Universidad Nacional de Educación a Distancia
Universidad Complutense de Madrid

RAILWAY TRACK FAULT DETECTION ANALYSIS BASED ON DEEP LEARNING TECHNIQUES

Máster en Ingeniería de Sistemas y Control
E.T.S Ingeniería Informatica

Academic Year: 2021 - 2022

Author: Jon Rengel Jimenez

Supervisors: Matilde Santos
Ravi Pandit

September 2022

Universidad Nacional de Educación a Distancia
Universidad Complutense de Madrid

RAILWAY TRACK FAULT DETECTION ANALYSIS BASED ON DEEP LEARNING TECHNIQUES

Máster en Ingeniería de Sistemas y Control
E.T.S Ingeniería Informatica

Academic Year: 2021 - 2022

Author: Jon Rengel Jimenez

Supervisor: Matilde Santos
Ravi Pandit

September 2022

ABSTRACT (Eng)

Global passenger rail activity is expected to double from present levels in 2050 [1]. With current capacity bottlenecks, it is estimated that the required rail networks need to be extended by more than 430 000 track-kilometres by 2050. This growth highlights the importance of regularly inspecting the railway network for maintaining safe and reliable train operations. The development in computer power together with computer vision techniques, have boosted the use of video cameras for inspecting railway tracks.

This document presents the analysis and comparison of different Deep Convolutional Neural Network architectures to detect railway track faults on small datasets. Moreover, it includes an analysis on input image resolution impact on the results. After a thorough analysis on state of the art architectures, ResNet and EfficientNet families have been selected for being the former, the most used one in railway applications and, the latter, for its high performance values and low computation cost [31].

Results show that every EfficientNet family (B0 to B7) architecture outperformed that of ResNet, where validation accuracies did not exceed the 80% threshold. In agreement with Ref. [31], EfficientNet B7 family has achieved the highest validation accuracy, obtaining a best value of 89.1%, while B2 scored an 87.5% of accuracy being the best performer in computational cost to performance ratio. In average score accuracy, B7 and B2 were also the best performers, scoring 84.6% and 84.2% respectively. On the other hand, results show that image resolution has an impact on validation accuracy, not always being the image size the network was developed with the best performer. To validate the trained model, an object detection algorithm based on sliding windows, pyramid scaling and non-maxima suppression has been implemented, showing that the model satisfactorily detects the faulty area on the image.

Keywords: Deep learning, Neural Network, CNN, EfficientNet, Fault detection, Railway

RESUMEN

Los estudios de movilidad ferroviaria prevén que el transporte de pasajeros se duplique a nivel mundial respecto a los niveles actuales en 2050 [1]. El existente cuello de botella indica que las redes ferroviarias deberán ampliarse en más de 430.000 kilómetros para dicho año. Este crecimiento pone de manifiesto la importancia de inspeccionar periódicamente las vías para mantener la seguridad y la fiabilidad de las operaciones de los trenes. El desarrollo de la tecnología computacional, junto con las técnicas de visión por ordenador, han impulsado el uso de cámaras de vídeo para inspeccionar las vías férreas.

Este documento presenta un análisis y comparación de múltiples arquitecturas de Redes Neuronales Convolucionales Profundas para detectar fallos en vías férreas en pequeñas bases de datos. Además, incluye un análisis del impacto de la resolución de la imagen de entrada en los resultados. Tras un análisis exhaustivo de las arquitecturas existentes, se han seleccionado las familias ResNet y EfficientNet por ser la más utilizada en aplicaciones ferroviarias y por sus altos valores de precisión y bajo coste computacional respectivamente [31].

Los resultados muestran que las arquitecturas de la familia EfficientNet (B0 a B7) superan la red ResNet, donde la precisión de validación es inferior al 80%. De acuerdo con la Ref. [31], la familia EfficientNet B7 ha logrado la mayor precisión, obteniendo un valor del 89,1%, mientras que B2 obtuvo un 87,5%, siendo la mejor en el ratio coste computacional-rendimiento. En cuanto a la precisión media, B7 y B2 también fueron las arquitecturas con mejores resultados, 84,6% y 84,2% respectivamente. Por otro lado, los resultados confirman que la resolución de la imagen tiene un impacto en la precisión, no siendo siempre el tamaño de la imagen con el que se desarrolló la red la mejor. Para validar el modelo, se ha implementado un algoritmo de detección de fallos basado en ventanas deslizantes, escalado de pirámides y supresión de no máximos, mostrando que el modelo detecta satisfactoriamente la zona defectuosa.

Palabras clave: Aprendizaje profundo, redes neuronales, CNN, EfficientNet, detección de fallos, vías ferroviarias.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Universidad Nacional de Educación a Distancia (UNED) and Universidad Complutense de Madrid (UCM) for joining forces to create this mater's and giving me the chance of taking part in it. Thank you for showing me the fascinating world of controls and systems engineering.

I am deeply grateful to Matilde Santos and Ravi Pandit, for guiding me throughout the project and making this possible. Further thanks go to Matilde for his support, encouragement and always open door, even in summer break, during the completion of this project.

I am grateful to Ravi Pandit, whose technical support and expertise throughout the project made this possible. Their guidance and collaboration have been indispensable.

To my family and friends, you should know that your support and encouragement was worth more than I can express on paper. It really made these two academic years much easier.

I would also like to thank Matilde Santos for proof reading this master's thesis from cover to cover.

TABLE OF CONTENTS

ABSTRACT (Eng) 1
RESUMEN 2
ACKNOWLEDGEMENTS 3
LIST OF FIGURES 6
LIST OF TABLES 8
LIST OF ABBREVIATIONS 9
1 Introduction 11
1.1 Project context and motivation 11
1.2 Research objectives 13
1.3 Thesis structure 14
1.4 Publications 14
2 Literature Review and Deep Learning Fundamentals 15
2.1 Literature review 15
2.2 Convolutional Neural Networks 16
2.2.1 Overview of CNNs 17
2.2.2 Architectures 20
2.2.2.1 VGG16 20
2.2.2.2 ResNet 22
2.2.2.3 EfficientNet 24
2.3 Fault detection 28
2.3.1 Main algorithm 28
3 Dataset analysis, pre-processing and data augmentation 31
3.1 Dataset analysis 31
3.2 Dataset partitioning 34
3.3 Pre-Processing and Data Augmentation 36
3.3.1 Pre processing 36
3.3.1.1 Data normalization: 37
3.3.1.2 Image resizing: 37
3.3.2 Data augmentation 38
3.3.2.1 Position and orientation shift 39
3.3.2.2 Shear and zoom 40
3.3.2.3 Brightness 41
4 Methodology 43
4.1 Analysed scenarios 43
4.1.1 ResNet cases 43
4.1.2 EfficientNet cases 44
4.2 Model Setup 45
4.2.1 Fault detection parameters 48
4.3 Evaluation criteria 49
4.3.1 Accuracy 49

4.3.2 F1 Score 50
5 Discussion of the results 53
5.1 ResNet 53
5.1.1 Number of parameters and FLOPS 53
5.1.2 Validation accuracy 54
5.1.3 Test accuracy 55
5.1.4 Others 55
5.2 EfficientNet 56
5.2.1 Number of parameters and FLOPS 57
5.2.2 Validation accuracy 58
5.2.3 Test accuracy 60
5.2.4 Others 61
5.3 CNN results conclusions 62
5.4 Fault detection 64
5.4.1.1 Defective images detected as defective 65
5.4.1.2 Defective images detected as non-defective 71
5.4.1.3 Non-Defective images detected as defective 72
6 Conclusions and future works 75
REFERENCES 77
APPENDICES 81

LIST OF FIGURES

Figure 1: Global line kilometres of metro, light rail and high-speed rail, 2000-2018 in thousand line-km [2]..... 11

Figure 2: (a) Partial crack; (b) Wheel burnt issue; (c) Damage of surface of rail head due to super elevation issue; (d) Shelling. Image sources [3], [4]..... 12

Figure 3: CNN image classification pipeline [27] 17

Figure 4: Example of a convolutional layer filter process [28]..... 18

Figure 5: Max pooling layer example [21]..... 19

Figure 6: VGG16 architecture [Researchgate.net] 21

Figure 7: Comparison with the state of the art in ILSVRC classification. VGG16 denoted as VGG [29]..... 21

Figure 8: Common CNN vs ResNet [Researchgate.net] 22

Figure 9: Comparison of error rates, number of layers and number of parameters of different architectures based on ImageNet [30] 23

Figure 10: Model scaling illustration [31] 24

Figure 11: Architecture of baseline network EfficientNet-B0 [32]..... 25

Figure 12: Comparison of EfficientNet accuracy vs other architectures on ImageNet dataset[31] 26

Figure 13: Model parameters vs Transfer learning accuracy pretrained on ImageNet [12] 27

Figure 14: NMS [39] 29

Figure 15: Example of 9 randomly selected images showing defective and non-defective cases..... 32

Figure 16: Maintenance vehicle. Multipurpose tamper (MPT) [43] 33

Figure 17: Examples of fault types on railway tracks..... 33

Figure 18: Most common data partition percentages for test, validation and test 35

Figure 19: Cropped image to make it square 38

Figure 20: Augmented images applying position and rotation shifts..... 40

Figure 21: Augmented images applying zoom and shear intensity 41

Figure 22: Brightness analysis of 4 randomly selected images. 42

Figure 23: Model setup process 45

Figure 24: Confusion matrix composition 49

Figure 25: Comparison of number of parameters and FLOPS between the original and processed resolution with ResNet 54

Figure 26: Comparison of number of parameters and FLOPS between the original and processed resolution with EfficientNet 57

Figure 27: Comparison of average validation accuracy between original and processed resolution with EfficientNet 58

Figure 28: Comparison of best validation accuracy between original and processed resolution with EfficientNet 59

Figure 29: Comparison of average test accuracy between original and processed resolution with EfficientNet 60

Figure 30: Comparison of average validation accuracy between original and processed resolution with EfficientNet 61

Figure 31: (a) Original defective image (1/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area 66

Figure 32: (a) Original defective image (2/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area 67

Figure 33: (a) Original defective image (3/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area 68

Figure 34: (a) Original defective image (4/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area 69

Figure 35: (a) Original defective image (5/6); (b) ROIs above 60% fault probability; (c) highest fault probability area 70

Figure 36: (a) Original defective image (6/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area 71

Figure 37: Defective images that have been detected as non-defective 72

Figure 38: (a) Non-defective images that have been detected as defective (1/2); (b) ROIs above 60% fault probability; (c) highest fault probability area 73

Figure 39: (a) Non-defective images that have been detected as defective (1/2); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area 74

LIST OF TABLES

Table 1: Run cases with ResNet architecture..... 44
Table 2: Run cases with EfficientNet architecture 45
Table 3: Results with ResNet simulation models..... 53
Table 4: Standard deviations for every case with ResNet 55
Table 5: Results with EfficientNet simulation models 56
Table 6: Standard deviations for every case with EfficientNet..... 62
Table 7: Confusion matrix for EfficientNet-B2 case with original resolution..... 64

LIST OF ABBREVIATIONS

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CMYB	Cyan Magenta Yellow Black
ConvNet	Convolutional Network
DCNN	Deep Convolutional Neural Network
FLOPS	Floating-Point Operations Per Second
FN	False Negatives
FP	False Positives
IEA	International Energy Agency
ILSVRC	ImageNet Large-Scale Visual Recognition Challenge
MBConv	Mobile Bottleneck Convolutions
NAS	Neural Architectural Search
NMS	Non Maxima Suppression
ReLU	Rectified Linear Unit
TN	True Negatives
TP	True Positives
RGB	Red Green Blue
ROI	Region Of Interest
VGG	Visual Geometry Group

1 Introduction

1.1 Project context and motivation

Since the invention of the first steam engine, railways have become one of the most popular and safe land transportation for people and goods. The increase in fossil fuel prices together with the development of train technology, which is reducing travel times, make the railway industry one of the most powerful ones. According to the International Energy Agency (IEA), the global passenger rail activity is expected to double (116%) from present levels in 2050 [1].

Looking at the global line kilometres of most common railway passenger transports, shown in Figure 1, it can be seen that the number of kilometres in thousand line-km has considerably increased in the last two decades. This is especially visible with the high-speed rail.

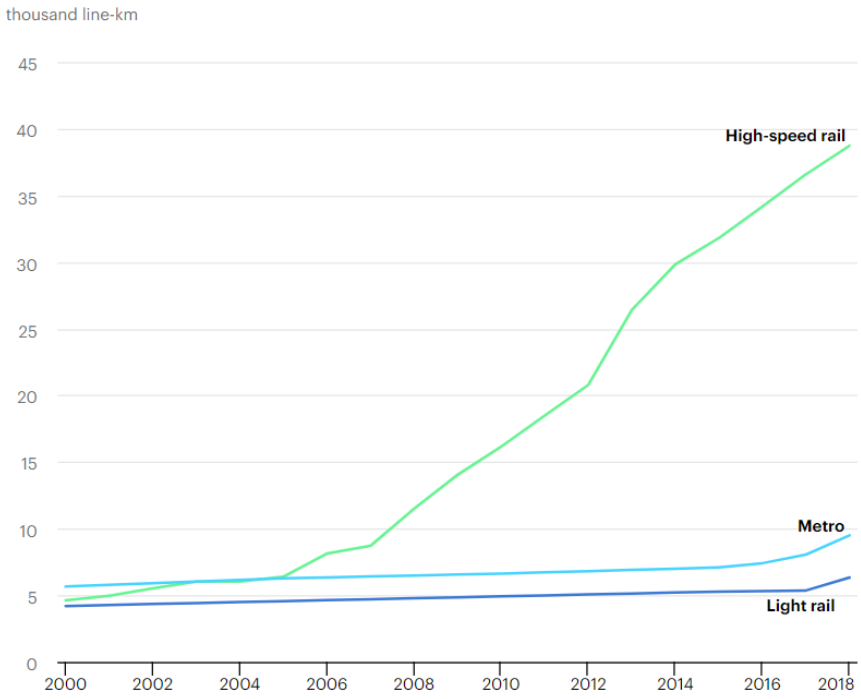


Figure 1: Global line kilometres of metro, light rail and high-speed rail, 2000-2018 in thousand line-km [2]

With the increasing transport demand and current capacity bottlenecks, it is estimated that the required rail networks need to be extended by more than 430 000 track-kilometres through to 2050, a 27% increase from 2016 [1].

The large number of already existing rail tracks together with the expected increase of it, highlights the importance of regularly inspecting the railway network for maintaining safe and reliable train operations. Historically, these regular inspections were manually performed using a railway cart; however, more novel and automatic methods are required to ensure a high effectiveness and to avoid human errors.

There are many types railway track faults than can cause issues on the train, the main ones can be categorized into ballast issues, superelevation, wheel burn, loose nuts and bolts [3]. However, there are some others thar might not be so easily recognizable such as squats, corrugation, shelling or head check [4]. Some of these defects are shown in Figure 2.

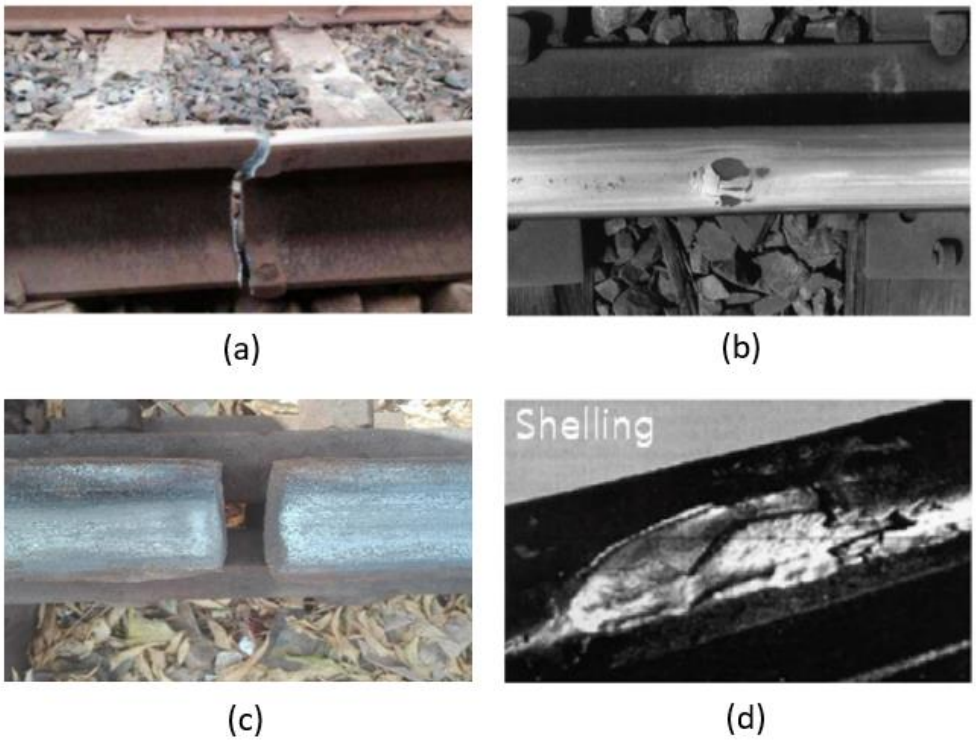


Figure 2: (a) Partial crack; (b) Wheel burnt issue; (c) Damage of surface of rail head due to super elevation issue; (d) Shelling. Image sources [3], [4]

The detection method of the fault might vary from one fault to another. While some of the faults might be detected by sight as in (a) in Figure 2, some others might require analysing the inner microstructure of the railway track.

Thus, the motivation of this Final Master Project is to develop a computational tool based in deep learning techniques to detect fault in railways in an automatic way.

To fulfil this objective, several state of the art deep learning algorithms have been studied and compared as well as the effect of image resolution on the fault detecting capabilities.

The specific research objectives are detailed in the following section.

1.2 Research objectives

The **main objective** of this research work is to develop a computational tool based on deep learning techniques and trained in a small dataset capable of detecting faults in non-professional railway track images.

The **specific objectives** developed along the master's thesis are the following:

- Study of the state of the art regarding railway faults
- Study of the state of the art regarding method used for fault detection in images
- Study of the state of the art regarding deep learning techniques and architectures
- Successfully train a deep learning architecture on a very small dataset of non-professional images
- Analyse and compare the capability of fault detection on previously used deep learning techniques and the state of the art algorithms for railway tracks
- Analysis the effect of modifying input image resolution on fault detection
- Generate a work procedure for future deep learning algorithm technique comparison

1.3 Thesis structure

This work has been divided into the following chapters.

Chapter 1 introduces the problem we are going to deal with, the motivation of the work and the objectives of the PFM are described.

Chapter 2 provides the theoretical background for deep learning techniques, especially Convolutional Neural Networks (CNN). Furthermore, it introduces three different architectures from which two are used in this study. A literature review on different fault detection methods on railway tracks is also presented.

In Chapter 3, an analysis of the initial image dataset is carried out together with an explanation of its partition. The applied pre-processing and data augmentation techniques to this dataset are described.

Chapter 4 outlines the methodology, starting with the analysed scenarios and following with an explanation of the models' set-up. It also describes the evaluation criteria that will be applied to evaluate the results of the different cases.

The results obtained in this study are shown in Chapter 5. Firstly, results for ResNet architectures are provided and then compared with those from EfficientNet families. A discussion of the results is included as well as a fault detection process of the images.

Chapter 6 presents the conclusions obtained and gives an outlook for recommended future steps and research.

1.4 Publications

- Jon Rengel, Matilde Santos and Ravi Pandit. EfficientNet Architecture Family Analysis on Railway Track Defects. 23rd International Conference on Intelligent Data Engineering and Automated Learning (IDEAL), 24-26/11/2022. Submitted.

2 Literature Review and Deep Learning Fundamentals

In this chapter, a literature review on deep learning techniques for image classification is presented. For each technique, a detailed technical explanation is provided as well as a specific literature review on the explained method.

