FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Cross-Sensor Face Detection

José Duarte Penetro Mesquita Ferreira

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Armando Jorge Sousa, FEUP

External Supervisor: Diego Santos Wandeley, CES

External Supervisor: Pedro Diogo Amorim, CES

June 28, 2021

Resumo

A distração e o cansaço de condutores na estrada são dos principais fatores causadores de acidentes rodoviários a nível mundial. Com o objetivo de minimizar o número destes acidentes, sistemas de monitorização do condutor, capazes de captar e avaliar as condições de condutores enquanto conduzem, têm vindo a ser desenvolvidos e implementados no interior dos veículos.

Estes sistemas de monitorização utilizam diversos tipos de câmaras (tais como RGB, térmicas ou infravermelho) capazes de operar em diferentes condições de luminosidade, permitindo extrair diversas informações relativas ao estado do condutor, tais como a sua atenção na condução, o seu grau de fadiga, sinais vitais, conforto térmico, entre outros. De modo a poder inferir estes fatores, é necessário detetar de forma autónoma e automática a face do condutor assim como pontos-chave (tais como boca, olhos ou testa) para recolha de informação. Para tal, algoritmos de visão computacional são implementados nestes sistemas de monitorização. Estes algoritmos podem ser baseados em técnicas de visão computacional clássica ou em métodos de aprendizagem profunda, tais como redes neuronais. Para cada tipo de câmara, abordagens diferentes são utilizadas para extrair informação das imagens na sua saída.

No entanto, com o elevado custo computacional de redes neuronais profundas e com as limitações de hardware de aplicações em tempo-real, a integração de diversas câmaras torna-se um processo demasiado dispendioso, sendo que a necessidade de capacidade de processamento aumenta com o número de câmaras no sistema.

Como tal, a implementação de um algoritmo singular, capaz de se adaptar a qualquer tipo de câmara e imagem, levaria a uma redução drástica nos custos de processamento.

Esta dissertação apresenta a verificação de que uma rede neuronal única é capaz de detetar a face para qualquer tipo de imagem fornecida. No geral, os resultados obtidos foram bastante positivos, conseguindo demonstrar que uma rede neuronal é capaz de generalizar e conseguir detetar a face de uma pessoa a partir de imagens capturadas com câmaras RGB, térmicas ou infravermelho. Também é demonstrado o efeito positivo que a aplicação de técnicas de transferência de aprendizagem têm nos resultados de deteção.

Palavras-chave: sistemas de monitorização do condutor, deteção de face, deteção de pontos-chave, visão computacional, aprendizagem profunda, redes neuronais convolucionais.

Abstract

Driver distraction and fatigue are some of the leading causes of car accidents worldwide. To prevent and minimise the number of these accidents, driver monitoring systems are implemented inside the vehicle, capable of extracting and evaluating drivers' conditions.

These monitoring systems use several types of camera sensors (such as RGB, infrared or thermal), capable of operating in different lighting conditions, whose image outputs allow the extraction of information on drivers' attentiveness, alertness, vital signs, thermal comfort and other factors. To extract this information, a process able to automatically and autonomously detect the driver's face and facial keypoints is required. To perform these detections, computer vision methods are implemented, using classical computer vision algorithms or deep neural networks, with the latter achieving much better detection results. For each type of camera sensor, different methods and solutions are used to extract information from the images provided.

However, with the high computational cost of deep neural networks and the hardware limitations of real-time applications, integrating several cameras is an expensive process, as the demand for processing power scales with the number of cameras. This is due to the fact that in the current approach, for each camera sensor a process is implemented.

Thus, the implementation of a single algorithm capable of adapting to each type of camera sensor and its' image, would drastically reduce processing costs.

The work developed in this dissertation verifies that a single neural network is capable of face detection for any type of image provided.

Overall, the results obtained are very positive, demonstrating that a single neural network can detect the human face for any image type, captured with RGB, thermal or infrared sensors. It is also demonstrated that applying transfer learning techniques for network training results in much better detection performances.

Keywords: driver monitoring systems, face detection, keypoint detection, computer vision, deep learning, convolutional neural networks.

Acknowledgements

Todos dizem que os agradecimentos são a parte mais fácil duma dissertação, mas já estou aqui há uns bons minutos a pensar no que escrever e pouco sai. Espero não me esquecer de ninguém.

Em primeiro lugar, gostaria de agradecer aos meus orientadores Diego Wanderley e Diogo Amorim, por todo o apoio, todas as lições e todo o conhecimento que me passaram nestes últimos meses. Sem vocês este trabalho não seria possível.

A toda a equipa de DMS da CES Porto, por me fazerem sentir bem vindo e integrado desde o primeiro dia. Não podia ter pedido um melhor grupo de pessoas com quem trabalhar.

Ao professor Armando Sousa, por me ter escolhido e orientado ao longo deste trabalho. As suas dicas e conselhos provaram ser extremamente valiosos.

Aos meus pais, Carlos e Paula, pelo constante apoio e orgulho que demonstram, pelos valores e pela educação que me deram. Esta conquista também é vossa.

À avó Tina, à avó Minda, ao avô Fredo e ao avô Penetro, pelo amor, apoio e orgulho incondicional que demonstraram ao longo deste percurso e ao longo da minha vida. Estarão para sempre comigo.

À minha irmã e a toda a minha família que esteve presente durante este caminho.

Ao Tiago, Mariana, Maria, Rita. Desde Setembro de 2004 (quando entramos no 1º ano) até o dia de hoje passaram 6144 dias. Desde que aprendi a escrever até à escrita desta frase vocês estiveram aqui. Que continuem por mais 6144 dias e por muitos mais.

Ao Fábio, António, Ana, Rafa, Brocas e Miguel. Amizades como estas são difíceis de encontrar. Que continuemos a partilhar histórias e a viver aventuras e momentos inesquecíveis juntos.

Ao Costa, Gonçalo e Luís, companheiros e amigos com quem percorri estes últimos 5 anos, e com quem vivi experiências que nunca esquecerei.

A todos os meus amigos e todas as pessoas que conheci na Faculdade de Engenharia da Universidade do Porto, e que me ajudaram de alguma forma a chegar aqui.

Obrigado.

José Ferreira

"There are many paths to the top of the mountain,
but the view is always the same."

Chinese Proverb

Contents

List of Figures

List of Tables

Abbreviations and Symbols

ANN	Artificial Neural Network
AP	Average Precision
AUC	Area Under the Curve
BCE	Binary Cross-Entropy
CE	Cross-Entropy
CES	Continental Engineering Services
COCO	Common Objects in Context
CNN	Convolutional Neural Network
DL	Deep Learning
DMS	Driver Monitoring Systems
DNN	Deep Neural Network
FIR	Far Infrared
FPN	Feature Pyramid Network
GPU	Graphics Processing Unit
IoU	Intersection over Union
ML	Machine Learning
MSE	Mean-Squared Error
NIR	Near-Infrared
NMS	Non-maximum Suppression
NN	Neural Network
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RPN	Region Proposal Network
SGD	Stochastic Gradient Descent
SL	Supervised Learning
UL	Unsupervised Learning

Chapter 1

Introduction

1.1 Context

Nowadays, there are an estimated 1.4 billion cars on the road worldwide [1] and nearly 1.25 million people are killed in car accidents each year [2]. That translates to an average of 3,287 deaths per day, with close to 1,000 of those deaths being from people under 25 years old. Car crashes are the leading cause of death for people between the ages of 15 and 29 [2]. To prevent and reduce the number of car accidents, driver monitoring systems (DMS) are applied, focused on identifying distracting activities while driving, one of the primary causes of accidents worldwide.

Driver monitoring systems use different camera sensors, whose outputs provide images that can be useful in different tasks. For example, near-infrared (NIR) cameras can operate in low-light conditions and therefore can be used to extract detailed landmarks of the face. Such landmarks are relevant for computer algorithms used for the evaluation of the driver condition (e.g. driver fatigue and drowsiness). Far infrared (FIR) cameras can produce thermal images where each pixel represents a temperature value. By analysing the pixels corresponding to the driver's forehead, its temperature can be obtained. All this information can be used to monitor driving conditions and help prevent one of the main causes of car disasters in the world.

1.2 Motivation

To evaluate the driver's condition, information on his/her vital signs, attentiveness and alertness, thermal comfort, and other factors, is obtained using different types of camera sensors that are placed inside the vehicle. Camera sensors can convert different wavelengths of light into two-dimensional images that allow human viewing. RGB camera sensors capture images from wavelengths inside the visible spectrum. Both infrared wavelength bands, on the other hand, are invisible to the human eye. NIR camera sensors can capture images from a lower frequency lightwave when compared to the visible spectrum, while FIR camera sensors can capture images from a higher frequency. Furthermore, these camera sensors react differently to varying light conditions. High power lights can create a glaring effect in FIR images while it goes unnoticed in NIR images.

Computer vision methods can be applied to autonomously detect the driver's face and his/her landmarks/keypoints. These methods can be developed using classical computer vision algorithms as well as using Deep Learning (DL) networks.

With the high computational cost of deep neural networks and the hardware limitations of real-time applications, integrating several cameras with deep neural networks is an expensive process, as the demand for processing power scales with the number of cameras. Furthermore, the recent increase in support for the use of electric cars results in the need to reduce energy consumption to increase the cars' efficiency.

For each type of camera, different methods and solutions are used to extract information from its output, with the integration in a single hardware or operating system (OS) made by different algorithms, resulting in the need for standalone algorithms running in parallel for each camera.

Consequently, an algorithm that adapts to any sensor type would drastically reduce the costs and processing power for suppliers, and thus provide a better experience to the end-user for a fraction of the price, increasing the number of possible system features and reducing energy consumption. That being said, the motivation behind this project laid in the need to determine whether a single deep neural network (DNN) can receive images from different light wavelengths and correctly detect the human face.

1.3 Goals

To verify if a single process can extract the required information from different wavelength images, there was the need to first create an annotated dataset, capturing images from different camera sources with several individuals in different poses. The development of this dataset is of key importance to the project, as this data will be used to train the detection models. A small-scale dataset with low diversity can lead to a problem described as overfitting, so it's important that the capturing process involves a large number of individuals and that the widest degree of variability is achieved.

The main goal is to ensure the detection of the region of interest (ROI) of the face using Deep Learning methods, more particularly using convolutional neural network (CNN) models. The predictions given by the models are compared with the ground-truth annotations, using selected metrics to evaluate their performance.

That being said, the following objectives were proposed for this project:

1. **Create a ground-truth system** containing annotated images captured from the three types of camera sensors;

2. **Detect the RoI of the face** for any type of provided image using a CNN model, and compare this Deep Learning method's performance with classical computer vision algorithms;

3. **Detect the respective facial landmarks** for any type of provided image using a CNN model.

