JOURNAL of
MATERIALS and MECHATRONICS:A

e-ISSN 2717-8811
JournalMM, 2024, 5(2), 263-285
https://doi.org/10.55546/jmm.1512549

Araştırma Makalesi / Research Article

Deep Learning Application and Analysis In Detection of Metal Plate Surface Defects

Can TUNCER[1*], Cemil KÖZKURT[2], Serhat KILIÇARSLAN[3]

[1*] GESBEY R&D Center, Balıkesir, Türkiye,
ORCID ID: https://orcid.org/0000-0003-0539-1381, can.tuncer@gri.com.es
[2] Bandırma Onyedi Eylül University, Faculty of Engineering and Natural Sciences, Department of Computer Engineering, Bandırma, Balıkesir, Türkiye,
ORCID ID: https://orcid.org/0000-0003-1407-9867, ckozkurt@bandirma.edu.tr
[3] Bandırma Onyedi Eylül University, Faculty of Engineering and Natural Sciences, Department of Software Engineering, Bandırma, Balıkesir, Türkiye,
ORCID ID: https://orcid.org/0000-0001-9483-4425, skilicarslan@bandirma.edu.tr

Geliş/ Received: 09.07.2024; **Revize/Revised:** 26.09.2024 **Kabul / Accepted:** 18.11.2024

ABSTRACT: In industrial manufacturing processes, detection of defects on the surfaces of metal plates supplied from iron and steel main industry manufacturers to be processed by machining and non-machining methods has an important place in estimating the values of the relevant plate such as safety and maintenance cost. With the developing technology and computer vision and deep learning applications finding a place in the industry, it has become possible to detect and classify metal plate surface defects more quickly and effectively with a lower error rate at an advanced technological level. Within the scope of this study, a deep learning model was created by using the TensorFlow library in the Python environment with using NEU Metal Surface Defects Dataset to detect metal plate surface defects. Then as an industrial application, a device prototype developed using Nvidia Jetson Nano and USB Camera, in order to test this model under real conditions.

Keywords: Metal plate, Surface defect, Deep learning, Computer vision, Artificial intelligence, Machine learning

*Sorumlu yazar / Corresponding author: can.tuncer@gri.com.es
Bu makaleye atıf yapmak için /To cite this article

Tuncer, C., Közkurt, C., Kılıçarslan S. (2024). Deep Learning Application and Analysis in Detection of Metal Plate Surface Defects. Journal of Materials and Mechatronics: A (JournalMM), 5(2), 263-285.

1. INTRODUCTION

Metal plates are one of the materials used as the main construction material in sectors such as machinery, automotive, shipbuilding, energy, and home electronics. Metal plates are shaped by various machining or chipless manufacturing methods such as cutting, bending, welded joining and used in structures within the framework of certain design and engineering decisions. There are many standards like EN ISO 8501-1:2007, EN 10130, ASTM A480/A480M, ASTM E45 that must be followed for the metal plates used to meet sufficient quality criterias. Compliance with the standards in which the basic physical and chemical properties of the material, surface roughness and quantity, effect and acceptance values of surface defects are defined emerge as the basic quality requirement of the enterprises.

One of the main problems seen in metal plates is the surface defects that have occurred on the plate surface. These defects may be caused by variables determined in various processes during the manufacturing phase of metal plates, as well as by different external factors such as environmental conditions, transportation, stacking. In this context, one of the quality control processes applied in enterprises is the inspection of surface defects of metal plates, and the rapid and accurate detection of these defects stands out as a critical condition for providing the necessary strength conditions in industrial production and increasing product quality. Within the scope of the control process, it can be decided to repair the metal plate or to discard it as a result of evaluations such as determining the presence of surface defects and their location on the plate, classifying them, determining their quantity, measuring the depth or height differences caused by the defect with various devices, determining the presence rate of the defects in the entire area and determining whether they are within tolerance. These defects have been described in the literature and generally classified in 6 main groups as crazing, inclusion, patches, pitted, rolled, scratches (Fu et al., 2019).

Different methods can be preferred to detect surface defects. The most basic, simple, fast and easy-to-apply method is visual control. Although visual controls are carried out by experienced operators, it is obvious that it is a method that is quite open to subjective errors related to humans. In this method, it is possible to experience negativities such as overlooking, neglecting, and misclassification of critical defects. Overlooked intolerance surface defects can cause negativities that may threaten human life, such as thinning of the wall, notch effect or the risk of corrosion formation.

Another method used for the detection of metal plate surface defects is the applications developed by using computer vision and deep learning models, which are among the sub-branches of artificial intelligence studies that are developing and becoming widespread today. This method can provide the detection of surface defects with much higher reliability and practicality than traditional methods. In this study, it was aimed to develop a deep learning model based on Convolutional Neural Network (CNN) for the detection of metal plate surface defects by deep learning. Convolutional Neural Networks are a subclass of deep learning models designed specifically to work on visual data. CNN's are highly effective in areas such as image processing, video analysis, and medical imaging, and are often used successfully in tasks such as object recognition, facial recognition, automated vehicle driving systems (Mo et al., 2018). It will be ensured that the model to be developed with the study will be tested by using a large number of classified image data of metal surface defects in a data set. It is aimed to run the resulting model file on a mobile minicomputer and to detect and classify metal plate surface defects by detecting live images under real conditions with the camera connected to the computer.

In the "Introduction" section, the reasons for the initiation of the study are presented, and in the "material, method" section, the methods applied during the execution of the study are discussed in detail. The fundamentals of artificial intelligence and deep learning have been examined, and concepts such as libraries, layers, activation functions, optimizers, which are the sub-elements necessary for the construction of deep learning architecture using artificial neural networks, have been clarified in detail. In the "experimental studies" section, the details of how the deep learning model architecture is created, the training process of the model and the industrial application stages are revealed. In the "results and discussion" section, results and the value of the findings obtained was discussed and future studies were mentioned. Finally in the “conclusions” section, a summary of the results of the study is shared.

2. MATERIALS AND METHODS

A common feature of the case studies discussed to shape the methodology of the study in the preliminary research process is that they are based on the creation of a certain data set or its use by providing it ready-made. Since data collection specific to metal plate surface defects will require a long period of time on a yearly basis in order to encounter a sufficient number of defects to allow a healthy deep learning process to be carried out under the types of defects that occur, it was decided to bypass the data collection process, and instead of it, the open access NEU Metal Plate Surface Defects dataset (Song and Yan, 2019) is decided to be used.

The necessary codes to work on the Visual Studio Code application were created and run in the Python programming language for processing the data in this dataset, setting up the deep learning model, carrying out the training process, obtaining the performance values and to test the results practically. This program consists of 6 main parts: preparation, data preprocessing, model creation and training, evaluation of training performance, testing of the model and finally exporting the model.

To use the necessary libraries and activation functions during the preparation phase, it is ensured that they are called to the program and the locations of the training, test and verification clusters are specified. In the data pre-processing phase, data augmentation methods were applied by subjecting the image data in the data set to operations such as rotation, scrolling, mirroring, scaling, resizing, categorical classification. In the model creation and training phase, definitions were made about the use of the TensorFlow / Keras library, layer parameters were determined, the metrics to be used were selected, and the training process of the CNN model, which was developed by determining the number of cycles and iterations, was carried out.

After the completion of the training, the necessary code pieces were added to create the accuracy and loss functions to reveal how successful the training process was. The model is exported from the compiler so that it can run on a computer-independent device. In this way, it has become possible to test the developed deep learning model in real industrial conditions using a minicomputer with a camera integrated with high image processing capacity.

2.1 Artificial Intelligence

Artificial intelligence is a branch of science and engineering that aims to enable computer systems to gain human-like capabilities. It is the set of abilities that computers gain by using algorithms, data, and computational power to enable them to perform tasks associated with human intelligence, such as visual perception, speech recognition, learning, decision-making, language understanding and problem solving. Artificial intelligence applications, which are divided into many different sub-branches such as machine learning, deep learning, natural language processing, expert systems, artificial neural networks and genetic algorithms, have become widely used in areas such as engineering, health, finance, automotive, education, media and industrial production. The concept of artificial intelligence, which has occupied the thoughts of scientists such as ancient Greek and Egyptian philosophers since ancient times, was introduced in the early period of today, and the concept of "expert systems" that combine techniques such as logical thinking, symbolic calculation and problem solving to reveal human-like logical judgments and decision abilities (Turban and Watkins, 1986). With the combination of modern computer science based on the algorithmic perspective and a series of innovative ideas and discoveries based on statistical studies, the concept of artificial intelligence made further development in the mid-20th century. The Dartmouth Conference, which took place in 1956, was an important milestone for the introduction and development of the concept of artificial intelligence as a branch of science, which was officially put forward by scientists such as John McCarthy, Marvin Minsky, Nathaniel Rochester and Claude Shannon (Moor, 2006). The main goal of artificial intelligence-oriented scientific studies is that computer systems have human-like intelligence, and that they reach results and perform better than humans in many processes such as data analysis, pattern recognition, and making predictions, and as of today, an artificial intelligence structure equivalent to human intelligence has not yet been revealed and continues to develop in many ways (Shinde and Shah, 2018).

