Sexual Content Detection on Resource-Limited Devices Using Deep Learning

Detección de Contenido Sexual en Dispositivos de Recursos Limitados mediante Aprendizaje Profundo

TRABAJO FIN DE GRADO
GRADO EN INGENIERÍA INFORMÁTICA
CURSO 2022–2023

Berta de Pablo García

Directores
Luis Javier García Villalba
Sandra Pérez Arteaga

Departamento de Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2023

Agradecimientos

Quiero agradecer a mis tutores de este proyecto Luis Javier García Villalba y Sandra Pérez Arteaga por la oportunidad que me dieron de poder realizar este TFG con ellos y confiar en mí en todo momento.

Estoy especialmente agradecida, por la ayuda y apoyo que me han ofrecido durante todo el desarrollo del trabajo, a Luis Alberto Martínez, Ana L. Sandoval y Daniel Povedano que, además, siempre han estado dispuestos a resolver todas mis dudas a lo largo de estos meses con mucha paciencia.

También a mis amigos de la universidad Iñigo y Víctor, que me han acompañado y apoyado durante toda la carrera y la han hecho mucho más llevadera.

Sin vosotros no hubiese podido realizar este proyecto.

Contents

Index of Figures

Index of Tables

Listings

Acronyms List

AI	Artificial Intelligence
CNN	Convolutional Neural Network
Conv	Standard Convolutional Layers
DA	Data Augmentation
DL	Deep Learning
DNN	Deep Neural Network
DSC	Depthwise Separable Convolutional Layers
DW CONV	Depthwise Convolution Layer
ML	Machine Learning
NAS	Hardware Network Architecture Search
PW CONV	Pointwise Convolution Layer
RGB	Red Green Blue
SGD	Stochastic Gradient Descent

SL	Supervised Learning
UL	Unsupervised Learning

Abstract

Nowadays, access to sexual content published on the Internet is quickly and easily available to anyone. This could harm anyone who does not have control over its consumption and, that is why in this project we want to find a tool that detects and limits it as much as possible. Our main objective is to detect sexual content in images on smartphones because they are the electronic devices that most people have access to. To do this, we retrained the MobileNetV1, MobileNetV2, MobileNetV3-Small and MobileNetV3-Large architectures of the TensorFlow library. This was done with datasets of images with sexual content and images without sexual content and, in this way, we get the models to be able to classify an image as sexual or not. In addition, we use the data augmentation technique to amplify our training datasets. In this way, we also managed to prepare the models to be able to perform good classifications to input images with different characteristics, for example, rotated images, images with lower quality, zoomed images, or, rotated images, among others. The results obtained with different datasets and different modifications of the architecture of the four versions of MobileNet show that the model that performs better and generalizes better when detecting this type of content is MobileNetV3-Small. With this work, we demonstrate that Deep Learning architectures are very useful for detecting explicit content in images and, that they could be used in the future for many different applications.

Keywords: sexual content, convolutional neural networks, MobileNetV1, MobileNetV2, MobileNetV3-Small, MobileNetV3-Large, data augmentation, deep learning.

Resumen

A día de hoy, el acceso a contenido sexual publicado en Internet está al alcance de cualquiera de manera rápida y sencilla. Esto podría llegar a perjudicar a cualquiera que no tenga un control sobre su consumo y, es por eso que, en este trabajo queremos encontrar una herramienta que lo detecte y lo restrinja lo máximo posible. Nuestro principal objetivo es conseguir detectar contenido sexual en imágenes dentro de smartphones porque son los dispositivos electrónicos a los que más personas pueden acceder. Para ello, hemos reentrenado las arquitecturas basadas en redes neuronales convolucionales MobileNetV1, MobileNetV2, MobileNetV3-Small y MobileNetV3-Large con conjuntos de datos de imágenes con contenido sexual e imágenes sin contenido sexual y, de esta forma, conseguimos que los modelos puedan clasificar una imagen como sexual o no. Además, utilizamos la técnica de aumento de datos para ampliar nuestros conjuntos de datos de entrenamiento. De esta forma, también conseguimos preparar a los modelos para que sean capaces de realizar buenas clasificaciones a imágenes de entrada con distintas características, por ejemplo, imágenes rotadas, imágenes con menos calidad, imágenes con zoom o imágenes rotadas, entre otras. Los resultados obtenidos con distintos conjuntos de datos y distintas modificaciones de las cuatro versiones de MobileNet muestran que, el modelo que mejor se comporta y consigue generalizar mejor a la hora de detectar este tipo de contenido es el MobileNetV3-Small. Con este trabajo, demostramos que las arquitecturas de aprendizaje profundo son muy útiles frente a la detección de contenido explícito en imágenes y que podrían ser utilizadas en un futuro para muchas aplicaciones diferentes.

Palabras clave: contenido sexual, redes neuronales convolucionales, MobileNetV1, MobileNetV2, MobileNetV3-Small, MobileNetV3-Large, aumento de datos, aprendizaje profundo.

Chapter 1

Introduction

We will start this paper, in Section 1.1 by stating the main reasons why we decided to investigate and obtain a model that identifies sexual content in images. Then, in Section 1.2 we will explain the context in which this project takes place. After, we will make clear the objectives of our work in Section 1.1. Section 1.4 discusses the work plan we have followed during these months and, finally, Section 1.5 explains how the rest of this project is structured.

1.1 Motivation

The evolution of technology and the Internet has brought with it some serious problems, including the ease with which users can access and share explicit content quickly and easily. While it is true that all of this information could be used to provide young people with safe and healthy sexual education, in real life, unmonitored and uncontrolled access is very worrying as young people are not aware of the harm that accessing this content regularly can cause them, as some researches, such as [SLD+16], have demonstrate.

Some studies conducted at Florida [OBMR12] reveal that adolescents who consume this type of content are more likely to have unrealistic and misleading attitudes to relationships. They also explain that many of the compulsive or aggressive sexual behaviors are directly related to multimedia with sexual content. Another effect of the exaggerated consumption of this content is the fact of being more permissive with incorrect sexual attitudes or, being continually thinking about this and do not give room in their mind to generate curiosity for other more enriching topics. On top, this does not depend on the country, culture, or race, this is something completely globalized.

For these reasons, over the last few years, various computational strategies have been developed to regulate the use of explicit content in adolescents and, above all, in people who may be much more sensitive to this type of videos or images. One of the ways in which

this regularization is done is through the detection of images or videos with sexual content on electronic devices and their subsequent deletion, generation of warnings, or whatever action we want to implement after detecting this content. That is why in this project we are going to try to generate a tool that can identify sexual content in multimedia files.

1.2 Context

This Final Degree Project is part of a research project called Novel Strategies to Fight Child Sexual Exploitation and Human Trafficking Crimes and Protect their Victims - HEROES, approved by the European Commission within the Horizon 2020 Framework Programme (call H2020-SU-SEC-2020) under grant agreement number 101021801 and in which the GASS Group of the Universidad Complutense de Madrid (Grupo de Análisis, Seguridad y Sistemas, https://gass.ucm.es, group 910623 of the catalogue of research groups recognised by the UCM).

In addition to the Universidad complutense de Madrid, 21 organisations from 17 countries are participating in HEROES: 11 from EU countries (Austria, Belgium, Bulgaria, France, Greece, Ireland, Latvia, Lithuania, Portugal, Spain, United Kingdom), 1 associated country (Switzerland) and 5 third countries (Bangladesh, Brazil, Colombia, Peru, Uruguay). These entities are: University of Kent (UK), The Free University of Brussels (Belgium), The French National Research Institute for Digital Science and Technology - INRIA (France), Center for Security Studies - KEMEA (Greece), International Centre for Migration Policy Development - ICMPD (Austria), International Center for Missing and Exploited Children - ICMEC (Switzerland), IDENER Research & Development Agrupación de Interés Económico (Spain), Athena Research Center - ARC (Greece), Trilateral Research and Consulting (United Kingdom), Centre for Women and Children Studies - CWCS (Bangladesh), Center Against Human Trafficking and Exploitation - KOPZI (Lithuania), Portuguese Association for Victim Support - APAV (Portugal), Fundación Renacer (Colombia), The Greek Council for Refugees - GCR (Greece), Brazilian Association for the Defense of Children of Children and Youth - ASBRAD (Brazil), Hellenic Police (Greece), Latvia National Police (Latvia), General Directorate for the Fight against Organized Crime (Bulgaria), Dirección General de la Policía - DGP (Spain), Federal Police (Brazil), Federal Highway Police (Brazil), Secretaria de Inteligencia Estratégica de Estado - Presidencia de la Republica Oriental del Uruguay (Uruguay).

More information is available at:

https://cordis.europa.eu/project/id/101021801
https://heroes-fct.eu

1.3 Object of the Research

Due to the fact that nowadays anyone has access to mobile devices, including children and teenagers, the main objective of the research is to generate a robust Deep Learning (DL) model that allows the classification of images with sexual content that is able to adapt to devices with low resources. This way, it will reduce the risk of exposure to this type of content that a young person may suffer. This type of models are able to learn by themselves essential characteristics of the images. These can be general, such as those perceived by the eye of any human being, for example, objects, landscapes, certain shapes or colors, or, more specific ones that we ourselves cannot identify, for example, specific shades of colors, tiny patterns that are repeated in the images (such as a stain in a medical image) or textures (such as the skin of animals). For those reasons, we think it is a great tool to use for identifying sexual content on images.

1.4 Working Plan

Here we will show the three main phases of the project. Figure 1.1 shows the different times in which these phases were carried out and, in addition, when the writing of this report started.

1. **Investigation**: during the first three months I had to familiarize myself with the context of the overall research project and, above all, with the functioning of DL algorithms. For this purpose, during these three months I was taking an introductory course on Machine Learning (ML) and reading and summarizing articles related to the state of the art of the project. Once I had a general idea of what was being done in the project and how DL models worked, I had to start investigating which DL architectures I could use to develop my project since I had to find one that could be exported later to cell phones and, that could detect patterns in images. Throughout this research process, we held weekly meetings with my tutors to see if I was on the right track with the articles I was reading and, above all, to discuss any doubts I had about any of the topics I was studying.

2. **Development**: thanks to all the previous research, we managed to find the DL models we were going to use. We started doing small tests with them in Google Colaboratory to see how effective those models were and how they behaved. But, to start doing retraining on our larger datasets we had to start testing them in Visual Studio with the TensorFlow and Keras libraries. In addition, we were fortunate to be able to use a GPU that managed to reduce the retraining time by many hours. During all this phase we also made a deep understanding of the architectures of the 4 models as it was the basis of the project.

3. **Experimentation**: after all the re-training we did, understanding how each model worked and obtaining different metrics with each of them, we were able to select the one that performed the best. We started to test how it behaved against images that the algorithm had never seen to see if it was able to generalize well and find sexual content in them or not. This experimentation process was important to see if in the future it could be viable for the model to be used for different applications on cell phones.

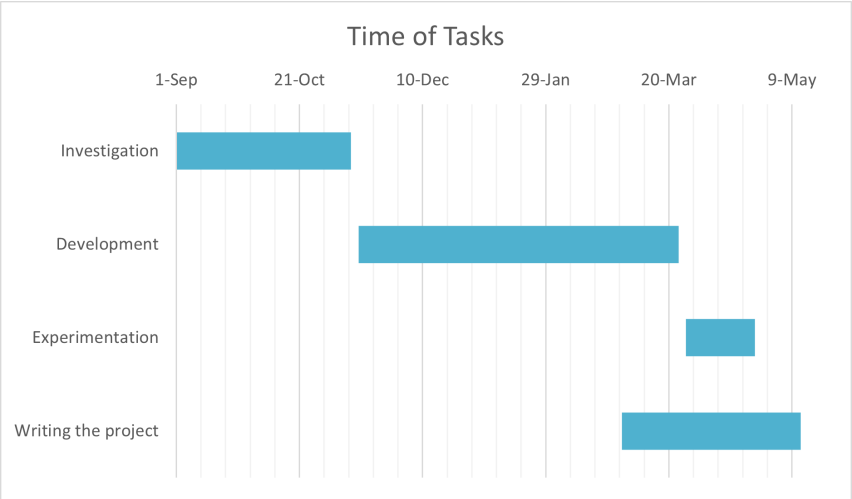


Figure 1.1: Diagram of Gantt of the Working Plan

1.5 Working Structure

The project is divided into 6 chapters (the first one being this one), as follows:

Chapter 2 explains some of the main concepts that are necessary to know in order to understand the rest of the project.

Then, in Chapter 3 we review the state of the art explaining previous studies and tools that have been used to detect sexual content in images, the in-depth explanation of the architectures that we will use in our retrainings and, finally, some previous papers that have used the technique of Data Augmentation (DA) to improve their models of DL.

A thorough explanation of the methodology of the project is given in Chapter 4. First, we make indications about what to do before retraining a model and, then, all the retraining performed with each architecture and the metrics obtained with them.

Then, in Chapter 5 we discuss and analyze the various tests and experiments we performed on the model we consider to be the best performing.

Finally, in Chapter 6 we discuss the conclusions obtained with the work done and the possible future work that could be done based on our project.

Chapter 2

Research Context

This chapter introduces the context in which the work is developed. Section 2.1 is a brief introduction to Machine Learning inside the Artificial Intelligence field then, in Section 2.2, we introduce the Deep Learning branch of Machine Learning focusing mainly on Convolutional Neural Networks and how convolutional operations work. In Section 2.3 we explain how a Deep Learning architecture is trained and what parameters are used to improve the learning of a model. We also introduced the interpretability concept in Section 2.4 with some interesting tools and, finally, in Section 2.5 we explain how to convert computer models to mobile phone models.

2.1 Introduction to Machine Learning

Within the broad field of Artificial Intelligence (AI) there exists ML. AI, in general terms, is the ability of computer systems, whether cell phones, computers, cars or other electronic devices, to be able to perform tasks automatically. One of the big issues facing AI today, as reviewed in [Fje20], is the ability to perform many tasks correctly, what is called "General Artificial Intelligence". This is a great field of study since, getting an algorithm to be able to perform many tasks would be like imitating what the human brain does. So far, the best advances that have been achieved are on the resolution of specific tasks, which is called "Specific Artificial Intelligence". These tasks include, among many others, being able to play video games, voice assistants that attend to tasks suggested by users, management of traffic lights and road traffic, predictions of any kind by analyzing large amounts of information, recommendation systems to satisfy users' tastes in music applications, videos, series, books, or text translations from one language to any other. As accustomed as we are to have all these tool available on our daily basis, they are still build on complex artificial intelligence algorithms.

The ML branch is one of the most interesting in the field of AI, as deeply explained in

[Zho21], because these are algorithms that manage to learn new forms of behavior on their own, i.e. they do not need the help of a person to reprogram themselves. As they acquire and process new data, ML algorithms manage to shape themselves and learn autonomously. Thus, we can say that an important part of ML algorithms is the data they use to learn. The higher the quality of the information received by an algorithm is, the better it will perform. Within ML there are two distinct types of learning, Supervised Learning (SL) and Unsupervised Learning (UL). These differ mainly in the input data, i.e., in the information that reaches each algorithm for learning.

- **Supervised Learning**: in SL algorithms, the input data consists of all its features and its actual label [CCD08]. For example, if we want the algorithm to predict the price of a car, the input data will contain all the essential characteristics of the car to establish a price (such as: the number of horsepower, engine specifications, brand or number of doors and seats) and also, we would have in the input itself the actual price of that car. In this way, after the algorithm has been trained on a lot of input data, it will have been able to learn what are the essential characteristics it has to look at to establish a price and also to associate certain characteristics of the car with certain prices. In this way, when we do not know the price of a car but we do know all its properties, we can pass them to the algorithm and it will predict an estimated price based on previous examples.

- **Unsupervised Learning**: in this case the data entered into the algorithms is not associated with any real label, i.e. the input is known but not the output [Gha04]. Continuing with the example of the car, this time the input data to the algorithm training would contain the characteristics of the car but not its real price. The goal of this algorithm then is to find patterns within that data that will help to divide it in distinct groups. In this way, after the algorithm has been trained with different data, when it receives a new input data it will be able to include it in one of those groups. Since it does not know the labels that can be applied to this data, the UL algorithms will include the input data in a group and will label that group with the name it thinks its more accurate. One of the best known UL algorithms is clustering.

2.2 Deep Learning

Within ML there exists DL (reviewed in [LBH15]) this algorithm is an advance of ML based on deep artificial neural networks, unlike ML that uses algorithms such as decision trees, regression algorithms or clustering algorithms, among others. The main idea behind the creation of DL algorithms was to mimic the functioning of the human brain, although it is still far from being achieved. In DL algorithms we also have neurons and neural

connections, a simple representation of what the structure of DL is like would be the one on Figure 2.1 (the circles would imitate the neurons and the arrows the neural connections):

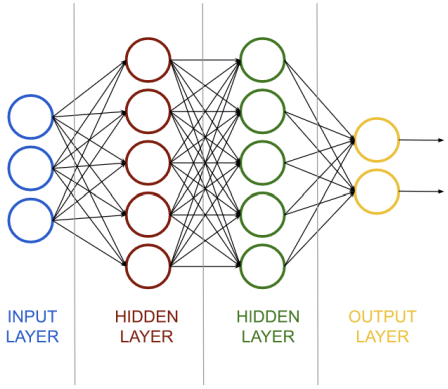


Figure 2.1: Representation of a DNN

We will explain with a simple example how a DNN works. Let's say that the DL algorithm has to predict from input data whether a house is likely to be sold or not. These input data would be the ones that would provide information to the neurons of the input layer. The intermediate layers would be in charge of selecting the important values to make the predictions. In addition, they could combine these values to get more information about the house and thus, get a better prediction of the probability of purchase. If, for example, we have as input the price of the house, the location and the square footage we could combine that information and infer more knowledge as follows:

- Location combined with price: if it meets the average price of the area or not.
- Square meters: number of bedrooms and bathrooms.
- Location: proximity to areas of interest.
- Price: average years of mortgage.
- Location: public transportation available.
- Square footage combined with price: if it the meets average price per square meter.

The first hidden layer will then have 7 neurons, each of them will be in charge of calculating one of the above points, combining or not some of the inputs. After generating this new information, it will be passed to the next layer. This last layer, the output layer, will join all this information and according to all this data will predict if the house has a high chance of selling or not. Figure 2.2 shows how the neural network would look like. All the outputs generated by each layer are called "activation values" and the functions within each neuron that generate new information from the input data are called "activation functions". Basically the intermediate layer analyzes and processes the input information

in order to get a reliable answer. In a real DL algorithm, the number of hidden layers is much higher than 1.

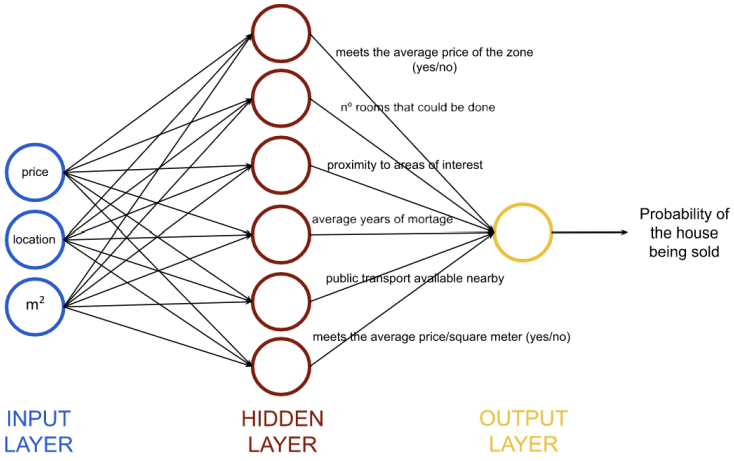


Figure 2.2: Content example of a DNN

2.2.1 Convolutional Neural Networks

Within DL neural networks there exists what are called Convolutional Neural Network (CNN), usually used for image analysis or classification as explained in [ON15]. This type of neural network is able to identify important patterns and characteristics in the images and then classify them into one class or another. What is different about these networks is that most of them are formed by layers called Standard Convolutional Layers (Conv) that have a pattern identification mechanism. Although these layers are the majority in quantity forming the CNN networks, they can also coexist with other types of layers and, in fact, they usually do. These layers behave like any other hidden layer; they receive an input data from the previous layer, process it with a function called convolution operation, which we will explain later how it works and, once the transformed input is obtained, it is passed to the next layer.

As explained in [AMAZ17], convolution operations use filters, which are the ones that actually find and identify the patterns in the images. Each of these filters focuses on identifying one of the image features that the model needs in order to further classify an image, for example, a filter can be dedicated to identify corners, circles, horizontal or vertical lines, edges, squares or textures, among many others. In addition, it is important to note that, as explained in [NVK+15] the initial convolution layers have filters that identify simple features of the images such as those just mentioned, but, as you go deeper into the network, the filters that are applied to the images are dedicated to identifying other types of more complex patterns, such as, objects, faces, animals, backgrounds or landscapes, among others. In each layer you can use as many filters as patterns/characteristics you want to identify. In addition, as we will explain now, each of

these filters will be applied individually to the input.

To explain a convolution operations it is first necessary to know that a filter, also known as a kernel or convolution filter, is a 3-dimensional matrix of size $height \times width \times depth$, where the depth is equal to the number of channels of the input image. In our case, the depth will be equal to three because we will use color images, which are represented by Red Green Blue (RGB) channels. Later we will explain the possible values of $height \times width$ that the kernel can have since it is necessary to know how Conv works to understand the consequences of setting these variables to one value or another.

Now, we will go on to explain how the usual convolution operation works [AMAZ17]. The process involves multiplying the weights of the kernel with the pixel values of the image and adding them together (Figure 2.4). Normally, the size of the kernel is smaller than the size of the image so, in order to extract the most important features of the image, the kernel traverses it from left to right and from top to bottom to scroll through all the pixels in that image. The following Figures (2.4 and 2.5) show an example of how the kernel goes through an image, performs the necessary operations, and returns as a result what is called a feature map. The feature map is a matrix that represents the most important features of the image, in short, the kernel has filtered the initial image and has returned a matrix with the most interesting features of it.

0	1	1	1	0	1
1	0	1	1	0	1
0	0	1	0	0	1
0	0	1	1	1	0
1	1	0	0	0	1

(a) Initial image

1	0	1
0	0	1
0	1	1

(b) Kernel

Figure 2.3: Initial Image (5x6) and Kernel (3x3) examples

0*1	1*0	1*1	1	0	1
1*0	0*0	1*1	1	0	1
0*0	0*1	1*1	0	0	1
0	0	1	1	1	0
1	1	0	0	0	1

(a) First Operations

0	1*1	1*0	1*1	0	1
1	0*0	1*0	1*1	0	1
0	0*0	1*1	0*1	0	1
0	0	1	1	1	0
1	1	0	0	0	1

(b) Second Operations

0	1	1	1	0	1
1	0	1	1	0	1
0	0	1	0*1	0*0	1*1
0	0	1	1*0	1*0	0*1
1	1	0	0*0	0*1	1*1

(c) Last Operations

Figure 2.4: First, Second and Last multiplications, applying the Kernel to the Initial Image from Figure 2.3

The values in Figure 2.5 are taken from the sum of the multiplications made in each operation, thus, in Figure 2.4(a) we will obtain the following sum $0*1+1*0+1*1+1*0+0*0+1*1+0*0+0*1+1*1=3$ which gives us the "3" of the top left cell and, in Figure 2.4(b), is where we will obtain the "4" ; $1*1+1*0+1*1+0*0+1*$

3	4		
			2

Figure 2.5: First, Second and Last value of the Feature Map

$0 + 1 * 1 + 0 * 0 + 1 * 1 + 0 * 1 = 4$. And, for Figure 2.4(c), we have the following sums $0 * 1 + 0 * 0 + 1 * 1 + 1 * 0 + 1 * 0 + 0 * 1 + 0 * 0 + 0 * 1 + 1 * 1 = 2$, whose result belongs to the last cell in the feature map matrix.

As you can see, the kernel moves 1 position at a time. It moves to the next row when its rightmost column matches the rightmost column of the image matrix. And, the mathematical operations end when the rightmost column and the last row of the kernel match the rightmost column and the last row of the image, Figure(2.4(c)). After all these operations are performed, we will get a feature map with a size equal to 3x4, which is smaller than the initial size of the image 5x6 as we were expecting. Finally, after all the sums and multiplications, we will have the feature map of Figure 2.6 as a result.

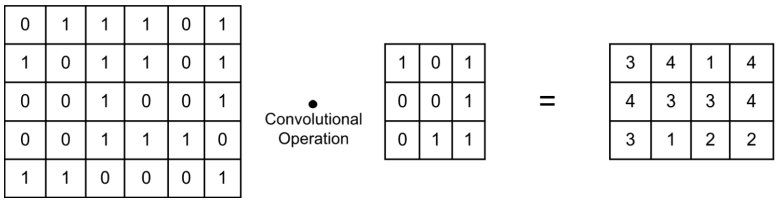


Figure 2.6: Final Feature Map obtained

The previous example was on an image with only 1 channel, i.e. with depth 1, but, as the training images used in our case are in color, there will be three input channels corresponding to the RGB channels. So, the kernel will have a size of 3x3x3 instead of 3x3x1 because the depth of the kernel must be equal to that of the input image. However, the feature map obtained will have the same size as the one we obtained with the 3x3x1 kernel, as can be seen in the following Figure 2.7.

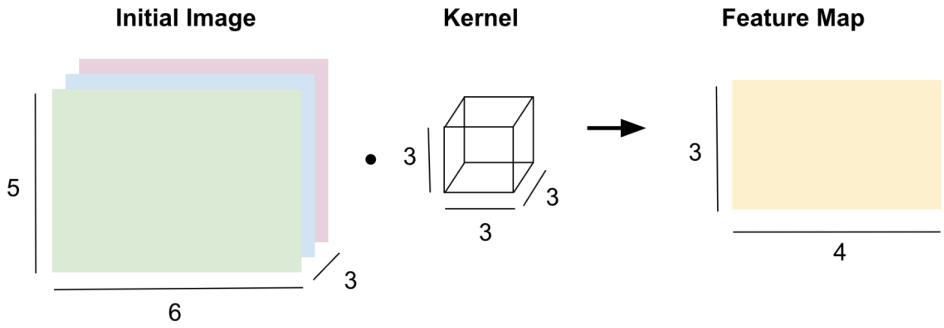


Figure 2.7: Convolution Operation with Input Image and Kernel of depth 3

Here, in Figure 2.8, is an example of how the value of the first pixel of the feature map is extracted (the same is done for the rest of the pixels of the feature map, and for that, we have to move the kernel through the initial image as we did in the previous example):

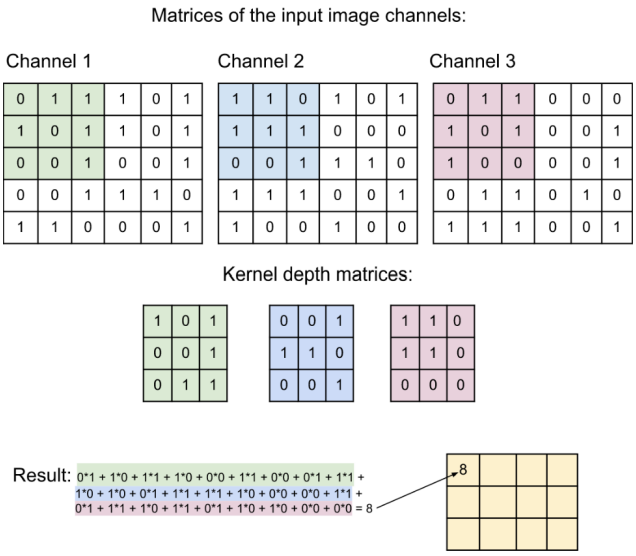


Figure 2.8: First value of the Feature Map calculated with Input Image and Kernel of depth 3

So, convolution layers perform the process of filtering out the most important features and combining these features, in order to produce a new set of image characteristics, in the same layer. The first process is done when each of the layers of the kernel goes through its corresponding channel of the image (from left to right and top to bottom doing the scalar multiplications). The second process of combining the features is done when the sums of the three scalar multiplications obtained above are done (these sums obtain a single output pixel which will belong to the final feature map), this combination is done so that the model can learn new features and patterns.

