Anomaly detection in cable placement in the power meter production process using Xception convolutional neural networks

Jonathan Cavalcante de Oliveira
Federal Institute of Education, Science and Technology - Amazonas
IFAM - Campus Manaus Centro - Manaus - AM, Brazil
j.cavalcante23@gmail.com

Marcelo Chamy Machado
Federal Institute of Education, Science and Technology - Amazonas
IFAM - Campus Manaus Centro - Manaus - AM, Brazil
marcelo.chamy@ifam.edu.br

Emmerson Santa Rita da Silva
Federal Institute of Education, Science and Technology - Amazonas
IFAM - Campus Manaus Centro - Manaus - AM, Brazil
emmerson.silva@ifam.edu.br

Márcio Aurélio dos Santos Alencar
Federal Institute of Education, Science and Technology - Amazonas
IFAM - Campus Coari - AM, Brazil
marcio.alencar@ifam.edu.br

Abstract—**In this study, we explore the use of computer vision and deep learning techniques for anomaly detection in cable positioning in the power meters' manufacturing process. The dataset used has been generated directly in the manufacturing process, which consists of 62 RGB images with dimension of 4000x1800 pixels, 45 images classified as normal and 17 classified as anomalies. The approach used in this work consists of using transfer learning from the Xception convolutional neural network (CNN) architecture together with data augmentation techniques and proposing a convolutional neural network model based on CNN Xception. To decide the best optimization, experiments were performed with different optimizer algorithms: Adam, RMSProp, SGD and AdamW, using 50 epochs to train and test the CNN architectures. A fully connected layer with 256 neurons was also added in order to improve anomaly detection. The results of the experiments showed that the best optimization algorithm of the Xception architecture was Adam, achieving an accuracy and recall of 99.84% with an additional layer of 256 neurons.**

Index Terms—**computer vision, deep learning, anomaly detection, manufacturing, transfer learning, data augmentation, convolutional neural network, Xception.**

I. Introduction

Industries are becoming increasingly competitive, and their production processes are highly automated, using technological advances and the evolution of tools that are increasingly available in relation to the variety of possible solutions. Despite this, there are still many manufacturing processes that are prone to failure due to human intervention, which is often still necessary [1]. Manufacturing problems that generate defects in industrialized products are one of the factors that cause resource waste.

Hence, industries can use advanced inspection solutions capable of performing this task with a lower error rate. However, in some cases, such an investment has a high Return on Investment (ROI), making its application unfeasible, since what is available in the manufacturing facilities of large companies are usually solutions using equipment with a high cost-benefit ratio, performing highly specialized tasks such as assembly, welding, insertion, and screwing; despite this, there are cases in which the inspection of operations is performed by humans using instruments, guides, standard tools, or, in the worst case: using only vision.

In this context, aiming to reduce the problems caused by human intervention in processes, the application of deep convolutional neural network (CNN) algorithms is one of the possible solutions that have enabled the inspection automation in the manufacturing environment [2], [3], [4] increasing the reliability of production processes.

This paper applies the Xception convolutional neural network model, trained with ImageNet weights, exploring data augmentation and transfer learning techniques aiming detecting anomalies in cable positioning in the manufacturing process of power meters.

II. Theoretical Foundations

A. Data Augmentation

Data augmentation is the process of generating new samples from each existing instance in the data by randomly transforming the existing data [4]. In the image application field, some possible examples of transformations are: rotations, flipping, cropping, brightness and contrast variation, etc. When the original dataset has few samples in each class, this method is essential to provide better generalization of the model, increase accuracy, reduce overfitting and reduce data collection time. During the training of neural network models, data augmentation process is executed with the image generator using the CPU to generate the variations of the new images to be used, while the GPU trains the generated images simultaneously.

B. Convolutional Neural Networks (CNN)

CNNs have been used for image classification since its first version, proposed by [5]. They can be used in several fields such as sound and image processing, natural language processing [6], and in medicine [7], aiding in the detection of diseases with a high reliability level [8]. CNNs are deep convolutional neural networks composed basically of convolutional layers, pooling layers, and fully connected layers, each one with specific function.

The convolutional layer performs the process of mapping one matrix to another matrix of smaller size, called kernel (K), extracting and mapping features [9]. The kernel is a convolution matrix responsible for detecting edges, traces, lines and other important features of the input image. The main parameters of this layer are the size of the kernel, the size of the stride that the kernel performs after each mapping, and the padding which is the process of filling the edge of the matrix that represents the image with zeros, so that the kernel can map the entire image completely. This layer has the greatest weight of any convolutional network, because here is where the learning occurs.

