

**UNIVERSIDAD DE INVESTIGACIÓN DE
TECNOLOGÍA EXPERIMENTAL YACHAY**

Escuela de Ciencias Matemáticas y Computacionales

**TÍTULO: A computer vision model to identify the
incorrect use of face masks for COVID-19 awareness**

Trabajo de integración curricular presentado como requisito para la
obtención del título de ingeniero en Tecnologías de la Información

Autor/a:

Crespo Yaguana Jonnathan Fabricio

Tutor/a:

Ph.D. - Morocho Cayamcela Manuel Eugenio

Urcuquí, agosto de 2022

Urcuquí, 19 de agosto de 2022

**SECRETARÍA GENERAL
ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES
CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN
ACTA DE DEFENSA No. UITEY-ITE-2022-00025-AD**

En la ciudad de San Miguel de Urcuquí, Provincia de Imbabura, a los 19 días del mes de agosto de 2022, a las 09:00 horas, en el Aula S_CAN de la Universidad de Investigación de Tecnología Experimental Yachay y ante el Tribunal Calificador, integrado por los docentes:

Presidente Tribunal de Defensa Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Miembro No Tutor ARMAS ANDRADE, TITO ROLANDO , Ph.D.
Tutor Dr. MOROCHO CAYAMCELA, MANUEL EUGENIO , Ph.D.

Se presenta el(la) señor(ita) estudiante **CRESPO YAGUANA, JONNATHAN FABRICIO**, con cédula de identidad No. **0302616107**, de la **ESCUELA DE CIENCIAS MATEMÁTICAS Y COMPUTACIONALES**, de la Carrera de **TECNOLOGÍAS DE LA INFORMACIÓN**, aprobada por el Consejo de Educación Superior (CES), mediante Resolución **RPC-SO-43-No.496-2014**, con el objeto de rendir la sustentación de su trabajo de titulación denominado: **A computer vision model to identify the incorrect use of face masks for COVID-19 awareness** , previa a la obtención del título de **INGENIERO/A EN TECNOLOGÍAS DE LA INFORMACIÓN**.

El citado trabajo de titulación, fue debidamente aprobado por el(los) docente(s):

Tutor Dr. MOROCHO CAYAMCELA, MANUEL EUGENIO , Ph.D.


Y recibió las observaciones de los otros miembros del Tribunal Calificador, las mismas que han sido incorporadas por el(la) estudiante.

Previamente cumplidos los requisitos legales y reglamentarios, el trabajo de titulación fue sustentado por el(la) estudiante y examinado por los miembros del Tribunal Calificador. Escuchada la sustentación del trabajo de titulación, que integró la exposición de el(la) estudiante sobre el contenido de la misma y las preguntas formuladas por los miembros del Tribunal, se califica la sustentación del trabajo de titulación con las siguientes calificaciones:


Tipo	Docente	Calificación
Miembro Tribunal De Defensa	ARMAS ANDRADE, TITO ROLANDO , Ph.D.	10,0
Tutor	Dr. MOROCHO CAYAMCELA, MANUEL EUGENIO , Ph.D.	10,0
Presidente Tribunal De Defensa	Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.	8,0


Lo que da un promedio de: **9.3 (Nueve punto Tres)**, sobre 10 (diez), equivalente a: **APROBADO**


Para constancia de lo actuado, firman los miembros del Tribunal Calificador, el/la estudiante y el/la secretario ad-hoc.


CRESPO YAGUANA, JONNATHAN FABRICIO
Estudiante


Dr. CUENCA PAUTA, ERICK EDUARDO , Ph.D.
Presidente Tribunal de Defensa


Dr. MOROCHO CAYAMCELA, MANUEL EUGENIO , Ph.D.
Tutor


ARMAS ANDRADE, TITO ROLANDO , Ph.D.
Miembro No Tutor


MEDINA BRITO, DAISY MARGARITA
Secretario Ad-hoc

Autoría

Yo, **JONNATHAN FABRICIO CRESPO YAGUANA**, con cédula de identidad 0302616107, declaro que las ideas, juicios, valoraciones, interpretaciones, consultas bibliográficas, definiciones y conceptualizaciones expuestas en el presente trabajo; así cómo, los procedimientos y herramientas utilizadas en la investigación, son de absoluta responsabilidad de el/la autor/a del trabajo de integración curricular. Así mismo, me acojo a los reglamentos internos de la Universidad de Investigación de Tecnología Experimental Yachay.

Urcuquí, agosto de 2022.

Jonnathan Fabricio Crespo Yaguana

CI: 0302616107

Autorización de publicación

Yo, **JONNATHAN FABRICIO CRESPO YAGUANA**, con cédula de identidad 0302616107, cedo a la Universidad de Investigación de Tecnología Experimental Yachay, los derechos de publicación de la presente obra, sin que deba haber un reconocimiento económico por este concepto. Declaro además que el texto del presente trabajo de titulación no podrá ser cedido a ninguna empresa editorial para su publicación u otros fines, sin contar previamente con la autorización escrita de la Universidad.

Asimismo, autorizo a la Universidad que realice la digitalización y publicación de este trabajo de integración curricular en el repositorio virtual, de conformidad a lo dispuesto en el Art. 144 de la Ley Orgánica de Educación

Urcuquí, agosto de 2022.

Jonnathan Fabricio Crespo Yaguana

CI: 0302616107

Dedication

To my parents, Polivio and Pilar, and my brother Anthony for all their support and for being my inspiration.

Jonnathan Fabricio Crespo Yaguana

Acknowledgment

Thanks to my family for all their support throughout these years. Thanks to my friends in my hometown and at Yachay Tech for becoming my family and giving me their company and motivation. Thanks to Yachay Tech and the staff of professors for give me invaluable knowledge inside and outside the classrooms; it was a wonderful experience. Last but not the least, a very special thanks to my advisor Manuel Eugenio Morocho Cayamcela, Ph.D., for being patient and sharing all his experience and knowledge in this work with me.

Jonnathan Fabricio Crespo Yaguana

Resumen

La detección de mascarillas faciales se ha convertido en un gran desafío en la visión por computadora, lo que exige la unión de la tecnología con la conciencia de COVID-19. Los investigadores han propuesto modelos de aprendizaje profundo para detectar el uso de mascarillas. Sin embargo, el uso incorrecto de una mascarilla puede ser tan perjudicial como no llevar protección alguna. En esta tesis, proponemos una arquitectura de red neuronal convolucional (CNN) basada en dos tareas de visión por computadora: localización de objetos para descubrir rostros en imágenes/videos, seguida de un modelo de CNN de clasificación de imágenes para categorizar los rostros y mostrar si alguien está usando una mascarilla correctamente, incorrectamente o no utiliza alguna. La primera CNN se basa en RetinaFace, un modelo para detectar rostros en imágenes; mientras que la segunda CNN utiliza una arquitectura Resnet-18 como columna vertebral de clasificación. Nuestro modelo permite una identificación precisa de las personas que no están siguiendo correctamente las recomendaciones sanitarias de COVID-19 sobre el uso de mascarillas. Hemos lanzado al público tanto el conjunto de datos utilizado para entrenar el modelo de clasificación como nuestro modelo propuesto de visión artificial, y los hemos optimizado para la implementación de sistemas integrados, lo que permite un uso global de nuestra tecnología.

Palabras Clave:

Inteligencia artificial, aprendizaje profundo, visión artificial, reconocimiento de rostros con mascarilla, detección de objetos, clasificación de imágenes, COVID-19.

Abstract

Face mask detection has become a great challenge in computer vision, demanding the coalition of technology with COVID-19 awareness. Researchers have proposed deep learning models to detect the use of face masks. However, the incorrect use of a face mask can be as harmful as not wearing any protection at all. In this thesis, we propose a compound convolutional neural network (CNN) architecture based on two computer vision tasks: object localization to discover faces in images/videos, followed by an image classification CNN to categorize the faces and show if someone is using a face mask correctly, incorrectly, or not wearing any mask at all. The first CNN is built upon RetinaFace, a model to detect faces in images; whereas the second CNN uses a Resnet-18 architecture as a classification backbone. Our model enables an accurate identification of people who are not correctly following the COVID-19 healthcare recommendations on face masks use. We have released both the dataset used to train the classification model and our proposed computer vision pipeline to the public, and optimized it for embedded systems deployment, empowering a global use of our technology.

Keywords:

Artificial intelligence, deep learning, computer vision, face mask recognition, object detection, image classification, COVID-19.

Contents

Dedication	v
Acknowledgment	vii
Resumen	ix
Abstract	xi
Contents	xiii
List of Tables	xvii
List of Figures	xix
1 Introduction	1
1.1 Problem statement	1
1.2 Justification	2
1.3 Contribution	2
1.4 Objectives	3
1.4.1 General objective	3
1.4.2 Specific objectives	3
2 Theoretical Framework	5
2.1 Artificial intelligence	5
2.1.1 Machine learning	5
2.1.2 Deep learning	6
2.1.3 Computer vision as a field of artificial intelligence	6
2.2 Artificial neural networks	7

2.2.1	Standard neural network	7
2.2.2	Training an ANN	7
2.2.3	Activation functions	11
2.3	Convolutional neural networks	12
2.3.1	Convolution in CNN	12
2.4	CNN architecture	18
2.4.1	Convolutional layers	18
2.4.2	Pooling layers	19
2.4.3	Fully connected layers	20
2.5	Convolutional neural networks from the beginning to VGG	21
2.5.1	Simple and complex cells	21
2.5.2	Neocognitron	22
2.5.3	From invariant recognition to convolutional neural networks: LeNet-5	22
2.5.4	Convolutional neural networks from 1998 to 2010	23
2.5.5	AlexNet, 2012	24
2.5.6	VGG	25
2.6	ResNet: the way to effective deeper CNNs	26
2.6.1	The problem	26
2.6.2	Deep residual learning framework as a solution to the degradation problem	28
2.7	Training a CNN	30
2.7.1	Datasets	31
2.7.2	Deep learning frameworks	34
2.7.3	Training approaches	35
2.7.4	Hyperparameters	36
2.7.5	Analyzing results	37
3	State of the Art	41
3.1	Face and face-mask datasets	41
3.1.1	Face datasets	41
3.1.2	Face-mask datasets	42

3.2	Relevant face detection models	43
3.3	Face mask detection and classification for COVID-19	45
4	Methodology	49
4.1	System design	49
4.1.1	The datasets used to train the proposed model	49
4.1.2	Detection and clasification models	54
4.1.3	The two-stage pipeline CNN	56
4.1.4	Loss function and optimizers	56
4.1.5	Framework and hardware acceleration	57
4.1.6	Performance measures	58
4.1.7	Prepare the dataset	58
5	Results and Discussion	61
5.1	Phase I: Test deep learning classification models with different optimizers .	61
5.2	Phase II: Improving the dataset, training the classification model for two classes, and extending it to 3 classes	65
5.2.1	Datasets	65
5.2.2	Training	66
5.2.3	Hardware	67
5.2.4	Results and analysis	67
5.3	Visualizing some feature maps and filters	77
5.3.1	ResNet-18 filters	78
5.4	Grad-CAMs through the layers	80
5.4.1	COVID-19 mask classification model with ResNet-18 against other approaches	80
6	Conclusions	85
6.1	General conclusions	85
6.2	Future work	86
	Bibliography	89

List of Tables

4.1	Summary of the construction of the FMLD dataset. FMLD takes the donor datasets MAFA and WiderFace to construct the “compliant” and “non-compliant” classes.	52
5.1	An overview of the dataset split used to train the different classification models.	61
5.2	Models with different optimizers.	63
5.3	New FMLD dataset after cleaning.	66
5.4	New dataset formed by FMLD + MMD datasets extended to three classes.	66
5.5	Additional evaluation metrics for COVID-19 mask wearing classification model on New Test dataset.	67
5.6	Additional evaluation metrics for COVID-19 mask wearing classification model on New Test dataset for 3 classes.	71
5.7	ResNet-18 in comparisson with other approaches.	82

List of Figures

2.1	Linear Regression and a Neural Network representation of COVID-19 infections prediction.	8
2.2	A simple neural network composed by a input layer, one hidden layers and an output.	8
2.3	A basic ANN composed by inputs, weights, bias, activation function and the output. Note that in the neuron is performed the sum of the product between the vector x and the weights b and the the activation function is applied to generate the output.	10
2.4	Examples of 2D and a volume convolution. In this particular case, the volume convolution is 3D.	13
2.5	Examples of complete 2D and a 3D volume convolution.	14
2.6	A summary of a convolution with padding is presented.	16
2.7	A convolution with padding and stride is presented.	17
2.8	A convolution with two filters is presented. The outputs of each filter are stacked to form the final output.	18
2.9	A convolution with two filters is presented. The bias term is added to the each output and then the ReLU function is applied. The final result is stacked in order to form the final output.	19
2.10	Two types of figures are presented: max pooling and average pooling.	20
2.11	A CNN is presented. It is composed by the input layer, the convolutional layers, the fully connected layers and the output layer. Pooling layers are implicity next to the CONVs.	21
2.12	LeNet-5 architecture for digits recognition.	23

2.13 AlexNet architecture.	24
2.14 VGG-16 architecture.	26
2.15 Deeper plain networks error rate.	27
2.16 Deeper networks error in the training dataset. In theory, as more layers, less error rate. However, from a certain number of stacked layers, in reality, the error rate begins to increase.	28
2.17 Residual learning: a building block.	28
2.18 Residual learning: a building block.	29
2.19 Conv Block.	30
2.20 ResNet error improvement in the training dataset with deeper layers. . . .	31
2.21 Plain and Residual Networks	32
2.22 A confusion matrix for two classes is presented. We can see the TP, TN, FP and FN.	38
3.1 Several pictures obtained from the MAFA dataset that includes general oc- clusions considered as masks	42
3.2 Image extracted from the RMFRD dataset showing 4 pictures of different persons using masks.	43
3.3 Pictures extracted from the Moxa3k database: a) Blurred faces b) top angle camera view c) rotated face d) crowded image e) foggy environment f) sunny day faces	44
3.4 RefineFace structure	45
3.5 Context-Attention R-CNN-based model schematic.	47
3.6 FMRN architecture has a total of eleven layers except for the input layer and the output layer.	48
4.1 Workflow followed to develop the two-stage pipeline to detect the incorrect use of face masks for COVID-19 awareness	50
4.2 WiderFace dataset images with different features.	51
4.3 FMLD dataset proportions according to gender, pose and ethnicity of people.	52
4.4 Two-stage pipeline. The first stage is in charge to detect all the faces in an image, while the second stage is in charge to make the classification.	53

4.5	A subset of images taken from the “compliant” class of the FMLD dataset. The examples (a)-(h) show faces with the correct use of masks according to the WHO advice.	53
4.6	A subset of images taken from the “non-compliant” class of the FMLD dataset. The examples (a)-(h) show faces that are incorrectly using (or not using) the masks.	54
4.7	RetinaFace model has three main components: 1) Feature Pyramid Network, 2) Context Head Module, and 3) Cascade Multi Task Loss.	55
4.8	ResNet-18 architecture composed by residual blocks with two types of shortcuts: identity shortcuts and projection shortcuts.	55
5.1	ResNet-34 accuracy on train dataset.	62
5.2	ResNet-34 loss on train dataset.	62
5.3	Resnet-34 accuracy on validation dataset.	63
5.4	Resnet-34 loss on validation dataset.	63
5.5	Test results of all the considered models with different optimizers.	64
5.6	Confusion matrix of the results obtained from ResNet-34 with Adagrad optimizer in the classification stage.	65
5.7	Confusion matrix about the testing of ResNet-18 COVID-19 mask wearing classification model on the new test dataset.	68
5.8	Compliant images classified as non-compliant.	68
5.9	Compliant images classified as non-compliant analyzed with Grad-CAMs.	69
5.10	Compliant images classified as non-compliant.	69
5.11	Non-compliant images classified as compliant analyzed with Grad-CAMs.	70
5.12	ResNet-18 accuracy and loss on train dataset for 2 classes model.	70
5.13	Confusion matrix about the testing of ResNet-18 COVID-19 mask wearing classification model on the new test dataset for three classes.	71
5.14	Compliant images wrong classified. Each one has the predicted label assigned by Resnet18 classification model.	72
5.15	Compliant images wrong predicted analyzed with Grad-CAMs.	73

5.16 Non-compliant images wrong classified. Each one has the predicted label assigned by Resnet18 classification model.	74
5.17 Non-compliant images wrong predicted analyzed with Grad-CAMs.	74
5.18 Incorrect images wrong classified. Each one has the predicted label assigned by Resnet18 classification model.	75
5.19 Incorrect images wrong predicted analyzed with Grad-CAMs.	76
5.20 Resnet-18 accuracy and loss on train dataset for 3 classes model.	77
5.21 Features maps corresponding to each convolutional layers of Resnet-18.	77
5.22 64 Filters corresponding to the first convolutional layer of Resnet-18.	78
5.23 5 images together after their features maps after applying seven different filters in the first convolutional layer.	79
5.24 All 64 filters of the first convolutional layer with the feature maps they produce.	79
5.25 Filters with their respective featuremaps of convolutional layers 1, 5, 10, 15 and 17.	81
5.26 Grad-CAMs per convolutional layer for an image of each class.	82
5.27 Confusion matrix corresponding to MobileNetV2 + SVM and VGG19 + KNN	83
5.28 ResNet-18 results after be applied in the other dataset.	83

Chapter 1

Introduction

1.1 Problem statement

The beginning of 2020 was an atypical year for different areas of research: a pandemic began to spread worldwide; with Wuhan, Hubei, China, as the epicenter of the new virus. The first cases of a strange respiratory disease appeared, whose worst consequence was the death of the infected patient [1]. The new virus was called “severe acute respiratory syndrome coronavirus-2” (SARS-CoV-2, 2019-nCoV) [2], causing the disease named COVID-19 [3]. The origin of the disease is still being studied today, but some research works point the root to zoonotic transmission [4]. However, the human-to-human transmission plays an essential role in the spread of the disease. This fact is mainly due to asymptomatic patients’ presence, and the time of incubation of the virus, which can take several days [3]. The latter facts make the detection of the virus imperceptible in most cases, facilitating the spread of the disease between individuals. At the beginning of the pandemic, the principal actions taken by almost all the countries around the world was confinement and quarantine. Nevertheless, confinement was not a feasible solution due to the economic impact it entailed. It is known that one of the principal ways of SARS-CoV-2 transmission is through tiny droplets ejected from a pre-symptomatic patient while sneezing, coughing, or only speaking. Therefore, the use of face masks was imposed by the World Health Organization (WHO), especially in public areas to reduce the rates of virus spreading [5]. The mandatory use of face masks varies over countries; for instance, due to the arrival of COVID-19 vaccines and the low contagion rate, some countries like Israel have waived the obligatory face mask use [6].

However, some countries still have high rates of COVID-19 cases, and a significant part of the population is not vaccinated yet, making the use of face masks a mandatory rule [7, 8, 9, 10]. These countries have been facing the challenge of creating a preventive system to detect the incorrect (or the absence) use of face masks.

1.2 Justification

In general, monitoring people who do not respect the use of face masks is a challenging task, especially in crowded public areas where it becomes impossible to track if everybody is using face masks according to the WHO recommendations. For this reason, the need of a computer system that enables the automatic monitoring of correct/incorrect face mask use becomes highly-demanded. Currently, computer vision (CV) tasks are useful for solving problems related to object detection, classification, object counting, visual surveillance, etc., while taking advantage of video resources from public surveillance cameras located in most of the public areas (i.e., shopping malls, supermarkets, airports, train stations, stadiums, etc.). The problem related to the correct/incorrect wearing of face masks implies two CV tasks: 1) object detection, and 2) object classification. The object detection task is helpful to find the faces of people in images or videos, and the object classification task uses the faces detected by the object locator to classify them in different classes (e.g., correct or incorrect use of face masks).

1.3 Contribution

In this work, a compound convolutional neural network (CNN) pipeline for COVID-19 face mask detection and classification is proposed. In addition, we introduce an optimal hyperparameter configuration for the cost function and the gradient-based optimizer in the classification stage. Moreover, a new cleaned dataset is presented to train the object classification model. To enable reproducible research, we have used the public and open Face-Mask Label Dataset (FMLD) [11] and Medical Mask Dataset (MMD)[12] to train the classifier with distinct DL models.

1.4 Objectives

1.4.1 General objective

CV is an exciting approach that provides the necessary tools to build systems capable of detecting and classifying the correct/incorrect use of face masks. However, to increase the accuracy of the detection and classification models, CV researchers are needed to help in the development of models that enhance the existent systems. In the last year, the CV research community have worked hard to propose deep learning (DL) models to tackle this vital area in benefit of society's healthcare [13, 14, 15]. However, a significant part of the research is focused on generating suitable datasets to train the models, especially for the detection and classification of occluded faces. Therefore, the general objective of this thesis is to address related approaches to the classification of the COVID-19 face mask-wearing to give an accurate DL model.

1.4.2 Specific objectives

- Design a covid mask-wearing classification pipeline to identify the incorrect use of face masks for COVID-19 awareness.
- Training different DL models in order to perform the covid mask classification task.
- Determine an optimal set up of hyperparameters such as the learning rate, number of epochs, and optimizer to be used during the training of a DL model.
- Apply different CV techniques to pre-process public datasets in order to guarantee their quality and, therefore, the training of the DL models.
- Extend the covid mask classification classes from two compliant and non-compliant to three: compliant, incorrect, and non-compliant.

Chapter 2

Theoretical Framework

2.1 Artificial intelligence

Artificial intelligence (AI) was first defined in 1955 by John McCarthy as the science and engineering of making intelligent machines [16]. Currently, AI has multiple applications in fields such as business, healthcare, education, military, and manufacturing, among others [17]. Furthermore, some of the branches of AI include machine learning (ML), natural language processing (NLP), DL, CV, robotics, and speech recognition.

2.1.1 Machine learning

ML is a branch of AI that can be defined as a set of computational techniques capable of improving performance and precision through experience, and the usage of data [18]. However, this learning does not occur in a unique manner, but can be carried out in different ways, depending on the amount of human supervision. In this manner, ML can be divided into three main categories that are: *supervised learning*, *unsupervised learning*, and *reinforcement learning* [19]. In supervised learning, the learning process uses labeled data, which contains the information that needs to be predicted. Classification and regression are the main applications of supervised learning. Unsupervised learning makes use of unlabeled datasets, and the model by itself is in charge of inferring patterns from it. This type of learning is mainly applied in clustering and dimensional reduction tasks. Finally, there are three fundamental elements in reinforcement learning: an agent, the environment and actions. In this type of learning, the agent observes the environment, selects and performs

actions, and obtains a reward or penalty in return. The agent's objective is to choose the actions that maximize the expected reward in a time interval, in other words, to choose the best strategy or policy [19]. Reinforcement learning is commonly used in gaming, robotics, and navigation. [20].

2.1.2 Deep learning

DL is a subfield of ML that involves the use of artificial neural networks (ANNs) and algorithms to train them [21]. ANNs are inspired by the functioning of biological neural networks. A network can be made to learn how to solve different problems by applying algorithms that mimic the process of real neurons [22]. A DL model requires more than one hidden layer in a neural network. Additionally, different types of neural network architectures can be found. Among them, it can be mentioned convolutional neural networks (CNNs), recurrent neural networks (RNNs), and deep belief networks (DBNs).

In ML, classification tasks are performed in sequential steps that include preprocessing, feature extraction, intelligent feature selection, learning, and classification. However, a non-optimal selection of features can lead to incorrect results in discrimination between classes. Different from this, DL allows for the automation of feature learning so that learning and classification can be accomplished at once [23].

2.1.3 Computer vision as a field of artificial intelligence

CV is a field of AI that aims to endow a computer with the ability to obtain a detailed understanding of visual data like human vision systems [24]. CV is applied in different tasks such as object detection, face recognition, action and activity recognition, human pose estimation, and semantic segmentation [25]. Some examples of CV applications include facial detection and recognition, optical character recognition (OCR), autonomous driving, image generation (GAN), virtual reality (VR), and the detection of pathologies and tumors in medicine.

2.2 Artificial neural networks

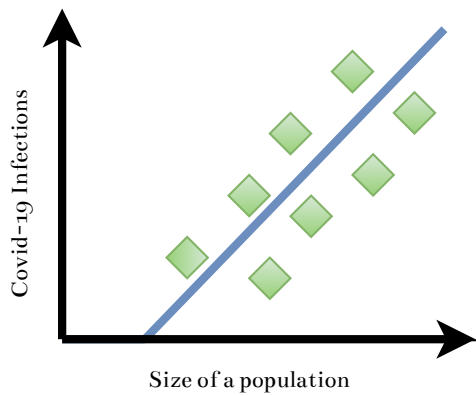
2.2.1 Standard neural network

An ANN is a collection of stacked layers, each one composed of neurons. This concept by itself can be challenging to understand, so it is better to introduce them in parts and examples. Suppose you want to predict the number of COVID-19 infections according to the size of a population. It is clear that a linear regression representation to predict this problem is as in Figure 2.1a. If we want to transform the linear regression representation presented in Figure 2.1a into a neural network representation can be something similar to Figure 2.1b. Here, we can see a simplest neural network. It has three principal components: the input, which is the size of the population; the output, which is the number of infections, and a “neuron” between them. A neuron is a unit in charge of taking the input, performing the linear function, and producing the output. Later, we will see that the linear function is called the activation function. In particular, this primary neural network represents only two features: the size of the population and the number of infections. However, many real applications have a considerable number of features. Representing them as a neural network is by adding multiple basic units presented in Figure 2.1b and stacking multiple layers. A layer is a set of neurons. Therefore, a more accurate representation of a neural network is presented in Figure 2.2. This example shows a standard neural network with four inputs, a hidden layer composed of three neurons (hidden because it is between the input and output), and unique output. If we pay attention to the connection between the neurons, we can see all are connected to the other ones, and in this case, we say they are dense connected. This means that all the neurons will contribute to the computation of the output of each one. Finally, all the neural networks can be understood as inputs/features/training samples that produce a prediction.

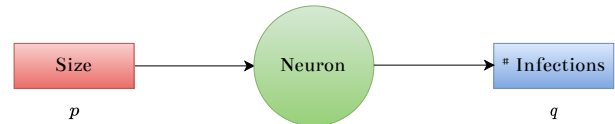
2.2.2 Training an ANN

Let us define a set of training examples

$$M = \left\{ \left(x^{(1)}, y^{(1)} \right), \left(x^{(2)}, y^{(2)} \right), \dots, \left(x^{(m)}, y^{(m)} \right) \right\}, \text{ with } x \in \mathbb{R}^{n_x}, n, m \in \mathbb{R}^+, y \in \mathbb{R} \quad (2.1)$$



(a) A Linear Regression to predict the number of COVID-19 infections according to the size of a population.