Now that we know how convolution operations work, we will explain the possible values that the height and width of a kernel can have. Let the stride be equal to 1 in this explanation. If we set the kernel to have a small size, for example, 1x1, the feature map obtained will have the same size as the initial image and, if we have a kernel with the same size as the image, we will obtain a feature map of only 1 pixel. The good thing about large kernels is that the computational time is much lower since much fewer multiplications would have to be done (the kernel would not be able to move through the image). The problem is that, if we use a very large kernel we will lose the majority of the details of the image because the feature map obtained would have few pixels and, therefore, little information. This would lead us to the model suffering an under-fitting and not being able to predict the class of any image correctly, since it really did not have learned anything or almost nothing. But, if the kernel has a very small size, such as 1x1, a large amount of image features will be obtained in the feature map, actually, the kernel would not be

filtering anything, since that feature map would have the same size as the input image. Also, the learned features would be local, that is, they would not have any relation with the neighboring pixels (because the convolution operation has been done pixel by pixel and not combining several pixels). All this could lead to over-fitting, that is, the model learning too much and this would generate a problem for the model to generalize, i.e., when it receives an image completely foreign to its training dataset, an image it has never seen, it would have a hard time identifying determining features and classifying the image. So, the decision of the kernel size depends on what we want to achieve doing the convolution operations.

2.3 Training Deep Learning Architectures

The training of an architecture, as well explained in [SM19], is the process in which it learns to identify patterns or characteristics of the input data. To evaluate whether this model is behaving as it should and making correct predictions, we have a series of parameters and a function that check it.

First, we have the two essential trainable parameters that are modified and improved throughout the training of the architecture and help to improve the behavior of the model. These are the weight (w) and the bias (b). In each neuronal connection between one neuron and another there is a weight w. What this weight indicates is the intensity of the connection between one neuron and another so, it really show how important is the result of a neuron with respect to the objectives to be achieved by the training process. Then we have the bias, this value indicates whether the neuron tends to be active or inactive. The value of the weights of the neuronal connections and the values of the bias of the neurons start out random. Figure 2.9 is a representation of the computations performed at the output of the neuron. X are the inputs, W the weights of each neuron connection, b is the bias and "y" is the output.

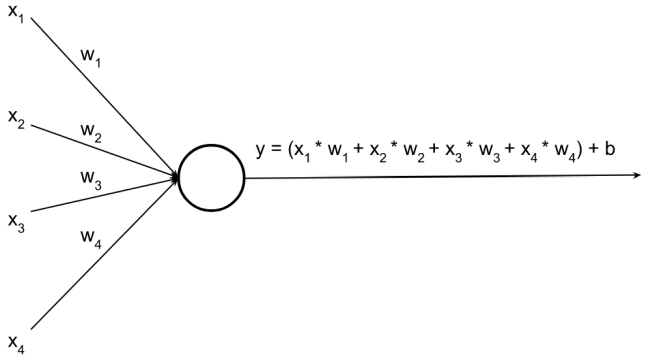


Figure 2.9: Operation made in a neuron

We can see that the result of "y" is a linear function and, it is important to point out that,

in order to increase the representation power of each neuron, we must apply a non-linear function, also known as non-linear activation, to that result. For example, normally the last layer of a neural network has as many neurons as possible classes because each of those neurons is associated with each of the classes. So, it would be very useful for the output of each neuron to return what is the probability that the input data belongs to the class with which it is associated. For that, we apply the non-linear function softmax, which converts the output of each neuron into the probability of belonging to its class.

To find out if the model parameters w and b are good, the loss function is used. This function compares the prediction made by the model with the actual label of the data and, if the model was right this function will get a low value and if it was wrong the function will get a high value. This loss function is expressed as in Equation 2.1.

$$\frac{1}{m}\sum_{m}^{i=1}\left[L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})\right] \tag{2.1}$$

Where m is the number of input data, L is the loss function, i.e., the one that indicates how good the parameters w and b are for a single input $\vec{x}^{(i)}$ based on the model prediction and the actual class of the input. The function f is the transformation applied to the input data $\vec{x}^{(i)}$ and $y^{(i)}$, so $f_{\vec{w},b}(\vec{x}^{(i)})$ is the final output of the model and finally, $y^{(i)}$ is the actual output of that input.

So the fundamental objective of the training is to minimize the output of this loss function. For this purpose, the parameters w and b of the model are improved. Before modifying the values of w and b, the backpropagation algorithm is performed. This algorithm consists of propagating from the output to the input the result of the loss function and at the same time calculating how much each weight of each neural connection has contributed to the decision of the last layer of the network. Once this has been calculated, the next step is to improve those weights with the gradient descent algorithm.

The gradient descent algorithm works as follows. Let's say we have the following loss function, Figure 2.10 that depends only on one weight (w) applied in the model, i.e., we have only one value as input data and we assume that bias is equal to 0.

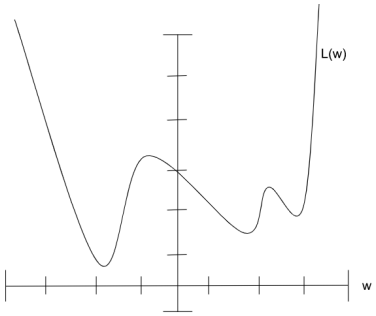
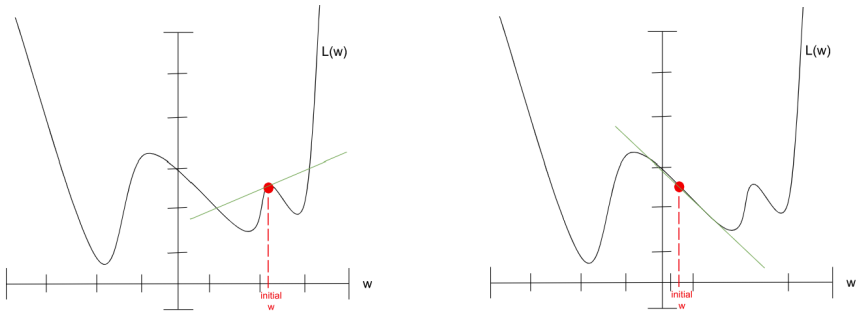


Figure 2.10: Example graph of a loss function that depends on w

As we can see this function has several local minima and a global minimum. Depending on the initial value of w we can end up in a local minimum or in the global minimum, it is not always guaranteed to reach the global minimum. If for example w starts at the position of the Figure 2.11(a) and we calculate its slope, it will be positive. Since it is positive, to reach a lower value of the function we must move to the left, that is give a lower value to w. If w starts at Figure 2.11(b) and we calculate its slope we will see that this is negative, in this case w will have to move to the right, increase its value, to reach a lower point of the function L(x). Therefore, the result of the slope will be calculated as the value of w is updated. When we reach the minimum we will see that the result of the cost function will not improve by decreasing or increasing w, so we will stay in that local minimum.



(a) (b)

Figure 2.11: Graphs with distinct initial value for w

In addition, w decreases and increases as a function of the slope, the straighter the slope, the less w changes. If we have more than one input value, we will no longer have only two dimensions, but several, so the model must find out in which direction to move within that space to reach the lowest point of the loss function (gradient descent algorithm). To do so, it uses optimizers such as SGD, Adam or others that take care of modifying the values of w and b to reach the minimum value for the loss function in the most efficient way.

2.4 Interpretability

One of the properties that a ML or DL algorithm should meet is that the outputs obtained from them can be interpreted in some way, what is called Explainable Artificial Intelligence (XAI). It is important to know and understand why one decision has been made and not another since this type of algorithms is often used in important fields such as medicine, science, in trials or in the economic field, among many others, as explained in [CTR+17]. For example, if a bank uses such algorithms to help it decide whether to grant a loan to a customer or not, it must say what characteristics it has looked at and what conclusions

it has drawn to say yes or no to a loan application. In this way, interpretability adds veracity and makes any user to trust the algorithm being used.

In this project we are going to work with images and we will want the DL models to classify them into some classes or others, that is why we will briefly review three tools that extract heat maps from the images. These heat maps will indicate the areas of the images where the algorithms have been more or less focused to classify the image into a certain class. This is a way to understand better the model and see why it has classified an image into one class or another one. Before explaining the three tools, we must say that each of the example heat maps that we are going to show were obtained classifying the same image of a cat with the MobileNetV3-Small model loaded from TensorFlow (which is trained over ImageNet). So, the three tools are the following:

- SHAP: is the acronym of Shapeley Additive exPlanations, this tool uses calculations from the field of game theory to obtain the characteristics in which our DL algorithm is focusing more in order to classify an image. The heat map obtained with the SHAP tool is showed in Figure 2.12 and, in order to implement this tool we use the following code:

```
background = x_train[np.random.choice(x_train.shape[0},
                                      50, replace=False)]
explainer = shap.GradientExplainer(new_model, background)
shap_values = explainer.shap_values(x_test)

#function that displays the heat map of each class
shap.image_plot([shap_values[i][0] for i in range(5)],
        np.uint8(x_test)[:][0],
        ['Egyptian_cat', 'bow_tie', 'tabby',
        'lynx', 'tiger_cat'])
```

Listing 2.1: SHAP implementation

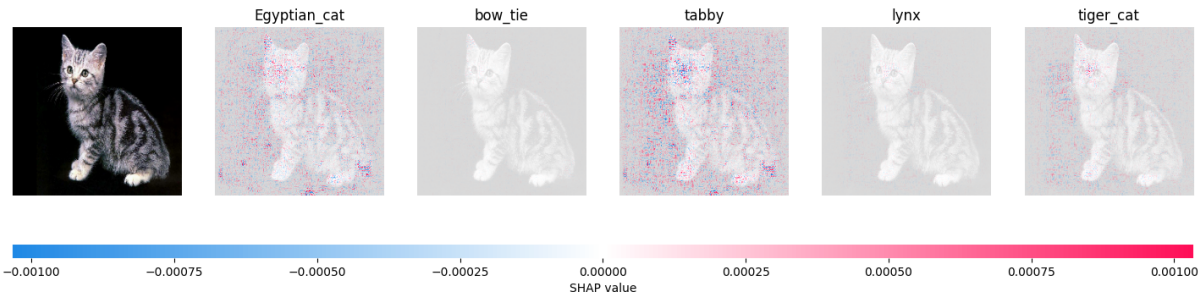


Figure 2.12: SHAP output

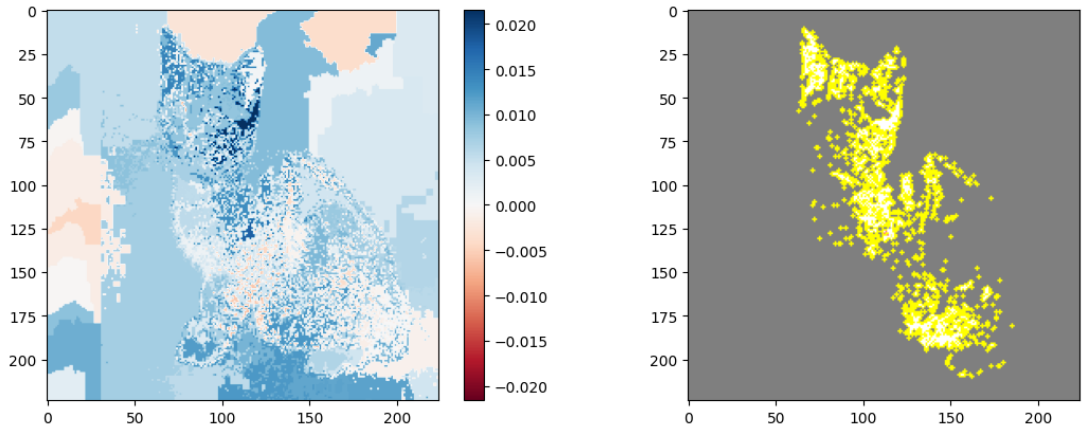
- LIME: is the acronym for Local Interpretable Model-Agnostic Explanations. It is an AI used to explain DL models and, among them, those that classify images. As we know, a DL algorithm dedicated to classifying images manages to detect patterns

such as edges or lines and also, regions of interest, i.e., sets of pixels that form more complex figures such as faces or objects. What LIME's AI does is to also identify these regions of interest, just like a DL algorithm in its deepest layers, and detect whether these sets of pixels are useful for image classification or not. The nice thing about this tool compared to SHAP is that it is much faster. The heat maps obtained are shown in Figure 2.13 and the following is the code we used to test this tool (after importing the LIME library).

```
explainer = lime_image.LimeImageExplainer()
explanation = explainer.explain_instance(img_array[0].astype('double'),
new_model.predict, top_labels=5, hide_color=0,
num_samples=100)

def explanation_heatmap(exp, exp_class): #generate the heat-map
    dict_heatmap = dict(exp.local_exp[exp_class])
    heatmap = np.vectorize(dict_heatmap.get)(exp.segments)
    plt.imshow(heatmap, cmap = 'RdBu', vmin  = -heatmap.max(),
           vmax = heatmap.max())
    plt.colorbar()
    plt.show()

#call that generates the colored heatmap
explanation_heatmap(explanation, explanation.top_labels[0])

temp, mask = explanation.get_image_and_mask(
explanation.top_labels[0],
positive_only=True,
num_features=10, #10 top important features
hide_rest=True)

#call that generates the grey and yellow heatmap
plt.imshow(skimage.segmentation.mark_boundaries(temp / 2 + 0.5, mask)
```

Listing 2.2: LIME implementation

- Grad-CAM: this tool focuses on the value of the algorithm weights to identify to which features are given more importance by the algorithm, especially in the last layer of the model. To subsequently generate the heat map, it focuses on the output feature maps of the model and, with the help of the weights found for each of the features that represent each of the layers of the feature map, we obtain the mask that will be applied to the image to obtain the heat map. The Figure 2.14 obtained after applying this tool. The pixel map obtained is really useful to be able to state that our model is focusing on the right sets of pixels to classify an image as one class or the other. Some notes on the following code used are that, the "make_gradcam_heatmap" and "save_and_display_gradcam" functions are directly obtained from the official Keras page "Grad-CAM class activation visualization".

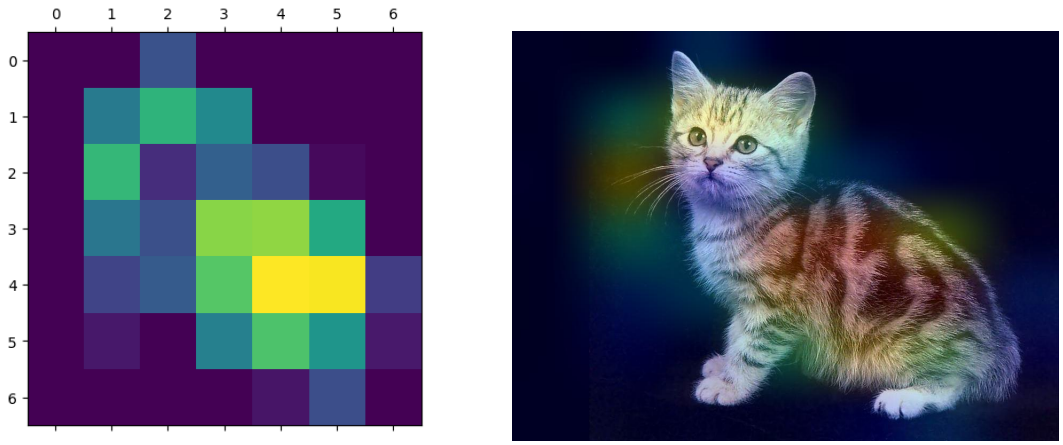


(a) (b)

Figure 2.13: LIME outputs

```
#generate class activation heatmap
heatmap = make_gradcam_heatmap(img_array, new_model,
           last_conv_layer_name)

#display the first heatmap, colored big pixels
plt.matshow(heatmap)
plt.show()

#display the second heatmap, with the original image
save_and_display_gradcam(img_path, heatmap)
```

Listing 2.3: Grad-CAM implementation



(a) (b)

Figure 2.14: Grad-CAM outputs

2.5 TensorFlow Lite

One of the objectives of the project is to use DL models that can later be transferred to mobile devices. In order to transform the model trained on the computer to a smartphone, we suggest using the TensorFlow Lite library that manages this transformation. The following code is the required one to make the conversion:

```
tf.saved_model.save(model, 'folder_path')

#creates a TFLiteConverter object from a SavedModel directory.
converter = tf.lite.TFLiteConverter.from_saved_model('folder_path')

#converts a TensorFlow GraphDef based on instance variables.
#returns: The converted data in serialized format.
tflite_model = converter.convert()

os.chdir('folder_path')
with open('model.tflite','wb') as f:
    f.write(tflite_model)
```

Listing 2.4: Transform model to TensorFlow Lite

Chapter 3

State of the Art

The sections that make up this chapter are the following ones, Section 3.1 discusses the previous computational mechanisms that have been performed to detect explicit content in photographs. Then, an in-depth explanation of the DL architectures that will be used in this project is given in Section 3.2. Specifically, this section is divided into three subsections, the MobileNetV1 architecture 3.2.1, the MobileNetV2 architecture 3.2.2 and, last but not least, the MobileNetV3 architecture 3.2.3. Finally, in Section 3.3, we define and explain what the Data Augmentation technique is and what it is used for along with some previous studies that have used it.

3.1 Explicit content detection

One of the first tools was developed in 1996 [FFB96], they were able to detect naked bodies based on the color and texture of the skin and, to determine whether those groups of pixels were part of the human body or not, they implemented an appropriate and effective model which was able to detect human body shapes. This system displayed 52% recall and 60% precision which is a decent result but, the amount of time it took to detect explicit content in the image was more than six minutes, which does not satisfy the time-efficiency expectations.

The fact that an image is only identified as sexual or non-sexual based on the amount of skin that appears in the photo was not very accurate and that is why there were other alternatives to the previous one. In 2008, [DPN08] implemented a bag-of-visual-words (BOVW) based model that was used previously in [CDF+04] for object classification in images and subsequent categorization, and obtained good results. The steps followed by this algorithm were as follows: (1) identification and description of image patches (patches are sets of pixels that form an image pattern) (2) these patch descriptors were assigned to a set of predetermined clusters which were ”visual words” that denote the local features

extracted from the images (these words were part of a vocabulary with which the model trained to learn from it) (3) a "bag of keypoints" is constructed which counted the number of occurrences of image patterns (i.e., the number of patch descriptors assigned to each cluster) in a given image and is represented as a histogram and, finally, (4) depending on how many patch descriptors were assigned to each cluster, a multi-class classifier chose the category or categories of the image. The outcomes were acceptable, but they would have improved if the visual words were combined with the skin color characteristics.

Another example of detection of explicit content was developed in 2010 by [SZA11] where they used the HSV model. It transformed RGB images into HSV images, i.e., the images instead of being combinations of red, green, and blue, they were combinations of the hue, saturation, and color value of each pixel. They observed that after transforming images from RGB to HSV, the part of the skin in the image turned out with different color than the rest of the objects, and thus, a simple filter could be used to detect skin areas. This filter would use certain values of Hue, Saturation and Value, namely $H > 0$ and $H < 0.25$, $S > 0.15$ and $S < 0.9$, $V > 0.2$ and $V < 0.95$, to determine whether a pixel had the skin color or not. The problem is that they rely solely on the percentage of skin that appeared in the image to classify it as unsafe (specifically, with more than 50% it would already be classified as sexual content), it was still a challenge to detect whether that area of skin that appeared in the image was actually a sexual organ or not. So again, as in [FFB96], this method could fail with images of people in swimsuits that are totally safe.

Another approach was done in 2011 [VdALJ+11] using local descriptors which have been demonstrated to be very resistant to noise, are usually not affected by irregularities in the dataset and have high discrimination power in [SLC04]. Among others, they used one of the most famous local descriptors called SIFT, which detects points of interest. The results obtained were quite good, out of 10 videos with sexual content that they checked, they managed to identify 9 correctly and get very few false positives. In addition, the false positives were again due to videos of people on the beach or breastfeeding sequences ones, where sexual organs may even appear.

Additionaly, in 2018, from Mataram University in Indonesia [HWA18] they proposed to create an application for any Android mobile device that aimed to prevent and moderate the dissemination of erotic content material from the source. This application used a matching pattern method to detect explicit content at the same moment a photo was being taken and, in the event that intimate areas of the person appeared, the image would be saved in the mobile's gallery with red circles censoring those parts. To implement the pattern-matching mechanism they used an external XML file that served as a parameter file, i.e., this file was compared with the examined image and, based on the number of similarities found, the image was classified as secure or non-secure. The results obtained indicate that the method is efficient and effective as they were able to detect intimate areas in the images in less than one second. The effectiveness of the algorithm depend mainly

on the accuracy of the XML files, but the research did not pretend to offer us a new super accurate method of detecting explicit content in images, but a possible application of our models.

In the same year, [QWW+18] presented a way to distinguish swimsuit pictures from pornographic ones and thus, solve one of the biggest problems we encounter when classifying images as safe or unsafe. Their procedure started with the extraction of salient features from the image for later use in classification. For example, pixels that had the skin color. They established a series of thresholds in the RGB and YCbCr color space values that a pixel must had in order to be considered a skin-color pixel. This process managed to extract areas of skin in the image. The second and innovative step they did was to recognize faces, since in portraits there are usually many pixels with the color of the skin so the algorithm could get confused and classify them as sexual content. Once the model recognized the faces it counted how many there were and how much image space they occupied. The method continued with the classification phase, where they trained several machine learning models with images belonging to one of the two classes they wanted to detect (sexual content and swimsuits). They used Multilayer Perceptron, Decision Tree, Random Forest and Logistic Regression, and the best results were obtained with Random Forest, where they achieved 96.96% accuracy and 94.31% F1-score with the AIIA-PID4 dataset.

Finally, recently, in 2022, [KSCJ22] presented a CNN to detect adult content on images. They used what is called "Gaussian-Bernoulli limited-time" to extract the most relevant features of the images. All of these features extracted were compressed with the Boltzmann machine. This approach allowed them to achieve 99.16%.

3.2 MobileNet

The following CNN models were created for use on mobile devices as they have far fewer hyper-parameters than the other models seen above. By occupying much less memory space, they can be transferred from a computer to a cell phone and meet the resource constraints of the latter. In addition, apart from being small size models compared to the ones we have seen previously, they meet the speed expectations that any user expects in an application.

Before explaining each architecture, we must point out that these models have been used previously for other studies in different fields obtaining very good results. For example, the models that we used were used to classify skin lesion images [CGP21], skin disease images [VPA+19] or skin cancer images [WHW20], to recognize facial expressions [SS19] or to identify which kind of garbage an image has [RCS+18], among many others. So now, in the next three sections we will explain in detail the architectures we used.

3.2.1 MobileNetV1

One of the major advances in the MobileNetV1 architecture is the use of Depthwise Separable Convolutional Layers (DSC) instead of Conv. Both kind of layers are used to extract the most salient features from the images, the same features that the model will later use to classify an image into a class. The problem with Conv is that they add a lot of mathematical operations, in particular multiplications, to the model training execution. But, as they are in charge of extracting the most important features from the images, we cannot do without them. So, the solution for the MobileNetV1 architecture was to replace them with the DSC which, achieves the same results, but with much less mathematical operations.

To decide which spatial size the filters have, MobileNetV1 architects took into account that the goal is to extract important features from the images without making the model overlearn. So, they choosed to use a kernel of size 3x3. A good justification of why this size is a good decision is given in [OOAS18], they made tests with kernels of size 3x3, 5x5, 7x7 and 9x9 and obtained the best results with a 3x3 kernel. We can see those in the graphs at the end of their paper where they show how the error function of training and validation are even, they never cross or deviate and, the error of training is always a little above that of validation, which indicates that the model is not over-learning. So, we can say that a good decision was made regarding the size of the kernel in MobileNetV1.

The new method proposed for MobileNetV1 to reduce the number of multiplications in the Conv was born from the idea of separating the spatial operations (height and width) from the depth operations. This new way of performing convolution operations, DSC, is divided into two different layers, Depthwise Convolution Layer (DW CONV) and Pointwise Convolution Layer (PW CONV).

- **Depthwise Convolution**

  This layer is where the filtering of the image is done i.e., where the most important features of the image are obtained and. We will adjust the explanation to our tests, where we use color images (which have three channels). The procedure starts by separating the input image channels as if they were three independent matrices of the form $height \times width \times 1$. Then, we separate each kernel depth layer and treat them as if they were three separate kernels of the form 3x3x1. When we have the three input channel matrices and the three distinct kernels, the scalar multiplications are performed on them. Specifically, each kernel is iterated by one of the three channels so, in total, we will have three kernel-matrix pairs to perform the operations. Specifically, all the channels of the input image will be filtered by one of the kernel depth layers. At the end of this layer, we will obtain 3 completely differentiated feature maps. Figure 3.1 is a representation of how this process works.

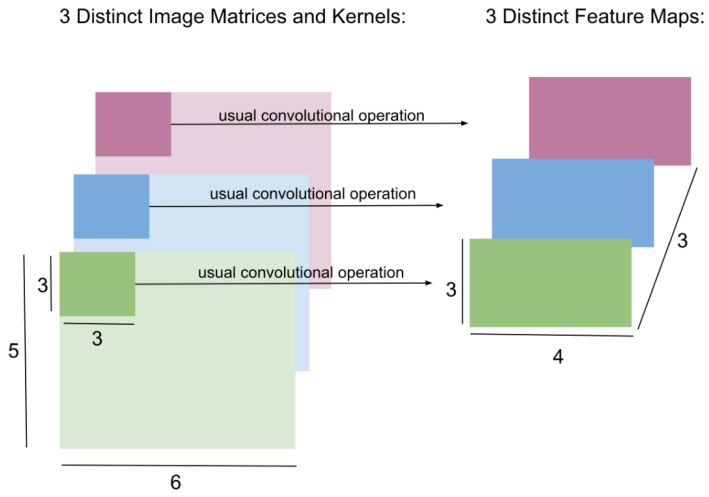


Figure 3.1: Depthwise Convolution

The "*usual convolutional operation*" which is written above the arrows in Figure 3.1 refers to the multiplications and additions made between the kernel weights and the values of the input image matrix to obtain each feature map, obtained as in Figure 2.6. We can see that there is no interaction between the different layers of the image, we simply do three separate convolution operations and obtain the three distinguished feature maps. If we put the three feature maps together, we will obtain an output tensor of size 3x4x3, so we go from a 5x6x3 matrix to a 3x4x3 one.

- **Pointwise Convolution**

  Since we want to get the same shape of the output tensor as we got in normal convolution operations, in Figure 2.6 i.e. 3x4x1, we must apply this layer, as it deals with the depth operations. What is achieved in this layer is that, by combining the three feature maps, we obtain new features that can be useful for the model to learn. Specifically, this layer uses a kernel of size $1 \times 1 \times DEPTH$ (in our case, as we said before, the DEPTH of the kernel will be equal to 3). The kernel will have to iterate through the $height \times width$ of the feature maps obtained in the DW CONV layer in order to traverse all of their pixels and, perform the convolution operation to combine them in each iteration. The spatial dimensions of the kernel is what gives the particular name "pointwise" to this layer, since being 1x1 it is as if it were a point.

  The process is represented in the following Figure 3.2. And, in Figure 3.3 we show an example of the convolution operation performed in each iteration to extract each value of the final feature map.

In addition, in both Conv and DSC, we can use more than one filter, and thus obtain a result with more than one channel. We will obtain a final Feature Map with as many

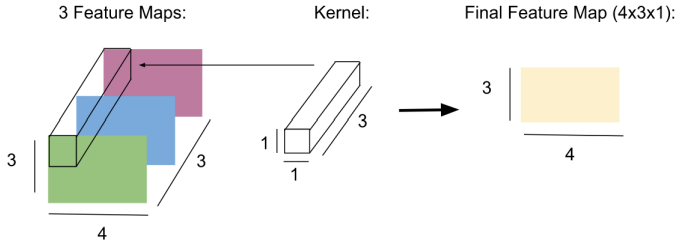


Figure 3.2: Pointwise Convolution

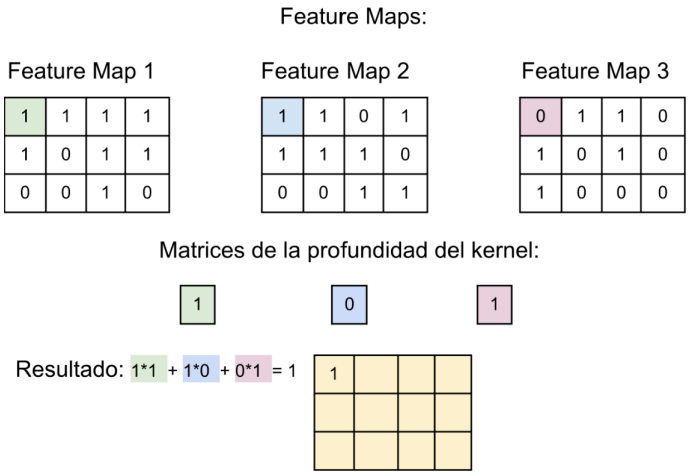


Figure 3.3: Operation of Pointwise Convolution to extract 1 value of the Final Feature Map

channels as kernels as we apply. Each filter passes through the input image separately and, the result of applying each filter will become a channel of the output tensor. So, if we apply 256 kernels to the input image, we will obtain a result like in Figure 3.4.

