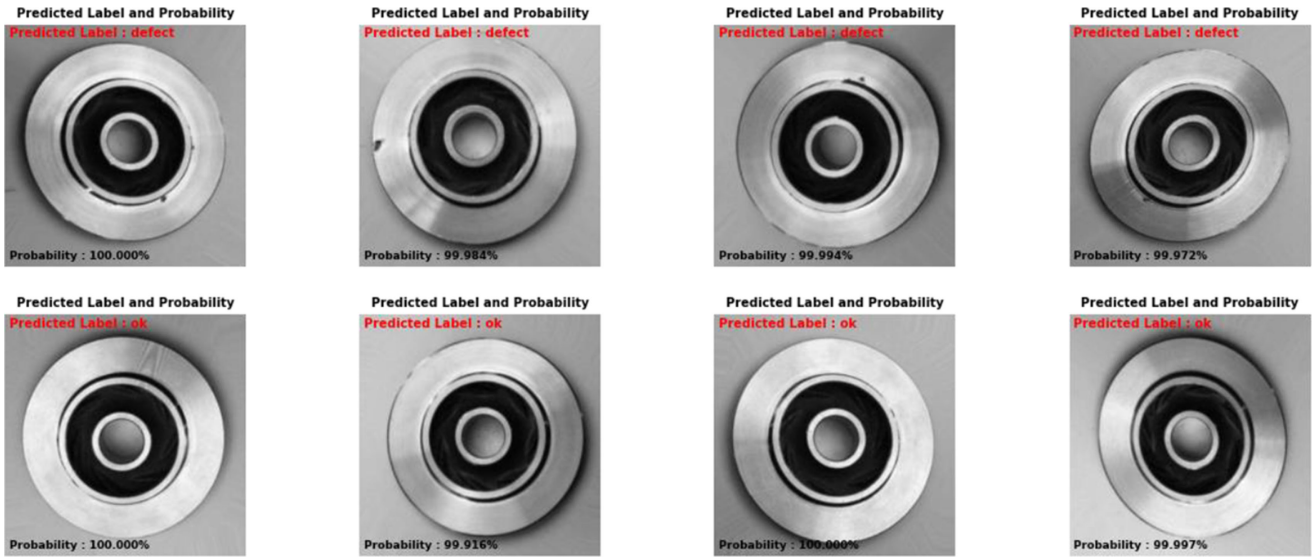
The classification report displays a representation of the major classification metrics on a per-class basis. This provides a more in-depth understanding of the classifier's behavior rather than global accuracy [22, 25]. The report for the case is shown below, where (a) precision is a measure of a classifier's exactness, (b) recall is the classifier's completeness, (c) the F1 score is a weighted harmonic mean of precision, the best score is 1.0 and the worst is 0.0, and (d) support is the number occurrences of the classes in the dataset. Generally, it provides with the overall accuracy of the model or the percentage of all samples that the classifier successfully identified.

	precision	recall	f1-score	support
0	0.9978	0.9934	0.9956	453
1	0.9886	0.9962	0.9924	262
accuracy			0.9944	715

In the last stage, the trained model is tested against 8 random pictures as samples from test dataset that it did not see throughout the training process. The purpose is to compute the probability along with the prediction label that each image belongs to their class. This testing phase enables to assess how well the model works on data on which it has not been explicitly trained, imitating real-world settings. By evaluating the probabilities, we can determine the model's effectiveness, its reliability, and applicability in practical manufacturing environments. The visualization of this stage is given in Figure 11 below.

As the results demonstrate high accuracy (99.44%) and low loss (1.31%) on a specialized test dataset, demonstrating that the model performs well in categorizing defects, potential disadvantages need to be considered. The low degree of misclassification indicates room for improvement, likely linked to data imbalance or specific defect types that the model finds challenging. Relative analysis against other state-of-the-art models like Vision Transformers or advanced ensembles used for similar purposes in literature could provide a clearer understanding of the model's relative performance, pros, and cons. In addition, the impressive results seen on the test set might indicate overfitting, as evidenced by slightly elevated metrics when applied back to the training set. These refer back to the need for enhanced generalization strategies. Data augmentation, more large model datasets featuring different defect types and environmental conditions, and techniques such as transfer learning could be used to train models on more generic and expansive datasets. It might be vital to assess the model's robustness across numerous manufacturing settings. Future research might explore implementing explainable AI mechanisms to gain further transparency into the model's predictions, hence boosting practitioners' trust in defect detection robustness. Discovering these areas holds strong potential for not only refining the existing model but also extending its applicability and reliability in various real-world manufacturing environments.