Pooling layers are a key component of the downsampling process, when using convolutional networks. These layers reduce the dimensions of the feature maps, retaining only the most relevant information. This dimensionality reduction leads to fewer network parameters, decreasing the model complexity, and consequently, reduced computational cost in subsequent layers [9].

One of the most commonly used pooling technique is max pooling, through which a matrix (which can be the mathematical representation of an image) when passing through the pooling layer is subdivided into small regions and for each region, the maximum value is calculated and goes on to represent this region in a subsampled version of the input.

The main parameters of a pooling layer are the filter size (or pooling window) and the stride, that determines how far the windows moves at each step.

Generally, the filter size and the stride that the filter moves are the same in order to avoid overlap or loss of information.

The pooling layer performs an operation similar to the convolutional layer, except that in the convolutional layer there is a kernel moving and mapping the input matrix, while in the pooling layer the filter always uses the maximum value of the sliding window.

Fully connected layers or Dense are layers in which all neurons in a node are connected to neurons of the previous layer [9]. This type of layer is usually placed after the convolutional and pooling layers or at the end of convolutional neural networks, in the classification block; that is, this layer is responsible for the model's predictions. The number of neurons in the decision layer is the same as the number of classes that the model needs to identify, if the network was designed for a classification problem. Neurons are activated or deactivated based on the transformation applied by the activation function to the output value of each neuron, based on the probabilities of each class.

Dropout layers are used to randomly turn off a certain number of neurons in each training round (epoch). This approach prevents overfitting in small datasets when training deep neural network models with many hidden layers. Each iteration performed to update the parameters, the turned off neurons are not included in any computation, but can be used in subsequent epochs. These turned off neurons, therefore, do not contribute to the data passing to the decision layer, nor to the backpropagation process. As demonstrated in the paper [10], the application of this technique increases the performance of the model in most datasets, when comparing models trained on the same datasets without dropout.

1) Activation Function: Activation functions are used in convolutional neural networks to introduce nonlinearity into the network, playing an important role in the hidden layers of the network, enabling the learning of more complex tasks. This section will only cover the two activation functions used in this paper: ReLU and Sigmoid.

The Rectified Linear Unit (ReLU) is the most commonly used activation function in convolutional neural networks [11]. This function returns 0 if the input is negative, and if the input is positive, the output is the value of the input. The ReLU does not activate all neurons in the network at the same time, since negative inputs are converted to zero, resulting in the activation of only a few neurons. With this behavior, the computational cost of this function is lower compared to other activation functions. Furthermore, ReLU helps to attenuate the vanishing gradient problem, which happens when the gradients of loss function of previous layers become very small, allowing effective model convergence. Because of its simplicity and effectiveness, ReLU has become the preferred choice among researchers. Figure 1 shows the ReLU function plot.

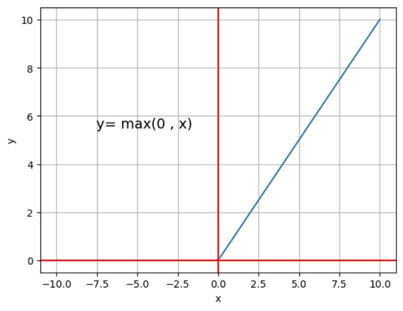


Fig. 1: ReLU function plot.

The Sigmoid function is the activation function that most closely resembles a biological neuron. A biological neuron has only two states: active and inactive, and the Sigmoid function behaves analogously. The plot of the sigmoid function in Figure 2 shows that when the values of x are small, the value of the sigmoid function is close to zero, and when the value of x is large, the value of the function is close to 1 [12]. This activation function is not used in the hidden layers of neural networks because it suffers from the saturation problem, where the gradients become close to zero, but rather in the output layers for tasks with binary variables.

C. Xception Architecture Features

In this paper, the convolutional neural network architecture chosen was Xception, which, according to the related works,

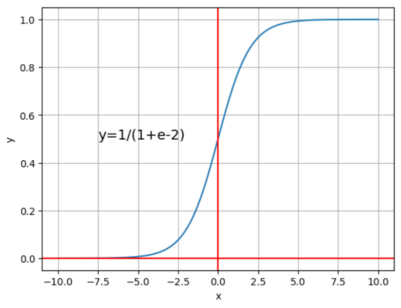


Fig. 2: Sigmoid Function plot.

has proven to be more robust for detection tasks when compared to other convolutional neural networks. Xception was born with a proposal to improve the reduction in time and complexity of InceptionV3 convolutional neural network, and is often used in experiments comparing the metrics of both networks. Traditional convolutions perform operations in spatial and depth dimensions at the same time, and Xception uses separable convolutions, reducing the number of parameters and, consequently, also reducing computational costs, which is its great differential.