1.4 Contributions

This work provides several contributions in the deep learning and computer vision fields. These contributions are:

- Literature review on:
  - Deep learning and deep neural networks
  - Convolutional neural networks and their working principles
  - Computer vision methods for face detection
  - State-of-the-art object detection CNN architectures
- Review on object detection and keypoint detection metrics
- Demonstration of the impact of transfer learning techniques on network training and performance
- Confirmation on whether a single deep neural network can perform face detection from different wavelength images
- Verification of the impact that different depth model backbones have in detection tasks

Chapter 2

Literature Review

This chapter introduces and explains the working principles that define the main approaches to face detection, more specifically Deep Learning methods and classical Computer Vision algorithms.

In Section 2.1, a description of how DNNs work and why they have achieved state-of-the-art results in recent years can be seen. Section 2.2 presents a more detailed explanation about how Convolutional Neural Networks function and the different architectures, with early-stage architectures being implemented for simple classification tasks and more recent ones being able to solve detection and regression problems. Finally, some of the main Computer Vision approaches as well as different CNN architectures used in Face and Object Detection are analysed in Section 2.3.

2.1 Deep Learning

Machine Learning (ML) is a field of computer algorithms that automatically improve the results of a given problem by learning from data [3], where learning is a procedure consisting of estimating the parameters of a model so that the trained model can perform a specific task [4].

Deep Learning is a subset of machine learning (Figure 2.1), based on Artificial Neural Networks (ANN) [5], a model inspired by the human brain, composed of layers of "neurons" that work as a block of mathematical operations linking entities [6]. In Section 2.1.2, a deeper explanation of these "neurons" is provided.

Deep Learning algorithms have gained immense popularity throughout recent years, achieving ground-breaking results in computer vision tasks such as image classification, image segmentation and object detection, and having shown impressive results in numerous contests in pattern recognition and machine learning [7]. Due to the rapid evolution in hardware, with better graphics processing units (GPU) that allow for faster algebraic calculations, and due to access to bigger datasets (huge collection of data from past years), Deep Learning is taking off [6] and is described as a universal learning approach capable of solving almost all kinds of tasks in different application domains [4].

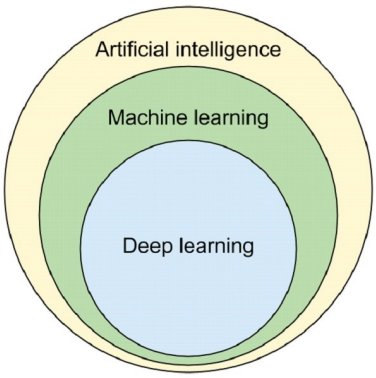


Figure 2.1: Subfields of Artificial Intelligence (from [8])

Machine Learning and Deep Learning tasks are usually described in terms of how the system should process a collection of features from the data or event [3]. In images, features are small descriptive or informative patches [5] such as edges, corners, textures, gradients or colours, that allow the detection, classification and segmentation of objects or landmarks of interest.

Before Deep Learning, a step called feature extraction was carried out to obtain the most relevant features [5]. In classical Machine Learning algorithms, human expertise is necessary for feature extraction, where the features are manually selected or selected through a specific extractor. Deep learning automatically learns the relevant features required for model processing, updating them through each iteration, given the update of model parameter's, and represents them hierarchically in multiple levels [4], as illustrated in Figure 2.2.

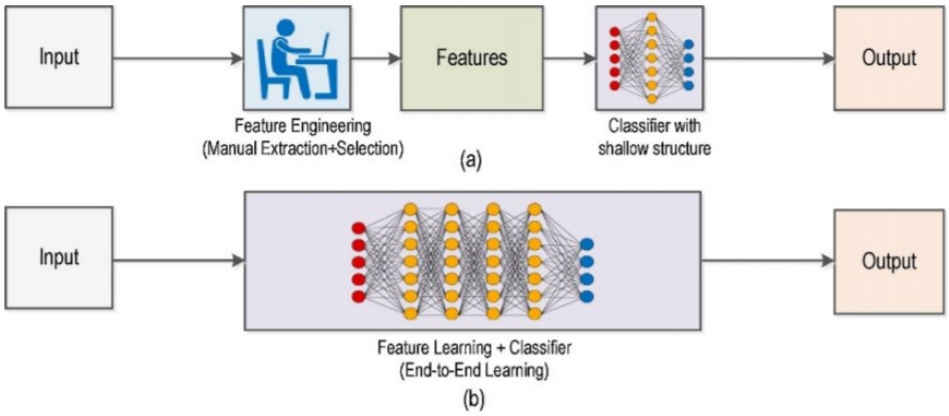


Figure 2.2: (a) Machine Learning workflow vs. (b) Deep Learning workflow (from [5])

2.1.1 Deep Learning Overview

Deep Learning approaches can be broadly categorised as supervised and unsupervised [3]. There are other approaches such as semi-supervised learning, which is a combination of both supervised and unsupervised learning approaches, and more recently reinforcement learning (RL). RL is often defined as an approach in which the algorithms interact with an environment and is often discussed as part of semi-supervised learning or sometimes as part of unsupervised learning [4]. However, in this Section only supervised and unsupervised learning will be addressed.

2.1.1.1 Deep Supervised Learning

Supervised Learning (SL) is a technique where the model is first trained on a dataset containing a set of inputs and corresponding outputs, or, in other words, using a labelled dataset. In this training phase, the model makes a prediction based on the given input and compares the predicted output with the ground-truth input, resulting in a loss value. Then, the parameters of the model are modified and updated (using optimisation algorithms detailed in Section 2.1.2) to reduce this loss value and obtain a more accurate prediction [4].

SL is the most popular approach in Deep Learning. Regression and classification problems are referred to as supervised learning problems [3], with Convolutional Neural Networks being one of the most popular deep supervised learning architectures.

2.1.1.2 Deep Unsupervised Learning

Unlike SL, Unsupervised Learning (UL) algorithms use an unlabelled dataset [3], where only inputs are provided. The model receives a dataset containing many features, and learns useful properties and important features from the input data, covering relationships and associations inside the dataset. Clustering problems are an example of UL, where the model divides the dataset into clusters of data with similar characteristics.

2.1.2 Deep Neural Networks

As mentioned before, throughout recent years Deep Learning has evolved remarkably, achieving groundbreaking results in several areas of the image analysis domain. However, the idea of trying to artificially recreate the behaviour of the human brain has emerged long before the recent development of Deep Learning.

The first attempt to mimic a biological neuron came in 1943 when McCulloch and Pitts developed the McCulloch-Pitts Neuron [9], a neuron based on a linear model that could only output true or false based on the inputs and the weights associated with each input, where these weights should be set manually [10].

Later, in 1958, Frank Rosenblatt introduced the Perceptron, a single layer neural network (NN) that would learn its parameters (weights) automatically [11]. However, due to the limitations of these Perceptrons, research on neural networks reached a stalemate until 1985, with Geoffrey Hinton's back-propagation algorithm revitalising the field [4]. Since then, numerous neural network architectures have been developed and improved, with an emphasis on CNNs, detailed in Section 2.2.

2.1.2.1 The Neuron

A standard NN consists of many simple connected processors called neurons [7]. These neurons receive a set of inputs and a bias, multiply each input by its respective weight, perform the overall sum, and return the output of an activation function applied to the mentioned sum.

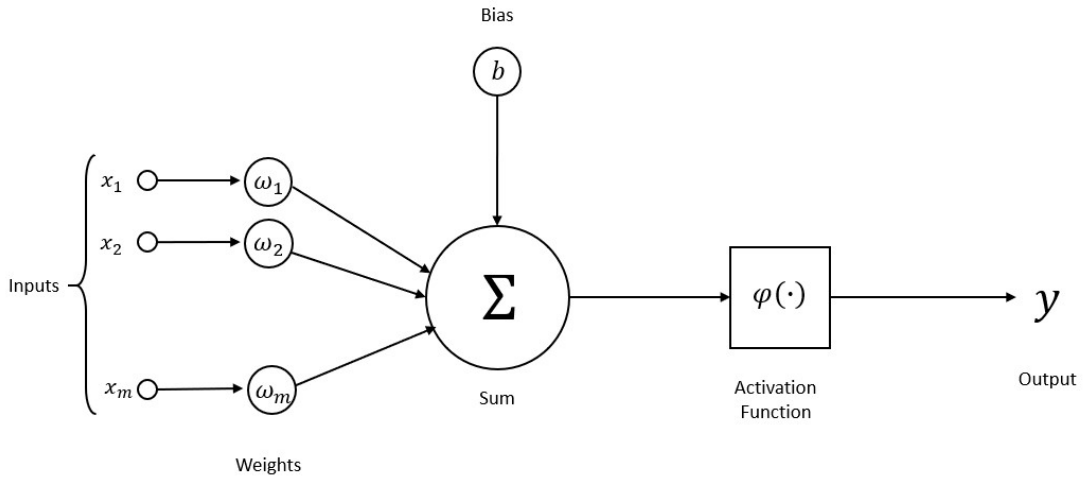


Figure 2.3: Scheme of a neuron

As seen in Figure 2.3, a neuron is divided into two main blocks, and the neuron output is defined as:

$$z = \sum w_m * x_m + b \tag{2.1}$$
$$y = \varphi(z), \tag{2.2}$$

where w_i is the weight associated with the input x_i, b is a bias factor and φ is a chosen activation function.

Combining both Equations (2.1) and (2.2), the output of the neuron can be defined as:

$$y = \varphi(X * W^T + b), \tag{2.3}$$

where X is a vector with the inputs ($X = [x_1, x_2, ..., x_m]$) and W is a vector with the weights ($W = [w_1, w_2, ..., w_m]$).

2.1.2.2 Feedforward and Backpropagation

A standard fully connected Neural Network consists of layers of a defined number of neurons, where each neuron of each layer is connected with each neuron from the next layer. The initial layer is the input layer, which receives the raw data, followed by a defined number of what are called hidden layers and ending with an output layer that returns the prediction of the network, as illustrated in Figure 2.4.

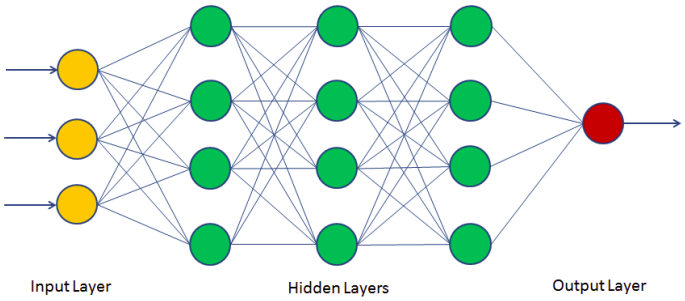


Figure 2.4: Neural Network Scheme

When a NN is trained, it's provided with an input x. This initial input from the dataset is then propagated through the network, in a sequence of regressions, each followed by an activation function, to produce a prediction $\hat{y}$. This process of propagating information through the network, from the input layer to the output layer, is called feedforward propagation. During training, feedforward propagation continues until a loss value $J(\theta)$ is produced [3], defined by a selected loss function, better detailed in Section 2.1.2.5.