Besides, an object detection method that will be used to detect the faulty areas on the images is also explained at the end of the section.

2.1 Literature review

Railway track faults have happened since the first tracks were installed. However, the methods and techniques used to detect them have developed along time and with the advances of the technology.

Some decades ago, and still in some underdeveloped countries, some of the track inspections are performed visually, this is, by individually inspecting and checking each section of the track by a human operator.

One technique that has been considerably used in the last two decades has been acoustic analysis. Different acoustic analysis have been performed to detect faults in railway tracks, being Acoustic Emissions (AE) one of the best known ones [5], [6]. It is important to highlight that the acoustic analysis has been used specially to detect defects on the microstructure of the tracks.

Nevertheless, the development in computer technology and artificial vision techniques have lately boosted the use of video cameras for inspecting the tracks [7], [8]. To this day, many image processing [9], [10] and deep learning techniques [11]–[15] have been applied to railway track monitoring. As explained in Ref. [16], not every technique is valid for every track issue.

Among the available techniques, image classification with deep learning methods have become popular as they have shown to get considerably high accuracy, especially those involving Convolutional Neural Networks (CNN) [17]–[19]. According to [9], the Deep Convolutional Neural Network (DCNN) architecture developed by them "*uses only one network for fastener defects detection and achieves a sound classification accuracy of 97.14%".*

To this point, the scientific community has mainly focused on railway track fasteners, using deep neural networks to detect them on images and to analyse the different type of faults they might present.

The available dataset is key when training the model, as the more cases are loaded, the higher the chances of getting large accuracy values are. In Ref. [13], with a total dataset of 19970 images, *"The precision rate and recall rate of fasteners detection including defects were 89% and 95%, respectively".*

Many DCNN architectures have been developed so far, however, not all of them are valid for every study case. In Ref. [14], the authors perform an analysis and comparison on railway fastener fault diagnosis of different architectures, being VGG16 and ResNet the ones with best results. From the analysis, the authors deduced that *"it can be seen that ResNet has the best effect"* and combining it with the adequate pre-processing methods, did *"accurately classify each type of fastener with a general accuracy of over 98%"*.

2.2 Convolutional Neural Networks

Convolutional Neural Network (CNN) is a well-known deep learning architecture inspired by the natural perception mechanism of living creatures [20]. CNNs, also known as ConvNets, have been used for visual tasks since the late 1980s [21]. However, the recent developments in computational technology has turned them into one of the most popular kind of networks to classify images.

A CNN is a type of Artificial Neural Network (ANN) primarily used for image recognition and classification that is designed to process pixel data [22]. Compared to standard ANNs, CNN are just neural networks that use convolution. A convolution is applied to the input data to filter the information and generate a feature map. These filters are also commonly known as Kernels.

The main advantages of ConvNets are that they are able to successfully capture the spatial and temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters or weights. In

other words, the network can be trained to understand the sophistication of the image better.

Nowadays, CNNs have probably become one of the most popular machine learning techniques and are applied in many different fields. In engineering, they are used in every discipline, from high voltage fault detection [23] to turbine blade crack propagation assessment [24]. In medicine for example, they have recently been used to detect COVID-19 cases [25], [26].The ones mentioned here are just a few examples that show there is no theoretical limit to where this technique can be applied.

2.2.1 Overview of CNNs

There are hundreds of variants of CNN architectures in the literature, however, their basic components are very similar [20]. Every CNN architecture is divided into the following steps: input image, convolutional layers, fully connected layers and output. This process is represented in Figure 3.

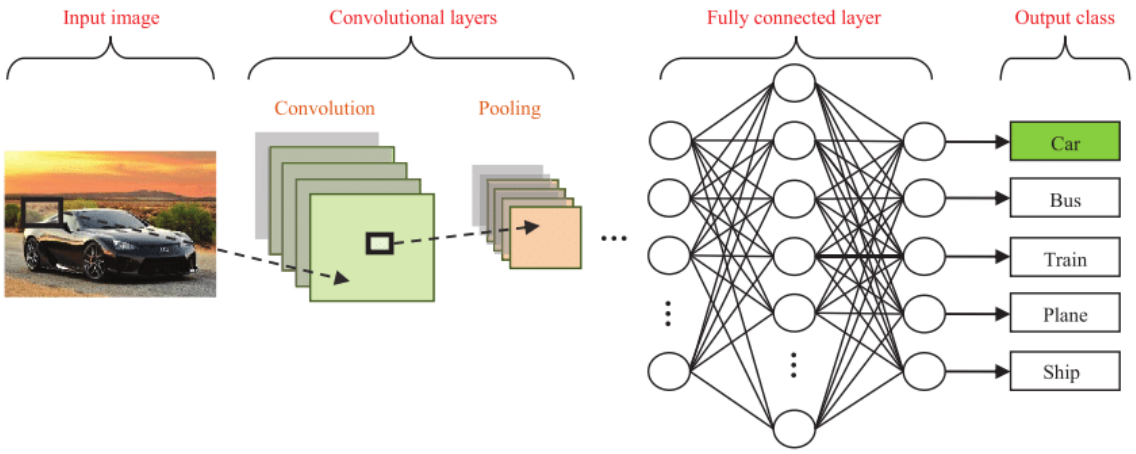


Figure 3: CNN image classification pipeline [27]

In this subsection, every of the above mentioned components will be shortly explained:

- **Input Layer:** the first step is feeding the ConvNet with an image. This image can be RGB, CMYB or any other as the architecture works for any.

- **Convolutional Layers**: The convolutional layer aims to learn feature representations of the inputs by arranging neurons into feature maps [20], [21]. Inputs are convolved with learned weights in order to compute a new feature map and results are evaluated through a nonlinear activation function.

  The convolutional layers are made up of two main layers that can be repeated several times to improve accuracy. The most common ones are:

  - **Convolution:** is a mathematical operation that allows to merge sets of information. In CNNs, convolutions are applied to the input data (image pixels) to filter the information and generate feature maps.

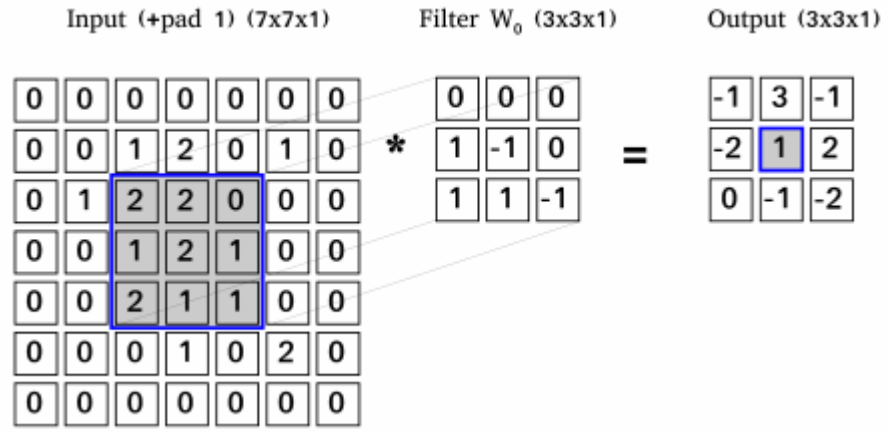


Figure 4: Example of a convolutional layer filter process [28]

The sets of filters contain learnable parameters, which are changed while training. The objective is to iterate with those learning parameters, called weights, so that it optimizes the whole CNN accuracy.

- **Pooling:** The pooling layer receives the output of the convolution layer and compresses it. This means that the filter used is always smaller than the generated feature map. Usually, this is a 2x2 or 3x3 filter.

  There are two types of pooling, maximum and average. The first one calculates the maximum value of each zone of the feature map while the later calculates the average.

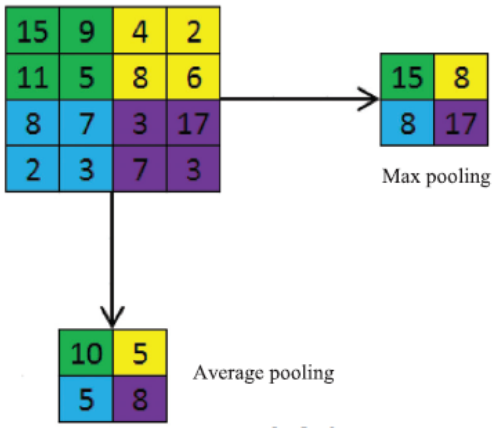


Figure 5: Max pooling layer example [21]

Apart from these two main layers, some others have been lately added to improve the results [20]. The most popular ones are:

- **Activation function**: the right activation function significantly improves the performance of a CNN in certain tasks. The most common function is the Rectified Linear Unit (ReLU) which converts the negative parts to zero and retains the positive values.

- **Loss function**: The purpose of the loss function is to obtain the quantity that a model should seek to minimize during training. In the state-of-the-art analyses, *softmax* and *binary* are the most used ones. The first one is usually used for multiclass problems while the second is used for binary cases.

- **Regularization**: To avoid the so common overfitting issues that NN and particularly CNN can have, regularization has shown it can effectively reduce it. The most used method is the *dropout*.

- **Fully Connected Layer**: this stage performs the same process found in standard ANNs, attempting to generate class scores from the activations. All the neurons from previous layer are connect to every single neuron of the current layer in order to generate a global information output [20].

- **Output**: this layer does not perform any operation; it just receives the output of the fully connected layers which consists of a set of probabilities associated to each class that has been defined.

2.2.2 Architectures

In the last decade, hundreds of new ConvNets have been designed. The number and combination of the previously mentioned layers has no end, however, some of them are well known due to their accuracy and are mostly mentioned in the literature.

These architectures have become deeper and deeper along with the computer technology progress. This section discusses three popular CNN architectures that have been selected and that correspond to probably the best architectures at their time. This work applies the last two.

2.2.2.1 VGG16

VGG is a deep CNN that was proposed by the Visual Geometry Group (VGG) from Oxford University. The findings were the basis of the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) in 2014, where the research team secured first and second position in localisation and classification tracks [29].

VGG16 is composed of 13 convolutional layers, 5 max-pooling layers, and 3 fully connected layers. Therefore, the number of layers having tuning parameters is 16 (13 convolutional layers and 3 fully connected layers). That is the reason why the model name is VGG16. A kernel size of 3x3 is applied to all convolution layers and a max pooling size of 2x2 is used to halve the resolution after each block. This results in an architecture that consists of 138M parameters [29]. The architecture is shown in Figure 6.

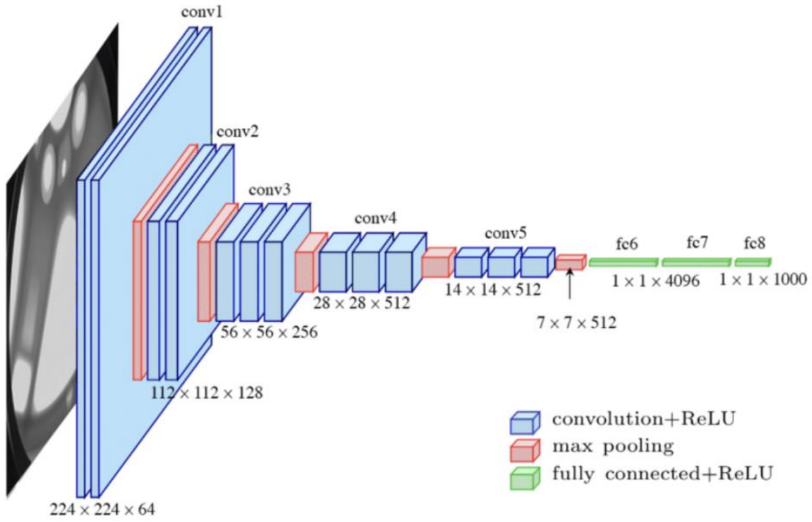


Figure 6: VGG16 architecture [Researchgate.net]

In the conference paper where this method was introduced, architecture effectiveness was measured in terms of validation error. Compared to previous state of the art architectures, VGG proved to have the minimum error in every evaluated case: Top-1 val. error, top-5 val. error and top-5 test error. These results are shown in Figure 7.

Method	top-1 val. error (%)	top-5 val. error (%)	top-5 test error (%)
VGG (2 nets, multi-crop & dense eval.)	**23.7**	**6.8**	**6.8**
VGG (1 net, multi-crop & dense eval.)	24.4	7.1	7.0
VGG (ILSVRC submission, 7 nets, dense eval.)	24.7	7.5	7.3
GoogLeNet (Szegedy et al., 2014) (1 net)	-	7.9	
GoogLeNet (Szegedy et al., 2014) (7 nets)	-	**6.7**	
MSRA (He et al., 2014) (11 nets)	-	-	8.1
MSRA (He et al., 2014) (1 net)	27.9	9.1	9.1
Clarifai (Russakovsky et al., 2014) (multiple nets)	-	-	11.7
Clarifai (Russakovsky et al., 2014) (1 net)	-	-	12.5
Zeiler & Fergus (Zeiler & Fergus, 2013) (6 nets)	36.0	14.7	14.8
Zeiler & Fergus (Zeiler & Fergus, 2013) (1 net)	37.5	16.0	16.1
OverFeat (Sermanet et al., 2014) (7 nets)	34.0	13.2	13.6
OverFeat (Sermanet et al., 2014) (1 net)	35.7	14.2	-
Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)	38.1	16.4	16.4
Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)	40.7	18.2	-

Figure 7: Comparison with the state of the art in ILSVRC classification. VGG16 denoted as VGG [29]

Results of the submitted errors are gathered in row three. However, the VGG research team kept improving the architecture by combining the outputs of several models by averaging their soft-max class posteriors and developed a deeper architecture called VGG-19.

The authors of the paper also proved that the architecture did generalize properly to a wide range of tasks and datasets such as VOC-2012 or Caltech 256.

2.2.2.2 ResNet

ResNet architectures are based on residual learning. The trend until the moment in which this architecture was designed was to create deeper networks each time, that is, to include more layers to the model. However, as the model gets deeper, the training of it becomes more difficult, saturating the accuracy and sometimes even degrading it [30].

Residual learning tries to solve these two issues by subtracting learned features from inputs of the layers. This is done by using shortcut connections, directly connecting inputs of a layer with some other deeper layer (that comes after the one that has been extracted). This process is shown in 8.

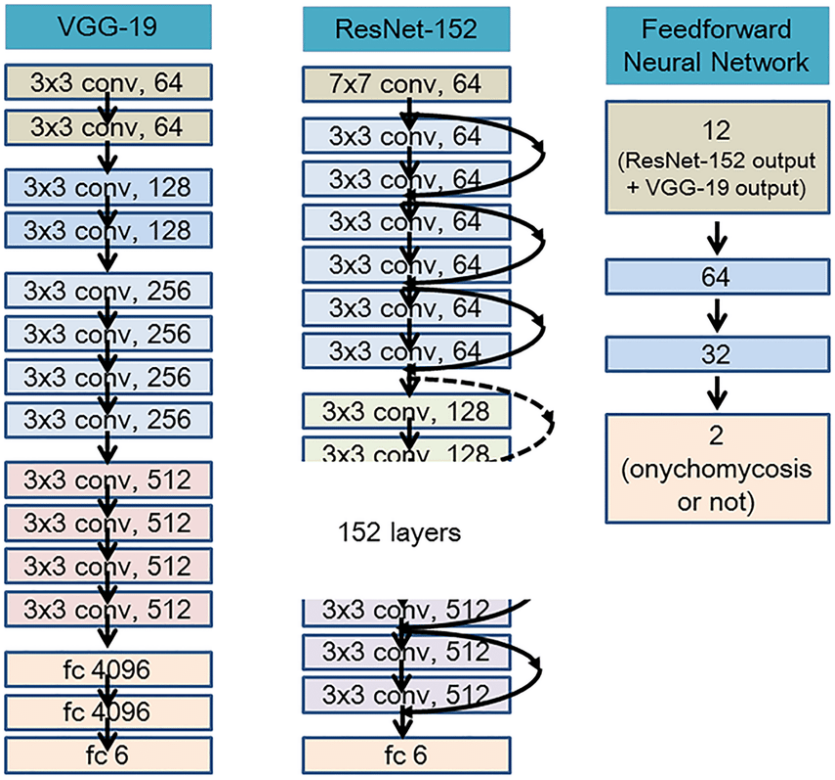


Figure 8: Common CNN vs ResNet [Researchgate.net]

This method has proved that it makes training networks easier, faster and it also solves the problem of degrading accuracy that might come up with very deep CNNs.

Similarly to VGG, there are multiple ResNet architectures available in the literature; probably, the most known one is ResNet-50. Apart from that, ResNet-101 and ResNet-152 are the better known ones. As it can be guessed, the higher the ResNet-X number, the deeper the architecture becomes.

The left-hand table of Figure 9 shows a comparison of error rate between different architectures while the right hand one compares the depth of the models and the number of the parameters of each one.

method	top-1 err.	top-5 err.
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

method			error (%)
Maxout [10]			9.38
NIN [25]			8.81
DSN [24]			8.22
	# layers	# params	
FitNet [35]	19	2.5M	8.39
Highway [42, 43]	19	2.3M	7.54 (7.72±0.16)
Highway [42, 43]	32	1.25M	8.80
ResNet	20	0.27M	8.75
ResNet	32	0.46M	7.51
ResNet	44	0.66M	7.17
ResNet	56	0.85M	6.97
ResNet	110	1.7M	**6.43** (6.61±0.16)
ResNet	1202	19.4M	7.93

Figure 9: Comparison of error rates, number of layers and number of parameters of different architectures based on ImageNet [30]

It can be seen that ResNet outperforms every architecture shown in the tables. The most remarkable thing is that this is achieved with a relatively low number of parameters. Comparing ResNet-32 to VGG16, it has almost the same error percentage just using 0.45M parameters, this is, 300 times less parameters.

ResNet architecture has been widely used in railway applications and track fault detections [11]–[13]. However, in the papers found in the literature and as mentioned above, they were mainly related to classification or fault detection on railway fasteners.

2.2.2.3 EfficientNet

In the International Conference on machine Learning, held in California in 2019, Brain Team from Google research presented EfficientNet [31]. In this paper, T. Mingxing studies model scalability and propose a new scaling method that uniformly scales all dimensions of depth, width and resolution using what they defined as *compound coefficient*.

The width of the model refers to the number of channels in the layers; the depth refers to the number of layers in the model, and the resolution refers to the input image size for the model. This technique is illustrated in Figure 10.

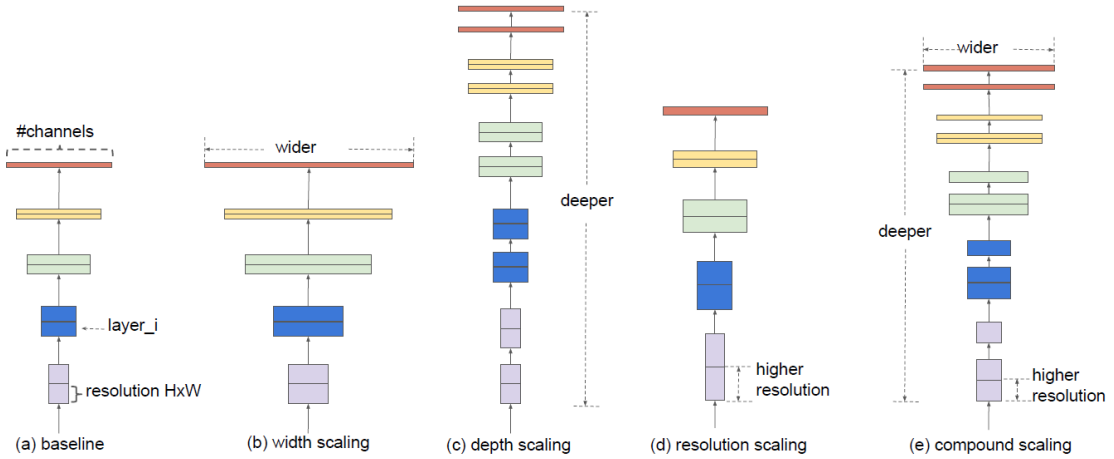


Figure 10: Model scaling illustration [31]

The proposed compound scaling method uses a compound coefficient φ to uniformly scale the network width, depth and resolution in a principal way [31]:

$$Depth: d = \alpha^{\phi}$$

$$Width: w = \beta^{\phi} \qquad \textbf{Equation 1}$$

$$Resolution: r = \gamma^{\phi}$$

Where,

$$\alpha * \beta^2 * \gamma^2 \approx 2$$

$$\textbf{Equation 2}$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

The compound scaling can be used with any ConvNet; however, the Brain Team decided to design their own baseline network using the Neural Architectural Search (NAS). They set a multi-objective search where they tried to optimize both, accuracy and FLOPS. The search resulted in an efficient network or architecture named EfficientNet-B0 which is shown in Figure 11.