2.2 Machine Learning and Deep Learning

Machine learning (ML) is a sub-branch of artificial intelligence and refers to the ability of computers to perform certain tasks by providing learning using data. Machine learning structures, which have different structures from traditional programming logic, are structures that have the ability to create patterns by processing data and to make predictions and decisions by learning the relationships between these patterns, rather than operating based on pre-programmed rules. It is examined in 3 subcategories as supervised learning, unsupervised learning and reinforcement learning. Supervised learning is a type of machine learning that is mostly used in studies such as classification and regression, and enables the matching of input data with the right outputs by providing learning on labeled data. Unsupervised learning is based on the principle of discovering the differences by the learning model itself by applying it on unlabeled data, which is preferred in studies such as clustering and size reduction. Reinforcement learning, robotic education and game development are followed with a learning methodology based on a reward or punishment system, which is preferred. While the decision-maker is rewarded when he makes the right decisions, he provides learning by receiving negative signals in the wrong decisions.

The concept of deep learning (DL), on the other hand, is a machine learning technique performed using artificial neural networks and is generally used in studies that need to develop learning ability using large and complex data sets. There are many subtypes of deep learning, and the term "deep" in the concept of deep learning, which refers to the "concept of deep artificial neural networks", is related to the use of a large number of layers (Shinde and Shah, 2018). Deep learning is a sub-application of machine learning, and machine learning is under artificial intelligence applications, which is the widest scope. To give an example of the main areas where deep learning is successfully applied; We see that studies on applications such as image recognition and processing, natural language processing, autonomous driving, and voice recognition come to the fore. Model structures, called artificial neural networks, are mathematical models inspired by the functions of the human brain. Deep learning refers to the situation where these networks are made up of multiple layers (usually a large number of hidden layers). Thanks to these layers, deep learning models have the ability to perform an automatic feature extraction by starting from simple features on the data and progressing to more complex features (Sahu and Dash, 2021).

2.3 Key Elements for Deep Learning Application

Deep learning applications are carried out in the computer environment using various programming languages. Within the scope of these studies, researchers need to master many subjects and make the necessary preparations for applications in order to reveal an efficient code structure and to operate it effectively. In this context, the correct and appropriate selection of the programming language, compiler environment, and hardware capacity of the computer to be used is important for a successful deep learning application. In addition, a good understanding of concepts such as many libraries, layers, activation functions, metrics that should be used under the deep learning architecture, learning the working principles, and determining the relationships and parameters affecting the operation correctly are important issues for model performance. We can explain the importance of these basic elements and the reasons for their preference by summarizing them as follows.

Today, Python is the most widely used programming language in similar studies, and it was preferred to take this language as a basis in our study. In order to create a deep learning architecture on Python, it is necessary to first select a computer, operating system and compiler (IDE) to run this programming language on (Raschka and Mirjalili, 2019).

In our study, it was possible to carry out code development experiments in different environments and compilers by using 2 different computers with Windows and Linux operating systems, Visual Studio Code, Kaggle and Google Colab compilers, and the Python program directly from the terminal on a Linux-based computer. In the deep learning architecture we are working on, it is aimed to use the Keras API (API, Application Programming Interface) and sub-APIs under the TensorFlow library. TensorFlow is an open-source machine learning library developed by the Google Brain team and published by Google. It is designed for complex data processing tasks and has been shown to be particularly effective in large-scale deep learning applications. Its main features include flexibility due to its support for various computational operations and its ability to run on different platforms, the ability to perform calculations in parallel and efficiently by converting operations into a structured graph using a data flow graph, and the ability to automatically adjust model parameters using optimization techniques such as gradient descent with automatic derivative calculation (Helms et al., 2018).

Keras; TensorFlow is a high-level neural network API that runs on core libraries such as Theano and CNTK, offers a user-friendly and modular structure, enabling rapid prototyping of deep learning models (Manaswi, 2018). Thanks to Keras, which is integrated into TensorFlow, the application becomes user-friendly and provides practicality to practitioners.

Groups with different features formed by artificial neural networks in a deep learning architecture can be defined as "layers". Deep learning models are made up of several layers of compute that process data from the input layer to the output layer. Layers act as filtering structures designed to learn a specific set of features from data. The characteristics of the layers used in the study and why they are preferred are explained in detail in the following sections. The types of layers used in deep learning studies are generally classified as input layers, hidden layers, and output layers. The neurons in each layer receive and process the incoming signals and transmit them to the next one. With these complex interactions between layers, the systematic working function of deep learning structures takes place (LeCun et al., 2015).

The mathematical expressions used to calculate the output of each neuron in the network are called activation functions. Non-linear activation functions (e.g., ReLU, sigmoid, tanh) allow the model to learn nonlinear and more complex relationships. It is possible to process the data coming to a neuron in the artificial neural network by subjecting it to the mathematical process defined by the specified activation function and to transform it as a different element for the next stage as the output of the neuron. The mathematical expression of the activation function and the way it works is very important for the performance of deep learning architecture. Instead of applying the trial-and-error method for the selection of the right activation function for the deep learning architecture built for each different purpose, choosing inspired by case studies will speed up the process. The activation functions used in the study are discussed in detail in the relevant section (Rasamoelina et al., 2020).

Each neuron contains weights and biases that are applied to the input data. During the training of the network, these parameters are adjusted to provide the best performance over the data. Bias is a constant added to the activation function that the neuron will process, and it is useful to get results in different ranges by changing the function curve. Weight, on the other hand, is a numerical element of information that is processed in one neuron and transferred to another neuron to reveal a feature. The weights randomly selected at the beginning of the training process are optimized by the learning algorithm in accordance with the data set during the training process (Dung and Mizukawa, 2007).

The loss function is used to evaluate model performance during the training of the model, to be an indicator of the optimization of parametric settings, and to optimize algorithms. It is used to measure how close their predictions are to the actual data. Designing the training process to minimize this loss is essential for a successful model training. There are varieties such as Mean Squared Error (MSE), Mean Absolute Error (MAE), Cross Entropy Loss, Hinge Loss, Poisson Loss, Huber Loss, each of which can be expressed mathematically separately. Details about the loss function used in the study are given in the relevant section (Wang et al., 2022).

Optimization algorithms are used to improve the training process of the deep learning model and to obtain the best performance. It has basic tasks such as updating the model parameters, minimizing the loss function, adjusting the learning rate, preventing the problem of overfitting, and accelerating the training process.

Different types of algorithms such as Stochastic Gradient Descent (SGD), Momentum, Adagrad, RMSProp (Root Mean Square Propagation), Adam (Adaptive Moment Estimation), Adadelta, Nadam (Nesterov-accelerated Adaptive Moment Estimation) are used, and in our study, it was decided to use the "Adam" optimizer due to its performance in similar studies. The relevant section provides the necessary explanations about the Adam optimizer (Haji and Abdulazeez, 2021).

3. EXPERIMENTAL STUDY

The establishment of deep learning architecture requires the application of theoretical knowledge in the field of practical experience, that is, the introduction of coding work. For this, it is essential to run the code by working on a Python compiler. During this coding study, it is possible to construct different architectures, to achieve high or low performance results by using different libraries or layers with different parameters, or to reveal application details. In addition, it is possible to optimize the accuracy and lost output values, which are the training result metrics, by trying different parametric values during the construction of the model.

3.1 Dataset

It is the dataset that contains the data on which the deep learning model will apply the feature layers. Depending on the type of deep learning study to be conducted, this data may include different types of data such as audio, image, numbers, lists, video footage, etc. In addition, the data files that make up the data set should consist of the same type of data and should have forms and features ready to be used in the training process. The dataset we will work on is the dataset containing image files of a large number of metal plate surface defects, which have been shared as open source under the name of "NEU Metal Plate Surface Defects Dataset" (Song and Yan, 2019). Surface defect image files in 6 different categories are distributed in folders with different names and divided into training, test and verification sets. Each image file is a bitmap (BMP) file with a size of 200x200 pixels and a color depth of 8 bits. Thanks to Keras' ImageDataGenerator sub-API, there is no need for labeling for images, and tags can be matched automatically thanks to the nested folder structure (Lv et al., 2020).