In a reverse direction, backpropagation, or simply backprop, allows this loss information to propagate backwards through the network, using the gradient of the loss to learn and update the parameters. Backprop only refers to the method of computing the gradient of the function with

respect to its parameters. An optimisation algorithm (Section 2.1.2.6), based on a gradient descent method, performs the learning and the computation of the parameters, to minimise the value of the loss produced in the forward propagation process [3].

Gradient descent methods are used for minimising a given function $y = f(x)$, by updating the parameters in a certain direction (opposite the direction of the gradient) in the parameter space, to make a small improvement in y. A local minima is obtained when the value of $f(x)$ is lower than all neighbouring points, while a global minima is obtained with the absolute lowest value of $f(x)$ [3].

A parameter x is updated to minimise a function $f(x)$, using a gradient descent approach with a defined step ε, as described below:

$$x = x - \varepsilon \nabla f(x). \tag{2.4}$$

To summarise, training a NN can be divided in two processes: feedforward and backpropagation, where the goal is to minimise a loss function using gradient-based methods to update the NNs' parameters.

2.1.2.3 Parameters and Hyperparameters

An NN is an iterative algorithm that learns its parameters (weights W and bias b) through the iterations, and updates its values through back-propagation.

The design of an NN contains a set of parameters and hyperparameters. Hyperparameters are variables that can be tuned to improve the network or the functions that determine its behaviour. Some of these hyperparameters are the:

- Activation function, Section 2.1.2.4;
- Loss function, Section 2.1.2.5;
- Optimisation method, Section 2.1.2.6;
- Learning rate

  The step size considered during training in gradient-based methods. The choice of this hyperparameter is very sensitive. If a large value is chosen, the network may start to diverge instead of converging. If a very small value is chosen, it will take more time for the network to converge, and it may get stuck in local minima. To solve this problem, a common approach is to gradually update this value during training [4];
- Number of epochs

  Defines the number of times the learning algorithm will go through the entire training dataset;
- Batch size

Defines the number of samples of the training data to work through before updating the parameters;

- Number of layers in the NN;
- Number of neurons in each layer.

In the following sections, some of these hyperparameters are further explained.

2.1.2.4 Activation Functions

As mentioned before, the output of each neuron in an NN is defined by a chosen activation function. The activation function is a non-linear differentiable function that acts as a sort of transfer function and allows the propagation of selected data throughout the network [6]. The most commonly used activation functions are described in Table 2.1.

Table 2.1: Activation Functions

Function	Definition
Sigmoid	$\varphi(x) = \frac{1}{1+e^{-x}}$
Hyperbolic Tangent (tanh)	$\varphi(x) = tanh(x)$
Rectified Linear Unit (ReLU)	$\varphi(x) = max(0,x)$
Leaky ReLU	$\varphi(x) = max(0,x) + \alpha min(0,x)$

If these functions were linear, the model would be easy to optimise, but then the feed-forward network as a whole would remain a linear function of its input. Therefore a non-linear function must be used [3].

Out of the mentioned functions, the default activation function recommended for use with most neural networks is the Rectified Linear Unit (ReLU). For positive inputs, this function returns x and for negative inputs, it returns 0, as illustrated in Figure 2.5, which means this non-linear function preserves many of the properties that make linear models easy to optimise with gradient-based methods [3].

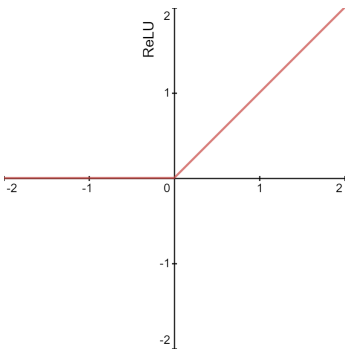


Figure 2.5: ReLU Activation Function

However, one drawback of the ReLU activation function is that it is not differentiable when its output is zero. Therefore, an improvement was made, adding a small slope when its input, x, is negative [3]. This small change makes the function fully differentiable, perfectly adequate for gradient-based learning. The Leaky ReLU activation function, whose graph is illustrated in Figure 2.6, fixes the value of this slope as a hyperparameter α (default $= 0.01$), while a parametric ReLU (PReLU), treats this scope value as a parameter to be learned by the network.

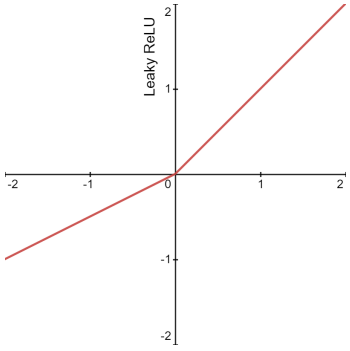


Figure 2.6: Leaky ReLU Activation Function with $\alpha = 0.5$

2.1.2.5 Loss Functions

The main goal of training a DNN is to minimise a differentiable function that measures the error associated with the prediction made by the network, assessing the network predictions quality by assigning a low/high loss value in correspondence to a correct/wrong prediction. This function, known as the loss function, is also chosen in the design of an NN and is the last computation step in the forward pass, and the first step in backward pass [12].

Making this loss function differentiable in relation to the parameters (W and b) ensures that efficient, gradient-based optimisation methods, from Section 2.1.2.6, can be used to find a global minima [13]. The loss function is defined as:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} L(\hat{y}_i^{(\theta)}, y_i), \tag{2.5}$$

where N is the size of the training data (or batch), θ are model parameters (W and b), $\hat{y}_i^{(\theta)}$ is the predicted value associated with the parameters θ and L is the chosen loss function.

The most common of these loss functions are the Mean Squared Error (MSE) and the Cross-Entropy (CE) loss. The MSE is defined as:

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} (\hat{y}_i^{(\theta)} - y_i)^2, \tag{2.6}$$

while the CE loss is defined as:

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N} y_i \log(\hat{y}_i^{(\theta)}), \tag{2.7}$$

where the variables N, θ and $\hat{y}_i^{(\theta)}$ are the ones defined previously.

The cross-entropy loss function is the most commonly used while training NNs, mainly for classification problems, as cross-entropy refers to the maximum likelihood, or the negative log-likelihood, between the training data and the model's output [3].

On the other hand, the MSE loss function often leads to poor results with gradient-based learning [3], one of the reasons why cross-entropy is the most used loss function.

Another loss function commonly used, especially in regression tasks, is the smooth-L1 loss (used in [14]), defined in the following equations:

$$\text{smooth}_{L1} = \begin{cases} 0.5(x_i - y_i)^2/beta, & \text{if } |x_i - y_i| < beta \\ |x_i - y_i| - 0.5 * beta, & \text{otherwise} \end{cases} \tag{2.8}$$

$$J(\theta) = \frac{1}{N}\sum_{i=1}^{N} \text{smooth}_{L1} \tag{2.9}$$

Both the CE and smooth-l1 losses, as well as other newly introduced losses (such as the Focal Loss in RetinaNet [15]), were used throughout the project for network training.

2.1.2.6 Optimisation

The loss function $J(\theta)$ tends to be a convex and differentiable function with a global minima and possibly several local ones.

The gradient descent approach is an algorithm used for finding the local minima of the loss function, which can or cannot be the global minima. Despite being used for training NNs successfully in the past decades [4], optimisation methods and algorithms were developed and are used to reach the global minima of the loss function, allowing for better results with lower error while at the same time reducing the training time and the computational cost.

Most Deep Learning optimisation methods divide the training data into batches, traditionally called minibatch methods, using more than one but less than all training examples at a time to compute the gradient of the loss function. Dividing a large training dataset into smaller batches allows the algorithm to converge much faster [3].

As far as optimisation algorithms go, there are basic algorithms, such as Stochastic Gradient Descent (SGD), with the addition of Momentum, the latter designed to accelerate learning. The SGD algorithm and its variants are the most used optimisation algorithms in Deep Learning, and consist in obtaining an estimate of the gradient by averaging the gradient on a minibatch, then updating the parameters based on this averaged gradient and a defined learning rate value. However, this algorithm can lead to slow convergence and learning. Adding momentum to SGD helps to

accelerate the convergence of the network by introducing a variable v that plays the role of velocity (direction and speed that the parameters move to reach the required values for the obtainment of the global minima). A hyperparameter $\alpha \in [0,1]$ is now created, associated with the velocity variable [3]. Algorithm 1, from [3], describes the SGD algorithm with momentum:

Algorithm 1: SGD with Momentum

Require: Learning rate ε, momentum hyperparameter α
Require: Initial parameters θ, initial velocity v
while θ *not converged* **do**
  Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)},...,\mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$
  Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(f(\mathbf{x}^{(i)};\theta),\mathbf{y}^{(i)})$
  Update velocity: $\mathbf{v} \leftarrow \alpha\mathbf{v} - \varepsilon\mathbf{g}$
  Update parameters: $\theta \leftarrow \theta + \mathbf{v}$
end

However, with these algorithms, a problem lies with the choice of the learning rate, which has a significant impact on the performance of a learning model. To solve this problem, several methods have been introduced that adapt the learning rates of model parameters [3].

Some of the most used algorithms are:

- AdaGrad

  The Adaptive Gradient (AdaGrad) algorithm [16] adapts the learning rate to the model parameters. The parameters with the largest partial derivative of the loss have a rapid decrease in their learning rate, and parameters with small partial derivatives have a small decrease in their learning rate [3]. In other words, it uses a different learning rate for every parameter θ_i, and it performs either larger or smaller updates depending on the parameter.

- RMSProp

  This algorithm (proposed by Tieleman & Hinton in 2012, unpublished) modifies AdaGrad, deriving from the need to resolve the issue of radically diminishing learning rates [17] and poor convergence when applied to a non-convex function [3]. It includes second-order momentum plus a slight change in the parameters update [6].

- Adam

  Deriving from "adaptive moment estimation", Adam [18] is designed to combine the advantages of AdaGrad and RMSProp while adding momentum. It stores an exponentially decaying average of past squared gradients (used in AdaGrad and RMSProp) and also keeps an exponentially decaying average of past gradients (used in momentum) [17].

  This method is shown to outperform other methods for several models and datasets [18] and is regarded as being very robust to the choice of its hyperparameters [3].

Adam is presented in Algorithm 2, adapted from [3] and [18].