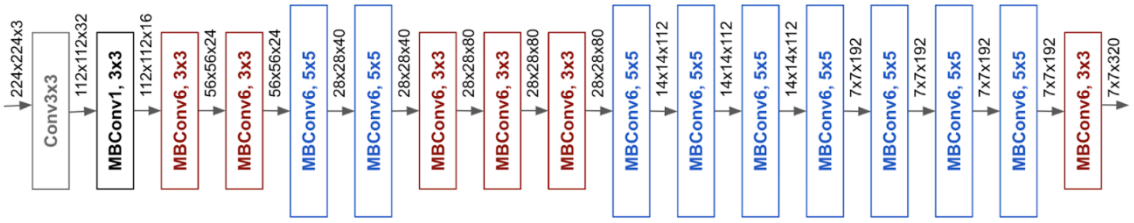
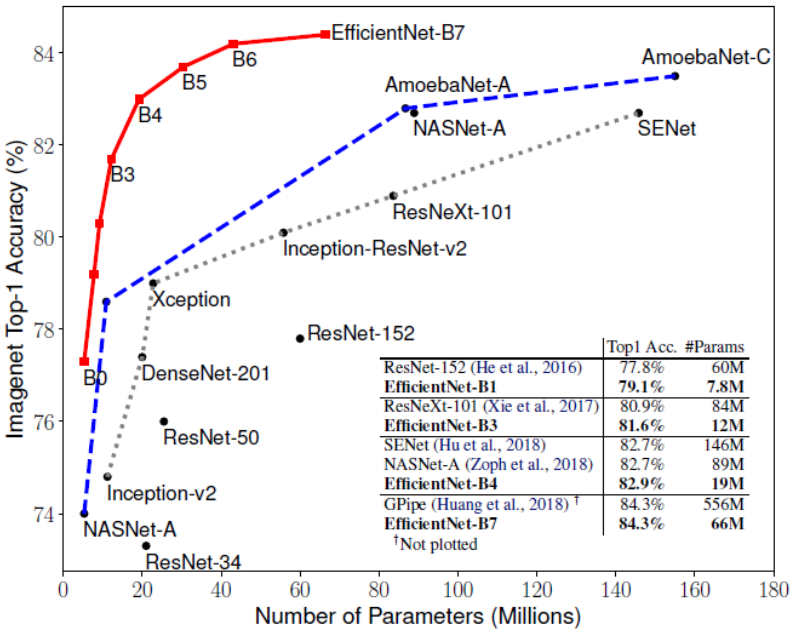


Figure 11: Architecture of baseline network EfficientNet-B0 [32]

The architecture uses mobile inverted bottleneck convolutions (MBConv), which are a type of residual block applied for image models that uses an inverted structure for efficiency reasons [33]. The researches then scaled up this baseline network to obtain a family of deep learning models.

This resulted in an EfficientNet family that goes from B0 to B7. Each of the families was originally run with a different size of image, corresponding the smallest images to ConvNet family B0 and the largest to B7.

Figure 12 shows a comparison of the different EfficientNet (B1-B7) and other CNN that were performing best at that moment.



	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

Figure 12: Comparison of EfficientNet accuracy vs other architectures on ImageNet dataset [31]

It can also be seen their low computational cost, in number of parameters required. The author also includes a comparison in FLOPs of each model, going from 0.39B to 37B (from EfficientNet B0 to B7). The number of FLOPs is compared to other available architecture and EfficientNet always proves to have a lower computational cost. For the case of ResNet-152, for example, EfficientNet-B0 shows similar accuracy results but with one sixteenth of FLOPS (or 7.6 times less parameters) [31].

Even if the model was developed and initially tested on ImageNet, the Efficient architecture family was also tested against other datasets such as CIFAR-10, CIFAR-100 or Birdsnap.

Finally, it is important to mention that a higher level of EfficientNet architecture does not always imply getting a higher accuracy. This fact was captured in [31] and is illustrated in Figure 13, where it is shown that CIFAR100 EfficientNet-B5 provided higher accuracy than EfficientNet-B6, for instance.

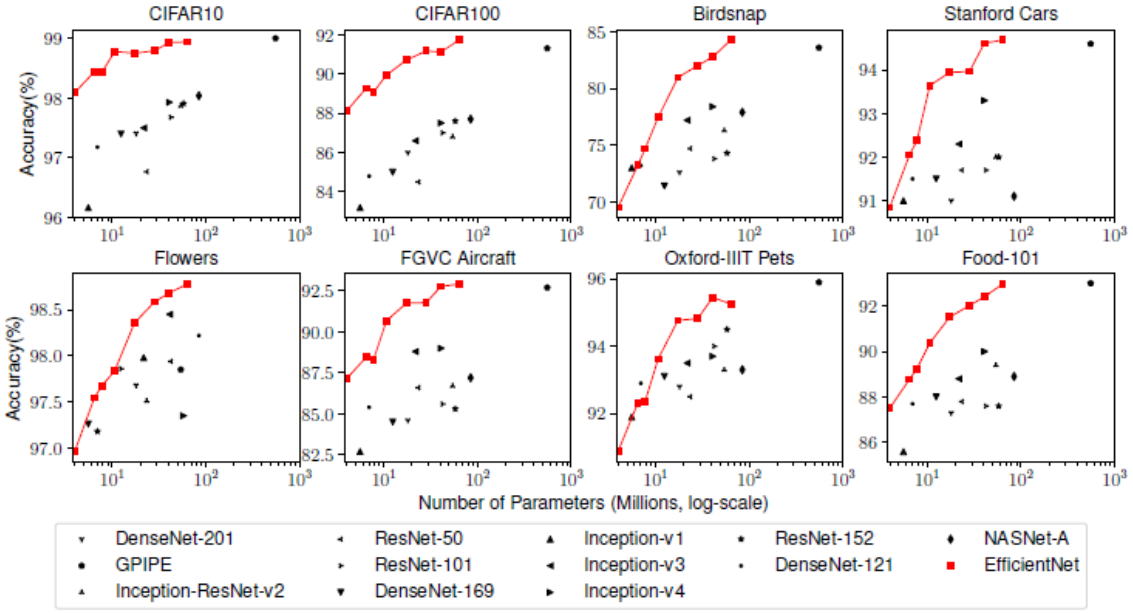


Figure 13: Model parameters vs Transfer learning accuracy pretrained on ImageNet [12]

Since the development of this architecture, it has been widely applied in multiple disciplines. The literature review shows that it is especially popular in medicine, where it has been used for all sort of analysis, from detecting brain tumours [34] to helping detect Covid-19 [25], [26], [35], [36].

2.3 Fault detection

Once the image classification is performed and labelled as faulty, it is necessary to further find where the fault is detected or located in the image. For that reason, an object detection, in this case, fault detection is usually applied after the image classification process.

Visually locating the fault in the image helps understand what the trained model is detecting as fault and what not. It can pave the way for knowing if the new dataset needs to be included in the training model so it can properly identify the faults.

In this case, as the trained models with their corresponding weights will be already available, the same image neural classifier has been used to locate the fault. In other words, the CNN image classification has been turned into an object detector. The selected method is based on [37] and has been modified to match the current problem.

2.3.1 Main algorithm

To understand how the object detection works, it is necessary to understand the three different computer vision techniques that the algorithm applies:

- **Image pyramid:** To detected faults of different sizes in the image
- **Sliding windows:** To make sure the whole image is scanned
- **Non maxima suppression (NMS):** To collapse weak and overlapping bounding boxes

First, an image pyramid of the input image is created. Image pyramids are known as multi-scale representations of images [37]. The objective of this step is to progressively subsample the image so that it allows to find objects at different scales. The image is subsampled until a stop criterion is met.

Secondly, a fixed-sized rectangle slides from left to right and from top to bottom within an image generating multiple Regions Of Interests (ROI). This is

known as sliding windows technique. Each of the ROI is passed through the image classifier, obtaining a classification of it.

Finally, when obtaining the results of the image classifier, the detector will typically produce several overlapping bounding boxes. That is, as the sliding window gets closer to the fault, it returns higher and higher probabilities of detecting it. However, there is usually *one* object or defect, so the NMS is applied.

The goal of NMS is to obtain only one window when identifying a given fault or object. This will correspond to the precise local maximum of the predicted values, ideally obtaining one detection per detected object or fault [38]. NMS uses a threshold which determines that two frames overlapped means that the detected object is identified as one and the rest of the boxes can be removed.

Figure 14 shows an example of how NMS works on a car object detection case.

Figure 14: NMS [39]

3 Dataset analysis, pre-processing and data augmentation

The dataset for this work has been extracted from Kaggel website [40] , an online community of data scientists and machine learning practitioners which is a subsidiary of Google. It allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges [41].

In this case, the dataset consist of a total of 300 railway track images (downloaded from Ref. [42]), which have already been classified into ‘Defective’ or ‘Non defective’. Out of those 300, 87 contain defective information, this is a 29% of the images.

In this section, some of the images that make the dataset will be analysed in order to get familiarized with the type of images and faults that might appear.

3.1 Dataset analysis

In every image classification problem, the first step is to understand the data to work with. This is crucial as it will directly affect the steps that will need to be taken on the pre-processing. For that, an example of 9 randomly selected figures showing both, defective and non-defective cases, are shown in Figure 15.

Figure 15: Example of 9 randomly selected images showing defective and non-defective cases

The first thing that can be noted in the images shown in Figure 15 is that the sizes of the images are not the same. This is generally an issue when running most deep learning algorithms. There is an option of scaling the pictures although doing so the quality and the ‘faults’ may be distorted and, therefore, can cause training issues. This set of railway track fault images contained images of the same size (4000x3000), 11 of which were 90 degrees flipped.

Another noteworthy aspect of the images is the differences between them in terms of angle from which the photo is taken, the distance to the railway and orientation. The small dataset, combined with the differences between the images, makes the classification problem more challenging.

According to the Indian Railway Institute of Civil Engineering, railway track maintenance is performed periodically. To reduce man labour, maintenance vehicles such the ones shown in Figure 16 are used [43]. If a more professional

pictures wanted to be taken for the kind of analysis proposed in this work, camaras should be installed in them and this way, all pictures would have same angles and sizes.

Figure 16: Maintenance vehicle. Multipurpose tamper (MPT) [43]

Finally, and even if the individual pictures of Figure 15 are small, the defects on the tracks are different from one picture to another. To show this, some of the images have been zoomed and are shown in Figure 17:

Figure 17: Examples of fault types on railway tracks.

From left to right, it looks like the faults are the following: crack on track, excessively worn-out track at some areas, misalignment and maybe a larger separation between tracks than allowed.

The available data does not differentiate between different faults and so, this work will classify between defective and non-defective ones. To identify every type of fault, an expert would have to determine the number of different faults in an image and to label them. But the limited number of available image and the lack of information makes this identification not feasible.

3.2 Dataset partitioning

As the available dataset is very small for a deep learning image classification problem, the partition of the dataset is key for the classification, mainly for the training process.

When working with small datasets, the following issues might arise:

- Overfitting. With small datasets it becomes very easy to overfit when trying to achieve good accuracy. This means that there is a high risk of generating a model which would not generalize well to the unseen data and, therefore, could not classify rightly new images.

- Not been capable of learning enough features. As shown in section 3.1, there are different types of defects that railway tracks may have. Not having enough training images means that there could be very few images representing a specific defect. This implies that the training process would not be able to learn and detect all the required features and would result in not being able to detect some of the defects.

- Incorrect data partitioning. Usually data is partitioned into training (often involving training and validation) and testing. The first two sets are used to train the model and the latter to verify it works properly.
  If all the images that represent a given defect are in the training set, the model will never be able to identify that fault as it has never seen it before. So even if the validation accuracy reached 100%, test accuracy could end up been 0% if an incorrect data partitioning is made.

The most common way to address the two first issues highlighted above is by applying data augmentation. However, as it will be explained in the following section, augmenting data may not resolve every issue.

This section will focus on how to partition the available dataset. Attending to the literature, there is no partition that fits every problem. This means that every problem will have an optimum partition which will differ from problem to problem.

Consensus on the percentages that training, validation and testing datasets should have are shown below. In all the examples found in the literature, training dataset was always between 70-80% of the total data.

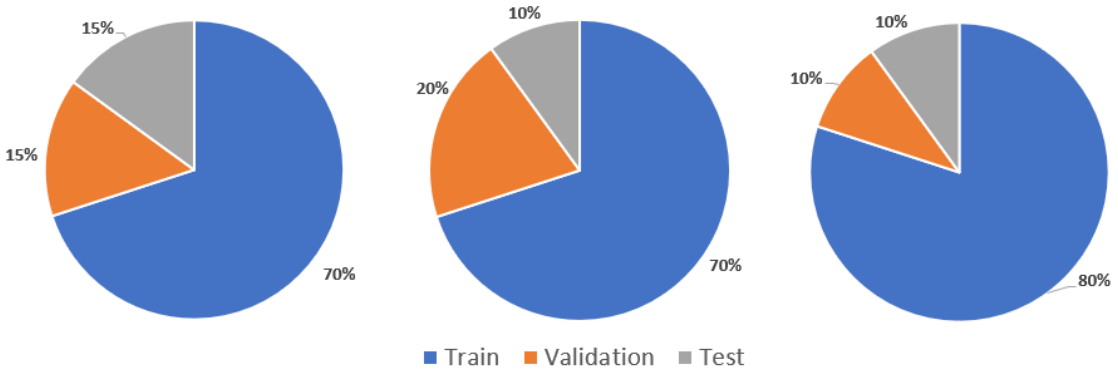


Figure 18: Most common data partition percentages for test, validation and test

For this problem, the middle option will be applied, this is, 70% of the data will be used for training, 20% for validation of the model and 10% to test it.

It is important to keep in mind that alternative methods exist to address data partitioning. The common ones are K-Fold and stratified K-Fold cross validation. Even if this strategy will not be used in this work, it will be briefly explained for academic reasons.

K-Fold is a cross-validation method that divides the dataset into k folds. The general procedure of this method is the following:

1- Dataset is shuffled so that the selection is random
2- Dataset is divided into k groups
3- For each group:
    a. Take the generated group as training data
    b. Take the remaining dataset as testing data
    c. Fit/train de model and evaluate test results
    d. Keep evaluation results

The main advantage of this method is that it gives an idea about how the model will generalize to an unknown dataset. However, the computational cost of the method is high and, therefore, it requires high training time.

Stratified K-Fold is an extension of the K-Fold method. This method advantage is that each fold of the dataset has the same proportion of observations with a given label. This way, it makes sure that the training dataset is not formed only of a single data label (it makes sure that the random selection does not select only defective cases for training, for example).

This technique has been discarded due to the small dataset available. In this case, the amount of data is very limited, which means that data augmentation is necessary. Data augmentations and K-Fold cross validation (or Stratified) are not compatible, as it could provide distorted data for having the same (augmented) image in the training set and the validation or test set.

3.3 Pre-Processing and Data Augmentation

One of the main advantages of deep learning is that image features are automatically extracted without having to manually define each of them one by one. Deep learning models can detect these features no matter the position on the images. These two characteristics are especially useful when the features to be extracted are unknown, or not so easily identified by sight.

Mainly for these two reasons, in deep learning the pre-processing of the dataset is usually related to data augmentation. However, as with every image analysis, some pre-processing may be necessary. In the following two subsections, image pre-processing and data augmentation will be analysed.

3.3.1 Pre processing

As it has been mentioned, pre-processing for image classification with deep learning is minor. This pre-processing is not aimed at maximizing feature extraction but to help with data processing. Two main processes have been applied in this project:

3.3.1.1 Data normalization:

Data normalization is the first step in any image classification problem to ensure that each pixel has a similar data distribution. Doing so, it makes convergence faster while training the network.

In this case, as images provided are RGB, there are three different layers per image with pixel values in the range of [0,255]. This means that to get a range of [0,1], every pixel has to be divided by 255. Once this is performed, the data would look like a Gaussian curve centred in zero.

3.3.1.2 Image resizing:

In general, the bigger the images (resolution or pixels per image), the more computational operations per layer are performed, requiring more memory and computational cost. For this reason, the analysed architectures, and most of the architectures found in the literature, resize their images to different resolutions. It is important to mention that each architecture has been developed for a specific resolution, being in most cases square.

After analysing the images, all of them have a size of 3000x4000. Out of the 300 images, 289 of them are in a landscape orientation and the remaining ones (11) in vertical orientation. This is not an issue as later, with data augmentation, images will be rotated to generate new data.

As briefly mentioned above, in the deep learning image classification literature, authors and architectures usually work with images that are square, vertical and horizontal axis with the same number of pixels. If an image is resized without maintaining the aspect ratio, features might be distorted.

To avoid possible information alteration, images will be cropped so that they are square. For this work (and as visually checked in multiple images), it will be assumed that all the relevant data on the images are centred. Therefore, the centre 3000 pixels will be kept minimizing information loss.

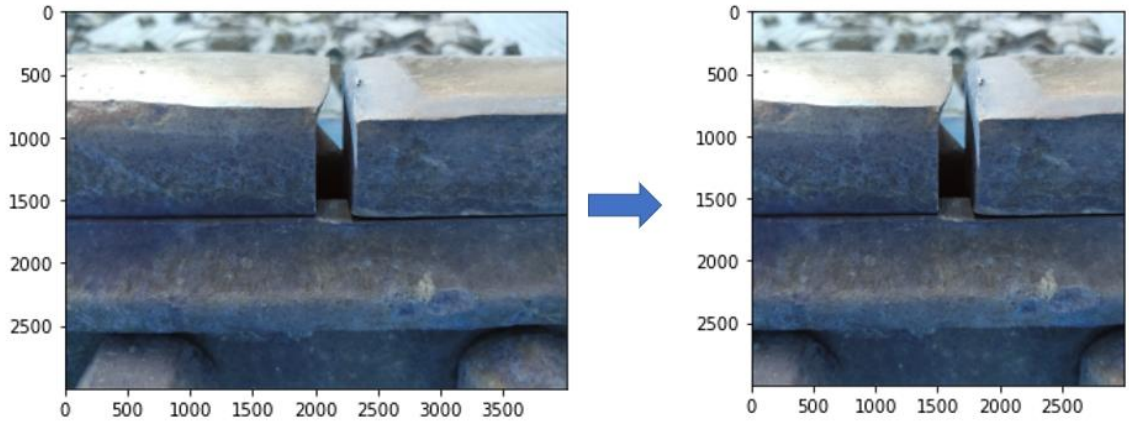


Figure 19: Cropped image to make it square

It is important to mention that reorientating the images will not impact the deep network as it does not work with full images, but rather with a fixed window/filter that slides over the image.

3.3.2 Data augmentation

Data Augmentation covers different techniques with the aim of enhancing the size and quality of training datasets such that better deep learning models can be built [44], [45].

For this project, the Keras [46] open source library for Python has been used. The library provides an *ImageDataGenerator* class that facilitates the implementation of geometric augmentations. However, the input arguments of the class need to be carefully chosen in order to get a useful data augmentation set.