3.2 Preparation and Data Pre-Processing

It is ensured that the libraries and layers to be used in coding are transferred to the program, called (imported), and the address definitions of the location where the training, test and verification sets are located in the computer environment are made at this stage. Importing basic Python libraries such as NumPy, Pandas, Matplotlib and TensorFlow into the program is essential to reveal the functions required in the working process of the Python program. At this point, in order to facilitate the operation of the program, instead of importing the entire large-sized library, it is generally preferred to import the small cube sub-modules needed in the code group. Thus, it is possible for the program to run faster with lower dimensional data.

At this stage, a data generator is created for the training dataset. With this generator, various data boosting techniques have been applied on the images, such as scaling, random rotation, random panning in width and height, and horizontal flipping. With these techniques, it is possible to generalize the model and reduce the problem of overfitting. Scaling is also implemented for test and validation datasets through the creation of a data generator. In addition, the creation of image streams from data generators is ensured (Maharana et al., 2022).

3.3 Setting Up the Model

Tensorflow and Keras libraries and sub-APIs and sub-APIs and layers, such as Sequential, Conv2D, MaxPooling2D, Activation, Flatten, Dense, Dropout, were imported at this stage and parametric definitions were made (Sobhana et al., 2023). Figure 1 shows an example deep learning architecture image that matches the structure in our original model.

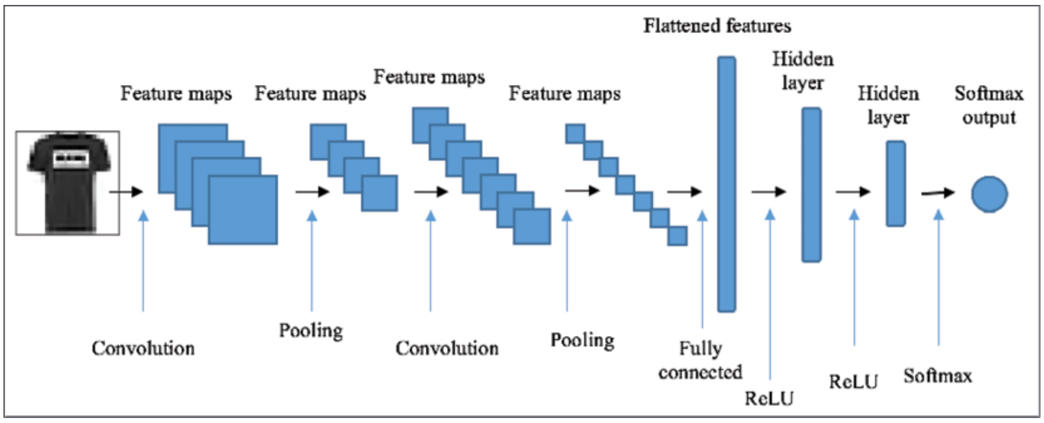


Figure 1. A representative structure of deep learning architecture (Bbouzidi et al., 2024)

Sequential is a class used to build models in Keras, and it is a library that allows layers to be ordered in relation to each other (Gulli and Pal, 2017).

The Functional model structure, which is another type of Sequential, can provide multiple inputs and outputs and enables a more complex architectural setup. In our study, it was preferred to use Sequential architectural structure. In Figure 2, ordered and functional API structures are shown schematically.

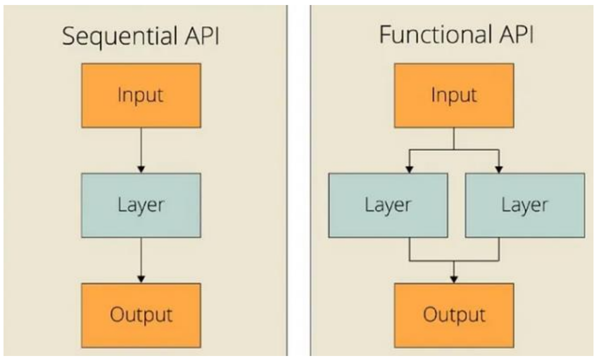


Figure 2. Algorithmic expression of the Sequential and Functional API difference (Vansh, 2022)

Conv2D is the convolution layer used to extract features from such data that has been shown to be successful on 2D data, such as image data. A representative comparison of the input data and the convolution kernel is presented in Figure 3.

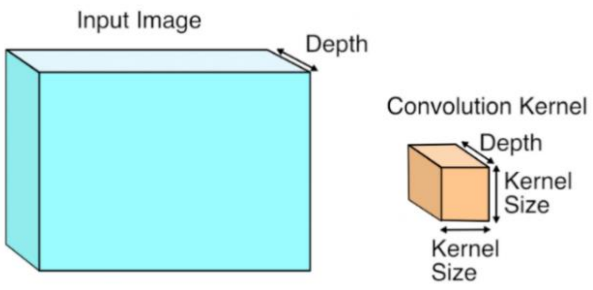


Figure 3. Representative expression of the Conv2D convolution layer (Warden, 2015)

MaxPooling2D is used in the architectural structure of deep learning models, it is the pooling layer that usually follows the convolution layers. Thanks to this layer, it is possible to reduce feature maps, while preserving important features and reducing data size (Mastromichalakis, 2023). In Figure 4, the effect of applying Max Pooling is expressed on the image.

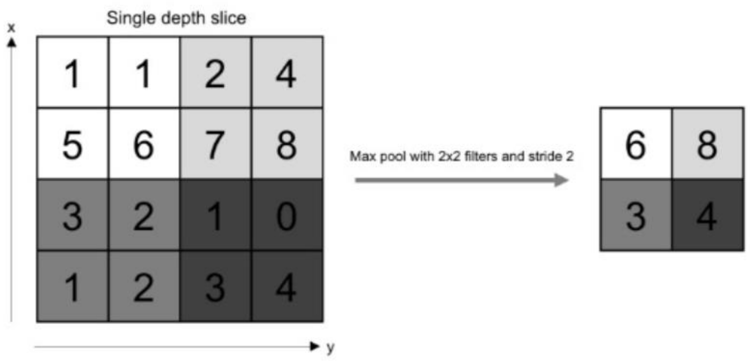


Figure 4. Example demonstration of the effect of the MaxPooling2D digestion layer

Flatten is the layer used for converting multidimensional feature maps into a one-dimensional vector. Preparation is made for the processing of the data in the "Dense" layer by providing a size reduction transformation (Jin et al., 2015). The effects of the flattening layer are expressed in the image given in Figure 5.

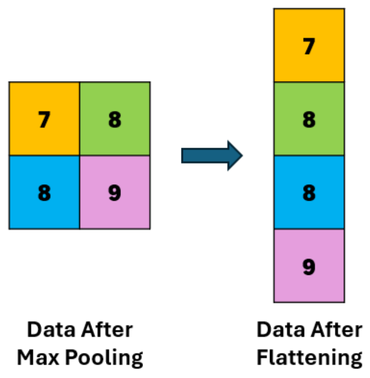


Figure 5. Example illustration of the size reduction transformation of the flatten layer

Dense is the layer in which all the neurons in the previous layer are fully connected to each other. In this layer, parametric transformations are performed that express the complexity of a data set and help the model learn. The process of reducing the input data to a certain size and producing various feature outputs by using the representation of this data is carried out in this layer. Dense layers are commonly used in many tasks such as classification, regression, language processing, and so on, and they play an important role within the architecture to increase the complexity of the model and learn patterns in the data set. The image in Figure 6 schematically expresses the dense layer.