Algorithm 2: Adam algorithm

Require: Step size ε (Default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1, ρ_2 in [0,1) (Defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilisation (Default: 10^{-8})
Require: Initial parameters θ
Initialise 1st and 2nd moment variables $\mathbf{s} = 0, \mathbf{r} = 0$
Initialise time step $t = 0$
while θ *not converged* **do**
    Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$
    Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $s \leftarrow \rho_1 s + (1 - \rho_1) g$
    Update biased second moment estimate: $r \leftarrow \rho_2 r + (1 - \rho_2) g \odot g$
    Correct bias in first moment: $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$
    Correct bias in first moment: $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$
    Compute update: $\Delta\theta = -\varepsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$ (operations applied element-wise)
    Apply update: $\theta \leftarrow \theta + \Delta\theta$
end

2.2 Convolutional Neural Networks

Classification and recognition tasks can be computationally very costly using DNN. For example, a 128×128 RGB image has $128 \times 128 \times 3$ pixels. Using a DNN with fully connected layers to classify an image of this dimension, with each pixel corresponding to an input, would mean 49,152 inputs would be provided to the network. If the first hidden layer of the network has a large number of neurons, then only in the first layer thousands or millions of weights would have to be computed. To fix this problem, a special type of NN was developed, the Convolutional Neural Network, specialised for processing data with a known grid-like topology, such as images [3].

This particular type of multi-layer neural network is used mainly in image and video recognition, image analysis, recommendation systems, natural language processing, and several other tasks [19]. This network structure was first proposed by Fukushima in 1988 [20]. However, it was not widely used due to hardware limitations at the time [4].

A decade after, LeCun and Bengio applied gradient-based learning algorithms to a CNN and obtained ground-breaking results in handwritten digit recognition [21]. Since then, major improvements have been made and several different CNN architectures have achieved tremendous results in many classification and recognition tasks. These architectures and their performance are better analysed in Section 2.2.2.

The name "Convolutional" comes from its fundamental building block, the convolution mathematical operation, described as:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{2.10}$$

where I is the input image matrix and K is a filter matrix (more commonly named kernel). The output of this operation applied to an image is often called a feature map [3].

Section 2.2.1 provides further detail on the CNN and its particular layers, explaining how a convolution layer, along with other types of layers, enables the extraction of high-level features from an image input.

2.2.1 Overview

A CNN's architecture is usually built of two main parts: the feature extractor and the classifier, illustrated in Figure 2.7, with the classifier being a fully connected network in this example.

The feature extractor can be seen as a small number of complex layers or a large number of simple layers, as illustrated in Figure 2.8. For better analysis, we'll treat the feature extractor as a large number of simple layers. This feature extractor is built around three layers: the convolution layer (Section 2.2.1.1), the detector layer (Section 2.2.1.2) and the pooling layer (Section 2.2.1.3).

To summarise, the various layers of a CNN are presented in Table 2.2, with X representing the input of the layer and y the output.

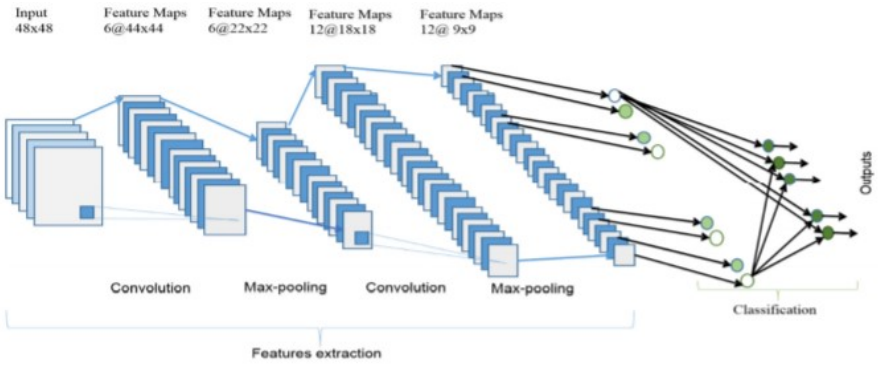


Figure 2.7: Overall architecture of a CNN, with a 48×48 image as input (from [4])

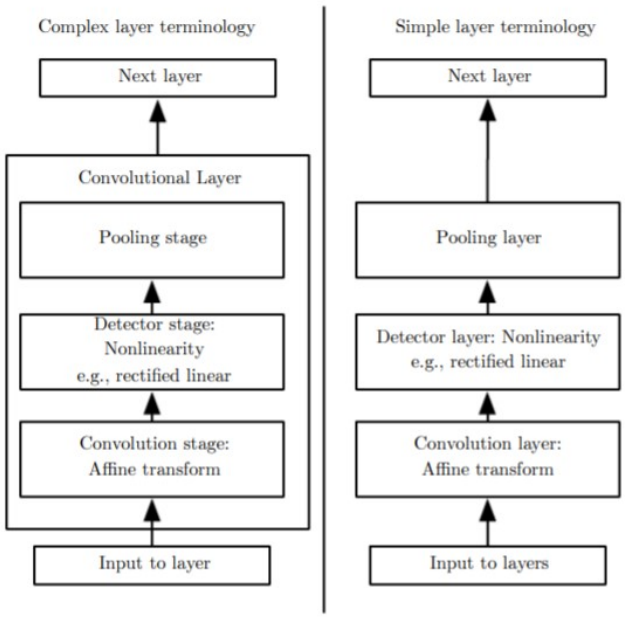


Figure 2.8: Components of a typical CNN. (*Left*) the layer of the feature extractor is seen as complex layer with different stages. (*Right*) every stage of the previous terminology is seen as an individual layer. (from [3])

Table 2.2: CNN layers

Layer	**Definition**
Convolution	$y = conv(X, kernels, p, s)$
Activation	$y = \varphi(X)$
Pooling	$y = pool(X, f_p, s_p)$
Fully connected	$y = X * W^T + b$
Final activation (SoftMax)	$y = \varphi(x) = \frac{e^x}{\sum e^x}$

2.2.1.1 Convolution layer

This layer is defined by applying several convolutions in parallel between an input image matrix and different filter matrices [3]. This operation, defined in Equation 2.10, is better visualised in Figure 2.9. The convolution product is applied through the image with a certain stride s.

The value of s defines the step used in the convolution operation, or, in other words, the number of pixels shifts in the input image matrix [22]. A large value of s changes the dimension of the output image, but normally a stride value of 1 is used in the convolution layer.

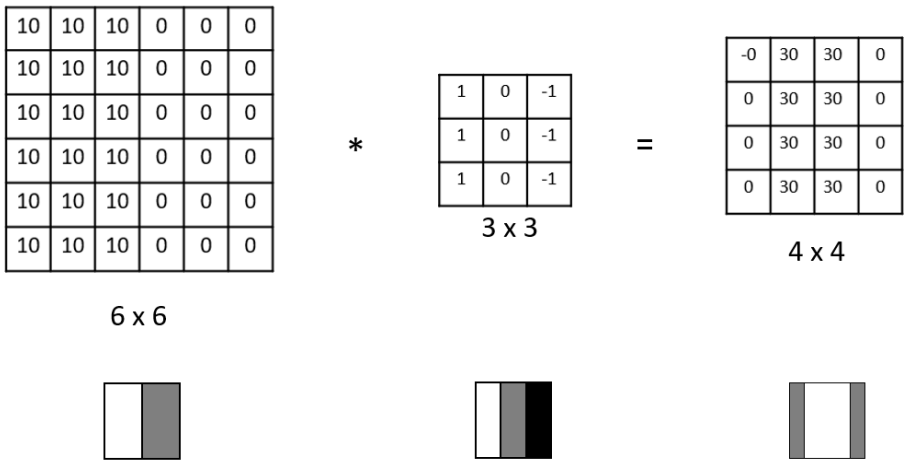


Figure 2.9: Convolution operation applied to a 6x6 image with a 3x3 filter

The values of a filter are learned by the network, similar to weights in a typical, fully-connected, DNN. Each convolution layer has many filters, each with a size of $c \times f \times f$, where c is the depth of the input image (an RGB image has 3 channels while a grayscale image has only 1 channel) and f is the filter hyperparameter, usually an odd number (most common filters in CNNs have a 3×3 or 5×5 dimension). A common 3×3 filter has 9 parameters that are learned by the network through back-propagation.

The output of the convolution operation is called a feature map (or representation), showing certain features of an image depending on the filter used in the convolution. The first convolution layers of a CNN extract low-level features such as corners or edges. These low-level layers are combined with high-level layers to extract higher-level features, used in the classification phase to classify the image. The particular filter shown in Figure 2.9 is a vertical edge detection filter, and when applied to an image, allows for the detection of vertical edges, as shown in Figure 2.10.

However, there are downsides to only performing the filtering product to obtain features of an image. As seen in Figure 2.9, the size of the output image decreases when compared to the input image every time a convolution operation is performed. The convolution product of an $(n \times n)$ image with a $(f \times f)$ filter results in an output image of $((n-f+1) \times (n-f+1))$ dimension. Another downside is that when convolution occurs, the pixels on the edges are only used once,

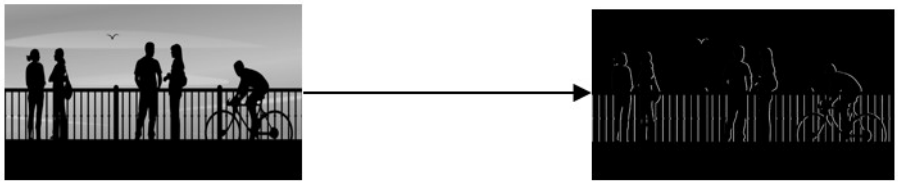


Figure 2.10: (*Left*) Input grayscale image. (*Right*) Feature map containing vertical edges

and pixels on the corners are less used than other pixels in the centre, resulting in information loss [22].

To solve these issues, before the convolution operation, padding is applied to the input image, adding a border of pixels around the image, typically with a value of zero (zero-padding, illustrated in Figure 2.11). Padding expands the size of the input image so that there is no loss in dimensions when convolution is performed.

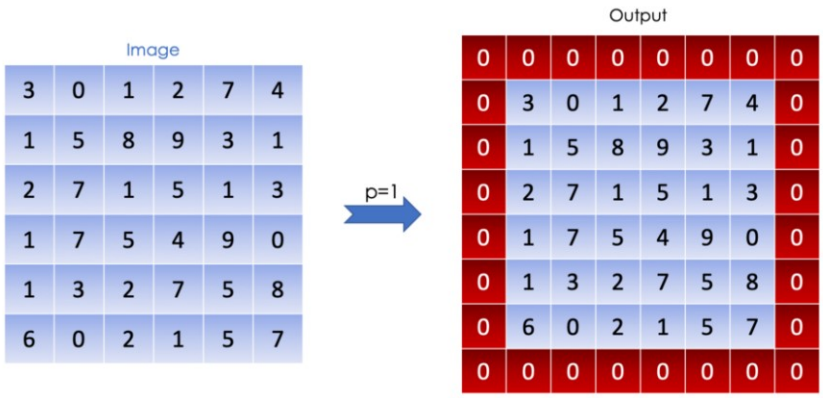


Figure 2.11: Zero-padding of a 6x6 image

The size of the output image of a convolution with padding p and stride s applied to an $(n \times n)$ image with an $(f \times f)$ filter will now be $(n+2p-f+1)/s \times (n+2p-f+1)/s$. When designing a CNN, the kernel size, as well as padding size and stride value are hyperparameters, chosen while defining its convolutional layers.