In this section, an analysis of the data and the selected data transformation via *ImageDataGenerator* will be performed. The number of cases and images that can be generated are infinite, and it is not feasible to study the effect on the results of each one of them independently and combined. The selected values have been chosen attending to cases observed in the open literature.

Before detailing the specifics of the performed augmentation, It is important to clarify what the meaning of *augmentation* is. When *ImageDataGenerator* is used, means that the model is using a different

transformation of the images in each epoch. Therefore, if for example we used 10 epochs, we would be using 10 different versions of each original image in training, this is 10x300 = 3000 different images in the whole training instead of the original 3000.

As it will be explained in section 4.2, a maximum of 40 epochs have been set (if no early stopping is activated), this means that a total of 40 x 300 = 1200 images will be used for training.

3.3.2.1 Position and orientation shift

One way to generate more images from the existing ones is by moving around the pixels inside the 3000x3000 pixel frame. The image data generator allows many of these transformations. The mains ones are the following:

- Width and height shift: Both, width and height shift are randomly generated from the specified shift range. In this case, a 20% shift has been defined.

- Horizontal and vertical flips: Randomly flips inputs horizontally and vertically. For this project, both features have been set to True.

- Rotation: Keras randomly selects a rotation degree from the stablished limit. In this case, the limit has been set to 40 degrees which means that images will be rotated in the range of [0,40] º.

An interesting matter is what happens to the area of the images from which pixels have been moved. In this sense, Keras allows to choose between different alternatives. After quickly running a few models to try the differences between them, no clear difference was found in the results. Therefore, the default fill mode “nearest” has been used.

A result of selecting these features is shown in Figure 20, where 9 images have been randomly generated using different combinations of the above-mentioned pre-processing methods. If the images of Figure 20 were ordered from 1 to 9, starting from top to bottom and from left to right, It can be seen that all but number 4 and 7 have been rotated.

On the other hand, all of the images show traces of the “nearest” fill method. Attending to the trace, it can be deduced what operations the image has undergone. For example, looking at image 5 (the one in the centre), the traces are on the left and bottom part of it. This means that apart from any flip that it underwent, it has also been shifted in width to the right and in height to the top. The rotation and flip effect can easily be detected in any of them too.

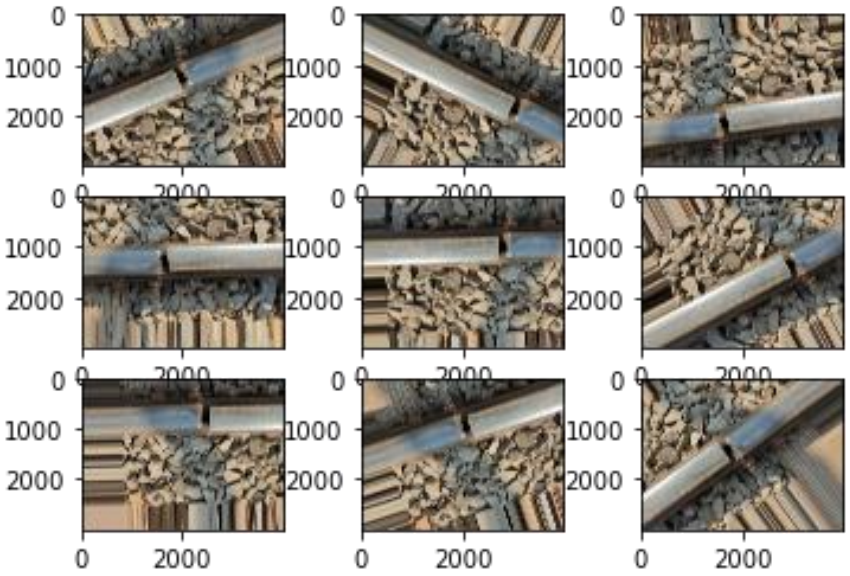


Figure 20: Augmented images applying position and rotation shifts

3.3.2.2 Shear and zoom

Another option to generated new data is to apply a shear intensity and a random zoom. The zoom operator zooms in and out the image keeping the original size while shear means that the image will be distorted along an axis, mostly to create or rectify the perception angles. This is the best way to make a computer *see* things from different angles like humans do.

To get a better understanding of what this last feature means, an example has been generated assuming a 0.2 zoom range and a 10-degree shear. As the effect of this data augmentation technique might not be so visible, 9 randomly generated images using different combinations of the above-mentioned pre-processing methods have been generated and are shown in Figure 21.

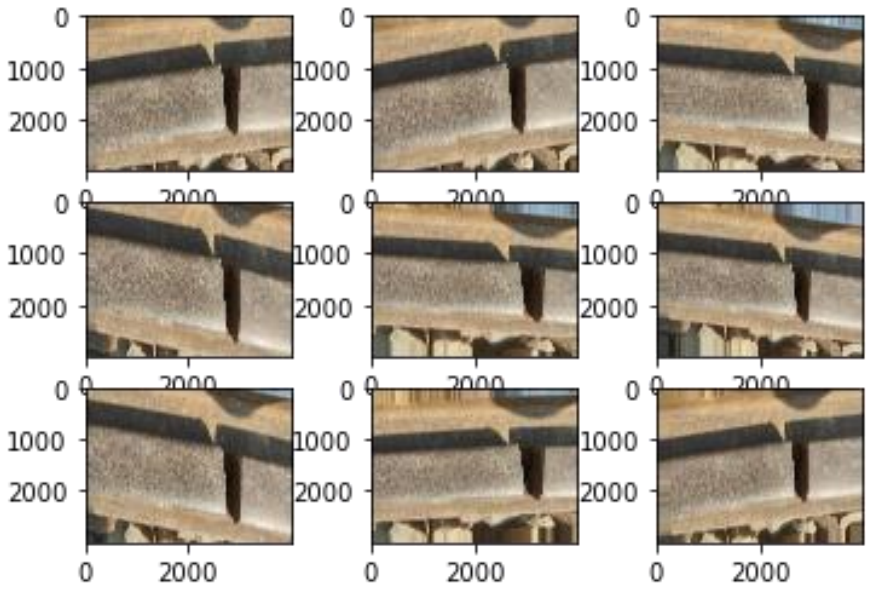


Figure 21: Augmented images applying zoom and shear intensity

However, including a high shear intensity value is not always desirable. Distorting the image might generate a modification of the defects that are trying to be detected. This is an issue because the results of such augmentation would only generate noise in the dataset. For that reason, this value is usually kept small.

For this project, a zoom range and a shear intensity of 0.2 has been selected.

3.3.2.3 Brightness

ImageDataGenerator also allows to generate new images by modifying their brightness. Before applying any brightness change, an analysis on the already existing brightness has been performed.

The graphs shown in Figure 22 are four randomly selected images and their brightness histogram per colour channel (red, green and blue).

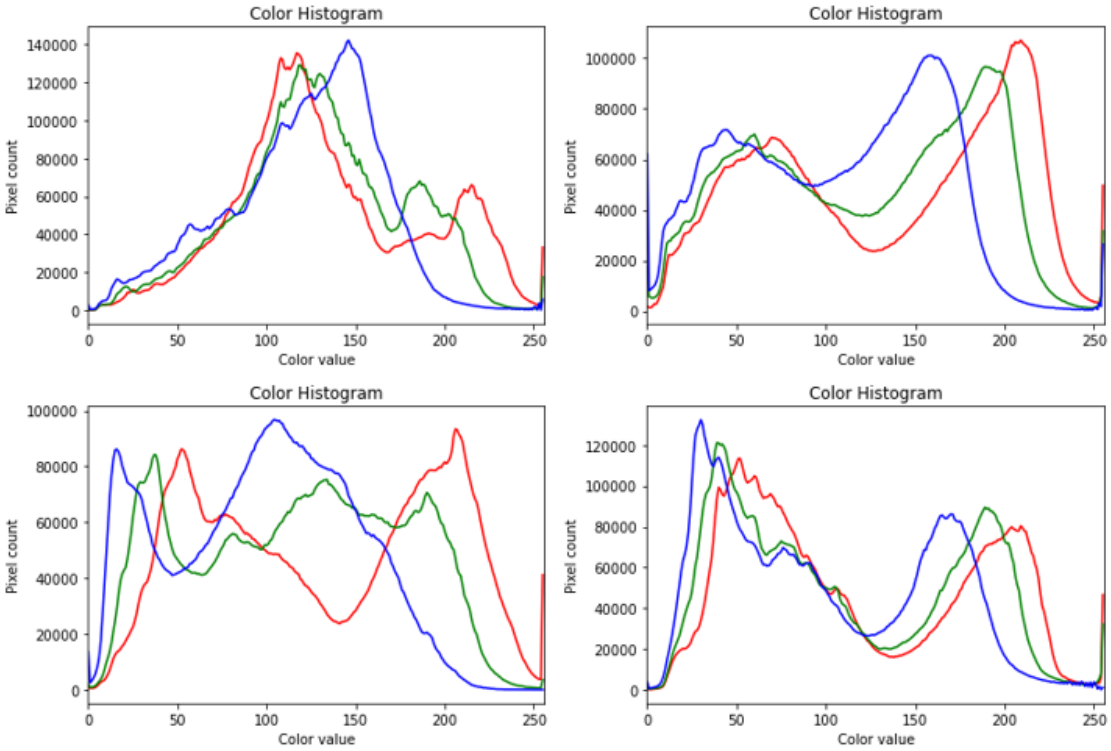


Figure 22: Brightness analysis of 4 randomly selected images.

As it can be seen, there is no clear trend on the brightness between images. If there was a trend, brightness modification could be a powerful tool as it would help the model to be trained for cases when there is not much light or the opposite. This would help making the model more general, which in this case it already is.

Thus, as there is no clear trend, the brightness modification on the data augmentation has been neglected.

4 Methodology

Following the analysis performed in Section 2.1 on different ConvNet architectures, ResNet and EfficientNet architecture have been chosen as the CNN methods for this case study.

On the one hand, ResNet has been selected because it has been widely applied in many different industry applications. Moreover, it has been applied to railway track fault detection, particularly fasteners, with positive results.

On the other hand, the main reason for choosing EfficientNet was the high accuracy rate compared to the rest of the state-of-the-art methods. Indeed, it presents a relatively small number of parameter and FLOPS compared to the rest of the analysed options, i.e., less computational power will be required to process the model, key aspect when a home laptop is used.

This section will first explain the analysed scenario, then the model setup and will conclude with the applied evaluation criteria.

4.1 Analysed scenarios

As each architecture has a different layer configuration, different scenarios have been analysed. However, the scenarios have been carefully selected so that the architectures can be compared to each other.

As it will be explained in next sections, the way in which the model has been setup implies that training images are randomly input in batches. This means that every time the model is run it will give different results. Therefore, each of the analysed scenarios explained in this section has been run several times. To be more precise, each model has been run a total of five times.

4.1.1 ResNet cases

Figure 9 showed that multiple families of ResNet architecture are available, though mostly ResNet-50 and ResNet-101 are used. Therefore, those two architectures have been selected together with ResNet-152, as the latter has been proved to perform better than the rest on ImageNet dataset [30].

These architectures were developed for a 224x224 image size. Nevertheless, with the objective of capturing the effect of resizing the 3000x3000 pictures into these values, and in order to be able to compare them, additional analyses have been run with 300x300 resolution. This simulations with such resolution have been named as *Processed*.

	Resolution	
Base model	**Original**	**Processed**
ResNet-50	224	300
ResNet-101	224	300
ResNet-152	224	300

Table 1: Run cases with ResNet architecture

The objective of analysing these scenarios is to obtain the accuracy when using different ResNet architectures as well as determining the effect of image input size.

4.1.2 EfficientNet cases

As it has been previously explained, going to a higher EfficientNet family does not always imply getting better results. Therefore, in this case, EfficientNet B0 to B7 architectures have been modelled and analysed to perform a comparison of them to obtain the most suitable architecture for this case study.

On the other hand, for B0 to B7 base models the input images are different. Table 2 shows the base model original image resolution or size of each of the architectures [46]:

However, as explained in the pre-processing section, reducing the original image size from 3000x3000 to the ones shown in Table 2 implies a loss of information. Therefore, an additional case has been run for each of the architectures where the resolution was fixed to 300. This resulted in the following studied cases:

	Resolution	
Base model	Original	Processed
EfficientNetB0	224	300
EfficientNetB1	240	300
EfficientNetB2	260	300
EfficientNetB3	300	300
EfficientNetB4	380	300
EfficientNetB5	456	300
EfficientNetB6	528	300
EfficientNetB7	600	300

Table 2: Run cases with EfficientNet architecture

One of the objectives of this last comparison is to learn whether in this case is worth working with higher resolution images or not. This is key as a higher resolution is directly linked to a larger computational cost of the training.

4.2 Model Setup

For each of the analysed scenarios, the same model setup process has been used. This process is shown in Figure 23 and will be analysed in detail in this section.

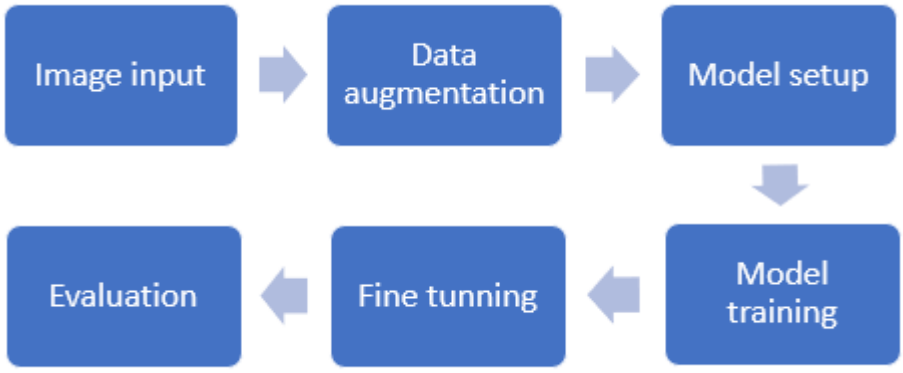


Figure 23: Model setup process

- **Image input:** The dataset described in the pre-processing section are the inputs to this process. Loading them to the model would require more

memory than available, therefore, the *flowfromdirectory* command from python has been use.

- **Data augmentation:** As previously mentioned, due to the small dataset available, the data were augmented. The following additional settings have been activated to maximize feature extraction:
    - Batch size: It has been set to 16
    - Image shuffle: allowed for the training set dataset
    - Class mode: Defined as binary.

- **Model setup**: The model has been built in Keras, which already incorporates a package of ResNet and EfficientNet architectures. As a first iteration, and considering the small dataset, ImageNet pretrained weights have been used.

  To make it compatible with the analysed case, additional layers have been added to the model. Each architecture required different additional layers explained below:

  ResNet
    - Flatten: It flattens the output to reduce dimension
    - Dense layer: A 1 unit layer with a sigmoid activation. This implies that 0 will be the classification for defective cases and 1 for non-defective (or the other way around).

  EfficientNet
    - Pooling: A global 2D average pooling to reduce dimension
    - Dense layer: A 128 units densely connected neural network
    - Dense layer: A 1 unit layer with a sigmoid activation. This implies that 0 will the output for defective cases and 1 for non-defective (or the other way around)

  On the other hand, the model has been compiled with the following specifications:

- Losses: As this is a binary classification problem, the binary-cross entropy has been selected. This computes the cross-entropy loss between true labels and predicted labels
- Optimizer: Among the available options, Adam algorithm has been selected with a default learning rate. The method uses stochastic gradient descent to optimize results
- Metrics: It has been set to Accuracy. It calculates how often outputs match the labels.

- **Model training**: Finally, 40 epochs have been used for training, with two callbacks:
  - Early stopping: in case after 15 epochs the loss no longer decreases. The objective is to avoid running more epochs if the model has already converged.
  - ModelCheckpoint: Monitoring accuracy, it makes sure that the best model weights are saved. This callback avoids overfitting issues.

- **Fine tuning:** a first training process has been performed using already existing pre-trained weights on ImageNet. With the aim of improving accuracy, a fine-tuning step has been included.
  For the fine tuning, the learning rate of the optimizer has been fixed to 10^{-6} and some layers have been defined as trainable. For ResNet, the first 25 layers have set to trainable while in Efficient the first block layers have been defined as trainable. It is important to highlight that in every EfficientNet architecture, the number of layers that corresponds to the first block may vary, in general, the higher the EfficientNet family, the larger the number of layers that needed to be blocked.

- **Evaluation:** Once the model has been trained, the weights of the model with highest accuracy have been loaded to perform the validation. To do so, the *predict* python function has been used.

As this is a binary problem, a threshold of 0.6 has been set to determine whether the predicted value is fault or not.

The parameters that have been modified at each iteration are: image size, efficientNetB0 to EfficientNet B7, and the trainable layers (as every family member has a slightly different number of layers).

The model has been coded with Python Keras, where some deep learning algorithms and functions are already embedded and validated. Regarding the processor, the models have been run on a i7 10th generation Intel processor, as a graphic card to accelerate the computational time was not available.

4.2.1 Fault detection parameters

Regarding fault detection, the following parameters have been used for the image pyramid, sliding windows and NMS threshold:

- Pyramid scale: It controls how much the image is resized. Therefore, smaller values yield to bigger ROIs. In this case, it has been set to 1.5

- Step size: Determines how many pixels are skipped on the sliding windows. Looping over pixel would imply higher computational cost so skipping some pixels is necessary to make the analysis feasible in time. This value has been set to 30 pixels.

- Minimum ROI size: It determines the minimum size of the ROI. This is crucial as the algorithm requires a threshold in order not to be decreasing the size of the image indefinitely. In this case, it has been limited to 50x50.

On the other hand, the threshold for every output performed for the ROIs has been set to 60%. To those ROIs above that threshold, NMS algorithm has been applied.

4.3 Evaluation criteria

There are multiple ways in which the performance of a deep learning model can be checked. This section describes the most common ones, as well as the evaluation criteria that will be used in this case study.

Confusion Matrix

The confusion matrix is a visual representation of the output results vs the real values. The composition of this matrix is shown in Figure 24:

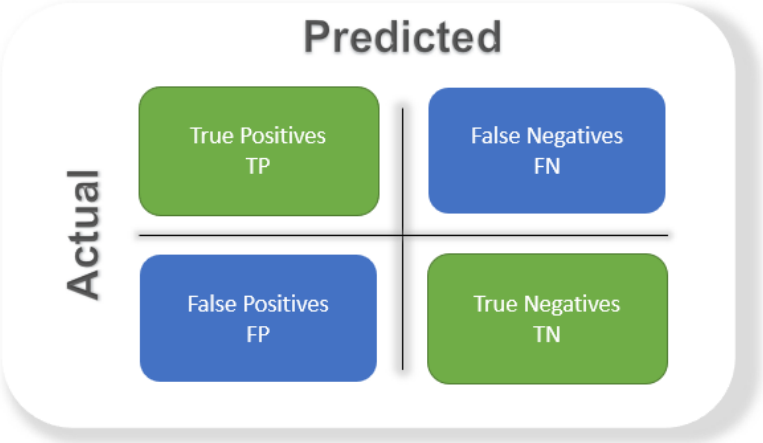


Figure 24: Confusion matrix composition

For this study case, any output with a probability of being defective over 60% has been considered as defective.

4.3.1 Accuracy

This is probably the most used metric to judge the performance of a model. It is calculated as follows:

$$Acc. = \frac{TP + TN}{TP + FP + TN + FN} \qquad \textbf{Equation 3}$$

The main problem with using accuracy as performance evaluator metric is that it can become an unclear indicator when the classes are imbalanced. For example, if 90/100 images where non-defective, it could provide a 90% accuracy without detecting even 1 defective image. However, if the established threshold

exceeds the percentage of the non-defective images, then it can be assured that the trained model is at least detecting one of them.

All the analysed scenarios surpass the 100 simulations; therefore, a rapid performance indicator is needed. Moreover, the study case dataset is relatively small, which means that small changes on classification can cause a great impact on these metrics.