Figure 11
The predicted labels and probabilities



Source: Own representation using Matplotlib library

5. Conclusions and Recommendations

In conclusion, this research paper described a method for classifying surface defects using CNNs that can extract supervised features directly from the pixel representation of the steel defect photos and classify them as defect or non-defect. TensorFlow-Keras is used to implement the CNN model, which consists of convolutional, pooling, flattening, and dense layers. In order to collect and extract pertinent features from the input photos, the model architecture is created. The model is trained on the specific dataset and tested on other images which the model did not see during training. The model achieved very good performance, 99% accuracy, on classifying objects and detecting surface anomaly on this particular product; however, learning on new products is also possible with small modifications.

It is crucial to remember that the method we have suggested is specialized to a particular kind of activity and can have some application restrictions. For instance, complex 3D-manufactured items may need further analysis. Particularly, the tasks that can be expressed as binary classification problems with pixel-wise interpretation are best suited for the architecture we have created. This project can also be furthered by applying the flaw detection approach in a real work workplace. By creating a user-friendly interface, manufacturing staff may quickly upload photos for examination and get real-time feedback on the existence of flaw. For instance, Cognex is a major provider of machine-vision systems, particularly those built on CNN models. Cognex has an example-based system that rapidly and accurately identifies complicated abnormalities without the requirement for visual knowledge [19, 26, 27]. Overall, there is tremendous potential to enhance quality control procedures with the introduction of an automated defect detection system using picture recognition in manufacturing. Manufacturers can improve their capacity to recognize and categorize errors properly by utilizing CNN models and picture pre-processing techniques. This can then result in greater production, less waste, and higher levels of client satisfaction.

Despite there being wide research on deep learning methodologies for defect detection, this study's novel findings include achieving exceptionally high accuracy within the specific context of defect detection and processing a real-time, end-user-friendly deployment interface for practical manufacturing scenarios, adopting both high-performance and real-world applicability. In addition, while a comprehensive comparative analysis with different models would enrich the future research, the results provide a promising baseline. The work for the future can indeed involve more broad experiments and benchmarks to further validate and compare the present approach.

Ethical Statement

This study does not contain any studies with human or animal subjects performed by any of the authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest to this work.

Data Availability Statement

Data sharing is not applicable to this article as no new data were created or analyzed in this study.

Author Contribution Statement

Sercan Dikici: Conceptualization, Methodology, Software, Investigation, Resources, Data curation, Project administration. **Rachel John Robinson:** Conceptualization, Methodology, Validation, Formal analysis, Writing – Original draft, Writing – Review & editing, Visualization, Supervision, Project administration.

References

[1] Ahsan, M. M., Luna, S. A., & Siddique, Z. (2022). Machine-learning-based disease diagnosis: A comprehensive review. *Healthcare*, *10*(3), 541.

[2] Harjoseputro, Y. (2020). A classification Javanese letters model using a convolutional neural network with KERAS framework. *International Journal of Advanced Computer Science and Applications*, *11*(10). https://doi.org/10.14569/IJACSA.2020.0111014

[3] Ieracitano, C., Mammone, N., Hussain, A., & Morabito, F. C. (2020). A novel multi-modal machine learning based approach for automatic classification of EEG recordings in dementia. *Neural Networks*, *123*, 176–190.

[4] Brennan, M. C., Keist, J. S., & Palmer, T. A. (2021). Defects in metal additive manufacturing processes. *Journal of Materials Engineering and Performance*, *30*, 4808–4818. https://doi.org/10.1007/s11665-021-05919-6

[5] Chen, L., Li, S., Bai, Q., Yang, J., Jiang, S., & Miao, Y. (2021). Review of image classification algorithms based on convolutional neural networks. *Remote Sensing*, *13*(22), 4712.

[6] Gavrikov, P. (2023). *Visualkeras for Keras/TensorFlow [Python]*. Retrieved from: https://github.com/paulgavrikov/visualk-eras

[7] Faghih-Roohi, S., Hajizadeh, S., Núñez, A., Babuska, R., & De Schutter, B. (2016). Deep convolutional neural networks for detection of rail surface defects. In *2016 International Joint Conference on Neural Networks*, 2584–2589. https://doi.org/10.1109/IJCNN.2016.7727522

[8] Jiang, T., Gradus, J. L., & Rosellini, A. J. (2020). Supervised machine learning: A brief primer. *Behavior Therapy*, *51*(5), 675–687.