2.2.1.2 Activation layer

In the convolutional layer, several convolutions are performed in parallel to produce a set of linear activations. In the following layer, each of the linear activations produced previously is run through

a non-linear activation function, such as ReLU, defined in Section 2.1.2.4, ensuring the networks non-linearity [3]. This layer, in which an activation function is applied to the linear activations provided by the convolution layer, is often referred to as the activation or detector layer.

2.2.1.3 Pooling layer

After performing convolution on an input image, to reduce the complexity of the network and computational cost, an operation is required to progressively reduce the spatial size of the feature representation map [23].

The pooling function replaces the output of the previous layers with a summary statistic of the nearby outputs [3]. In other words, it summarises the information while downsampling the feature representation.

Similar to the convolution layer, a $(f_p \times f_p)$ window is slid across an image with a certain stride value s_p, selecting elements in the image. Padding can also be applied before pooling, although usually no padding is used. A function is then applied to the elements selected by the filter. Often, one of these two functions is used:

- **Average pooling**: The elements selected by the filter are averaged
- **Max pooling**: The maximum of all selected elements is returned

Figure 2.12 illustrates a 2×2 average pooling filter applied to a 6×6 image. In convention, a squared 2×2 filter is used with a stride $s_p = 2$ [22].

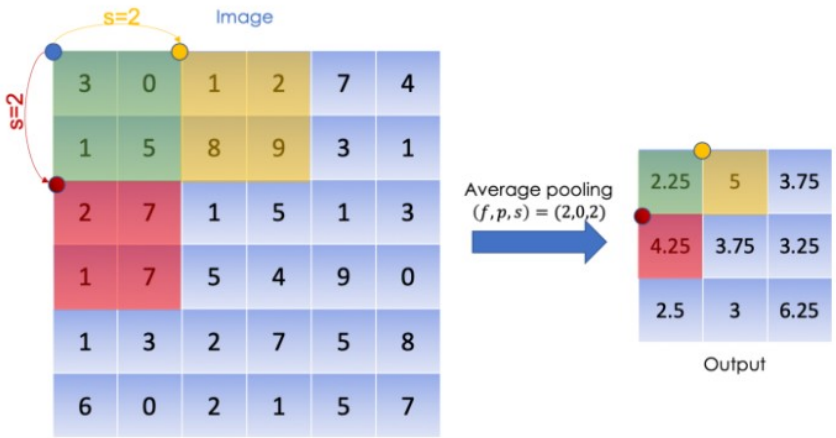


Figure 2.12: Average pooling of a 6×6 image

In addition to reducing the spatial size of the feature representation, pooling also helps to make it invariant to small translations in the input. If a translation of a small amount is applied to the input, most of the pooled output values do not change. This property is very useful to detect if a feature is present in the image rather than where the particular feature is. For example,

when determining whether a face is in an image or not, the exact location of the eyes with perfect accuracy is not required, only the information of whether there's a left eye and a right eye [3].

The three mentioned layers, namely, convolution, activation and pooling, are the main blocks of the feature extractor in a CNN, automatically extracting features from an image in a very computationally efficient way. Figure 2.13 shows an example of a feature map obtained after performing convolution and pooling to an image.

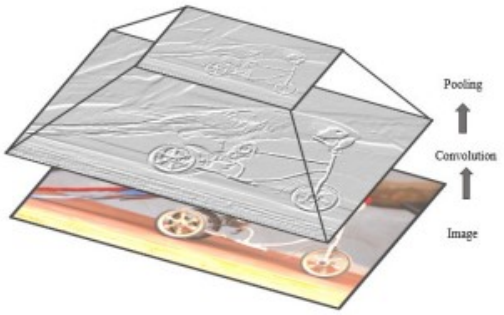


Figure 2.13: Feature map after performing convolution and pooling (from [4])

2.2.1.4 Classifier

Following feature extraction, the final feature maps are represented as vectors with scalar values which are passed into fully connected layers (DNN: Section 2.1.2), the classifier [4]. The feature vector values are then propagated through the fully connected layers (normally with a ReLU activation function), which can output not only a score value for given classes but also a high-dimensional structured object, for example, when pixel-wise labelling is required [3].

In between these fully connected layers, a dropout layer can be included to prevent overfitting and regularise the model. Dropout provides a computationally inexpensive and powerful method, removing neurons from a network by multiplying their output by zero with a certain probability value p [3].

In the last layer of the classifier, however, a SoftMax activation function is normally used to represent the probability distribution over n different classes [3]. This particular activation function is defined in Equation (2.11).

$$\varphi(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}} \tag{2.11}$$

2.2.2 CNN Performance

Being some of the first working deep networks trained with back-propagation, CNNs have played a key role in the history of Deep Learning. The arrival of modern, more powerful hardware has

allowed these deep networks to achieve state-of-the-art results in several tasks, winning numerous machine learning and computer vision contests throughout recent years.

A typical CNN is made of a key set of simple layers, as explained in Section 2.2.1. CNN architectures consist of stacks of these layers, organised in a particular structure, always beginning with a convolution layer. Out of the more popular architectures created, some are specialised for large-scale data analysis (such as the ResNet architecture), while others are only regarded as general architectures (such as the VGG-16 architecture) [4].

One of the most influential CNN architectures in history, the LeNet-5 [21] architecture, proposed by LeCun et al. in 1998, was designed to correctly identify handwritten digits. It consists of three convolutional layers, the first two followed by an average pooling layer, ending with two fully connected layers.

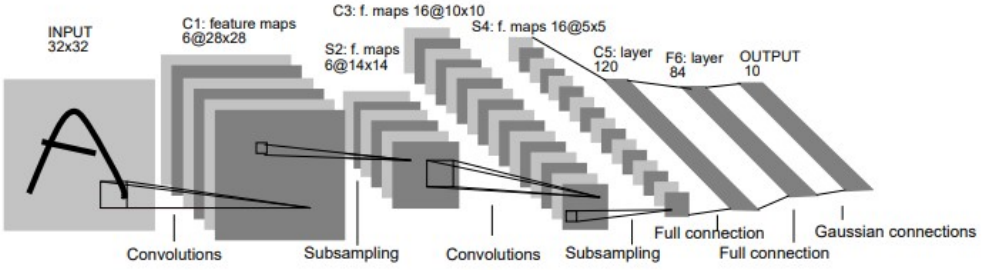


Figure 2.14: LeNet-5 architecture for digit recognition (from [21])

The convolution layers of this architecture use kernels of size 5×5 with a stride value of 1 and no padding, while the pooling layers use a kernel of size 2×2 with a stride value of 2. The output of the feature extractor is 120 1×1 feature maps. The values of these feature maps are then propagated through the two fully connected layers, which output a probability distribution of 10 different classes.

Since then, several popular state-of-the-art architectures have been developed. A large-scale challenge in computer vision is the ImageNet Large Scale Visual Recognition Challenge (LSVRC). A key turning point to CNNs was in 2012, when AlexNet [24] won the challenge, and ever since a CNN model has won this challenge, with ever-improving results. To demonstrate the improving performance and ground-breaking results of these different architectures towards LSVRC, Table 2.3 and Figure 2.15 show the accuracy in image classification on the ImageNet dataset, which has 1000 different classes to classify. This information illustrates how CNN performance has improved throughout recent years.

When designing the architecture of a CNN, there are a lot of points to take into account. The number and type of layers, the size of the kernels, the value of the stride and padding for both the convolution layers and the pooling layers, and many other points are carefully chosen. Table 2.4 compares some of the developed models mentioned earlier.

Table 2.3: Accuracy for ImageNet dataset classification (from [4])

Architecture Model	Year	Error (%)
AlexNet [24]	2012	16.4
VGG-16 [25]	2014	7.4
ResNet-152 [26]	2015	3.57
SENet [27]	2017	2.3
Human		5

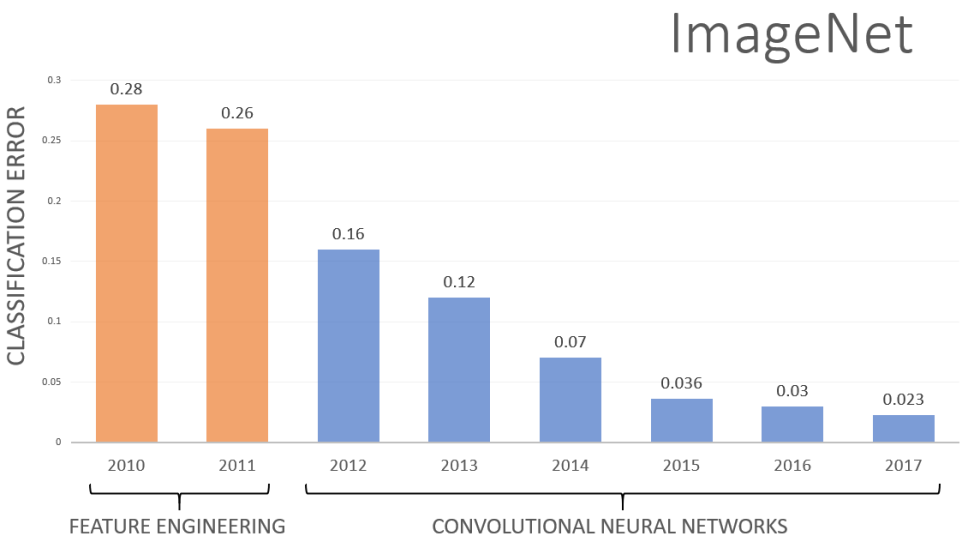


Figure 2.15: ImageNet dataset classification accuracy throughout recent years

Table 2.4: CNN architectures comparison (from [4])

Topic	LeNet-5	AlexNet	VGG-16	ResNet-50(v1)
Error (%) LSVRC	-	16.4	7.4	5.3
Input Image Size	32x32	227x227	224x224	224x224
Number of Convolution Layers	3	5	16	50
Filter size f	5	3,5,11	3	1,3,7
Feature Maps (input image $\rightarrow$ final convolution)	1-120	3-256	3-512	3-1024
Activation Function	*tanh*	ReLU	ReLU	ReLU
Total Weights	431 k	61 M	138 M	25.5M
Total Multiply and Accumulates (MACs)	2.3 M	724 M	15.5 G	3.9 G

2.3 Face Detection

Face detection has been one of the most studied topics in computer vision literature and is a key step in many computer vision applications such as facial landmark extraction, facial expression analysis, face recognition, animation, among others [28]. Given an image, the goal of face detection algorithms is to compute and determine if a face is present in the image and, if a face is present, return its location in the image. Facial landmark detection algorithms seek to identify the location of key facial landmarks, such as an eye corner or a facial contour.