(b) COVID-19 infections according to the size of a population represented as a neural network. The input p represents the size of the population and the output q is the number of infections.

Figure 2.1: Linear Regression and a Neural Network representation of COVID-19 infections prediction.

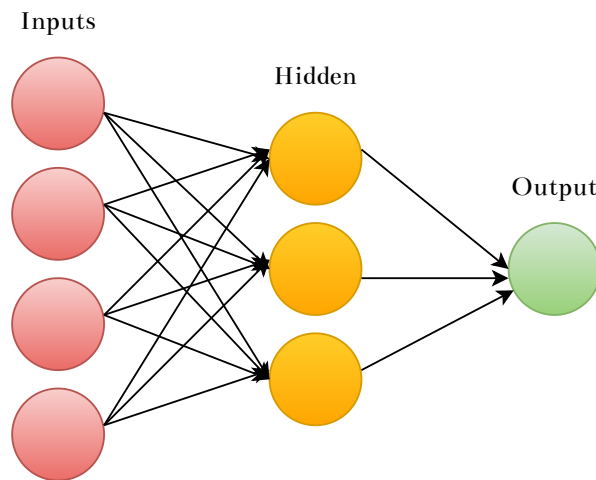


Figure 2.2: A simple neural network composed by a input layer, one hidden layers and an output.

where $x^{(i)}$ is the vector of dimension $n \times 1$ corresponding to one training example, $y^{(i)}$ as its corresponding label, n_x as the dimension of the input feature x and, m as the number of training examples. From here, we can define a matrix of inputs x as

$$X = [x^{(1)}, x^{(2)}, \dots, x^{(m)}], \text{ with } X \in \mathbb{R}^{n_x \times m} \tag{2.2}$$

and a vector of outputs y

$$Y = [y^{(1)}, y^{(2)}, \dots, y^{(m)}], \text{ with } Y \in \mathbb{R}^{1 \times m} \quad (2.3)$$

The main purpose of training is to find a set of parameters for the ANN to make predictions for each training sample. These predictions are found through probabilities in the final layer of the ANN and said probabilities would lean towards one class or another. A set of predictions \hat{y} for the training samples X can be defined as

$$\hat{Y} = [\hat{y}^{(1)}, \hat{y}^{(2)}, \dots, \hat{y}^{(m)}], \text{ with } \hat{Y} \in \mathbb{R}^{1 \times m} \quad (2.4)$$

Additionally, there are two principal parameters to be fine-tuned: The weights and the bias term. The weights w can be seen as a set of coefficients that will multiply each component of some input vector to produce an output determining how important is that particular input for the prediction. On the other hand, the bias b term is a constant added to the product between the inputs and weights as an offset, empowering or weakening the importance of that input to give more flexibility and generalization to the model. In formal notation we can define them as

$$w \in \mathbb{R}^{n_x} \text{ and } b \in \mathbb{R} \quad (2.5)$$

Finally, we can not forget the activation function σ . This function will determine the output for a particular neuron. There are many activation functions presented in section 2.2.3. Therefore, with all elements described, we can define a prediction as follows:

$$\hat{y} = \sigma(w^T x + b) \quad (2.6)$$

Also, Equation 2.6 can be represented as in figure 2.3. The way to know how far is the prediction from the real label of a particular training sample is using a loss function L

$$L(\hat{y}, y), \text{ with } L \in \mathbb{R} \quad (2.7)$$

One of the more used loss function is cross-entropy described in section 4.1.4. In addition, we can define a cost function J to measure the average performance over all the

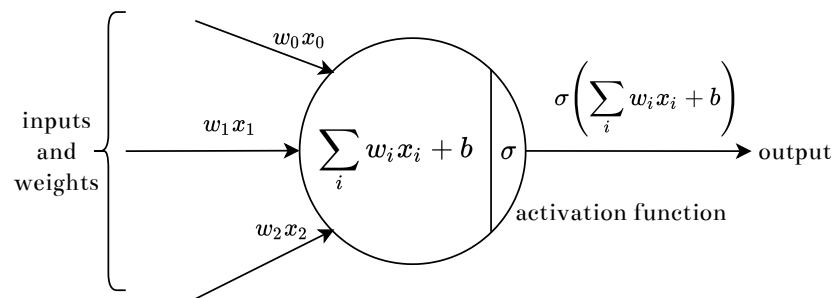


Figure 2.3: A basic ANN composed by inputs, weights, bias, activation function and the output. Note that in the neuron is performed the sum of the product between the vector x and the weights b and the the activation function is applied to generate the output.

training samples taking into account each loss function per sample:

$$J(w, b) = \frac{1}{m} \sum_{i=0}^m L(\hat{y}^{(i)}, y^{(i)}) \quad (2.8)$$

The main objective of the training is to try to keep the cost function lower as possible. In order to get this, it is necessary to update the weights w and bias b parameters in such a way that the cost function converges to a global minimum. There are many optimizers to converge the cost function, such as the ones described in section 4.1.4. The most simple is gradient descent, and it is going to be used to show how the parameters are updated. For example, suppose you define a e number of epochs to do the training, and on each epoch, the parameters w and b will be updated. Therefore, we can define the gradient descent optimization as follows:

Repeat e times {

$$w := w - \alpha \frac{\partial J(w, b)}{\partial w} \quad (2.9)$$

$$b := b - \alpha \frac{\partial J(w, b)}{\partial b} \quad (2.10)$$

}, where α is the learning rate.

During training, two principal stages are performed: forward propagation and backpropagation.

1. **Forward propagation:** A stage where the prediction (output) of the neural network is computed (equation 2.6), i.e., perform computing from the input layer to the output

layer. Before starting this stage, it is preferable to initialize the parameters w and b with random numbers.

2. **Back propagation:** Stage where the update of the parameters w and b is performed. As the cost function is computed at the final of the forward propagation stage, backward implies computing the “gradients”, i.e., computing the derivatives to update the parameters as is shown in the gradient descent optimization along with the neural network from the last layer to the first layer.

Surely, the way to represent the training process in this work is quite simple but no less useful because the same concepts can be extrapolated to shallow and deeper ANNs due to these basic ideas are the same used to construct ANNs.

2.2.3 Activation functions

As the section above shows, the activation function is an essential part of ANN training because it produces a neuron’s output to feed the next layer. It is common to use non-linearity functions to predict real phenomena better than linearity functions. Then, we are going to introduce some non-linearity activation functions.

Sigmoid activation function

This function is one of the most used in ANN. It is defined as follows:

$$f(x) = \frac{1}{1 + e^{-x}} \quad (2.11)$$

Since the output of the sigmoid function is between 0 and 1, it is very used to give probabilities in the last layer, i.e., the layer in charge to make the prediction. In addition, it is helpful to normalize the output of the neurons.

Hyperbolic tangent activation function (\tanh)

It is similar to the sigmoid function in its definition and its graph, but it has some advantages. Let us define this function:

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (2.12)$$

Note that the range of the output of the function is between -1 and 1, i.e., data is centering to 0, and, as a result, the learning in the next layer will be faster and less complex. In addition, if there is a negative input in the neuron, the *tanh* function will highlight it even more in inhibiting the activation of the said neuron. A disadvantage of this function is the presence of small gradients leading to slow convergence of the cost function when the inputs are large or small. This function is generally used in the middle layers of an ANN, while the sigmoid function is used in the final layer to make predictions.

Rectified linear unit activation function (ReLU)

ReLU activation function is specially used in deep neural networks and CNNs because it prevents the gradient saturation problem. ReLU is defined as follows:

$$f(x) = \begin{cases} \max(0, x) & , x \geq 0 \\ 0 & , x < 0 \end{cases} \quad (2.13)$$

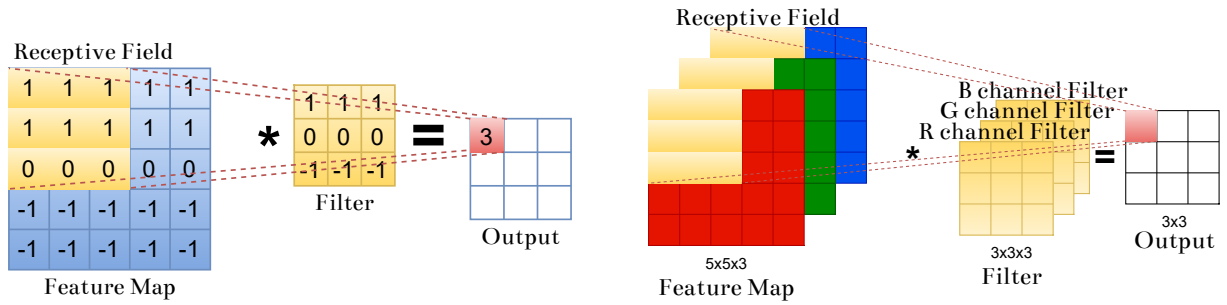
As all negative inputs will be mapped to 0, the computing is faster than other activation functions. However, if the input is negative, the derivative in the backpropagation will be zero, meaning that the neuron will not contribute to the training. This problem is called dying ReLU, and it can be solved with the use of the LeakyReLU activation function.

2.3 Convolutional neural networks

2.3.1 Convolution in CNN

Convolution is the main idea behind CNN. Convolution is an output matrix in which each element is the sum of the element-wise product between the pixels of a receptive field and the values of the filter, which are the weights.

Mathematically, the element-wise product of two matrixes can be defined as follows:



(a) Sum of the element-wise product between a receptive field and a filter (both in yellow). The result is written in the red square in the output feature map.

(b) Convolution in volumes. An RGB image is convolved with a 3D filter. There is a filter per channel and the output is a 3x3 feature map.

Figure 2.4: Examples of 2D and a volume convolution. In this particular case, the volume convolution is 3D.

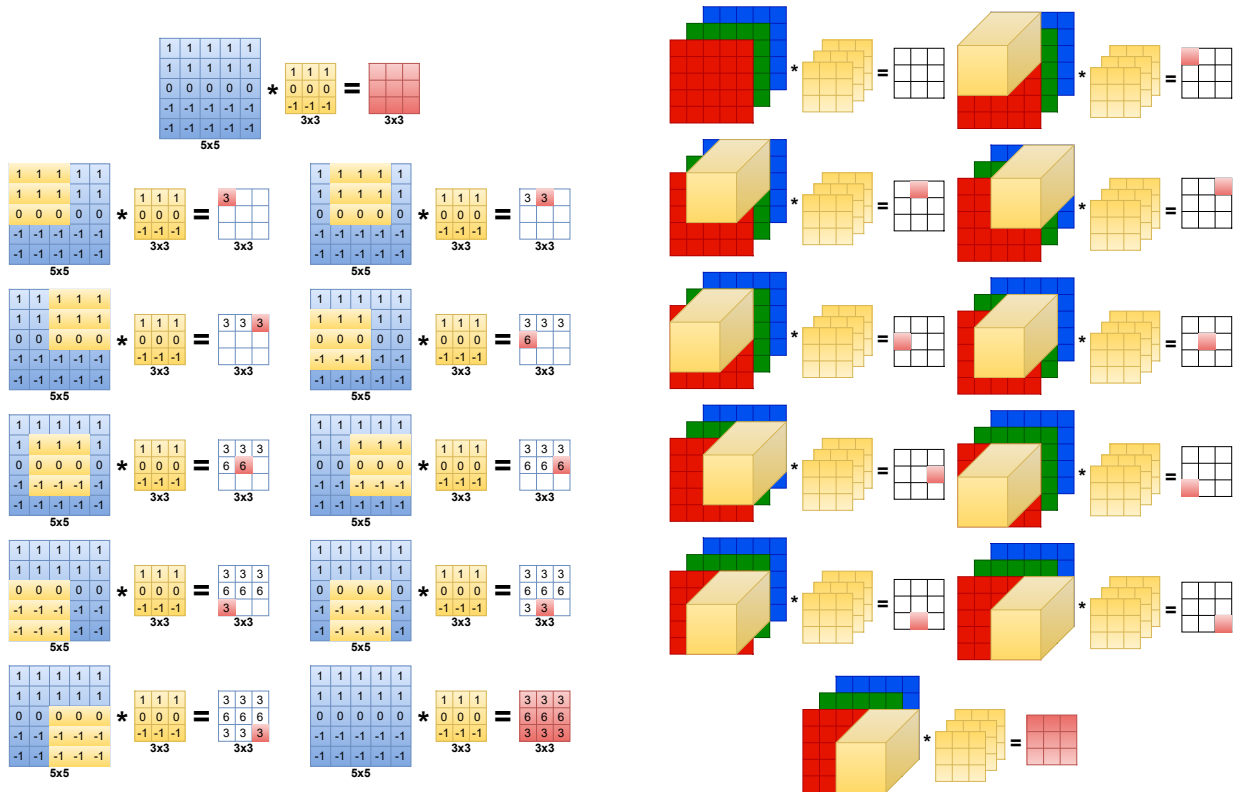
$$\begin{bmatrix} u_1 & u_2 \\ u_3 & u_4 \end{bmatrix} \odot \begin{bmatrix} v_1 & v_2 \\ v_3 & v_4 \end{bmatrix} = \begin{bmatrix} u_1 \cdot v_1 & u_2 \cdot v_2 \\ u_3 \cdot v_3 & u_4 \cdot v_4 \end{bmatrix} \tag{2.14}$$

Taking the idea of equation 2.14, Figure 2.4a shows the sum of element-wise product between a particular 2D receptive field and a filter. For simplicity, the 2D feature map is a gray-scale image. Take into account that the size of the receptive field should be the same as the filter. In this case, we have a $n \times n$ feature map and $f \times f$ filter, where $n = 5$ and $f = 3$. The dimension of the output will be $(n - f + 1) \times (n - f + 1)$, therefore 3×3 . The $*$ symbol means convolution. The operation made in Figure 2.4a is

$$(1 \cdot 1) + (1 \cdot 1) + (1 \cdot 1) + (1 \cdot 0) + (1 \cdot 0) + (1 \cdot 0) + (0 \cdot (-1)) + (0 \cdot (-1)) + (0 \cdot (-1)) = 3 \tag{2.15}$$

As the filter slides through the feature map, in figure 2.5a we can see all the convolution of the complete 2D feature map with the filter.

The example in Figure 2.5a shows a simple case, but in reality, there are a couple of concepts related to convolution. These concepts are padding and stride.



(a) A complete 2D convolution is performed. The filter slides across the feature map and each sum of the element-wise product is located in the corresponding space in the output feature map.

(b) A complete 3D convolution is performed. The volume filter slides across the volume feature map, and each sum of the element-wise product is located in the corresponding space in the output feature map.

Figure 2.5: Examples of complete 2D and a 3D volume convolution.

Padding

If we see the sliding filter through the feature map, we can observe that the external pixels of the features maps are less used than the inner pixels. However, these external pixels have valuable information. In addition, as the convolution es performed, the size of the feature maps decreases. For both reasons, we can add extra zeros columns and zeros rows to the features maps' external pixels to avoid wasting valuable information and keep the size of the feature maps. The value of padding is represented by the letter p . There are two types of padding:

1. **Valid padding:** The valid padding means no padding, i.e., no extra columns and rows. On this case, the output will be $(n - f + 1) \times (n - f + 1)$. When we define a model in some framework like Tensorflow or Torch, we should specify the valid

padding because we need to say to the model that all values of the feature maps are valid for the convolution; otherwise, the model will not take into account the more external values.

2. **Same padding:** Adding padding with the purpose of keeping the size of the input. For this particular case, the filter size is $f \times f$ with f usually odd. Consequently, the formula to find the value of the padding is $p = (f - 1)/2$. In Figure 2.6 we can see a convolution with $p = 1$.

Stride

As we can see in the examples, the convolution slides the filter across the feature map at every column and row. However, what happens if we want the filter to move two columns/rows simultaneously? or three? Here comes the necessity to define a value to the step size through the columns and rows. This value is known as a stride. In Figure 2.7 we can see a convolution with padding $p = 1$ and stride $s = 2$.

Finally, to find the the dimension of the output, we need to use the formula described in Equation 2.3.1. This formula uses the floor operator.

$$\left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \times \left\lfloor \frac{n + 2p - f}{s} + 1 \right\rfloor \quad (2.16)$$

Previously, the convolution has same padding, i.e., no padding with $s = 1$ but, if we like to perform same padding with stride, the formula is as follows:

$$p = \frac{s(n - 1) - n + f}{2} \quad (2.17)$$

Before, we considered the most straightforward case with 2D feature maps and 2D filters, which can be understood as a convolution over a grayscale image. However, it is common to work with RGB images in real applications. This means that the CNN should be able to take as input 3D images and make convolution over them.

In Figure 2.4b, we can see that both the feature map and the filter are three-dimensional, where the feature map represents an RGB image with dimension $5 \times 5 \times 3$, i.e., an image

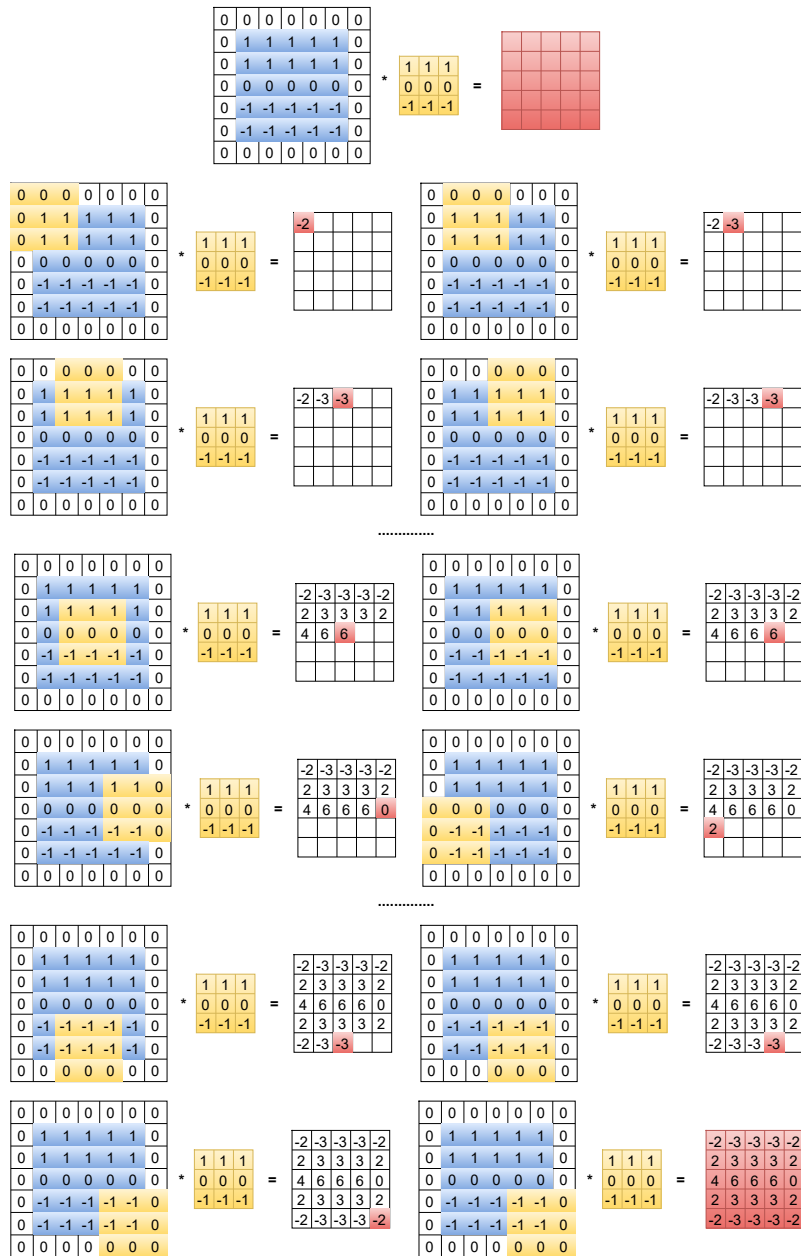


Figure 2.6: A summary of a convolution with padding is presented.

of 5×5 pixels with three channels. If it is taken as a volume, the dimensions represent the width, the height, and the depth, respectively. As the number of channels is three, the convolution needs a filter per channel. Therefore, the filter dimension is $3 \times 3 \times 3$. Finally, the output will be of dimension 4×4 .

The convolution follows the same idea applied for 2D images but with the difference that now it should work with volumes. Consequently, the 3D filter slides across the RGB image to find the sum of the element-wise product between the filter and each receptive

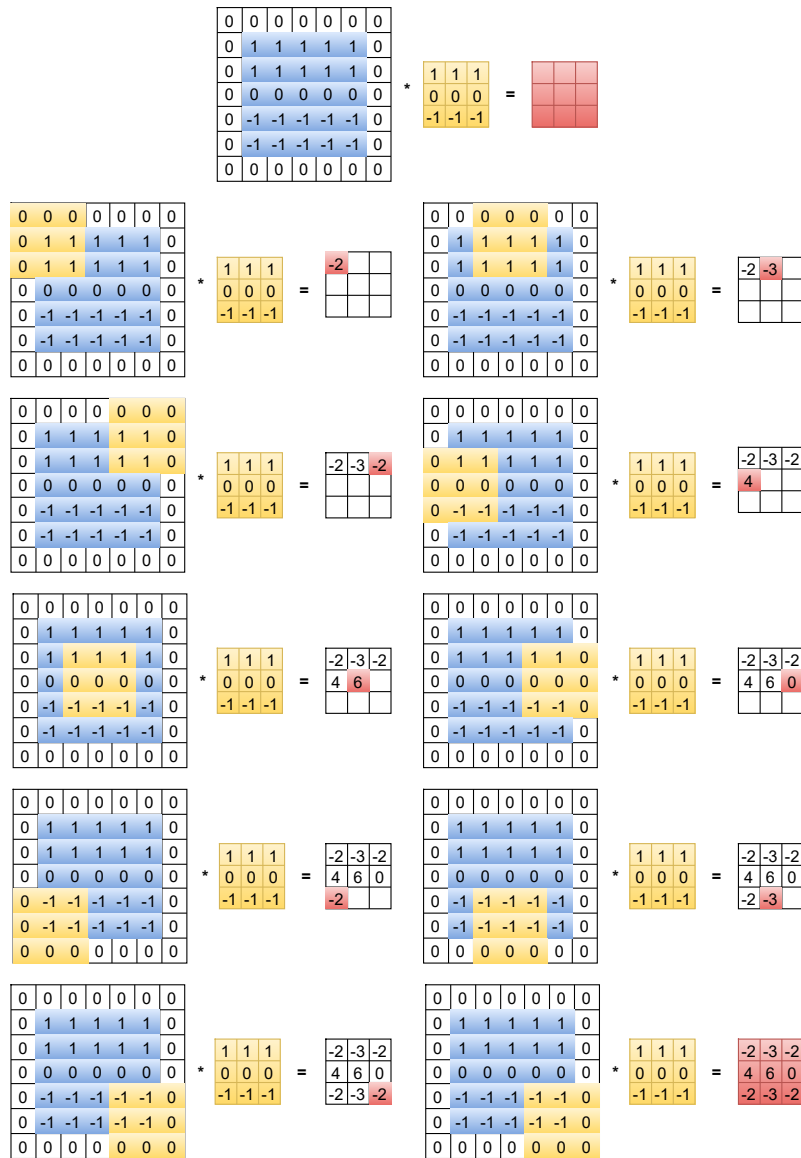


Figure 2.7: A convolution with padding and stride is presented.

field. As the filter dimension is $3 \times 3 \times 3$, there are 27 numbers, and each one should be multiplied by its respective one in the receptive field. The sum of the element-wise product fills the respective space in the output. Figure 2.5b represents a complete convolution in volumes.

In Figure 2.5b we only use one type of filter (for example, it can be specialized in detecting vertical edges). However, in practice, it is desirable to pass not just one but multiple filters at a time in order to extract more information about the image. As a result of multiple filters, we will get a stack of outputs, one output by filter as in Figure 2.8.

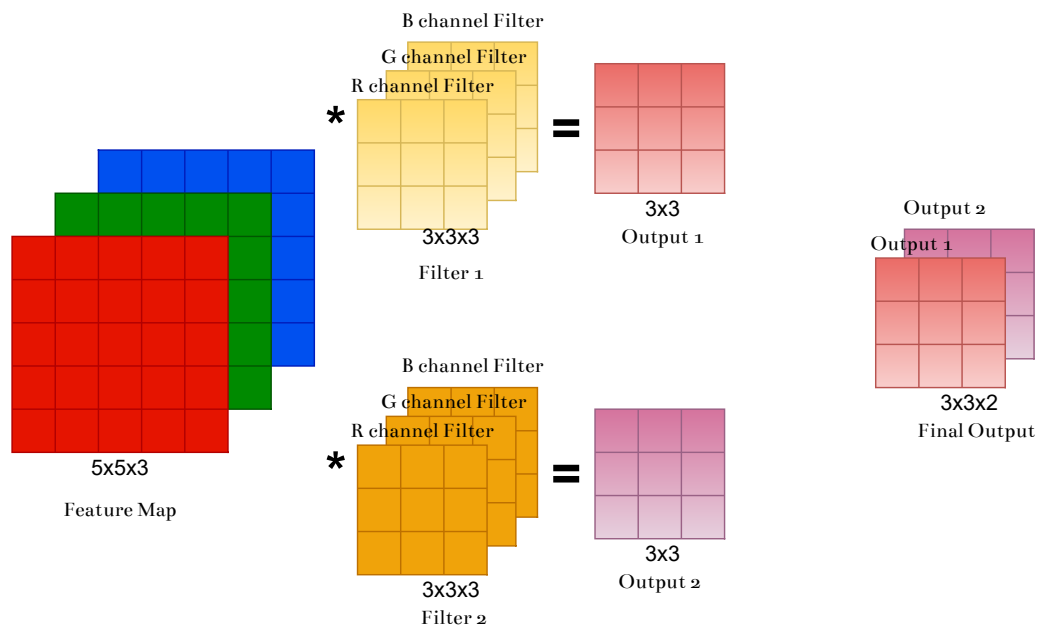


Figure 2.8: A convolution with two filters is presented. The outputs of each filter are stacked to form the final output.

2.4 CNN architecture

CNNs are composed of three principal layers: convolutional (CONV) layers, pooling (POOL) layers, and fully connected (FC) layers.