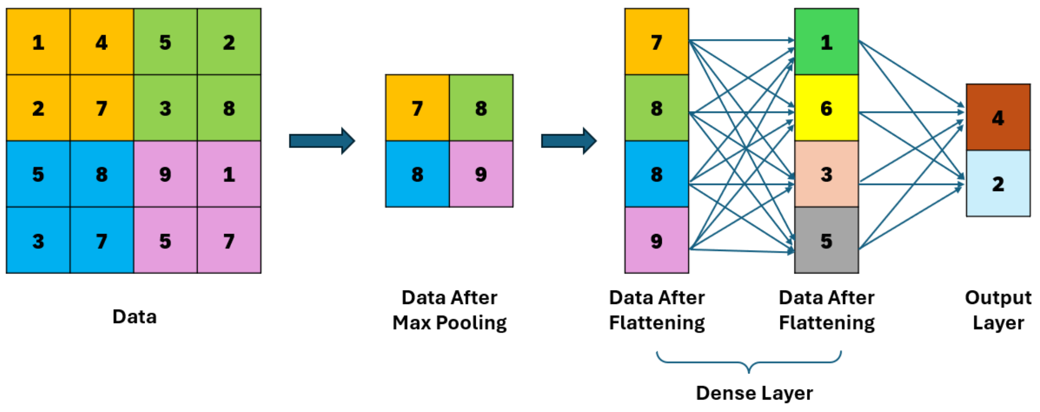


Figure 6. The location of dense layers in the order of layers

The so-called overfitting or overlearning condition refers to the situation in which a model loses its ability to make predictions in real-world data due to over-adaptation to training data and occurs when the model overreacts to noise or randomness in the training data. Underfitting or underlearning, on the other hand, means that the model fails by failing to catch even the basic patterns due to its simplicity and failing to show the expected performance (Rice et al., 2020).

In Figure 7, the under-fitting, right-fit and over-fitting situations encountered in classification studies are shown on the graph depending on the time.

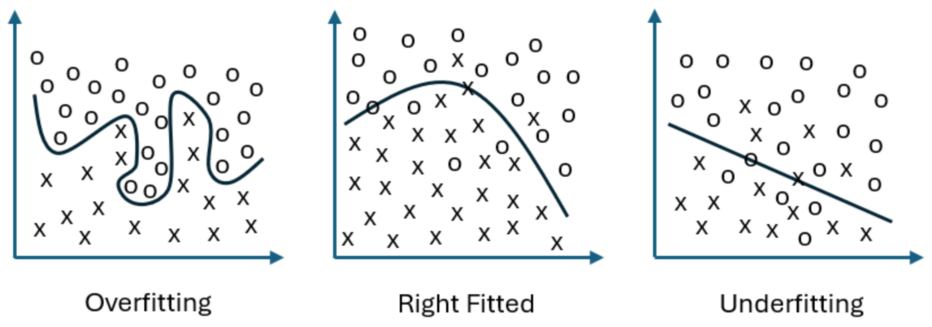


Figure 7. Representation of overfit, right-fit, underfit graphical representation

Dropout is a layer used to ensure that the model is resistant to overfitting. Overfitting, right fitting and underfitting situations can be seen in Figure 7. This layer is based on the principle of covering a certain proportion of neurons randomly determined in each iteration during the training of the model, that is, stopping their activity. Thanks to the neurons that remain dysfunctional, it is possible for the remaining network elements to gain the ability to learn more independently (Baldi and Sadowski, 2013).

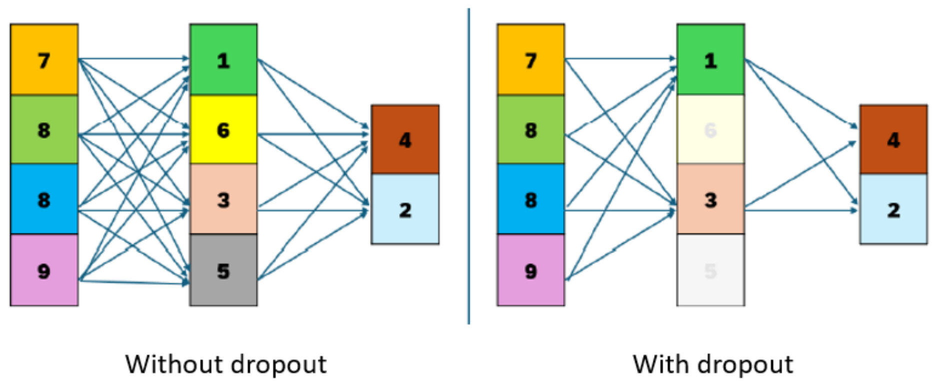


Figure 8. Representative expression of with and without dropout layers

EfficientNetB7 is a ready-to-use model developed by the Google Brain team for use in computer vision studies, which has been previously trained on the ImageNet data set and allows large data sets consisting of complex images to be processed in a short time with high performance. For this reason, the use of this model was preferred in the study (Helms et al., 2018).

Relu is an abbreviation of "Rectified Linear Unit" and is a popular activation function used in deep learning models. This function checks whether the value it receives as input is greater than zero, leaves the value as it is if it is greater than zero, and works to define it as zero if the value is less than zero. Its mathematical equation (1) and graphical representation are as follows (Agarwal et al., 2021).

$$ReLU(x) = max(0, x) \tag{1}$$

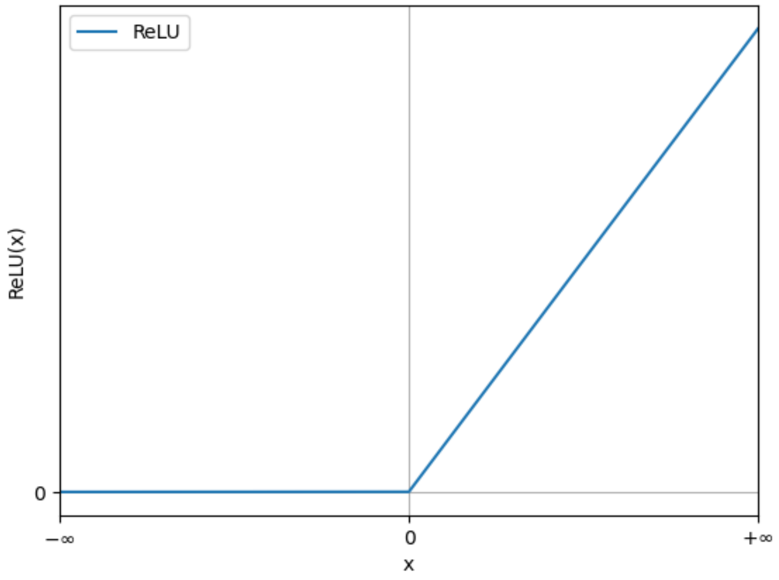
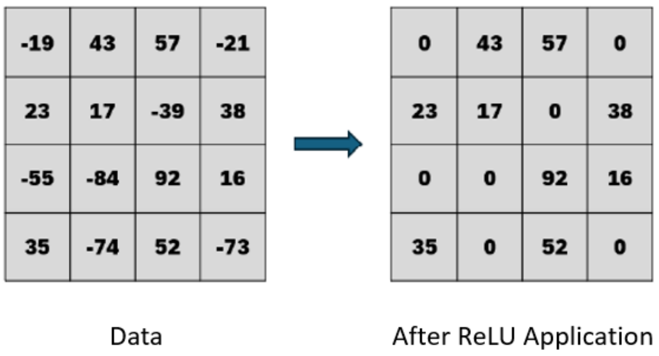


Figure 9. Graph of the ReLU activation function

This function helps to speed up the training process of neural networks. Since the derivative is 1 for values greater than zero, gradients are propagated quickly and efficiently. For values less than zero, it is possible for the model to work more efficiently by causing some neurons to remain dysfunctional and closed since the derivative is zero.



Data

-19	43	57	-21
23	17	-39	38
-55	-84	92	16
35	-74	52	-73

After ReLU Application

0	43	57	0
23	17	0	38
0	0	92	16
35	0	52	0

Figure 10. Transformed matrix with the ReLU activation function applied

Softmax function is a mathematical function that takes a set of real numbers, converts them to values between 0 and 1, and normalizes them by equalizing the sum of all outputs to 1. With the Softmax function, which is frequently used in the output layer in multi-class classification problems, the probability distribution for each class is revealed. The Softmax function works with the logic of normalization by starting by transforming each element with an exponential function and dividing it by the sum of these transformations. Mathematically, the Softmax i-th element for a vector "z" is calculated by the following equation (2) (M. Wang et al., 2018).