Before the rise of Deep Learning architectures and techniques, several algorithms were developed to perform fast object and facial detection. Section 2.3.1 reviews popular computer vision algorithms applied to face and facial landmark detection.

However, Deep Learning methods have since then achieved state-of-the-art performance and results in this field. In Section 2.3.2, CNN architectures built specifically for object detection, which are also applied in face detection, are reviewed and detailed.

2.3.1 Classical Computer Vision Algorithms

2.3.1.1 Haar Cascade Classifier

Very few algorithms had a greater impact in the field of face detection than the Viola-Jones face detector (or Haar Cascade Classifier) [29]. Proposed by Paul Viola and Michael Jones in 2001, the algorithm made face detection feasible in real-world applications [30]. It combines the concepts of Haar-like features, Integral Images, the AdaBoost learning algorithm and the attentional cascade structure, to quickly and accurately detect objects in an image.

A Haar-like feature can be seen as a rectangular filter with dark and light features, used to extract useful information from an image, such as edges, and vertical, horizontal and diagonal lines. Figure 2.16 shows some of these Haar-like features, used in the Viola-Jones face detector.

Figure 2.16: Examples of Haar-like features

The model uses an integral image to compute these rectangular features and a learning algorithm called AdaBoost to select the best subset of the computed features. A cascade detector is then applied to check if a face is present in the image. This detector is a multi-stage classifier, capable of performing detection quickly and accurately, with a form of a decision tree called a "cascade" [29]. Figure 2.17 illustrates the first and second Haar-like features selected by AdaBoost for face detection. “The first measures the difference in intensity between the region of the eyes and a region across the upper cheeks. The feature capitalises on the observation that the

eye region is often darker than the cheeks. The second feature compares the intensity in the eye regions to the intensity across the bridge of the nose." [29].

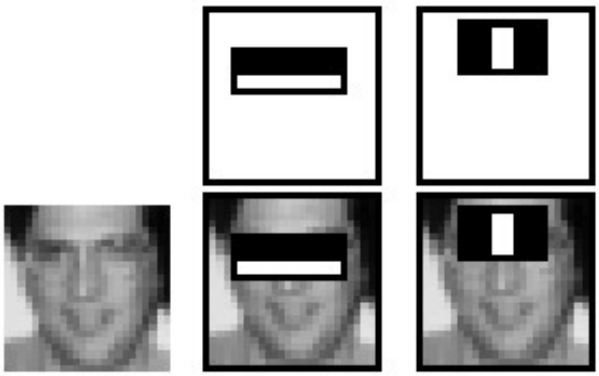


Figure 2.17: Haar-like features applied for face detection (from [29]).

Providing an input sub-window from the image, if a positive result is output in the first stage (if the first of the features is present in the image), the algorithm moves on to the second stage of the cascade and so on. If a negative result is obtained from a stage, the sub-window input is immediately discarded, and the next sub-window is analysed, as illustrated in Figure 2.18. These stages consist of classifiers produced by the AdaBoost algorithm.

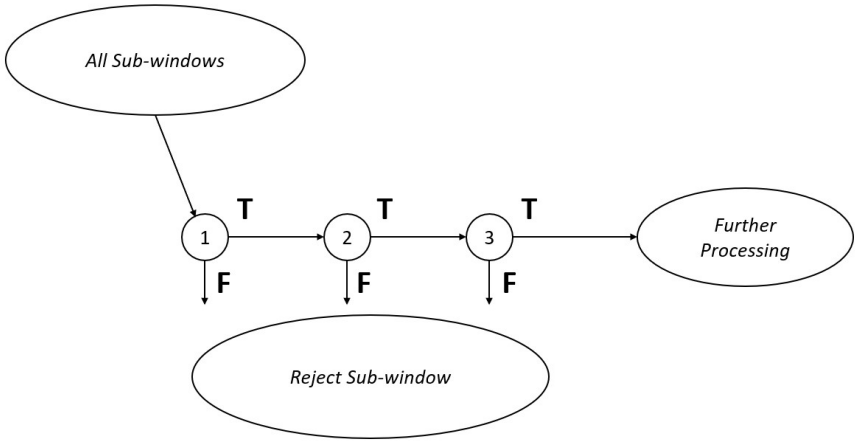


Figure 2.18: Schematic depiction of the detection cascade (from [29])

The combination of all mentioned concepts makes up the Haar Cascade face detector, one of the most influential object and face detection algorithms developed. For a deeper understanding of all the concepts and working principles of this algorithm, see [29].

2.3.1.2 Histogram of Oriented Gradients with Support Vector Machine

A classical algorithm for detection applies the combination of grids of Histograms of Oriented Gradients (HOG) and a linear Support Vector Machine (SVM) classification algorithm [31]. This combination, commonly named HOG+SVM, was first employed in [31] for human profile recognition, and since then several works have been published. In this sense, the HOG+SVM method is also applied in other fields, including face detection and recognition [32, 33].

HOG is a feature-based descriptor that uses a gradient vector orientation histogram to extract features from an image and construct a feature pool. Since HOG is a descriptor with 2D rotation invariance [33], it is a widely applied method for feature extraction in computer vision fields.

The descriptor feature extraction process can be divided into four steps. First, a calculation of each pixel gradient is performed by applying one-dimensional masks in both horizontal and vertical axis [34]. The second and third steps consist of dividing the image into cells, and perform histograms of the oriented gradients of each pixel in each cell are created. These HOG blocks can be of two types: rectangular HOG and circular HOG blocks [35]. Finally, these histogram blocks are normalised and the descriptor is assembled. Figure 2.19 shows the HOG descriptor extractor steps and Figure 2.20 shows the HOG features of an input image of a face. Each white point represents the gradients of one of the cells obtained from the input image.

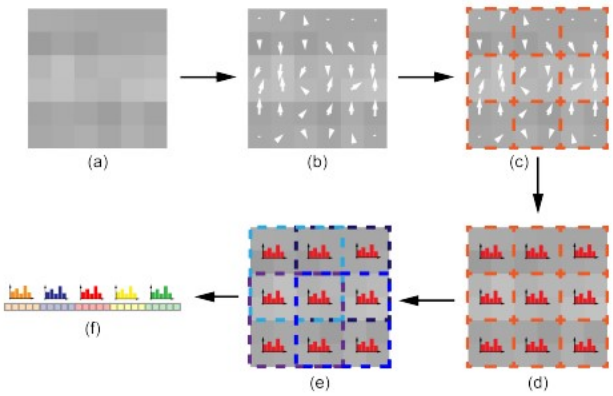


Figure 2.19: HOG descriptor extraction steps (from [34])

After the descriptor is assembled and the feature vectors are computed, a trained SVM classifier is used to classify these vectors correctly. The SVM is a supervised learning algorithm used mainly for classification and regression, better detailed in [36]. The algorithm works by mapping the data to a high-dimensional feature space, in a process called kernelling. And then finding a separator that allows the classification of the data.

2.3.2 Deep Learning

Most recent advances in Deep Learning techniques have drastically improved the performance of methods for face and facial landmark detection. This subsection will focus on state-of-the-art CNN

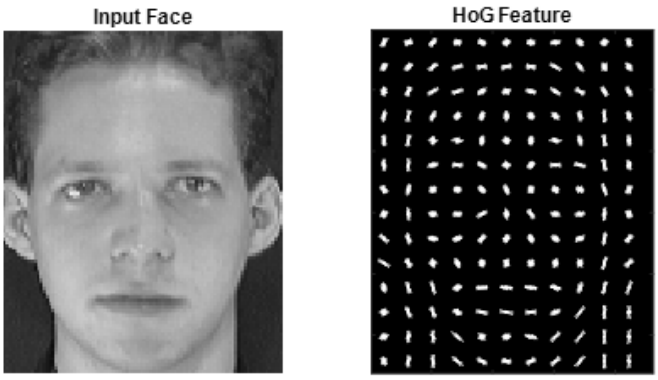


Figure 2.20: Circular HOG features of an image (from [35])

architectures for object detection. Current face detection methods have inherited the achievements of these general object detectors

Currently, there are two main types of object detectors. The two-stage detectors, such as R-CNN [37] and Faster R-CNN [38], first generate Regions of Interest (RoI), a set of regions that should contain all objects, filtering out the majority of negative locations of an image [39, 15], and then compute the features of these regions using a CNN architecture and classify those features. This approach, while computationally more costly and slower, achieves higher accuracy rates than one-stage detectors, such as YOLO [40] and RetinaNet [15]. The one-stage detectors treat detection problems as a simple regression problem [39], using a CNN to learn the bounding box coordinates and classify the object in a single shot (or inference) within an input image. This approach, despite usually achieving lower accuracy rates, is much faster than a two-stage approach.

2.3.2.1 Faster R-CNN

Two-stage object detectors are based on the generation of object region proposals, a set of candidate locations that can contain an object and then the feature extraction and classification of each region proposal.

The use of CNNs with region proposals for object detection was first introduced with the R-CNN [37], where a selective search is run on an image, extracting around 2000 region proposals. These proposals are warped into a square and forward propagated through the CNN which outputs a feature vector that is then classified by a trained SVM classifier. The choice of the CNN backbone has a large effect on performance. In [37], an architecture with top performance at the time was chosen, the OxfordNet, consisting of 13 convolution layers (3x3 kernels) and 5 max-pooling layers dispersed in between with three fully connected layers at the end. This two-stage detection method is called Regions with CNN features (R-CNN), illustrated in Figure 2.21.

However, R-CNN had some drawbacks. First, it takes a lot of time to process and classify the 2000 extracted regions, making the training time of the network extremely expensive. The model

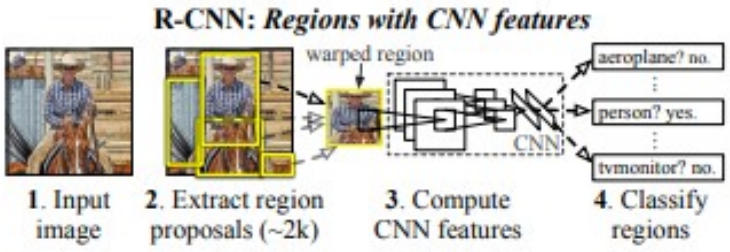


Figure 2.21: R-CNN system overview (from [37])

is also not useful in real-time applications, as it takes many seconds for the network to process each test image. A final drawback is that the selective search is a fixed algorithm with no learning, which can lead to the generation of bad candidate region proposals.