Xception is trained on the Imagenet dataset, a large-scale dataset with over 14 million images and widely used for computer vision tasks. With 71 layers of depth, Xception classifies objects into 1000 different categories, as proposed by [13], who is the creator of the deep learning library Keras. The architecture of Xception is shown in Figure 3.

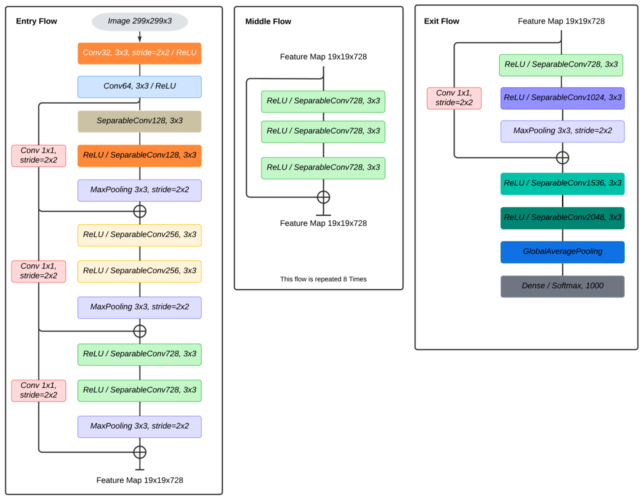

Entry Flow
Image 299x299x3
Conv32, 3x3, stride=2x2 / ReLU
Conv64, 3x3 / ReLU
SeparableConv128, 3x3
Conv 1x1, stride=2x2
ReLU / SeparableConv128, 3x3
MaxPooling 3x3, stride=2x2
ReLU / SeparableConv256, 3x3
Conv 1x1, stride=2x2
ReLU / SeparableConv256, 3x3
MaxPooling 3x3, stride=2x2
ReLU / SeparableConv728, 3x3
Conv 1x1, stride=2x2
ReLU / SeparableConv728, 3x3
MaxPooling 3x3, stride=2x2
Feature Map 19x19x728
Middle Flow
Feature Map 19x19x728
ReLU / SeparableConv728, 3x3
ReLU / SeparableConv728, 3x3
ReLU / SeparableConv728, 3x3
Feature Map 19x19x728
This flow is repeated 8 Times
Exit Flow
Feature Map 19x19x728
ReLU / SeparableConv728, 3x3
Conv 1x1, stride=2x2
ReLU / SeparableConv1024, 3x3
MaxPooling 3x3, stride=2x2
ReLU / SeparableConv1536, 3x3
ReLU / SeparableConv2048, 3x3
GlobalAveragePooling
Dense / Softmax, 1000

Fig. 3: Xception Achitecture [13].

D. Transfer Learning

Transfer learning is a widely used deep learning technique that enables the use of knowledge and weights from a pre-trained model on one dataset to perform another task [4].

Pretrained models already contain layers with weights that reflect the detection of important image features such as colors, contours and textures, making them an excellent choice for classifying images into predefined categories in its original training dataset or other datasets.

When loading a pretrained model, the strategy of preserving all the weights of the hidden layers is used, excluding the last layer: the classification layer. All the other layers of the pretrained model are frozen to avoid weight adjustments according to the errors detected in the classification of the new instances submitted to the network. A new classifier is added to perform other tasks on a new data set, as shown in Figure 4.

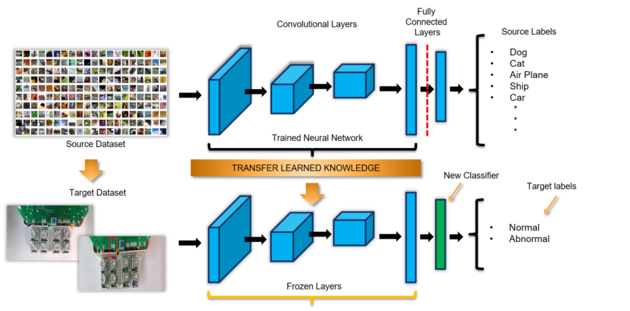


Fig. 4: Transfer Learning

III. Metodology

In this section, we will describe the methodology for building a solution for classifying images of power meter cables using the Xception convolutional neural network. Then, we will describe the strategies for generating sample data to be used for model training, preprocessing, data augmentation, definition of experimental models, training and tuning its hyperparameters, and evaluation of the metrics of each architecture and, finally, the definition of the best architecture. Figure 5 shows a flowchart with the methodology of this paper.