In this case, a validation accuracy threshold of 80% has been stablished for the initial analysis. Knowing that this indicator is not the most appropriate one, the following indicators have been also used with the models that achieved highest accuracy.

4.3.2 F1 Score

The F1 score is also very popular as performance indicator. To understand it, it is necessary first to define precision and sensitivity.

Precision is defined as the percentage of positive instances out of the total predictive instances. The equation that evaluates precision is the following:

$$P = \frac{TP}{TP + FP} \qquad \textbf{Equation 4}$$

On the other hand, sensitivity is defined as the percentage of positive instances out of the total actual positive instances. The equation that evaluates sensitivity is shown below:

$$S = \frac{TP}{TP + FN} \qquad \textbf{Equation 5}$$

F1 score is the harmonic mean of both. Equation 6 illustrates how it is calculated.

$$F1 = \frac{2}{\frac{1}{P} + \frac{a}{S}} = \frac{2 * P * s}{P + S} \qquad \textbf{Equation 6}$$

Attending to this definition, it can be seen that due to the product in the numerator, if sensitivity or precision go down, the final F1 score is directly

affected. Therefore, a model will perform satisfactorily if the positive predicted values are positive (if it is precise) and if it does not miss positives and predicts them negative (Sensitivity). Consequently, the higher the F1 score, the better the performance of the model.

To ensure that F1 score is consistent, its value will be obtained for different detection thresholds (50% onwards) on the best performing models of the analysed scenarios.

5 Discussion of the results

This chapter gathers the results and discussions of the performed studies.

5.1 ResNet

This section focuses on the results obtained following the procedure defined in section 4.2 and running the case scenarios specified in section 4.1.1.

Table 3 shows the average and top 1 validation and testing accuracy results for ResNet-50, ResNet-101 and ResNet-152 architectures for original resolution and processed resolution. It also includes the number of parameters and FLOPS that corresponds to each model. Note that these two numbers refer to the total amount, including the first training and the fine tuning.

			Original resolution			Processed resolution			
Arch.	**Score**	**Parameters**	**Val. Acc.**	**Test Acc.**	**FLOPS**	**Val. Acc.**	**Test Acc.**	**FLOPS**	**FLOPS Diff.**
ResNet-50	Avg	4.76E+07	73.1%	69.9%	1.55E+10	71.7%	67.4%	2.90E+10	-47%
	Top 1		75.7%	76.2%		72.9%	71.4%		
ResNet-101	Avg	8.55E+07	73.1%	65.3%	3.04E+10	72.7%	67.2%	5.64E+10	-46%
	Top 1		75.7%	76.2%		**77.1%**	76.2%		
ResNet-152	Avg	1.17E+08	72.6%	67.2%	4.53E+10	73.4%	66.5%	8.38E+10	-46%
	Top 1		74.3%	76.2%		77.1%	71.4%		

Table 3: Results with ResNet simulation models

5.1.1 Number of parameters and FLOPS

One of the aspects to analyse is the number of parameters and the FLOPS. The latter is usually used as a measure of the computational cost required to solve a problem. This means that both parameters will have a strong correlation as the higher the number of parameters, the more computational power it will require.

Figure 25 illustrates the difference in number of parameters and FLOPS for the analysed scenarios. As expected, the deeper the network, the higher the number of parameters and the FLOPS.

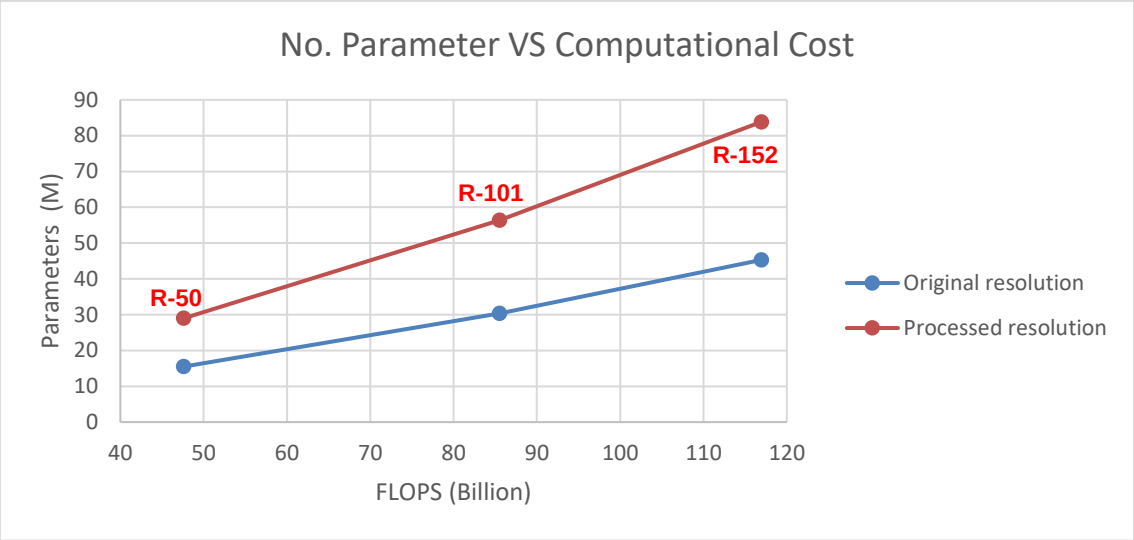


Figure 25: Comparison of number of parameters and FLOPS between the original and processed resolution with ResNet

5.1.2 Validation accuracy

Another criterion to evaluate the efficiency of a model is the validation accuracy. This figure becomes especially important in this case, as the number of images available is relatively small.

As it can be seen in Table 3, a maximum validation accuracy of 77.1% is achieved with both, Resnet-101 and ResNet-152 architecture with the processed resolution cases.

Regarding the highest average validation accuracy, ResNet-152 is also the best, achieving 73.4% precision. Attending to the original resolution results, ResNet-50 and ResNet-101 come close with a 73.1%, achieving this value with a less deep CNN and fewer FLOPS.

Nevertheless, attending to the obtained validation accuracy and following the evaluation criteria defined in section 4.3, the obtained values are below the threshold of 80%, therefore, the model is not considered good enough for this case study.

5.1.3 Test accuracy

Even if the validation dataset accuracy is already an indicator of the performance of the trained model, a small dataset of 21 images in total (considering both, defective and non-defective cases) where kept to run some analysis.

Table 3 also shows that the test accuracy is similar to the validation result. Best test accuracy obtained is 76.2% for every ResNet architecture with original resolution. However, the average for all these cases remained below 70%, getting a minimum of 65.3% for ResNet-101 architecture.

5.1.4 Others

Apart from the parameters analysed above, a few remarkable things were learned during the analysis.

First, as mentioned earlier, a difference on the results was found between iterations (results of each iteration are included in Appendix A.1). As previously explained, this is caused by the random data augmentation and batch selection during the training. For this reason, each model was run five times.

To analyse the effect of this variation, standard deviation has been calculated on every scenario for validation and test accuracy. Results are shown in Table 4.

	Original		Processed	
Architecture	**Val. Acc.**	**Test Acc**	**Val. Acc.**	**Test Acc**
ResNet-50	1.6%	7.9%	1.2%	4.0%
ResNet-101	1.9%	6.2%	3.3%	6.2%
ResNet-152	1.2%	6.2%	2.2%	3.4%

Table 4: Standard deviations for every case with ResNet

As shown, the deviation of the validation accuracy is relatively low for every case but for ResNet-101 with processed resolution. This means that the behaviour has been relatively stable, and more iterations would not affect validation accuracy. On the other hand, the large deviations on test accuracy are

explained by the small test dataset. Small changes generate large effects on deviation.

5.2 EfficientNet

This subsection focuses on the results obtained following the procedure defined in section 4.2 and running the case scenarios specified in section 4.1.2 for EfficientNet.

Table 5 gathers the average and best validation and testing accuracy results for EfficientNet architectures B0 to B7 for original resolution and fixed resolution. It also includes the number of parameters and FLOPS that corresponds to each model. Note that these two numbers refer to the total amount, including the first training and the fine tuning.

			Original resolution			Processed resolution			
Arch.	**Score**	**Parameters**	**Val. Acc.**	**Test Acc.**	**FLOPS**	**Val. Acc.**	**Test Acc.**	**FLOPS**	**FLOPS Diff.**
B0	Avg	8.43E+06	82.2%	78.7%	1.59E+09	79.7%	74.1%	3.01E+09	-47%
	Top 1		84.4%	85.7%		81.3%	76.2%		
B1	Avg	1.35E+07	82.8%	76.8%	2.82E+09	81.2%	77.1%	4.46E+09	-37%
	Top 1		84.4%	**85.7%**		84.4%	81.0%		
B2	Avg	1.59E+07	**84.2%**	84.7%	4.07E+09	81.9%	81.7%	5.15E+09	-21%
	Top 1		87.5%	**85.7%**		82.8%	**85.7%**		
B3	Avg	2.20E+07	78.3%	76.8%	7.48E+09	78.3%	76.8%	7.48E+09	0%
	Top 1		82.8%	**85.7%**		82.8%	**85.7%**		
B4	Avg	3.58E+07	81.8%	79.7%	1.79E+10	79.0%	67.4%	1.17E+10	54%
	Top 1		85.9%	**85.7%**		81.3%	71.4%		
B5	Avg	5.76E+07	82.5%	74.4%	4.18E+10	79.3%	75.1%	1.83E+10	129%
	Top 1		84.4%	85.7%		82.8%	81.0%		
B6	Avg	8.25E+07	83.7%	75.3%	7.75E+10	84.3%	74.7%	2.61E+10	198%
	Top 1		85.9%	85.7%		87.5%	81.0%		
B7	Avg	1.29E+08	84.6%	75.1%	1.53E+11	**85.9%**	69.2%	4.01E+10	283%
	Top 1		**87.5%**	81.0%		**89.1%**	76.2%		

Table 5: Results with EfficientNet simulation models

Where original resolution corresponds to the image size defined in Table 2 and processed resolution to the 300x300 one defined in this work for analysis.

5.2.1 Number of parameters and FLOPS

Comparisons of the obtained number of parameters and FLOPs with the general results shown in Figure 12 is not an option. This is not only because the developed model included a 128 unit dense layer as ref [31] did not, but also because the model applies a fine tuning which was not included in [31].

Attending to Table 5, attention is drawn to the negative FLOPS diff. column with architectures B0 to B2. To understand the results, Figure 26 shows the results. In this figure 26, a comparison of the number of parameters and FLOPS between the original resolution and the processed resolution is shown, where the points in the graph specify the results of B0 to B7 from left to right.

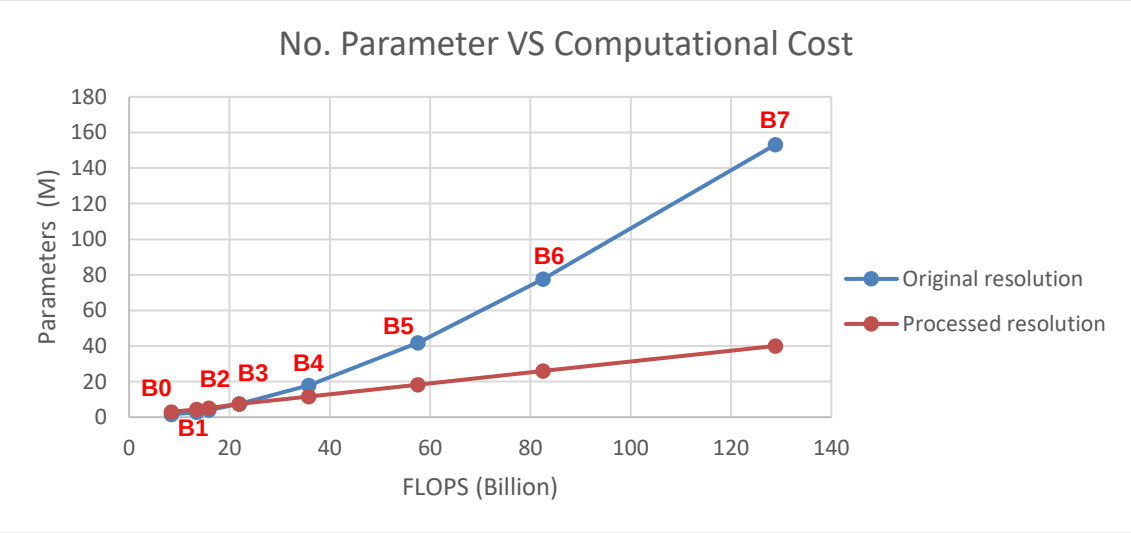


Figure 26: Comparison of number of parameters and FLOPS between the original and processed resolution with EfficientNet

As expected, a strong correlation exists between number of parameters and FLOPS. The larger the number of parameters, the higher the FLOPS. In other words, the more complex the architecture, the higher the required computation time. This is clearly seeing when looking at both curves.

On the other hand, it can be seen that the original resolution curve is above the processed one for architectures B4 on. For those architectures, the original resolution image size is larger than with the *processed* one, meaning that it works with a larger number of parameters. Therefore, it can slightly be seen that the

number of parameters for architectures B0 to B2 with *processed* resolution is larger because for B2, the original resolution is 260, smaller than the processed 300. This is why the FLOPS diff. column is negative for architectures B0 to B2.

5.2.2 Validation accuracy

In Figure 27, a comparison of the average validation accuracy is performed between original resolution and processed resolution.

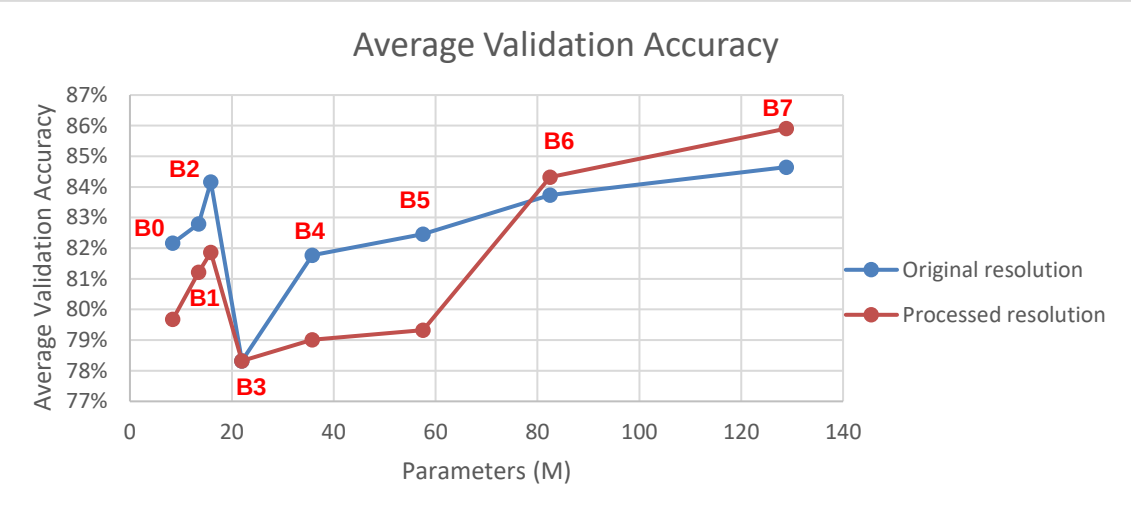


Figure 27: Comparison of average validation accuracy between original and processed resolution with EfficientNet

The first thing that can be seen in Figure 27 is that the average validation accuracy drops when going down from B7 to B3. This is what was expected and similar to results obtained with the ImageNet Dataset in Ref. [31], as shown in Figure 12. However, when moving to architectures below B3, it is found that the average validation accuracy no longer drops but increases. This was also found in Ref. [31] and Figure 13, when the architectures are applied to different datasets and found that lower level architectures provided sometimes higher accuracies than the higher level ones. Nevertheless, in this case, the efficiency obtained with B0 to B2 can be compared to B6 and B7 in value.

Another interesting effect that can be seen is that the original resolution (the resolution for which the architecture was designed) is always above the processed one except for architectures B6 and B7. No clear explanation has been

found to understand why the original resolution is below the processed one for those two cases.

On the other hand, it is important to clarify that validation accuracies match between original and processed resolution on architecture B3 as the original resolution is the same as the processed one.

Attending to the obtained values, the maximum average validation accuracy is obtained with the B7 architecture and the processed resolution and has a value of 85.9%. This is above the best accuracy (85.5%) obtained with the Inception V3 architecture [47].

Regarding accuracy, Figure 28 show the best validation accuracy for the original and processed resolution cases. In this case as well, a peak value is obtained for the processed resolution with a B7 architecture and has a value of 89.1%.

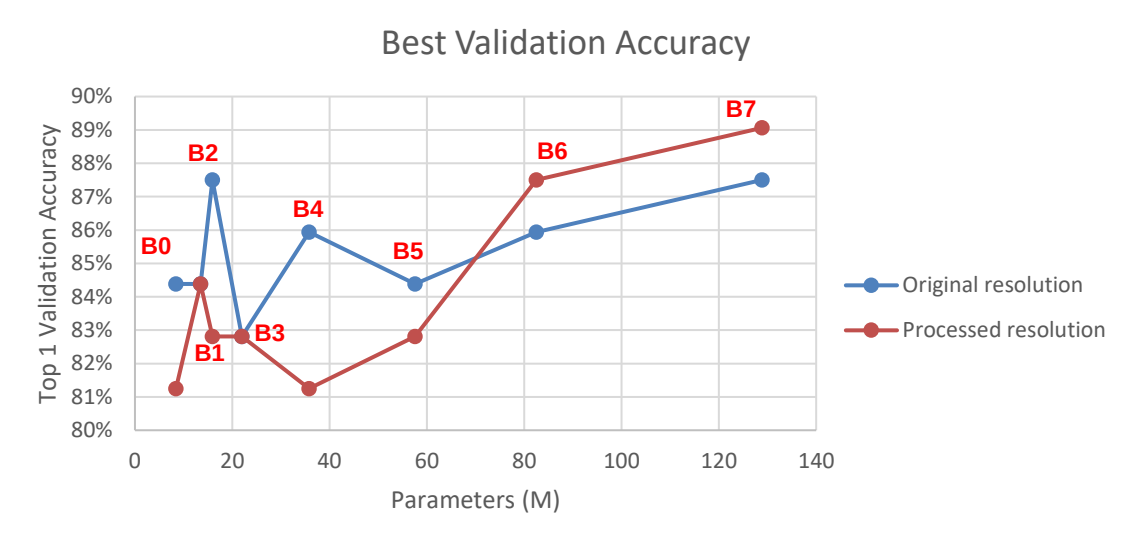


Figure 28: Comparison of best validation accuracy between original and processed resolution with EfficientNet

Like the average validation accuracy, original resolution best accuracy is always above the processed one except for B6 and B7 architectures. The best accuracy is achieved by the B2 architecture for the original resolution. The value exceeds the one obtained by the B7 with the original resolution, and matches the B6 for the processed one with a value of 87.5%

5.2.3 Test accuracy

The results of this analysis are shown in Figure 29 and Figure 30, for average test accuracy and best accuracy, respectively.

Although similar performance regarding Figure 27 and Figure 28 could be expected, they do not look alike. Test accuracy does not improve when moving from lower to higher level architectures; in fact, the highest average test accuracy is obtained with the B2 architecture and has a value of 84.7%.