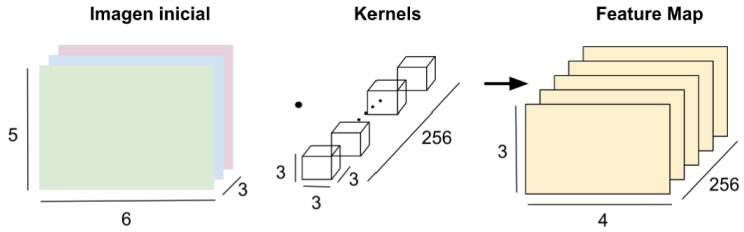


Figure 3.4: Feature Map obtained after applying 256 kernels/filters to the Input Image

As can be seen in all the previous explanation, we have obtained the same result with the Conv and with the DSC (an output feature map of 3x4x1 with the same information) but, the great advantage of using the second one, is the large amount of multiplications that we save for the benefit of the computational cost. In the following, we will see a mathematical comparison based on [HZC+17] of the amount of multiplications we have to complete with each kind of layer and, this way, prove how using DSC decreases the number of multiplications to perform.

The Conv has as input $\mathbf{F}$ which is a tensor of size equal to $D_F \times D_F \times M$ and,

after performing the convolution operation, it obtains a feature map **G** of size equal to $D_F \times D_F \times N$. In both cases D_F is used as the height and the width of the input image and feature map matrices. It is assumed that both have the same spatial dimensions and that they are square matrices (same height and width) but these calculations can be applied to any input and output size. **M** is the number of channels of the input image and **N** the number of channels of the output image. Then, we have the kernel, which has size $D_K \times D_K \times M \times N$ where, D_K are the spatial dimensions of the kernel (in this case it is square, since height and width are equal) and, M and N, are the measures that we have explained above (number of channels of the input and number of channels of the output).

The cost of this type of layers is equal to Equation 3.1. That is, it depends on the size of the kernel, the number of input and output channels and, the spatial size of the input image. The cost of $D_K \times D_K$ is due to the number of multiplications made between the kernel weights and the values of the input matrix and, it is multiplied by M ($D_K \cdot D_K \cdot M$) because there can be more than 1 channel and, if there are, then the operations performed by the kernel will have to be done as many times as the number of depth layers in the image. Then it is multiplied by N ($D_K \cdot D_K \cdot M \cdot N$) because, if we want the output layer to have a depth grater than 1, we must apply to the input image as many kernels (and therefore, convolution operations) as the number of channels we want the feature map to have. And, finally, it is multiplied by the size of the input image $D_F \times D_F$ ($D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$) because the kernel iterates through the whole matrix and makes multiplications between all its weights and the values of the matrix in each iteration.

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \tag{3.1}$$

The DW CONV, which has one filter per input channel of the image, can be expressed as follows

$$\hat{\mathbf{G}}_{k,l,m} = \sum_{i,j} \hat{\mathbf{K}}_{i,j,m} \cdot \mathbf{F}_{k+i-1,l+j-1,m} \tag{3.2}$$

K is the DW CONV's kernel whose size is $D_K \times D_K \times M \times 1$. The m^{th} filter of the kernel is applied to the m^{th} channel of the input image. The expression is multiplied by 1 ($D_K \cdot D_K \cdot M \cdot 1$) and not by N because the feature maps that will be obtained will all be of depth 1, so this 1 can be omitted. In our case M=3 since we will use color images containing all 3 RGB channels.

The DW CONV has a computational cost equal to Equation 3.3, which is the same as the Conv except that it is not multiplied by N. This is because there is no interaction in this layer between the N feature maps, so we do not have to add those N operations that

existed in the Conv.

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F \tag{3.3}$$

The PW CONV performs the linear combination of the feature maps obtained in the DW CONV, using a kernel of size 1x1xDEPTH, so its computational cost is equal to Equation 3.4. M, N, and D_F mean the same as in the convolution layer and these values are multiplied because, the kernel traverses all the feature maps (M) and its spatial dimensions ($D_F \times D_F$) and finally the N is for the amount of kernels we will have to reach the number of output channels we want the output tensor to have.

$$M \cdot N \cdot D_F \cdot D_F \tag{3.4}$$

Taking all this into account, the total computational cost of the DSC is equal to Equation 3.5, which is the sum of the computational cost of the DW CONV and PW CONV. It is a sum and not a multiplication because they are independent layers that do not cross or interact between them.

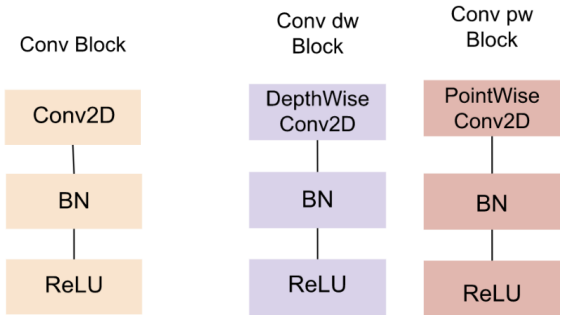
$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \tag{3.5}$$

The computational cost reduction achieved by using DSC instead of Conv is equal to Equation 3.6. So we can confirm that DSC has much less mathematical operations to perform than Conv.

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} \tag{3.6}$$

The MobileNetV1 architecture used in our project is the one that Kera's Tensorflow has in tf.keras.applications.mobilenet.MobileNet() function, which is developed in Table 3.1. The model starts with a Conv and the rest is mostly composed by DSC blocks. This layers are always followed by one BatchNormalization layer and another activation layer, specifically, ReLU. So, the convolution blocks that exist in the model are the following:

These two additional layers (BN and ReLU) are applied for the following reasons. Normalizing data, in general, means modifying their values so that all of them are in a certain range and, thus, do not increase the bias of the model. The batch normalization layer is applied just after having performed the convolution operations since after having implemented them, the values of the input matrices may have varied a lot and, with them, their distribution. If we did not apply batch normalization after the data has been modified, the distribution of the input of each layer of the model would be very different.



(a) Standard Convolution (b) Depthwise Separable Convolution

Figure 3.5: Standard Convolution Block and Depthwise Separable Convolution Block [HZC+17]

This phenomenon is called covariate shift and causes the model to have difficulties in learning. Applying BN reduces this problem [LC15] and makes the model converge faster. So, according to the good results obtained in other tests [IS15], BN is applied just before ReLU function to obtain the best results.

Then, ReLU, which stands for Rectified Linear Unit, is an activation function that deactivates a neuron if the value is less than 0. So, this function can be represented mathematically as $f(x) = max(0, x)$. Specifically, in images, which is what we are going to train this model for, what it does is to change all the cells with negative value (of the normalized feature maps) to 0. That is, it will put those pixels in black because, the value 0 in a pixel of an image represents the color black. One of its major advantages is that, by not activating all the neurons, the computational time is shorter and therefore the training of the model is faster.

As we can see in Table 3.1, the first Conv and, in some DW CONV, the stride applied is equal to 2, and it is basically to achieve downsampling. Downsampling an image serves to reduce its spatial dimensions, $height \times width$, this helps the model training process to be faster, since smaller input images are obtained in the intermediate layers of the model and, therefore, any mathematical operation to be performed will be much faster.

Some of the last layers that are not part of the blocks, are added for good reasons and we will explain them in the following lines. One of the disadvantages of feature maps is that they store the position of important features at the exact location where they are in the input image. The problem with this is that, at the slightest change in an image, for example a flip of any degree, going into mirror mode or any kind of rotation, would produce a completely different feature map, because those features would not be in the same position as the "original" image. The downsampling that is applied in MobileNetV1 in some of the convolution layers, using the stride equal to 2 helps these feature maps not to have the highlight features in the exact places of the input image because they reduce

Table 3.1: Body Architecture of MobileNetV1 [HZC+17]

Layer	Stride	Filter Shape	Input Image Size	Output Image Size
Conv Block	2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$	$\mathbf{112 \times 112 \times 32}$
Conv Block dw	1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$	$112 \times 112 \times 32$
Conv Block pw	1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$	$112 \times 112 \times \mathbf{64}$
ZeroPadding2D	No Stride	No Filter	$112 \times 112 \times 64$	$\mathbf{113 \times 113} \times 64$
Conv Block dw	2	$3 \times 3 \times 64$ dw	$113 \times 113 \times 64$	$\mathbf{56 \times 56} \times 64$
Conv Block pw	1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$	$56 \times 56 \times \mathbf{128}$
Conv Block dw	1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$	$56 \times 56 \times 128$
Conv Block pw	1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$	$56 \times 56 \times 128$
ZeroPadding2D	No Stride	No Filter	$56 \times 56 \times 128$	$\mathbf{57 \times 57} \times 128$
Conv Block dw	2	$3 \times 3 \times 128$ dw	$57 \times 57 \times 128$	$\mathbf{28 \times 28} \times 128$
Conv Block pw	1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$	$28 \times 28 \times \mathbf{256}$
Conv Block dw	1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$	$28 \times 28 \times 256$
Conv Block pw	1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$	$28 \times 28 \times 256$
ZeroPadding2D	No Stride	No Filter	$28 \times 28 \times 256$	$\mathbf{29 \times 29} \times 256$
Conv Block dw	2	$3 \times 3 \times 256$ dw	$29 \times 29 \times 256$	$\mathbf{14 \times 14} \times 256$
Conv Block pw	1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$	$14 \times 14 \times \mathbf{512}$
5x Conv Block dw	1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$	$14 \times 14 \times 512$
Conv Block pw	1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$	$14 \times 14 \times 512$
ZeroPadding2D	No Stride	No Filter	$14 \times 14 \times 512$	$\mathbf{15 \times 15} \times 512$
Conv Block dw	2	$3 \times 3 \times 512$ dw	$15 \times 15 \times 512$	$\mathbf{7 \times 7} \times 512$
Conv Block pw	1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$	$7 \times 7 \times \mathbf{1024}$
Conv Block dw	1	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$	$7 \times 7 \times 1024$
Conv Block pw	1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$	$7 \times 7 \times 1024$
GlobalAveragePooling2D	1	Pool 7×7	$7 \times 7 \times 1024$	$\mathbf{1 \times 1} \times 1024$
Reshape	No Stride	No Filter	$1 \times 1 \times 1024$	**1024**
Dropout	No Stride	No Filter	1024	1024
Dense	No Stride	No Filter	1024	**2**

the size of the output matrix and, therefore, it is more complicated for the pixels to be exactly in the same place as in the initial image. But, to reduce this problem usually pooling layers are also used. Since this layer will reduce the size of the feature maps, it is applied after having done the convolution and the non-linear (ReLU) operations (these operations have to be done with all the values of the feature maps). This process averages those cells occupied by the chosen kernel in the feature map. In order to obtain a Global Average, the averaging will be done only once with all the values of the feature map so, a kernel occupying the whole image will be needed. Given that the GlobalAveragePooling2D layer has an input of 7x7x1024, a kernel of size 7x7 will be used. So, the output of this average operation will be 1x1x1024.

The Reshape layer is used to adjust an input's dimensions with the advantage of not altering its actual information. In the instance of MobileNetV1, a 1-dimensional input of size 1024 is required for the Dropout layer hence, the Reshape function simply removes

the 1x1 of the 1x1x1024 input. In this case, eliminating the $height \times width$ does not affect the content of the input because, a size of 1x1x1024 is equal to a size of 1024. Finally, in terms of content, we will get an output equal to the input.

When we train models with many layers with small datasets it is likely that the model will over learn. This is because it will reach a point where it will start to learn too specific details of the images and it will be much more complicated to generalize. A solution to this problem is to use a Dropout layer, this layer has a simple but effective operation. According to [SHK+14] it consists of randomly dropping out some of the output neurons and leaving the others untouched. What is achieved with this is to temporarily eliminate the input and output connections of the layer and, with this, it is possible to eliminate the over-fitting that is caused by trying to solve all the calculation errors of the previous layers. In addition, the model during each training epoch becomes smaller in terms of functional units. So, applying dropout on a model would be like training small variations of that model. If we define that we have n units in a CNN, we can say that there are 2^n possible combinations of units nulled or not and therefore 2^n possible reduced models. To make the test predictions one would have to combine the predictions of the reducted models, average them and get a result, but this would be very computationally expensive. The solution proposed in [SHK+14] is to use a single model to do the test. This model will have all the active units, but each of them will be multiplied by a probability p (the same probability with which the unit was removed from the model at the time of training). Thus, we can guarantee that for the hidden units we will get the same prediction in the training as in the test. And finally, they also check and confirm that performing this technique minimizes the generalization error.

After applying a convolution operation to an input, we obtain an output with different dimensions, specifically, if we have an input of spatial size $n \times n$ and a filter of size $f \times f$, the output will have size $(n-f+1) \times (n-f+1)$. The problem is that, when reducing the size of the image, if there are important features at the edges it is possible that they are lost when applying convolution operations. This happens because the filter does not get to be applied completely at the edges (since it goes down to the next row as soon as the size of the input is exceeded by the filter). Having Zero Padding consists of applying cells around all the edges of the image with a value equal to 0 (so, the result of the convolution operation will not change) and thus, the filter would reach all the edges of the input performing the convolution operations over all of them and, this way, managing not to lose any information.

Finally we have the dense layer, the differentiating factor of this layer is that all the neurons are connected with all the neurons of the previous layer in order to get all the information collected throughout the training of the architecture, which is why it is usually added at the end of the deep learning models.

3.2.2 MobileNetV2

MobileNetV2 also uses DSC, instead of Conv layers, with the same size of kernels as MobileNetV1; *height* × *width* equal to 3x3 so, the computational cost with respect to Conv operations is also 8 to 9 times lower [HZC+17]. What is incorporated to what already existed is a new block called Inverted Residual with Linear Bottlenecks.

3.2.2.1 Linear Bottlenecks

Linear Bottleneck is a convolution layer that manages to reduce the number of input feature maps (which are the input channels) by using fewer filters than in other layers (because the number of output feature maps depend on the number of filters used). The objective of reducing the number of feature maps, in the case of MobileNetV2, is to group in less space the most relevant features of the images. The Linear Bottleneck uses kernels of size 1X1Xdepth and, as in PW CONV, using those kind of kernels allow the model to join all the values from the input feature maps into a single matrix. Linear Bottleneck learns where the important features are in the input tensor so, they use as many filters as they need to keep this information in the output. The most important information is usually a small portion of the higher dimensional activation, which is called Manifold of Interest, so this layer will only need a few filters to keep that Manifold of Interest in a lower dimensional activation. Figure 3.6 is a representation of what would happen with an input tensor of 32 channels going through a Bottleneck.

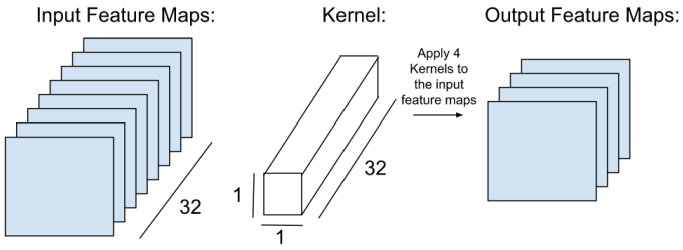


Figure 3.6: Bottleneck Layer operation

As we mention above, not all the features of an image are important for the learning of the model so, we could think that it would be a much more convenient approach to reduce the input to a low dimensional activation. This way the model would only be trained with the features it is most interested in learning and, as an extra advantage, the number of parameters would be reduced. The problem is that we need to add non-linear operations to the CNN models and, if MobileNetV2 is training with low dimensional activations, when applying the ReLU operation there would be lot of information lost from the important features. This loss of information is due to the fact that the ReLU function, as we have explained above, is of the form f(x) = max(0,x), then, any information that exists on the image with a negative value would be lost, in fact, there could end up existing channels

with all values at 0 which would cause the model to lose the complete information of that channel. The good thing about having higher dimensional activation is that, by having more channels, if for some reason important features belonging to the MoI are removed, this information can be recovered with the rest of its channels (since, especially in images, the values of the matrices are related to each other and, if any of those values are lost, with the rest of the channels this important information could be recovered). In summary, if we have for example a higher dimensional activation with 256 channels, the MoI is concentrated in 3 of those channels and, the model looses information of those 3 channels, there will be another 253 channels with which to recover the lost information. Therefore, the MoI data will be maintained even after applying ReLU if the model works with higher dimensional activations.

This process can also be done in reverse, i.e. instead of reducing the number of output channels, you could increase them. The same shape of kernels would be applied and, in order to get higher dimensional activations, more filters than the input channels would be used. Architects of MobileNetV2 thought it would be a good idea to apply it right before the convolution operations and the non-linear operations to increase the number of channels and, thus, reduce the possibility of losing any of the Manifold of Interest information. This action is called Expansion Convolutional Layer.

Taking into account all this explanation, it makes sense to insert linear bottleneck in the body of the model also at the end of the architecture. With this mechanism, we could reduce the size of the higher dimensional activation and then, obtain as an output at the end of the training process a lower dimensional activation (containing only the Manifold of Interest). In Figure 3.7 we can see how applying an Expansion Convolutional Layer at the beginning and a Linear Bottleneck at the end of some layers would look like.

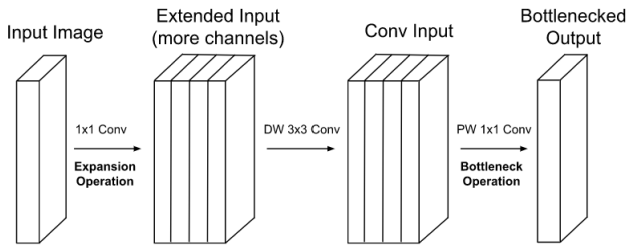
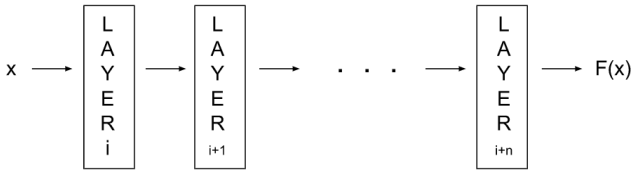


Figure 3.7: Expansion Layer and Bottleneck Layer [SHZ+18]

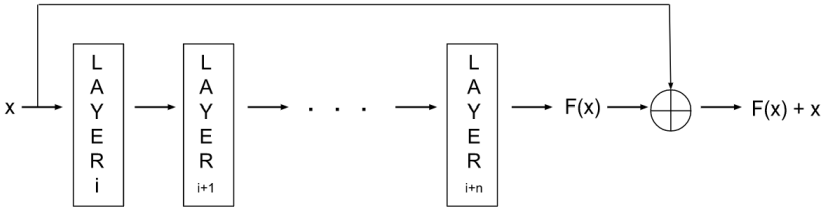
3.2.2.2 Inverted Residuals

Normally, in the training of a deep learning model, the information travels from layer to layer from beginning to end without skipping any of them. By applying what is called "Residual Connections", this information can arrive from the model input to the output through different paths. What these new connections allow is that the information does

not have to go though absolutely all the layers, since it will have the possibility of skipping some of them. In Figure 3.8 we show a simple comparison between the flow of information with and without residual connection.



(a) Information Flow Chart without Residual Connection



(b) Information Flow Chart with Residual Connection

Figure 3.8: Comparison between the two possible Information Flow Charts [HZRS16]

We can observe in the first flow chart of Figure 3.8 how the input x goes through all the layers, from i to i+n, in order to reach the output. At the end, F(x) is obtained, which is the function that represents the operations performed in all the intermediate layers. In the second flow chart of Figure 3.8 we can see that a residual connection is applied from the input to the output, thus, achieving to have at the end the identical information of x. It can be seen that the intact information of x is summed with F(x) and, that is why, residual connection can only be applied if the shape of the input x is equal to the shape of the output tensor of layer i+n.

One of the major advantages of the residual connections is that when the backpropagation is done (to take the gradient information from the output to the input of the model) it is not necessary to go through all the layers of the model and, therefore, it is much faster. This also helps to reduce the probability of this information to be lost. In addition, as we have explained above, when non-linear functions are applied to the algorithm, in this case the ReLU, a lot of information of the input image is lost along the layers so, by directly connecting the input image to the output, we are ensuring that almost no important (or any) feature of the image is lost.

It is called *Inverted* Residual connections because the normal approach is to connect two separate layers with higher dimensional activations (inputs with more channels) instead of connecting the layers with the lowest dimensional activations (as it is done in MobileNetV2). There is a comparison of both ways of performing residual connections in Figure 3.9.

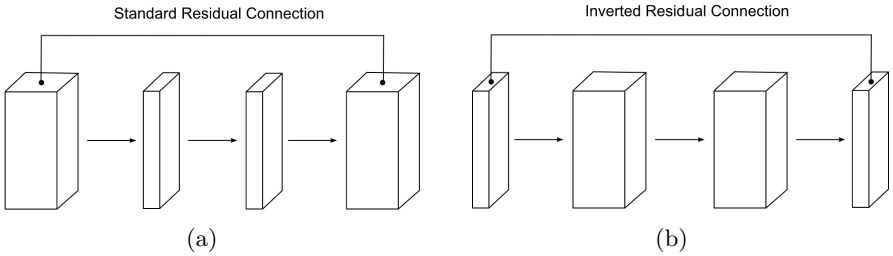
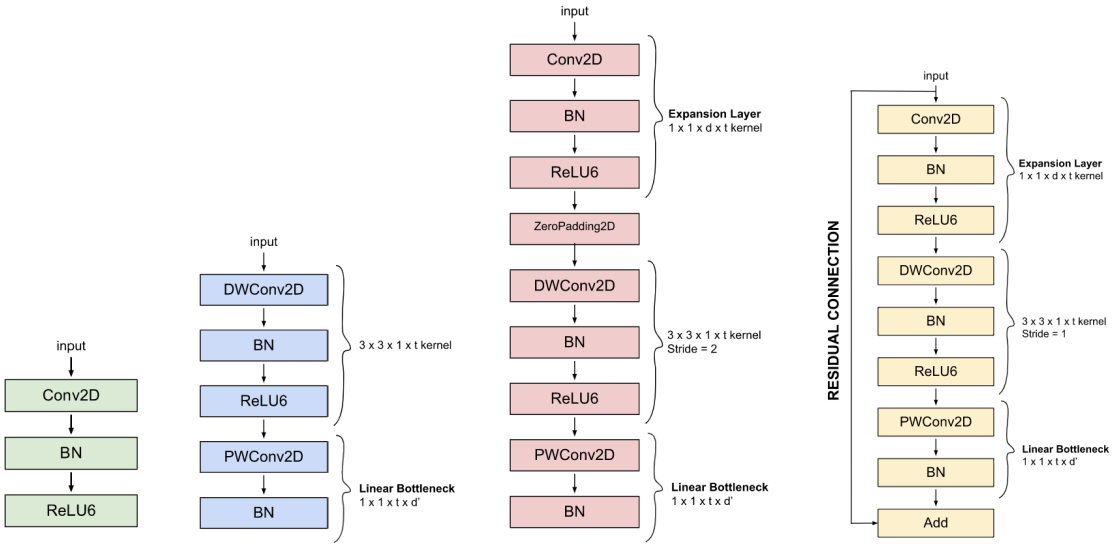


Figure 3.9: Standard vs Inverted Residual Connections

In the case of MobileNetV2 these residual connections are made throughout the model, i.e., these connections are applied several times between intermediate layers. In Figure 3.10 we have a representation of the blocks that conform the model and we can see how residual connections is applied in 3.10(d).



(a) Conv2D (b) Block 1 (c) Block 2 (d) Block 3

Figure 3.10: Some of the principal blocks that conform the body of MobileNetV2

The shape of the input of the Conv (Conv2D) layer of each block is $h \times w \times d$ (height x width x depth), the kernel used in this layer has shape $1 \times 1 \times d$ (where d is the number of input channels) and t is the number of filters used. We call it expansion layer because here is where the number of channels is expanded just before the non-linear operations (ReLU6) take place. As we are expanding the number of channels, we will apply a lot of filters, so t will have a high value.

The input shape of the DW CONV (DWConv2D) layer of each block is $h \times w \times t$ (height x width x depth=t) and the kernel we use in this layer is 3x3x1 (as in MobileNetV1). As we explained earlier in MobileNetV1, this layer does not minimize the number of input channels but it can change the spatial dimensions of it. In the case of Block 2, the DW CONV layer halves the $height \times width$ size of the input images using stride equal to 2.

And, the ZeroPadding2D layer is added before going through the DWConv2D so that the convolution operation can be done (converts the value of height and width to odd so that the stride equal to 2 does not give any problem).

Finally, we have the 1x1 convolution layer, which is used to perform the Bottleneck. It uses a *height* × *width* kernel equal to 1x1 and with t number of channels (1x1xt). As we are talking about the Bottleneck layer, it is time to reduce the input channels so, the factor *d'*, which is the number of output channels, will be much smaller than the input channels (t). If we want to apply a residual connection, as in Block 3, the shape of the output has to be equal to the input of the block to be able to do the final addition, so d' will be equal to d. As we expected, after the bottleneck the non-linear ReLU6 operation is not applied. In Block 3, what it is applied after the Bottleneck is the Add layer which allows doing the addition of F(X) (output of the BN layer) + x (identity of the input).

We will now show an example of Block 2 and Block 3 that make up the body of the architecture taken directly from the *summary()* function of *tf.keras.applications.mobilenet_v2.MobileNetV2()* to verify that the operations explained above are actually being performed. First we must indicate that the second column of Figures 3.11 and 3.12 is the output shape of the layer written at the left column. We also have to say that the first block we are going to talk about is called "block_6" because it is the sixth block of the whole body of MobileNetV2 but has the architecture of Block 2 in figure 3.10 and, the second block we are going to talk about is called "block_7" but has the architecture of Block 3 in figure 3.10.

```
block_5_add (Add)              (None, 28, 28, 32)   0           ['block_4_add[0][0]',
                                                                 'block_5_project_BN[0][0]']

block_6_expand (Conv2D)        (None, 28, 28, 192)  6144        ['block_5_add[0][0]']

block_6_expand_BN (BatchNormal (None, 28, 28, 192)  768         ['block_6_expand[0][0]']
ization)

block_6_expand_relu (ReLU)     (None, 28, 28, 192)  0           ['block_6_expand_BN[0][0]']

block_6_pad (ZeroPadding2D)    (None, 29, 29, 192)  0           ['block_6_expand_relu[0][0]']

block_6_depthwise (DepthwiseCo (None, 14, 14, 192)  1728        ['block_6_pad[0][0]']
nv2D)

block_6_depthwise_BN (BatchNor (None, 14, 14, 192)  768         ['block_6_depthwise[0][0]']
malization)

block_6_depthwise_relu (ReLU)  (None, 14, 14, 192)  0           ['block_6_depthwise_BN[0][0]']

block_6_project (Conv2D)       (None, 14, 14, 64)   12288       ['block_6_depthwise_relu[0][0]']

block_6_project_BN (BatchNorma (None, 14, 14, 64)   256         ['block_6_project[0][0]']
lization)
```

Figure 3.11: Screenshot of an example of Block 2 from the summary of MobileNetV2

We can observe in Figure 3.11 that the input (which is underlined in yellow) to Block 2 has size 28x28x32 and after going through the Conv2D layer (expansion layer), the output has shape 28x28x192 (the number of channels have been increased, as we can see in the red underlined number, and a higher dimensional activation has been obtained). With this higher dimensional activation the non-linear operation ReLU can be applied and, in fact,

it is. ZeroPadding2D is applied before the tensor entering the DepthwiseConv2D layer in order to allow the convolution operation to be performed with stride equal to 2, this way the model manages to reduce the spatial dimensions and achieve the downsampling. We can see this downsampling in the blue underlined numbers, the tensor changes from 28x28x192 to 14x14x192. And, in the last Conv2D, the Bottleneck is performed by reducing the number of output channels from 192 to 64 (which is underlined in green). Because the input channels are 32 and the output channels are 64 i.e., they are not equal, the addition layer can not be added so there is no residual connection in this Block 2 (as we expected from Figure 3.10). We can check how after the Bottleneck operation (final Conv2D) ReLU is not applied. Everything that we explained previously for this Block 2 is fulfilled.