2.4.1 Convolutional layers

In Figure 2.8, there is a CONV layer with multiple filters. For simplicity, the addition of the bias b and the application of the nonlinearity for each output have been ignored. The bias is a real number added to each value of an output from a particular filter. Finally, the nonlinearity function is applied to this new output. A common choice for the nonlinearity function is the ReLU, defined as follows:

$$R(z) = \max(0, z) \tag{2.18}$$

The equation before says that all negative numbers will be changed by 0. Therefore, a better representation of a CONV layer can be seen in Figure 2.9.

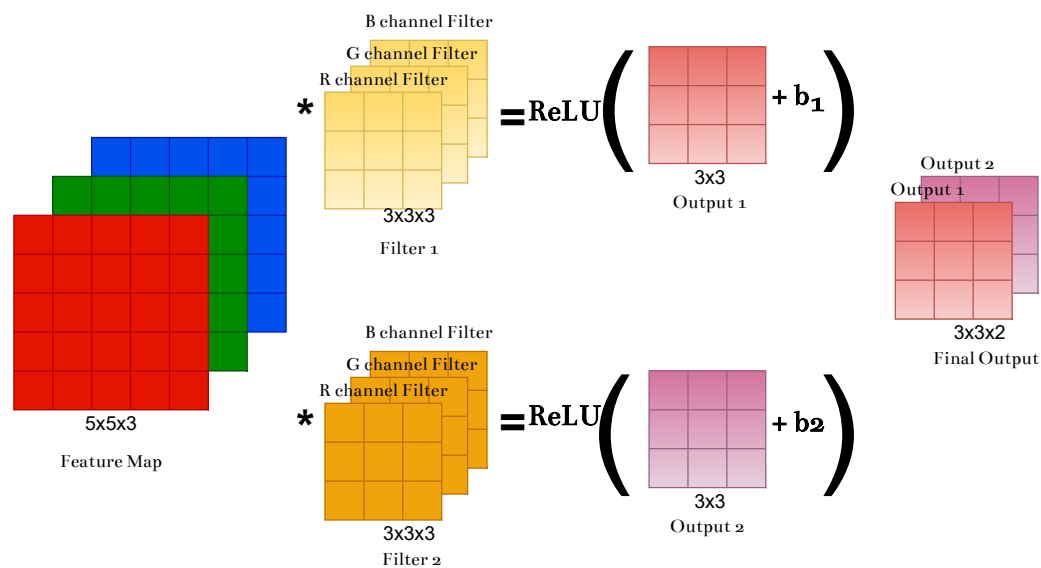


Figure 2.9: A convolution with two filters is presented. The bias term is added to the each output and then the ReLU function is applied. The final result is stacked in order to form the final output.

Number of parameters in one CONV layer

As a consequence of the complete representation of a CONV layer, a question arises: how many parameters are in a CONV layer? Remember that the CONV layer parameters are the values of the filter, most commonly known as weights. The weights are adjusted in the backpropagation stage. Not only the weights are parameters but the bias values. As we can appreciate in Figure 2.9, a bias value is needed per filter. Consequently, for this particular case, there are two $3 \times 3 \times 3$ filters. Each filter has 27 weights and a bias value leading to 28 parameters. Since there are two filters, this particular CONV layer's total number of parameters is 56.

2.4.2 Pooling layers

CNNs use POOL layers to reduce the space of the feature maps. Reducing the size of a feature map reduces the number of trainable parameters and, therefore, the computational overhead caused by the network's training. POOL layers have two principal hyperparameters: the size of the pooling layer filter defined by f and the stride defined by s . The most common value for both hyperparameters is 2. Also, padding can be used, but its use is rare. For example, in Figure 2.10, we can find the two types of pooling: max pooling and

average pooling.

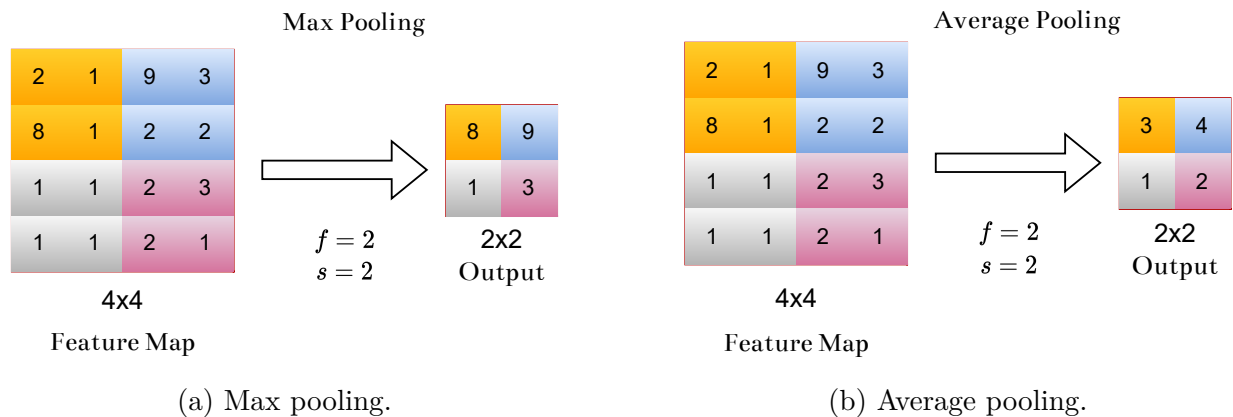


Figure 2.10: Two types of figures are presented: max pooling and average pooling.

Figure 2.10a shows a max-pooling operation over a 4×4 feature map. With f and s equal to two, max-pooling takes the maximum value of each group and fills the corresponding space in the output feature map. The maximum value indicates that there is a strong feature on this pixel that helps to make the right object recognition. On the other hand, in Figure 2.10b there is an average pooling. As its name says, it takes a region of pixels and computes the average of its values. Actually, max pooling is often used because it has been shown to work better than average pooling.

2.4.3 Fully connected layers

These layers are used in the final part of CNNs. FC layers are similar to the layers in the traditional ANNs, and therefore, each neuron is connected to all the neurons from the previous layer. If the previous layer is a volume, a squeezing of the volume is performed, and the dimension is converted to $1 \times 1 \times n$, where n is the depth of the volume. The final FC layer will be of dimension $1 \times 1 \times n$, with n in this particular case as the number of classes. The neurons on this layer are in charge of giving each class scores to make the final classification.

The overall architecture of a CNN can be seen in Figure 2.11

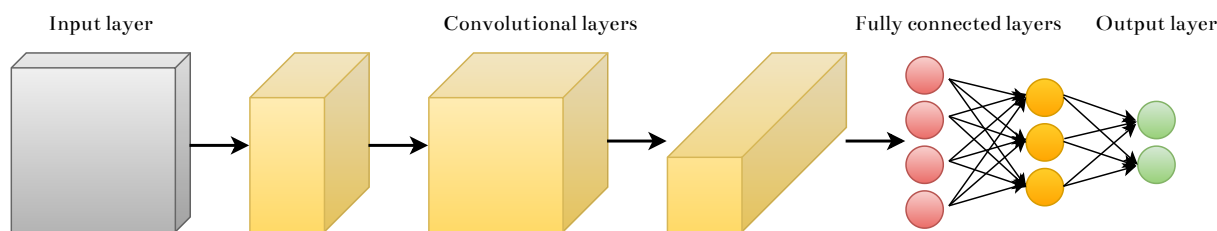


Figure 2.11: A CNN is presented. It is composed by the input layer, the convolutional layers, the fully connected layers and the output layer. Pooling layers are implicitly next to the CONVs.

2.5 Convolutional neural networks from the beginning to VGG

CNNs are a particular class of ANNs widely used in computer vision systems and image processing. The base of CNNs is a mathematical operation called “Convolution”; therefore, their name. In the last decade, notably, CNNs have been one of the most studied areas leading to the fast development of new states of the art in different image processing tasks. Thus, the CV field became one of the most developed in AI. Contrary to what many people may think, CNNs have been used since the eighties, not in the same way we know them today, but with the main concepts. Furthermore, if we go further back, we can notice that CNNs inspiration came from biology. Next, we will present some of the most relevant works related to the development of CNNs over time.

2.5.1 Simple and complex cells

David Hubel and Torsten Wiesel, in their work called “Receptive Fields of Single Neurones in the Cat’s Striate Cortex” [26], present two types of cells: simple and complex. They said both are used in pattern recognition in human vision, but they have a fundamental difference. Simple cells can detect edges and bars in a specific orientation in a specific part of the scene. Complex cells can do the same but with an extra advantage: they can detect edges and bars in a specific orientation but any location of the scene. We can call the location receptive field. For example, a simple cell can respond to vertical bars in the superior part of the scene. However, a complex cell can detect all the vertical bars in the superior, inferior, or central part of the scene. This property of complex cells is known as

“spatial invariance”.

Hubel and Wiesel, in the second part of their study in 1962 [27] showed that the spatial invariance is gotten by making a sum of the output of the simple cells, all of them with a particular orientation but with different receptive fields. In this way, the complex cell acquired the capacity to respond to any receptive field from anywhere. This simple concept, i.e., summing individual outputs from simple cells to give complex cells the ability to see anywhere, is the basic idea in CNNs.

2.5.2 Neocognitron

In 1980, the Japanese scientist Kunihiko Fukushima, in his work called “Neocognitron: A self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position” [28] presents a neural network applied to pattern recognition inspired in the visual nervous system described by Hubel and Wiesel. Fukushima designed two types of layers: the first one is composed of “S-cells,” and the second layer is composed of “C-cells”, both with the same function as the simple and complex cells described by Hubel and Wiesel years before. The cells are nothing more than mathematical operations. He made a simulation in a digital computer to present a set of stimulus patterns to the neural network. The first layer was composed of S-Cells and the last two layers of C-cells. A particular stimulation produces an output in a unique C-cell of the output layer, and this C-cell only will respond to this particular stimulation. The principal idea behind the neural network is to reflect the concept of going from simple to complex applied to an intelligent model for pattern recognition.

2.5.3 From invariant recognition to convolutional neural networks: LeNet-5

The Ph.D. Yann Lecun presented the first CNN called LeNet-5 applied to document recognition. It was developed from 1989 to 1998 with minor improvements each year applied to different tasks like recognizing handwritten characters, special-purpose chip (ANNA), and reverse time-delay neural net. In 1989, Lecun became interested in how to solve the problem of invariant visual perception. He demonstrated that local connections and shared

weights improved the neural network's performance in general. He called "Convolutional Neural Networks", the networks optimized by local connections and shared weights. In 1993 Lecun made the presentation of the early version of LeNet for character recognition. In the subsequent years, Lecun continued conducting some studies and experiments. Finally, in 1998 he presented his work named "Gradient-Based Learning Applied to Document Recognition" [29] where he designed a multilayer neural network trained with a backpropagation algorithm applied to standard handwritten digit recognition. The neural network was called LeNet-5, and also, it was inspired by the Neocognitron. Lecun preserves the idea presented by Fukushima, adding simple features to more complex features using complex cells to enhance the performance of handwritten character recognition. The CNN was trained on the MNIST dataset, and it beats other digits' contemporary classification methods. The architecture of LeNet-5 is Conv-Pool-Conv-Pool-FC-FC layers with a total of sixty million parameters, and it can be seen in Figure 2.12.

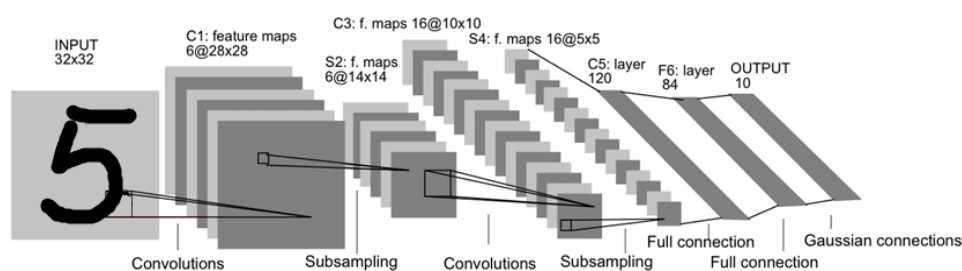


Figure 2.12: LeNet-5 architecture for digits recognition [29].

2.5.4 Convolutional neural networks from 1998 to 2010

In this period, computer sciences continue to apply simple CNNs to different simple tasks. In general, the progress of CNNs was slow due to limited available hardware and data to train the models. As time passed, cheaper cameras and cell phones appeared, making more images available to form extensive datasets. In 2005 appears, the PascalVoc dataset [30] with twenty thousand images with twenty different tasks. Five years later, i.e., in 2010, the PascalVoc team feat other collaborators presented the ImageNet dataset [31]. Nowadays, this dataset has approx 15 million images. At the same time, more powerful CPUs and GPUs appears oriented to train CNNs. The improvements in datasets and hardware became CNNs more interesting for researchers. In 2010, Dan Claudiu and Jurgen

Schmidhuber, in their work called “Deep Big Simple Neural Nets Excel on Handwritten Digit Recognition” [32] present the first neural network trained on a GPU. The GPU was the NVIDIA GTX 280, and nine layers formed the neural network. Therefore, a new era started for the CNNs.

2.5.5 AlexNet, 2012

In 2012, Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton presented their paper called “ImageNet Classification with Deep Convolutional Neural Networks” [33]. Their design is an eight layers CNN composed of five CONV layers (some of them followed by POOL layers) and three FC layers. The final FC layer has one thousand outputs, the number of classes of the ImageNet dataset. AlexNet achieved a 17% error rate on this dataset, an important improvement concerning the previous state of the arts. It was the first time a deep CNN with sixty million updatable parameters was presented. Some of the most important features of this network are the presence of non-saturating neurons, very efficient implementation of the convolutional operation, dropout method to avoid overfitting in the FC layers, ReLU as non-linearity for the activation, and overlapping max pooling. The training was able using NVIDIA GTX 580 GPUs. It is important to take into account that before AlexNet, the standard activation function was $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$. Both are known as saturating nonlinearities and they are slower than non-saturating nonlinearity $f(x) = \max(0, x)$. Therefore, the neurons called ReLUs are faster than other ones computed with for example $f(x) = \tanh(x)$. The AlexNet architecture can be appreciated in Figure 2.13.

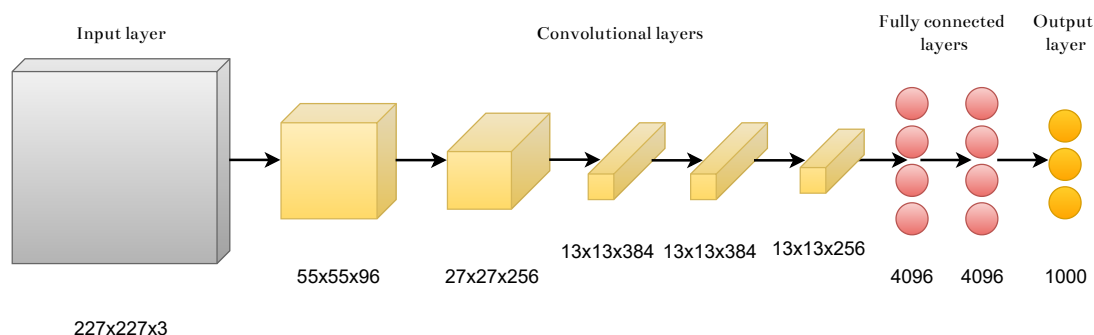


Figure 2.13: AlexNet architecture with eight layers: five convolutional layers and three fully connected layers.

All these improvements and the performance of AlexNet meant the beginning of a new era for CNNs. With more large available datasets, and faster GPUs, the need to apply CNNs to the real world induced the look for new models with more advanced optimized techniques.

2.5.6 VGG

Karen Simonyan and Andrew Zisserman from Oxford University 2014 presented a work called “Very Deep Convolutional Networks for Large-Scale Image Recognition”[34] which focused on the design of the CNN. The main idea is to show that the depth of a CNN is critical to get better accuracies. They increased the depth of the CNN by adding more CONV layers. To avoid the loss of the spatial size volume on each layer, they use 3×3 filters in stacked CONV layers of two and three. It is different from AlexNet, where we have 11×11 filters. Putting staked CONV layers of two and three CONV layers with 3×3 filters, we can get a similar behavior as using large receptive fields like 5×5 and 7×7 , respectively but with the advantage of getting depth in the model. In addition, we can use one ReLU for each CONV layer. The architecture is generally homogeneous, doubling the number of filters after each maxpool layer. Finally, they present the following models: VGG-11, VGG-13, VGG-16, and VGG-19. The architecture won the ImageNet challenge in 2014 with an error rate of 7.3%, becoming state of the art. Some facts to consider on VGG are an increase in the number of parameters up to 140 million and the size of the model to 46.6MB (AlexNet has only 1.9MB). Finally, an important fact presented in the paper is that VGG-19 only performs a minimal improvement in the accuracy of the model; therefore, after VGG-16 the accuracy of the model stays saturated. This fact became the starting point for new researchers to give a solution to the problem that from VGG-16 it was impossible to increase the accuracy of increasing the depth of a CNN. The architecture of VGG-16 can be seen in Figure 2.14.

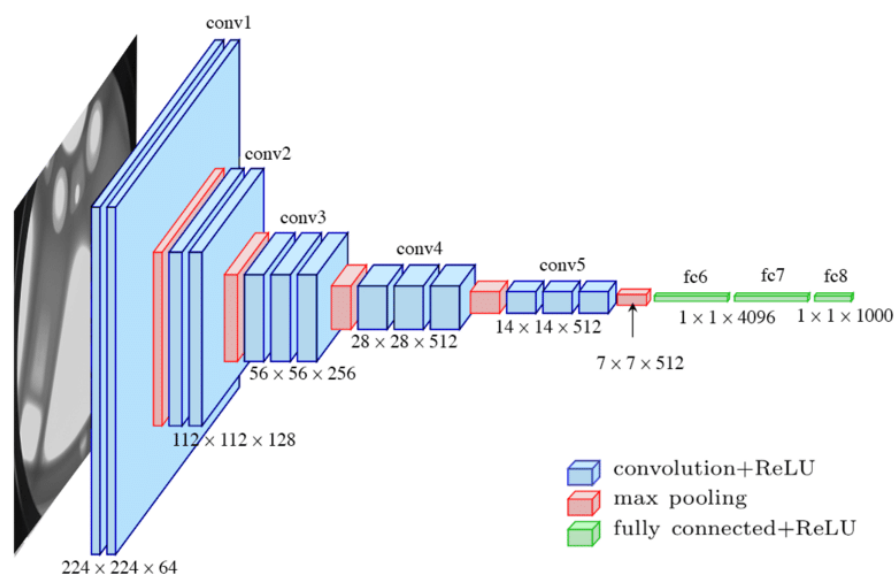


Figure 2.14: VGG-16 architecture [35].

2.6 ResNet: the way to effective deeper CNNs

2.6.1 The problem

As we said before, the CNNs faced a principal problem: deeper architectures began to be saturated in terms of accuracy. It was not enough anymore to add more CONV layers to the network to get more accuracy even when Karen Simonyan and Andrew Zisserman [34] had shown that depth is a critical factor in the design of an accurate CNN because deeper layers can extract complex features. As a consequence, Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, in their work called “Deep Residual Learning for Image Recognition” [36] addressed the following question: “Is learning better networks as easy as stacking more layers?”. It was clear that stacking more and more layers was not useful anymore because it does not increase, and contrary, it begins to drop at a certain point in the training.

Some facts that showed the above behavior were that CNNs were hampered by problems related to vanishing/exploding gradients, leading to avoiding convergence. However, vanishing/exploding gradients were solved in some works using normalized initialization and intermediate normalization layers, allowing the models to converge for stochastic gradient descent (SGD) and backpropagation.

Although resolving the problem of vanish/exploding gradients, He et al. [36] observed

that the accuracy in the training dataset dropped in models with more layers. The graph presented in Figure 2.15 shows this phenomenon.

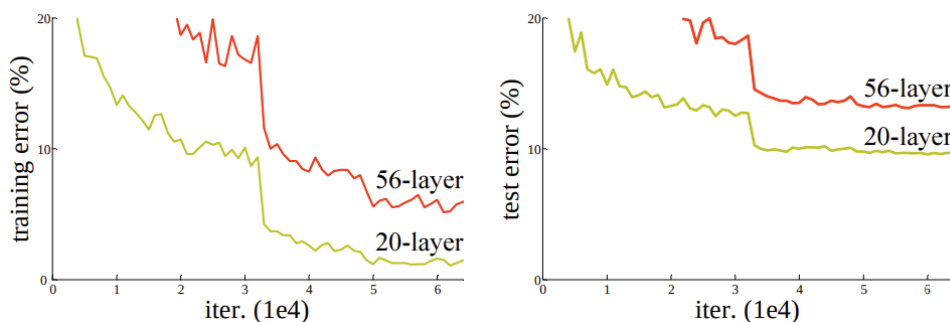


Figure 2.15: Deeper plain networks error rate [36].

Note that the authors of the paper began to distinguish between CNNs given the name of “plain network” to the ones similar to the VGG architectures. Continuing with the idea, they noticed that when deeper models were able to start converging, a degradation problem appeared. This problem leads to the saturation of the accuracy in the training dataset, but they affirm it is not caused by overfitting. They adduced that the graph can appreciate a high training error, not only a high test error. In other words, if it was overfitting, there should only be a high test error, not a high training error.

Deeper CNNs having higher training errors than shallower CNNs is counterintuitive. The theory holds that deep CNNs can extract complex information from the data. Therefore, very deep CNN should be more accurate. They make the following assumption. Suppose there are two plain CNN of deep a and b layers, respectively, with $a < b$. If the error of the first CNN, i.e., the CNN of depth a , is e , the error of the deeper CNN should be at least the same or better as the first one. Of course, the deeper network is composed of $a + (b - a)$ layers. As the error of the shallow network is e , the rest of the network, i.e., $(b - a)$ should keep the same error as an identity function (if the network does not need more learning) or smaller because theoretically, this part of the network is in charge to learn complex features and therefore, get better accuracy. However, the reality is that the error begins to increase in the $(b - a)$ part; therefore, a worse accuracy is gotten shown in Figure 2.16.

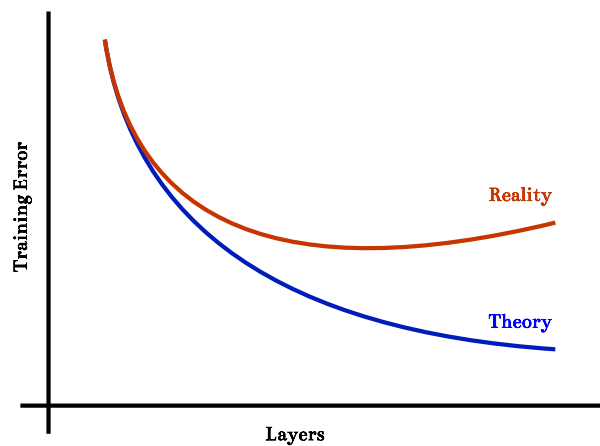


Figure 2.16: Deeper networks error in the training dataset. In theory, as more layers, less error rate. However, from a certain number of stacked layers, in reality, the error rate begins to increase.

2.6.2 Deep residual learning framework as a solution to the degradation problem

He et al. [36] presented ResNets as a solution to the degradation problem. They realized that the deeper layers were not identity mapping. They called “identity mapping” the ability to maintain at least the same error across the deeper layers like an identity function avoiding hurt performance. Therefore, it would be the desired underlying mapping. Instead of waiting for the layers to fit the desired underlying mapping, the authors explicitly let the layers fit a residual mapping. Figure 2.17 and some mathematical formulas let us explain this new approach.

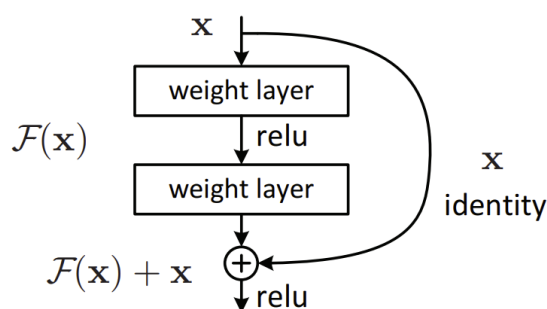


Figure 2.17: Residual learning: a building block [36].

Lets denote the desired underlying mapping of a few stacked layers as $\mathcal{H}(x)$ with x as the inputs to the first of these layers, and $\mathcal{F}(x) = \mathcal{H}(x) - x$ as another mapping fitted

by the same stacked layers. $\mathcal{F}(x)$ can be defined as the residual mapping. Thus, the original mapping is recast to $\mathcal{H}(x) = \mathcal{F}(x) + x$. It is better to try to convert the residual function $\mathcal{F}(x)$ to zero and add to it the identity x rather than let the stacked layers approximate the original $\mathcal{H}(x)$ by themselves. As we see in Figure 2.17, x is the identity copied from the input of the first stacked layer to the last. It is called shortcut connections or skip connections, and they are an important part of addressing the desired mapping and avoiding layers that can hurt performance.

A mathematical formulation to define a building block is as follows:

$$y = \mathcal{F}(x, \{W_i\}) + x \quad (2.19)$$

On this formula, x and y are the input and output vector of the stacked layers and

$$\mathcal{F}(x, \{W_i\}) \quad (2.20)$$

represents the residual mapping. The building block presented on Figure 2.17 can be formulate as

$$\mathcal{F} = W_2 \sigma(W_1 x) \quad (2.21)$$

with σ as ReLU activation function. Here, biases are omitted to avoid complexity. Finally, $\mathcal{F} + x$ represents the operation performed by the shortcut connection and element-wise addition. Then, the second nonlinearity is performed. Graphically, this building block can be represented as in Figure 2.18.

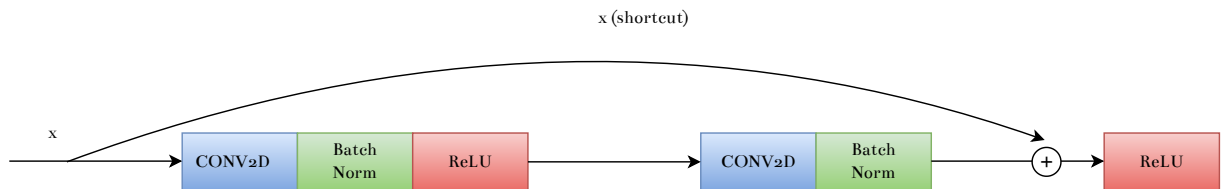


Figure 2.18: Residual learning: a building block.