A. Data collection and preprocessing

Forty-five RGB images without anomalies ($label = 1$) and 17 RGB images with inverted cables in the P2P and P2M positions of the power meter ($label = 0$) were collected, with dimensions of 4000×1800 pixels, taken with the camera of the Redmi Note 11 Pro $+$ smartphone. The samples without anomalies were generated during the production process, after confirmation of the correct sequencing with an inspection standard work instruction. After confirmation, the photographic record was always taken from the same height and position. For the samples with anomalies, it was necessary to invert the P2P and P2M cables on purpose to generate the images with anomalies. Figure 6 shows two instances of the dataset.

The images were cropped to the dimension of 300×400 pixels, removing the unwanted area and selecting only the area of interest to submit to the model (the red square shown in Figure 6). This approach preserves the anomaly area information. Due to GPU memory limitations, the cropped images were resized to $71 \times 71 \times 3$. Even in such small image is still possible to

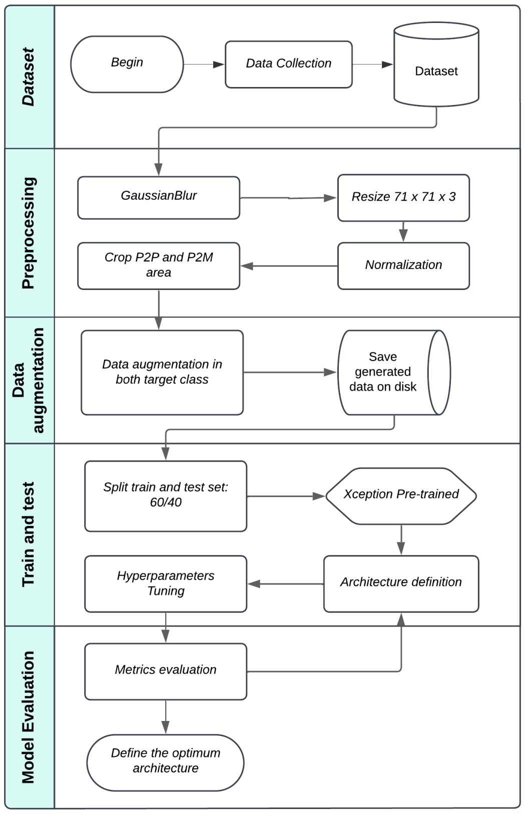


Fig. 5: Flow Chart

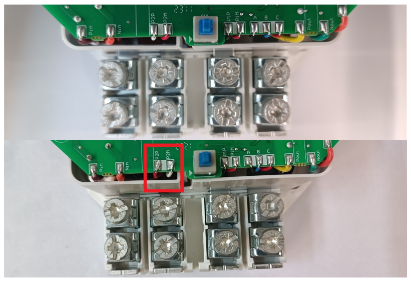
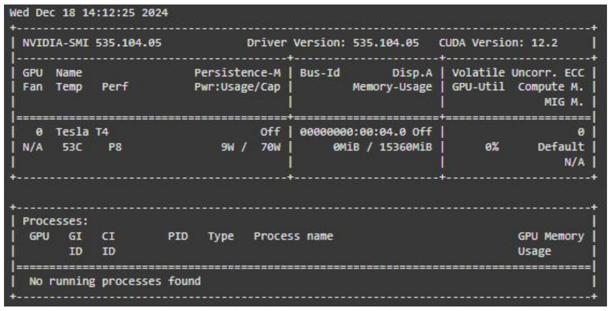


Fig. 6: Normal sample (above) and sample with cable inversion (below).

identify the cable position. The interpolation chosen to resize the images was `INTER_AREA` in `OpenCV`, which delivers good results with low computational cost in image reduction transformations. To smooth the images, the Gaussian blur function `GaussianBlurr` from OpenCV was applied with a kernel of size $k = 11$. The value of each pixel is obtained by the weighted average of the values of the pixels itself and the ones around it [14]. Its application in images causes the 2D representation of the image to improve qualitatively, according to [15]. Finally, the images were normalized by dividing each pixel by 255. In this way, the value of each pixel is within the range of 0 to 1, the same range in which the activation functions of the deep learning models work, resulting in a convergence of the weights (learning) in a more optimized way.