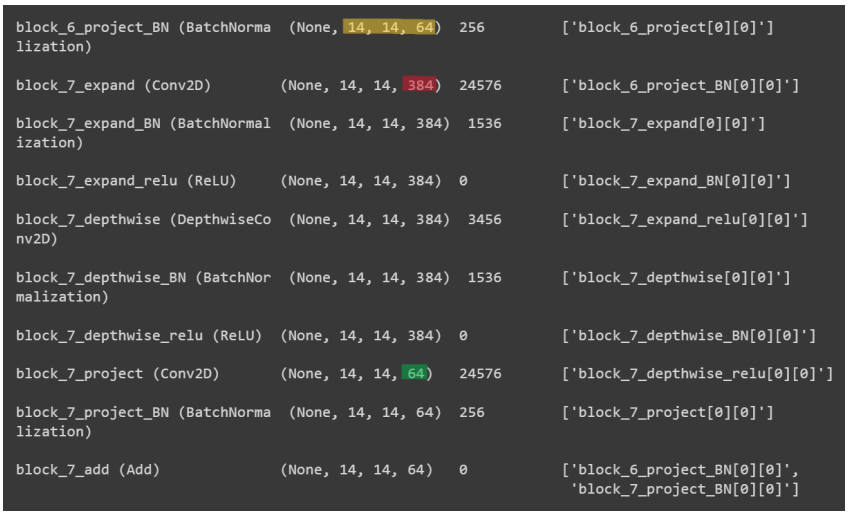


Figure 3.12: Screenshot of an example of Block 3 from the summary of MobileNetV2

In Figure 3.12 it can be seen that the input to Block 3 has size 14x14x64 (underlined in yellow) and, after this tensor going through the Conv2D layer (expansion layer), the output has size 14x14x384, which means that the expansion layer has acted as it should increasing the number of output channels (to obtain the higher dimensional activation). As in Block 2, with this higher dimensional activation the model can apply the non-linear ReLU function to it. When performing DeptwiseConv2D any dimension changes (since Block 3 has stride equal to 1). Going through the last Conv2D, the Bottleneck is performed reducing the number of output channels and establishing them equal to the number of input channels, this way it is allowed to include the Add layer for the residual connection. We can see again how the ReLU non-linear function is not applied after the Bottleneck layer in order to not loose too much information. Once more, everything that we explained previously for Block 3 is satisfied.

The final body of the MobileNetV2 architecture is represented on Table 3.2.

Table 3.2: Body Architecture of MobileNetV2 [SHZ+18]

Layer	Input Image Size	Output Image Size	Stride
Conv2D	224x224x3	112x112x32	2
Block 1	112x112x32	112x112x16	1
Block 2	112x112x16	56x56x24	2
Block 3	56x56x24	56x56x24	1
Block 2	56x56x24	28x28x32	2
Block 3	28x28x32	28x28x32	1
Block 3	28x28x32	28x28x32	1
Block 2	28x28x32	14x14x64	2
Block 3	14x14x64	14x14x64	1
Block 3	14x14x64	14x14x64	1
Block 3	14x14x64	14x14x64	1
Conv2D + Block 1	14x14x64	14x14x96	1
Block 3	14x14x96	14x14x96	1
Block 3	14x14x96	14x14x96	1
Block 2	14x14x96	7x7x160	2
Block 3	7x7x160	7x7x160	1
Block 3	7x7x160	7x7x160	1
Conv2D + Block 1	7x7x160	7x7x320	1
Conv2D 1x1	7x7x320	7x7x1280	1
GlobalAverage Pooling2D	7x7x1280	1280	-
Dropout	1280	1280	-
Dense	1280	2	-

3.2.3 MobileNetV3

The MobileNetV3 architecture combines the hardware network architecture search, also known as NAS, with the NetAdapt algorithm. New enhancements developed by the MobileNetV3 architects are then added to this algorithm. The objective of this new implementation is to improve the trade-off between the accuracy and latency of the model within cell phones in order to achieve the best results. They implement blocks that have a combination of the MobileNetV2 blocks with Squeeze-and-Excitation. They built two versions of MobileNetV3, the MobileNetV3-Small version and the MobileNetV3-Large version, with the same kind of building Blocks. So, we will start explaining what Squeeze-and-Excitation Block is and how it works.

3.2.3.1 Squeeze-and-Excitation

One of the main objectives of these blocks is to minimize the size of the representation of an image and obtain the essential features in that reduced space. In this way, the model would save space and time (since it would focus on the most important data and would not

waste time learning unnecessary data). To do this, they exploit the relationship between the image channels, i.e., instead of treating them as independent channels, what they do is to establish a relationship between them. This relationship is based on assigning a percentage of importance to each input channel. As we explained before, each of these channels represents a specific characteristic of the image (such as edges, shadows,...) and, as these features are related to each other, the percentage of importance of a channel will have a direct relationship with the rest. According to the percentage of importance of each channel, the model will know which is the most relevant data. The exact procedure followed by this block, based on [HSS18], is showed in Figure 3.13.

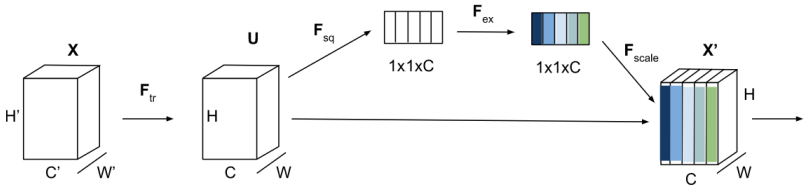


Figure 3.13: Squeeze-and-Excite Block [HSS18]

Say that a transformation function $\mathbf{F}_{tr}$ is applied to the tensor $\mathbf{X}$, in the case of MobileNetV3 that transformation will be convolution operations. After those operations, we will obtain the set of feature maps $\mathbf{U}$ to which we want to pass through the Squeeze-and-Excite block, just before the output is passed to the projection layer. The Squeeze-and-Excitation process consists of three steps based on Figure 3.13.

- *Squeeze*: $\mathbf{F}_{sq}$, as the name suggests, the spatial dimensions of U are reduced using Global Average Pooling. At the end we will have an output of size 1x1xC since, by averaging all the values of each feature map of U, we reduce those feature maps to a single value and we will no longer have spatial dimensions (we will only have one value per feature map/channel). Then each value of the output vector would be calculated as Equation 3.7.

$$z_c = F_{sq}(u_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} u_c(i,j) \tag{3.7}$$

  Where u_c is the c^{th} channel of input U, that is, the c^{th} feature map of input U. And the variables H and W are the height and width of the feature maps (all the feature maps of U have the same width and height). To make the global average is as simple as going through all the cells of the c^{th} feature map, sum them (that is why there are two summations, because we have to go through a 2 dimensional matrix) and finally, divide them by the total number of cells (HxW). By having only one vector of values as output, where each value represents a whole feature map, it will be easier to assign a percentage to each feature.

- *Excite*: $\mathbf{F}_{ex}$, this is a sub block which, in the case of MobileNetV3 implemented in TensorFlow, has the following form: $Conv2D \rightarrow ReLU \rightarrow Conv2D \rightarrow HardSigmoid$. This mini block is where the model can learn what importance each output channel of the Squeeze layer has in relation to the rest of the channels. Once the importance is learned, the nonlinear hard sigmoid function is applied to set a percentage of importance to each cell of the input vector, this function will return a value between 0 and 1 (in [HSS18] they use normal sigmoid Function but in [HSC+19] they note that it can be inefficient and so they use hard sigmoid Function 3.11 instead). The Equation of this step is 3.8.

$$s = F_{ex}(z, W) = \sigma(g(z, W)) = \sigma(W_2\delta(W_1 z)) \tag{3.8}$$

  W_1 is a reduction convolution operation. The input to this convolution operation is z, which is the squeeze´s output with size 1x1xC so, if the output of this layer has size 1x1xC', as it is a reductional operation, we will be sure that $C' < C$. Delta (δ) is a non-linear function, in the case of any of the two versions of MobileNetV3 it will be ReLU. W_2 is an expansion convolution operation, i.e., the tensor goes from 1x1xC' to 1x1xC again. And, finally, the sigmoid function is applied but, in the case of both versions of MobileNetV3, instead of standard sigmoid it will be hard sigmoid.

- *Scale*: $\mathbf{F}_{scale}$, the output of the block is rescaled, that is, now the output feature maps will have spatial dimensions HxW different from 1x1. And the scaling operation is, for the sake of redundancy, a scalar multiplication (that is why in MobileNetV3-Small and in MobileV3-Large the last layer of the Squeeze-and-Excitation Block is Multiply), in Equation 3.9 we show how to calculate each of these matrices, based again on [HSS18].

$$\tilde{x}_c = F_{scale}(u_c, s_c) = s_c u_c \tag{3.9}$$

  Where x_c is the c^{th} feature map of the output tensor, u_c is a feature map of the tensor before entering the Squeeze-and-Excite block and s_c is one value of the percentages vector that we have extracted in the Excite layer. By multiplying all the values of the u_c matriz by the scalar s_c we get a completely re scaled feature map.

Considering that the MobileNetV3 architectures are a fusion between the principal MobileNetV2 block (Inverted Residuals with Linear Bottlenecks) and the Squeeze-and-Excitation block, the following blocks for MobileNetV3-Small and Large are obtained (Figures 3.14 and 3.15). This is the TensorFlow implementation which is based on [Koo21].

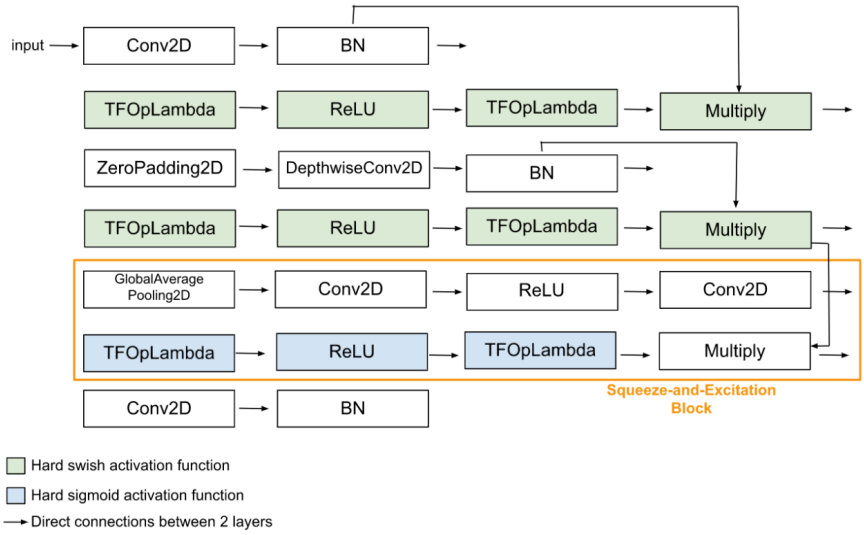


Figure 3.14: Block 1: first possible Block of MobileNetV3-Small and MobileNetV3-Large

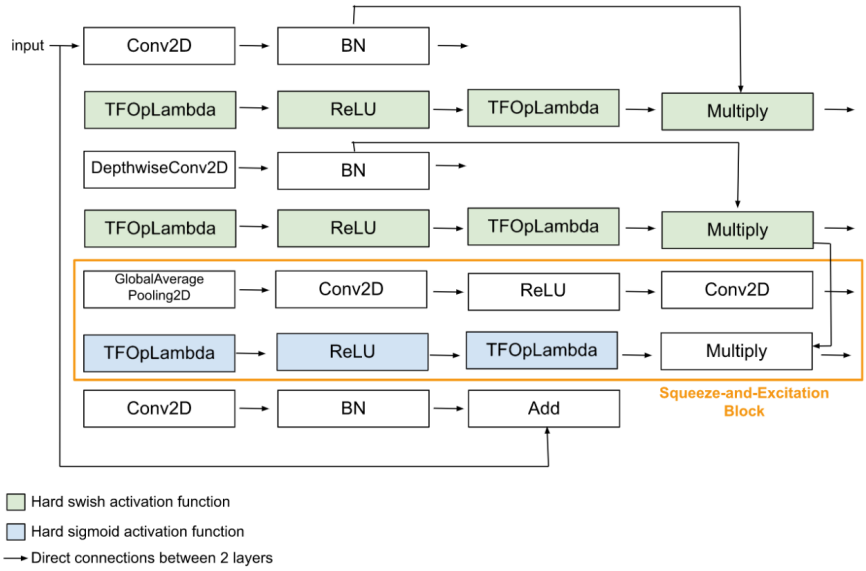


Figure 3.15: Block 2: second possible Block of MobileNetV3

Similar to MobileNetV2, there are the two possible block options regarding residual connections, the one that does not have it (Figure 3.14) and the one that has it (Figure 3.15). As in MobileNetV2, the residual connection and the Add layer are incorporated if and only if the two entries of the Add layer have the same shape. We can observe in these blocks that after the BN layer, instead of the activation function ReLU, the model is using h-swish[x] whose Equation is 3.10. This equation was introduced by [HSC+19], which is a modified version of the standard non-linear swish function.

$$h\text{-}swish[x] = x\frac{ReLU(x+3)}{6} \tag{3.10}$$

And we can also observe that instead of using the normal sigmoid function we are using the hard sigmoid whose Equation is 3.11.

$$h\text{-}sigmoid[x] = \frac{ReLU(x+3)}{6} \tag{3.11}$$

That is, the function h-swish[x] can also be expressed as $h\text{-}swish = x * h\text{-}sigmoid[x]$, these functions have been purchased from [Koo21].

3.2.3.2 Hardware Network Architecture Search

Hardware Network Architecture Search (NAS) is used to improve the properties of the model once it has been built, in a faster and more effective way than if we were to do it manually. Its main objective is to achieve a good balance between model accuracy and latency. The main scheme that follows this technique is shown in Figure 3.16 taken from [TCP+19].

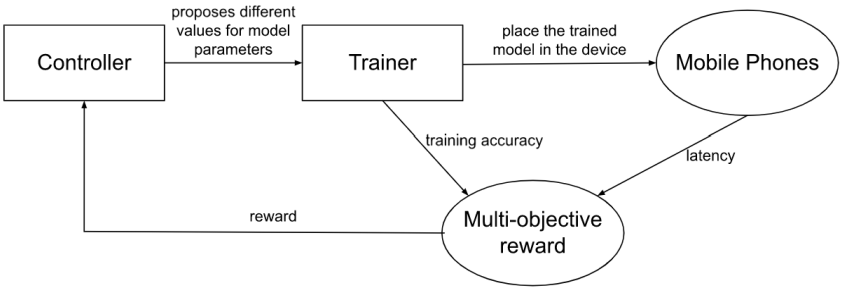


Figure 3.16: Platform-Aware Neural Architecture Search for Mobile [TCP+19]

The *Controller* offers the *Trainer* any of the two possible MobileNetV3 architectures with certain parameters so that it can train it (each time this process is repeated, it will be done with different characteristics of the model). When trained, the model can then be imported to a cell phone and test how it works on it. Once trained and tested, we can get the accuracy and latency we want to optimize. The step of getting a better balance between accuracy and latency is tricky because when one increases the other increases as well, and, as usual, we are interested in having the lowest latency and the highest accuracy possible. So, the perfect balance between both must be found. To do this, we will use the formula 3.12 which is the one used to obtain the reward between accuracy and latency.

$$maximize_m \;\; ACC(m) \times \left[\frac{LAT(m)}{T}\right]^{\omega} \tag{3.12}$$

Where m is the model, ACC(m) is the accuracy of the model, normally in the training a validation dataset is used and the accuracy obtained is calculated with it. LAT(m) is the latency of the model when used on a cell phone and, T, is the maximum latency expected

(i.e., the model cannot exceed this latency because otherwise it would mean that the model is too slow and does not meet the desired properties). Finally, omega(ω) is the weight factor, i.e., it is in charge of controlling and regulating the trade-off between latency and accuracy.

In Formula 3.12 we can observe that, the higher LAT(m) is, the higher will be the result of the multiplication, which at most will be equal to ACC(m). This is because LAT(m) will always be less than or equal to T (to satisfy the minimum requirements) so, the highest value by which we can multiply ACC(m) is 1. But, if LAT(m) == T, we will be at the worst possible situation for LAT(m) since it will have the highest possible value. And, if LAT(m) == 0, we will be at the best possible situation for LAT(M) but the reward would be equal to 0, which is a very bad result since we want to maximize it.

That is why the weight factor is used, because it balances the values of accuracy and latency to obtain the best possible reward without harming the ACC(m) or the LAT(m). Specifically, omega (ω) is expressed as in Equation 3.13.

$$\omega = \begin{cases} \alpha, & if LAT(m) \leq T \\ \beta, & otherwise \end{cases} \tag{3.13}$$

To decide what value to give to alpha (α) and beta (β) in [TCP+19] they take into account that the reward obtained must be equal between any pair of values of accuracy and latency. To calculate the optimal values of alpha (α) and beta (β) they are also based on observation Equation 3.14.

$$2 \times latency \rightarrow (0.5 \times accuracy) + accuracy \tag{3.14}$$

That is, if the latency of the model is twice that of the previous one, then there is a 5% improvement in accuracy. So, as in [TCP+19], if we have two models, M1 and M2 (with twice the latency of M1 and, consequently, 5% more accuracy), defined as in Equations 3.15 and 3.16.

$$Reward(M1) = ACC(M1) \times \left[\frac{LAT(M1)}{T}\right]^{\omega} \tag{3.15}$$

$$Reward(M2) = (ACC(M1) \times (1 + 0.5)) \times \left[\frac{2 \times LAT(M1)}{T}\right]^{\omega} \tag{3.16}$$

Given that we know the reward of the M1 model has to be more or less equal to the reward of the M2 model, by equaling both rewards and clearing beta (β), in [TCP+19] they obtain that beta (β) must be equal to -0.07 (best value obtained for omega (ω), so

this is the value normally used for both alpha (α) and beta (β)).

The only change made for the two architectures of MobileNetV3 is with respect to the value of the weight factor (ω). That is because the developers of this model realized that when applying the model to cell phones, the accuracy is modified much more if the latency is also modified, than in models used for environments with more lax properties. So, instead of using omega (ω) = -0.07, omega = -0.15 is used, which they discovered to compesate more for the difference in accuracy when the latency changes.

3.2.3.3 NetAdapt Algorithm

NetAdapt is the next step followed in [HSC+19] to get the best design of both versions MobileNetV3. This algorithm is applied right after NAS so, we assume that the model to which we are going to apply the NetAdapt has an acceptable accuracy and latency. This algorithm aims to further improve the accuracy and latency of the model. To do so, it performs the following steps in each iteration:

1. It proposes improvements of the architecture characteristics that help the model to lower the latency value. An example of a modification that can be applied is to reduce the number of channels of an expansion channel, taking into account the residual connections since the number of channels of the two inputs of the Add layer must remain the same.

2. After having applied this proposal, an approximation of the accuracy of the model with the latency reduction is obtained.

Once the latency and accuracy are obtained, the proposal that returns the best values is chosen. To choose the best option we will pay attention to the Equation 3.17. This calculates the change in accuracy with respect to the change in latency, so we will choose the proposal that returns a lower result in this equation.

$$\frac{\Delta\ Accuracy}{\Delta\ latency} \tag{3.17}$$

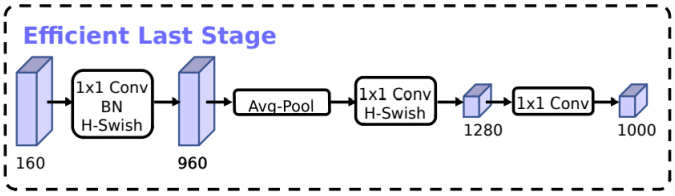
3.2.3.4 Network Improvements

Finally they realized that the last block of the model they are inspired by, MobileNetV2, is very inefficient in terms of the time it uses to execute. So, the solution they propose for this is to move the GlobalAveragePooling layer just before the expansion layer. If we look at the MobileNetV2 architecture 3.2 we can see that the last convolution layer (Conv2D 1x1) works with tensors of spatial size 7x7x320. This is the layer that expands the number of channels to obtain a higher dimensional tensor, that is, it applies 1280 filters to the

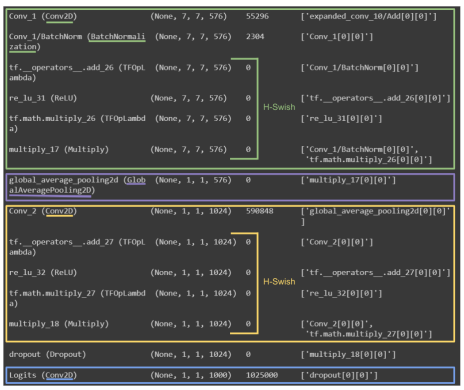
input of size 7x7x320 to get a 7x7x1280 tensor, that is a lot of computational cost which will make the model much slower.

As this layer is important to be able to have better features to classify with later and, cannot be removed, in MobileNetV3-Small and Large they propose to put that layer just after the GlobalAveragePooling. This way the expansion operation will have to work with an input size of 1x1xDEPTH. Also, by doing this we do not need a bottleneck at the end to reduce the dimensions, since we already have 1x1xDEPTH tensors.

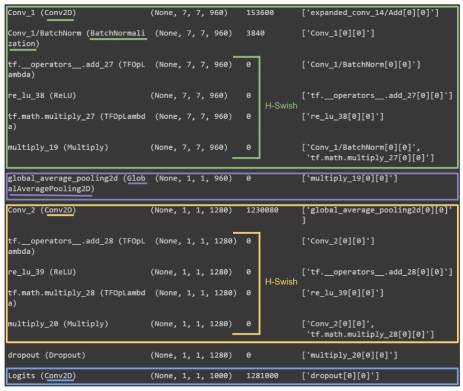
Below is an image of the architecture of the last layers of the two versions of MobileNetV3 taken from the architecture implemented in TensorFlow (Figure 3.17(b) and Figure 3.17(c)). We can see how the GlobalAveragePooling layer is placed just after the Conv2D layer. This pooling layer transforms the input spatial shape 7x7 to 1x1. Just after that, the expansion layer will be able to increase the number of channels much faster because it is working with 1x1 tensor (we can see that the DW CONV layer is no longer needed). We can see in Figure 3.17 that the implementation of TensorFlow of both models matches with the original scheme from [HSC+19].



(a) Efficient Last Stage of MobileNetV3 ([HSC+19])



(b) Efficient Last Stage of MobileNetV3-Small (TensorFlow)



(c) Efficient Last Stage of MobileNetV3-Large (TensorFlow)

Figure 3.17: Comparison between Original and TensorFlow implementation of the Efficient Last Stage of the two versions of MobileNetV3

Finally, as in MobileNetV1 and MobileNetV2, we show a table of the entire body of the MobileNetV3 architectures implemented in TensorFlow, Table 3.3 and 3.4. We would still have to define only the initial block of the architecture, which is the following (Figure 3.18). We can compare this block with the Conv2D from MobileNetV2 (Figure 3.10(a)) and see

that it has the same structure except that in MobileNetV2 their non-linear function was ReLU6 and, in MobileNetV3, the non-linear funcion used is Hard Swish.

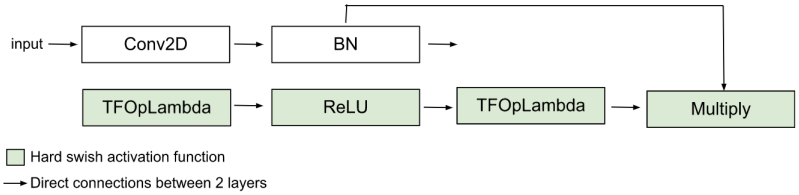


Figure 3.18: Conv2D: Initial Block of MobileNetV3-Small and Large

Table 3.3: Body Architecture of MobileNetV3-Small implemented in TensorFlow

Block / Layer	**Input Shape**	**Output Shape**	**Expansion Size**	**Squeeze and Excitation**	**Non-Linearity**	**Stride**
Conv2D	224x224x3	112x112x3	-	NO	Hard Swish	2
Block 1	112x112x16	56x56x16	16	YES	ReLU	2
Block 1	56x56x16	28x28x24	72	NO	ReLU	2
Block 2	28x28x24	28x28x24	88	NO	ReLU	1
Block 1	28x28x24	14x14x40	96	YES	Hard Swish	2
Block 2	14x14x40	14x14x40	240	YES	Hard Swish	1
Block 2	14x14x40	14x14x40	240	YES	Hard Swish	1
Block 1	14x14x40	14x14x48	120	YES	Hard Swish	1
Block 2	14x14x48	14x14x48	144	YES	Hard Swish	1
Block 1	14x14x48	7x7x96	288	YES	Hard Swish	2
Block 2	7x7x96	7x7x96	576	YES	Hard Swish	1
Block 2	7x7x96	7x7x96	576	YES	Hard Swish	1
Conv2D, 1x1	7x7x96	7x7x576	-	-	Hard Swish	1
GlobalAverage Pooling	7x7x576	1x1x576	-	-	-	1
Conv2D, 1x1	1x1x576	1x1x1024	-	-	Hard Swish	1
Dropout	1x1x1024	1x1x1024	-	-	-	1
Standard Conv2D	1x1x1024	1x1x1000	-	-	-	1
Flatten	1x1x1000	1000	-	-	-	1
Predictions (Activation)	1000	1000	-	-	-	1

It can be seen in Tables 3.3 and 3.4 that, as in MobileNetV2, Block 1 is the one that modifies the spatial dimensions and therefore has stride equal to 2 but it does not have residual connection nor the Add layer. The only exception that exists is the layer of the row number 8 in the case of Table 3.3 and the layer of the row number 12 of Table 3.4, where there is a Block 1 but its stride is equal to 1, this is because in these two layers the number of channels are modified and not the spatial dimensions therefore, we do not have stride equal to 2 and can not have residual connection or the Add layer. That is because

Table 3.4: Body Architecture of MobileNetV3-Large implemented in TensorFlow

Block / Layer	**Input Shape**	**Output Shape**	**Expansion Size**	**Squeeze and Excitation**	**Non-Linearity**	**Stride**
Conv2D	224x224x3	112x112x16	16	NO	Hard Swish	2
Block 2	112x112x16	112x112x16	16	NO	ReLU	1
Block 1	112x112x16	56x56x24	64	NO	ReLU	2
Block 2	56x56x24	56x56x24	72	NO	ReLU	1
Block 1	56x56x24	28x28x40	72	YES	ReLU	2
Block 2	28x28x40	28x28x40	120	YES	ReLU	1
Block 2	28x28x40	28x28x40	120	YES	ReLU	1
Block 1	28x28x40	14x14x80	240	NO	Hard Swish	2
Block 2	14x14x80	14x14x80	200	NO	Hard Swish	1
Block 2	14x14x80	14x14x80	184	NO	Hard Swish	1
Block 2	14x14x80	14x14x80	184	NO	Hard Swish	1
Block 1	14x14x80	14x14x112	480	YES	Hard Swish	1
Block 2	14x14x112	14x14x112	672	YES	Hard Swish	1
Block 1	14x14x112	7x7x160	672	YES	Hard Swish	2
Block 2	7x7x160	7x7x160	960	YES	Hard Swish	1
Block 2	7x7x160	7x7x160	960	YES	Hard Swish	1
Conv2D, 1x1	7x7x160	7x7x960	-	NO	Hard Swish	1
GlobalAvarage Pooling	7x7x960	1x1x960	-	NO	-	1
Conv2D, 1x1	1x1x960	1x1x1280	-	NO	Hard Swish	1
Dropout	1x1x1280	1x1x1280	-	NO	-	1
Standard Conv2D	1x1x1280	1x1x1000	-	NO	-	1
Flatten	1x1x1000	1000	-	NO	-	1
Predictions (Activation)	1000	1000	-	NO	-	1

the number of output channels of the block is not the same as the input ones. As for Block 2, you can also see that it is the same implementation as in MobileNetV2, it has stride equal to 1 so the output dimensions do not change and, therefore, the residual connection and the Add layer can be included.

We would like to point out that the Flatten layer is applied in both architectures of MobileNetV3 to convert the input that has three dimensions (1x1x1000) into a single dimension (1000) without modifying its values. Once we have the modified input, we can make predictions with the Predictions layer, which in turn, will apply a non-linear activation function in order to offer the user a series of values that indicate the percentage of possibility that the studied object belongs to each class.