This building block is composed of two stacked layers with an identity shortcut. On Equation 2.19, \mathcal{F} and x must be of the same dimension but it is not always possible. A solution is to perform a linear projection W_S by the shortcut to make a match between the

input and output dimensions. This is known as a projection shortcut, and the equation is as follows:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s \mathbf{x} \quad (2.22)$$

After applying the above equation, we can get the building block in a 3 stacked layer as is present in Figure 2.19.

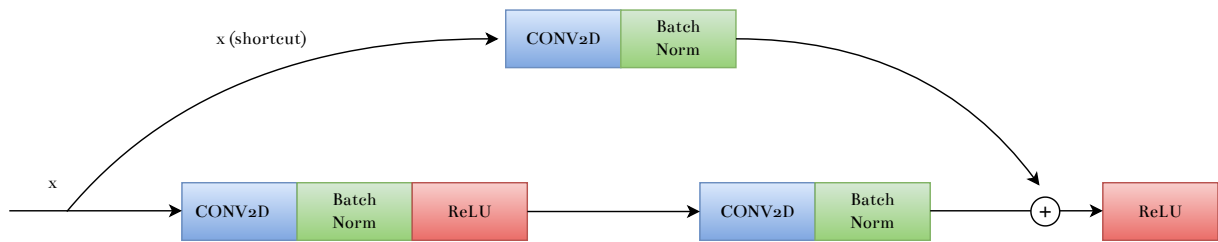


Figure 2.19: Conv Block.

Any advantage is gotten if a building block is composed of a unique layer. For this reason, the most common building blocks are composed of two and three layers. Next, a Resnet with 34 layers is presented in Figure 2.21 with its counterpart, a 34-layer plain network, and VGG-19. The objective of the Figure 2.21 is to show the difference between a residual and plain network as well the identity shortcuts (solid lines) and the projection shortcuts to make a match between the input and output dimensions (dotted lines). It is essential to say that another way to match input and output dimensions in a building block is by adding extra zeros without introducing extra parameters. In both cases, a stride of two is used across features maps.

After applying the residual blocks in deeper CNNs, a significant improvement was made. Figure 2.20 shows a comparison of the error rate between plain networks and CNNs.

2.7 Training a CNN

Training a CNN can be quite a challenge because it depends on several factors like the nature of the problem, the availability of sufficient data, a DL model capable to performs the learning in a proper way, some hyperparameters which need to be fine tuned, complexity of the task to be solved, the hardware infrastructure to make the trainings (specially

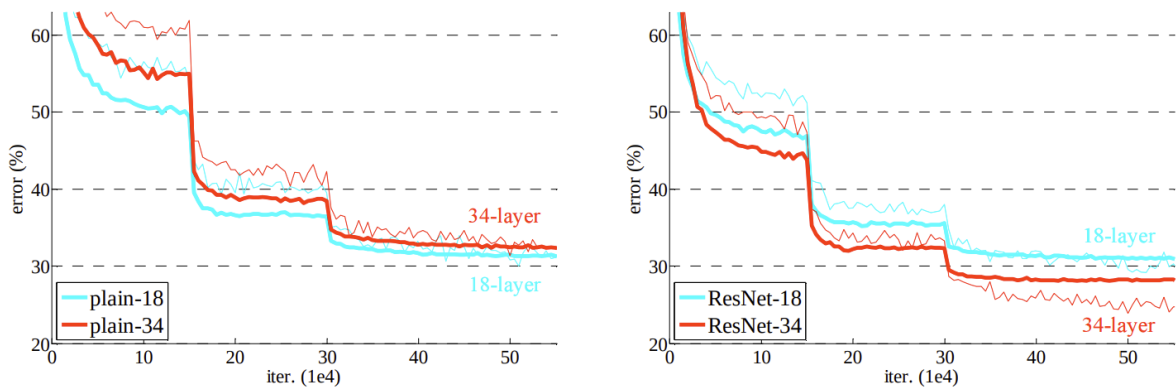


Figure 2.20: ResNet error improvement in the training dataset with deeper layers [36].

powerful GPUs and storage capacity), and other factors that may appear in the training and testing of the CNN. In general, the training of a CNN follows the same guideline presented in section 2.2.2 which refers to the training of an ANN.

2.7.1 Datasets

The datasets are a fundamental part of the training of a CNN because, depending on the data, the model would be capable of learning the desired task. Basically, the dataset is a set of samples as was described in Equation 2.1.

Types of datasets

1. **Public datasets:** This dataset type is available on platforms like GitHub, Google Drive, DropBox, and Kaggle. In general, these datasets are released by the scientific community to let the computer science community give solutions to general problems like image classification, object detection, panoptic segmentation, image segmentation, etc. Thanks to these datasets, significant advances have been made in CV, such as the availability of pre-trained models to perform transfer learning. Among the most important, we can find ImageNet, CIFAR 10 and CIFAR 100, COCO, and MNIST. In addition, these datasets often are used as benchmarks to test different improvements and techniques in DL models.
2. **Custom Datasets:** They are used to train models to solve specific problems. Sometimes, we need to make a model specialized in a particular task, and there are no

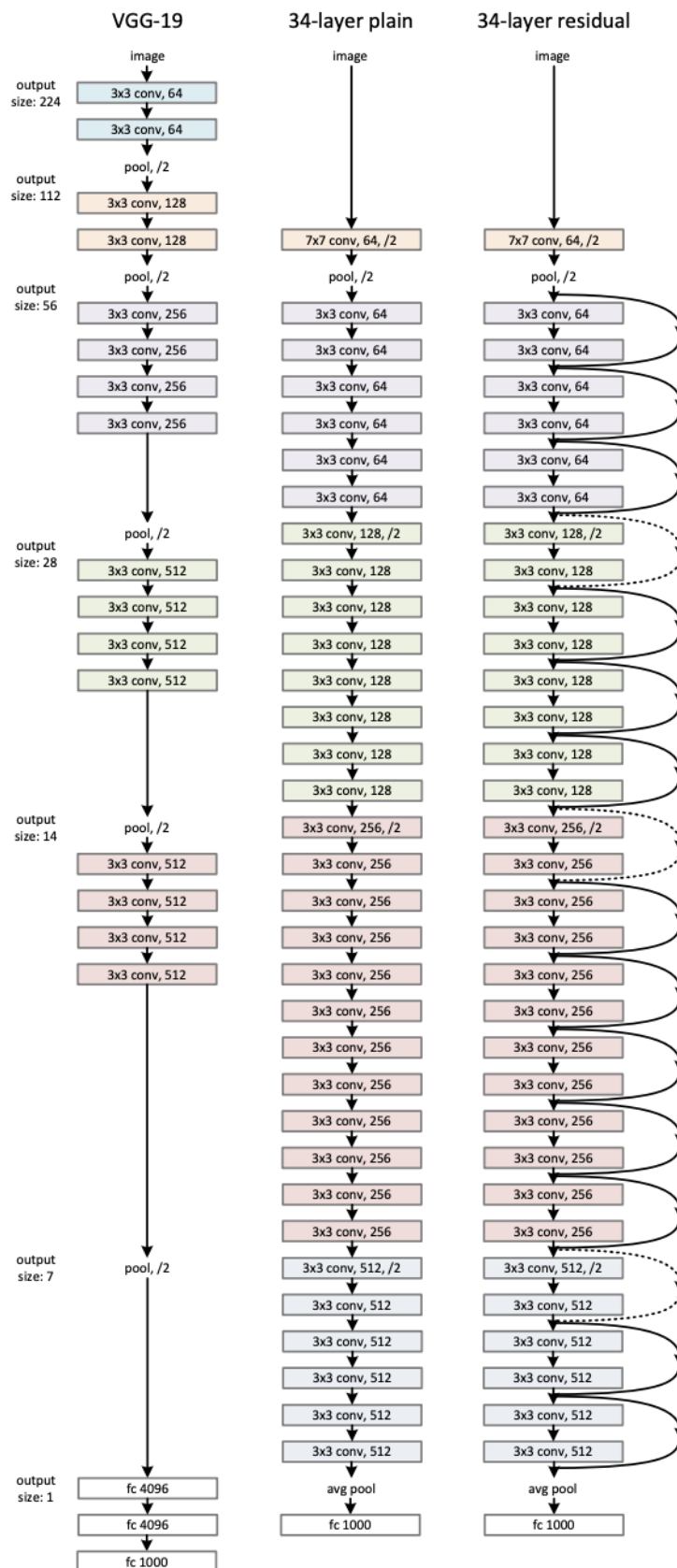


Figure 2.21: Plain and Residual Networks [36].

available public datasets to satisfy it. For example, suppose we need a model capable of recognizing dogs and their breed. For this particular task, we need to form a dataset with images of different breeds of dogs to let the model learn specific features related to the breed of dogs.

Techniques to prepare a dataset

1. **Dataset labeling:** When we construct a custom dataset, it is necessary to assign labels to the samples. This task can be difficult depending on the size of the dataset if we do not use suitable tools. For example, if we use the image viewer of Windows or Ubuntu and manually copy and paste the images to each folder class, the labeling of the dataset would be stressful. Therefore, it is preferable to use publicly available tools to make the labeling task. Some helpful tools are IBM Cloud Annotations¹ and VGG Image Annotator². However, there are cases when we need to do complex labeling. For example, imagine you need to label a dataset to identify sex in pedestrians. Some features to make a difference between males and females are personal objects and clothes. In this particular case, we need to assign many labels to the same image, and finding a public tool to perform this task can be difficult. For this reason, it is preferable to develop a custom app with an image viewer and a check options component to assign the labels. The simplest way to do this custom app can be using Python and a library like PySimpleGUI³.
2. **Dataset cleaning:** It is necessary, primarily when we use public datasets because they can have images wrong labeled, images with low resolution, or noise which can hurt the learning and performance of our model. It is common to reassign images to another class in a cleaning process and remove some of them. This task can be done using some of the tools described above.
3. **Data augmentation:** This is useful when there is not enough data to train a CNN. Some techniques applied to make data augmentation are rotations, flips, crops and random crops of the images, change illumination, and saturation of the images or a

¹<https://cloud.annotations.ai>

²<https://www.robots.ox.ac.uk/vgg/software/via/>

³<https://pysimplegui.readthedocs.io/en/latest/>

combination of them. We can identify two principal types of data augmentation:

- The data augmentation is made at running time each time a batch of images enters the CNN without storing the images. It can be done with the transforms package in PyTorch.
- The data augmentation performed before the training where each image produced by the data augmentation techniques is stored to increase the size of the dataset.

Dataset splitting

In order to train the model, it is necessary to split the dataset in three parts:

1. **Training dataset:** This dataset will contain the images used by CNN to learn features.
2. **Validation dataset:** We need a validation dataset to ensure the CNN is learning and to prevent overfitting, i.e., the model should be capable of generalizing the learning to new images that it has never been seen. The validation dataset enters the CNN at the final of each epoch. The computing of the accuracy and loss in the validation dataset compared with the accuracy and loss in the training dataset will show if there is overfitting.
3. **Test dataset:** This dataset is used to test the model. CNN has never seen the images in this dataset, and testing the model on these images is an excellent way to know if the model can generalize or not the learning to new images.

It is common to split the dataset as follows: 80% for the training dataset and 10% for both the validation and test dataset. The percentage can be changed depending on the size of the dataset.

2.7.2 Deep learning frameworks

A framework in computer science is a platform provided with the necessary tools to speed up development. In the case of DL, the frameworks will contain libraries to work with

images, download pre-trained models, do training, construct custom CNNs, analyze results and, in some cases, deploy models to production. The most used Deep Learning frameworks are TensorFlow by Google Inc., PyTorch by Facebook, Keras, MXNet, and NVIDIA TAO toolkit.

2.7.3 Training approaches

Training a model from scratch

Training a model from scratch is suitable when we have enough data to ensure the learning in the CNN. In this approach, the parameters like the weights and bias are randomly initialized, and then they are updated in the training to learn about the data. We can use standard CNNs (like VGG or ResNet) available in the repository of the framework or ensemble a custom CNN.

Transfer learning

Transfer Learning is a method based on using a pre-trained model in a new task related to the original one used to train it. A great example is the ResNet classification models available in the PyTorch Hub repository. These models were trained in the ImageNet dataset to perform classification in one thousand classes. These classes are common real-life objects like animals, means of transport, cookware, etc. Resuming the example proposed in the section of custom datasets, we can use a ResNet pre-trained model to specialize it to recognize breeds of dogs. There are two type of transfer learning:

1. **Fine tuning:** In fine-tuning, we train the pre-trained model to specialize it to the new custom task updating the parameters (weights and biases) of all or some layers. To keep the weights unchanged of some layers, we freeze them. If we want to update the weights in the new training, we should unfreeze the desired layers. The idea of freezing and unfreezing layers is the basis behind fine-tuning.
2. **Feature extractor:** In this transfer learning, we use the feature extractor provided by the pre-trained model to get meaningful features from the new dataset. In addition, a new classification layer is added to the top of the pre-trained model to take the features and make the predictions. The classification layer is trained from scratch.

The use of fine-tuning or feature extractor will always depend on the complexity of the new task. Sometimes, the feature extractor of the pre-trained model can be sufficient to make transfer learning, and sometimes it will not be sufficient, and we have to update the parameters of the layers with fine-tuning.

2.7.4 Hyperparameters

The hyperparameters are parameters selected by the user before the training, and they are not upgradeable. For example, in the training of a CNN, the principal hyperparameters are:

Number of epochs

The number of epochs is the number of times the entire training and validation dataset will pass across the CNN.

Batch size

Often, it is impossible to pass all the images to CNN in one go. Therefore, the dataset should be divided into groups of images called “batches”. As a result, the batch size is the number of images entering the CNN by batch. The batch size depends on the GPU capacity in terms of memory and computation capacity, affecting the total time of training.

Loss function

The Loss Function is in charge of indicating how far the predictions from the real labels are. The most used is cross-entropy defined in section 4.1.4.

Optimizer

The optimizer is used to make the cost function converge to a global minimum. There are many optimizers like SGD, ADAGRAD, ADAM, and RMSprop.

Learning rate α

If we remember the Gradient Descent optimization presented in section 2.2.2, we can notice the presence of parameter α . This parameter α is called learning rate. The learning rate

will determine how much to update the weights and biases in the backpropagation stage. A common value to the learning rate is $\alpha = 0.001$. In addition, sometimes, it is desirable to manage a dynamic value for the learning rate as the number of performed epochs increases to improve the adjusting of the parameters after a certain number of epochs. This means that we can decrease the learning rate value in a γ value after s number of epochs. The s number of epochs is called step size. The assigning of dynamic learning rate can be performed using a learning rate scheduler that receives as parameters s and γ . Some common choices for both hyperparameters are s as the third part of the number of epochs and $\gamma = 0.1$. Taking $\alpha^{(0)} = 0.001$ and $\gamma = 0.1$, the new learning rate $\alpha^{(1)}$ after s number of epochs will be computed as follows:

$$\alpha^{(1)} = \alpha^{(0)} \cdot \gamma \quad (2.23)$$

$$\alpha^{(1)} = 0.001 \cdot 0.1 \quad (2.24)$$

$$\alpha^{(1)} = 0.0001 \quad (2.25)$$

2.7.5 Analyzing results

The analysis of results lets us understand the performance of a CNN model once the training is over. Next, we will explain a set of tools that will allow us to analyze the performance of a model.

Confusion matrix

A confusion matrix is a graph used to visualize the performance of a model trained with supervised learning. It is similar to a table where each cell is filled with numbers corresponding to the comparatives between the true labels and predicted ones. The comparatives are the true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN). An example of a confusion matrix is presented in Figure 2.22.

Metrics

The metrics are formulas used to measure the performance of a particular model based on the results obtained in the test dataset. Metrics show us how good or bad is a particular

		Actual Values	
		Positive	Negative
Predicted Values	Positive	TP	FP
	Negative	FN	TN

Figure 2.22: A confusion matrix for two classes is presented. We can see the TP, TN, FP and FN.

model overall and per class. In general, metrics use the information provided by the confusion matrix.

1. **Accuracy:** The accuracy is the percentage to represent the correct predictions. It is defined as follows:

$$accuracy = \frac{\text{all correct predictions}}{\text{all samples}} = \frac{TP + TN}{TP + FN + FP + TN} \quad (2.26)$$

2. **Precision:** It indicates how many predictions were correct. The precision formula is defined as follows:

$$precision = \frac{TP}{TP + FP} \quad (2.27)$$

3. **Recall:** The recall metric tells us what proportion of the true positives were correct. The formula is:

$$recall = \frac{TP}{TP + FN} \quad (2.28)$$

4. **F1-score:** It is very used when there is an imbalance in the dataset. It is a combination of the precision and recall metrics. Formally, we say that the F1-score is the harmonic mean between the precision and recall. One crucial fact is that these metrics consider how many errors the model has per class and its influence on the final performance of the model and not only the number of absolute right or wrong

predictions. The formula for F1-score is:

$$F1\text{-Score} = 2 \cdot \frac{\textit{Precision} \cdot \textit{Recall}}{\textit{Precision} + \textit{Recall}} \quad (2.29)$$

Additional tools to analyze results

It is said in the CV community that a CNN is similar to a black box where we cannot see what is happening inside it and how the predictions are performed. To make CNNs more understandable, we can make plots about some principal components in a convolution.

- 1. Plotting filters and feature maps:** In summary, we can say that plotting the filters is equivalent to plotting the weights as pixels. The feature maps are the images produced after applying a particular filter. These plots let us understand that some filters specialize in identifying particular features in an image, like vertical or horizontal borders. After being applied to the feature map, they can activate particular zones of the image that will contribute to making the final predictions. Most of the time, filters and features maps can be abstract to understand.
- 2. Gradient-weighted Class Activation Mapping (Grad-CAM):** This technique was presented in a work called “Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization”, and it is explained as – *Gradient-weighted Class Activation Mapping (Grad-CAM), uses the gradients of any target concept (say ‘dog’ in a classification network or a sequence of words in captioning network) flowing into the final convolutional layer to produce a coarse localization map highlighting the important regions in the image for predicting the concept [37].* In simple terms, the Grad-CAM will let us plot the images as heatmaps where the zones of the image that more contribute to the prediction will be highlighted.

All the ways to analyze results described in this section will be used in Chapter 5.

Chapter 3

State of the Art

As soon as the pandemic hit the globe, research works related to the generation of datasets containing images of face masks were released to the CV community. However, before the pandemic, there was an active research field studying the detection of face masks and related datasets formation. In addition, some papers put special attention on occluded faces (i.e., detecting any face containing accessories like sunglasses, face masks, and partially visual faces). In the following subsections, related works on the topic are outlined.

3.1 Face and face-mask datasets

3.1.1 Face datasets

They are specialized datasets used to train models that detect faces on images or frames of a video. *WiderFace* is one of the most extensive datasets oriented to CV tasks on human faces [38]. The dataset was presented in 2016, and it contains 32203 images with 393703 face labels. It inherits images from the extensive *Wider* dataset.

Another important dataset is *face detection dataset and benchmark* (FDDB) [39]. FDDB was published in 2010, and it contains the annotations from 5171 faces in 2845 images. The dataset was generated using real-life scenarios like occluded faces. FDDB has been one of the pioneer datasets in the topic.

Annotated face in-the-wild (AFW) is composed of 205 images with 468 faces. It was released in 2012 as a part of the work in face detection and pose estimation [40].

3.1.2 Face-mask datasets

Researchers have already addressed an important challenge in face detection: occluded faces. Years before the pandemic, the CV community had been trying to generate a standard dataset oriented to accomplish this task without success. Notwithstanding, one of the most relevant datasets is called *MAFA*, which was published in 2017 with 30811 images downloaded from the internet, containing 35806 masked faces annotations [41].



Figure 3.1: Several pictures obtained from the MAFA dataset that includes general occlusions considered as masks [41].

On each image is at least one face occluded with a mask. This dataset's labels only indicate if there is an occlusion in the face, rather than telling if a facial mask is appropriately placed. The COVID-19 pandemic demands special attention to creating new datasets representing modern COVID-19 face masks scenarios, either real or virtual. In this connection, *MaskedFace-Net* was released in 2021 to solve the lack of large mask datasets [42]. It comprises 137016 editable images (i.e., masks added to faces through photo edition programs), with two classes: “correctly masked dataset”, and “incorrectly masked face dataset”.

Additionally, work [43] introduced three datasets: the *Masked-Face Detection Dataset* (MFDD), the *Real-World Masked Face Dataset*, and the *Simulated Masked-Face Recognition Dataset* (SMFRD) having 24771, 95000 and 500000 images respectively. MFDD contains images crawled from the internet, and the labels are only if the subjects wear masks. RMFRD is the world's largest dataset of real-world masked faces according to authors. The dataset contains 5000 images, 525 images of people using mask, and 90000

images of real people without masks. The dataset is free and available in GitHub. However, the dataset has also some faces wearing masks inappropriately. So, a future classification to train the model in order to recognize this new class might be developed. Besides, SMFRD contains real face images with facial masks added artificially to simulate faces wearing masks; only one part of this dataset is publicly available for download.



Figure 3.2: Image extracted from the RMFRD dataset showing 4 pictures of different persons using masks [43]

Finally, the *Moxa3k* dataset [44] contains images captured from Russia, Italy, China, and India during pandemic. The dataset has 3000 images in total. The number of faces that can be extracted from the dataset is 9161 without masks; 3015 masked.

3.2 Relevant face detection models

Through the years, multiple research works have been presented on face detection. One of the most influential works is the paper presented by Paul Viola and Michael Jones [45]. The model was based on rapid object detection using a boosted cascade of simple features. The Viola and Jones proposal is known worldwide for being the first CV model that employed a rudimentary ML technique known as boosted decision trees. The central idea was to create an algorithm capable of recognizing faces and drawing a box around them. With



Figure 3.3: Pictures extracted from the Moxa3k database: a) Blurred faces b) top angle camera view c) rotated face d) crowded image e) foggy environment f) sunny day faces [44].

the advent of CNNs, the face detection error decreased significantly [46, 47]. Subsequently, DL research accelerated its pace, and novel deep neural networks architectures were proposed to tackle the face detection problem. The main advantage of using DL architectures is the increased accuracy of the models. In [48], the authors used a faster region-based CNN framework to perform face detection. The latter work obtained the state-of-the-art face detection performance on the Fddb benchmark.

Refine Face: A face detection improved model that is based in ResNet with 6-level feature pyramidal structure used as the base of the network [49]. RefineFace is also based on face detector RetinaNet with five newly modules: STR, STC, SML, FSM, and RFE. Results varying the modules are presented in the paper having accuracy near and above 90%. A diagram of the model is presented in Figure 3.4.

Face detection using improved TinyYOLOv3 and attention mechanism: The necessity for a model to detect faces in a complex background and the long time it consumes are reasons the authors take to present a new algorithm. It is composed of an improved TinyYOLOv3 capable of extracting significant semantic information by changing the traditional convolution by deep separable convolution, i.e., dividing a single convolution into two or more parts to get a model of less size and with high detection speed. Also, the attention mechanism is added to the feature extractor layers to improve the detection position [50].

Face Detection Algorithm Based on Double-Channel CNN with Occlusion Perceptron: This method used a VGG16 as a backbone with the addition of a specialized unit to judge occluded areas becoming an occlusion perceptron neural network. The double channel makes reference to the capacity of the residual network to extract features of the whole face while the perception neural network extracts the features of the occluded parts. Both results are combined to produce the final prediction. In addition, this method improves detection speed and accuracy [51].

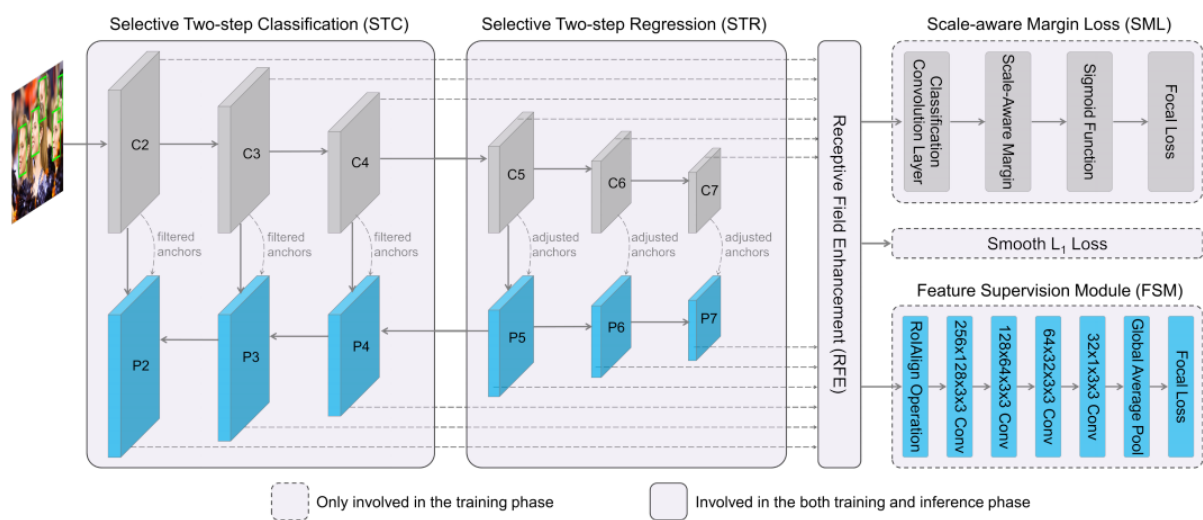


Figure 3.4: RefineFace structure. Modules, SML and FSM are only involved in training. STC, STR and RFE introduce a small amount of overhead in the model [49].

3.3 Face mask detection and classification for COVID-19

Multiple DL models were released from the beginning of the COVID-19 pandemic with different approaches to solve the face mask detection and classification challenges. For example, in [14], the authors propose a hybrid model using DL and classical ML. The first part uses a ResNet-50 architecture as a feature extractor, while decision trees and support vector machines are used to classify the position of the masks. Similarly, in [15], the authors presented a model capable of identifying face mask-wearing conditions by combining super-resolution images and classification networks such as SRCNet. Also, the use of

Resnet-50 as a feature extractor and YOLOv2 as a detector has been recently proposed [10].

In [52] proposed the so-called FMD-Yolo framework to detect whether people wear masks in public spaces correctly. This framework employs Im-Res2Net-101 and path aggregation network En-PAN for the steps of feature extractor and feature fusion, respectively. They test their approach against several state-of-art detection algorithms using two publicly available datasets. Their results on both datasets outperform the works that were compared.