B. Experimental setup

For data collection, the camera of a smartphone model Redmi Note 11 Pro $+$ was used. The photo sensor is the HM2 $1/1.52$" manufactured by Samsung, with 108MP and pixels of size 0.7μ with a $24mm$ attached lens and focal length f/1.9. This smartphone was positioned $15cm$ high on a transparent polycarbonate base. The Google Colab virtual environment was used to train the convolutional network architectures. The GPU used was the Nvidia Tesla T4 with 16 GB DDR6 memory. By the !nvidia-smi command it is possible to view details about the hardware available in the virtual environment, as shown in Figure 7. All applications were developed using Python 3.10.12, `TensorFlow` 2.17.1, `Matplotlib` 3.8, `Keras` 3.5.0, `OpenCV` 4.10.0, `Scikit-learn` 1.6.0, `Numpy` 1.26.4;



```
Wed Dec 18 14:12:25 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   53C    P8               9W /  70W |      0MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
|  No running processes found                                                           |
+---------------------------------------------------------------------------------------+
```

Fig. 7: Hardware used in the Google colab virtual environment.

C. Application of transfer learning

To perform the transfer learning technique, the Xception convolutional neural network was initialized by loading the weights of the pretrained network on the ImageNet dataset. During initialization, the layers that will be reused are kept frozen, which in Xception correspond to the input and intermediate streams. As shown in Figure 4 in the output stream, the classification layer of the original CNN task, which contains 1000 neurons, was excluded and the remaining layers were kept frozen. This remaining structure is called the base model. The main parameters when initializing the network were:

- `weights`=imagenet, to load pretrained weights of the network;
- `input_shape`$= 71 \times 71 \times 3$, to set the network input to the same size as the dataset images;
- `include_top`$= False$, to not load the convolutional layers of the model;

D. Data Augmentation Application

The Keras `ImageDataGenerator` was used to generate data on disk for both classes, with the aim of balancing the samples in the dataset. Some of the transformations performed on the images were: vertical and horizontal flipping, zoom varying by 10%, height and length displacements varying

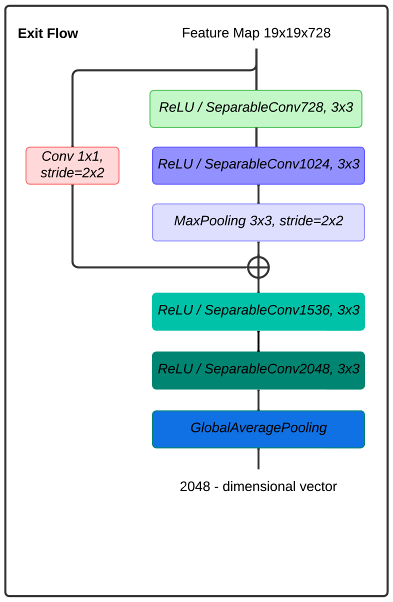

Exit Flow
Feature Map 19x19x728
ReLU / SeparableConv728, 3x3
Conv 1x1, stride=2x2
ReLU / SeparableConv1024, 3x3
MaxPooling 3x3, stride=2x2
ReLU / SeparableConv1536, 3x3
ReLU / SeparableConv2048, 3x3
GlobalAveragePooling
2048 - dimensional vector

Fig. 8: Preparation of the base model for transfer learning.

by 10% and rotations by 10%, always keeping the center of the image as a reference. It is important to define the parameters for data augmentation well and not to use excessive transformations that can end up worsening the model's performance. The `ImageDataGenerator` function used to generate images on disk was `datagen.flow`. Figure 9 shows some images generated by applying the data augmentation technique. The data generated from data augmentation in this paper was used in training and testing the model.

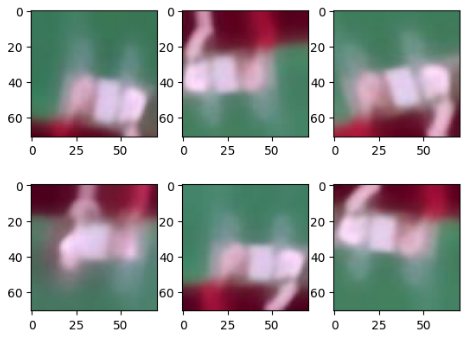
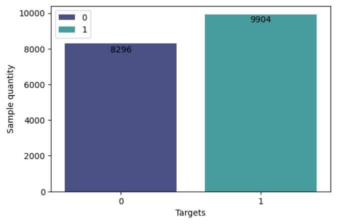


Fig. 9: Images generated with data augmentation.