We can observe that the MobileNetV3-Small and MobileNetV3-Large architectures

have the same input block and the same final blocks, the only difference is that MobileNetV3-Large has more intermediate blocks that combine the Squeeze-and-Excitation block with the non-linear ReLU function and others that do not incorporate the Squeeze-and-Excitation block but implement the non-linear Hard Swish function.

To finish this explanation of the four MobileNet architectures, we made a comparative Table 3.5 between them indicating the number of parameters of each one and the number of layers they have. In this way, we will be able to see how the versions evolve. This values were obtain from the models imported from Keras TensorFlow.

Table 3.5: Comparison between versions of MobileNet

Model	Nº Layers	Nº Parameters
MobileNetV1	91	4,253,864
MobileNetV2	156	3,538,984
MobileNetV3-Small	239	2,554,968
MobileNetV3-Large	273	5,507,432

It can be observed in the Table 3.5 that, as the architecture of the models were improved, the number of layers increased but the number of parameters of each model decreased. In other words, as the versions advance, the architectures improved. It can be seen that the only one that does not decrease the number of parameters is MobilenetV3-Large because, basically, this architecture is an extension of MobileNetV3-Small. In short, each of the versions is more adaptable to a cell phone because decreasing the number of parameters and, therefore the size of the model, makes the models more accurate for a mobile.

3.3 Data Augmentation

One of the most commonly used techniques to increase the amount of data in a dataset is DA. The main advantage of this mechanism is that it allows us to reduce the chances of the model to under-fit and the main disadvantage is that, by having to apply this technique to each image, the training time increases. What DA does is to make changes to the images that already exist in the dataset, some of the changes that can be applied are rotating the image, zooming in or out, adding noise, modifying the brightness or increasing the contrast of the colors, among many others. A great definition of DA is the one that [SK19] gives; applying DA to an image is as if our brain imagines a picture that it has never really seen. In this way, the DL algorithm manages to learn more patterns and features of the images although they do not really exist in the initial image.

There are many previous works where they apply DA in the training phase to improve the model results. For example, in [POSW17] they use DA to detect the class of plants. They

apply some techniques such as rotation, blur, scaling, contrast, illumination or projective and find out that applying DA manages to increase the accuracy of the tested model (AlexNet) by 8.8% with some of the combinations of the techniques indicated above. And, for their second model tested (GoogleNet), it achieves the best accuracy, 99.42%, by applying the DA technique of illumination to all its images. They test with two more plant datasets and in those they also get the best results by applying the different DA techniques. Later in [PVAV18] they used DA in images of skin lesions to help doctors identify what disease any patient is suffering. They managed to obtain good results specifically applying geometric techniques (such as random flip, rotate or zoom) combined with color transformations, specifically, it achieves 85.4% accuracy for the Inception-v4 model, 88.2% accuracy for the ResNet model and 87.9% accuracy for the DenseNet model. After that, in 2021, there was a really interesting research [ES21] which utilized DA methods to recognize plant leaf disease with MobileNetV2 model. They tried with several techniques and the best results were obtained with zoom (95.39% of accuracy), rotation (94.62% of accuracy) and rotation+zoom (93.85% of accuracy). So, they obtained an upgrade of 2.31% because the original percentage without any DA was of 93.08%. This gives us more confidence in thinkin that DA will be useful in our research.

Although, as indicated and verified in [KCM22] there are some DA techniques, such as the change of brightness, that worsen the accuracy obtained by the models. In addition, [ES21] which was trained and tested over MobileNetV2, confirms that the brightness DA technique obtained worst results than any of the other methods. Later we will perform tests applying different DA options and we will see if our results agree with this.

Giving all this background, we can confirm that DA is going to help the model perform better predictions. Later on, in our experiments, we will show which techniques of DA we applied to our images and which ones gave us the best results.

Chapter 4

Benchmarking of Deep Learning Models

In this chapter we will explain and analyze all the training we have performed with the four models explained in the previous chapter, MobileNetV1, MobileNetV2, MobileNetV3-Small and MobileNetV3-Large. To do so, we will introduce the datasets we have used for the different trainings and the specifications of each of them. All training and tests were performed on a Windows 10 computer with Intel Core i7-7920HQ CPU with 3.10GHz and a RAM of 16.0 GB. Also, a GPU was used with the following specifications, GForce RTX 3070 OC Edition with 8 GDDR6. And, to communicate the computer and GPU we used an external box Razer Core X Chroma. In addition, work was done in the Visual Studio environment, using the Python programming language. The Keras library, which exists within TensorFlow, was also used to import the architectures of the 4 models to be tested and was also used to obtain different functions that are already implemented there. The chapter is organized as follows, in Section 4.1 we explain the different code used to make all the preparation before training the models. In Section 4.2 the four models are trained within a dataset containing 2 classes, this section is divided into 5 subsections. The 4 first Subsections 4.2.1, 4.2.2, 4.2.3 and 4.2.4 are the trainings made on each of the model with the same dataset. The Subsection 4.2.5 refers to the selection of the best of the 4 models based on the previous training and testing metrics. Then, we have Section 4.3 that is Data Augmentation trials on the best model chosen. And, finally, we train that best model on three different datasets on Section 4.4.

4.1 Preparation before training

The first step to be able to perform any training is to import the images and, for this, the flow_from_directory function of the ImageDataGenerator class was used. This class is

inside the module TensorFlow Keras Preprocessing Image and allows us to load batches of images into an output generator. What the flow_from_directory function does is: take the address of a single path, go through each of the folders inside that path and, finally, save each image inside the folder in the output generator. The real label of each of those images is the name of the folder in which it is stored. One of the arguments of the ImageDataGenerator class is the preprocess function. By allowing us to pass the preprocess function at the same time the flow_from_directory function is called, each image extracted from disk can go directly through the necessary preprocessing. This is applied right after the image resize is done.

The preprocessing functions that we used on the images in each of the four models are already implemented in the Keras Applications set of models Each of those models have some methods defined in Keras and, fortunately, one of them is the preprocessing function. In summary, the functions that we used were the following:

- "tf.keras.applications.mobilenet.preprocess_input" for **MobileNetV1** converts the values of the pixels of the input image from the range 0-255 to the range -1 to 1.
- "tf.keras.applications.mobilenet_V2.preprocess_input" for **MobileNetV2** converts the values of the pixels of the input image from the range 0-255 to the range -1 to 1.
- "tf.keras.applications.mobilenet_V3.preprocess_input" for **MobileNetV3-Small** and **MobileNetV3-Large** does not modify the entry.

Moreover, we know from the body of the architectures, that the input images in any of the four MobileNet architectures have to have size 224x224 (showed in Tables 3.1, 3.2, 3.3 and 3.4). So, when we import the images we adjust their size to 224x224. For this, we only specified in the target_size argument of the flow_from_directory function the size (224,224). The last thing we had to indicate to the function was the batch_size, which we usually set to 64. This value indicates the number of images that the algorithm trains or validate at a time in the training process and, the number of images that are predicted at a time in the prediction process. Therefore, an example of loading a set of training images into MobileNetV1 would be as follows:

- **train_batches** = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(directory=train_path, target_size=(224,224), batch_size=64)
- **valid_batches** = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(directory=valid_path, target_size=(224,224), batch_size=64)

- **test_batches** = ImageDataGenerator(preprocessing_function=tf.keras.applications.mobilenet.preprocess_input).flow_from_directory(directory=test_path, target_size=(224,224), batch_size=64, shuffle=False)

For the rest of the models we employed the same structure except that the preprocessing function was the one corresponding to each model. We must also point out that the validation set is different from the test set, as we can see in the preceding list. The difference is that the validation set is used in the training in order to calculate the accuracy and error of the model throughout the training process and not afterwards. In each epoch, the model is first trained with the training set and, just after, before jumping to the next training epoch, the validation images are passed and the predictions are made with what the model has learned so far. In this way we can get the training accuracy and the validation accuracy and compare them to see if any of the two worst scenarios is happening; the model has over learned or, on the contrary, it has not learned enough.

Once we had the train, validation and test images loaded in the respective generators, the next step was to load the model. As we have indicated above, the four models are implemented in Keras Applications already pre-trained. To load each of these models, you only have to call each of these functions:

- **MobileNetV1** = tf.keras.applications.mobilenet.MobileNet()
- **MobileNetV2** = tf.keras.applications.MobileNetV2()
- **MobileNetV3Small** = tf.keras.applications.MobileNetV3Small()
- **MobileNetV3Large** = tf.keras.applications.MobileNetV3Large()

Once the necessary model is obtained, we move on to the phase of modifying the model (fine-tunning). But, before fine-tunning any model, we will reason why we do not retrain any architecture from scratch. This is because we have used the Transfer Learning technique. This technique consists of reusing models that have been already trained on a related but different dataset from the one to be used. The great advantage of this is that it saves a lot of time in training since the model has already learned essential characteristics of the images and we will not have to "teach" it from cero. Additionally, if the model is pre-trained on a larger dataset, the chances of the model to under-fit decrease considerably even if the dataset we use is small. And that is why the retraining we did was done on some of the last layers of the model and not all of them. Since by default the layers are loaded as trainable when we import the model, we must go through all the layers of the model that we do not want to be trained and tell them to freeze. We will do that as follows:

```
    for layer in model.layers[:-4]: #leave the last 4 layers trainable
        layer.trainable = False
```

Listing 4.1: Freeze Layers

Another important point is that Batch Normalization layers must never be unfrozen, i.e. they cannot be retrained. The peculiarity of this layer is that its two non-trainable weights, the mean and variance, are updated at each iteration as they adjust to their input data. Thanks to this, these layers acquire more and more information as more training is performed and, consequently, the values of the mean and variance get better and better adjusted to any input data. Then, when we do the Transfer Learning technique, instead of retraining the batch normalization layers and losing all this information that had been obtained in the other training, what is done is to take advantage of the mean and variance that has been previously calculated with many data and thus, improve the quality of the retraining learning.

As we have seen, the architecture of the four models have the Batch Normalization layer and to prevent them from unfreezing we must make sure that, after having unfrozen the rest, these have not been modified, for that we have the following piece of code which is applied to every model before the training process starts:

```
    for layer in model.layers:
        if 'bn' in layer.name:
            layer.trainable = False
```

Listing 4.2: Freeze every BatchNormalization Layer

In addition, the four models that we import from the Keras Applications set are already trained on the ImageNet image set. This dataset contains images belonging to 1000 possible categories and, in the case of the four MobileNet models that we are going to use, about 1.2 million images were used for the training process. These images can be cars, airplanes, boats, any kind of animal, mountains, beaches, food, etc. Thus, we need to retrain only part of the models with our dataset because they will have already learned to identify details such as edges, contours, shapes, image backgrounds, among many others, on any image.

Moreover, we have the two different optimizers that we used to retrain the models. These are the Adam optimizer and the SGD optimizer.

- Stochastic Gradient Descent (SGD) Optimizer: this optimizer is an improvement of the standard gradient descent. Both try to find the point where the model's loss function is lowest. The difference is that the standard gradient descent modifies the weights and bias of the model a number of times and chooses the values that reduce the cost to the current value. In contrast, SGD optimizer randomly modifies

the weights and bias values and follows the search for the minimum cost with those values, thus the computational time is much lower.

- Adam Optimizer: it is also an improvement of the standard gradient descent. They realized that, if the minimum cost search was always going in the same direction, why not increase the learning rate and thus reach the final objective much earlier, and this is how it was implemented. And it also fits the opposite case, if we have a too high learning rate, i.e. the cost is oscillating between very different values, why not minimize the learning rate and thus reach the minimum cost at some point. So these were the two improvements that were implemented by the Adam optimizer.

Finally, just before calling the "fit" function that retrains the model, we must compile it. To the method that compiles the model we pass as arguments the optimizer that we want to apply, the loss function that we want to use (that is, the function that measures the loss that the model has and that, later, will allow us to get the graph of the train and validation loss) and finally, we pass the metric that we want to evaluate while training the model. For the loss function we use the categorical crossentropy since it measures the difference between the probability that the model classifies the image in a class and, the actual label of that image. And the metric that we want evaluate is the accuracy, so that's what we pass to that argument. An example of a compiler call with any of the four models using the Adam optimizer would be the following:

```
1    model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.001),
2    loss='categorical_crossentropy', metrics=['accuracy'])
```

Listing 4.3: Model Compilation

After running the various tests without any modifications to the input images, we started to apply DA to the images to improve the future performance of the model. We tested two ways of applying DA:

1. Data Augmentation applied before training: this was the first attempt of applying DA to our dataset. The thing is that by adding DA to the images before training them, we had to save them somewhere so that we could then pass them to the training process. This mechanism caused us to double the number of images on the disk and, therefore, the space occupied on it. We tested this technique with the first dataset that we used, which as we will indicate later had 24,998 images per class. So, 49,996 extra images were stored on disk. In addition, the time it took to generate these augmented images was 1 hour and 17 minutes.

2. Data Augmentation on the fly (online, during training): this was the option chosen to apply DA to all models as it allowed DA to be applied to each image directly in the training. Also, is much more effective because in each training epoch the images

are enhanced in a different way. This is because, as we will see in the following code, we give a probability of application to each DA technique and, depending on the value randomly chosen by the algorithm, more or less transformation will be applied to the image. It adds some time to the model execution but nothing to do with applying DA before executing the training phase. To apply this way of adding DA we first had to change the way of importing the images:

```
train_batches = tf.keras.utils.image_dataset_from_directory(
    directory=train_path, validation_split=0.2, subset="training",
    label_mode='categorical', image_size=(224,224), seed=1,
    batch_size=64)
valid_batches = tf.keras.utils.image_dataset_from_directory(
    directory=train_path, validation_split=0.2, subset="validation",
    label_mode='categorical', image_size=(224,224), seed=1,
    batch_size=64)
```

Listing 4.4: Import training and validation images to apply DA

As we can see, instead of using the ImageDataGenerator class and the flow_from_directory method we had to use the image_dataset_from_directory method from the tf.keras.utils module. This way, we were able to add DA to the images at the training phase time. The following code is the one we used to apply DA to every image:

```
def preprocess(images, labels): #preprocess function for MobileNetV1
    return (tf.keras.applications.mobilenet.preprocess_input(images),
    labels)

AUTOTUNE = tf.data.AUTOTUNE

#Add every DA technique
data_augmentation = tf.keras.Sequential([
tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal"),
tf.keras.layers.experimental.preprocessing.RandomRotation(0.05,
fill_mode="constant"),
tf.keras.layers.RandomContrast(0.1),
tf.keras.layers.GaussianNoise(0.2),
tf.keras.layers.RandomBrightness(factor=(0.2)),
tf.keras.layers.RandomZoom(height_factor=0.2, width_factor=0.2,
fill_mode="constant") ])

def prepare(ds, shuffle=False, augment=False):
    ds = ds.map(preprocess)
    if shuffle:
        ds = ds.shuffle(len(ds))

    if augment:
        ds = ds.map(lambda x, y:
                    (data_augmentation(x,training=True), y),
```

```
                    num_parallel_calls=AUTOTUNE)

    return ds

train_batches = prepare(train_batches, shuffle=True, augment=True)
train_batches = train_batches.prefetch(tf.data.AUTOTUNE)

valid_batches = prepare(valid_batches, shuffle=False, augment=False)
valid_batches = valid_batches.prefetch(tf.data.AUTOTUNE)

```

Listing 4.5: Apply DA on the fly

We can see that to each DA technique we add a percentage of application, all these values must be low because if we raise it to more than 0.2 we obtain very unrealistic images. It is very unlikely that the model will receive such modified images and therefore, the model will be learning useless information. At most we give 0.2 probability to each DA technique and later, in training, the model will randomly choose what percentage of that technique is added to the images (between 0 and the percentage we have indicated). In Figures 4.1 and 4.2 we show two examples of how the images change with each of the six techniques of DA.

(a) Normal (b) Flip (c) Rotation (d) Contrast (e) Gaussian Noise (f) Brightness (g) Zoom

Figure 4.1: Data Augmentation techniques applied to a Husky image

(a) Normal (b) Flip (c) Rotation (d) Contrast (e) Gaussian Noise (f) Brightness (g) Zoom

Figure 4.2: Data Augmentation techniques applied to a coffee image

The flip technique acts as if it was a mirror and flips the image completely, specifically, that flip is done horizontally because we pass to the RandomFlip class the value “horizontal”. Then, the rotation technique in this case flips the images in the same direction in the husky and the cooffee image but, it could have been applied to the other side. Also, when the image is rotated and leaves empty gaps, we specify that they are filled with black

pixels by passing to the fill_mode argument the value "constant". We can also appreciate how brightness technique can be applied either to lighten or darken the image. Then, the zoom technique can be applied to zoom in or zoom out the image and, when the image is zoomed out, we specified that the borders are filled with black pixels by passing to the RandomZoom class fill_mode="constant". Finally, the contrast technique simply applies a certain amount of contrast to the image and the Gaussian technique applies Gaussian noise to the image. As mentioned before, all these modifications are applied randomly to each image in the training process.

It is important to point out that DA is added to the training dataset but not to the validation dataset. Although the training and validation datasets that we are using are already different, by adding DA to the training data and not adding it to the validation data, the differences that already exist between them will increase a lot. This is very useful because in this way we can better check how the model manages to generalize, i.e., how it manages to guess the class of an image it has never seen. Finally, we would like say that in the following tests, we added these 6 types of DA to see and select which model performed best in the face of these changes.

Now we will go on to explain version by version and dataset by dataset what changes we have applied and the different results obtained.

4.2 Model Training on 2 Classes

Once we understood the architecture of each model, we went on to perform the training of each one of them. The goal we wanted to achieve was being able to classify as accurately as possible the class of each of the images that came to the model. For this, we trained the four models on the same dataset and chose the one that gave us the best results.

For this first phase we used a training dataset with two classes, the first class is identified as "normal" (i.e. images without any danger or risk of harm to anyone viewing it) and the second class is labeled as "sexual content". This dataset consists of 33,096 images of which 24,998 are training images (12,499 images from each class), 4,099 validation images (2,049 images from the normal class and 2,050 images from the explicit class) and, finally, 3,999 test images (2,002 images from the normal class and 1,997 images from the explicit class). All of these images occupied 6.47 GB in total on disk, because they had high quality.

4.2.1 MobileNetV1 trained on 2 Classes

Based on previous experiments, we decided to replace the last 5 layers of the model with a Dropout layer and a Dense layer. The Dropout layer was added because we realized that this model overlearned from the second training epoch and, the removal of some of the

connections between neurons, helped the model not to overlearn and better predict new input images. The Dense layer is applied at the end of the model with 2 output neurons because the model is going to make predictions on 2 classes. In addition, we apply the final activation function "softmax". To do this we used the following lines:

```
x = mobileNetV1.layers[-5].output #x = all the layers except the last 5
x = tf.keras.layers.Dropout(.5, input_shape=(1024,))(x)
output = Dense(units=2, activation='softmax')(x) #output layer
mobileNetV1 = Model(inputs=mobileNetV1.input, outputs=output) #final model
```

Listing 4.6: Modify last MobileNetV1 layers

The reason why we apply the softmax activation function on the Dense layer is because it returns a vector of probabilities. In our case, this vector has two positions, the first one referring to the normal class and the second one to the sexual content class. Each of the positions of that vector contains the percentage of chances that the image has to belong to each of those classes. So, if it returns [0.76, 0.24], it means that the image has a 76% probability of being normal and a 24% probability of having sexual content. Finally, the image will be classified as the class with the highest probability.

After these modifications in the layers we will explain what changes on the training parameters we apply to the model. We mainly focus on the following points:

1. Number of trainable layers: we started by unfreezing (letting retrain) only the last 3 layers, then the last 4, then the last 5 and so on up to the last 13 layers.

2. Optimizer: we applied the two optimizers that we have explained, the Adam optimizer and the SGD.

3. Learning rate: for this parameter we tried a naive approach giving several values to the learning rate. We discovered that starting with 0.001 was a good approach. So, from here on, our experiments are done with initial rate equal to 0.001. In fact, this is the value applied to the learning rate by default in both optimizer's functions.

Table 4.1 will show the metrics obtained by training MobileNetV1 applying the different parameters. It also indicates the amount of time each training has taken. After training and validating the model for 10 epochs, we also made the testing phase to see how the model perform the predictions. To analyze the testing phase, we obtained the confusion matrix of each one of them shown in Table 4.2. The table indicates the true negatives, false negatives, true positives and false positives of each test.

In row 4 of Table 4.1 we can observe that the best result obtained with Adam's optimizer is 95.16% in training accuracy and 94.29% in validation accuracy. This result is not very far from the best result obtained with the SGD optimizer. The problem is that if we look at the graphs obtained with Adam's optimizer, Figure 4.3, we can see that the trajectory

Table 4.1: Training results on MobileNetV1 with different parameters

	Nº Trainable Layers	Optimizer	Time spent (mins)	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1º	3	Adam	47,57	0.2419	0.9118	0.1680	0.9354
2º	4	Adam	48,35	0.2067	0.9187	0.1556	0.9427
3º	5	Adam	55,20	0.2016	0.9216	0.1608	0.9383
4º	10	Adam	83,33	0.1248	0.9516	0.1434	0.9429
5º	3	SGD	52,21	0.2336	0.9080	0.1750	0.9334
6º	**4**	**SGD**	**52,36**	**0.1939**	**0.9234**	**0.1511**	**0.9471**
7º	5	SGD	55,25	0.2327	0.9105	0.1683	0.9361
8º	10	SGD	56,56	0.1841	0.9279	0.1520	0.9444
9º	11	SGD	62,51	0.1820	0.9279	0.1478	0.9478
10º	12	SGD	70,24	0.1787	0.9302	0.1484	0.9478
11º	13	SGD	85,02	0.1586	0.9381	0.1360	0.9478

Table 4.2: MobileNetV1 Confusion Matrix Summary

	TN	FN	TP	FP
1º	1870	93	190	129
2º	1904	110	1887	97
3º	1935	159	1838	66
4º	1974	185	1812	27
5º	1916	164	1833	85
6º	**1901**	**112**	**1887**	**98**
7º	1899	148	1849	102
8º	1909	129	1868	92
9º	1914	125	1872	87
10º	1903	105	1892	98
11º	1923	111	1886	78

followed by both functions oscillates between several values instead of converging to a unique point. This indicates that the optimizer fails to establish a safe path to model convergence and this is why we did not keep training the model with it.

We will now focus on the SGD optimizer. We can observe that the best results were obtained with 4, 11, 12 and 13 trainable layers. In fact, 11, 12 and 13 obtain exactly the same validation accuracy, although we can see that the training accuracy is increasing as we go from 11 to 13 trainable layers. We could think that the best option is to choose to unfreeze 13 layers since it is the one that obtains better values of accuracy and it also get good results on the confusion matrix with respect to the rest of training experiments, as we can see in Table 4.2. The problem is that as we increase the number of trainable layers, the accuracy function and the loss function starts to oscillate between different values instead of converging. We can see the poor evolution of these functions in the Figure 4.4. In addition, the training execution time increases more and more as we increase the

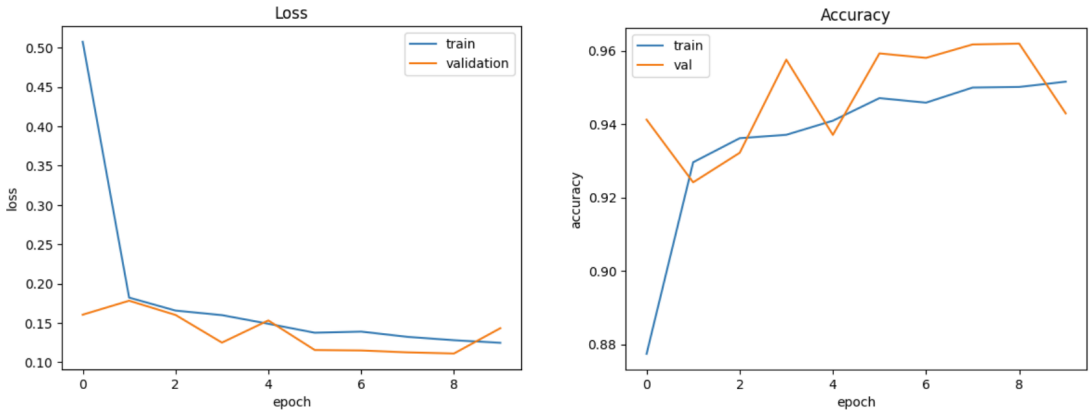


Figure 4.3: Graphs obtained from the 4º test of MobileNetV1 4.1

trainable layers (as it is shown in the Table 4.1).

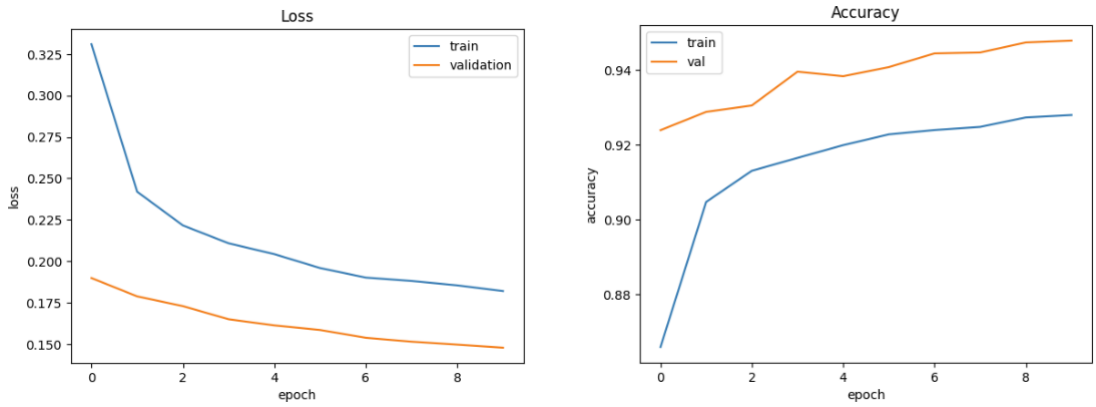
For all these reasons we decided that the best result obtained is training 4 layers and using SGD optimizer. As shown in Table 4.1 this configuration achieves 94.71% accuracy (which is only 0.07% less than with 13 trainable layers) and it takes 52,36 minutes (that is, 33 minutes less than with 13 trainable layers). Moreover, we can see in Figure 4.5 that the training loss is always above the validation loss, there is a very small distance between both (around 1%) and they converge to the same point, this indicates that the model has a good fit, i.e. it does not underfit nor overfit. In conclusion, it will be able to generalize and predict correctly.

So, we can say that SGD optimizer is doing a better job than Adam optimizer. This is supported by previous studies [ZFM+20], where they show that SGD is much better to use in deep learning models that want to classify or detect patterns in the input data.

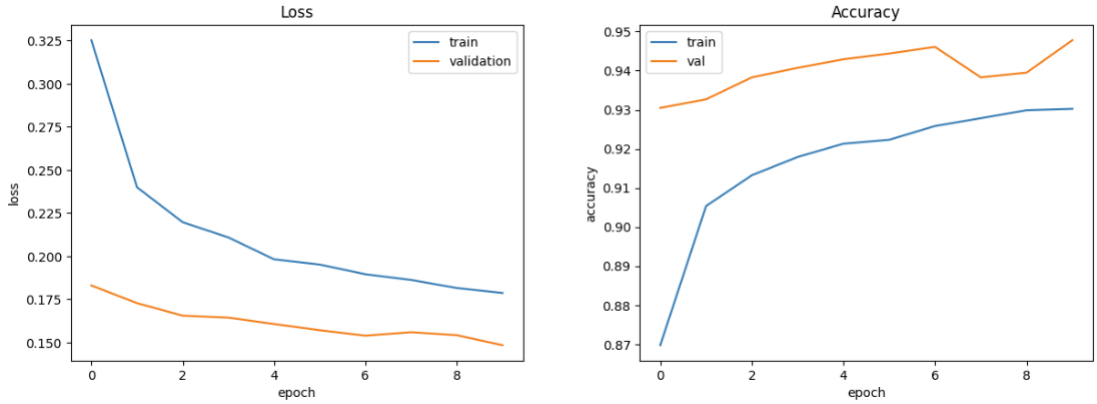
Then we applied the six techniques of DA (flip, rotation, contrast, Gaussian noise, brightness and zoom) to make the model learn more patterns on the images. For this training we used the best results obtained from the previous tests, that is: 4 trainable layers, SGD optimizer and initial learning rate equal to 0.001. The final metrics obtained were the ones on Tables 4.3 and 4.4 and the loss and accuracy functions were Figure 4.6. This values will be compared in the following pages with the results of the other models trained with DA.