Some of these problems are fixed with new versions of R-CNN. The Fast R-CNN [14], by the same author, is a faster object detector with a similar approach to R-CNN. However, instead of feeding the region proposals to the CNN, the CNN generates a convolutional feature map from the input image, from which the region proposals are identified. A Region of Interest pooling layer is used to extract a feature vector from any valid region of interest. Each vector is then fed into a sequence of fully connected layers that finally branch into two output layers, one that produces probability estimates over the possible object classes and the other that produces a set of four values that correspond to the bounding box positions of one of the classes [14]. This architecture is illustrated in Figure 2.22. This method yields faster computation since the CNN does not have to process the 2000 regions extracted using R-CNN.

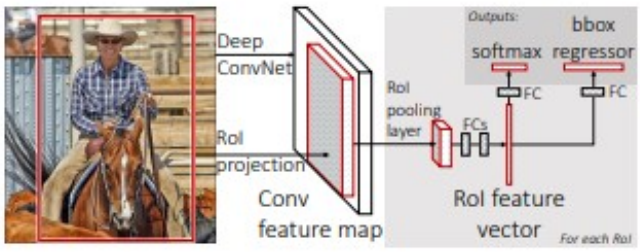


Figure 2.22: Fast R-CNN architecture (from [14])

In both cases, a selective search algorithm is employed, which is still a very time-consuming process. Faster R-CNN [38] is an improvement that aims to solve this problem by eliminating the selective search algorithm, letting the network learn the region proposals instead.

This detection system is composed of two independently trained modules. The first, a Region Proposal Network (RPN), is a fully convolutional network that takes an image of any size as input and returns a set of rectangular region proposals. The second is the Fast R-CNN detector that uses the proposed regions combined with the feature maps from the backbone. Both modules share

computation, meaning a shared common set of convolution layers is used. In [38], experiments are made using either 5 or 13 shareable convolution layers. This process is illustrated in Figure 2.23.

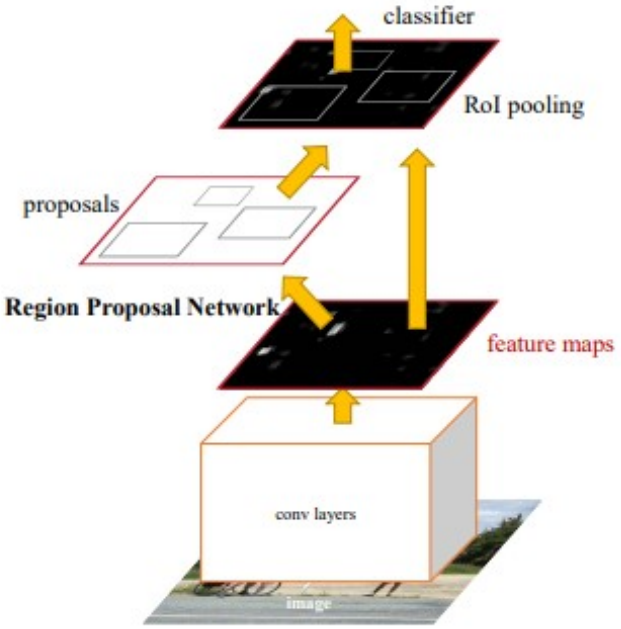


Figure 2.23: Faster R-CNN network (from [38])

To generate the region proposals, a sliding window approach is used, where the feature map backbone is the input. Each sliding window is mapped to a lower-dimensional feature, that is then fed into two fully connected layers (a box regression layer *reg* and a box-classification layer *cls*). At each sliding position, $k = 9$ anchors are yielded, with an anchor referring to a proposal relative to a reference box, meaning that a sliding window generates k proposals. The RPN diagram is illustrated in Figure 2.24.

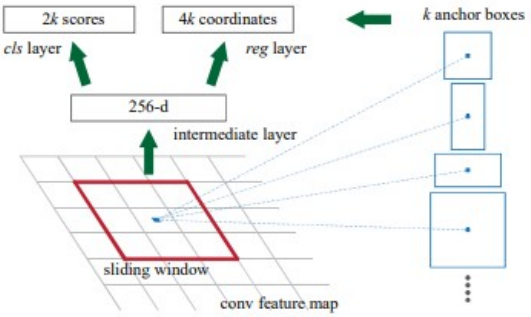


Figure 2.24: RPN approach (from [38])

As the network performs both classification and regression, two loss values are computed, the classification loss L_{cls} and regression loss L_{reg}. The loss function used while training the Faster

R-CNN is an objective function defined as:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i,p_i^*) \quad + \quad \lambda\frac{1}{N_{reg}}\sum_i pi^* L_{reg}(t_i,t_i^*), \tag{2.12}$$

where $L_{cls}(p_i,p_i^*)$, the classification loss, is defined as a log loss between the predicted probability p_i of an object detected in an anchor and the ground-truth label p_i^*, defined as 1 if the anchor is positive and 0 if the anchor is negative), and $L_{reg}(t_i,t_i^*)$, the regression loss, is defined as $R(t_i - t_i^*)$, with R being the smooth loss function used in [14], t_i the vector representing the coordinates of the predicted bounding box and t_i^* the ground-truth box associated with a positive anchor.

N_{cls} refers to the mini-batch size and N_{reg} to the number of anchor locations, both terms used to normalise the loss, while a balancing parameter λ is used to equally weigh both *cls* and *reg* parameters. By default, a value of $\lambda = 10$ is used.

The original training of the RPN was done using SGD with momentum, and the training of the whole model was done through a process named Step Alterning Training [38]. Both modules are trained separately and then combined to form a unified network.

The Faster R-CNN allows for the development of a unified object detection system capable of running at near real-time frame rates, while also improving region proposal quality and overall detection accuracy.

2.3.2.2 RetinaNet

As mentioned before, one-stage detectors, while fast, are still far behind in terms of accuracy, when compared to two-stage detectors. The RetinaNet was designed to verify whether one-stage can match or even surpass two-stage detectors in terms of accuracy performance [15].

RetinaNet is a single network composed of a feature detection backbone based on ResNet with a Feature Pyramid Network (FPN) [41] on top and two task-specific subnetworks, one used for classification and the other for bounding box regression.

One of the problems of one-stage object detectors occurs when there is an extreme imbalance between foreground and background classes during training. To solve this problem, a novel loss function was introduced, based on the binary cross-entropy (BCE) loss used in classification, the Focal Loss. BCE is described as:

$$\text{BCE}(p,y) = \begin{cases} -\log(p), & \text{if } y = 1 \\ -\log(1-p), & \text{otherwise} \end{cases}, \tag{2.13}$$

where $y \in \{\pm 1\}$ is the ground-truth class and $p \in [0,1]$ is the model's estimated probability for the class in which $y = 1$. In [15], p is redefined as p_t for notational convenience:

$$p_t = \begin{cases} p, & \text{if } y = 1 \\ p-1, & \text{otherwise} \end{cases}, \tag{2.14}$$

with the BCE then applied as:

$$\text{BCE}(p_t) = -\log(p_t). \tag{2.15}$$

Finally, the Focal Loss function is then introduced, adding a modulating factor $(1-p)^\gamma$ to the BCE function:

$$\text{FL}(p_t) = -(1-p_t)^\gamma \log(p_t), \tag{2.16}$$

where $\gamma \geq 1$ is a "focusing" hyperparameter, allowing the loss function to focus training on hardly classified negatives and not on the easily classified negatives that comprise the majority of the loss [15].

An α-balanced variant is used in practice, yielding slightly improved accuracy over the original function. A weighing factor $\alpha \in [0,1]$ is introduced, with α_t defined analogously to p_t. The final Focal Loss can be defined as:

$$\text{FL}(p_t) = -\alpha_t(1-p_t)^\gamma \log(p_t). \tag{2.17}$$

The use of this novel loss allows this one-stage model to achieve accuracy levels only reached by two-stage models.

Regarding the architecture of the detector, Figure 2.25 illustrates RetinaNet's framework. The feature extractor uses an FPN backbone on top of a ResNet architecture. Originally, the FPN model is a two-stage detector with state-of-the-art results [41]. In RetinaNet it is used to construct a rich multi-scale feature pyramid, where each level of the pyramid contains feature maps that can be used to detect objects at a different scale.

In addition, RetinaNet uses translation-invariant anchor boxes, each assigned a one-hot vector of classification targets with the length equal to the number of classes and a vector of box regression targets. There are 9 anchors per pyramid level.

For each pyramid level, two subnets are attached. The classification subnet predicts the probability of object presence at each spatial position for each of the A anchors and K classes. This subnet is composed of four convolutional layers, each with 3×3 filters and ReLU activations. Finally, a convolution layer with $KA3 \times 3$ filters is used, with a sigmoid activation to return the KA binary predictions per spacial location [15].

The box regression subnet works in parallel with the classification subnet, to regress the offset from each anchor box to a nearby ground-truth object, if one exists. It is identical to the classification subnet, terminating in $4 \times A$ linear outputs per spatial location [15].

The original RetinaNet was trained in [15] with the SGD optimiser with momentum, with a decaying learning rate starting as 0.01. Evaluating the performance of the trained network, it is shown that it reaches state-of-the-art results, performing even better accuracy-wise than top-performing two-stage detectors.

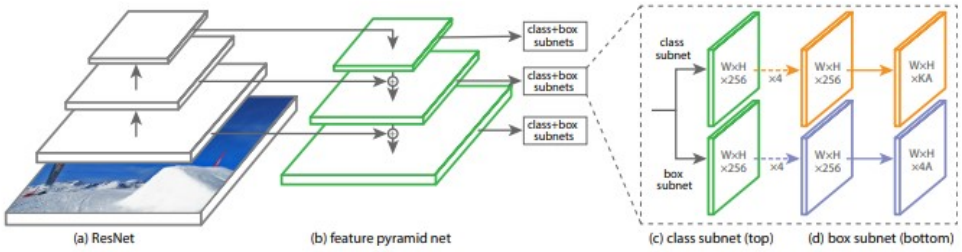


Figure 2.25: RetinaNet architecture (from [15])

2.4 Summary

This chapter's literature review focused on three topics.

The first topic, related to DL, reviewed the basics of deep learning and the working principles of deep neural networks. Section 2.1.1 addresses the two main types of DL methods, briefly explaining supervised and unsupervised learning methods. Section 2.1.2 goes deeper in detailing the key concepts that make a deep neural network, including activation and loss functions and optimisation algorithms. The understanding of these concepts is essential when building and training a Deep Learning model to perform a specific task.

The second topic addresses Convolutional Neural Networks, detailing the main layers that compose this kind of neural networks and why they have gained much interest in recent years. Section 2.2.1 details the general architecture and the main layers used in CNNs, while Section 2.2.2 demonstrates some ground-breaking architectures and state-of-the-art accuracy results achieved by these architectures throughout recent years.