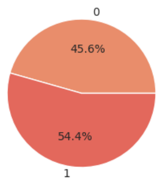
All images generated through data augmentation process were saved in **JPEG** format. The final dataset consisted of both the augmented and the original real images. Figure 10 illustrates the number of samples per class and the resulting class distribution, demonstrating the effectiveness of the data augmentation strategy in mitigating class imbalance.

E. Model definition and hyperparameter tuning

For Models 1 through 4, the upper layers of the pretrained base model were frozen, and a Dense layer - referred to as the *decision layer* - was appended. This layer consisted of a single neuron with a sigmoid activation function.

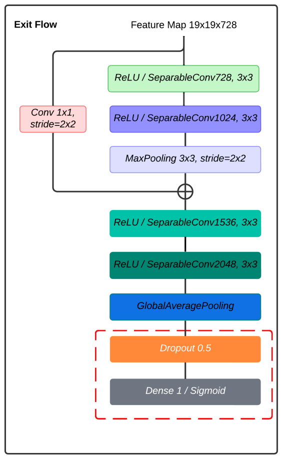


(a) Sample quantity of each class

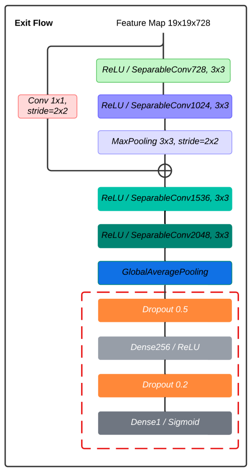


(b) Dataset balancing

Fig. 10: Analysis of dataset distribution after data augmentation.


Exit Flow
Feature Map 19x19x728
ReLU / SeparableConv728, 3x3
Conv 1x1, stride=2x2
ReLU / SeparableConv1024, 3x3
MaxPooling 3x3, stride=2x2
ReLU / SeparableConv1536, 3x3
ReLU / SeparableConv2048, 3x3
GlobalAveragePooling
Dropout 0.5
Dense 1 / Sigmoid

(a) Architecture of models 1, 2, 3 and 4


Exit Flow
Feature Map 19x19x728
ReLU / SeparableConv728, 3x3
Conv 1x1, stride=2x2
ReLU / SeparableConv1024, 3x3
MaxPooling 3x3, stride=2x2
ReLU / SeparableConv1536, 3x3
ReLU / SeparableConv2048, 3x3
GlobalAveragePooling
Dropout 0.5
Dense256 / ReLU
Dropout 0.2
Dense1 / Sigmoid

(b) Architecture of models 5, 6, 7 and 8

Fig. 11: Architecture diagram of the proposed models.

Since the classification problem addressed in this work is binary, the sigmoid function was employed to produce an output in the range [0,1]. With the same reasoning, the **binary_crossentropy** loss function was used to measure prediction error, since the target labels are either 0 or 1.

In a different approach, in models 5, 6, 7 and 8 a dense layer with 256 neurons and ReLU activation function was added and, after that, another layer with dropout of 20%. Dropout works by randomly deactivating neurons in the hidden layers of the model at each training cycle, making the model more robust [8].

These layers were trained jointly with the decision layer, increasing the depth of the network. All models were trained for 50 epochs, and the experimental details are presented in Table I. The following additional training parameters were configured:

- `batch_size` set to 128;
- `ReduceLROnPlateau` with learning rate reduction after 5 epochs without improvement, and a minimum learning rate of 1×10^{-7};
- `EarlyStopping` with a patience of 10 epochs;

- `ModelCheckpoint` enabled to save the model with the lowest validation loss;
- All three callbacks—`ReduceLROnPlateau`, `EarlyStopping`, and `ModelCheckpoint`—monitored the validation loss (`val_loss`);
- `TensorBoard` was integrated via `keras.callbacks` to visualize training and validation loss and accuracy metrics.

TABLE I: Proposed models

Model	Learning rate	Optimizer
1	$1e-4$	Adam
2	$1e-4$	RMSProp
3	$1e-2$	SGD
4	$1e-4$	AdamW
5	$1e-4$	Adam
6	$1e-4$	RMSProp
7	$1e-2$	SGD
8	$1e-4$	AdamW

IV. Results

This section presents the evaluation of the metrics of each experiment performed.