Faster R-CNN-based: The author employed a faster method based R-CNN structure [53]. This new method allows the detection of complex faces poses, beard faces, different types of masks, and images of people using scarfs. The accuracy of detection for the method is 93.4%.

Context-Attention R-CNN-based is another detection method that shows a new framework used in masked faces recognition [54]. The method is based in multiple context feature extractor components, decoupling branches component, and attention components. Also, the detection method researches created a dataset with 8635 faces in different experimental conditions. The *mean average precision* (mAP) for the model is 84.1% over the dataset before mentioned, 6.8% mAP higher than Faster R-CNN. An overall schematic of the architecture is observed in Figure 3.5.

MobileNet-based: Dey et al [55] show a deep learning method that has multi-phase facial mask detection. The classifier depends on ResNet-10 and ROI detection. The model requires minimal processing capability and has a lightweight model. Also, the MobileNet-V2 is a good option for embedded systems.

Neural Network + Hand-crafted Feature: This classification method is an hybrid of deep learning and machine learning with the purpose of detecting facial masks. The model is based on two components: ResNet-50 that is the feature extractor, and SVM. The SVM classifier achieves accuracy of 99.49% on the dataset the author used [14].

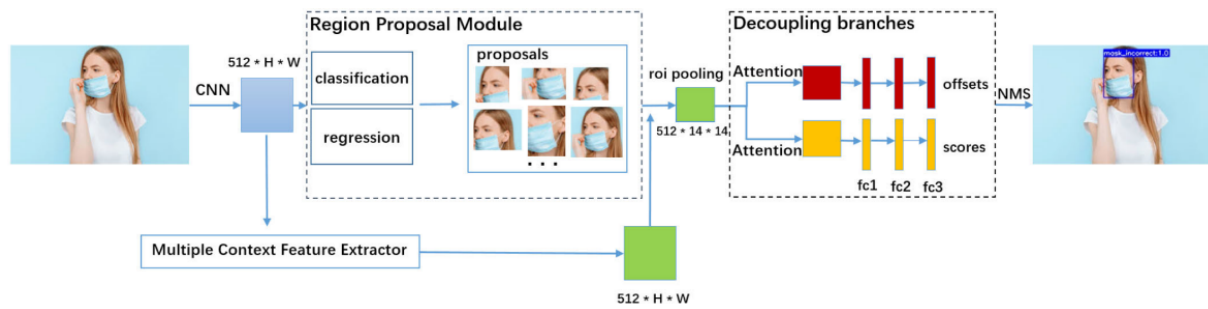


Figure 3.5: Context-Attention R-CNN-based model schematic: In the left, the region proposal module outputs some class-agnostic region proposals. At the right, the decoupling classification-localization branches, which consists of an effective attention module and also three fully-connected layers. They predict the classification scores and localization offsets. Finally, Non-Maximum Suppression(NMS) method is applied and the final results can be obtained [54].

Face Mask Recognition Network (FMRN): This detection model uses an algorithm to classify images by posture recognition. Then, the network processes the images according to the classes [56]. A design of the architecture used in this method can be observed in Figure 3.6. Also, daytime and nighttime settings were used to train the network. The accuracy in daytime was 95.8%. The accuracy in nighttime was 94.6%. The processing time for a single face was around 1.826 seconds in daytime, and 1.791 seconds in nighttime. This processing time might vary depending on the hardware used.

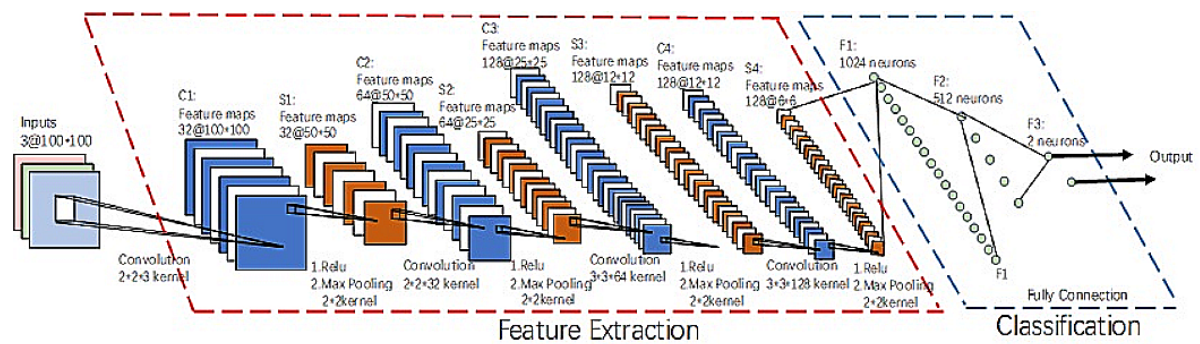


Figure 3.6: FMRN architecture has a total of eleven layers except for the input layer and the output layer. The input layer receives an image of size $100 \times 100 \times 3$ (because RGB channels). The middle part of FMRN model serves as feature extraction that has four convolutional layers and four pooling layers which are alternatively arranged. Each layer has its own training parameters. Also, each layer contains multiple feature maps [56].

Chapter 4

Methodology

Before beginning a Computer Vision project, it is important to set the guidelines to follow in order to achieve the objectives and satisfy the requirements. Therefore, Figure 4.1 shows the workflow followed to develop the two-stage pipeline to detect the incorrect use of face masks for COVID-19 awareness. As the pipeline consists of two tasks: the detection of faces and the classification of them according to the covid mask-wearing, two DL models are needed. The detection model to be used is the pre-trained RetinaFace. For the classification, different ResNet architectures will be trained to get two classification models: the first one to work with two classes (compliant and non-compliant) and the second to work with three classes (compliant, non-compliant, and incorrect). The datasets used for doing the training are the FMLD and MMD, which will be introduced in the following sections. MMD is used to complement the incorrect class dataset in the three classes classification model.

4.1 System design

4.1.1 The datasets used to train the proposed model

WiderFace dataset

This dataset was presented in the work “WIDER FACE: A Face Detection Benchmark” [38] and it was used by the authors of RetinaFace to produce the face detection model. WiderFace is a public dataset that has been previously used to train alternative face recognition models. Some important features of this dataset are challenging faces due

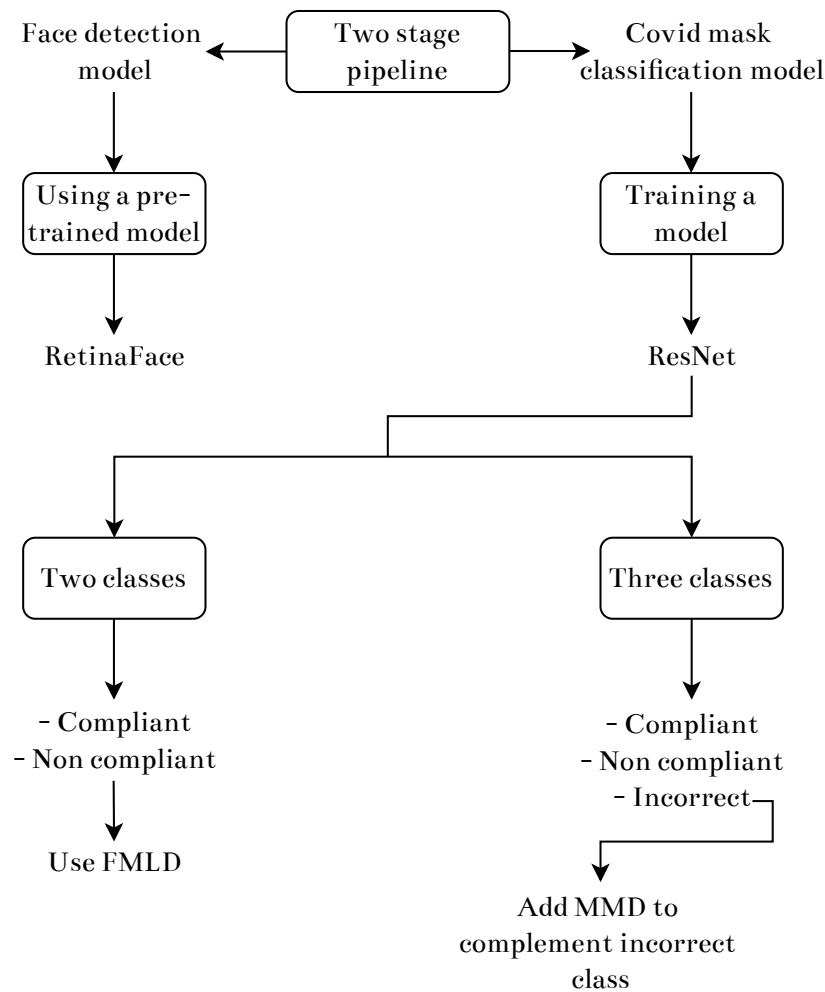


Figure 4.1: Workflow followed to develop the two-stage pipeline to detect the incorrect use of face masks for COVID-19 awareness .

to the presence of faces of different dimensions, poses, expressions, illumination, make-up, resolution, scale, and occlusion (See Figure 4.2). This fact closes the gap between real-world requirements and the available faces dataset. In addition, Wider Face is one of the most extensive publicly available datasets, becoming a benchmark in the face detection model task. Moreover, the dataset offers additional information like bounding boxes coordinates, categories, poses, and occlusions. This dataset is based on the WIDER dataset, and it has 32.203 images with 393.703 labeled faces with different poses, occlusions, and sizes [57].

Face-mask label dataset (FMLD)

This dataset is used in our model to detect the correct or incorrect use of face masks from the images obtained from the face recognition task using WiderFace. FMLD was first pre-

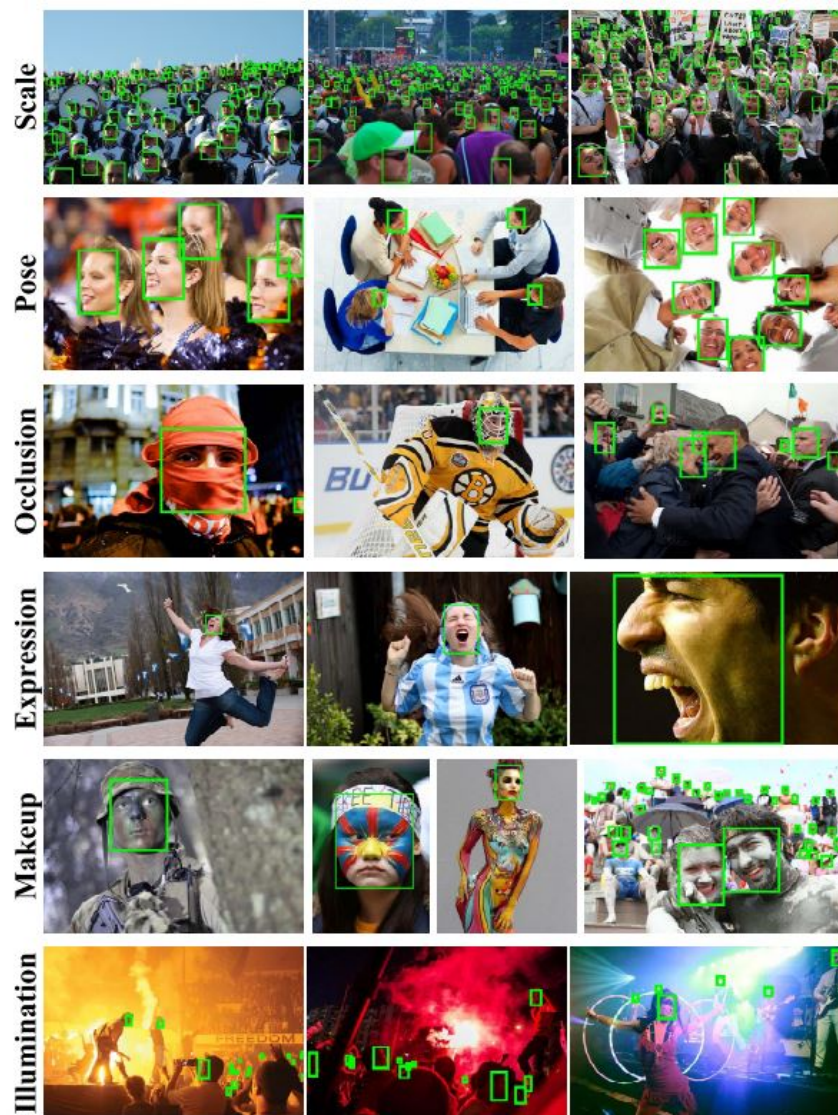


Figure 4.2: WiderFace dataset images with different features [38].

sented in work [11], where the authors released the dataset with bounding box annotations around face masks. This dataset has labels of images with faces with i) appropriate and ii) inappropriate use of COVID-19 masks. It is worth noticing that the masks used during the COVID-19 pandemic come in different shapes, colors, models, and patterns, and it requires an updated dataset to cover all those instances. The FMLD dataset takes images from two datasets donators: MAFA [41], and WiderFace [38]. The MAFA dataset is the principal source of correctly and incorrectly worn face masks. In contrast, the WiderFace dataset is used as the source of different images with faces without masks. Therefore, the FMLD dataset contains a realistic set of images of faces with and without COVID-19 masks in

different situations in real life. The FMLD dataset gives information such as the dataset donator, the folder and file name, the label (mask or no mask use) and the coordinates of the pixels where the face is located. Another important feature about FLML dataset is the variety in terms of the gender of the subjects, the pose of the face and the ethnicity of the subjects. This helps the DL models to learn a diversity of features present in real-world images. The proportion of each group is shown in Figure 4.3.

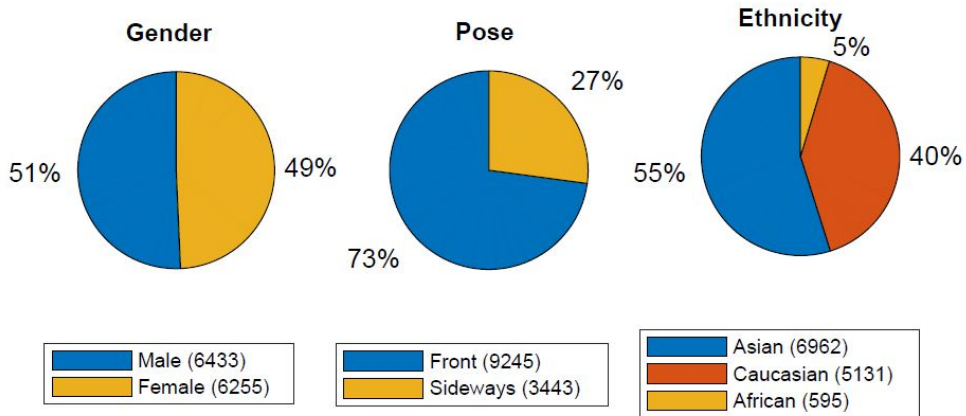


Figure 4.3: FMLD dataset proportions according to gender, pose and ethnicity of people [38].

FMLD dataset was used to train and test the different face mask detection DL models studied in this article. Figure 4.5 presents a subset of examples of the class “compliant”, where a correct use of face mask is shown; whereas Figure 4.6 shows examples of the “non-compliant” class. Table 4.1 summarises the FMLD dataset construction.

Table 4.1: Summary of the construction of the FMLD dataset. FMLD takes the donor datasets MAFA and WiderFace to construct the “compliant” and “non-compliant” classes.

Donor Dataset	Purpose	Images	Faces	Labels		
				With mask		Without mask
				Correct	Incorrect	
MAFA	Training	25876	29452	24603	1204	3645
	Testing	4935	7342	4929	324	2089
WiderFace	Training	8906	20932	0	0	20932
	Testing	2217	5346	0	0	5346
FMLD	Training	34782	50384	24603	1204	24577
	Testing	7152	12688	4929	324	7435
	Totals	41934	63072	29532	1528	32012

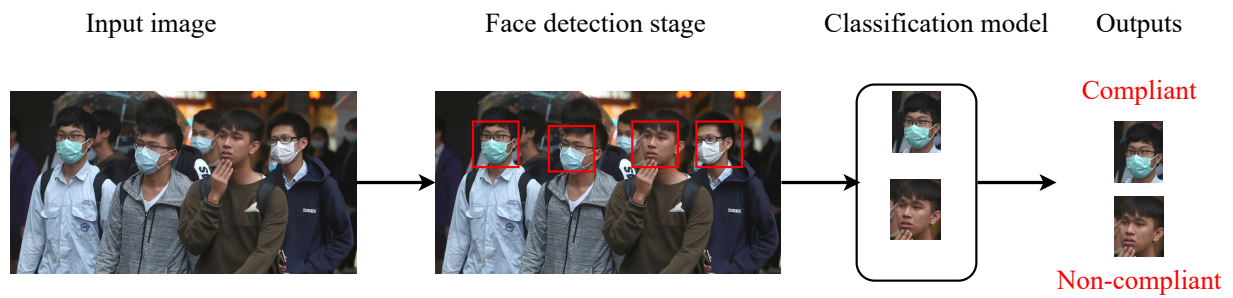


Figure 4.4: Two-stage pipeline. The first stage is in charge to detect all the faces in an image, while the second stage is in charge to make the classification.



Figure 4.5: A subset of images taken from the “compliant” class of the FMLD dataset. The examples (a)-(h) show faces with the correct use of masks according to the WHO advice.

Medical Mask Dataset (MMD)

MMD is a dataset composed of 6000 images of faces. This dataset has three principal classes: with a mask, without a mask, and incorrect mask-wearing. A special feature of this dataset is information about different accessories like caps, hats, and glasses, which may influence a covid mask-wearing classification task. It was released by the Human in the Loop organization, and it is available on the official organization’s website¹ after filling out a form to request the dataset. This dataset can complement the FMLD dataset, particularly for the incorrect class images, because FMLD does not have enough incorrect covid-mask wearing images.

¹<https://humansintheloop.org/>



Figure 4.6: A subset of images taken from the “non-compliant” class of the FMLD dataset. The examples (a)-(h) show faces that are incorrectly using (or not using) the masks.

4.1.2 Detection and clasification models

Face detection: RetinaFace

This is also known as single-shot multi-level face localization in the wild. This DL model is state-of-the-art technique in face detection that uses an end-to-end detector, and it performs three different face localization tasks together: i) face detection, ii) 2D face alignment, and iii) 3D face reconstruction. RetinaFace uses ResNet architecture plus fully pyramidal networks (FPN) to obtain a robust feature representation of the image. It outputs the bounding box of the face, five facial landmarks (denoting eyes, nose, and mouth), and a dense 3D mapping of points to represent the face [58]. The composition of RetinaFace is presented in Figure 4.7.

Face classification: ResNet

Since the revolution of CNNs with the successful introduction of AlexNet [33], researchers have tried to get more accurate models to work with CV image tasks. Nevertheless, making deeper CNNs does not only involve adding more layers together since problems like the vanishing gradient might appear (when the gradient is back-propagated to previous layers, it can converge to an infinitely small number causing low performance in the CNN). Various

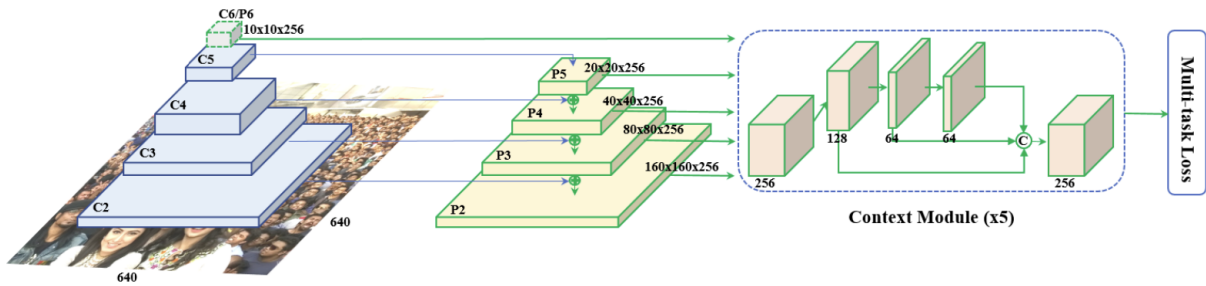


Figure 4.7: RetinaFace model has three main components: 1) Feature Pyramid Network, 2) Context Head Module, and 3) Cascade Multi Task Loss [58].

approaches have been presented to avoid this problem. ResNet is a robust solution to the latter problem since it introduces the concept of residual blocks (identity blocks). Residual blocks skip one or more connection layers to enable a fine-grained level of detail at the inference [36]. ResNet-18 is a variant of ResNet and its architecture is present in Figure 4.8.

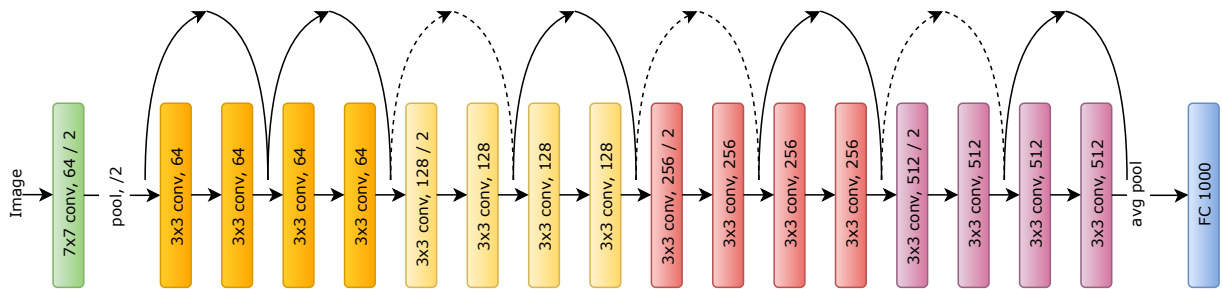


Figure 4.8: ResNet-18 architecture composed by residual blocks with two types of shortcuts: identity shortcuts and projection shortcuts.

Face classification: ResNeSt

This variant of the original ResNet takes advantage of the introduction of feature map attention and multi-path representation, both being essential techniques for visual recognition. In addition, ResNeSt adds split-attention blocks (i.e., computational units composed of feature groups and split attention operations). As a result, ResNeSt presented better transfer learning results when used as a backbone on many public benchmarks [59].

4.1.3 The two-stage pipeline CNN

The proposed pipeline is composed of 1) a face detection stage, with its output being the input for the 2) face classification stage to detect the use of COVID-19 face masks. According to the mask placement, the pipeline will take images or frames and classify them into “compliant” and “non-compliant”. The pipeline deployment was tested, keeping RetinaFace as the predefined detection model and changing the classification models, specifically the ResNet and ResNeSt variants. In addition, the inference behavior of all the models was studied under different optimization algorithms. The pipeline can be seen in Figure 4.4

4.1.4 Loss function and optimizers

Theorem 1 *Let f be the proposed classification function which takes the weight matrix \mathbf{W} as an input, and returns the predicted label $\hat{l} \in \{\text{compliant}, \text{non-compliant}, \text{incorrect}\}$. The classification task can be mathematically formalized as*

$$\hat{l} = f(\mathbf{X}, \mathbf{W}). \quad (4.1)$$

To measure the performance of our classification model, the loss \mathcal{L} between the ground truth and the distribution \hat{l} has to be computed. We define $\mathcal{L}(\mathbf{W})$ as the *cross-entropy* loss function. The classification network iteratively updates the values of \mathbf{W} by backpropagating the error through the neural network and converging to the local minimum value of $\mathcal{L}(\mathbf{W})$. The labeled images from the FMLD dataset can be formalized as

$$(\mathbf{X}^{(i)}, \mathbf{Y}^{(i)}) \text{ for } i \in \{1, \dots, n\}, \quad (4.2)$$

where n represents the total number of images in the dataset, and $\mathbf{X}^{(i)}$ and $\mathbf{Y}^{(i)}$ represents the i^{th} image and its estimated label, respectively. Our model computes the weights $\mathbf{W} = \{W^{(1)}, \dots, W^{(n)}\}$ that corresponds to the minimum value of $\mathcal{L}(\mathbf{W})$ as follows:

$$\mathcal{L}(\mathbf{W}) = -\frac{1}{n} \sum_{i=1}^n L(\mathbf{X}^{(i)}, \mathbf{Y}^{(i)}, \boldsymbol{\theta}), \quad (4.3)$$

which can be written in the logarithmic likelihood form as follows:

$$\mathcal{L}(\mathbf{W}) = -\frac{1}{n} \sum_{i=1}^n \mathbf{Y}^{(i)} \log(\hat{p}^{(i)}), \quad (4.4)$$

with $\hat{p}^{(i)}$ as the estimation probability that the i^{th} image matches the intended l value [60]. For the CNN optimization, we let the cross entropy with respect to W (eq. (4.4)) be represented as $\boldsymbol{\delta} = \nabla_{\mathbf{W}} \mathcal{L}(\mathbf{W})$

$$\boldsymbol{\delta} = \frac{1}{n} \nabla_{\mathbf{W}} \sum_{i=1}^n \mathcal{L}(\mathbf{X}^{(i)}, \mathbf{Y}^{(i)}, \mathbf{W}). \quad (4.5)$$

Adagrad [61], ADAM [62], SGD [63], SGD with Momentum [64], and RMSprop [64] were used to estimate the optimal set of values in \mathbf{W} that minimizes $\mathcal{L}(\mathbf{W})$.

4.1.5 Framework and hardware acceleration

ML framework

The framework employed to design and implement the pipeline is PyTorch since all the pre-trained models under study are implemented in the framework, and they can be easily imported through *torchvision*.

Hardware acceleration

On the hardware acceleration side, we used a Tesla V100 and Tesla P100 graphical processing units (GPUs) with 16GB from Google Colaboratory with a CPU Intel® Xeon(R) 2.00GHz. The GPUs play an important role in training a DL model because, depending on their capacity, the training can take more or less time. It is essential to know that the Tesla V100 GPU has a performance of 14,029 gigaflops while Tesla P100 has 10,609 gigaflops of performance. In addition, the Tesla V100 is also superior in the number of cores because it has 5120 and the Tesla P100 3584 cores. Therefore, it is clear that Tesla V100 is more powerful than Tesla P100.