Table 4.3: Results for DA applied to MobileNetV1

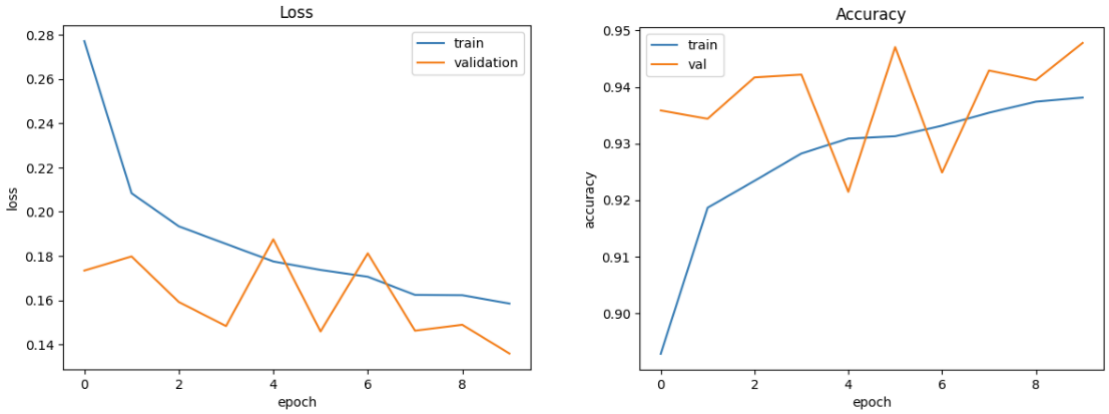
Training Loss	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
0.6310	0.6102	0.6652	0.6289



(a) Graphs obtained from the 9º test of MobileNetV1 4.1



(b) Graphs obtained from the 10º test of MobileNetV1 4.1



(c) Graphs obtained from the 11º test of MobileNetV1 4.1

Figure 4.4: Evolution Loss and Accuracy functions increasing the nº of trainable layers

Table 4.4: Confusion Matrix for DA applied to MobileNetV1

TN	FN	TP	FP
1541	984	1015	458

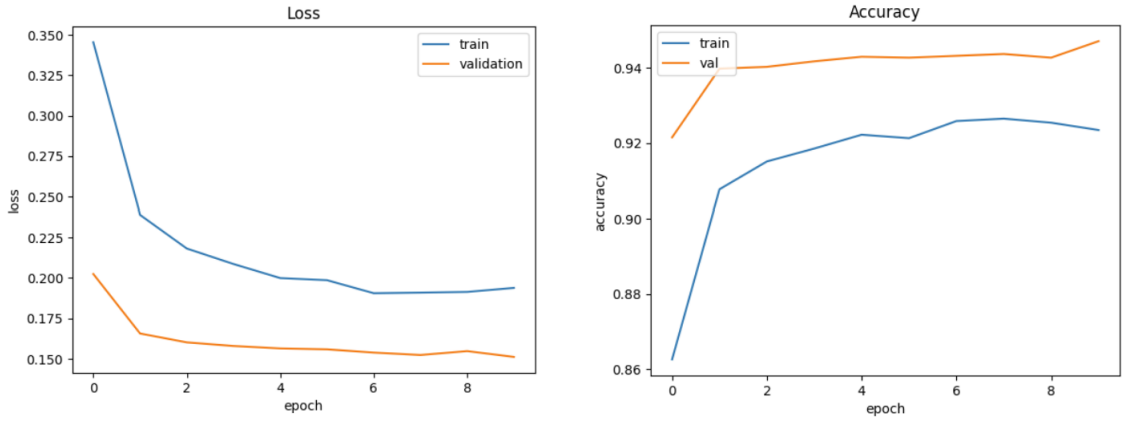


Figure 4.5: Graphs obtained from the 6º test of MobileNetV1 4.1

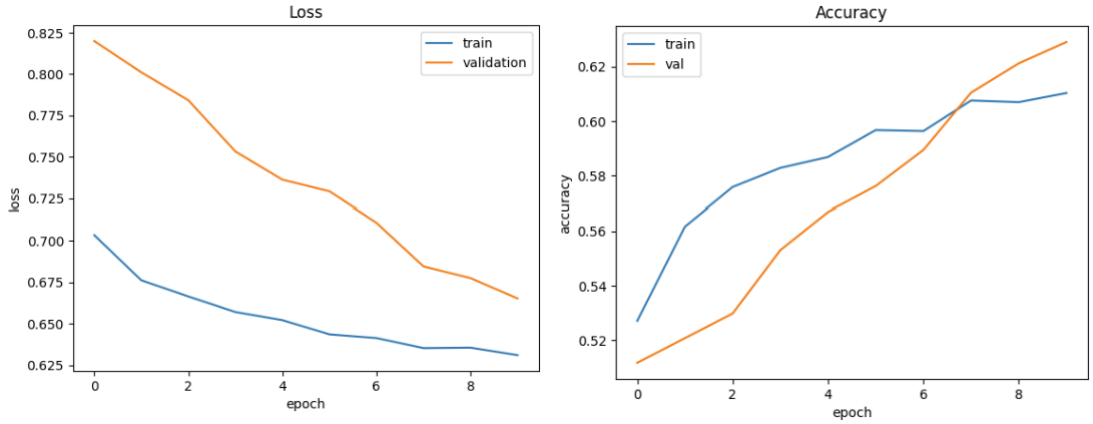


Figure 4.6: Graphs obtained from the DA test of MobileNetV1 4.3

4.2.2 MobileNetV2 trained on 2 Classes

We did not find previous studies indicating which layers had been modified in this model so, we simply added a Dropout layer at the end of the model so that the model would not over-learn. We also added the last Dense layer with 2 neurons, for the same reasons as in MobileNetV1. In addition, we modify the three parameters (number of trainable layers, optimizer and learning rate) for each training with the same values as in MobileNetV1. In Table 4.5 we show the results obtained.

Table 4.5: Training results on MobileNetV2 with different parameters

	Nº Trainable Layers	**Optimizer**	**Time spent (mins)**	**Training Loss**	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
1º	3	Adam	47,01	0.1899	0.9263	0.1519	0.9385
2º	3	SGD	52,37	0.1779	0.9305	0.1529	0.9422
3º	4	SGD	54,43	0.2196	0.9129	0.1737	0.9307
4º	5	SGD	55,50	0.2194	0.9121	0.1747	0.9306
5º	**10**	**SGD**	**58,46**	**0.1651**	**0.9356**	**0.1441**	**0.9461**
6º	11	SGD	65,23	0.1648	0.9366	0.1410	0.9436

Since the MobileNetV2 architecture is based on the MobileNetV1 architecture and in the MobileNetV1 trainings we did not get good results using Adam's optimizer, we knew that we were not going to have good results training MobileNetV2 with that optimizer either. Just in case, we did a test by unfreezing few layers. We can see in the first row of the Table 4.5 that the accuracy achieved is quite high but, if we look at the graphs of the loss and accuracy functions, Figure 4.7 we can see that already with only 3 layers defrosted, the functions do not follow a constant direction and start to oscillate between several values. Then, as we increase the number of trainable layers, it will happen as in the graphs in Figure 4.4. This means that the weights are not being updated correctly with this optimizer and, therefore, the model is not learning as efficiently as possible.

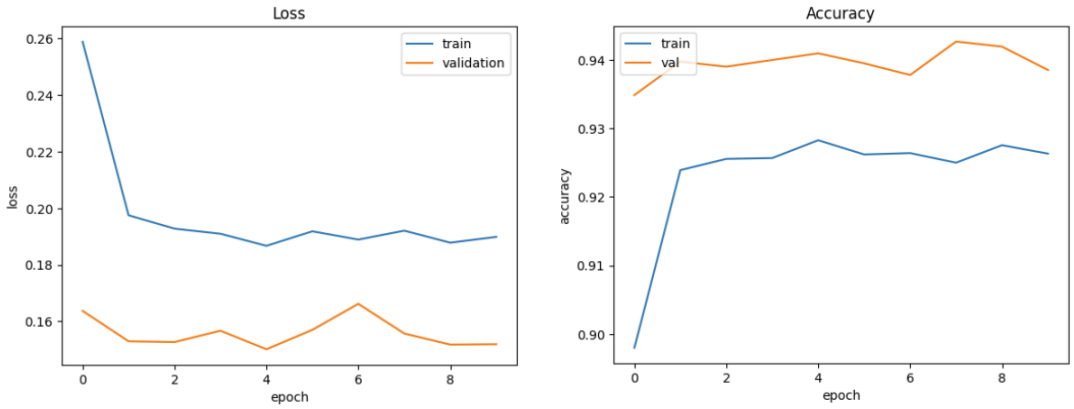


Figure 4.7: Graphs obtained from the 1º test of MobileNetV2 4.5

As we can see in the Table 4.5 we obtained the best results applying 10 trainable layers and with the SGD optimizer, specifically, 93.56% of training accuracy and 94.61% of validation accuracy. That is why we stopped testing after 11 trainable layers. Figure 4.8 shows that the loss and accuracy functions follow a constant direction, unlike Adam's optimizer. Moreover, since the training loss function is always above but close to the validation function, this means that the model is not over-learning and manages to generalize well.

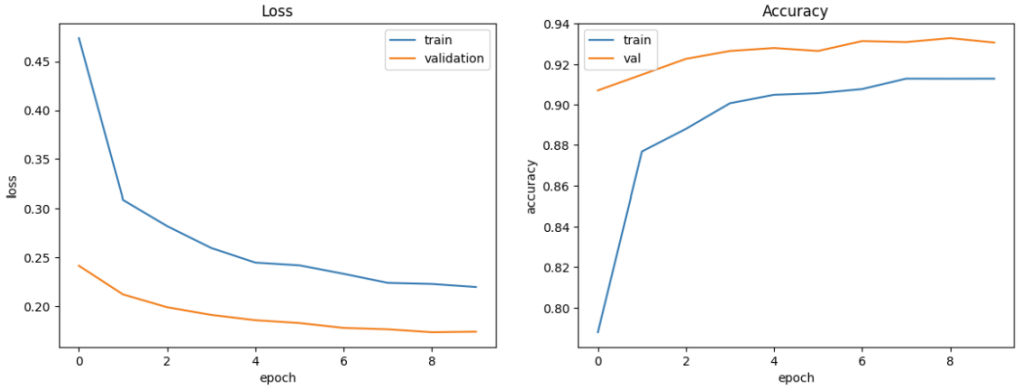


Figure 4.8: Graphs obtained from the 5º test of MobileNetV2 4.5

In addition, we ran the tests with all the trained models and extracted each of the confusion

matrices. As expected, we obtained the best results with 10 trainable layers. You can see all the confusion matrices in Table 4.6 and see that the best solution, again, is the one in row 5.

Table 4.6: MobileNetV2 Confusion Matrix Summary

	TN	FN	TP	FP
1º	1895	102	1896	105
2º	1907	111	1888	92
3º	1916	166	1831	85
4º	1894	154	1843	107
5º	**1944**	**125**	**1872**	**57**
6º	1916	166	1831	85

Then, as in MobileNetV1, we applied the six techniques of DA (flip, rotation, contrast, Gaussian noise, brightness and zoom). For this training we used the best results obtained from the previous tests, that is: 10 trainable layers, SGD optimizer and initial learning rate equal to 0.001. The final metrics obtained were the ones on Tables 4.7 and 4.8 and loss and accuracy functions were Figure 4.9. This values will be compared in the following pages with the results of the other four models trained with DA to choose the one which performs best.

Table 4.7: Results for DA applied to MobileNetV2

Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.3657	0.8391	0.5842	0.7414

Table 4.8: Confusion Matrix for DA applied to MobileNetV2

TN	FN	TP	FP
1953	979	1020	46

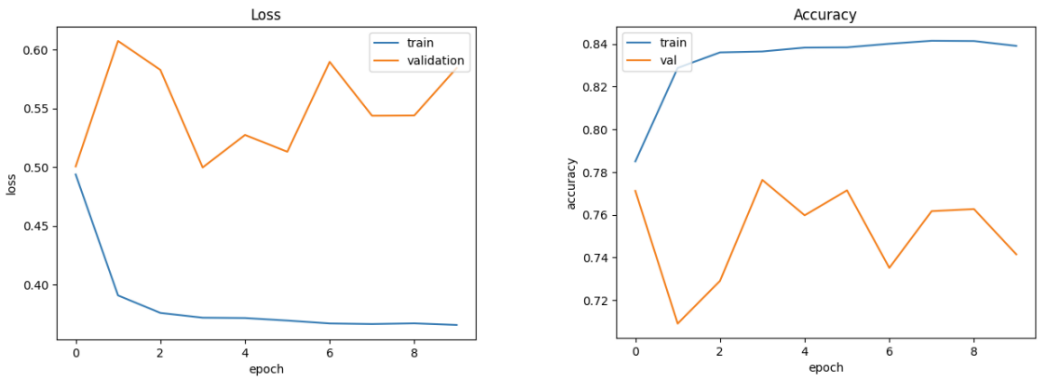


Figure 4.9: Graphs obtained from the DA test of MobileNetV2 4.7

4.2.3 MobileNetV3-Small trained on 2 Classes

For the same reason as in MobileNetV2, as this architecture is also based on that of MobileNetV1 (and MobileNetV2), we only did one test with the Adam optimizer to see if its ineffectiveness was still met. In addition, as in the last two sections, we made modifications on the number of trainable layers, the optimizer, and used a learning rate equal to 0.001. The results obtained in each training were as shown in the Table 4.9.

Table 4.9: Training results on MobileNetV3-Small with different parameters

	Nº Trainable Layers	**Optimizer**	**Time spent (mins)**	**Training Loss**	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
1º	8	Adam	55,51	0.1312	0.9500	0.1646	0.9419
2º	5	SGD	52,12	0.1508	0.9449	0.1542	0.9429
3º	8	SGD	54,32	0.1244	0.9538	0.1328	0.9515
4º	10	SGD	55,45	0.1238	0.9532	0.1352	0.9493
5º	11	SGD	55,40	0.1181	0.9552	0.1368	0.9463
6º	12	SGD	60,12	0.1250	0.9533	0.1352	0.9515

Again we can see in the Table 4.9 that using Adam's optimizer reaches a training accuracy of 95% and a validation accuracy of 94.19% which are very good results but, if we check again the graphs with the loss and accuracy functions, we can see that there is even a moment in which the training accuracy function exceeds the validation accuracy function, which means that the model starts to over learn and does not generalize as well as it should. We can see these plots in Figure 4.10.

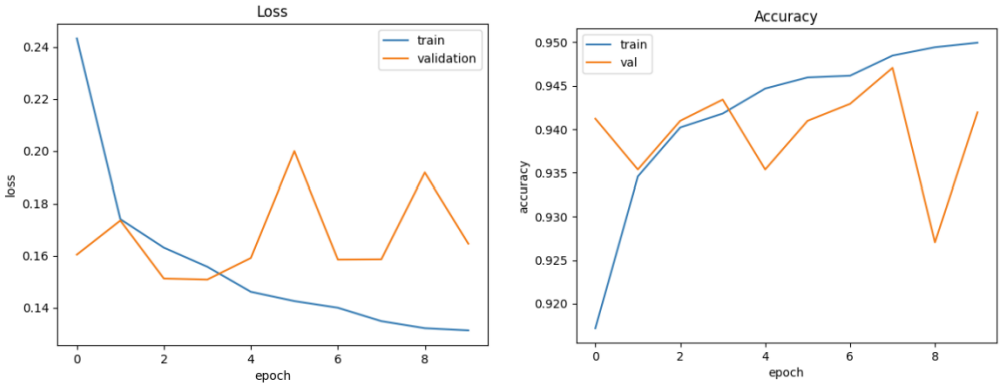


Figure 4.10: Graphs obtained from the 1º test of MobileNetV3-Small 4.9

We can see that in all the cases of Table 4.9 the training accuracy exceeds the validation accuracy after having trained them for 10 epochs. But, looking at the loss and accuracy plots, we saw that during the first epochs the training accuracy was below the validation accuracy and the training loss was above the validation loss in all the training trials, i.e., in the first epochs the model does not over-learn and manages to generalize better. That is why we analyzed all the output plots and saw that the best results were obtained by

training the model with 10 trainable layers and during 4 epochs, specifically, 14.66% training loss, 94.51% training accuracy, 14.51% validation loss and 94.56% validation accuracy were achieved. In particular, their output graphs are shown in Figure 4.11, where it can be seen that from the 4º epoch on wards the functions intersect and, the model begins to be less effective.

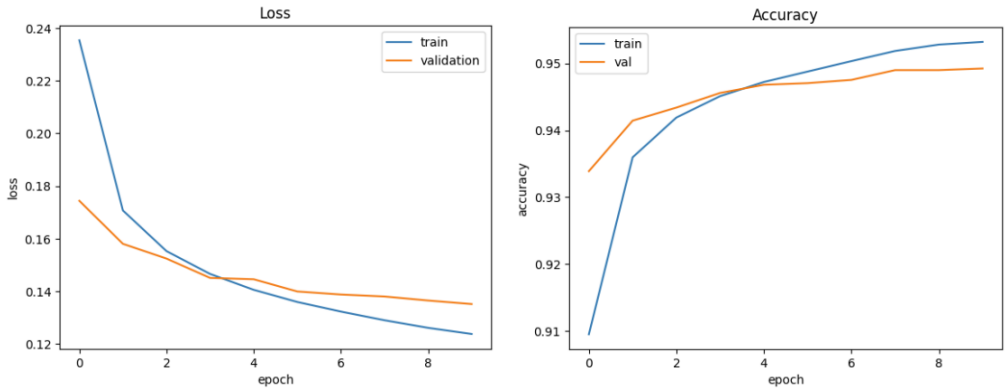


Figure 4.11: Graphs obtained from the 4º test of MobileNetV3-Small 4.9

In Table 4.10 we have the confusion matrices of each of the tests done over the models trained for 4 epochs instead of 10. It is clear that the 4º, which is the architecture trained over 4 epochs with 10 trainable layers, achieves the best results.

Table 4.10: MobileNetV3-Small Confusion Matrix Summary

	TN	**FN**	**TP**	**FP**
1º	1936	166	1833	63
2º	1911	147	1850	90
3º	1920	123	1876	79
4º	**1946**	**81**	**1908**	**63**
5º	1904	99	1898	97
6º	1916	117	1880	85

After these tests, we move on to apply DA to the training images. Again we applied all the techniques explained above (flip, rotation, contrast, Gaussian noise, brightness and zoom). We can see in the Table 4.11 that a training accuracy of 94.11% and a validation accuracy of 94.44% were achieved. In addition, we trained over 10 epochs and the accuracy and loss functions remained more or less constant, as can be seen in the Figure 4.12 and, moreover, without signs of overfitting because the validation dataset was better predicted than the training dataset. Finally, we ran the test over the test dataset and obtained the confusion matrix shown in the Table 4.12. As in MobileNetV1 and MobileNetV2, this DA results will be compared with the rest at the end.

Table 4.11: Results for DA applied to MobileNetV3-Small

Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.1528	0.9411	0.1458	0.9444

Table 4.12: Confusion Matrix for DA applied to MobileNetV3-Small

TN	FN	TP	FP
1906	126	1873	93

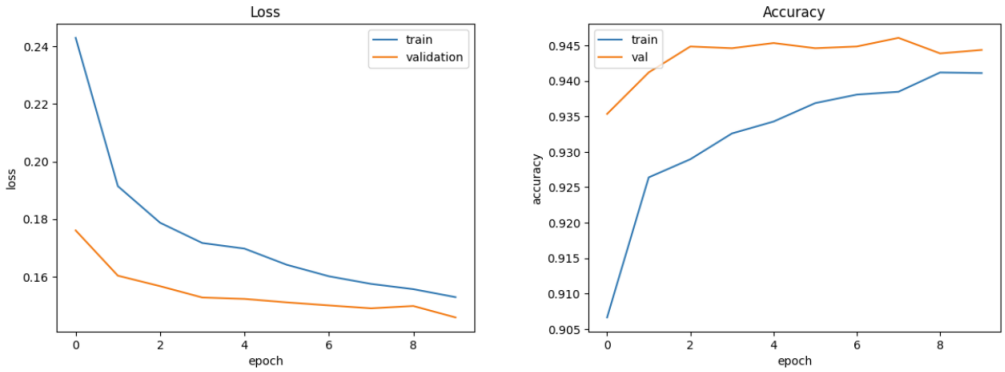


Figure 4.12: Graphs obtained from the DA test of MobileNetV3-Small 4.11

4.2.4 MobileNetV3-Large trained on 2 Classes

Since the architecture of this model is pretty similar to the one of MobileNetV3-Small and we obtained good results with that model, we replicated what we did to the layers of the Small model, which was, changing the last layer (Dense layer) for the model to classify between two different classes. For this we use the following code, which adds only two neurons to the Dense layer.

```
#x = all the layers except the last one
x = mobileNetV3Large.layers[-1].output
output = Dense(units=2, activation='softmax')(x)
mobileNetV3Large = Model(inputs=mobileV3Large.input,
                         outputs=output) #final model
```

Listing 4.7: Modify last MobileNetV3-Large layer

We did the trainings on MobileNetV3-Large showed in Table 4.13 where we can see that the model was only trained four times with the SGD optimizer. This is because, as we increased the number of trainable layers, the loss and accuracy functions of the training process were starting to get unpredictable, as is shown on Figure 4.13, and the model started to have problems generalizing. We also tested this model trained over different number of trainable layers and we obtained the confusion matrices of Table 4.14. Again, we confirm that the Adam optimizer works worse than the SGD optimizer just by looking

at its graphics, Figure 4.14.

Table 4.13: Training results on MobileNetV3-Large with different parameters

	Nº Trainable Layers	Optimizer	Time spent (mins)	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
1º	8	Adam	54,51	0.2648	0.9338	0.2098	0.9427
2º	5	SGD	53,53	0.1303	0.9506	0.1214	0.9563
3º	**8**	**SGD**	**54,24**	**0.1282**	**0.9512**	**0.1166**	**0.9585**
4º	10	SGD	55,45	0.1275	0.9516	0.1176	0.9576
5º	11	SGD	55,50	0.0406	0.9853	0.1133	0.9615

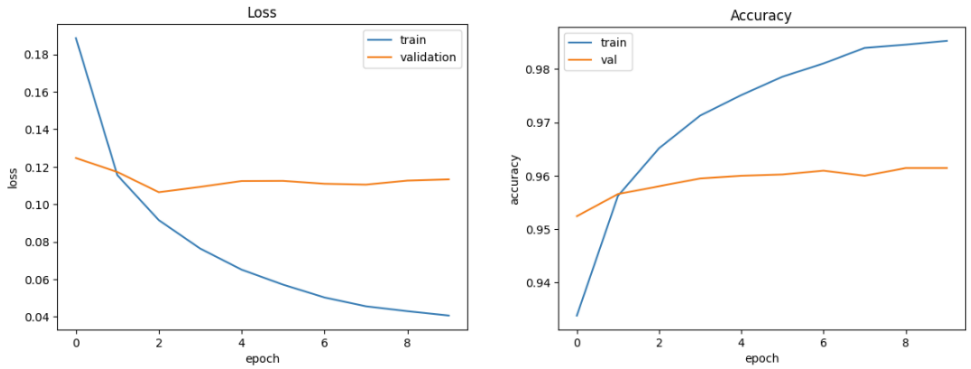


Figure 4.13: Graphs obtained from the 5º test of MobileNetV3-Large 4.13

Table 4.14: MobileNetV3-Large Confusion Matrix Summary

	TN	FN	TP	FP
1º	1900	139	1860	99
2º	1920	107	1892	79
3º	**1921**	**101**	**1896**	**80**
4º	1912	89	1908	89
5º	1938	89	1908	63

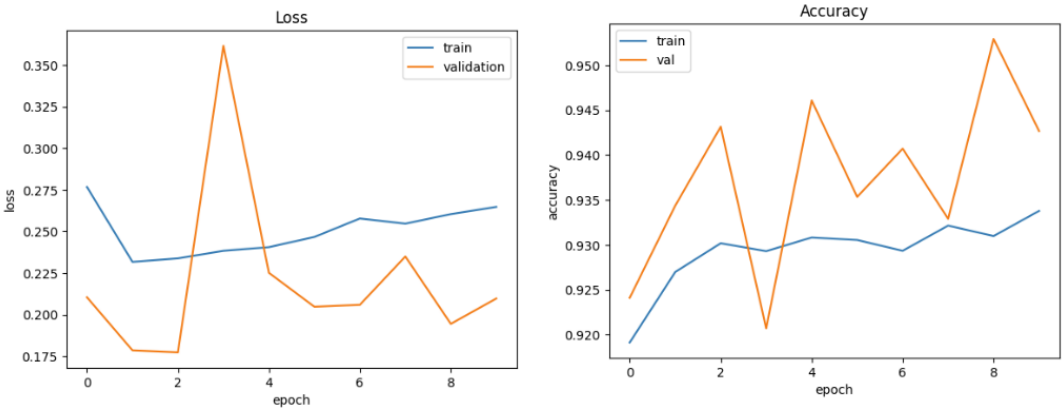


Figure 4.14: Graphs obtained from the 1º test of MobileNetV3-Large 4.13

In both Tables, 4.13 and 4.14, the 5º training with 11 trainable layers obtained best results but, if we observe the graphics on Figure 4.15, we can see that training the model with 8 trainable layers obtain graphics which indicate that the model is performing correctly. Specifically, the lost function is continuously heading towards the minimum value, which is the best performance that the model can do as it indicates that the w and b parameters are being updated correctly and that, therefore, the model is learning well. So, we choose as the best option the model trained over 8 trainable layers.

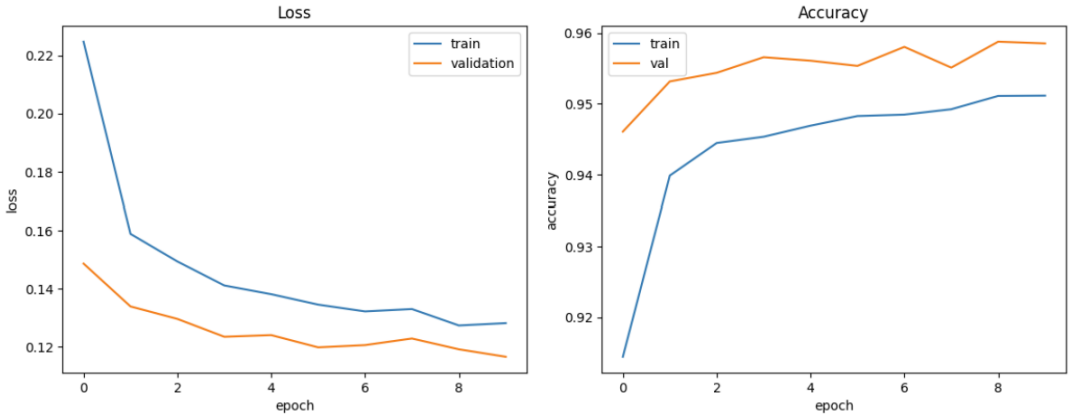


Figure 4.15: Graphs obtained from the 3º test of MobileNetV3-Large 4.13

The last test that was done on this model was applying the 6 DA techniques applied to the other three models (flip, rotation, contrast, Gaussian noise, brightness and zoom). Data augmentation was applied to the training images, with 8 trainable layers, with the SGD optimizer and with an initial learning rate at 0.001, as usual. The results obtained with this training were the ones shown on Tables 4.15 and 4.16 and, on Figure 4.16.

Table 4.15: Results for DA applied to MobileNetV3-Large

Training Loss	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
0.1868	0.9264	0.1953	0.9171

Table 4.16: Confusion Matrix for DA applied to MobileNetV3-Large

TN	**FN**	**TP**	**FP**
1763	76	1923	236

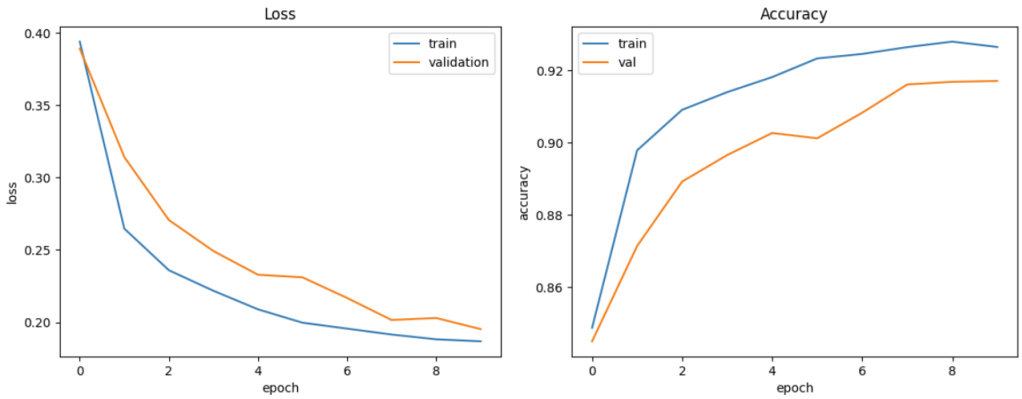


Figure 4.16: Graphs obtained from the DA test of MobileNetV3-Large 4.15

4.2.5 Choosing the best model

In order to choose the best model we base our choice on the results obtained with each of the four models by adding DA to them. We do it this way since we want to apply this mechanism to our final model so that it gets to learn features beyond the initial ones brought by the input images.