[9] Gosala, B., Chowdhuri, S. R., Singh, J., Gupta, M., & Mishra, A. (2021). Automatic classification of UML class diagrams using deep learning technique: Convolutional neural network. *Applied Sciences*, *11*(9), 4267. https://doi.org/10.3390/app11094267

[10] Khosla, P., Teterwak, P., Wang, C., Sarna, A., Tian, Y., Isola, P., . . . , & Krishnan, D. (2020). Supervised contrastive learning. *Advances in Neural Information Processing Systems*, *33*, 18661–18673.

[11] Pant, A. (2019). *Workflow of a machine learning project. Medium.* Retrieved from: https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94

[12] Kattenborn, T., Leitloff, J., Schiefer, F., & Hinz, S. (2021). Review on convolutional neural networks (CNN) in vegetation remote sensing. *ISPRS Journal of Photogrammetry and Remote Sensing*, *173*, 24–49.

[13] Kersten, J., Bongard, J., & Klan, F. (2022). Gaussian processes for one-class and binary classification of crisis-related tweets. In *ISCRAM*, 664–673.

[14] Ketkar, N., Moolayil, J., Ketkar, N., & Moolayil, J. (2021). Convolutional neural networks. *Deep Learning with Python: Learn Best Practices of Deep Learning Models with PyTorch*, 197–242.

[15] Masci, J., Meier, U., Ciresan, D., Schmidhuber, J., & Fricout, G. (2012). Steel defect classification with max-pooling convolutional neural networks. In *The 2012 International Joint Conference on Neural Networks*, 1–6. https://doi.org/10.1109/IJCNN.2012.6252468

[16] Kulkarni, A., Chong, D., & Batarseh, F. A. (2020). Foundations of data imbalance and solutions for a data democracy. In Batarseh, F. A. and Yang, R. X. (Eds.), *Data Democracy* (pp. 83–106). Academic Press. https://doi.org/10.1016/B978-0-12-818366-3.00005-8

[17] Louhenkilpi, S. (2024). Continuous casting of steel. In Seetharaman, S. (Ed.), *Treatise on Process Metallurgy* (pp. 343–383). Elsevier.

[18] Mahesh, B. (2020). Machine learning algorithms – A review. *International Journal of Science and Research*, *9*(1), 381–386.

[19] Petry, J. (2020). Deep learning expands into assembly, packaging, kitting onspection. *Quality*, 8VS–10VS.

[20] Völz, B., Behrendt, K., Mielenz, H., Gilitschenski, I., Siegwart,R., & Nieto, J. (2016). A data-driven approach for pedestrian intention estimation. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems*,2607–2612. https://doi.org/10.1109/ITSC.2016.7795975

[21] Mery, D. (2020). Aluminum casting inspection using deep learning: A method based on convolutional neural networks. *Journal of Nondestructive Evaluation*, *39*(1), 12.

[22] Patil, A., & Rane, M. (2021). Convolutional neural networks: An overview and its applications in pattern recognition. In *Information and Communication Technology for Intelligent Systems: Proceedings of ICTIS 2020*, *1*, 21–30.

[23] Tabernik, D., Šela, S., Skvarč, J., & Skočaj, D. (2020). Segmentation-based deep-learning approach for surface-defect detection. *Journal of Intelligent Manufacturing*, *31*(3),759–776. https://doi.org/10.1007/s10845-019-01476-x

[24] Nayak, D. R., Das, D., Dash, R., Majhi, S., & Majhi, B. (2020). Deep extreme learning machine with leaky rectified linear unit for multiclass classification of pathological brain images. *Multimedia Tools and Applications*, *79*, 15381–15396.

[25] Racki, D., Tomazevic, D., & Skocaj, D. (2018). A compact convolutional neural network for textured surface anomaly detection. In *2018 IEEE Winter Conference on Applications of Computer Vision*, 1331–1339. https://doi.org/10.1109/WACV.2018.00150

[26] Wang, Z. J., Turko, R., Shaikh, O., Park, H., Das, N., Hohman,F., . . . , & Chau, D. H. P. (2020). CNN explainer: Learning convolutional neural networks with interactive visualization. *IEEE Transactions on Visualization and Computer Graphics*,*27*(2), 1396–1406.

[27] Yathish, V. (2022). *Loss functions and their use in neural networks. Towards Data Science*. Retrieved from: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9

How to Cite: Dikici, S., & Robinson, R. J. (2024). Automated Defect Detection Using Image Recognition in Manufacturing. *Journal of Data Science and Intelligent Systems*. https://doi.org/10.47852/bonviewJDSIS42023833