A. Metrics results

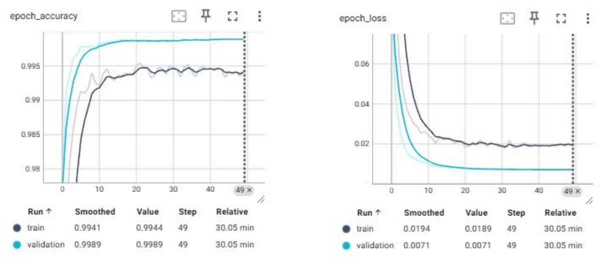
The evaluation metrics for each of the proposed models are presented in Table II. Among all evaluated models, Model 5 demonstrated superior performance across all key metrics: precision, recall, F1-score and accuracy. The result shows that, despite the data set used in this paper being limited, transfer learning and data augmentation techniques proved to be powerful allies of convolutional neural networks.

TABLE II: Metrics of the proposed models.

Model	Precision	Recall	F1 Score	Acuracy	Error
1	0.9853	0.9780	0.9803	0.9804	0.1182
2	0.9929	0.9854	0.9881	0.9882	0.0799
3	0.9933	0.9877	0.9898	0.9898	0.0632
4	0.9891	0.9774	0.9817	0.9817	0.1068
5	***0.9982***	***0.9994***	***0.9982***	***0.9984***	0.0071
6	0.9990	0.9975	0.9980	0.9990	0.0074
7	0.9977	0.9899	0.9932	0.9933	0.0313
8	0.9967	0.9985	0.9983	0.9984	0.0066

When analyzing the graphs generated by tensorboard of Model 5, we can see that accuracy increased over the 50 epochs, both in training and validation. The error decreased over the 50 epochs, which shows the absence of overfitting and good generalization of the model.

Models 1 and 5 were trained using the same optimizer, learning rate, and number of epochs. However, Model 1 employed a shallower architecture compared to Model 5. This comparison indicates that increasing the network depth by adding an additional fully connected layer led to significant improvements, reducing both false positives and false negatives.

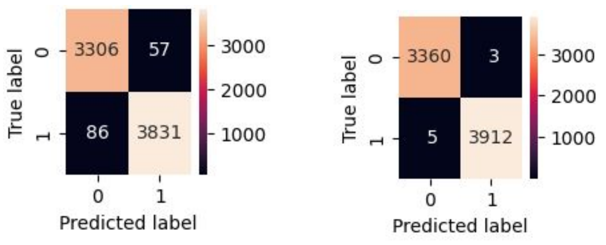


Run ↑	Smoothed	Value	Step	Relative
● train	0.9941	0.9944	49	30.05 min
● validation	0.9989	0.9989	49	30.05 min

Run ↑	Smoothed	Value	Step	Relative
● train	0.0194	0.0189	49	30.05 min
● validation	0.0071	0.0071	49	30.05 min

(a) Accuracy over 50 epochs (b) Error over 50 epochs

Fig. 12: Accuracy and error graphs generated by tensorboard.



(a) Model 1 (b) Model 5

Fig. 13: Confusion matrices.

B. Limitations and Improvements

Although Model 5 achieved promising results, further refinement is required before it can be deployed in a production environment. In an automated inspection station, even a misclassification rate of $0,1\%$ will cause disruptions due to unnecessary stoppages or, in worst cases, allow defective products to reach the customers.

To address this issue, the model architecture may be enhanced by adding convolutional, pooling, or fully connected layers after the base model to enhance the model's ability to correctly classify defective and non-defective items by reducing false positive and false negative rates. Alternatively **batch normalization layers** can be introduced following each **Dropout layer**, which can stabilize and accelerate training without significantly increasing model complexity. Furthermore, data augmentation can also be extended through **Generative Adversarial Networks (GANs)**, which allows the generation of synthetic data with greater diversity than traditional augmentation techniques, possibly generating previously unseen samples.

Despite its high accuracy, the proposed model may present limitations in terms of real-time performance when compared to architectures such as the YOLO (You Only Look Once) neural network. Real-time constraints include the communication speed between the camera and the processing unit, the time required for image pre-processing, and the model's inference latency. Since each inspection must be completed before the next product reaches the station, these aspects can become critical bottlenecks.