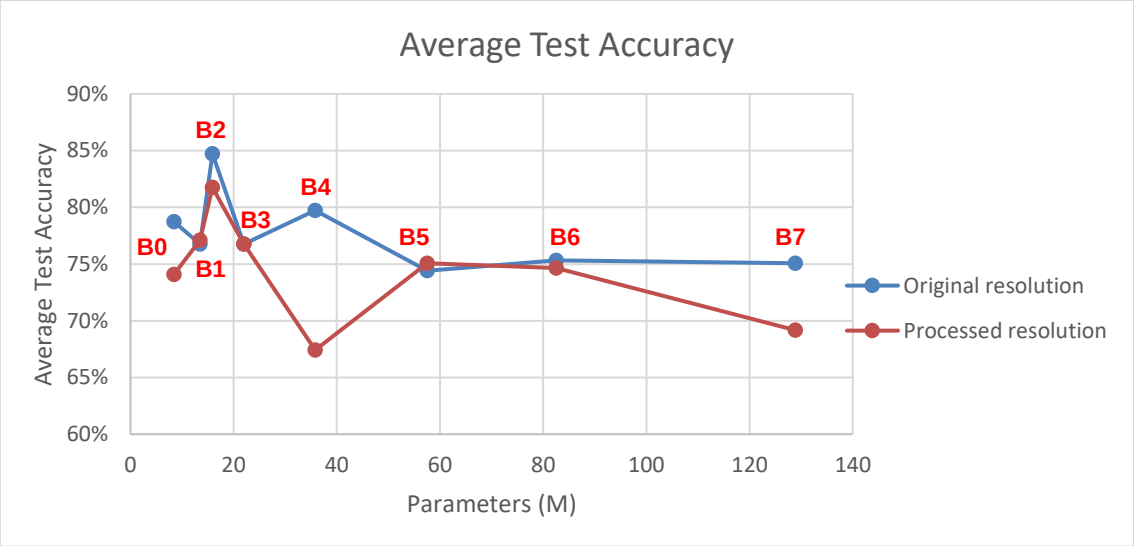


Figure 29: Comparison of average test accuracy between original and processed resolution with EfficientNet

The best test accuracy are obtained with B0 to B6 architectures, with a value of 85.7%. Results cannot be compared to Ref. [47] as test accuracy was not measured.

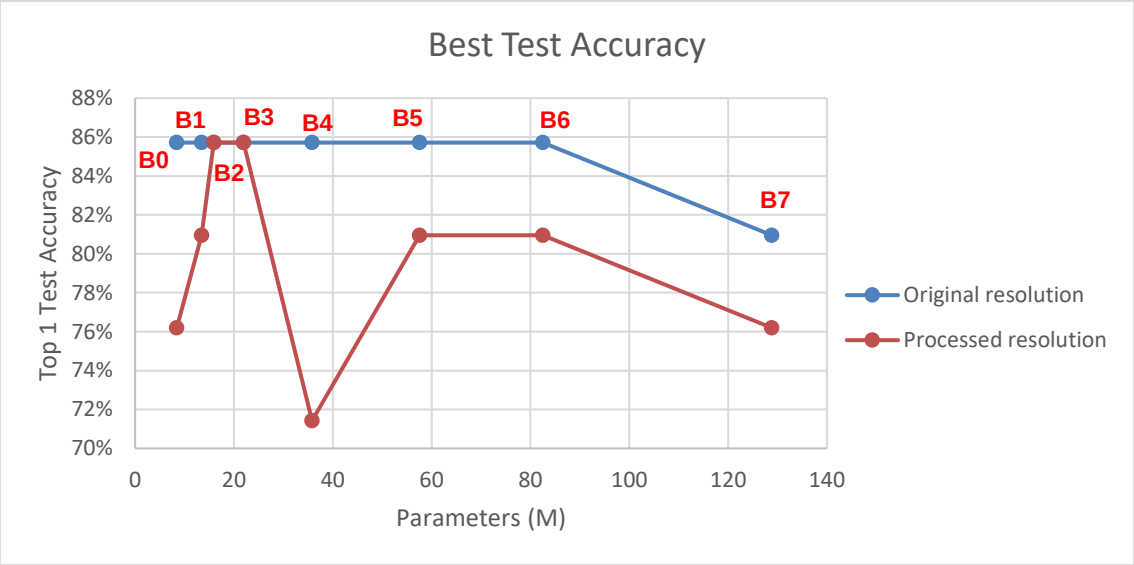


Figure 30: Comparison of average validation accuracy between original and processed resolution with EfficientNet

The difference between validation and test accuracies are explained by the small dataset. The main risk of having a small dataset is that in a randomly distributed case, datasets might not generalize well enough. Moreover, small differences on classification may make a large difference on results.

5.2.4 Others

In the scenarios run with EfficientNet, a considerable difference on the results was found between iterations.

To analyse the effect of deviation, the standard deviation has been measured in each case. The larger the deviation, the less reliable the results is as the accuracies are more likely to fluctuate. A summary of this deviations is shown in Table 6:

	Original		Processed	
Architecture	**Val. Acc.**	**Test Acc**	**Val. Acc.**	**Test Acc**
B0	2.0%	5.4%	1.1%	4.3%
B1	1.4%	4.0%	1.9%	2.1%
B2	2.0%	2.1%	0.9%	4.0%
B3	3.2%	6.2%	**3.2%**	6.2%
B4	3.4%	5.2%	2.1%	4.0%
B5	**4.6%**	8.5%	2.3%	4.0%
B6	1.1%	8.9%	2.5%	6.0%
B7	1.4%	4.0%	1.9%	5.4%

Table 6: Standard deviations for every case with EfficientNet

It can be seen in this Table that validation accuracy deviation goes up to 4.5% for the original resolution. Similarly, test accuracy deviation values go up to 8.9%, which is considerable. Looking at these deviations, validation accuracy especially, the reasonable next step would be running more models so that the deviation decreases to a more reasonable value.

On the other hand, it has been found that model fine tuning did not improve the results in every case. In fact, only about 10% of the models improved during the fine-tuning process. This is something to consider as the computational cost of this step is double of the basic one. Depending on the required accuracy, avoiding this step would be very cost effective. If a parametric analysis is to be performed on the model, the fine-tuning process should be avoided.

5.3 CNN results conclusions

Among the available architectures, ResNet and EfficientNet have been selected in this case study: The first one for been previously used for railway tracks applications, and the second on for having shown in Ref. [31] to have higher performance values and lower computation cost.

On the one hand, ResNet-50 to ResNet-152 architectures were run. The maximum validation accuracy obtained with these models was 77.1% and test accuracy of 76.2% with ResNet-101 processed resolution. As defined in section 4.3.1, none of the models exceeded the established 80% threshold for validation accuracy.

In terms of resolution with ResNet architecture, no clear outcome has been extracted. For some architectures, original resolutions performed better than the processed one and vice-versa. This effect is also translated to test accuracy.

On the other hand, EfficientNet results have shown a best average validation accuracy of 85.9% and best validation accuracy of 89.1% for EfficientNet-B7 architecture for processed resolution. It is worth highlighting that EfficienNet-B2 architecture with the original model resolution obtained an 84.2% validation accuracy and 87.5% test accuracy with more than six times fewer FLOPS.

Regarding resolution comparison with EfficientNet, original model resolution has proved to get better results in every aspect except for B6 and B7 validation accuracy. This greater accuracy comes at the cost of a higher computation requirement as the resolution is higher for models above B3, whose resolution matches the one selected as *processed*.

Between the analysed architecture and taking into account the considered scenarios, EfficientNet performs better in this case study. This is clearly noticeable in validation and test accuracy, where ResNet did not reached the established 80% threshold.

If a model had to be selected for further analysis, EfficientNet-B2 architecture would be chosen as it provides validation accuracies comparable to the B7 architecture, but with a much smaller computation cost as shown in Figure 26. Therefore, attending to the small validation accuracy and low computation cost compared to B7, EfficientNet-B2 with original resolution has been selected for further analysis.

5.4 Fault detection

It is also important to analyse in which area of the image the classifier is detecting the fault. In this section, the algorithm explained in section 2.3 has been applied.

In total, 70 models have been trained. As analysing all the results is not feasible, only the model that gave best average results (EfficientNet-B2 with *original* image resolution input) has been chosen. Among the run five iterations, the one that exactly matches de average will be used as it contains cases of defective detected images in both, real defective ones and in non-defective ones. The reason to choose this model is that in the ratio of computation cost vs results, it provided the best results. This case corresponds to iteration 3 (see Appendix A.2 for more details on the results of this case), of which confusion matrix is shown in Table 7. However, the results obtained with this model can be considered representative of all the cases.

With this model, the evaluation criteria variables have been measured. To understand how good the model is, this analysis has been done with multiple thresholds, going from 50% to 95% of probability. That is, a threshold of 60% means that the probability of being defective or non-defective will have to be larger than it. All the results are shown in Table 7.

	Thresholds					
	50%	**60%**	**70%**	**80%**	**90%**	**95%**
TP	6	6	6	6	4	3
FN	2	2	2	2	4	5
TN	11	11	11	11	9	6
FP	2	2	2	2	4	7
Accuracy	81.0%	81.0%	81.0%	81.0%	61.9%	42.9%
Precision	75.0%	75.0%	75.0%	75.0%	50.0%	30.0%
Sensitivity	75.0%	75.0%	75.0%	75.0%	50.0%	37.5%
F1	75.0%	75.0%	75.0%	75.0%	50.0%	33.3%

Table 7: Confusion matrix for EfficientNet-B2 case with original resolution

As expected, the higher the threshold, the lower the accuracy, precision, sensitivity and F1 score. This is logical as the more demanding the threshold is, the fewer the cases that will fall within those values.

Looking at the results, a threshold of 80% seems the most reasonable one, as it is the highest threshold that keeps initial accuracy and F1 results. For this reason, an 80% threshold will be applied to the images to detect the faulty area following the methodology explained in section 2.3.

5.4.1.1 Defective images detected as defective

Image 1

The first image obtained a probability of being defective of 87.96%. Figure 31 shows the first analysed original image (a) when the object detection is performed (b), after applying NMS (c) and the ROI with highest error probability (d).

The trained model has detected probabilities over 73% in three different areas of the railway track, apparently based on changes in colour and brightness of the railway track. In Figure 31, image (b) It can be seen that there are many areas with a fault probability above the threshold; however, after applying NMS (b), it is reduced to a total of 5 ROI, all of them with fault probability above 70%. Figure 31, (d) shows that the area with highest fault probability is the railway track fasteners with a value of 81.8%.

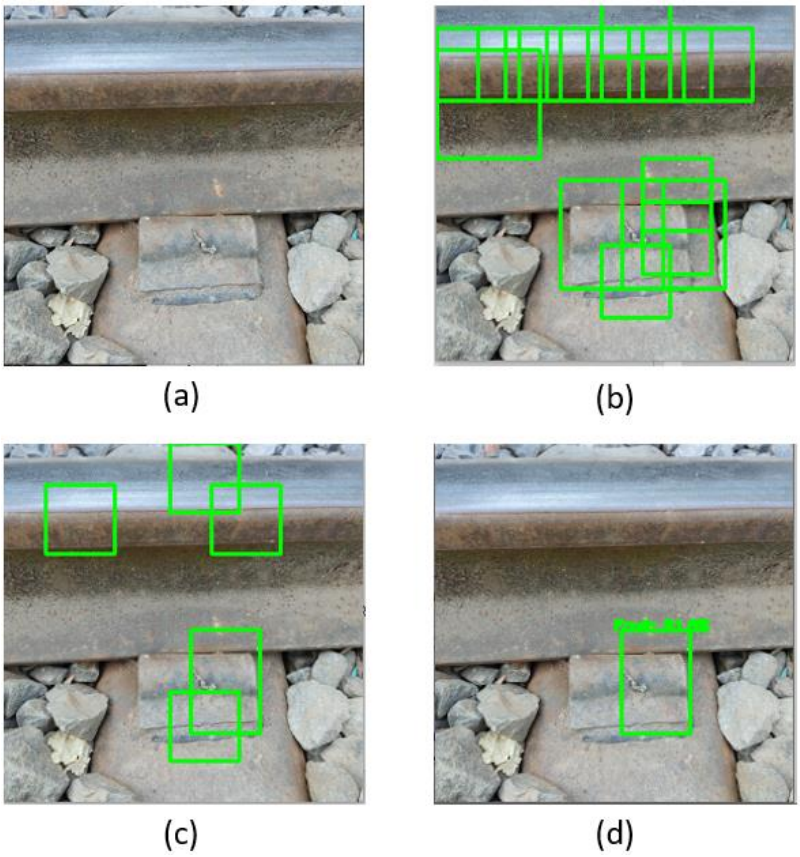


Figure 31: (a) Original defective image (1/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area

Image 2

For the second image (Figure 32 (a)), the trained model gave a probability of being defective of 96.6%. In Figure 32 (b), it is shown that there are many areas with a fault probability of above 60%. This also happens after applying NMS (c). This might mean that there are multiple faults in the image.

In this case, the area with highest probability (78.2%) of being defective is located at the track base, where it seems worn. This are has been highlighted in image (d) in Figure 32. However, it is important to mentioned that the second highest probability area, with a value of 77.4%, is the large box just next to the one highlighted in Figure 32 (d). This box is also framing a worn area close to the fastener.

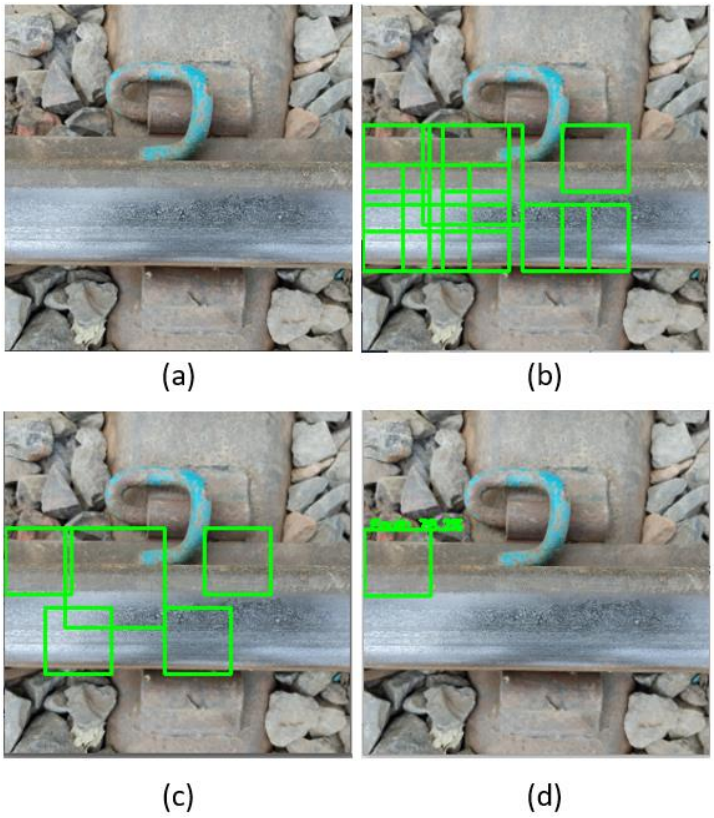


Figure 32: (a) Original defective image (2/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area

Image 3

The model detects a fault with 95.5% of probability on (a) image of Figure 33. In this case, the defects on the railway track are quite visible as it presents some dents on the corner of the left-hand side track and a deep scratch on the right-hand side track. All these areas are well captured in Figure 33 (b).

The ROI with highest probability is highlighted in Figure 33 (d) and has a value of 66.96%, closely followed by the other area shown in image (c) with a probability of 66.91%. In this case, the algorithm detects with a higher value the worn area of the tracks than the actual crack or dents that have been captured in Figure 33 (b).

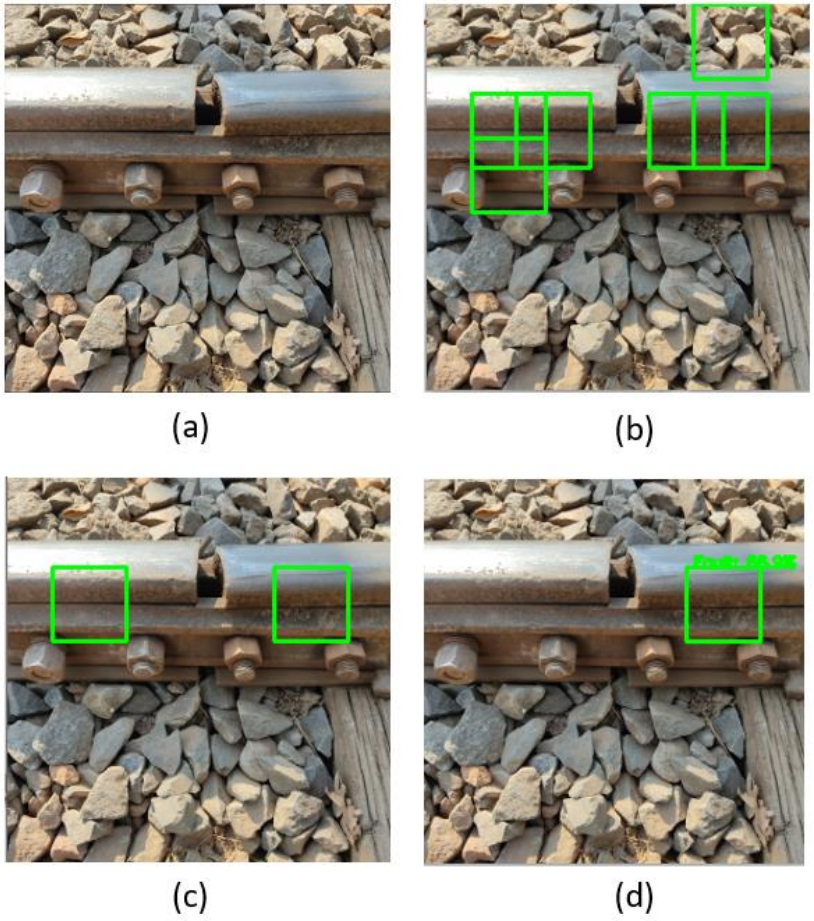


Figure 33: (a) Original defective image (3/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area

Image 4

The image shown in Figure 34 (a) returned a probability of being defective of 89.5%. In the original image, the only fault that is noticeable is the lack of fastener on the railway track.

Image (b) and (c) in Figure 34 show that the model has captured two areas; the missing bolt one and a rail area with slightely different colour that might be worn.

Satisfactorily, as shown in Figure 34 (d), the ROI pointing at the lack of fastener has been detected as a faulty area with a probability of 64%. On the other hand, the second area does not look like it is capturing any clear defect. It

seems it might be pointing towards a darkish area on the railway tracks that do not seem to be faulty.

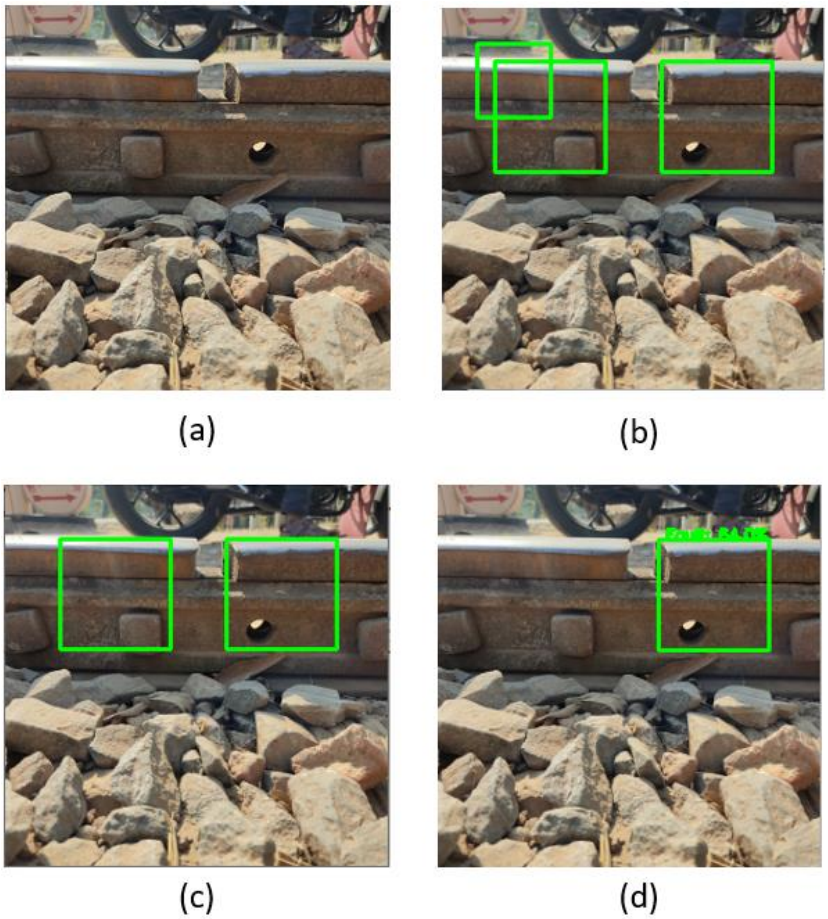


Figure 34: (a) Original defective image (4/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area

Image 5

Figure 35 (a) shows a defective image which has been classified as defective with a 97.9% probability.