4.1.6 Performance measures

To attest the performance of the classification stage, a mean average precision (*mAP*) was used as a metric to determine the most appropriate model. Also, additional metrics will be used:

1. **Accuracy:** Percentage to represent the correct predictions.

$$accuracy = \frac{\text{all correct predictions}}{\text{all samples}} \quad (4.6)$$

$$= \frac{TP + TN}{TP + FN + FP + TN} \quad (4.7)$$

2. **Precision:** The number of correct predictions.

$$precision = \frac{TP}{TP + FP} \quad (4.8)$$

3. **Recall:** The proportion of the true positives which were correct.

$$recall = \frac{TP}{TP + FN} \quad (4.9)$$

4. **F1-score:** The F1-score is the harmonic mean between the precision and recall.

$$F1\text{-Score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (4.10)$$

4.1.7 Prepare the dataset

The FMLD and MMD datasets need to be processed before the training because both give complete images of people in different scenes and not only the people's faces. For this reason, they provide annotations in XML format for each image. These annotations specify the bounding box of each face on each image, the class label, and additional information described in subsection 4.1.1. Therefore, a python code is made. This code can read the XML annotations, make the crops of the faces, and store them. The library used to read XML files in python is the XML Processing Module. On the other hand, the OpenCV library was used to read images and crop them. In the particular case of the FMLD

dataset, it was necessary to download the donor datasets, i.e., the MAFA and WiderFace datasets, and then use the annotations provided in work [11] to get the images of the faces.

Dataset splitting

In order to do the trainings, it is important to split the dataset in training, validation and test dataset. The proportion for each dataset is as follows:

1. Training dataset: 80%.
2. Validation dataset: 10%.
3. Test dataset: 10%.

Chapter 5

Results and Discussion

5.1 Phase I: Test deep learning classification models with different optimizers

In this section, the results obtained in the two-stage pipeline are presented. Special attention is taken to the classification stage since the pre-trained RetinaFace model performs the face detection part, requiring no fine-tuning. Therefore, the image classification task concerning COVID-19 masks is the pivotal part of this study. The principal objective is to gain an insight into the performance of DL models while solving an urgent real-life problem. We have implemented all the variants of ResNet and one variant of ResNeSt using transfer learning. The initial weights were borrowed from the pre-trained model with the ImageNet dataset. Then, all the models were trained using the FMLD dataset, with RGB input images of size $W \times H$ pixels as inputs, where W and H are ≥ 10 to avoid losing the activation maps during the pooling operations. The dataset was randomly divided into training, validation, and test sets. An overview of the resulting dataset can be found in Table 5.1.

Table 5.1: An overview of the dataset split used to train the different classification models.

Dataset	Classes	
	<i>Compliant</i>	<i>Non-compliant</i>
<i>Training</i>	7028	17525
<i>Validation</i>	1757	4382
<i>Test</i>	4795	13750
<i>Total</i>	13580	35657

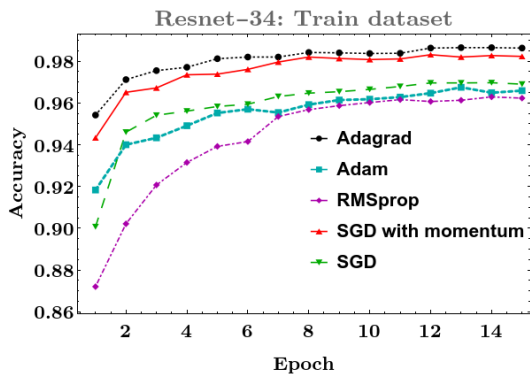


Figure 5.1: ResNet-34 accuracy on train dataset.

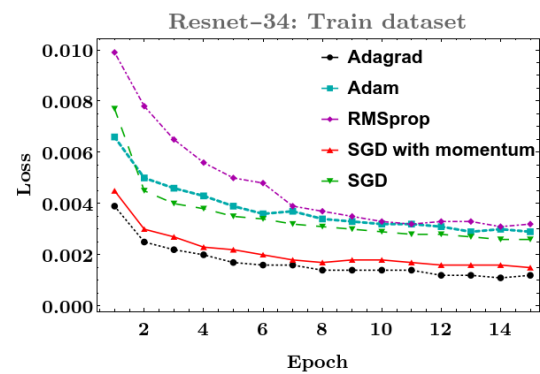


Figure 5.2: ResNet-34 loss on train dataset.

The cross-entropy was adopted as a loss function. At the same time, the following optimizers were considered: 1) SGD with a learning rate equal to 0.001 and a reduction factor of 0.1 after seven epochs, 2) SGD with a momentum value of 0.9, 3) Adam with a learning rate of 0.001, 4) RMSprop with a learning rate of 0.01, and 5) Adagrad with a learning rate of 0.01. The models were trained with 15 epochs, each with a batch size equal to 16. The behavior of the accuracy and loss during the training and validation stages of ResNet-34 with Adagrad optimizer was selected as the optimal configuration. The ResNet-34 with Adagrad results can be found in Figures [5.1-5.4]. The RMSprop, Adam, and SGD are the less effective optimizers either in training and validation datasets since they cause the highest losses and lowest accuracy. On the other hand, SGD with momentum and Adagrad optimizers got the highest accuracy and lowest loss, but their differences are minimal. The details are given as follows:

1. **ResNet-18 and ResNet-152:** The best performance is obtained with the Adagrad optimizer for both the training and validation dataset.
2. **ResNet-34, ResNet-50, ResNet-101 and ResNeSt-200 :** On these models, the best result in the training dataset is obtained with Adagrad optimizer, and for the validation dataset, SGD with momentum is more robust.

However, it is necessary to analyze the results obtained on the test dataset to verify the performance of each model. Therefore, the experiments were performed over 18545 images (4745 compliant and 13750 non-compliant), and the accuracy results are presented in Table 5.2.

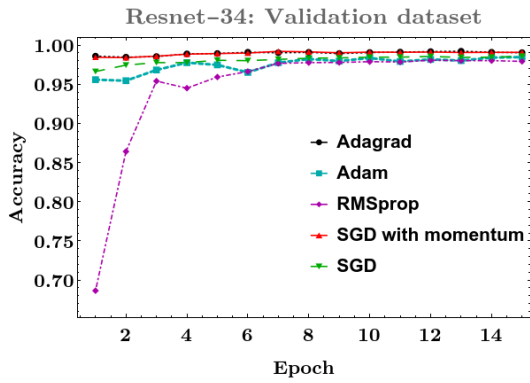


Figure 5.3: Resnet-34 accuracy on validation dataset.

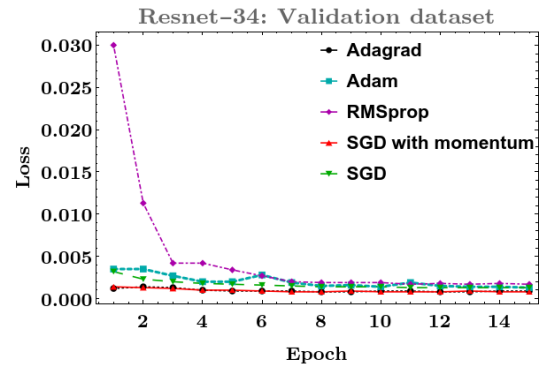


Figure 5.4: Resnet-34 loss on validation dataset.

Table 5.2: Models with different optimizers.

Model	Accuracy %				
	SGD	RMSprop	Adam	SGD with Momentum	Adagrad
ResneSt-200	96.44	92.41	95.79	97.01	96.78
Resnet-18	96.06	95.24	95.95	96.35	96.64
Resnet-34	96.06	95.28	95.86	96.4	97.24
Resnet-50	96.54	95.82	95.82	96.59	96.21
Resnet-101	95.64	94.95	95.29	96.8	96.63
Resnet-152	95.33	94.92	95.29	97.21	97.08

As can be seen, all the models achieved an accuracy greater than 94% except ResNetSt-200 with RMSprop optimizer, which obtains 92.41%. The more accurate models are the ones trained with SGD with momentum and Adagrad optimizers. ResNet-34 with Adagrad is the model configuration with the highest accuracy, followed by ResNet-152 with SGD with momentum and ResNet-152 with Adagrad. All the models ordered by their accuracy percentage can be seen in Figure 5.5. The lowest accuracy models are the ones trained with RMSprop optimizer.

To analyze the results obtained in the test dataset, we choose ResNet-34 with Adagrad optimizer as the model to perform the predictions. Therefore, a confusion matrix is presented in Figure 5.6 to understand the results. A value of 0 represents the label “compliant”, and a value of 1 represents the class “non-compliant”. According to Table 5.1, we have 4795 “compliant” images, and the model finds 4673 of them (with 122 wrong predictions). For the “non-compliant” class, the model finds 13360 from 13750 (with 390 wrong predictions). Therefore, the model gets an accuracy performance of around 97%.

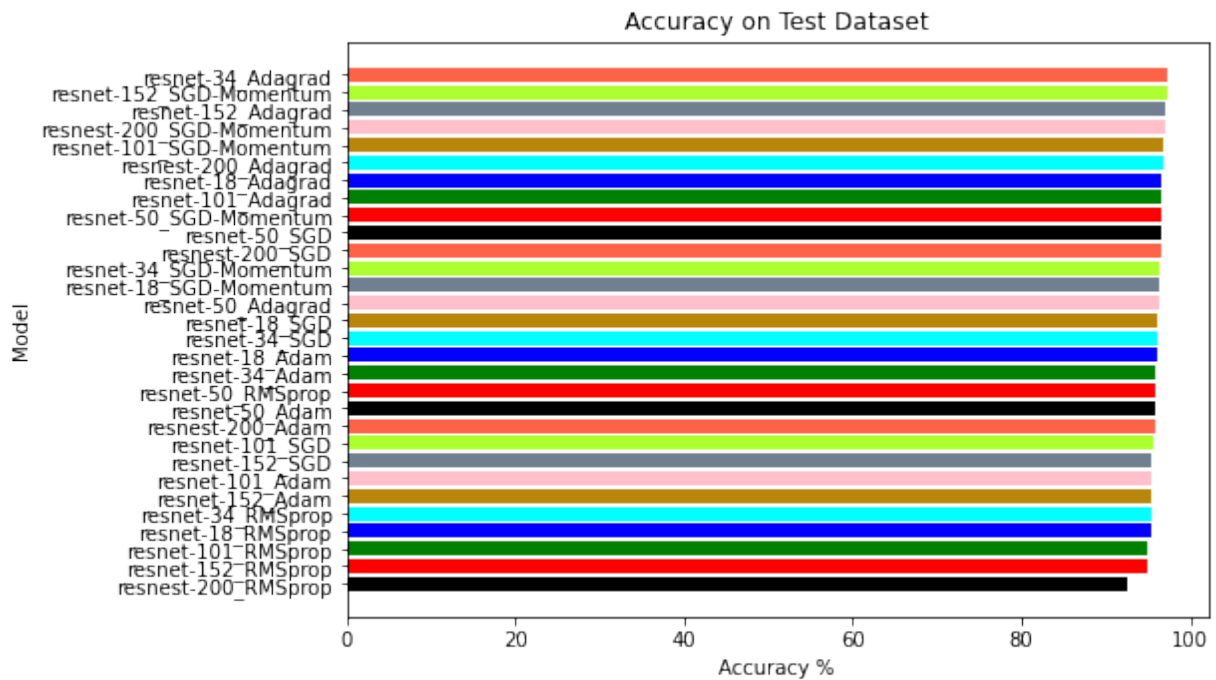


Figure 5.5: Test results of all the considered models with different optimizers.

Since all models perform with high accuracy, it is essential to know the training speed, and the value is proportional to the number of layers. Therefore, the deeper model, i.e., ResNeSt-200, was trained, consuming longer, while the less deep model, i.e., ResNet-18, finished the training in less time. The time is proportionally related to the computational power needed to train the dataset, demanding powerful GPUs. Sometimes these computational resources are not available, and the experiments are difficult to do on personal or desktop computers. From this phase, we can conclude that a light model like ResNet-34 can accurately perform the COVID-19 mask-wearing classification task using Adagrad as an optimizer. Therefore, we can part from here to implement different CV techniques to get a better classification model.

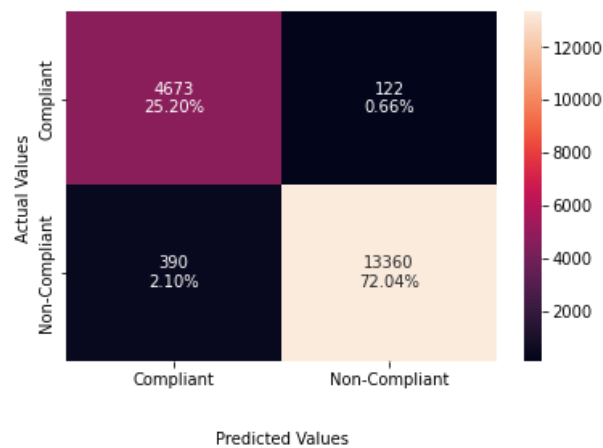


Figure 5.6: Confusion matrix of the results obtained from ResNet-34 with Adagrad optimizer in the classification stage.

5.2 Phase II: Improving the dataset, training the classification model for two classes, and extending it to 3 classes

5.2.1 Datasets

The accuracy obtained in the previous training was good (about 97.23%). In order to improve the accuracy of the model, it was necessary to check the dataset. We found that the dataset has many wrong labeled images and others are too noisy. For this reason, a data cleaning was performed. An important tool to perform this activity was IBM Cloud Annotations. It is a simple web platform where we can upload our dataset as zip files, add the classes and begin to classify. The images are shown as a mosaic, and we can select them one by one, or keep pressing the mouse's left button and move to the left and right, up and down, and select all the images we want to put inside a class. In our particular project, our classes for the first dataset were Compliant and Non-compliant, and for the second dataset were, Compliant, Incorrect, and Non-Compliant. After the cleaning, we should balance the dataset as is shown in Table 5.3.

In order to extend the classification task to three classes, it was necessary to look for another dataset similar to FMLD because there were not enough images for the 'incorrect' class. The dataset selected was the MMD. Although we add images from another dataset,

Table 5.3: New FMLD dataset after cleaning.

Dataset	Classes	
	<i>Compliant</i>	<i>Non-Compliant</i>
<i>Training</i>	20937	20937
<i>Validation</i>	1231	1231
<i>Test</i>	2463	2463
Total	24631	24631

Table 5.4: New dataset formed by FMLD + MMD datasets extended to three classes.

Dataset	Classes		
	Compliant	Incorrect	Non-Compliant
Training	20937	935	20937
Validation	1231	55	1231
Test	2463	109	2463
Total	24631	1099	24631

there is an imbalance in the number of images for incorrect classes concerning the other two classes. Later, in the results subsection, we will see if the imbalance affects or not the performance of the classification model. The new dataset formed by FMLD and MMD is detailed in Table 5.4.

In addition, a data augmentation was performed using transforms available in the Torchvision library. This type of data augmentation is different from the one we know because we will not produce n number of images from one. Instead, before a batch of images enters the CNN, randomly, some of them are modified by applying effects like horizontal flips and rotation. This data augmentation helps to improve the variety of the dataset in training. Moreover, as the input to the ResNet is $3 \times 224 \times 224$, a resize of the images should be applied. Finally, a normalization of the images using the means and standard deviation of the RGB channels of the ImageNet dataset is useful to take full advantage of transfer learning because the ResNet pre-trained model was trained on the ImageNet dataset and our COVID-19 face masks images pixels values should be in a similar scale.

5.2.2 Training

The training was performed with ResNet light models like ResNet-18 and ResNet-34 available on the TorchVision repository. The decision to only test these two models is based

on phase I where it was proved that ResNet-34 was enough to perform the classification. Therefore, the objective is to improve the accuracy of ResNet-34 or try to get better accuracy with ResNet-18, a smaller model.

Cross-entropy loss function was used. The optimizer used for the training was Adagrad with a learning rate of 0.001, step size equal to one third of the number of epochs, and gamma equal to 0.1. The training consisted of 10 epochs for the model of two classes and 15 for the three classes. Finally, the batch size of the training was 64 because we reached the maximum capacity of the GPU. Each epoch takes approximately 1.14 seconds to finish.

5.2.3 Hardware

In order to perform the training, a Tesla P-100 GPU was used on a Google Colab instance with a CPU Intel ® Xeon(R) 2.00GHz.

5.2.4 Results and analysis

To evaluate the model, a test dataset was used.

Two classes

The best accuracy obtained for the two classes (compliant and non-compliant) was 99.6% with a ResNet-18. Additional metrics like precision, recall and F1-score are given in Table 5.5. Also, a confusion matrix is plotted in Figure 5.7 to see the TPs, TNs, FPs and, FNs. There were thirteen compliant images classified as non-compliant and seven non-compliant images wrong predicted as compliant images. As we can see, the error rate is very low. Next, the wrong predicted images are plotted according to each class.

Table 5.5: Additional evaluation metrics for COVID-19 mask wearing classification model on New Test dataset.

	Precision	Recall	F1-score	Images
Compliant	1.00	0.99	1.00	2463
Non-Compliant	0.99	1.00	1.00	2463

- **Compliant images classified as non-compliant**

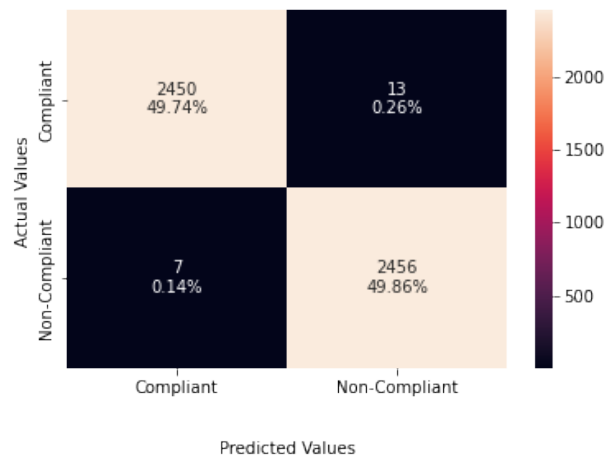


Figure 5.7: Confusion matrix about the testing of ResNet-18 COVID-19 mask wearing classification model on the new test dataset.



Figure 5.8: Compliant images classified as non-compliant.

In Figure 5.8, there are three images 5.8a, 5.8h, 5.8i with the wrong labels, i.e., the true label must be ‘noncompliant’ but they were labeled as compliant. The last six images were wrong classified by the model. We can see the model has problems with the classification of industrial masks (images 5.8b and 5.8d) and with profile faces like image 5.8c. We can use Gradient-weighted Class Activation Mapping (Grad-CAM) to visualize the most representative parts of these images for the classification model. With Grad-CAMs we can find two principal problems here: the first one is related to the location of the most representative part of the image for the model because it is not around the nose tip as should be (images 5.9a, 5.9b, 5.9c, and, 5.9d). The

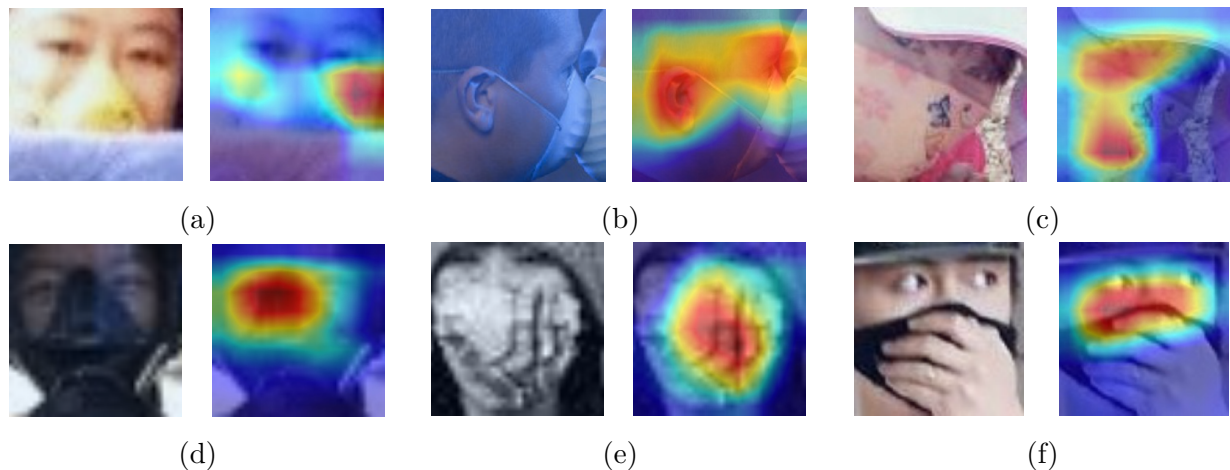


Figure 5.9: Compliant images classified as non-compliant analyzed with Grad-CAMs.

second is that although the position of the most representative area of the image is correct, the model cannot make a good classification (images 5.9e and 5.9f).



Figure 5.10: Compliant images classified as non-compliant.

- **Non-compliant images classified as compliant**

In Figure 5.10, there are wrong images predicted except the last one because it was wrong labeled. The principal cause for the bad performance of the model over these images can be the presence of different objects around the noses, and probably, the model takes these objects as a COVID-19 mask. Grad-CAMs presented in Figure 5.11 are helpful to prove this fact.

For images 5.11a and 5.11b, we can notice that we classification model is unable to distinguish between a COVID-19 mask and an object covering the nose. For the last two images (images 5.11c and 5.11d), the activation zone is wrong, therefore there are bad predictions.

In addition, we can see the results of the training for this model in the graph presented

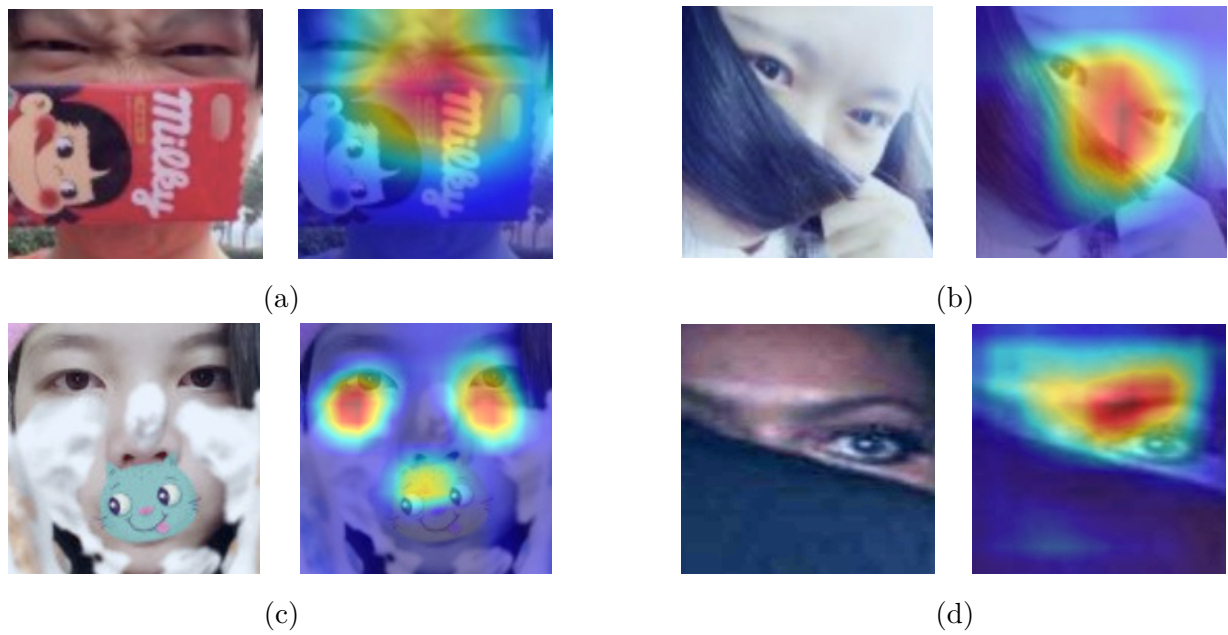


Figure 5.11: Non-compliant images classified as compliant analyzed with Grad-CAMs.

in Figure 5.12. The first plot represents the accuracy vs. the number of epochs in the train and validation dataset. Here, we can see that the accuracy on the training dataset increase on each epoch but in the validation dataset, the best accuracy is in the second epoch, then it decreases and finally, it remains constant. The second plot represents the loss vs. the number of epochs. Here we can notice that the loss only decreases in the training dataset but not in the validation dataset.

Next, the operation of the model with three classes will be explained.

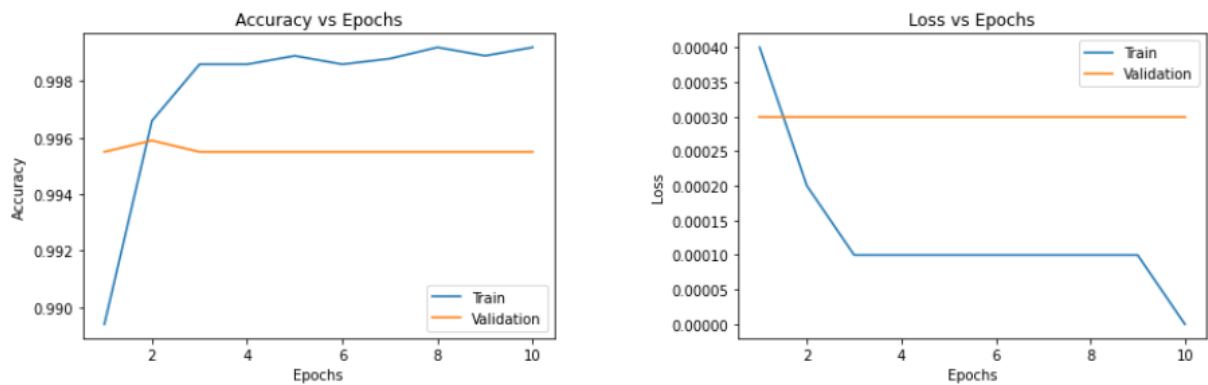


Figure 5.12: ResNet-18 accuracy and loss on train dataset for 2 classes model.

Extension to three classes

The accuracy obtained for the three classes model was 99.2% with a ResNet-18. Additional metrics are presented in Table 5.6 and a confusion matrix is given in Figure 5.13.

Table 5.6: Additional evaluation metrics for COVID-19 mask wearing classification model on New Test dataset for 3 classes.