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{2}$$

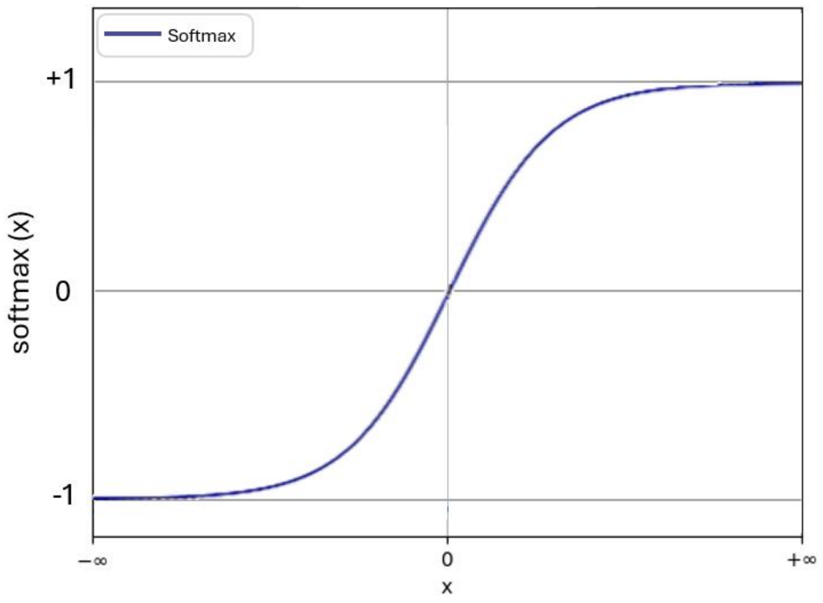


Figure 11. Graphical representation of the Softmax function

Where z is the input vector, e is Euler's number (about 2.718), zi is the i-th element of the input vector, and addition is done over all the elements in the vector. In addition to modeling the probability of each class, this function is also used to ensure that the model's predictions are more stable and balanced (Kılıçarslan et al., 2021).

3.4 Optimization and Compilation

In the optimization and compilation phase, which is one of the processes that should be applied before model training, the optimizer must be defined first. The optimizer of choice in the study is the Adam optimizer. Adam optimizer is an effective stochastic gradient descent algorithm that is widely used in the fields of machine learning and deep learning. The name "Adam" is an abbreviation for "Adaptive Moment Estimation". This optimizer provides adaptive learning rates for each parameter by calculating the unique learning speeds of each parameter. This feature allows different parameters to be updated more evenly and effectively, and often helps to achieve faster and more efficient results than different optimization algorithms. One of the main advantages of the Adam optimizer is that it uses both the first moment (average) and the second moment (variance) values, as shown in the following equations (3,4,5,6). Thus, it is possible for the algorithm to dynamically adjust the learning speed according to the size of the gradients.

$$v_t = \beta_1 * v_{t-1} - (1 - \beta_1) * \ g_t \tag{3}$$

$$s_t = \beta_2 * s_{t-1} - (1 - \beta_2) * \ g_t^2 \tag{4}$$

$$\Delta\omega_t = -\eta \frac{v_t}{\sqrt{s_t + \epsilon}} * \ g_t \tag{5}$$

$$\omega_{t+1} = \omega_t + \ \Delta\omega_t \tag{6}$$

v_t refers to the moving exponential average of time-varying gradients. s_t stands for the exponential moving average of the square of the gradients. β_1, β_2 are the exponential deceleration rates for the first and second moment estimates of gradients. Values can be taken as 0.9 for β_1 and

0.999 for β_2. η (Learning Rate) refers to the speed of the learning process. This value is typically set with a low initial value (for example, 0.001). g_t refers to the gradient of the parameters that the algorithm tries to optimize at time t.

Adam uses learning rates that can be individually adapted for each parameter. In each training step, the first moment estimate of the gradients (moving average of the gradients) and the second moment estimate (the moving average of the squares of the gradients) are calculated first. Subsequently, these calculated values are initially used to update the parameters by correcting the gradients and gradient squares with bias-correlation steps against low estimates (Bock and Weiß, 2019).

3.5 Metrics

In this study, the categorical cross-entropy function, which is highly preferred in multiclass classification problems, is used as the loss function. This function (7,8) is used to maximize the agreement between the probability distributions predicted by the model and the actual labels. In Figure 12, this issue is schematized.

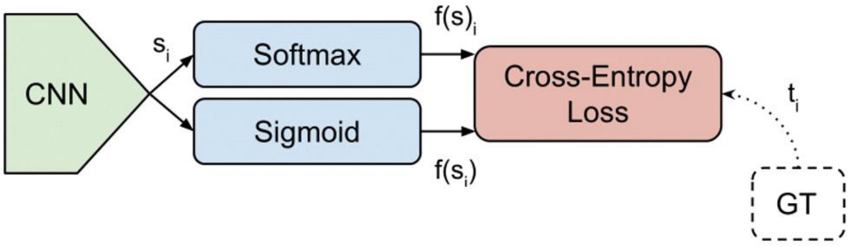


Figure 12. Loss function expression in CNN architecture (Swasthik, 2020)

$$CE = -\sum_{i}^{C} t_i \log(f(s)_i) \tag{7}$$

$$CE = -\sum_{i=1}^{C'=2} t_i \log(f(s_i)) = -t_1 \log(f(s_1)) - (1 - t_1) \log(1 - f(s_1)) \tag{8}$$

c Represents the number of classes. This is the number of different categories that the model tries to predict. ti, represents the actual labels. These values are usually 0 or 1; If it belongs to a class, it is encoded as 1, if not, it is encoded as 0. f(si) refers to the probabilities that the model predicts. log is a logarithm function. In cross-entropy loss, the logarithm of the estimated probabilities is taken. Logarithms play an important role in measuring the accuracy of the predictions made by the model; Correct predictions reduce loss, while incorrect predictions increase loss (Barz and Denzler, 2020).

In the study, accuracy metric was used to evaluate the performance of the model. Accuracy indicates the ratio of the model's correct predictions to total predictions and is often considered a measure of success in classification problems. Accuracy ratio can be calculated with the equation (9) below accorting to values determined in the matrix (Figure 13).

Confusion Matrix		Predicted	
		0	1
Actual	0	TN (True Negative)	FP (False Positive)
	1	FN (False Negative)	TP (True Positive)

Figure 13. Confusion matrix

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{9}$$

The table above is a confusion matrix used to evaluate the performance of the classification model. The confusion matrix is used to show the classification success of the model in detail by comparing the predictions of the model with the actual values. The terms used here are; TP (True Positive) is the number of cases that the model predicts as positive and that are actually positive. That is, it refers to the number of positive cases that are correctly classified. FP (False Positive) is the number of cases that the model predicted as positive but were actually negative. This can also be referred to as a false alarm. FN (False Negative) is the number of cases that the model predicted as negative but were actually positive. This can also be considered a missed opportunity. TN (True Negative) is the number of cases that the model predicts as negative and that are actually negative. That is, it is the negative cases that are correctly classified (Maxwell et al., 2021).

3.6 Model Training

After the optimization process, the training process of the model is started within the scope of a certain number of batches and epochs. The cycle can be schematized as seen in Figure 14.

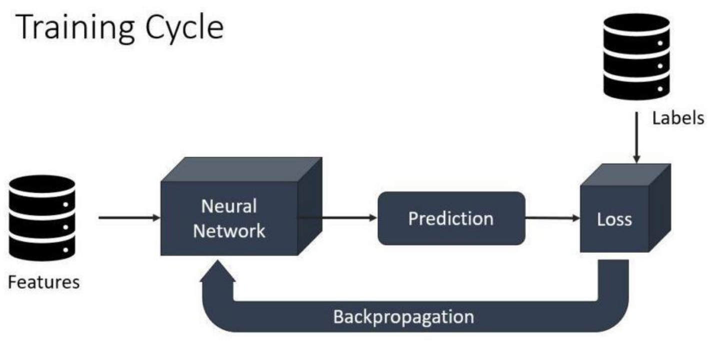


Figure 14. Representation of the model training process in the form of a flow diagram (Glasmacher, 2022)

Batch size specifies the number of instances that the model will use for training in each iteration. The batch_size parameter allows the model to make its updates more frequently, helping the training process go faster. The epoch number, on the other hand, indicates how many full cycles the training process will be run. For example, if 30 epochs are specified in the model, it means that the training data set will be processed by the model 30 times from start to finish. Figure 15 schematizes the iterations within 1 epoch.

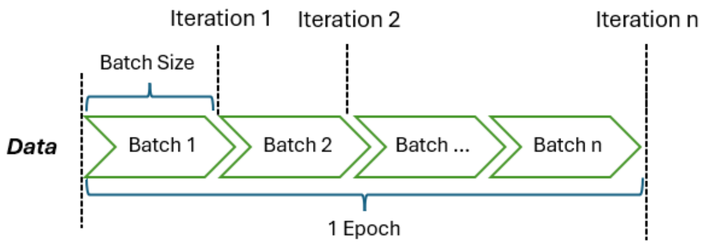


Figure 15. Formal expression of the concepts of Batch and Epoch

The duration of this process can take quite a long time, in the order of hours or days, depending on the complexity of the model and data set, and the performance of the computer hardware (Smith et al., 2018).