In this case, the defect on the railway track is quite visible as it presents multiple dents, some of them really deep, along the left-hand railway track. All these have been satisfactorily detected and captured in image (b) in Figure 35. These dents are found not only on the top of the track but also on the side that the picture is showing. Areas showing these dents have clearly been detected by the model with a fault probability of 63.8% as highlighted in Figure 35 (c).

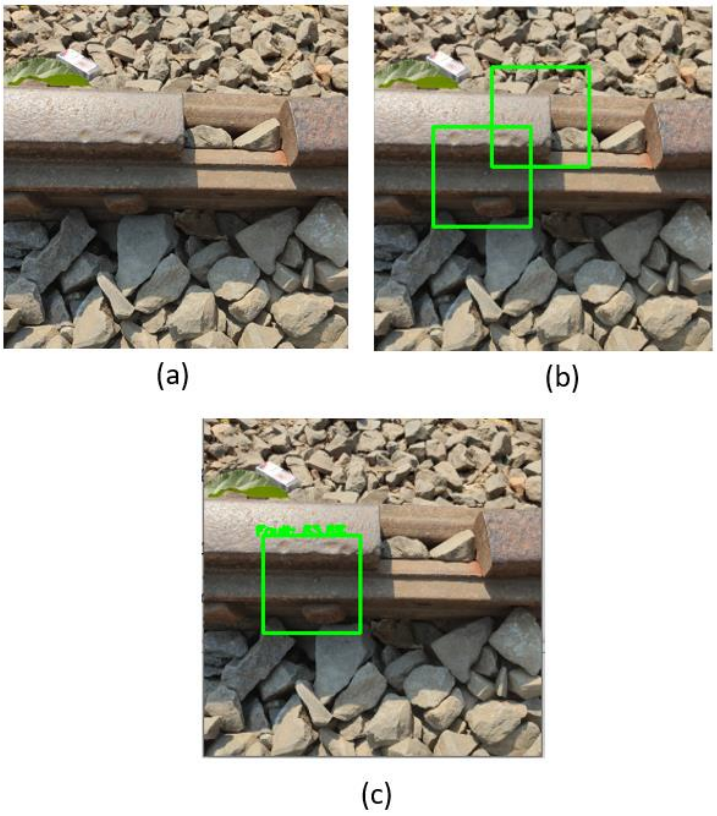


Figure 35: (a) Original defective image (5/6); (b) ROIs above 60% fault probability; (c) highest fault probability area

Image 6

The last test image that the model has detected as faulty is shown in Figure 36 (a). This image is especially tricky as it presents two railway tracks, one at the front and the other at the background. For this image, the model predicted a probability of being defective of 94.7%.

Once again, the model has detected several areas above the threshold as shown in image (b). Attending at image (c), it seems it is also detecting the defects on the worn area of the railway tracks. Even if it is not clear, it looks like it might have also detected some dents on both sides of the railway track.

On the other hand, Figure 36 (d) shows the ROI with highest probability pointing towards the missing bolt on the forward track and has returned a probability of error of 92.8%. However, the model is not detecting the remaining

missing bolt, probably because the ROI above or to its left returned a higher probability of fault.

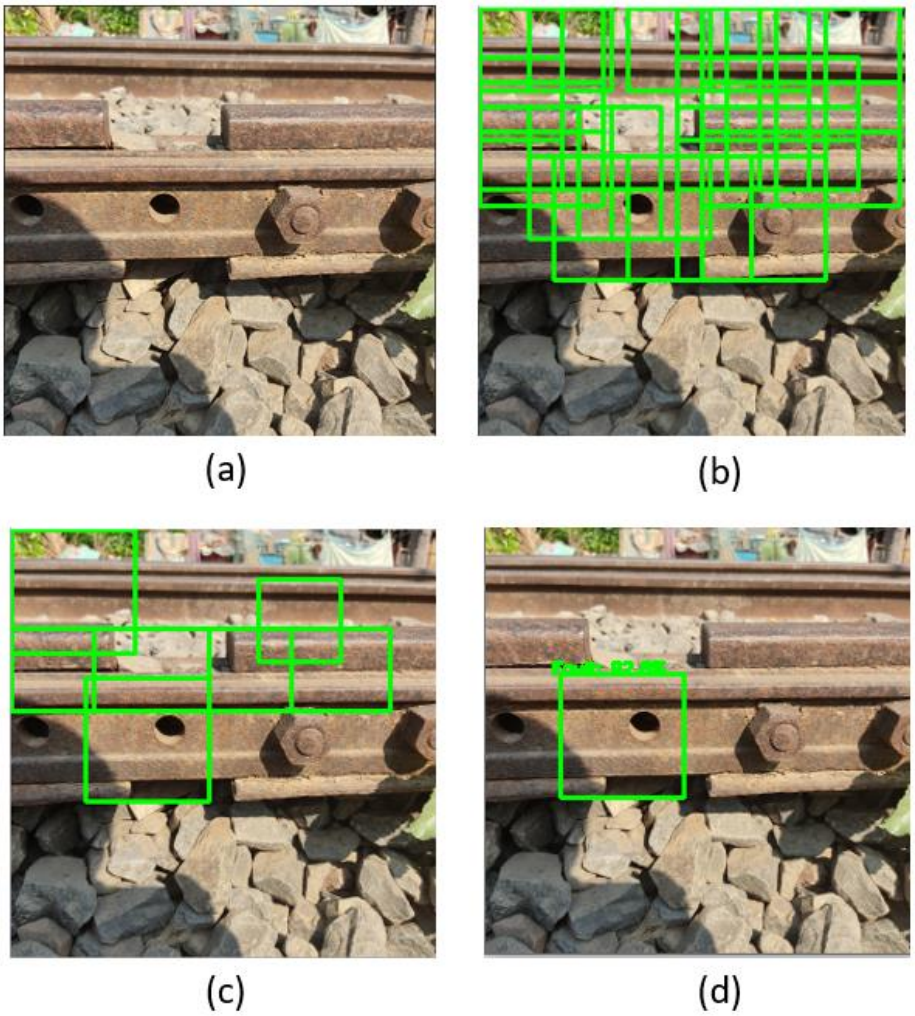


Figure 36: (a) Original defective image (6/6); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area

5.4.1.2 Defective images detected as non-defective

As shown in Table 7, two images of the defective dataset have been detected as non-defective. These two images are shown in Figure 37. From left to right, the probability of defect detected by the model was 15.6% and 37.4%, respectively.

Figure 37: Defective images that have been detected as non-defective

It is not clear which defects these two images have, but it looks like there is some erosion on the left railway track and some worn area and a missing bolt on the right-hand side image.

5.4.1.3 Non-Defective images detected as defective

As previously explained, the model has detected two images as defective that were not labelled as such.

Image 1

Figure 38 (a) shows the first image that is non-defective and has been detected as faulty with a 94% probability.

In this case, the two highlighted areas in image (b) have a fault probability below 65%. The small rectangle highlighted in Figure 38 (c) is pointing at the rocks next to the tracks, which is mixed up with railway track faults, while the large square is not pointing to any location.

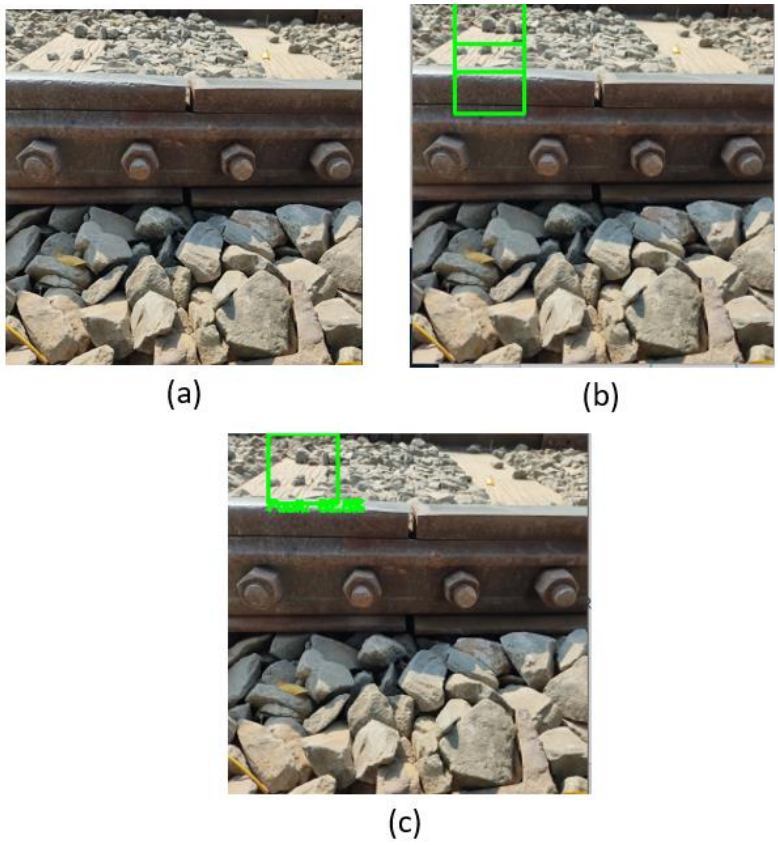


Figure 38: (a) Non-defective images that have been detected as defective (1/2); (b) ROIs above 60% fault probability; (c) highest fault probability area

Image 2

The second non-defective image that has been detected as faulty is shown in Figure 39 (a) and showed a probability of being faulty of 64.3%. This is again a tricky picture as there are two tracks close to each other.

Figure 39 (b) shows that multiple areas have exceeded the stablished error threshold while image (c) illustrates the results once NMS has been applied and *real* faulty areas have been reduced.

Attending to Figure 39 (d), the model has detected the area with the highest probability of been faulty on the right-hand side track with a confidence of 95% that matches the area where there is a colour change. It looks like this might be an excessively worn area (or even a crack), but the image has been classified as non-defective so there might not be any fault.

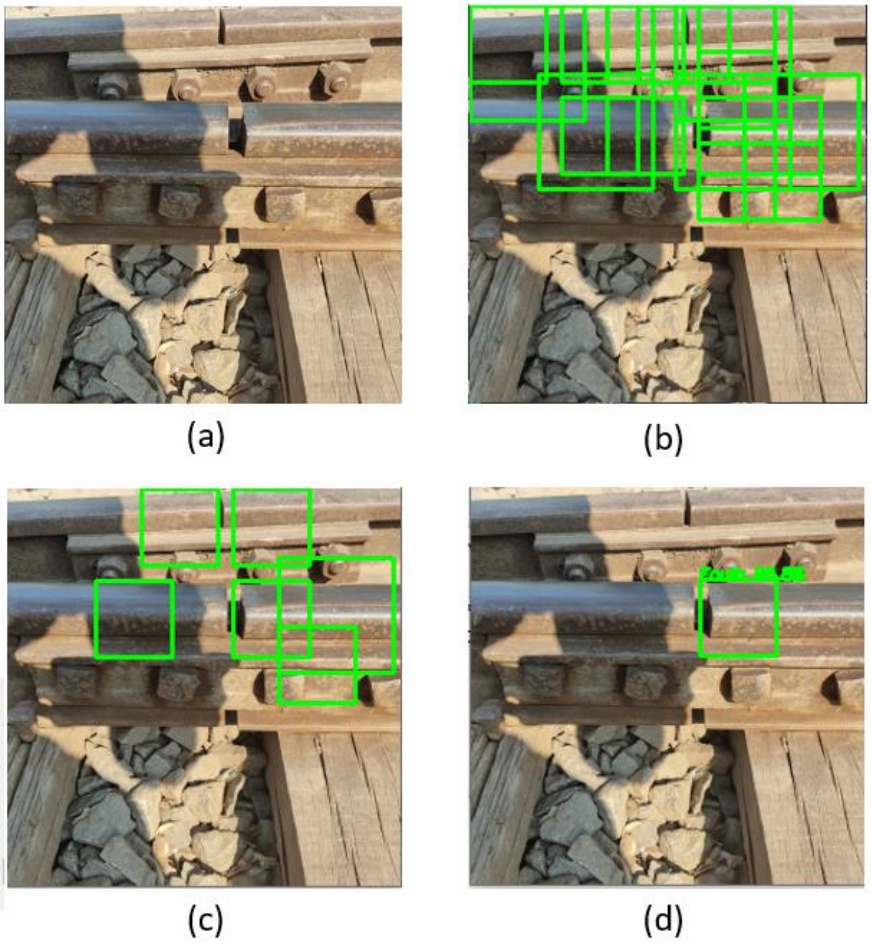


Figure 39: (a) Non-defective images that have been detected as defective (1/2); (b) ROIs above 60% fault probability; (c) ROIs after NMS; (d) highest fault probability area

6 Conclusions and future works

This project has analysed and compared multiple deep learning techniques to detect railway track faults on a small dataset. The primary focus of the work has been applying a novel DCNN architecture that outperformed ResNet, the most used architecture so far by the scientific community on railway track image classification problems.

Among the available options, EfficientNet has been selected as the novel ConvNet due to the high performance values and low computation cost shown in Ref. [31]. As shown in Table 3 and Table 5, every case of the EfficientNet architecture family (B0 to B7) has proved to exceed accuracies obtained with ResNet. Moreover, ResNet results did not meet the evaluation criteria set in section 4.3.

Results have shown a best average validation accuracy of 85.9% for the EfficientNet-B7 architecture on the processed resolution case. A best validation accuracy of 89.1% has been achieved for that same model. These results are consistent with Ref. [31].

Regarding resolution comparison, original model resolution has in overall proved to get better results in every aspect except for B6 an B7 on validation accuracy. This greater accuracy comes at the cost of a higher computation requirement as the resolution is higher for models above B3, whose resolution matches the one selected as *processed*.

On the other hand, faulty area detection has shown to be accurate. In most of the cases, the model positively predicted the area of defects for the analysed case of EfficientNet-B2 with original size images. However, an expert on railway track fault detection should analyse the data, as some areas that have been detected as faulty might not be.

Regarding future works, the following paths are suggested to be followed:

- It is believed that a parametric analysis on some of the EfficientNet architectures could improve the obtained results. As explained in Sections 3 and 4.2, the number of variables that take part when running and analysis are considerably large, which makes believe that modifying some might improve results. For example, only the first block of features have been defined as trainable, however, extending this to the first two or three blocks might improve results. For such analysis, EfficientNet-B2 architecture would be selected as it provides accuracies comparable to the B7 architecture, but with a much smaller computation cost as shown in Figure 26.

- Table 3 and Table 5 show some high standard deviations on a few cases. This means that if models were run again, some results are more likely to defer from the ones that have been obtained. Therefore, to reduce these deviations and to get a higher confidence on the results, each of the models would need to be run more than five times.

- Having such a small dataset directly affects the model training process, which in consequence, affects the obtained results. Ideally, the dataset would need to be enlarged with photos containing similar track fault defects.

- Some research have proven that combining Generative Adversarial Network (GAN) together with a CNN architecture can considerably improve accuracies [14]. This technique can be especially useful on small dataset problems at the cost of increasing the computation cost of training the model.

REFERENCES

[1] International Energy Agency, “The Future of Rail,” 2019.

[2] International Energy Agency, “Global line kilometers of metro, light rail and high-speed rail, 2000-2018,” 2020. [Online]. Available: https://www.iea.org/data-and-statistics/charts/global-line-kilometers-of-metro-light-rail-and-high-speed-rail-2000-2018. [Accessed: 06-Jul-2022].

[3] R. Shafique *et al.*, “A Novel Approach to Railway Track Faults Detection Using Acoustic Analysis,” pp. 1–19, 2021.

[4] D. Kang and S. Park, “Study on MFL Technology for Defect Detection of Railroad Track Under Speed- up Condition Study on MFL Technology for Defect Detection of Railroad Track,” no. June 2018, 2015.

[5] K. Bruzelius and D. Mba, “An initial investigation on the potential applicability of Acoustic Emission to rail track fault detection,” *ELSEVIER*, vol. 37, pp. 507–516, 2004.

[6] X. Zhang, N. Feng, Y. Wang, and Y. Shen, “Acoustic emission detection of rail defect based on wavelet transform and Shannon entropy,” *J. Sound Vib.*, vol. 339, pp. 419–432, 2015.

[7] Q. Li and S. Ren, “A Real-Time Visual Inspection System for Discrete Surface Defects of Rail Heads,” *IEEE*, vol. 61, no. 8, pp. 2189–2199, 2012.

[8] Q. Li and S. Ren, “A Visual Detection System for Rail Surface Defects,” *IEEE*, vol. 42, no. 6, pp. 1531–1542, 2012.

[9] X. Wei, Z. Yang, Y. Liu, D. Wei, L. Jia, and Y. Li, “Railway track fastener defect detection based on image processing and deep learning techniques : A comparative study,” *Eng. Appl. Artif. Intell.*, vol. 80, no. September 2018, pp. 66–81, 2019.

[10] U. Dube, S. Subramaniam, and S. Subramaniam, “Cost Effective Railway Track Fault Detection Using Image Processing,” no. Icicnis, 2021.

[11] S. Mittal, “Vision Based Railway Track Monitoring Using Deep Learning.”

[12] Y. Rezaeitabar, S. Dogru, and İ. Ulusoy, “Railway Fastener Inspection by Real-Time Machine Vision,” no. March, 2015.

[13] Y. Lin, C. Hsieh, W. Huang, S. Hsieh, and W. Hung, “Railway Track Fasteners Fault Detection using Deep Learning,” *2019 IEEE Eurasia Conf. IOT, Commun. Eng.*, pp. 187–190, 2019.

[14] D. Yao, Q. Sun, J. Yang, H. Liu, and J. Zhang, “Railway Fastener Fault Diagnosis Based on Generative Adversarial Network and Residual

Network Model," vol. 2020, 2020.

[15] N. Alnaimi, "IoT Based on-the-fly Visual Defect Detection in Railway Tracks," pp. 627–631, 2020.

[16] S. Bhat and A. G. Karegowda, "Railway Track Monitoring Using Deep Learning Techniques : A Preview," *Int. Jorunal Appl. Res. Inf. Technol. Comput.*, vol. 11, no. 3, pp. 93–114, 2020.

[17] A. James *et al.*, "TrackNet - A Deep Learning Based Fault Detection for Railway Track Inspection," *IEEE*, no. 1, pp. 1–5, 2018.

[18] O. Yaman, M. Karaköse, E. Ak, and İ. Ayd, "Ray Yüzeyi için Görüntü İş leme Tabanl ı Ar ı za Tespit Yakla ş ı m ı Image Processing Based Fault Detection Approach for Rail Surface," *IEEE*, no. 1, p. 4, 2015.

[19] X. Wei, D. Wei, D. A. Suo, L. Jia, and Y. Li, "Multi-target Defect Identification for Railway Track Line Based on Image Processing and Improved YOLOv3 Model," *IEEE Access*, vol. PP, p. 1, 2020.

[20] J. Gu *et al.*, "Recent Advances in Convolutional Neural Networks," *ScienceDirect*, 2017.

[21] W. Rawat and Z. Wang, "Deep Convolutional NeuralNetworks for Image Classification: A Comprehensive Review," *Neural Comput.*, 2017.

[22] K. O. Shea and R. Nash, "An Introduction to Convolutional Neural Networks," pp. 1–11, 2015.