	Precision	Recall	F1-score	Images
Compliant	1.00	0.99	0.99	2463
Incorrect	0.86	0.90	0.88	109
Non-Compliant	0.99	1.00	0.99	2463

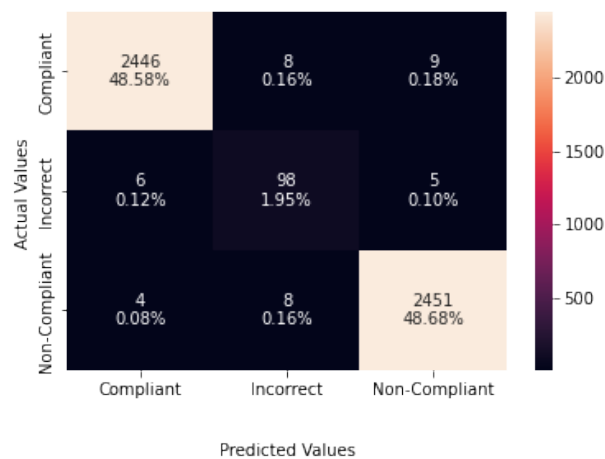


Figure 5.13: Confusion matrix about the testing of ResNet-18 COVID-19 mask wearing classification model on the new test dataset for three classes.

According to the confusion matrix in Figure 5.13, there are a few wrong predictions for each class. It is important to remember that there are only 109 images for the incorrect class; six were predicted as compliant, and five were predicted as noncompliant. Eight images were predicted as incorrect for the compliant class, and nine were predicted as compliant. Finally, four were predicted as compliant for the noncompliant class, and eight were predicted as incorrect. Therefore, we can conclude that the imbalance of the dataset in this particular case does not affect the accuracy of the model. However, the F1-score metric, which considers how many errors the model has per class and its influence on the final performance, shows that the imbalance of the dataset affects the performance for the incorrect class (see Table 5.6).

The matrix confusion shows us summarized information about the model's results, but it would be helpful to know the images where the prediction was wrong. For this reason, we plot the wrong predicted images per class according to as they have been classified in the labeling process.

- **Compliant images wrong predicted**

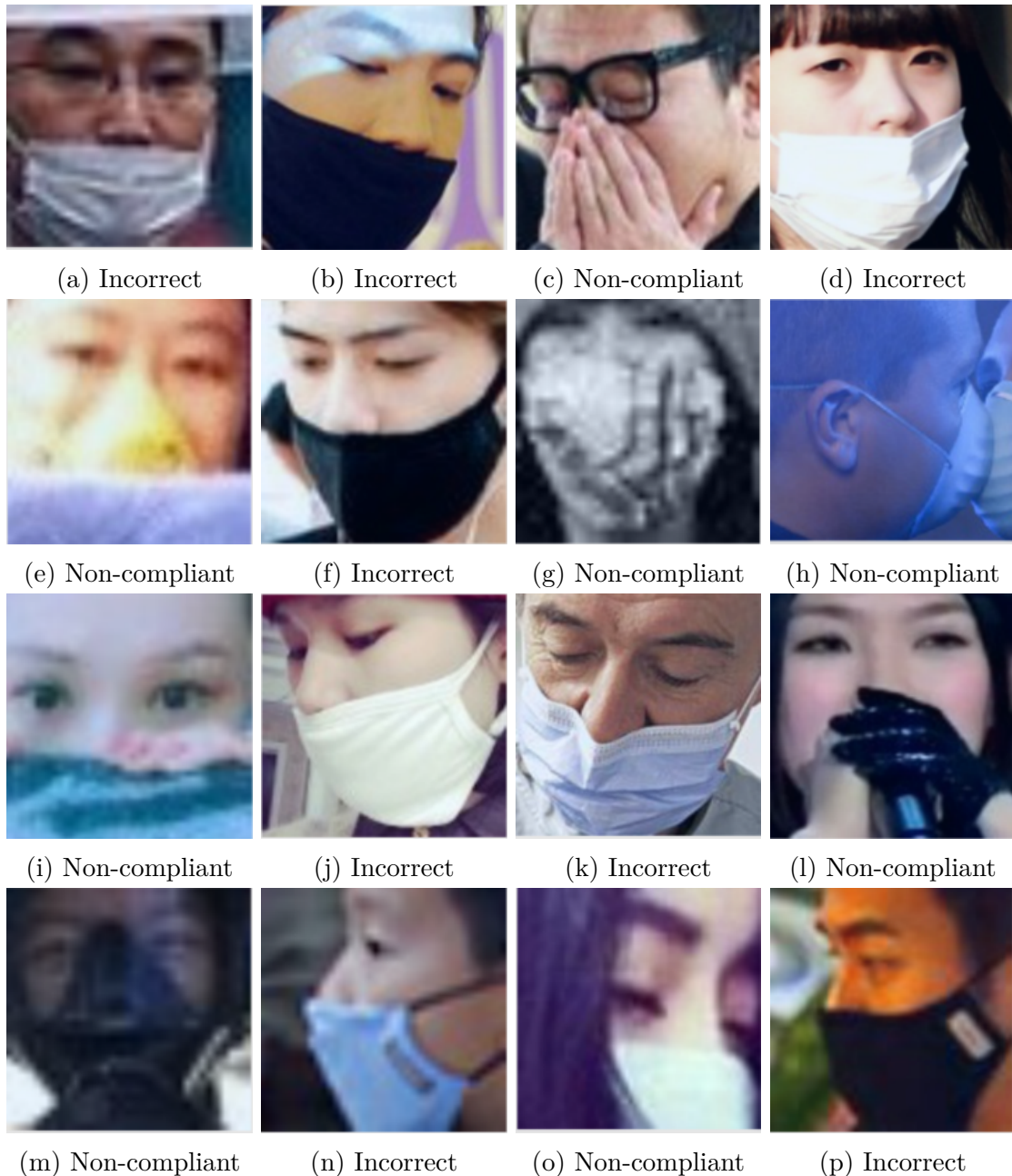


Figure 5.14: Compliant images wrong classified. Each one has the predicted label assigned by Resnet18 classification model.

In Figure 5.14, at most wrong predictions in this class are related to mistakes in the labeling of the data (images 5.14a, 5.14b, 5.14c, 5.14d, 5.14f, 5.14h, 5.14j, 5.14k, 5.14l, 5.14n, 5.14p), so, the prediction is correct but the true label is wrong. Also, we can notice that the model has problems with face profile images (5.14h) and with faces that are so close (5.14e, 5.14i).

There is a way to understand better the reason for wrong predictions using Grad-CAM. It enables us to visualize which parts of an image were more significant for a model to make a prediction. The Grad-CAM was computed for wrong predicted images.

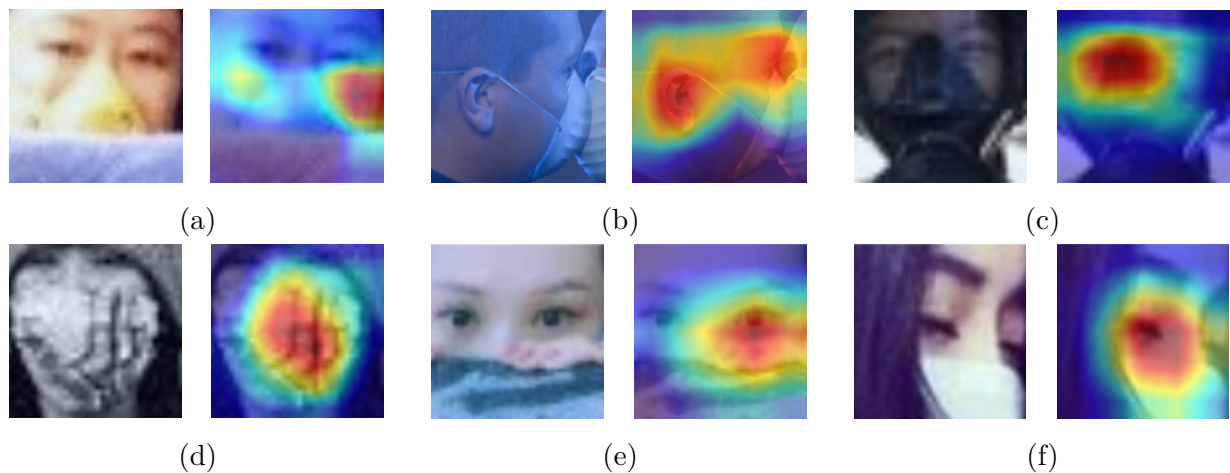


Figure 5.15: Compliant images wrong predicted analyzed with Grad-CAMs.

The Grad-CAMs show a common factor: the most significant part of the image for the model to make the prediction is not the tip of the nose as it should be. In the case of image 5.15b, there is a clear problem related to profile faces. This can be related to the lack of profile images in the training dataset because the proposed model is focused on frontal faces. The subjects of images 5.15a - 5.15c wear nose protection but it is not a COVID-19 mask face, therefore a bad prediction came up. Finally, images 5.15e - 5.15f are so close to the face, this may cause the failure in the predicted label.

- **Non-compliant images wrong predicted**

One more time there are problems related to wrong true labels in images 5.16e, 5.16f,

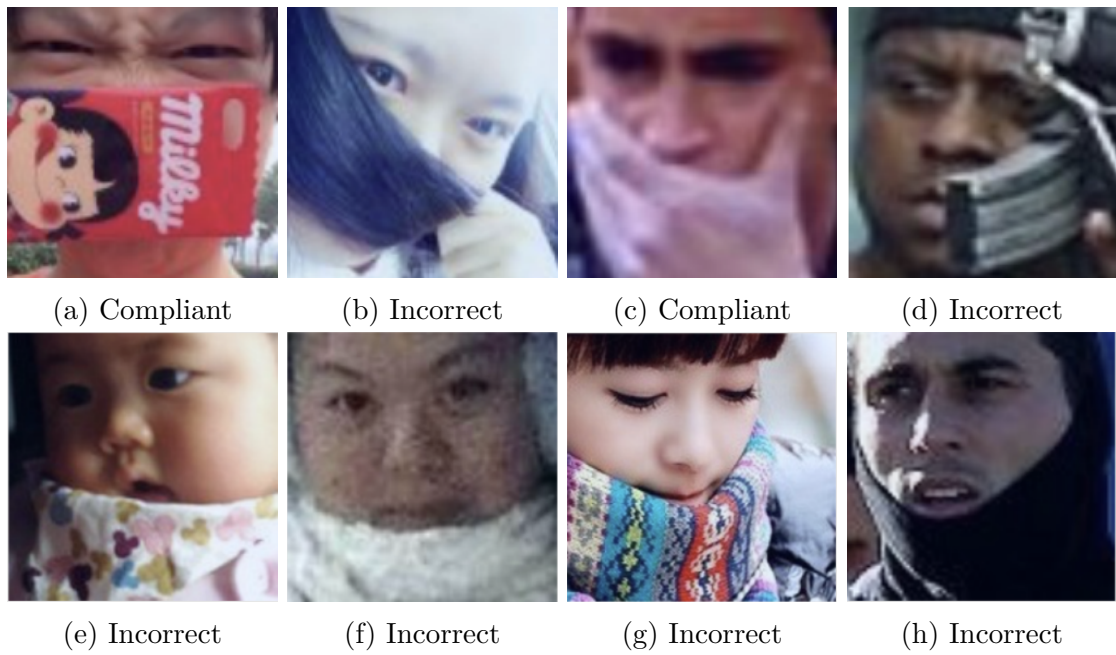


Figure 5.16: Non-compliant images wrong classified. Each one has the predicted label assigned by Resnet18 classification model.

5.16g, and, 5.16h. Also, there are wrong predictions especially based on the confusion of COVID-19 face masks with other objects (images 5.16a, 5.16b, 5.16c, 5.16d). This fact can be seen with the use of Grad-CAMs.

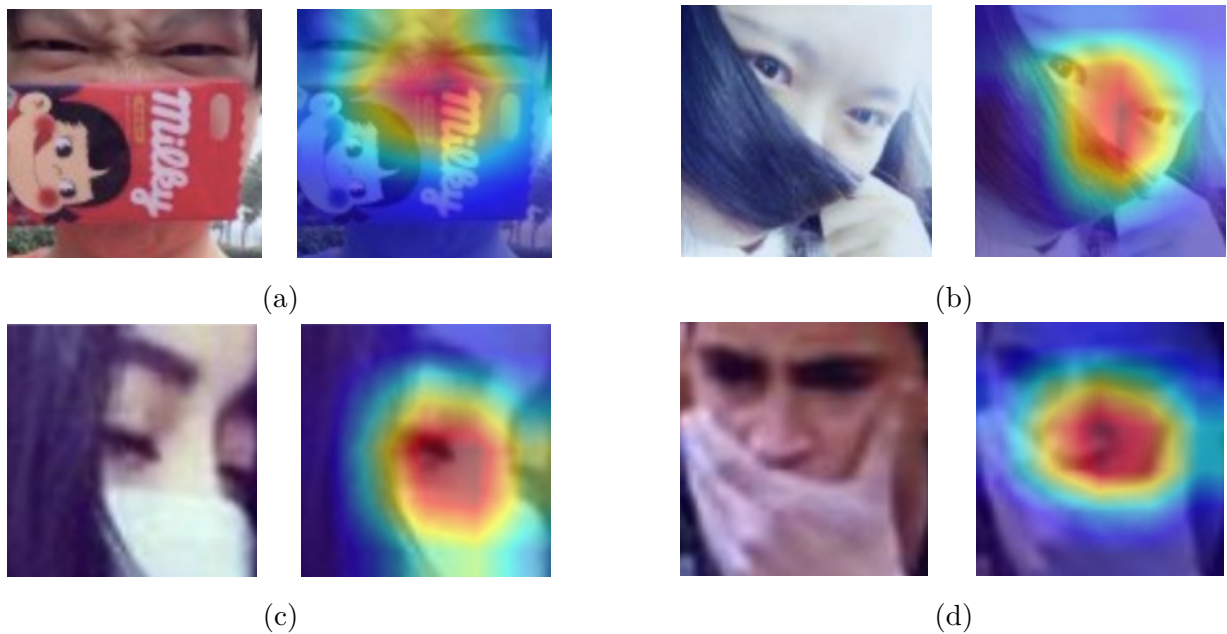


Figure 5.17: Non-compliant images wrong predicted analyzed with Grad-CAMs.

According to the Grad-CAM, the more significant parts of the image are located

around the nose. Here, we can see objects covering noses different from COVID-19 face masks. Therefore, it is probably that the model sometimes can become confused when a nose is covered with some object distinct from a mask. To solve this problem, it can be helpful to put in the training datasets images with noses covered by no COVID-19 faces masks and labeled as “non-compliant” to let the model learn this new feature.



Figure 5.18: Incorrect images wrong classified. Each one has the predicted label assigned by Resnet18 classification model.

- **Incorrect images wrong predicted**

The class “incorrect” is a particular case because it can be relative according to the position of the COVID-19 face mask concerning the nose. For example, some people consider that incorrect use of the mask is present if it only covers the nasal base. Other people consider an incorrect use of the COVID-19 mask face if it does not cover the ridge of the nose entirely. Finally, since many people prefer to put the COVID-19 face mask under the chin to avoid taking it out completely when they want to do not wear it for a short period of time, the position of the COVID-19 face mask under the chin can be considered as “incorrect”. As we can see, we can have

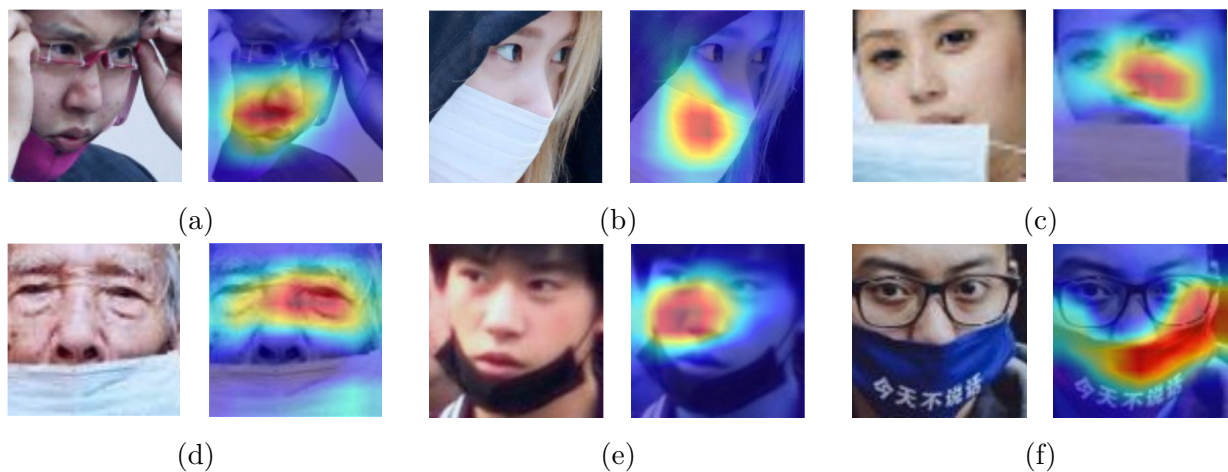


Figure 5.19: Incorrect images wrong predicted analyzed with Grad-CAMs.

different concepts for the incorrect class, and for this reason, some wrong predictions (see Figure 5.18) are related to this fact, and Grad-CAMs can help us to describe it. According to Grad-CAMs, the classification model focuses on two principal parts of the image. The first one is around the nasal base like in images 5.19a, 5.19c, and 5.19e. The second one is about the identification of the COVID-19 face mask (images 5.19b and 5.19f). For the first case, it may be impossible to classify the image as incorrect because the model does not focus on the COVID-19 mask due it is under the chin. As the model identifies the absence of the COVID-19 mask around the nose, the predicted label is Non-compliant. For the second case, the model only pays attention to the presence of the COVID-19 mask. For this reason, the model can not identify that the mask is only covering the nasal base, i.e., the use of the mask is incorrect, but the model predicts it as compliant.

In addition, we can see the results of the training for this model in Figure 5.20.

Similar to the previous subsection of the two models, we will see the training results. In the first plot, we can see that the accuracy of the training dataset increases. In the final epochs, it converges, but in the validation dataset, the accuracy increases, and decreases, and in epoch number 14, it reaches the maximum value. On the other hand, the second plot shows that in the training dataset, the loss decrease in the first four epochs to 0.0001, and then it keeps constant until the final. In contrast, the loss decreases along the epochs in the validation dataset, being a lower loss of 0.004.

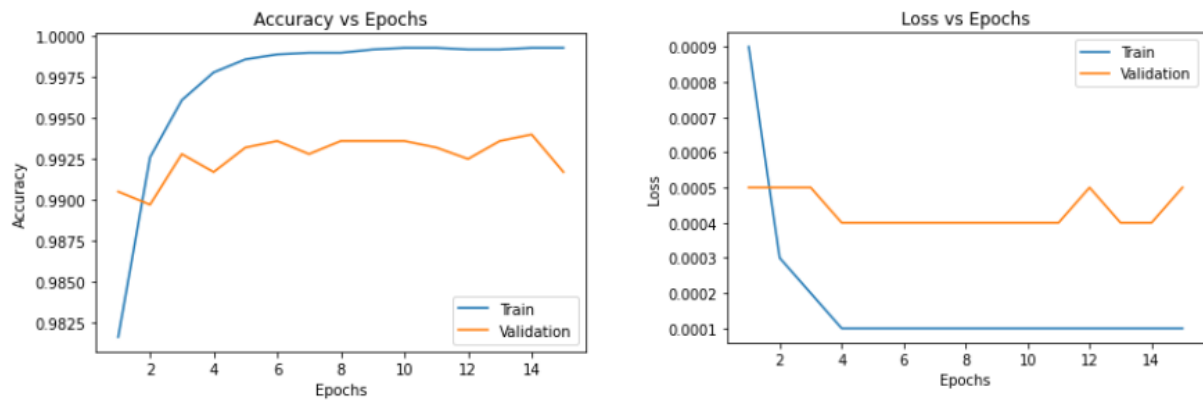


Figure 5.20: Resnet-18 accuracy and loss on train dataset for 3 classes model.

5.3 Visualizing some feature maps and filters

Next, the features maps are shown. As we said before, the feature maps are images/outputs of a layer after applying a group of filters, and then the output will be the new input for the next layer until it reaches the final layer. Features maps are helpful to understanding deep neural networks in a better way because we can see the features the model puts attention to and what filters are applied. Then, the features maps of each of the seventeen convolutional layers, after applying its filters, look as in Figure 5.21.

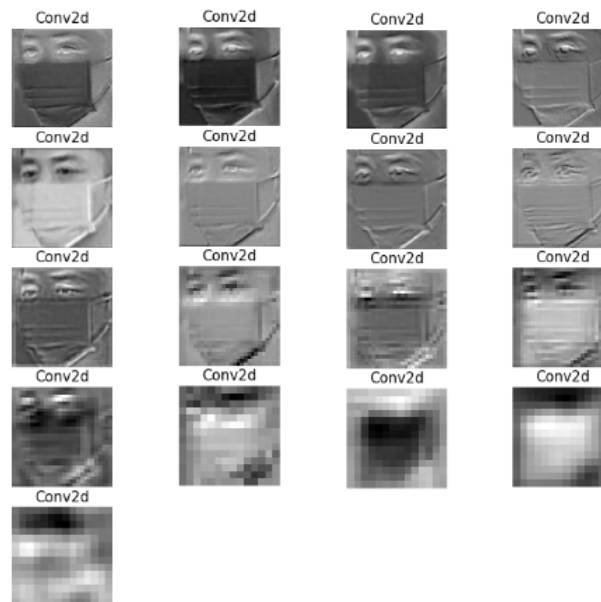


Figure 5.21: Features maps corresponding to each convolutional layers of Resnet-18.

5.3.1 ResNet-18 filters

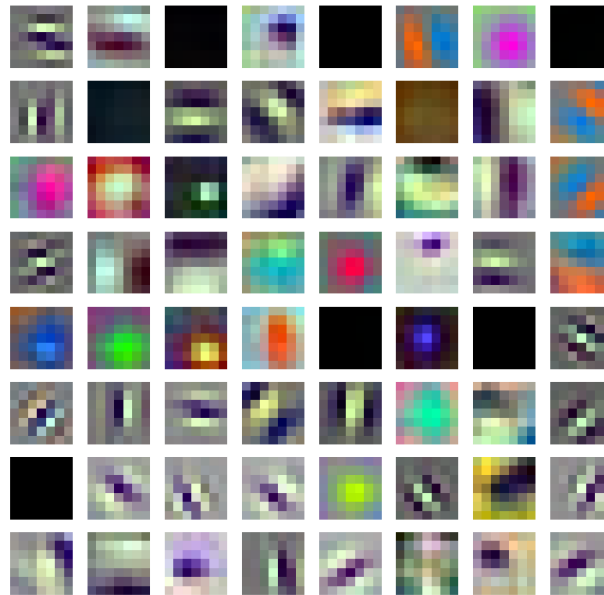


Figure 5.22: 64 Filters corresponding to the first convolutional layer of Resnet-18.

The values used to make the filter plots are the weights of the kernels. For example, the kernel size for the first convolutional layer is 7×7 (64 filters in total), and the filter looks as is present in Figure 5.22. As the image passes through the filters, it undergoes transformations. Next, in Figure 5.23, we have five images going through seven filters. The filters determine which parts of the image the model will focus on. Plotting the feature maps and filters in grayscale will make it easy to understand what is happening in a convolution. Therefore, we will plot all the 64 filters of the first convolutional layer together with the feature maps they produce. This result is plotted in Figure 5.24.

If we see one of the feature maps, we can see that some parts of it are dark and others are bright. This is due to the dark and the bright parts of the filters. The reason behind a part of the filter being dark or bright is the numerical value of the pixel. We know that a pixel is composed of three values on RGB images from 0 to 255 (values near zero to black color and values near 255 to white color). The values for the pixels are the weights. Low weights mean dark pixels, and high values mean bright pixels. Finally, the model will pay more attention to the parts of the image where the elementwise product between the weights and the pixel values are high. This means that the bright parts of the image are in charge of activating a particular layer's neurons depending on the values of its weights. This can

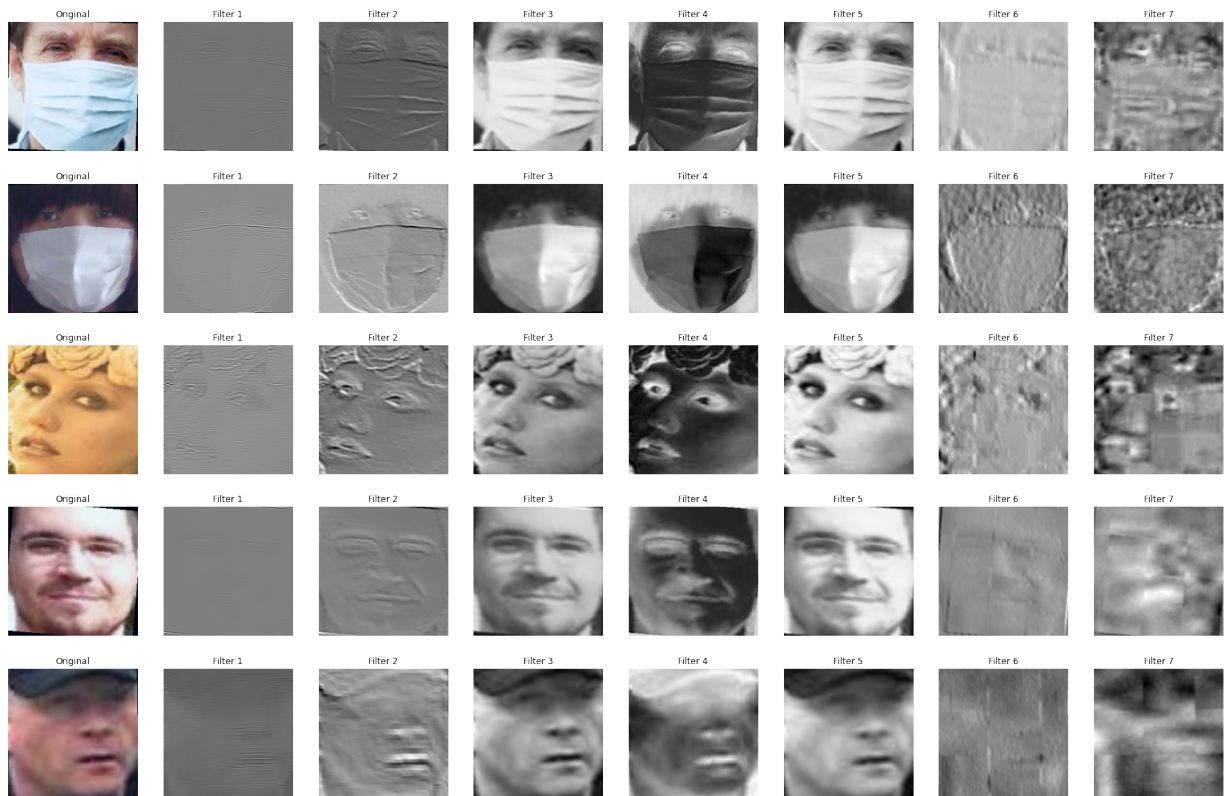


Figure 5.23: 5 images together after their features maps after applying seven different filters in the first convolutional layer.