To overcome these challenges, deploying a high-performance industrial camera, integrating trigger sensors, designing optimized lighting conditions, and implementing an Andon system for real-time signaling are recommended. Together, these components can support the practical application of the proposed architecture on the production line.

V. Conclusion

The automatic detection of anomalies in industrial processes plays a vital role in improving product quality and production reliability, thus contributing to the competitiveness of modern manufacturing. While computer vision is not new to industrial inspection systems, the adoption of convolutional neural networks (CNNs) introduces a significant advancement in quality control, with the potential to reduce labor costs and increase operational efficiency.

Future work will involve implementing the trained model on embedded hardware, including a user interface for real-time monitoring and the automatic upload of statistical data to the factory database. This integration will enhance statistical process control and provide continuous feedback for production optimization.

In this study, a system for detecting anomalies in the positioning of energy meter cables was developed using transfer learning and data augmentation techniques. Despite the limited dataset—consisting of only 62 RGB images—two approaches based on the Xception convolutional neural network were proposed. The best-performing model achieved accuracy and recall rates of up to 99.84%.

References

[1] X. Le, J. Mei, H. Zhang, B. Zhou, and J. Xi, "A learning-based approach for surface defect detection using small image datasets," *Neurocomputing*, vol. 408, pp. 112–120, 2020.

[2] J. P. d. V. M. T. L. S. J. F. João Alexandre Lôbo Marques, Francisco Nauber Bernardo Gois, "Chapter 4 - artificial neural network-based approaches for computer-aided disease diagnosis and treatment," in *Cognitive and Soft Computing Techniques for the Analysis of Healthcare Data*, ser. Intelligent Data-Centric Systems, A. K. Bhoi, V. H. C. de Albuquerque, P. N. Srinivasu, and G. Marques, Eds. Academic Press, 2022, pp. 79–99.

[3] H. Hu, S. Li, J. Huang, B. Liu, and C. Che, "Casting product image data for quality inspection with xception and data augmentation," *Journal of Theory and Practice of Engineering Science*, vol. 3, no. 10, p. 42–46, Oct. 2023.

[4] K. Kılıç, K. Kiliç, Doğru, and U. Özcan, "Using deep learning techniques for anomaly detection of wood surfaceprimjena tehnika dubokog učenja za otkrivanje anomalija na površini drva," *Drvna industrija*, vol. 75, pp. 275–286, 10 2024.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[6] K. Kaur and P. Kaur, "Bert-cnn: Improving bert for requirements classification using cnn," *Procedia Computer Science*, vol. 218, pp. 2604–2611, 2023, international Conference on Machine Learning and Data Engineering.

[7] V. Sathiya, B. Shenbagavalli, V. Nirupa, and K. Subramani, "Detection and classification of diabetic retinopathy using inception v3 and xception architectures," *International Journal of Nutrition, Pharmacology, Neurological Diseases*, vol. 14, pp. 128–136, 02 2024.

[8] S. A. Joshi, A. M. Bongale, P. O. Olsson, S. Urolagin, D. Dharrao, and A. Bongale, "Enhanced pre-trained xception model transfer learned for breast cancer detection," *Computation*, vol. 11, no. 3, 2023.

[9] M. M. Taye, "Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions," *Computation*, vol. 11, no. 3, 2023.

[10] N. Srivastava, "Improving neural networks with dropout," *University of Toronto*, vol. 182, no. 566, p. 7, 2013.

[11] B. Krishnamurthy. (2024) Uma introdução à função de ativação relu. Built in.

[12] Y. Chen, L. Li, W. Li, Q. Guo, Z. Du, and Z. Xu, "Chapter 2 - fundamentals of neural networks," in *AI Computing Systems*, Y. Chen, L. Li, W. Li, Q. Guo, Z. Du, and Z. Xu, Eds. Morgan Kaufmann, 2024, pp. 17–51.

[13] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 1800–1807.

[14] X. Lu and Y. Zhang, "Chapter eight - an improved canny detection method for detecting human flexibility," in *Intelligent IoT Systems in Personalized Health Care*, ser. Cognitive Data Science in Sustainable Computing, A. K. Sangaiah and S. Mukhopadhyay, Eds. Academic Press, 2021, pp. 207–234.

[15] G. Lugo, N. Hajari, and I. Cheng, "Semi-supervised learning approach for localization and pose estimation of texture-less objects in cluttered scenes," *Array*, vol. 16, p. 100247, 2022.