3.7 Accuracy and Loss Values, Confusion Matrix

The success indicator of a deep learning model can be considered as the convergence of the accuracy rate to 1 and the convergence of the loss rate to 0 as the number of iterations progresses in the training process. It is important to record the accuracy and loss values obtained during the iterations and to graph them at the end of the training in order to visualize the model performance (Figure 16, 17). The loss and accuracy values reached as of the last iteration emerged as follows for the model we created and trained within the scope of our study.

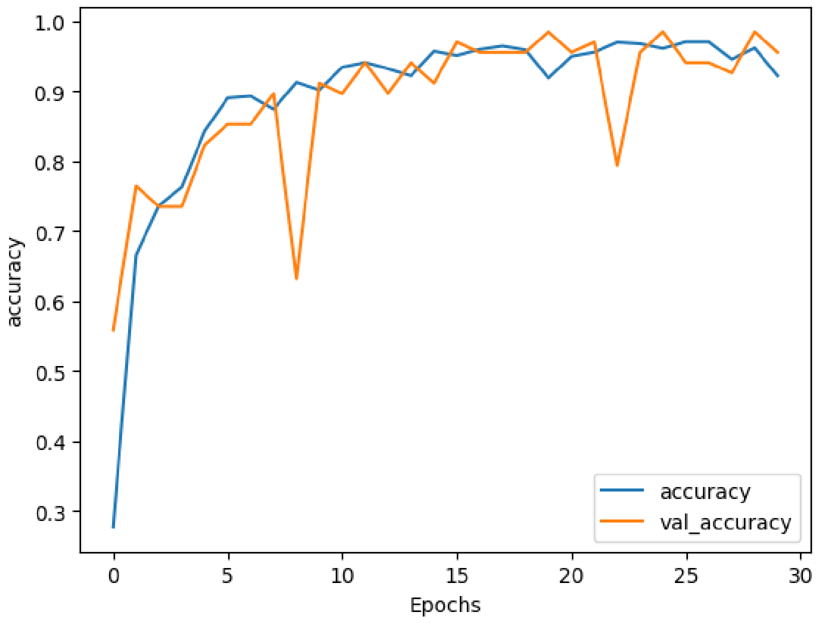


Figure 16. “Accuracy – Epoch” graph created at the end of the training

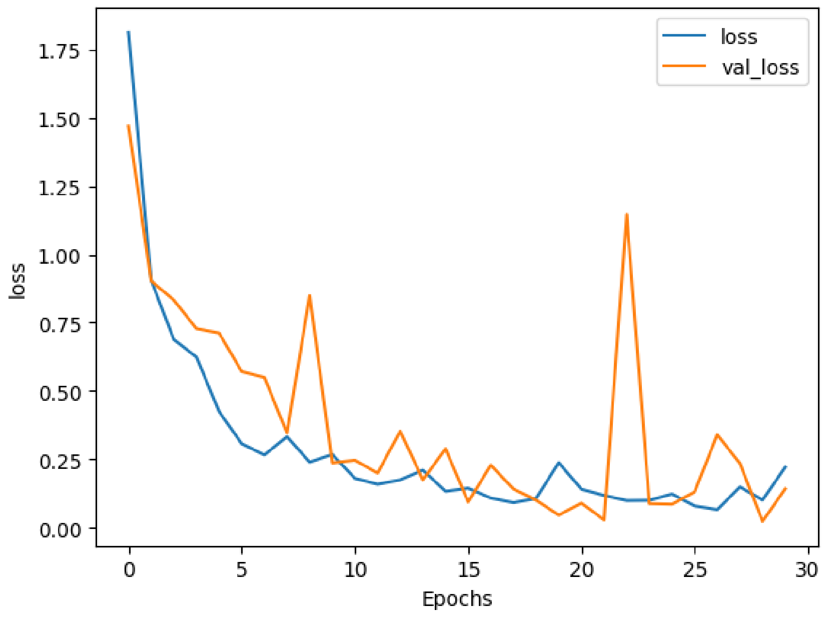
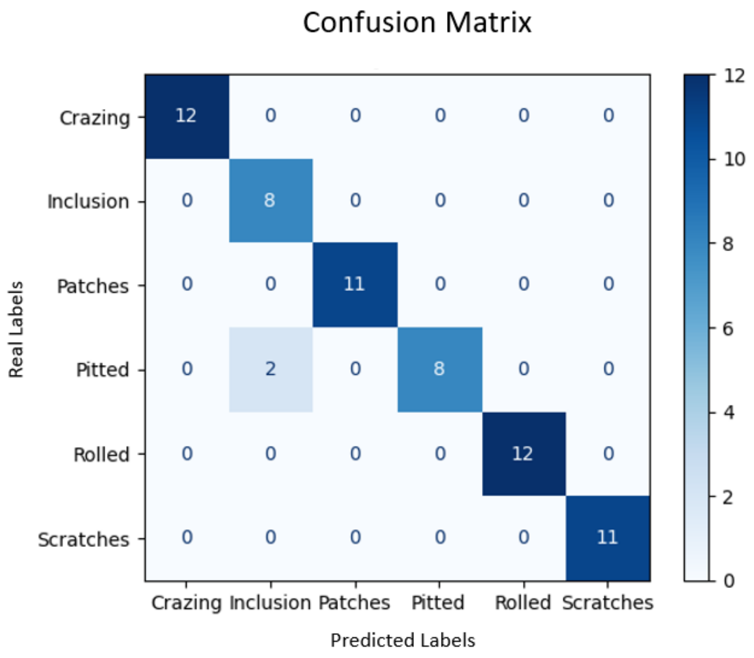


Figure 17. "Loss – Epoch" graph created at the end of the training

As a result of the training of the model, a confusion matrix was Created. The actual label values and the numbers of the estimated impact values were presented in the confusion matrix shown in Figure 18.



Real \ Predicted	Crazing	Inclusion	Patches	Pitted	Rolled	Scratches
Crazing	12	0	0	0	0	0
Inclusion	0	8	0	0	0	0
Patches	0	0	11	0	0	0
Pitted	0	2	0	8	0	0
Rolled	0	0	0	0	12	0
Scratches	0	0	0	0	0	11

Figure 18. Confusion Matrix obtained after model training

3.8 Testing the Model

After the training process of the model was completed, the test process was applied. At this stage, the performance of the model on the test data set was evaluated by visualizing. First, 9 images were randomly selected from the test data set and the model was provided with prediction. The prediction results were then compared with the actual labels. The images are visualized in a 3x3 matrix layout, and actual class information is added to each along with the model's prediction. If the prediction is correct, the label is shown in blue, and if it is incorrect, the classification accuracy of the model is easily understood visually. As a result of testing whether the sample images randomly taken from the test set were correctly predicted, the following result was reached. In the first test, it was seen that all 9 images examined were correctly estimated and can be seen in Figure 19.