Figure 5.24: All 64 filters of the first convolutional layer with the feature maps they produce.

be interpreted as what a neural network sees. In the particular case to classify the position of the COVID-19 face mask, we can see that some filters focus on the background of the

image while others focus on the person wearing the mask. In some cases, the background is dark, and the person becomes bright or vice versa. In addition, some filters produce an outline of the mask.

Finally, we will see the evolution of an image at first 64 filters of the convolutional layers 1, 5, 10, 15, and 17 with their respective feature maps in Figure 5.25. An important fact to consider is that the number of kernels of a convolutional layer can be greater than 64, but plotting all the kernels can be space-consuming. Therefore, we will plot the 64 filters of each convolutional layer for demonstrative purposes.

It is clear to see that, as going deep into the convolutional layers, the patterns on the features maps became more challenging to see for the human eye because the details of the image disappear, surely the neural network can identify the pattern to make the classification. The last feature maps look noisy, but they are the most important for the fully connected neurons (classification layer).

5.4 Grad-CAMs through the layers

Also, it is possible to see the Grad-CAMs' evolution through each layer. As we remember, ResNet-18 has 18 layers. However, the term layer has an additional meaning: a compound of two basic blocks. Therefore, taking this last meaning, ResNet-18 has four layers, each one composed of two basic blocks. In the same way, a basic block is formed by two operations, and an operation is formed by a convolution, a batch normalization, and a ReLU activation except for the last operation of a basic block. Therefore, we can appreciate the activations of each basic block after applying the operations over an image of each class. In this way, we can appreciate the beginning layer identifying general features, and then, as go deeper, the activations are focused on the COVID-19 mask face and around the nose (see Figure 5.26).

5.4.1 COVID-19 mask classification model with ResNet-18 against other approaches

In work [65], the authors use two different neural network combinations. First, they use MobileNetV2 as a feature extractor, and support vector machine (SVM) to make the clas-

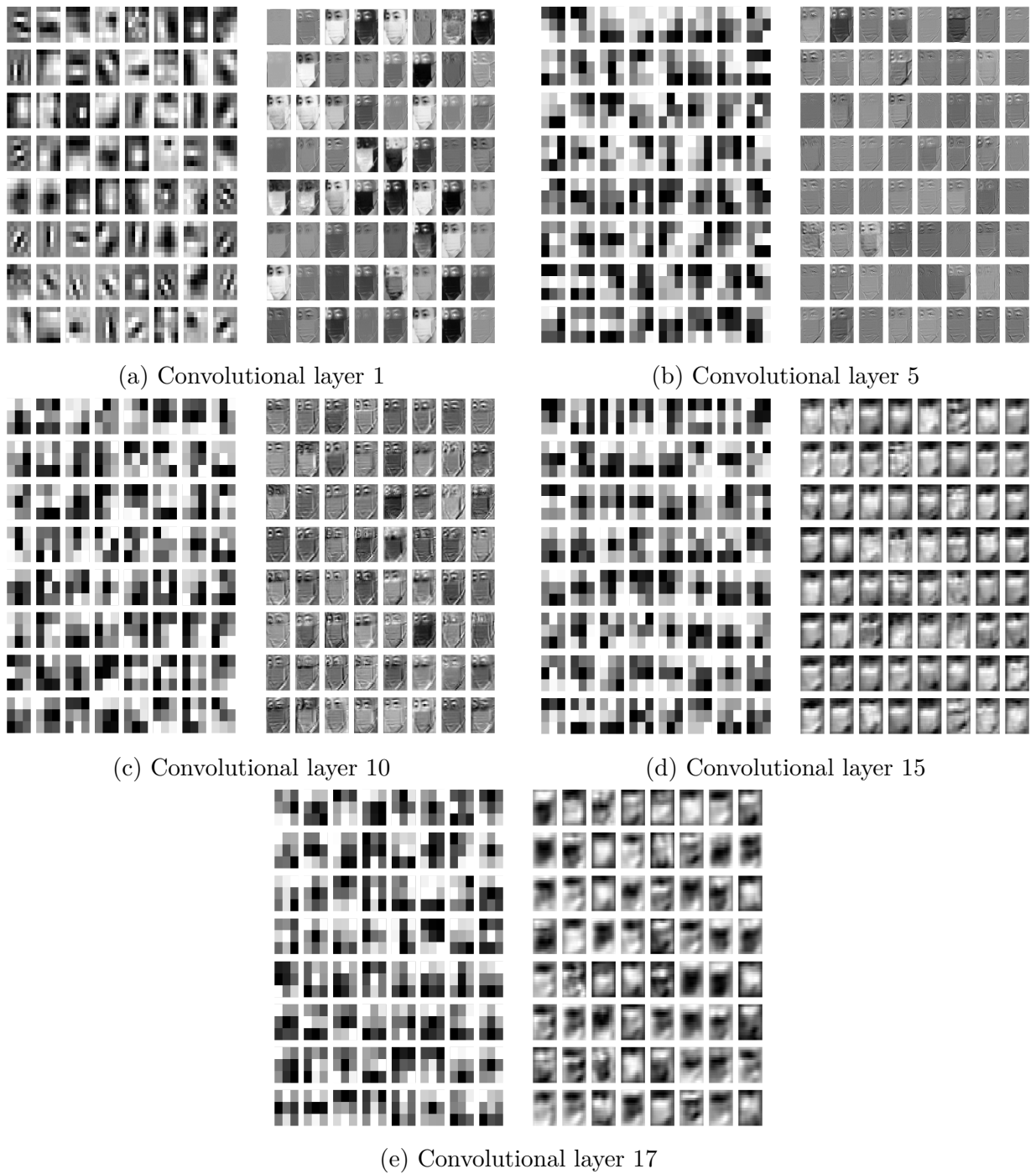
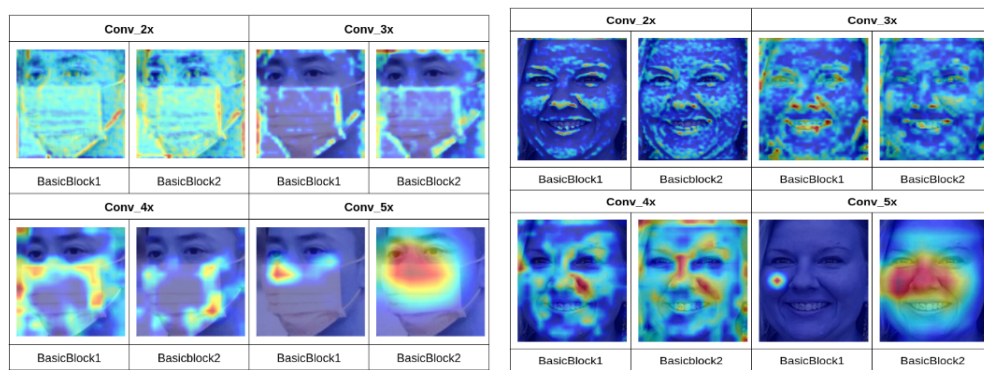
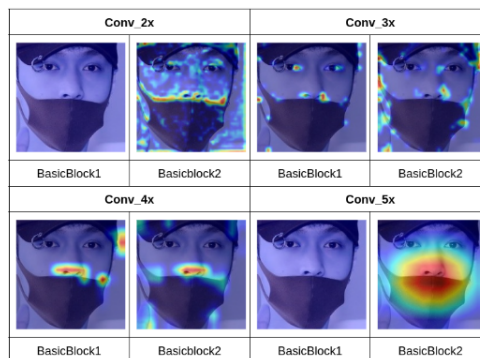


Figure 5.25: Filters with their respective featuremaps of convolutional layers 1, 5, 10, 15 and 17.



(a) Compliant

(b) Non-compliant



(c) Incorrect

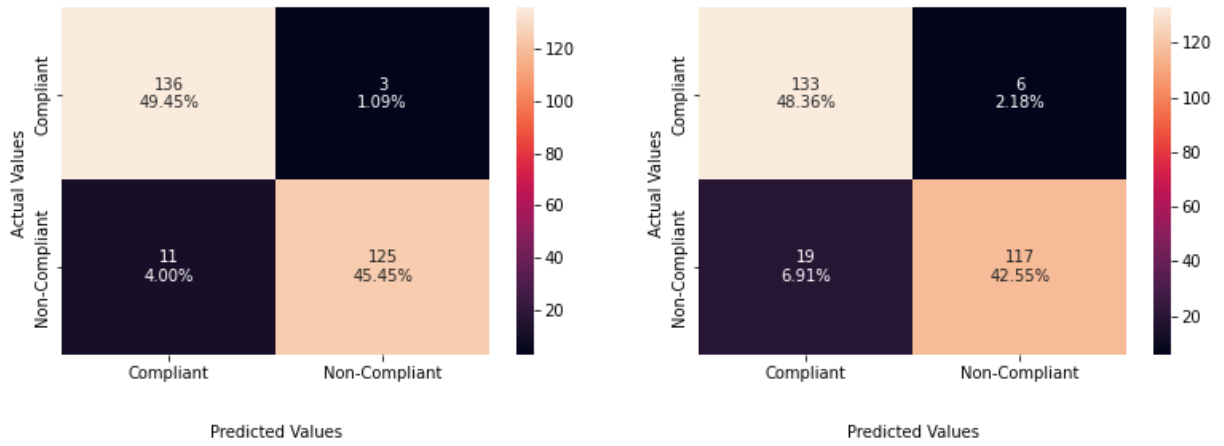
Figure 5.26: Grad-CAMs per convolutional layer for an image of each class.

sification. On the other hand, the second experiment uses VGG-19 as a feature extractor, and k -nearest neighbours (KNN) for the classification stage. The confusion matrix for both approaches is shown in Figure 5.27.

Table 5.7: ResNet-18 in comparisson with other approaches.

Models	Recall	Precision
MobileNetV2 - SVM	94.84%	95.08%
VGG19 - K-NN	90.09%	91.3%
ResNet-18	98%	98%

To enable a fair comparison, we used the same dataset with the same number of images to test the model (139 images for the compliant class, and 136 images for non-compliant class). After performing the classification with our model, the confusion matrix presented in Figure 5.28 was gotten. If we compare the three confusion matrices, it is clear that our model performs better classification, especially for the Non-compliant class, where our model reduces the wrong predictions from 11 (MobileNetV2 + SVM, Figure 5.27a) and 19



(a) MobileNetV2 - SVM

(b) VGG19 - K-NN

Figure 5.27: Confusion matrix corresponding to MobileNetV2 + SVM and VGG19 + KNN

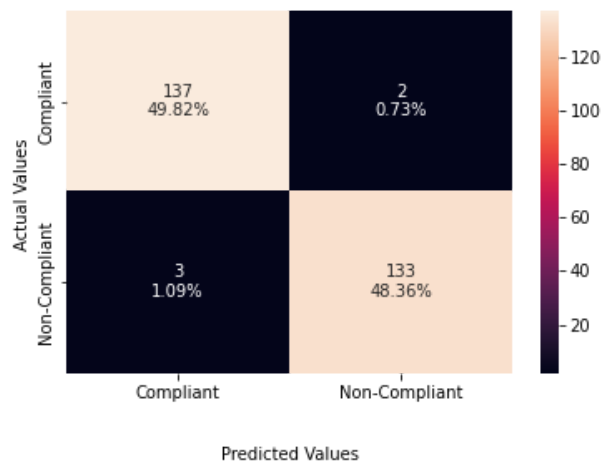


Figure 5.28: ResNet-18 results after be applied in the other dataset.

(VGG19 + KNN, Figure 5.27b) to only 3. In addition, we can use other metrics to see the results.

As we can see in Table 5.7, our Resnet18 has a recall and precision of 98%, an improvement of around 3% in relation to the better model presented in work [65], which is the MobileNetV2 with SVM.

Chapter 6

Conclusions

6.1 General conclusions

In this work, we present a CNN capable of classifying face masks wearing for use during the COVID-19 pandemic. Our proposal classifies the use of face masks in three different classes: compliant when people are wearing the mask well, incorrect when the person is not wearing the mask following the WHO guidelines (i.e., the mask should completely cover the mouth and nose), and non-compliant when people are not wearing any COVID-19 face mask. We start reviewing bibliographic material about different approaches to face the problem related to the classification of covid face mask-wearing from basic methods to the ones considered state of the art. Finally, we achieved our solid two-stage pipeline capable of performing the task assigned to this project. The metrics results showed our pipeline's excellent performance, encouraging its use in applications that requires a face detection, followed by a face mask classification stage.

Firstly, we testest DL models, especially ResNet family, using different optimizers in order to determine the performance of each one in the task of classifying the COVID-19 masks wearing into two classes: Compliant and Non-compliant. As a result of this preliminary study, our data showed that all models have an accuracy of over 94% (except ResNeSt-200 with RMSProp, 92%). ResNet-34 with Adagrad has been selected as the best classification model since the accuracy was 97.24% and it takes less time to train as it only has 34 layers compared with deeper models like Resnet-50 or Resnet-101.

Resnet-34 model and its accuracy of 97.24% were taken as the starting point of the work's second part. In this part, we improved the general performance and accuracy by applying different computer vision techniques both in the pre-process of the dataset and in the training of the classification model. Among the most important techniques, we have a cleanup, a relabeled, and a balance of the dataset; and the use of Torchvision transforms to perform data augmentation on each training batch guaranteeing the variability of the dataset. All these improvements lead to better results in the COVID-19 mask-wearing classification model in terms of accuracy, time to train, and the model's size up to even extend the number of classes from two (Compliant and Non-compliant) to three (Compliant, Incorrect, and, Non-compliant). The best model was a ResNet-18 for both cases: 99.6% accuracy for two classes and 99.2% accuracy for three classes, using Adagrad as an optimizer similar to the preliminary study. In addition, Hyperparameters setup is described in chapter 5 for both two and three classes models.

Furthermore, as a great model was obtained and to try to decipher what many people call a black box, we use Grad-CAMs to know which features or parts of the image the model pays attention to give a prediction for a specific class. Consequently, we discovered the model focused on the zone around the nose, especially the tip of the nose, to see the presence or absence of the COVID-19 mask and give a prediction. Additionally, we present many feature maps and the filters, i.e., the kernels which produce them across the model layers, to see the evolution of an image from the shallow layers until the last convolutional layer, the one before the fully connected layer, i.e., the classification layer.

6.2 Future work

Future work can be grounded on the deployment of our model in embedded systems. It can be helpful to transform our Pytorch COVID-19 mask-wearing classification model using NVIDIA TensorRT to make the deployment to production using NVIDIA DeepStream SDK. TensorRT lets us optimize our Pytorch model through techniques such as punning, layer/tensor fusion, kernel auto-tuning, etc., while NVIDIA DeepStream SDK provides us

with a multi-platform to deploy Vision AI applications and services to the real world. An important task that needs to be addressed is to increase the number of images available for the incorrect class dataset to extend the identification of the position of the mask if people are not protecting their nose or the chin. Finally, in terms of datasets, it would be helpful to try to do training with less number of images by class to probe if we can get higher or similar accuracy to the presented ResNet-18 model, with a dataset of less size given that, on this work, we show that with only 1100 images the model was capable of identifying incorrect class accurately.

Bibliography

- [1] J. Wu, M. A. Mamas, M. O. Mohamed, C. S. Kwok, C. Roebuck, B. Humberstone, T. Denwood, T. Luescher, M. A. De Belder, J. E. Deanfield *et al.*, “Place and causes of acute cardiovascular mortality during the covid-19 pandemic,” *Heart*, vol. 107, no. 2, pp. 113–119, 2021.
- [2] K. Yuki, M. Fujiogi, and S. Koutsogiannaki, “Covid-19 pathophysiology: A review,” *Clinical immunology*, p. 108427, 2020.
- [3] T. P. Velavan and C. G. Meyer, “The covid-19 epidemic,” *Tropical medicine & international health*, vol. 25, no. 3, p. 278, 2020.
- [4] J. Huynh, S. Li, B. Yount, A. Smith, L. Sturges, J. C. Olsen, J. Nagel, J. B. Johnson, S. Agnihothram, J. E. Gates *et al.*, “Evidence supporting a zoonotic origin of human coronavirus strain nl63,” *Journal of virology*, vol. 86, no. 23, pp. 12 816–12 825, 2012.
- [5] H. Elachola, S. H. Ebrahim, and E. Gozzer, “Covid-19: Facemask use prevalence in international airports in asia, europe and the americas, march 2020,” *Travel medicine and infectious disease*, vol. 35, p. 101637, 2020.
- [6] B. Y. Lee, “How israel ended outdoor face mask mandates with the help of covid-19 vaccines,” Apr 2021, url: <https://www.forbes.com/sites/brucelee/2021/04/20/how-israel-ended-outdoor-face-mask-mandates-with-the-help-of-covid-19-vaccines/?sh=63598b46680e>.
- [7] S. Feng, C. Shen, N. Xia, W. Song, M. Fan, and B. J. Cowling, “Rational use of face masks in the covid-19 pandemic,” *The Lancet Respiratory Medicine*, vol. 8, no. 5, pp. 434–436, 2020.

- [8] J. Howard, A. Huang, Z. Li, Z. Tufekci, V. Zdimal, H.-M. van der Westhuizen, A. von Delft, A. Price, L. Fridman, L.-H. Tang *et al.*, “An evidence review of face masks against covid-19,” *Proceedings of the National Academy of Sciences*, vol. 118, no. 4, 2021.
- [9] S. E. Eikenberry, M. Mancuso, E. Iboi, T. Phan, K. Eikenberry, Y. Kuang, E. Kostelich, and A. B. Gumel, “To mask or not to mask: Modeling the potential for face mask use by the general public to curtail the covid-19 pandemic,” *Infectious Disease Modelling*, vol. 5, pp. 293–308, 2020.
- [10] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, “Fighting against covid-19: A novel deep learning model based on yolo-v2 with resnet-50 for medical face mask detection,” *Sustainable cities and society*, vol. 65, p. 102600, 2021.
- [11] B. Batagelj, P. Peer, V. Štruc, and S. Dobrišek, “How to correctly detect face-masks for covid-19 from visual information?” *Applied Sciences*, vol. 11, no. 5, p. 2070, 2021.
- [12] H. i. t. Loop, “Medical mask dataset: Humans in the loop,” Jan 2022. [Online]. Available: <https://humansintheloop.org/resources/datasets/medical-mask-dataset/>
- [13] A. Kirillov, K. He, R. Girshick, C. Rother, and P. Dollár, “Panoptic segmentation,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 9396–9405.
- [14] M. Loey, G. Manogaran, M. H. N. Taha, and N. E. M. Khalifa, “A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the covid-19 pandemic,” *Measurement*, vol. 167, p. 108288, 2021.
- [15] B. Qin and D. Li, “Identifying facemask-wearing condition using image super-resolution with classification network to prevent covid-19,” *Sensors*, vol. 20, no. 18, p. 5236, 2020.
- [16] P. Hamet and J. Tremblay, “Artificial intelligence in medicine,” *Metabolism*, vol. 69, pp. S36–S40, 2017.

- [17] W. Wang and K. Siau, “Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity: A review and research agenda,” *Journal of Database Management (JDM)*, vol. 30, no. 1, pp. 61–79, 2019.
- [18] S. Ray, “A quick review of machine learning algorithms,” in *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, 2019, pp. 35–39.
- [19] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems.* ” O’Reilly Media, Inc.”, 2019.
- [20] P. Ongsulee, “Artificial intelligence, machine learning and deep learning,” in *2017 15th International Conference on ICT and Knowledge Engineering (ICT&KE)*. IEEE, 2017, pp. 1–6.
- [21] O. Campesato, *Artificial Intelligence, Machine Learning, and Deep Learning*. Stylus Publishing, LLC, 2020.
- [22] A. Krogh, “What are artificial neural networks?” *Nature biotechnology*, vol. 26, no. 2, pp. 195–197, 2008.
- [23] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-Dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, “Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions,” *Journal of big Data*, vol. 8, no. 1, pp. 1–74, 2021.
- [24] S. Xu, J. Wang, W. Shou, T. Ngo, A.-M. Sadick, and X. Wang, “Computer vision techniques in construction: a critical review,” *Archives of Computational Methods in Engineering*, vol. 28, no. 5, pp. 3383–3397, 2021.
- [25] A. Voulodimos, N. Doulamis, A. Doulamis, and E. Protopapadakis, “Deep learning for computer vision: A brief review,” *Computational intelligence and neuroscience*, vol. 2018, 2018.

- [26] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, p. 574, 1959.
- [27] —, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.
- [28] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.
- [29] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [30] M. Everingham, S. Eslami, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *International journal of computer vision*, vol. 111, no. 1, pp. 98–136, 2015.
- [31] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.
- [32] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, Big, Simple Neural Nets for Handwritten Digit Recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, 12 2010. [Online]. Available: https://doi.org/10.1162/NECO_a.00052
- [33] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.
- [34] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [35] M. Ferguson, R. Ak, Y.-T. T. Lee, and K. H. Law, "Automatic localization of casting defects with convolutional neural networks," in *2017 IEEE international conference on big data (big data)*. IEEE, 2017, pp. 1726–1735.

- [36] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [37] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-cam: Visual explanations from deep networks via gradient-based localization,” in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.
- [38] S. Yang, P. Luo, C.-C. Loy, and X. Tang, “Wider face: A face detection benchmark,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 5525–5533.
- [39] V. Jain and E. Learned-Miller, “Fddb: A benchmark for face detection in unconstrained settings,” UMass Amherst technical report, Tech. Rep., 2010.
- [40] X. Zhu and D. Ramanan, “Face detection, pose estimation, and landmark localization in the wild,” in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 2879–2886.
- [41] S. Ge, J. Li, Q. Ye, and Z. Luo, “Detecting masked faces in the wild with lle-cnns,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2682–2690.
- [42] A. Cabani, K. Hammoudi, H. Benhabiles, and M. Melkemi, “Maskedface-net—a dataset of correctly/incorrectly masked face images in the context of covid-19,” *Smart Health*, vol. 19, p. 100144, 2021.
- [43] Z. Wang, G. Wang, B. Huang, Z. Xiong, Q. Hong, H. Wu, P. Yi, K. Jiang, N. Wang, Y. Pei *et al.*, “Masked face recognition dataset and application,” *arXiv preprint arXiv:2003.09093*, 2020.
- [44] B. Roy, S. Nandy, D. Ghosh, D. Dutta, P. Biswas, and T. Das, “Moxa: A deep learning based unmanned approach for real-time monitoring of people wearing medical masks,” *Transactions of the Indian National Academy of Engineering*, vol. 5, no. 3, pp. 509–518, 2020.

- [45] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1. IEEE, 2001, pp. I–I.
- [46] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A convolutional neural network cascade for face detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5325–5334.
- [47] S. S. Farfade, M. J. Saberian, and L.-J. Li, "Multi-view face detection using deep convolutional neural networks," in *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, 2015, pp. 643–650.
- [48] X. Sun, P. Wu, and S. C. Hoi, "Face detection using deep learning: An improved faster rcnn approach," *Neurocomputing*, vol. 299, pp. 42–50, 2018.
- [49] S. Zhang, C. Chi, Z. Lei, and S. Z. Li, "Refineface: Refinement neural network for high performance face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 11, pp. 4008–4020, 2020.
- [50] J. Gao and T. Yang, "Face detection algorithm based on improved tinyyolov3 and attention mechanism," *Computer Communications*, vol. 181, pp. 329–337, 2022.
- [51] Y. Li, "Face detection algorithm based on double-channel cnn with occlusion perceptron," *Computational Intelligence and Neuroscience*, vol. 2022, 2022.
- [52] P. Wu, H. Li, N. Zeng, and F. Li, "Fmd-yolo: An efficient face mask detection method for covid-19 prevention and control in public," *Image and Vision Computing*, vol. 117, p. 104341, 2022. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0262885621002468>
- [53] M. Razavi, H. Alikhani, V. Janfaza, B. Sadeghi, and E. Alikhani, "An automatic system to monitor the physical distance and face mask wearing of construction workers in covid-19 pandemic," *SN computer science*, vol. 3, no. 1, pp. 1–8, 2022.

- [54] J. Zhang, F. Han, Y. Chun, and W. Chen, “A novel detection framework about conditions of wearing face mask for helping control the spread of covid-19,” *Ieee Access*, vol. 9, pp. 42 975–42 984, 2021.
- [55] S. K. Dey, A. Howlader, and C. Deb, “Mobilenet mask: a multi-phase face mask detection model to prevent person-to-person transmission of sars-cov-2,” in *Proceedings of International Conference on Trends in Computational and Cognitive Engineering*. Springer, 2021, pp. 603–613.
- [56] H. Lin, R. Tse, S.-K. Tang, Y. Chen, W. Ke, and G. Pau, “Near-realtime face mask wearing recognition based on deep learning,” in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2021, pp. 1–7.
- [57] Y. Xiong, K. Zhu, D. Lin, and X. Tang, “Recognize complex events from static images by fusing deep channels,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1600–1609.
- [58] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, and S. Zafeiriou, “Retinaface: Single-stage dense face localisation in the wild,” *arXiv preprint arXiv:1905.00641*, 2019.
- [59] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, H. Lin, Z. Zhang, Y. Sun, T. He, J. Mueller, R. Manmatha *et al.*, “Resnest: Split-attention networks,” *arXiv preprint arXiv:2004.08955*, 2020.
- [60] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, “Machine learning to improve multi-hop searching and extended wireless reachability in v2x,” *IEEE Communications Letters*, vol. 24, no. 7, pp. 1477–1481, 2020.
- [61] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization.” *Journal of machine learning research*, vol. 12, no. 7, 2011.
- [62] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

-
- [63] H. Robbins and S. Monro, “A stochastic approximation method,” *The annals of mathematical statistics*, pp. 400–407, 1951.
- [64] S. Ruder, “An overview of gradient descent optimization algorithms,” *arXiv preprint arXiv:1609.04747*, 2016.
- [65] A. Oumina, N. El Makhfi, and M. Hamdi, “Control the covid-19 pandemic: Face mask detection using transfer learning,” in *2020 IEEE 2nd International Conference on Electronics, Control, Optimization and Computer Science (ICECOCS)*. IEEE, 2020, pp. 1–5.