So, if we look at the output metrics tables for each of the four tests we can see the following:

- **MobileNetV1**: Table 4.3 shows that only 61.10% of training accuracy and 62.89% of validation accuracy are achieved. In Table 4.4 we can see that there are 984 false negatives and 458 false negatives, i.e., it fails with the predictions of about one third of the total test images. Finally, in Figure 4.6 we can see that the model starts with a very low accuracy in both training and validation sets and only manages to increase it by about 13% and, consequently, it starts with a very high loss rate in both data sets and also fails to reduce it by more than 13%. That is why we think that this model would not be very advisable to choose it to apply the different DA techniques, which is our ultimate goal. In addition, it can be seen that the training loss function is always below the loss function of the validation set, which means that it is behaving worse with the data it does not know and, in the future, this will mean that it will not be able to generalize correctly.

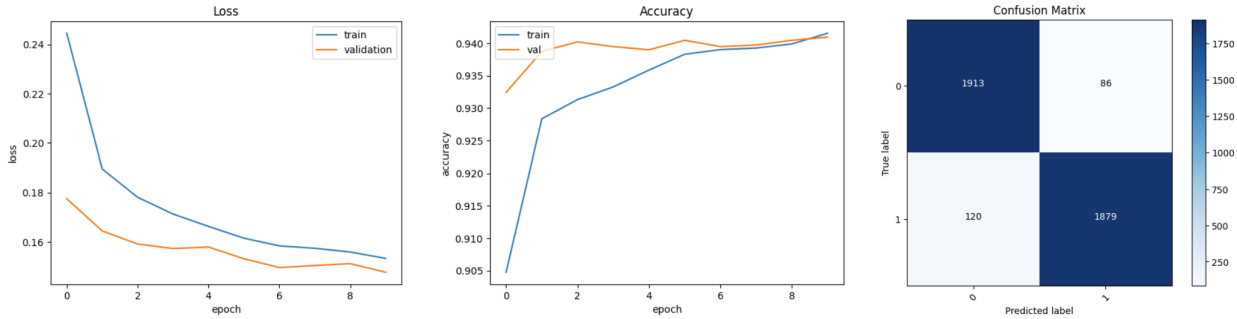
- **MobileNetV2**: in this case we get higher values of accuracy and lower values of loss, specifically as we can see in the Table 4.7, we reach 83.91% of training accuracy and 74.14% of validation accuracy. The problem again is that as we can see in these values and in Figure 4.8 the training metrics are better than the validation metrics, so again, this indicates that the model will find it difficult to generalize in the future. Also, if we check the confusion matrix in Table 4.9, we can see that there are few false positives but there are 979 false negatives, which is quite a high value over the

1,997 images with explicit content that we have for testing. This means that the MobileNetV2 architecture is failing to detect explicit content in the images when using DA.

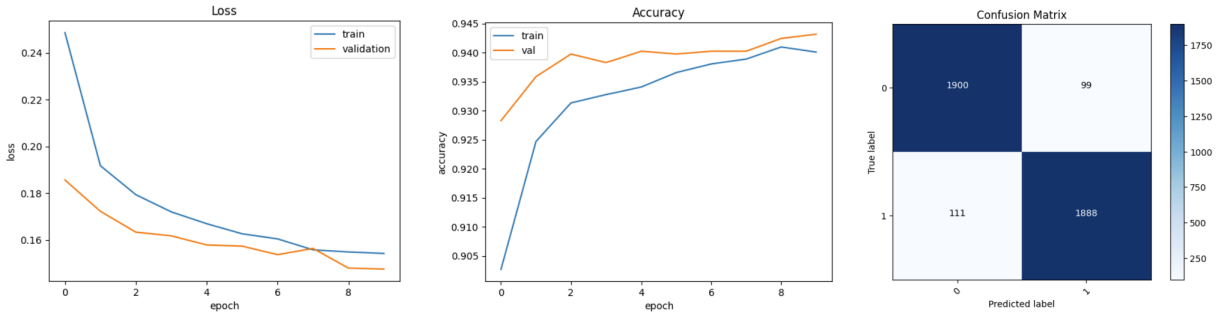
- **MobileNetV3-Small**: this architecture achieves quite good metrics by applying DA to the training images. Specifically, the Table 4.11 shows that the training loss is only 15.28%, the validation loss is 14.58%, the training accuracy is 94.11% and the validation accuracy is 94.44%. Furthermore, if we look at the confusion matrix Table 4.12, we can see that it only misclassifies 126 images with explicit content and 93 normal images, i.e., it manages to classify 3,779 images out of 3,999 total images. Moreover, if we look at the evolution of the cost and accuracy functions, in Figure 4.12, we can see that they are heading towards the minimum point (in the case of the loss function) and the maximum point (in the case of the accuracy function) and that, all the time, the validation dataset achieves better metrics than the test dataset. This means that the model will generalize better with unseen input images.

- **MobileNetV3-Large**: last but not least, we have the test on this model with DA, which like the MobileNetV3-Small gets good metrics (which makes sense because this is an extension of the small one). We can see in the Table 4.15 that it has only 18.68% training loss, 19.53% validation loss and achieves 92.64% training accuracy and 91.71% validation accuracy. Then, in the confusion matrix Table 4.16 we can see that it fails 76 images with explicit content, i.e. there are 76 false negatives and it has 239 false positives. Moreover, the graphs of the loss and accuracy functions also behave in a constant way and go towards the minimum point of the conste function and the maximum point of the accuracy function. But, although these metrics are quite good, those of MobileNetV3-Small are better and on top of that, as shown in Table 4.16, it has 3 million fewer parameters. This means that it will be easier to import on a cell phone and faster and more efficient.

Ultimately, the model chosen to do the rest of the tests with different larger datasets and testing different DA combinations will be the MobileNetV3-Small, due to the good metrics obtained in the training. In addition, when it came out as the best solution, we did two more training tests with this model, applying the same parameters (SGD optimizer and 10 trainable layers) and the six DA techniques (flip, rotation, contrast, Gaussian noise, brightness and zoom) to check that we were indeed still obtaining good results, and so it happened. We can see the metrics obtained in Figure 4.17.



(a) Second training execution of MobileNetV3-Small with DA



(b) Third training execution of MobileNetV3-Small with DA

Figure 4.17: Proof that MobileNetV3-Small with DA performs extremely well

4.3 Data Augmentation on MobileNetV3-Small

After choosing the MobileNetV3-Small as the best model among the four architectures we tested, we are going to test the different DA techniques applied on this model to check which combinations will help the most to the model to improve the detection of sexual content in images. We have:

- (A) Flip
- (B) Rotation
- (C) Contrast
- (D) Gaussian Noise
- (E) Brightness
- (F) Zoom

The best results are in bold and underlined in yellow in Table 4.17. The best option to improve the model is to add only the flip technique or, if we want to add several DA techniques, the best option would be contrast plus Gaussian noise plus brightness. As we

Table 4.17: Metrics of Combinations of Data Augmentation

Data Augmentation	Time spent (mins)	Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
Without Data Augmentation	52,25	0.1244	0.9538	0.1328	0.9515
A	**22,5**	**0.1319**	**0.9510**	**0.1211**	**0.9544**
B	33,30	0.1367	0.9462	0.1374	0.9495
C	25,35	0.1233	0.9543	0.1345s	0.9495
D	31,26	0.1237	0.9531	0.1349	0.9480
E	25,25	0.1270	0.9516	0.1380	0.9480
F	29,06	0.1362	0.9478	0.1361	0.9471
A + B + F	35,05	0.1496	0.9408	0.1498	0.9436
C + D + E	**31,44**	**0.1362**	**0.9497**	**0.1262**	**0.9528**
A + C + D + E	33,58	0.1273	0.9515	0.1365	0.9493
A + B + C + D + E + F	44,36	0.1543	0.9401	0.1476	0.9432

can see the other techniques applied on the model do not give bad results either, so we can further confirm that this architecture works well by modifying the input data with DA.

4.4 Training MobileNetV3-Small on 5 classes

We noticed that one of the biggest problems of the model when classifying images was that it was very confused with images with sexual content of cartoons, since it classified them as normal, and also with images of people in swimsuits, that is, where many pixels appeared with the color of the skin, since it classified them as images with sexual content. That is why we came up with the idea of training the model on datasets of 5 different classes which were “drawing” (which are basically anime images), “hentai”, “neutral”, “sexual content” and “sexy” (this class includes images with many skin color pixels but without explicit content). In this way we would get the model to differentiate the images that gave more problems (the hentai and the sexy ones). For these tests we used 3 different datasets with those 5 classes. Before explaining the characteristics of each dataset we have to point out that they were all tested on the same dataset. This had 9,991 images of which 2,031 were drawings, 1,963 were hentai, 2,029 were normal, 2,280 had sexual content and 1,688 were sexy.

1. Dataset 1: it had 66,763 images where 80% of them were used to train and the other

20% was used for the validation phase. Specifically, it contained 18,030 drawing images, 2,878 hentai images, 33,713 neutral images, 4,913 sexual content images and 7,229 sexy images. So, this dataset was not really balanced because there were too much neutral, drawing and sexy images compared to the other s classes.

2. Dataset 2: because of the large amount of neutral and drawing images, we decided to eliminate and balance the quantity of each of the classes. So for this dataset we had 2,878 drawing images, 2,878 hentai images, 4,980 neutral images, 4,913 images with sexual content and 4,380 sexy images.

3. Dataset 3: after making some training trials with the second dataset we noticed that the model needed more information of each of the classes so we decided to search for more images in order to feed it. So now, in this third dataset we had 8,454 drawing images, 8,464 hentai images, 8,473 neutral images, 8,474 images with sexual content and 8,474 sexy images. This images were obtained from the Dataset 1 and filled up with new images extracted from different sites.

Once we had the three datasets we went on to do the respective training by applying DA to them. Based on the results obtained in the previous section 4.3, we made two training tests with each dataset, one applying only the flip technique and the other applying contrast, Gussian noise and brightness. For the first dataset the best results were obtained by applying only the flip technique to the images. For the second dataset the best results were obtained with contrast, Gaussian noise and brightness. And, for the third dataset, the best option was to apply only flip to the training images. After the training with MobileNetV3-Small on these three datasets were done, the necessary tests were performed on each of them. To detect an image as mispredicted, we took into account that an image is mispredicted if:

- A drawing image is classified as hentai, sexual content or sexy.
- A hentai image is classified as drawing, neutral or sexy.
- A neutral image is classified as hentai, sexual content or sexy.
- An image with sexual content is classified as drawing, neutral or sexy.
- A sexy image is classified as anything that is not sexy.

This is because the main objective of the algorithm is to detect sexual content so, if any drawing image detects it as neutral or any neutral image detects it as drawing we do not classify it as a failure because these images are harmless and the model is classifying them as such. The same happens with hentai images or images with sexual content, if it classifies them as one class or another nothing happens because the algorithm is detecting

that there is danger in that image and that is our ultimate goal. And the sexy images we count them as an error if it does not detect them as such because, if we are using this model to detect inappropriate content on a minor's mobile, it should trigger some kind of warning that the image might not be suitable for him and someone or something should check it. To do this, the following code was used:

```
predictions = mobileNetV3Small.predict(x=test_batches, verbose=1)
preds = predictions.argmax(axis=1)
wrong_predictions = []
i = 0
for fp in test_batches.filepaths:
    if ('drawing' in fp) and (preds[i] == 1 or preds[i] == 3 or
       preds[i] == 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    elif ('hentai' in fp) and (preds[i] == 0 or preds[i] == 2 or
       preds[i] == 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    elif ('neutral' in fp) and (preds[i] == 1 or preds[i] == 3 or
       preds[i] == 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    elif ('prn' in fp) and (preds[i] == 0 or preds[i] == 2 or
       preds[i] == 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    elif ('sexy' in fp) and (preds[i] != 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    i = i+1
```

Listing 4.8: Find the wrong predicted images

So, in the previous code 0 corresponds to "drawing", 1 corresponds to "hentai", 2 corresponds to "neutral", 3 corresponds to "sexual content" and 4 corresponds to "sexy". We are saving in the array "wrong_predictions" some interesting information of each wrongly predicted image for future analysis (seen in chapter 5). And to know exactly how many images are wrongly predicted we just print the length of the "wrong_predictions" array.

For each of the trainings we obtained the function matrix to see how well the model performed on each class specifically. For the first dataset 8149 images were correctly predicted, for the second datset 8348 images were well predicted and for the third dataset, the model achieved to successfully predict 8425 images. Each of these confusion matrices are shown in Figures 4.18, 4.19 and 4.20.

With this results we can conclude that balancing the dataset and increasing the number of images to train the model is really important for the model to learn valuable information. Then, the best results were obtained with the biggest and more balanced datasets of the the three, the Dataset 3. So this model trained over 5 classes will be compared to the previous one trained over 2 classes in the following chapter.

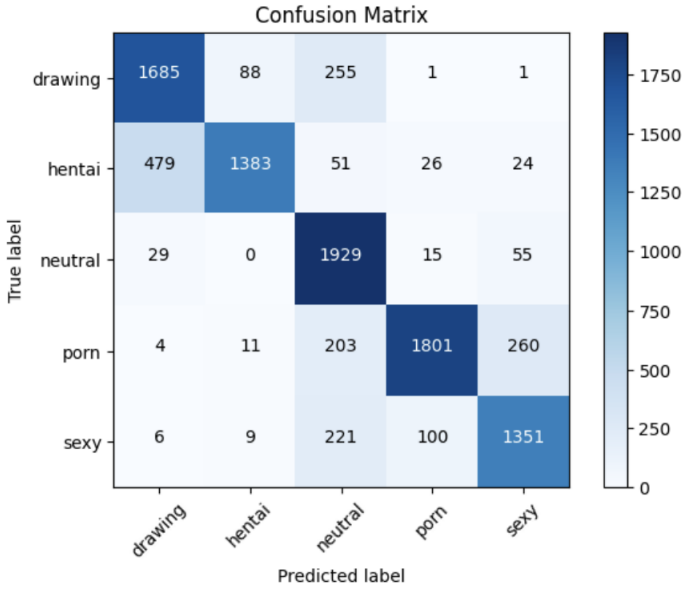


Figure 4.18: Confusion Matrix training MobileNetV3-Small over Dataset 1

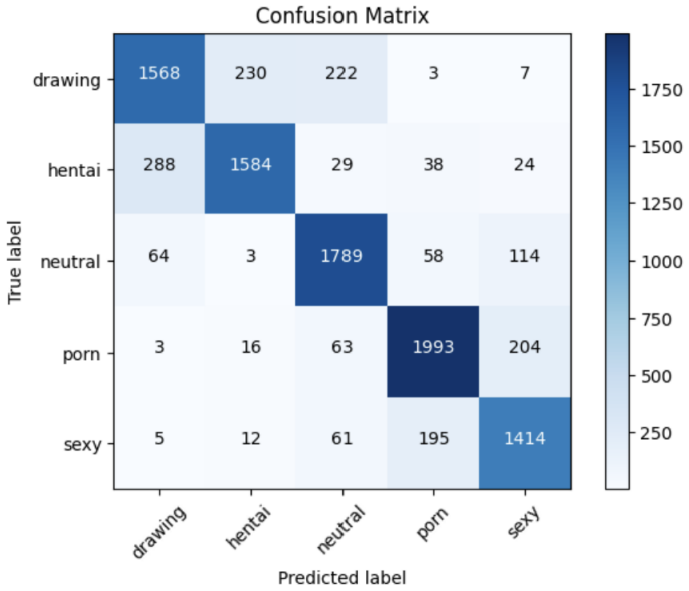


Figure 4.19: Confusion Matrix training MobileNetV3-Small over Dataset 2

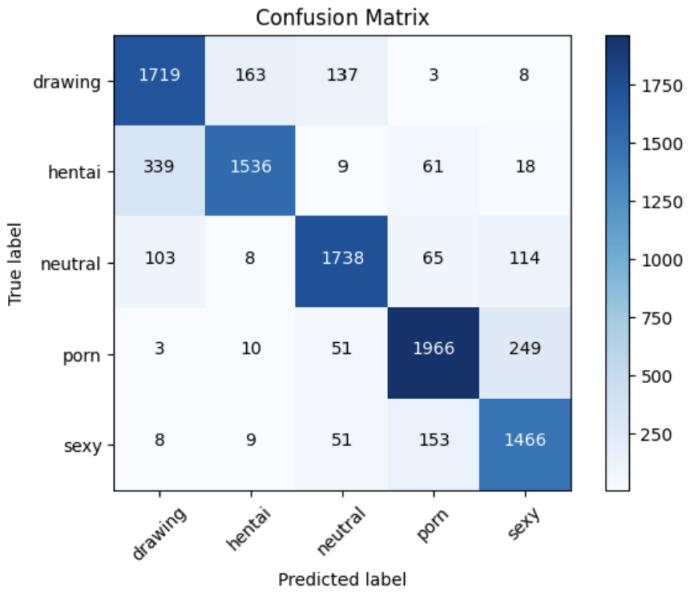


Figure 4.20: Confusion Matrix training MobileNetV3-Small over Dataset 3

Chapter 5

Experiments and Results

In this chapter we explain some experiments and results we did to analyze the performance of our models. It is divided in the following way, in Section 5.1 we explain some tests done with 3 different datasets, in Section 5.2 we explain a tool used to accelerate the image analysis process and, finally, in Section 5.3 we show and explain important heat maps that helps the model to be better understood or interpreted.

5.1 Tests with different datasets

To choose the best final model we will compare four models. The MobileNetV3-Small trained on two classes and with the DA techniques of contrast, Gaussian noise and brightness (which, as we saw in the table 4.17, is the best combination) and, MobileNetV3-Small trained on the 3 datasets of 5 classes with the best DA results obtained in each of them. To compare the performance of each of the models what we did was to make the predictions (test the models) on three different datasets:

- **Test Dataset with 5 classes**: this dataset is the one used to make the tests in section 4.4. It has 9,991 images of which 2,031 are drawings, 1,933 are hentai, 2,029 are normal, 2,280 have sexual content and 1,688 are sexy. In order to test this dataset with trained models with 2 classes we will say that, if the model classifies an image with real drawing or neutral label as neutral, then we will assume that the model is doing the predictions correctly and, if an image has real hentai, sexual or sexy content and the model classifies it as sexual content then we will say that the model is performing well. In any other case, we will assume that the model has failed. We include sexy in this last group of sexual content because, in the future, if this model is used to detect content not suitable for a minor, if a "sexy" image arrives, it would be convenient to send a warning so that the image can be reviewed by someone who decides what to do with it.

- **Test Dataset with 2 classes**: this dataset is the one used to make the tests in section 4.2. It has 3,999 images of which 2,002 are of the normal class and 1,997 have sexual content. In order to test with this dataset on the models trained with 5 classes we will say that, if an image whose real label is neutral and the model classifies it as drawing or normal, then that clasification will be correct and, if an image whose real label is sexual content and the model classifies it as hentai, sexual content or sexy, then the classification will be correct. In any other case, the model will fail. Again, we assume that a "sexy" image is sexual content for the same reasons as in the previous paragraph.

- **New Test Dataset with 2 classes**: this new dataset consists of 40,000 images, specifically, 20,000 normal and 20,000 sexual ones. The particularity of these images is that they are frames extracted from videos and, therefore, we will be able to see how our models behave to detect sexual content in a video from their frames. For the three models trained on 5 classes, the detection of misclassified images will be done as in the previous dataset.

The number of images wrongly predicted by each model tested by each dataset is shown in Table 5.1. Evidently, the model trained on 2 classes will perform better than the rest of the models in the test on the 2-class dataset (since that dataset was extracted from the same site as the training dataset). The same happens with the test on the 5-class dataset, the model trained on the 5 classes with the different datasets will perform better since those images were extracted from the same site as the training dataset. Even so, we can see that the four models behave quite well with respect to both datasets, highlighting that the one that performs better among the 3 models trained on 5 classes is the one trained with dataset 3 (which makes sense because it is the one that is more balanced and the one that has more images, therefore, the one that has learned more features to identify the images).

So, to find out which is the best model among the four, we will look at the last row of the Table 5.1. We can see that, by quite a difference, the best performers are the model trained on 2 classes and the model trained on 5 classes with dataset 3. In addition, the worst performer is the 5-class model trained on dataset 1, this is because the dataset of that training was not at all balanced and, therefore, the model has not managed to learn as many features from some classes as from others and, that is why, it is failing that much. Between the two models, the one with the best results was the model trained on 2 classes. That is why we will choose this model for the next tests. It should be noted that, if in future mobile applications it is required a model that identifies the images between the five classes, the MobileNetV3-Small trained on dataset 3 obtains great results and would be very useful. But, as our goal is to identify images between safe (without sexual content) and unsafe (with sexual content), we will choose the MobileNetV3-Small trained

on 2 classes.

Table 5.1: Number of images wrongly predicted

	MobileNetV3-Small trained over 2 Classes	MobileNetV3-Small trained over 5 Classes with Dataset 1	MobileNetV3-Small trained over 5 Classes with Dataset 2	MobileNetV3-Small trained over 5 Classes with Dataset 3
New Test Dataset with 5 Classes	1685	1290	1233	1192
Test Dataset with 2 Classes	205	369	303	291
New Test Dataset with 2 Classes	7392	14851	9830	7713

5.2 Kangas

Kangas is a Python library that allows us to display a table with the information we want. We use it to display the information of the wrongly or correctly predicted images in order to be more effective in analyzing them and finding any repeating pattern. To use Kangas we implement the following code (the following code is used to display the wrongly predicted images):

```
import kangas as kg
from kangas import DataGrid

dg = DataGrid(name="Images", columns=["Image", "GUESS", "Drawing",
      "Hentai", "Neutral", "Sexual Content", "Sexy", "Path name"])

predictions = mobileNetV3Small.predict(x=test_batches, verbose=1)
preds = predictions.argmax(axis=1)
wrong_predictions = []
i = 0
for fp in test_batches.filepaths:
    if ('drawing' in fp) and (preds[i] == 1 or preds[i] == 3 or
       preds[i] == 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    elif ('hentai' in fp) and (preds[i] == 0 or preds[i] == 2 or
       preds[i] == 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    elif ('neutral' in fp) and (preds[i] == 1 or preds[i] == 3 or
       preds[i] == 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    elif ('prn' in fp) and (preds[i] == 0 or preds[i] == 2 or
       preds[i] == 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    elif ('sexy' in fp) and (preds[i] != 4):
        wrong_predictions.append([fp, preds[i], predictions[i]])
    i = i+1
```

```
27
28 for wp in wrong_predictions:
29 dg.append([kg.datatypes.image.Image(data=wp[0]), str(wp[1]),
30     str(wp[2][0]), str(wp[2][1]),str(wp[2][2]), str(wp[2][3]),
31     str(wp[2][4]), wp[0]])
32
33 dg.show()
```

Listing 5.1: Kangas Implementation

An example of output from the function dg.show() would be the one shown in the Figure 5.1. It shows the drawing images wrongly predicted as hentai using the best model. The five columns with each class indicate the probability that an image belongs to that class.

IMAGE	GUESS	DRAWING	HENTAI	NEUTRAL	SEXUAL CONTENT	SEXY	PATH NAME
	1	0.18316	0.64139	0.15245	0.02085	0.00213	data/test\drawi...
	1	0.41854	0.57138	0.00226	0.00731	0.00047	data/test\drawi...
	1	0.24406	0.73213	0.00527	0.01835	0.00017	data/test\drawi...
	1	0.17857	0.79647	0.0077	0.01067	0.00657	data/test\drawi...
	1	0.21913	0.53897	0.00174	0.24001	0.00012	data/test\drawi...
	1	0.3125	0.67783	0.00027	0.00927	0.0001	data/test\drawi...
	1	0.07517	0.91621	0.00496	0.00344	0.0002	data/test\drawi...

Figure 5.1: Kangas example output

5.3 Interpretability

As explained in Section 2.4, the interpretability of the models in this case will be performed with heat maps. In our case, we will use the Grad-CAM tool since it is the one that gives us the most understandable results. The model that best performed the predictions was MobileNetV3-Small trained on 2 classes (as shown in Table 5.1). So, we will perform the analysis of the heat maps obtained with this models. We will focus on the last test dataset since, being the most different dataset from the training dataset of our best model, we will be able to better analyze how it behaves in front of unseen images.

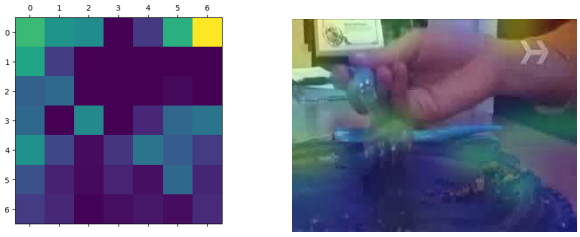
5.3.1 Interpretability on MobileNetV3-Small trained on 2 classes

So, next we are going to show some of the images predicted by the MobileNetV3-Small model trained on 2 classes. We will see the heat maps of images from both the normal

class and the class with sexual content. To select these images, we used the Kangas tool to go faster.

- **Images belonging to the Normal class**:

1. First Image: probabilities obtained are, 86.50% of belonging to Normal class and 13.49% of belonging to Sexual Content class.



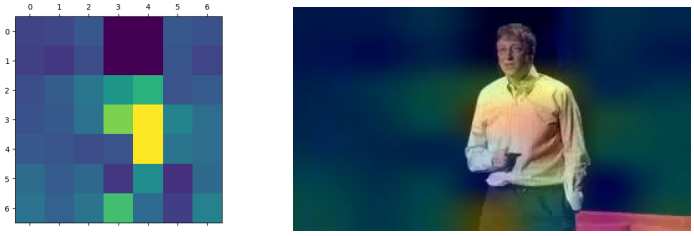
(a) (b)

Figure 5.2: Heat Map for normal class

(a) (b)

Figure 5.3: Heat Map for sexual content class

2. Second Image: probabilities obtained are, 99.35% of belonging to Normal class and 0.64% of belonging to Sexual Content class.



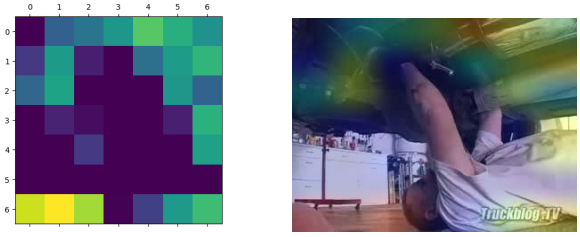
(a) (b)

Figure 5.4: Heat Map for normal class

3. Third Image: probabilities obtained are, 83.78% of belonging to Normal class and 16.21% of belonging to Sexual Content class.

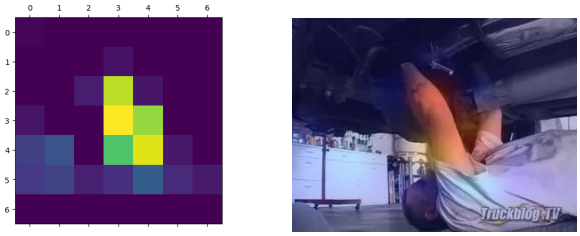
(a) (b)

Figure 5.5: Heat Map for sexual content class



(a) (b)

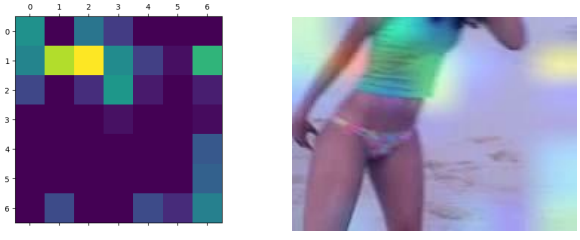
Figure 5.6: Heat Map for normal class



(a) (b)

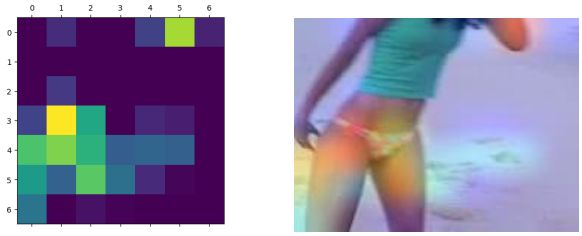
Figure 5.7: Heat Map for sexual content class

4. Fourth Image: probabilities obtained are, 92.82% of belonging to Normal class and 7.72% of belonging to Sexual Content class.