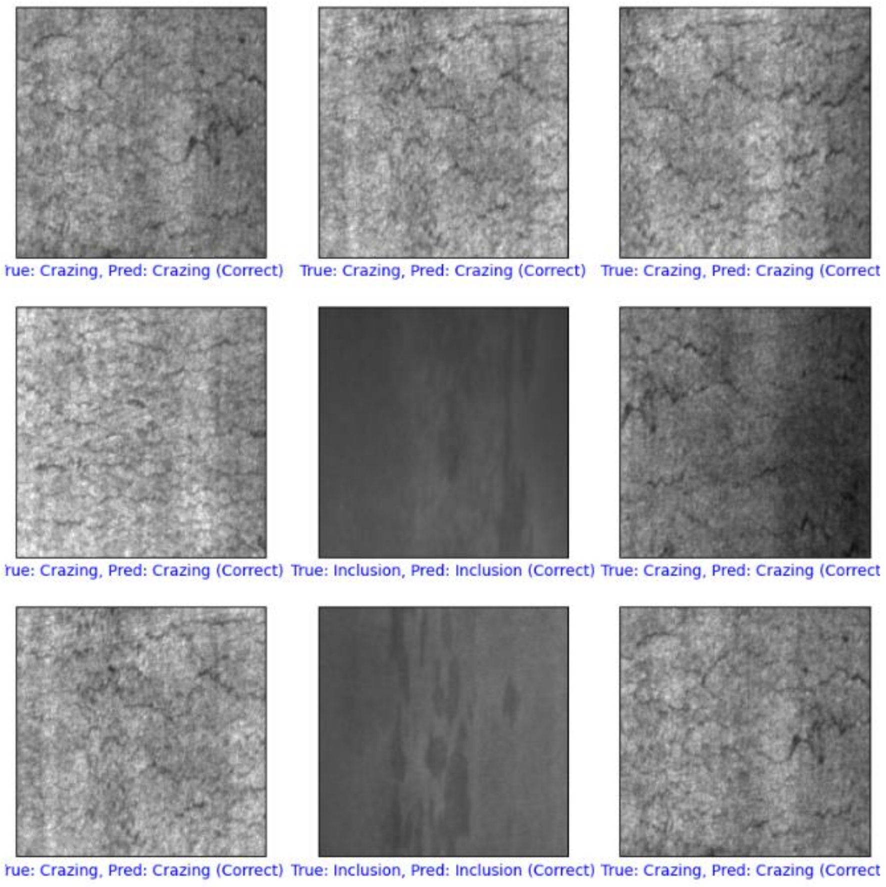

rue: Crazing, Pred: Crazing (Correct)
True: Crazing, Pred: Crazing (Correct)
True: Crazing, Pred: Crazing (Correct
rue: Crazing, Pred: Crazing (Correct)
True: Inclusion, Pred: Inclusion (Correct)
True: Crazing, Pred: Crazing (Correct
rue: Crazing, Pred: Crazing (Correct)
True: Inclusion, Pred: Inclusion (Correct)
True: Crazing, Pred: Crazing (Correct

Figure 19. Estimation of 9 randomly selected images from the test set. Incorrect predictions are expressed in red and correct predictions are expressed in blue writing, in this test all predictions are correct

In addition, the model was installed on the Nvidia Jetson Nano device and run in the python environment. This device is a developer computer with high graphics processing capability and can be used to analyze instant live video data with external cameras that can be integrated onto it. In the Linux environment, which is the operating system of the device, the model file was converted to the required format (.onnx) in order to run the relevant Python code, and the Python program was coded and run, where both the live image and how many errors were detected with how many probabilities can be read on the screen connected to the device, by performing the error classification function by detecting live images with the camera using this model file on the device (Oranen, 2021). Under real conditions, the errors on the samples were detected with a camera and tested in real time and the model was verified to work successfully as shown in Figure 20. As an example, the Patches error could be detected more than once with a performance rate of more than 90%.

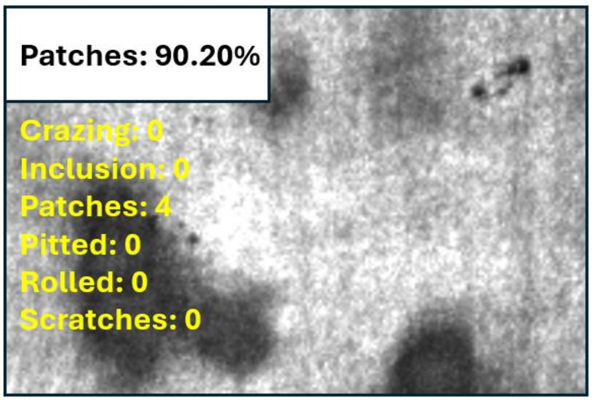


Figure 20. Testing the model with a camera

4. RESULTS AND DISCUSSION

After the completion of the training of the model, the status of the training performance was revealed by the metrics used. These metrics are accuracy and loss graphs and confusion matrix. Accuracy and loss graphs were created using the accuracy values reached at the end of each of the 30 cycles conducted in the model training. The values that make up the chart are given in the table below.

Epoch	Accuracy	Validation Accuracy (Val_Accuracy)
1	0.2111	0.5588
2	0.6227	0.7647
3	0.7344	0.7353
4	0.7618	0.7353
5	0.8407	0.8235
6	0.8917	0.8529
7	0.8878	0.8529
8	0.8999	0.8971
9	0.8958	0.6324
10	0.8746	0.9118
11	0.9257	0.8971
12	0.9470	0.9412
13	0.9261	0.8971
14	0.9306	0.9412
15	0.9572	0.9118
16	0.9607	0.9706
17	0.9455	0.9559
18	0.9722	0.9559
19	0.9550	0.9559
20	0.9471	0.9853
21	0.9403	0.9559
22	0.9568	0.9706
23	0.9616	0.7941
24	0.9664	0.9559
25	0.9564	0.9853
26	0.9598	0.9412
27	0.9769	0.9412
28	0.9489	0.9265
29	0.9644	0.9853
30	0.9159	0.9559

The tests carried out with the numerical and graphical results obtained have shown that the model creation and training work has sufficient reliability. The "Test loss" obtained using the test dataset was 0.1437525451183319, while the "Test accuracy" was 0.96875. In other words, it is understood that the success of the model has reached the level of 96.88%. The confidence interval was calculated as CI=[0.92621,1.01139] by using the test accuracy and test loss values obtained at the end of the testing of the model. A 95% confidence interval is acceptable between 0.926 and 1.0, as the confidence interval cannot be greater than 1. It is also aimed to calculate the p-value in order to support the successful result. However, in order to calculate the p-value, there must be two different groups of comparable results. Since validation results are obtained during the training process, it may be misleading to use them for calculating the p-value together with the test results. However, since we do not have any other comparison elements, it was decided to obtain the p value using validation data.

The values obtained at the end of the T-Test were "T-Statistic: -2.9503" and "P-Value: 0.0046". Since the value of 0.0046 is considerably smaller than the generally accepted significance value of 0.05, it can be said that there is a statistically significant difference between the two groups. If the T-statistic value is negative, it means that the Validation results are lower than the Test results. In other words, it is possible to say that the model exhibits a higher success in the Test set, that is, it produces a more successful result in real-world conditions that it does not encounter in the training process. On the other hand, some important parameters such as learning rate, number of batches, number of epochs, and optimizer type used in the establishment of the CNN model are factors that can affect the performance of the model. These values were selected based on the mean values used in the sample studies found in the literature, and the evaluation of the effects of different parameters on the model can be considered as a new study.

This study has many similarities and differences with similar studies in the literature. For example, in a similar study, innovative improvements were made to the Faster R-CNN algorithm, such as restructuring the network structure and using a deformed convolution network, to improve the detection of small and complex steel surface defects. In this way, the detection accuracy was increased by 12.8% to 75.2% (Zhao et al., 2021).

Another study is developing a CCVAE model that generates data for each type of defect using a Convolutional Variational Autoencoder (CVAE) to generate sufficient data for rare defects. This improves the generalization performance of DCNN-based classification, enabling defect detection with high accuracy in industrial applications (Yun et al., 2020).

In another application, a combination of improved ResNet50 and improved Faster R-CNN algorithms is proposed to enable automatic detection of steel surface defects. This method aims to overcome the limitations of traditional and existing deep learning-based methods by offering a 98.2% accuracy rate and faster uptime (Wang et al., 2021).

This model is ready for both academic and industrial use. At the end of the training process of the model, the final model file, which includes the weights and configuration properties learned, is saved out of the code as an external file. By using the weights in this file, it has become possible to make industrial-level applications using different data sets and real test samples (metal plates). In order to test the model in real conditions, it was decided to use the Nvidia Jetson Nano Developer Kit hardware to be used in the application development phase. It has been shown that these and similar devices can exhibit sufficient success for beginner and intermediate industrial applications.

It will be useful to work with a larger data set to increase the level of performance of the study. In addition, it may be possible to obtain a lower loss rate with a higher accuracy, as different results are obtained when model training is reconsidered using different layers, activation functions and parametric values. In this context, studies can be carried out on the use of different parameters to shorten the training time of the model. For example, the model can be retrained using different values for filter sizes and quantities, either individually or in different combinations. Thus, optimization can be made for model training in a shorter time with lower batch and epoch values. On the industrial application side of the study, application performance will increase when higher resolution cameras and industrial computers with higher image processing capabilities are used. The prototype devices to be developed can be equipped with many functions at the simple or advanced robotic level to record and transmit data, make statistical evaluations, perform mechanical marking or repair.

In this context, the study has been presented to the literature as a basic study in order to shed light on scientific, technological or industrial studies to be carried out in the future.