[23] X. Lei and Z. Sui, "Intelligent fault detection of high voltage line based on the Faster R-CNN," *ELSEVIER, Meas.*, vol. 138, pp. 379–385, 2019.

[24] A. Reddy, V. I. Gandhi, L. Ravi, and V. Subramaniyaswamy, "Detection of Cracks and damage in Wind Turbine blades using Artificial Intelligence-based Image Analytics," *Measurement*, 2019.

[25] Z. Müftüoglu, M. Ayyüce Kizrak, and T. Yildirim, "Differential Privacy Practice on Diagnosis of COVID-19 Radiology Imaging Using EfficientNet," *Cornell Univ. Libr.*, pp. 1–6, 2020.

[26] P. Abdou and K. Karou, "Accurate Detection of COVID-19 Using K-EfficientNet Deep Learning Image Classifier and K-COVID Chest X-Ray Images Dataset." pp. 1527–1531, 2021.

[27] F. L. Gambo, G. M. Wajiga, and E. Garba, "Prediction of Learning Style Using Facial Expressions with Convolutional Neural Prediction of Learning Style Using Facial Expressions with Convolutional Neural Network," *ResearchGate*, no. June, 2020.

[28] A. Vojteková, "Neural network noise reduction of astronomical images."

Brno, 2020.

[29] F. O. R. L. Arge and C. I. Mage, “Very Deep Convolutional Networks for Large-Scale Image Recognition,” 2014, pp. 1–14.

[30] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” Microsoft, 2015.

[31] M. Tan and Q. V Le, “EfficientNet : Rethinking Model Scaling for Convolutional Neural Networks,” in *International Conference on Machine Learning*, 2019.

[32] M. Tan, “Google AI Blog,” *Google AI*, 2019. [Online]. Available: https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html. [Accessed: 29-May-2022].

[33] M. Sandler, M. Zhu, A. Zhmoginov, and C. V Mar, “MobileNetV2: Inverted Residuals and Linear Bottlenecks,” 2019.

[34] B. Tumor and C. Using, “Brain Tumor Classification Using Dense Efficient-Net,” *Axioms*, no. 1, p. 13, 2022.

[35] A. S. Ebenezer, S. D. Kanmani, M. Sivakumar, and S. J. Priya, “Materials Today : Proceedings Effect of image transformation on EfficientNet model for COVID-19 CT image classification,” *Mater. Today Proc.*, vol. 51, pp. 2512–2519, 2022.

[36] G. Marques, D. Agarwal, I. De, and T. Díez, “Automated medical diagnosis of COVID-19 through EfficientNet convolutional neural network,” *Appl. Soft Comput. J.*, vol. 96, p. 106691, 2020.

[37] Pyimagesearch, “Turning any CNN image classifier into an object detector with Keras, TensorFlow, and OpenCV.” [Online]. Available: https://pyimagesearch.com/2020/06/22/turning-any-cnn-image-classifier-into-an-object-detector-with-keras-tensorflow-and-opencv/. [Accessed: 01-Jun-2022].

[38] R. Rothe, M. Guillaumin, and L. Van Gool, “Non-Maximum Suppression for Object Detection by Passing Messages between Windows Non-Maximum Suppression for Object Detection by Passing Messages between Windows,” no. November, 2015.

[39] TowardsDataScience, “Non-maximum Suppression (NMS).” [Online]. Available: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c. [Accessed: 28-Jun-2022].

[40] Kaggle, “Kaggle.” [Online]. Available: https://www.kaggle.com/. [Accessed: 01-May-2022].

[41] Wikipedia, “Kaggel.” [Online]. Available:

https://es.wikipedia.org/wiki/Kaggle. [Accessed: 01-May-2022].

[42] Kaggle, “Railway Track Fault Detection - Dataset_Version 2,” *Kaggle*, 2021. [Online]. Available: https://www.kaggle.com/datasets/salmaneunus/railway-track-fault-detection?resource=download&select=Railway+Track+fault+Detection+Updated. [Accessed: 01-May-2022].

[43] Indian Railways Institute of Civil Engineering, *Handbook for Track Maintenance*, no. 1. 2016.

[44] C. Shorten and T. M. Khoshgoftaar, “A survey on Image Data Augmentation for Deep Learning,” *J. Big Data*, 2019.

[45] A. Mikołajczyk and M. Grochowski, “Data augmentation for improving deep learning in image classification problem,” no. May, 2018.

[46] Keras, “Keras,” 2015. [Online]. Available: https://keras.io/. [Accessed: 05-May-2022].

[47] Kaggle, “Railway Track Fault Detection - Dataset_Version 1,” *Kaggle*, 2020. [Online]. Available: https://www.kaggle.com/code/salmaneunus/railway-track-fault-detection-1/data?select=dataset. [Accessed: 01-May-2022].

APPENDICES

Appendix A Convolutional Neural Network

Appendix A includes the whole set of results obtained in this work

A.1 Full set of ResNetresults

This appendix gathers the full set of results obtained for EfficientNet Convolutional Neural Network architecture.

ResNet-50 - Original resolution

Param	4.76E+07								
Flops	1.55E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	accuracy
1	62.2%	71%	56.0%	72.9%	3	5	0	13	76%
2	57.5%	70.8%	57.9%	72.9%	0	8	1	12	57%
3	125.3%	57.4%	60.5%	71.4%	4	4	1	12	76%
4	60.4%	70.3%	55.1%	75.7%	3	5	1	12	71%
5	71.2%	67.7%	64.1%	72.9%	4	4	2	12	73%
			Avrg	73.1%				Avrg	69.9%
			Max	75.7%				Max	76.2%
			Std dev	1.6%				Std dev	7.9%

ResNet-50 - Processed resolution

Param	4.76E+07								
Flops	2.90E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	accuracy
1	56.5%	76.4%	57.8%	72.9%	1	7	0	13	67%
2	335.3%	70.8%	299.2%	71.4%	0	8	0	13	62%
3	60.7%	69.7%	60.5%	72.9%	4	4	2	11	71%
4	61.8%	72.0%	63.5%	70.0%	3	5	2	11	67%
5	60.8%	69.2%	61.7%	71.4%	4	4	2	11	71%
			Avrg	71.7%				Avrg	67.4%
			Max	72.9%				Max	71.4%
			Std dev	1.2%				Std dev	4.0%

ResNet-101 - Original resolution

Param	8.55E+07								
Flops	3.04E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	accuracy
1	55.4%	72.8%	59.6%	72.9%	0	8	0	13	62%
2	68.7%	63.1%	57.3%	75.7%	3	5	0	13	76%
3	70.2%	62.1%	60.9%	71.4%	0	8	0	13	62%
4	68.2%	71.8%	65.3%	71.4%	0	8	0	13	62%
5	62.9%	71.8%	60.3%	74.3%	2	6	1	12	67%
			Avrg	73.1%				Avrg	65.3%
			Max	75.7%				Max	76.2%
			Std dev	1.9%				Std dev	6.2%

ResNet-101 - Processed resolution

Param	8.55E+07								
Flops	5.64E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	accuracy
1	227.0%	58.5%	148.0%	75.7%	1	7	0	13	67%
2	71.5%	66.7%	81.0%	70.0%	4	4	1	12	76%
3	77.1%	68.7%	79.7%	71.4%	2	6	2	11	62%
4	84.1%	61.5%	98.7%	70.0%	3	5	3	10	62%
5	62.7%	69.7%	62.1%	77.1%	3	5	1	12	71%
			Avrg	72.7%				Avrg	67.2%
			Max	77.1%				Max	76.2%
			Std dev	3.3%				Std dev	6.2%

ResNet-152 - Original resolution

Param	1.17E+08								
Flops	4.53E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	accuracy
1	62.4%	69.2%	56.9%	72.9%	0	8	0	13	62%
2	56.6%	71.3%	57.7%	74.3%	3	5	0	13	76%
3	102.5%	69.7%	96.2%	71.4%	0	8	0	13	62%
4	60.5%	72.3%	62.6%	71.4%	2	6	0	13	71%
5	57.3%	72.8%	57.5%	72.9%	1	7	0	13	67%
			Avrg	72.6%				Avrg	67.2%
			Max	74.3%				Max	76.2%
			Std dev	1.2%				Std dev	6.2%

ResNet-152 - Processed resolution

Param	1.17E+08								
Flops	8.38E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	accuracy
1	275.3%	72.3%	220.6%	71.4%	0	8	0	13	62%
2	66.4%	69.7%	54.6%	72.9%	3	5	1	12	71%
3	66.9%	64.6%	62.3%	72.9%	1	7	0	13	67%
4	71.1%	68.2%	53.6%	77.1%	2	6	1	12	67%
5	65.2%	67.2%	62.5%	72.9%	2	6	1	12	67%
			Avrg	73.4%				Avrg	66.5%
			Max	77.1%				Max	71.4%
			Std dev	2.2%				Std dev	3.4%

A.2 Full set of EfficienNet results

This appendix gathers the full set of results obtained for EfficientNet Convolutional Neural Network architecture.

EfficientNet B7 - Original resolution

Param	1.29E+08								
Flops	1.53E+11				Test				
	Train		Validation		Defective		Non Defective		Test accuracy
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	
1	48.0%	77%	40.7%	82.8%	5	3	3	10	71%
2	48.4%	76.2%	39.5%	87.5%	4	4	1	12	76%
3	39.8%	83.6%	39.2%	82.8%	4	4	1	12	76%
4	48.4%	75.1%	47.2%	85.9%	4	4	2	11	71%
5	51.3%	76.0%	39.5%	84.4%	5	3	1	12	81%
			Avrg	84.6%				Avrg	75.1%
			Max	87.5%				Max	81.0%
			Std dev	2.0%				Std dev	4.0%

EfficientNet B7 - Processed resolution

Param	1.29E+08								
Flops	4.01E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	on-Defecti	Defective	on-Defecti	accuracy
1	55.7%	72.5%	34.4%	84.4%	2	6	0	13	71%
2	60.1%	68.3%	35.9%	89.1%	3	5	0	13	76%
3	47.6%	77.3%	37.7%	85.9%	3	5	2	11	67%
4	44.9%	74.1%	40.1%	85.9%	4	4	4	9	62%
5	45.9%	78.3%	40.1%	84.4%	3	5	1	12	71%
			Avrg	85.9%				Avrg	69.2%
			Max	89.1%				Max	76.2%
			Std dev	1.9%				Std dev	5.4%

EfficientNet B6 - Original resolution

Param	8.25E+07								
Flops	7.75E+10				Test				
	Train		Validation		Defective		Non Defective		Test accuracy
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	
1	44.9%	79.4%	37.1%	82.8%	4	4	0	13	81%
2	60.8%	67.7%	38.0%	84.4%	2	6	1	12	67%
3	62.7%	73.5%	43.6%	82.8%	1	7	0	13	67%
4	44.0%	78.8%	35.4%	85.9%	5	3	0	13	86%
5	41.3%	82.0%	39.0%	82.8%	4	4	0	13	81%
			Avrg	83.7%				Avrg	75.3%
			Max	85.9%				Max	85.7%
			Std dev	1.4%				Std dev	8.9%

EfficientNet B6 - Processed resolution

Param	8.25E+07								
Flops	2.61E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	on-Defecti	Defective	on-Defecti	accuracy
1	45.5%	80.4%	40.6%	81.3%	5	3	1	12	81%
2	45.9%	79.2%	37.9%	85.9%	4	4	1	12	76%
3	53.1%	75.1%	31.0%	87.5%	3	5	2	11	67%
4	62.5%	69.8%	43.4%	82.8%	4	4	2	11	71%
5	51.4%	79.4%	37.8%	84.4%	6	2	2	10	80%
			Avrg	84.3%				Avrg	74.7%
			Max	87.5%				Max	81.0%
			Std dev	2.5%				Std dev	6.0%

EfficientNet B5 - Original resolution

Param	5.76E+07								
Flops	4.18E+10				Test				
	Train		Validation		Defective		Non Defective		Test accuracy
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	
1	36.0%	85.2%	43.3%	82.8%	2	6	2	11	62%
2	35.1%	82.0%	49.7%	81.3%	4	4	1	12	76%
3	46.4%	76.0%	44.3%	84.4%	4	4	1	12	76%
4	35.2%	85.7%	43.0%	84.4%	5	3	0	13	86%
5	33.5%	84.1%	0,,4371	79.7%	5	3	2	11	76%
			Avrg	82.5%				Avrg	74.4%
			Max	84.4%				Max	85.7%
			Std dev	2.0%				Std dev	8.5%

EfficientNet B5 - Processed resolution

Param	5.76E+07								
Flops	1.83E+10				Test				
	Train		Validation		Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	on-Defecti	Defective	on-Defecti	accuracy
1	58.1%	72.4%	48.3%	76.6%	5	3	3	10	71%
2	37.1%	82.0%	39.8%	79.7%	5	3	2	11	76%
3	48.3%	83.6%	43.9%	79.7%	6	2	3	10	76%
4	47.3%	79.9%	43.5%	78.1%	7	1	3	10	81%
5	46.9%	75.7%	44.3%	82.8%	7	1	5	8	71%
			Avrg	79.3%				Avrg	75.1%
			Max	82.8%				Max	81.0%
			Std dev	2.3%				Std dev	4.0%

EfficientNet B4 - Original resolution

Param	3.58E+07								
Flops	1.79E+10								
	Train		Validation		Test				
					Defective		Non Defective		Test accuracy
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	
1	38.4%	79.4%	37.6%	84.4%	6	2	3	10	76%
2	31.6%	85.2%	40.2%	81.3%	6	2	1	12	86%
3	52.6%	72.5%	44.4%	78.1%	5	3	2	11	76%
4	45.0%	78.3%	42.5%	85.9%	6	2	3	10	76%
5	39.9%	78.8%	43.1%	79.7%	7	1	2	11	86%
			Avrg	81.8%				Avrg	79.7%
			Max	85.9%				Max	85.7%
			Std dev	3.2%				Std dev	5.2%

EfficientNet B4 - Processed resolution

Param	3.58E+07								
Flops	1.17E+10								
	Train		Validation		Test				
					Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	n-Defecti	Defective	n-Defecti	accuracy
1	45.6%	78.3%	50.0%	76.6%	7	1	5	8	71%
2	47.7%	78.8%	46.1%	81.3%	2	6	2	11	62%
3	40.0%	82.0%	42.7%	78.1%	4	4	2	11	71%
4	44.2%	78.3%	37.8%	78.1%	4	4	3	10	67%
5	37.7%	85.7%	41.2%	81.3%	3	5	2	11	67%
			Avrg	79.0%				Avrg	67.4%
			Max	81.3%				Max	71.4%
			Std dev	2.1%				Std dev	4.0%

EfficientNet B3 - Original resolution = Processed resolution

Param	2.20E+07								
Flops	7.48E+09								
	Train		Validation		Test				
					Defective		Non Defective		Test accuracy
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	
1	52.8%	74.6%	57.6%	75.0%	6	2	2	11	81%
2	48.3%	76.2%	60.7%	76.6%	4	4	2	11	71%
3	40.4%	80.4%	47.7%	81.3%	7	1	2	11	86%
4	51.3%	73.0%	55.0%	76.6%	4	4	2	11	71%
5	45.4%	79.9%	55.0%	82.8%	5	3	2	11	76%
			Avrg	78.3%				Avrg	76.8%
			Max	82.8%				Max	85.7%
			Std dev	3.4%				Std dev	6.2%

EfficientNet B2 - Original resolution

Param	1.59E+07								
Flops	4.07E+09								
	Train		Validation		Test				
					Defective		Non Defective		Test accuracy
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	
1	37.0%	82.3%	373400.0%	84.4%	7	1	2	11	86%
2	42.0%	78.3%	34.8%	85.9%	6	2	1	12	86%
3	33.0%	82.5%	34.5%	87.5%	6	2	2	11	81%
4	34.7%	86.2%	45.8%	76.6%	7	1	2	11	86%
5	35.9%	83.6%	35.5%	87.5%	7	1	2	11	86%
			Avrg	84.2%				Avrg	84.7%
			Max	87.5%				Max	85.7%
			Std dev	4.6%				Std dev	2.1%

EfficientNet B2 - Processed resolution

Param	1.59E+07								
Flops	5.15E+09								
	Train		Validation		Test				
					Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	n-Defecti	Defective	n-Defecti	accuracy
1	40.3%	79.4%	42.7%	81.3%	6	2	2	11	81%
2	41.5%	82.5%	42.2%	82.8%	4	4	1	12	76%
3	33.3%	89.1%	38.1%	81.3%	6	2	1	12	86%
4	35.2%	84.1%	41.1%	82.8%	6	2	1	12	86%
5	37.6%	81.5%	43.0%	81.3%	5	3	1	12	81%
			Avrg	81.9%				Avrg	81.7%
			Max	82.8%				Max	85.7%
			Std dev	0.9%				Std dev	4.0%

EfficientNet B1 - Original resolution

Param	1.35E+07								
Flops	2.82E+09								
	Train		Validation		Test				
					Defective		Non Defective		Test accuracy
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	
1	45.3%	77.3%	48.9%	82.8%	4	4	2	11	71%
2	38.1%	79.4%	52.8%	82.8%	4	4	2	11	71%
3	52.8%	77.3%	51.8%	82.8%	6	2	2	11	81%
4	33.8%	82.0%	67.8%	81.3%	5	3	2	11	76%
5	39.8%	78.8%	47.5%	84.4%	6	2	3	10	76%
			Avrg	82.8%				Avrg	75.1%
			Max	84.4%				Max	81.0%
			Std dev	1.1%				Std dev	4.0%

EfficientNet B1 - Processed resolution

Param	1.35E+07								
Flops	4.46E+09								
	Train		Validation		Test				
					Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	on-Defecti	Defective	on-Defecti	accuracy
1	44.4%	78.3%	44.9%	81.3%	5	3	2	11	76%
2	35.7%	84.7%	49.0%	79.7%	5	3	2	11	76%
3	28.1%	88.4%	60.9%	79.7%	5	3	2	11	76%
4	50.8%	75.7%	40.8%	84.4%	5	3	2	11	76%
5	41.4%	77.3%	47.2%	81.3%	6	2	2	11	81%
			Avrg	81.2%				Avrg	77.1%
			Max	84.4%				Max	81.0%
			Std dev	1.9%				Std dev	2.1%

EfficientNet B0 - Original resolution

Param	8.43E+06								
Flops	1.59E+09								
	Train		Validation		Test				
					Defective		Non Defective		Test accuracy
Iteration	loss	Accuracy	loss	accuracy	Defective	Non-Defective	Defective	Non-Defective	
1	32.2%	85.2%	43.7%	84.4%	6	2	2	11	76%
2	35.5%	85.4%	54.4%	81.3%	4	4	2	11	71%
3	34.8%	83.6%	49.7%	81.3%	7	1	3	10	81%
4	34.9%	86.8%	47.2%	82.8%	7	1	2	11	86%
5	34.1%	87.8%	57.8%	81.3%	6	2	2	11	81%
			Avrg	82.2%				Avrg	78.7%
			Max	84.4%				Max	85.7%
			Std dev	1.4%				Std dev	5.4%

EfficientNet B0 - Processed resolution

Param	8.43E+06								
Flops	3.01E+09								
	Train		Validation		Test				
					Defective		Non Defective		Test
Iteration	loss	Accuracy	loss	accuracy	Defective	on-Defecti	Defective	on-Defecti	accuracy
1	41.5%	78.3%	62.3%	81.3%	5	3	2	11	76%
2	35.4%	84.7%	47.3%	79.7%	5	3	4	9	67%
3	36.0%	82.0%	58.8%	79.7%	5	3	2	11	76%
4	49.5%	77.8%	55.4%	79.7%	5	3	2	11	76%
5	41.9%	78.3%	52.0%	78.1%	5	3	2	11	76%
			Avrg	79.7%				Avrg	74.1%
			Max	81.3%				Max	76.2%
			Std dev	1.1%				Std dev	4.3%