(a) (b)

Figure 5.8: Heat Map for normal class



(a) (b)

Figure 5.9: Heat Map for sexual content class

5. Fifth Image: probabilities obtained are, 99.41% of belonging to Normal class and 0.58% of belonging to Sexual Content class.

(a) (b)

Figure 5.10: Heat Map for normal class

(a) (b)

Figure 5.11: Heat Map for sexual content class

We can see in these five images that the model could be confused because there are questionable areas, for example, in the image Figure 5.8 a person with a bikini bottom appears but, the model manages to identify the t-shirt and the background to classify the image as normal. In all five cases it manages to identify the sets of pixels that have non-sexual shapes. Furthermore, we can see that for the sexual content class it identifies the doubtful areas of the image, i.e. in the image Figure 5.3 for example it identifies the folds and the color of the hand, in Figures 5.5 and 5.11 it can be seen that the algorithm identifies the faces (consequently, the pixels with skin color), in Figure 5.7 identifies the arm of the mechanic that, apart from the skin color, could confuse it with a sexual organ and, finally, in Figure 5.9 the pixels

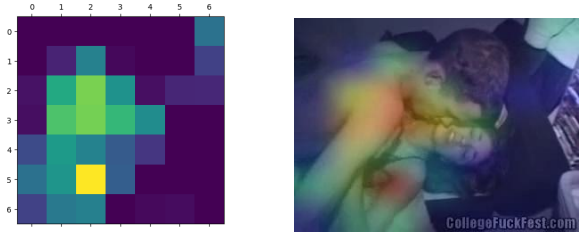
that could denote sexual content are focused on the swimsuit part of the person. In conclusion, based on these heat map examples, we can confirm that the model has learned to identify both classes. In any case, we will now look at some images with sexual content to reaffirm its good performance.

- Images belonging to the Sexual Content class: in this case we will show images that have sexual content but it is not very obvious, thus, demonstrating that the model still behaves correctly even if the sexual content is not so clearly shown (either because it is blurred, because the input image is not of good quality or because there is no sexual organ shown directly).

  1. First Image: probabilities obtained are, 56.82% of belonging to Sexual Content class and 43.27% of belonging to Normal class.

(a) (b)

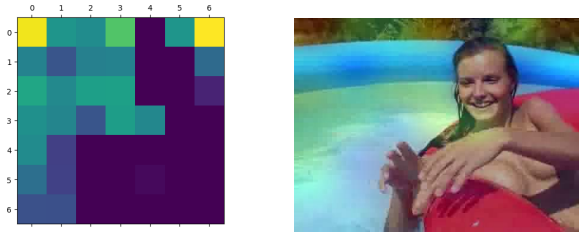
Figure 5.12: Heat Map for normal class



(a) (b)

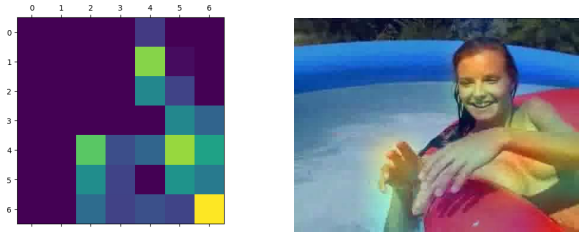
Figure 5.13: Heat Map for sexual content class

  2. Second Image: probabilities obtained are, 57.18% of belonging to Sexual Content class and 42.81% of belonging to Normal class.



(a) (b)

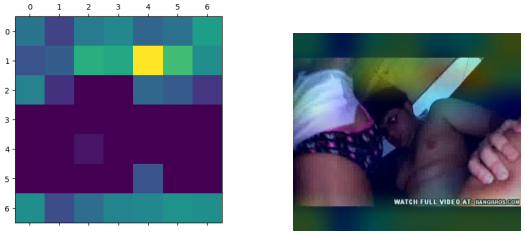
Figure 5.14: Heat Map for normal class



(a) (b)

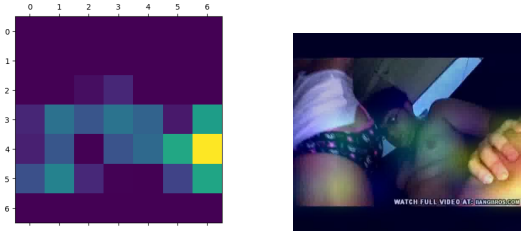
Figure 5.15: Heat Map for sexual content class

3. Third Image: probabilities obtained are, 63.13% of belonging to Sexual Content class and 36.86% of belonging to Normal class.



(a) (b)

Figure 5.16: Heat Map for normal class



(a) (b)

Figure 5.17: Heat Map for sexual content class

Being frames extracted from videos, it is normal that there are some of them in which there is something not as explicit but, they are still images with sexual content. That

is why the percentages of sexual content obtained for the three images are not very high, although they are still higher than those of the normal class. The good thing is that, thanks to the heat maps we can see that the model is looking at the indicated set of pixels to classify the image as not safe. Also, in the three images, we can see that the heat maps obtained for the normal class are fixated on the areas where there is nothing explicit. So, we can say that the model is working correctly on both detecting and classifying both classes.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this project we have tested different DL architectures for explicit content detection in images. For this purpose, we had to perform a large search of image sets, both for model retraining and for final testing. Although this process of collecting data sets was long and tedious, thanks to the fact that we managed to collect many images and of different types, we were able to perform good retrainings and tests of the MobileNet architectures. And we have been able to verify that, these AI algorithms are very useful for the detection of patterns in images and that, indeed, they can be effective in controlling the consumption of sexual content in people who may be affected in any way. In addition, we have been able to confirm that these neural networks can not only be used on computers with high storage and processing capacities but, that they are also very applicable to cell phones, which have many more restrictions, since the models we have retrained have far fewer parameters than others and are easily portable to cell phones.

Additionally, we have been able to see, thanks to the interpretability techniques that have been developed previously, that the DL models really do identify repeating patterns and manage to identify them in completely new images that they have never seen before. The fact that algorithms are able to do something like this, without the extra help of a human being, is fascinating and could be applied in many fields in the future. For me personally, this project has helped me to learn much more about this field and, thanks to it, I want to continue researching and learning new techniques and applications of these interesting algorithms.

6.2 Future Work

Some lines of work that could be continued after this project would be the following. Apply the model in smartphones using the TensorFlow Lite library and test it in a mobile application. This application could have several options for action, for example, if the model finds an image with sexual content but with a percentage of less than 70% of possibilities, the image could be reviewed and decide whether the image can be kept or should be deleted. Also, based on that choice, the model could be retrained with this new information and thus, it could be better adjusted to the user's preferences. Another option is that, as we have seen that the model performs very well detecting different patterns in images, this architecture could be retrained to be used in other fields of object detection in images, such as, recognition of medical problems in images, facial recognition or identification of objects for automatic driving vehicles.

Chapter 7

Introducción

Comenzaremos este capítulo con la Sección 7.1 exponiendo las principales razones por las que decidimos investigar y obtener un modelo que identifique contenido sexual en imágenes. A continuación, en la Sección 7.2 explicaremos el contexto en el que se desarrolla este proyecto. Después, dejaremos claros los objetivos de nuestro trabajo en la Sección 7.3. En la Sección 7.4 se expone el plan de trabajo que hemos seguido durante estos meses y, por último, en la Sección 7.5 se explica cómo se estructura el resto de este proyecto.

7.1 Motivación

La evolución de la tecnología y de Internet ha traído consigo algunos problemas graves, como la facilidad con la que los usuarios pueden acceder y compartir contenidos explícitos de forma rápida y sencilla. Si bien es cierto que toda esta información podría ser utilizada para proporcionar a los jóvenes una educación sexual segura y saludable, en la vida real, el acceso no vigilado y sin control es muy preocupante, ya que los jóvenes no son conscientes del daño que les puede causar acceder a estos contenidos de forma habitual, tal y como demuestran algunas investigaciones como [SLD+16].

Algunos estudios realizados en Florida [OBMR12] revelan que los adolescentes que consumen este tipo de contenidos son más propensos a tener actitudes poco realistas y erróneas respecto a las relaciones de pareja. También explican que muchos de los comportamientos sexuales compulsivos o agresivos están directamente relacionados con el consumo de contenido de carácter sexual. Otro efecto del consumo exagerado de estos contenidos es el hecho de ser más permisivos con actitudes sexuales incorrectas o, estar continuamente pensando en esto y no dar cabida en su mente a generar curiosidad por otros temas más enriquecedores. Además, esto no depende del país, cultura o raza, es algo totalmente globalizado.

Por estos motivos, en los últimos años se han desarrollado diversas estrategias

computacionales para regular el uso de contenidos explícitos en adolescentes o en cualquiera que pueda ser mucho más sensibles a este tipo de vídeos o imágenes. Una de las formas en las que se realiza esta regularización es a través de la detección de imágenes o vídeos de contenido sexual en dispositivos electrónicos y su posterior borrado, generación de avisos, o cualquier acción que queramos implementar tras detectar este contenido. Es por ello que en este proyecto vamos a tratar de generar una herramienta capaz de identificar contenido sexual en archivos multimedia.

7.2 Contexto

El presente Trabajo Fin de Grado se enmarca dentro de un proyecto de investigación titulado Novel Strategies to Fight Child Sexual Exploitation and Human Trafficking Crimes and Protect their Victims – HEROES, aprobado por la Comisión Europea dentro del Programa Marco Horizonte 2020 (convocatoria H2020-SU-SEC-2020) en virtud del acuerdo de subvención número 101021801 y en el que participa como coordinador del proyecto el Grupo GASS de la Universidad Complutense de Madrid (Grupo de Análisis, Seguridad y Sistemas, https://gass.ucm.es, grupo 910623 del catálogo de grupos de investigación reconocidos por la UCM).

Además de la Universidad Complutense de Madrid participan en HEROES 21 entidades ubicadas en 17 países: 11 de países de la UE (Austria, Bélgica, Bulgaria, Francia, Grecia, Irlanda, Letonia, Lituania, Portugal, España, Reino Unido), 1 país asociado (Suiza) y 5 terceros países (Bangladesh, Brasil, Colombia, Perú, Uruguay). Dichas entidades son: University of Kent (Reino Unido), The Free University of Brussels (Bélgica), The French National Research Institute for Digital Science and Technology – INRIA (Francia), Center for Security Studies – KEMEA (Grecia), International Centre for Migration Policy Development – ICMPD (Austria), International Center for Missing and Exploited Children – ICMEC (Suiza), IDENER Research & Development Agrupación de Interés Económico (España), Athena Research Center – ARC (Grecia), Trilateral Research and Consulting (Reino Unido), Centre for Women and Children Studies – CWCS (Bangladesh), Center Against Human Trafficking and Exploitation – KOPZI (Lituania), Portuguese Association for Victim Support – APAV (Portugal), Fundación Renacer (Colombia), The Greek Council for Refugees – GCR (Grecia), Brazilian Association for the Defense of Children of Children and Youth – ASBRAD (Brasil), Hellenic Police (Grecia), Latvia National Police (Letonia), General Directorate for the Fight against Organized Crime (Bulgaria), Dirección General de la Policía – DGP (España), Federal Police (Brasil), Federal Highway Police (Brasil), Secretaría de Inteligencia Estratégica de Estado – Presidencia de la República Oriental del Uruguay (Uruguay)

Tienen más información en:

https://cordis.europa.eu/project/id/101021801

https://heroes-fct.eu

7.3 Objeto de la Investigación

Debido a que hoy en día cualquier persona tiene acceso a dispositivos móviles, el objetivo principal de la investigación es generar un modelo de DL robusto que permita la clasificación de imágenes con contenido sexual que sea capaz de adaptarse a dispositivos con bajos recursos. De esta forma, se reducirá el riesgo de exposición a este tipo de contenidos que puede sufrir un joven. Este tipo de modelos son capaces de aprender por sí mismos características esenciales de las imágenes. Éstas pueden ser generales, como las que percibe el ojo de cualquier ser humano, por ejemplo, objetos, paisajes, determinadas formas o colores, o más específicas, que nosotros mismos no podemos identificar, por ejemplo, tonalidades concretas de colores, patrones diminutos que se repiten en las imágenes (como una mancha en una imagen médica) o texturas (como la piel de los animales). Por estas razones, creemos que es una gran herramienta para identificar contenido sexual en las imágenes.

7.4 Plan de Investigación

A continuación mostraremos las tres fases principales del proyecto. En la Figura 7.1 se muestra los distintos tiempos en los que se realizaron estas fases y, además, cuándo se empezó a redactar esta memoria.

1. **Investigación**: durante los tres primeros meses tuve que familiarizarme con el contexto del proyecto global de investigación y, sobre todo, con el funcionamiento de los algoritmos de DL. Para ello, durante estos tres meses estuve realizando un curso de introducción al ML y leyendo y resumiendo artículos relacionados con el estado del arte del proyecto. Una vez tuve una idea general de lo que se estaba haciendo en el proyecto y de cómo funcionaban los modelos de DL, tuve que empezar a investigar qué arquitecturas de DL podía utilizar para desarrollar mi proyecto ya que tenía que encontrar una que pudiera ser exportada posteriormente a teléfonos móviles y, que pudiera detectar patrones en imágenes. Durante todo este proceso de investigación, mantuvimos reuniones semanales con mis tutores para ver si iba por buen camino con los artículos que estaba leyendo y, sobre todo, para comentar cualquier duda que tuviera sobre alguno de los temas que estaba estudiando.

2. **Desarrollo**: gracias a toda la investigación previa, conseguimos encontrar los modelos de DL que íbamos a utilizar. Empezamos a hacer pequeñas pruebas

con ellos en Google Colaboratory para ver la eficacia de esos modelos y cómo se comportaban. Pero, para empezar a hacer reentrenamiento en nuestros conjuntos de datos más grandes tuvimos que empezar a probarlos en Visual Studio con las librerías TensorFlow y Keras. Además, tuvimos la suerte de poder utilizar una GPU que consiguió reducir el tiempo de reentrenamiento en muchas horas. Durante toda esta fase también realizamos un proceso de entendimiento profundo de las arquitecturas de los 4 modelos ya que era la base del proyecto.

3. **Experimentación**: después de todo el reentrenamiento que hicimos, entendiendo cómo funcionaba cada modelo y obteniendo diferentes métricas con cada uno de ellos, pudimos seleccionar el que mejor se comportaba. Empezamos a probar cómo se comportaba frente a imágenes que el algoritmo nunca había visto para ver si era capaz de generalizar bien y encontrar contenido sexual en ellas o no. Este proceso de experimentación era importante para ver si en el futuro podía ser viable que el modelo se utilizara para diferentes aplicaciones en teléfonos móviles.

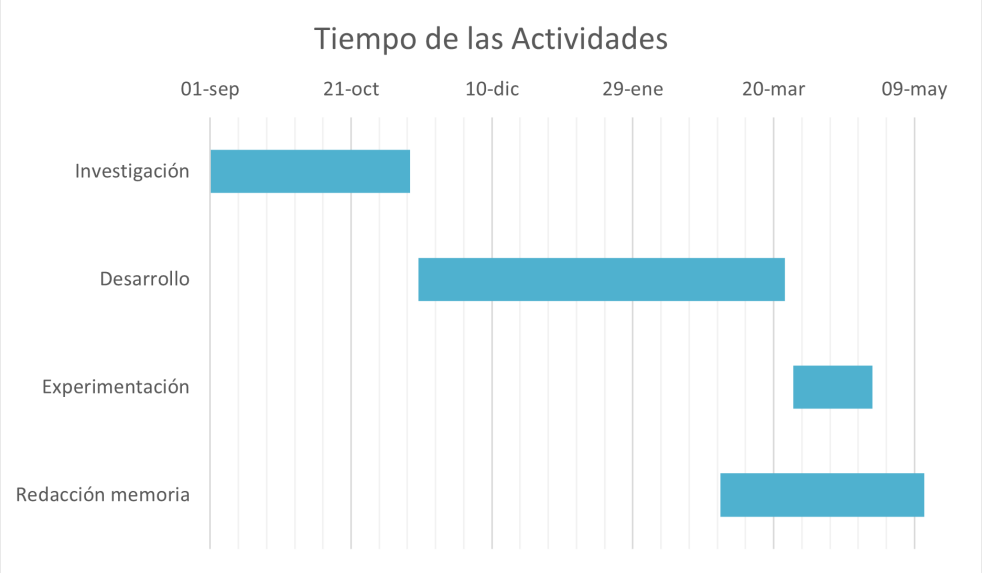


Figure 7.1: Diagrama de Gantt del Plan de Investigación

7.5 Estructura de Trabajo

El proyecto está dividido en 6 capítulos (el primero siendo éste), de la siguiente manera:

En el Capítulo 2 se explican algunos de los principales conceptos que son importantes de conocer para entender el resto del proyecto.

A continuación, en el Capítulo 3 repasamos el estado del arte explicando estudios y herramientas anteriores que se han utilizado para detectar contenido sexual en

imágenes, la explicación en profundidad de las arquitecturas que utilizaremos en nuestros reentrenamientos y, por último, algunos trabajos anteriores que han utilizado la técnica de DA para mejorar sus modelos de DL.

En el Capítulo 4 se ofrece una explicación exhaustiva de la metodología del proyecto. En primer lugar, se hacen indicaciones sobre qué hacer antes de reentrenar un modelo y, a continuación, todos los reentrenamientos realizados con cada arquitectura y las métricas obtenidas con ellos.

Después, en el Capítulo 5 comentamos y analizamos las distintas pruebas y experimentos que realizamos con el modelo que consideramos de mejor rendimiento.

Por último, en el Capítulo 6 comentamos las conclusiones obtenidas con el trabajo realizado y los posibles trabajos futuros que se podrían realizar en base a nuestro proyecto.

Chapter 8

Conclusiones y Trabajo Futuro

8.1 Conclusiones

En este proyecto hemos probado diferentes arquitecturas de DL para la detección explícita de contenido en imágenes. Para ello, tuvimos que realizar una gran búsqueda de conjuntos de imágenes, tanto para el reentrenamiento del modelo como para las pruebas finales. Aunque este proceso de recopilación de conjuntos de datos fue largo y tedioso, gracias a que conseguimos recopilar muchas imágenes y de diferentes tipos, pudimos realizar buenos reentrenamientos y pruebas de las arquitecturas de MobileNet. Hemos podido comprobar que, estos algoritmos de AI son muy útiles para la detección de patrones en imágenes y que, efectivamente, pueden ser eficaces para controlar el consumo de contenido sexual. Además, hemos podido comprobar que estas redes neuronales no sólo se pueden utilizar en ordenadores con altas capacidades de almacenamiento y procesamiento, sino que también son muy aplicables a teléfonos móviles, que tienen muchas más restricciones, ya que los modelos que hemos reentrenado tienen muchos menos parámetros que otros y son fácilmente transportables a estos teléfonos móviles.

Además, hemos podido comprobar, gracias a las técnicas de interpretabilidad que se han desarrollado anteriormente, que los modelos de DL realmente identifican patrones repetitivos y consiguen identificarlos en imágenes completamente nuevas que nunca antes habían visto. El hecho de que los algoritmos sean capaces de hacer algo así, sin la ayuda extra de un ser humano, es fascinante y podría aplicarse en muchos campos en el futuro. A mí personalmente, este proyecto me ha ayudado a aprender mucho más sobre este campo y, gracias a él, quiero seguir investigando y aprendiendo nuevas técnicas y aplicaciones de estos interesantes algoritmos.

8.2 Trabajo Futuro

Algunas líneas de trabajo que se podrían continuar después de este proyecto serían las siguientes. Aplicar el modelo en smartphones utilizando la librería TensorFlow Lite y probarlo en una aplicación móvil. Esta aplicación podría tener varias opciones de actuación, por ejemplo, si el modelo encuentra una imagen con contenido sexual pero con un porcentaje inferior al 70% de posibilidades, se podría revisar la imagen y decidir si se puede mantener la imagen o se debe borrar. Además, en función de esa elección, el modelo podría volver a entrenarse con esta nueva información y, así, podría ajustarse mejor a las preferencias del usuario. Otra opción es que, como hemos visto que el modelo funciona muy bien detectando diferentes patrones en imágenes, esta arquitectura podría ser reentrenada para ser utilizada en otros campos de la detección de objetos en imágenes, como por ejemplo, el reconocimiento de problemas médicos en imágenes, el reconocimiento facial o la identificación de objetos para vehículos de conducción automática.

Bibliography

[AMAZ17] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017.

[CCD08] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised learning. *Machine learning techniques for multimedia: case studies on organization and retrieval*, pages 21–49, 2008.

[CDF+04] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, pages 1–2. Prague, 2004.

[CGP21] Saket S Chaturvedi, Kajol Gupta, and Prakash S Prasad. Skin lesion analyser: an efficient seven-way multi-class skin cancer classification using mobilenet. In *Advanced Machine Learning Technologies and Applications: Proceedings of AMLTA 2020*, pages 165–176. Springer, 2021.

[CTR+17] Supriyo Chakraborty, Richard Tomsett, Ramya Raghavendra, Daniel Harborne, Moustafa Alzantot, Federico Cerutti, Mani Srivastava, Alun Preece, Simon Julier, Raghuveer M Rao, et al. Interpretability of deep learning models: A survey of results. In *2017 IEEE smartworld, ubiquitous intelligence & computing, advanced & trusted computed, scalable computing & communications, cloud & big data computing, Internet of people and smart city innovation (smartworld/SCALCOM/UIC/ATC/CBDcom/IOP/SCI)*, pages 1–6. IEEE, 2017.

[DPN08] Thomas Deselaers, Lexi Pimenidis, and Hermann Ney. Bag-of-visual-words models for adult image classification and filtering. In *2008 19th International Conference on Pattern Recognition*, pages 1–4. IEEE, 2008.

[ES21] Prem Enkvetchakul and Olarik Surinta. Effective data augmentation and training techniques for improving deep learning in plant leaf disease recognition. 2021.

[FFB96] Margaret M Fleck, David A Forsyth, and Chris Bregler. Finding naked people. In *Computer Vision—ECCV'96: 4th European Conference on Computer Vision Cambridge, UK, April 15–18, 1996 Proceedings Volume II 4*, pages 593–602. Springer, 1996.

[Fje20] Ragnar Fjelland. Why general artificial intelligence will not be realized. *Humanities and Social Sciences Communications*, 7(1):1–9, 2020.

[Gha04] Zoubin Ghahramani. Unsupervised learning. *Advanced Lectures on Machine Learning: ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, pages 72–112, 2004.

[HSC+19] Andrew Howard, Mark Sandler, Bo Chen, Weijun Wang, Liang-Chieh Chen, Mingxing Tan, Grace Chu, Vijay Vasudevan, Yukun Zhu, Ruoming Pang, Hartwig Adam, and Quoc Le. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019.

[HSS18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[HWA18] Ario Yudo Husodo, I Gede Pasek Suta Wijaya, and I Wayan Agus Arimbawa. Realtime porn image censor method for preventing smartphone users to take a pornographic photo. In *2018 5th International Conference on Data and Software Engineering (ICoDSE)*, pages 1–6. IEEE, 2018.

[HZC+17] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[KCM22] Ibrahem Kandel, Mauro Castelli, and Luca Manzoni. Brightness as an augmentation technique for image classification. *Emerging Science Journal*, 2022.

[Koo21] Brett Koonce. *Convolutional Neural Networks with Swift for Tensorflow: Image Recognition and Dataset Categorization.* 2021.

[KSCJ22] Sasan Karamizadeh, Saman Shojae Chaeikar, and Alireza Jolfaei. Adult content image recognition by boltzmann machine limited and deep learning. *Evolutionary Intelligence*, pages 1–10, 2022.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[LC15] Zhibin Liao and G. Carneiro. On the importance of normalisation layers in deep learning with piecewise linear activation units. *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–8, 2015.

[NVK+15] Maryam M Najafabadi, Flavio Villanustre, Taghi M Khoshgoftaar, Naeem Seliya, Randall Wald, and Edin Muharemagic. Deep learning applications and challenges in big data analytics. *Journal of big data*, 2(1):1–21, 2015.

[OBMR12] Eric W Owens, Richard J Behun, Jill C Manning, and Rory C Reid. The impact of internet pornography on adolescents: A review of the research. *Sexual Addiction & Compulsivity*, 19(1-2):99–122, 2012.

[ON15] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[OOAS18] saban Ozturk, Umut Ozkaya, Bayram Akdemir, and Levent Seyfi. Convolution kernel size effect on convolutional neural network in histopathological image processing applications. In *2018 International Symposium on Fundamentals of Electrical Engineering (ISFEE)*, pages 1–5. IEEE, 2018.

[POSW17] Pornntiwa Pawara, Emmanuel Okafor, Lambert Schomaker, and Marco A Wiering. Data augmentation for plant classification. In *Advanced Concepts for Intelligent Vision Systems Conference*, 2017.

[PVAV18] Fábio Perez, Cristina Nader Vasconcelos, Sandra Avila, and Eduardo Valle. Data augmentation for skin lesion analysis. In *OR 2.0/CARE/CLIP/ISIC@MICCAI*, 2018.

[QWW+18] Xuexin Qu, Xin Wang, Zihan Wang, Lei Wang, and Lingchen Zhang. Perceptual-dualgan: perceptual losses for image to image translation with generative adversarial nets. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[RCS+18] Stephenn L Rabano, Melvin K Cabatuan, Edwin Sybingco, Elmer P Dadios, and Edwin J Calilung. Common garbage classification using mobilenet. In *2018 IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, pages 1–4. IEEE, 2018.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, page 1929–1958, 2014.

[SHZ+18] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[SK19] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[SLC04] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local svm approach. In *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.*, volume 3, pages 32–36. IEEE, 2004.

[SLD+16] Lucy Watchirs Smith, Bette Liu, Louisa Degenhardt, Juliet Richters, George Patton, Handan Wand, Donna Cross, Jane S Hocking, S Rachel Skinner, Spring Cooper, et al. Is sexual content in new media linked to sexual risk behaviour in young people? a systematic review and meta-analysis. *Sexual Health*, 13(6):501–515, 2016.

[SM19] Ajay Shrestha and Ausif Mahmood. Review of deep learning algorithms and architectures. *IEEE access*, 7:53040–53065, 2019.

[SS19] P Sreelakshmi and M Sumithra. Facial expression recognition robust to partial occlusion using mobilenet. *International Journal of Engineering Research & Technology (IJERT)*, 8(6):1387–1391, 2019.

[SZA11] Maliheh Shabanzade, Morteza Zahedi, and Seyyed Amin Aghvami. Combination of local descriptors and global features for leaf recognition. *Signal & Image Processing*, 2(3):23, 2011.

[TCP+19] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2815–2823, 2019.

[VdALJ+11] Eduardo Valle, Sandra de Avila, Antonio da Luz Jr, Fillipe de Souza, Marcelo Coelho, and Arnaldo Araújo. Content-based filtering for video sharing social networks. *arXiv preprint arXiv:1101.2427*, 2011.

[VPA+19] Jessica Velasco, Cherry Pascion, Jean Wilmar Alberio, Jonathan Apuang, John Stephen Cruz, Mark Angelo Gomez, Benjamin Molina Jr, Lyndon Tuala, August Thio-ac, and Romeo Jorda Jr. A smartphone-based skin disease classification using mobilenet cnn. *arXiv preprint arXiv:1911.07929*, 2019.

[WHW20] Adi Wibowo, Cahyo Adhi Hartanto, and Panji Wisnu Wirawan. Android skin cancer detection and classification based on mobilenet v2 model. *International Journal of Advances in Intelligent Informatics*, 6(2):135–148, 2020.

[ZFM+20] Pan Zhou, Jiashi Feng, Chao Ma, Caiming Xiong, Steven C. H. Hoi, and E Weinan. Towards theoretically understanding why sgd generalizes better than adam in deep learning. *ArXiv*, abs/2010.05627, 2020.

[Zho21] Zhi-Hua Zhou. *Machine learning*. Springer Nature, 2021.