5. CONCLUSION

In this study, the developed model demonstrated sufficient reliability and robustness, confirming its readiness for both academic and industrial applications. The final model file, incorporating the learned weights and configuration properties, enables industrial-level applications with varied data sets and real test samples, such as metal plates. The use of the Nvidia Jetson Nano Developer Kit for real-world testing confirmed its capability for beginner and intermediate industrial applications. Future work with larger datasets could enhance the model's performance. Moreover, retraining the model with different layers, activation functions, and parametric values could yield lower loss rates and higher accuracy. Optimizing training time by adjusting filter sizes, quantities, batch, and epoch values is another avenue for improving the model. Industrial applications could benefit from higher resolution cameras and more powerful image processing computers. Enhanced prototype devices could incorporate functions for data recording, statistical evaluations, mechanical marking, or repairs at both simple and advanced robotic levels. This study contributes to the literature as a foundational work, providing valuable insights for future scientific, technological, and industrial research.

6. ACKNOWLEDGEMENTS

We would like to thank Bandırma Onyedi Eylül University Scientific Research Projects Unit for their support throughout our project numbered BAP 22-1010-002.

7. CONFLICT OF INTEREST

Author approve that to the best of their knowledge, there is not any conflict of interest or common interest with an institution/organization or a person that may affect the review process of the paper.

8. AUTHOR CONTRIBUTION

Can Tuncer and Cemil Közkurt contributed to the conceptualization, writing-original draft and visualation of the study. Can Tuncer, Cemil Közkurt, and Serhat Kılıçarslan contributed to the methodology and writing review and editing. Can Tuncer contributed to data curaiton and software. Cemil Közkurt contributed to the formal analysis.

9. REFERENCES

Agarwal M., Gupta S., Biswas K. K., A new Conv2D model with modified ReLU activation function for identification of disease type and severity in cucumber plant. Sustainable Computing: Informatics and Systems 30, 100473, 2021.

Baldi P., Sadowski P. J., Understanding Dropout. Advances in Neural Information Processing Systems 26, 2013.

Barz B., Denzler J., Deep Learning on Small Datasets without Pre-Training using Cosine Loss, In Proceedings of the IEEE/CVF winter conference on applications of computer vision, 2020, pp: 1371-1380.

Bbouzidi S., Hcini G., Jdey I., Drira F., Convolutional Neural Networks and Vision Transformers for Fashion MNIST Classification: A Literature Review. arXiv preprint arXiv:2406.03478, 2024.

Bock S., Weiß M., A Proof of Local Convergence for the Adam Optimizer. 2019 International Joint Conference on Neural Networks (IJCNN), July, 2019, pp: 1-8.

Dung L., Mizukawa M., A Pattern Recognition Neural Network Using Many Sets of Weights and Biases. 2007 International Symposium on Computational Intelligence in Robotics and Automation, 2007, pp: 285–290.

Glassmacher S., https://galaxyinferno.com/epochs-iterations-and-batch-size-deep-learning-basics-explained/, 2022, (24 October 2022).

Gulli A., Pal S., Deep Learning with Keras. Packt Publishing Ltd., 2017.

Haji S. H., Abdulazeez A. M., Comparison of Optimization Techniques Based on Gradient Descent Algorithm: A Review. PalArch’s Journal of Archaeology of Egypt / Egyptology 18(4), 2715-2743, 2021.

Helms M., Ault S. V., Mao G., Wang J., An Overview of Google Brain and Its Applications. Proceedings of the 2018 International Conference on Big Data and Education, March, 2018, pp: 72-75.

Jin J., Dundar A., Culurciello E., Flattened Convolutional Neural Networks for Feedforward Acceleration, arXiv preprint arXiv:1412.5474,2015.

Kılıçarslan S., Adem K., Çelik M., An overview of the activation functions used in deep learning algorithms. Journal of New Results in Science, 10(3), 2021.

LeCun Y., Bengio Y., Hinton G., Deep learning. Nature 521(7553), 436-444, 2015.

Lv X., Duan F., Jiang J., Fu X., Gan L., Deep Metallic Surface Defect Detection: The New Benchmark and Detection Network. Sensors 20(6) 1562, 2020.

Maharana K., Mondal S., Nemade B., A review: Data pre-processing and data augmentation techniques. Global Transitions Proceedings 3(1), 91-99, 2022.

Manaswi N. K., Understanding and Working with Keras. In N. K. Manaswi (Ed.), Deep Learning with Applications Using Python: Chatbots and Face, Object, and Speech Recognition With TensorFlow and Keras, Apress, pp:1–43, 2018.

Mastromichalakis S. Parametric Leaky Tanh: A New Hybrid Activation Function for Deep Learning arXiv preprint arXiv:2310.07720, 2023.

Maxwell A. E., Warner T. A., Guillén L. A., Accuracy Assessment in Convolutional Neural Network-Based Deep Learning Remote Sensing Studies—Part 2: Recommendations and Best Practices. Remote Sensing 13(13), 2591, 2021.

Mo X., Tao K., Wang Q., Wang G., An Efficient Approach for Polyps Detection in Endoscopic Videos Based on Faster R-CNN. 2018 24th International Conference on Pattern Recognition (ICPR), August, 2018, pp: 3929-3934.

Montesinos López O. A., Montesinos López A., Crossa J., Overfitting, model tuning, and evaluation of prediction performance. In Multivariate statistical machine learning methods for genomic prediction. Cham: Springer International Publishing, pp: 109-139, 2022.

Moor J., The Dartmouth College Artificial Intelligence Conference: The Next Fifty Years. AI Magazine 27(4), 87-87, 2006.

Oranen L., Utilizing deep learning on embedded devices, 2021.

Rasamoelina A. D., Adjailia F., Sinčák P., A Review of Activation Function for Artificial Neural Network. 2020 IEEE 18th World Symposium on Applied Machine Intelligence and Informatics (SAMI), 281–286, 2020.

Raschka S., Mirjalili V., Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow 2. Packt Publishing Ltd., 2019.

Rice L., Wong E., Kolter Z., Overfitting in adversarially robust deep learning. Proceedings of the 37th International Conference on Machine Learning, November, 2020, pp: 8093-8104.

Sahu M., Dash R., A Survey on Deep Learning: Convolution Neural Network (CNN). In D. Mishra, R. Buyya, P. Mohapatra, & S. Patnaik (Eds.), Intelligent and Cloud Computing, Springer, pp: 317-325, 2021.

Shinde P. P., Shah S., A Review of Machine Learning and Deep Learning Applications. 2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA), August, 2018, pp: 1-6.

Smith S. L., Kindermans P. J., Ying C., Le Q. V., Don't Decay the Learning Rate, Increase the Batch Size arXiv preprint arXiv:1711.00489v2, 2018.

Sobhana M., Hindu K., Girishma N., Bhavani P. S., Rajeswari S., A Comparitive Evaluation Of Custom CNN, Sequential CNN & Dense-Net Models to forecast Dementia. 2023 Second International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), 2023 pp: 287-293,2023.

Song K., Yunhui Y., NEU surface defect database. Northeastern University, 2019.

Swasthik, "Understanding Entropy and Losses and all those confusing names of losses,", 2020, https://medium.com/@swasthik0304/understanding-entropy-and-losses-and-all-those-confusing-names-of-losses-d7444711cf3c>, (21 April 2020).

Turban E., Watkins P. R., Integrating Expert Systems and Decision Support Systems. MIS Quarterly, 10(2), 121-136, 1986.

Vansh, "Predicting House Prices Using Keras Functional API" 2022, <https://www.analyticsvidhya.com/blog/2022/04/predicting-house-prices-using-keras-functional-api/>, (9 May 2022).

Wang M., Lu S., Zhu D., Lin J., Wang Z., A High-Speed and Low-Complexity Architecture for Softmax Function in Deep Learning. 2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS), October, 2018, pp: 223-226.

Wang Q., Ma Y., Zhao K., Tian Y., A Comprehensive Survey of Loss Functions in Machine Learning. Annals of Data Science 9(2), 187-212, 2022.

Wang S., Xia X., Ye L., Yang B., Automatic detection and classification of steel surface defect using deep convolutional neural networks. Metals 11 (3), 1-23, 2021.

Warden, Pete, "Why GEMM is at the heart of deep learning" 2015, <https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/>, (20 April 2015)

Yun J. P., Shin W. C., Koo G., Kim M. S., Lee C., Lee S. J., Automated defect inspection system for metal surfaces based on deep learning and data augmentation. Journal of Manufacturing Systems 55, 317-324, 2020.

Zhao W., Chen F., Huang H., Li D., Cheng W., A new steel defect detection algorithm based on deep learning. Computational Intelligence and Neuroscience 2021(1), 5592878, 2021.