Finally, Section 2.3 details both computer vision and deep learning approaches to face and object detection. In Section 2.3.1, two popular and accurate computer vision methods for face detection are described, and in Section 2.3.2 two popular CNN models used for object detection are detailed. Both can also be applied to face detection.

Chapter 3

Methodology

In this chapter, the overall methodology of the project is presented. Section 3.1 provides a detailed analysis of the datasets used in the project, including a private dataset created specifically for this project. In Section 3.2, an overview of the metrics used to evaluate the performance of each model is presented.

The different models implemented in this project are detailed in Section 3.3, and the machine learning concept of transfer learning is explained in Section 3.4. Finally, a description on how the models were trained and tested is provided in Section 3.5.

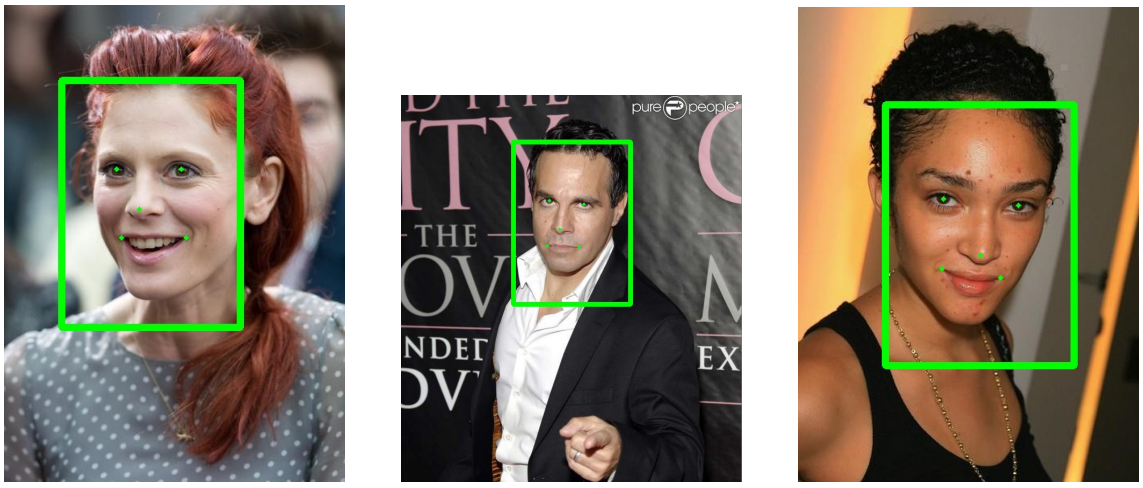
3.1 Datasets

One of the most important aspects of training ML and DL models is the data used to train them. The first step of the process to train and evaluate the performance of each model was to gather data, either from different datasets of different image types or by creating and annotating a dataset specific for this project. This section goes into greater detail in describing each of the datasets used for this project.

3.1.1 CelebA

The CelebFaces Attributes (CelebA) Dataset [42] is a large-scale face attributes dataset. It contains 202,599 RGB images of faces of over 10,000 identities, each with annotations for five facial keypoints and a bounding box for each face. This public dataset was chosen due to the large diversity in terms of race, gender, age, ethnicity, accessories used and other countless attributes, and the large quantity of images, covering different poses and backgrounds.

This was one of the datasets used for training and evaluation of the models in face detection tasks in RGB images (visible spectrum). Some examples of images with annotations of this dataset are shown in Figure 3.1.



(a) (b) (c)

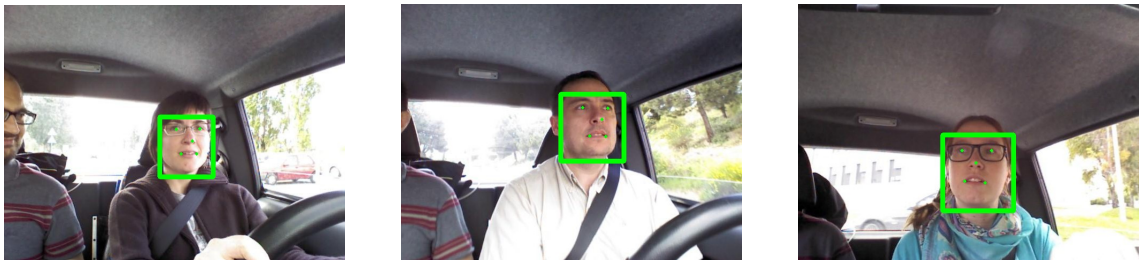
Figure 3.1: CelebA dataset images with ground-truth annotations

3.1.2 DrivFace

The Driver Face (DrivFace) dataset [43] is a small-scale face attributes dataset. It contains 606 RGB images of four subjects (two male and two female) driving in real scenarios.

Each image contains annotations for a face bounding box and five facial landmarks. It also contains labels for three possible gaze direction classes, depending on whether the subject is looking right, left or front.

One of the advantages of using this dataset is that it features the ideal scenario for this project (drivers in real scenarios gazing at multiple directions). However, due to the small number of images and lack of variability, this dataset was used for testing purposes and smaller training sessions. Some images with annotations from this dataset are shown in Figure 3.2.



(a) (b) (c)

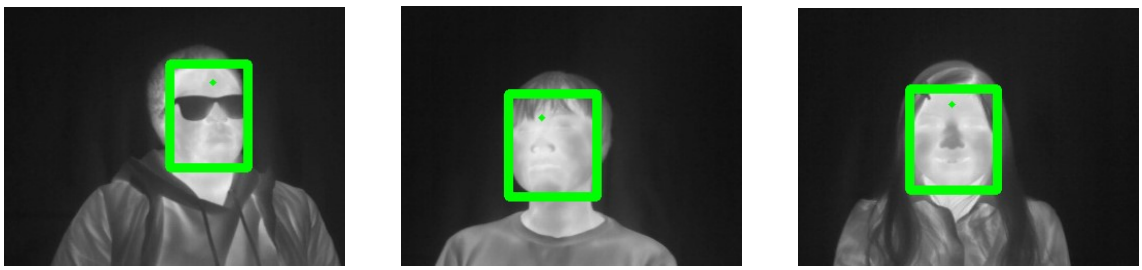
Figure 3.2: DrivFace dataset images with ground-truth annotations

3.1.3 Tufts

The Tufts Face dataset [44] is a large-scale face dataset, containing images captured from different sensors, including images captured from a thermal camera.

The dataset features 112 individuals (74 females and 38 males) from more than 15 countries, with ages ranging between 4 and 70 years old. Included in this dataset, there are 1557 thermal images, where the individuals were captured posing with five different facial expressions and looking at different fixed points.

As mentioned before, the goal of using thermal images in DMS is to capture and evaluate the temperature of the driver. Therefore, the annotations were updated with the addition of a ground-truth keypoint placed on the forehead, which allows for the acquisition of the required temperature. Tufts allowed greater variability in thermal images and was used for training and testing purposes. The annotations provided with the dataset only included a bounding box, so the annotation for the forehead keypoint was made manually using an appropriate labelling tool. Some images with annotations from this dataset are shown in Figure 3.3.



(a) (b) (c)

Figure 3.3: Tufts Thermal dataset images with ground-truth annotations

3.1.4 CES dataset

In addition to the available public datasets, a private dataset was also built as part of the project's goal to define a fit ground-truth system.

Approximately 5700 images were captured using three types of sensors and were annotated accordingly. To capture said images, a setup made of four distinct camera modules was assembled. The setup was composed of two thermal cameras of different resolutions, one infrared camera and one RGB camera, as illustrated in Figure 3.4.

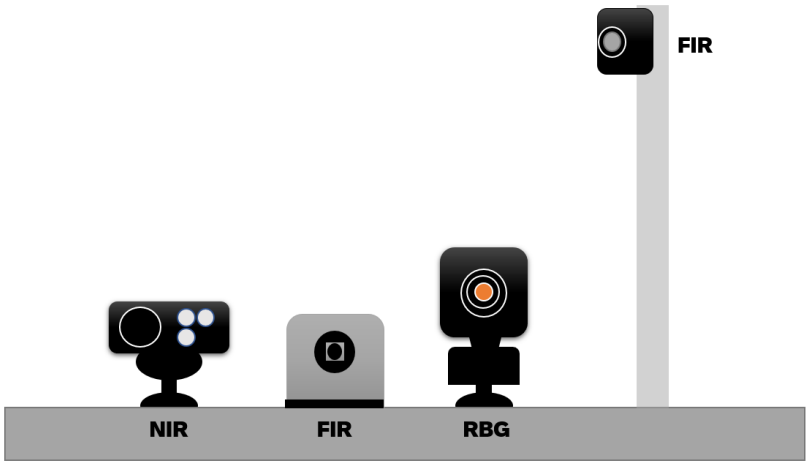


Figure 3.4: CES dataset camera setup illustration

In each session, the subjects were asked to stand in front of the camera setup and pose looking in five directions, imitating usual driving poses. Several capturing sessions were made with each subject, where some included the use of different accessories (such as a mask, sunglasses or a hat). Figure 3.5 displays a set of images from a single session and its five different poses.

The dataset is composed of fifty-two volunteers, 46 male and eight female, with ages ranging from 22 to 38 years old. This resulted in a dataset with a considerable number of images, although lacking in diversity when compared to other datasets. Nevertheless, the use of several accessories and extra components, as well as the positioning and angle of the cameras used while capturing, made this dataset the most suitable for the project.

In total, 1429 images were captured with each camera, resulting in 5,716 images for the full dataset (1,429 RGB, 1,429 NIR and 2,858 thermal, on account of the two thermal cameras used). Figure 3.6 illustrates some dataset statistics regarding these 1429 images, in terms of eye occlusion, the use of head accessories and facial masks.

In terms of eye occlusion, there is a similar distribution of images where the subjects used no eye occlusion accessory (43% of images) and images where the subject used sunglasses (40%). In the remaining 17%, subjects used seeing glasses or contact lenses. Regarding head accessories,

Figure 3.5: Example of capture session from CES dataset

in the majority of images (70%) no head accessory is used. 29% of the images show the subject wearing a head accessory that covered the forehead.

Concerning facial masks, 55% of the images featured subjects with no facial mask, while 38% featured subjects that wore a mask covering both the mouth and the nose. In the remaining 7%, the facial mask covered either the mouth or the nose.

Following the capture step, each image was manually annotated with a bounding box, five keypoints for RGB and NIR images and one keypoint for FIR images, following the same pattern as previous datasets. Figures 3.7 shows some annotated images from the dataset. Non-visible keypoints were also annotated, as they were required for training and testing of the models used further in the project.

3.1.5 Summary

The four facial image datasets obtained and utilised in this project are summarised in Table 3.1 (CES thermal dataset contains images with two different sizes as two thermal cameras with different resolutions were used in the capturing process). To take full advantage of the entirety of data in the datasets, transfer learning methods were employed, better described in Section 3.4.

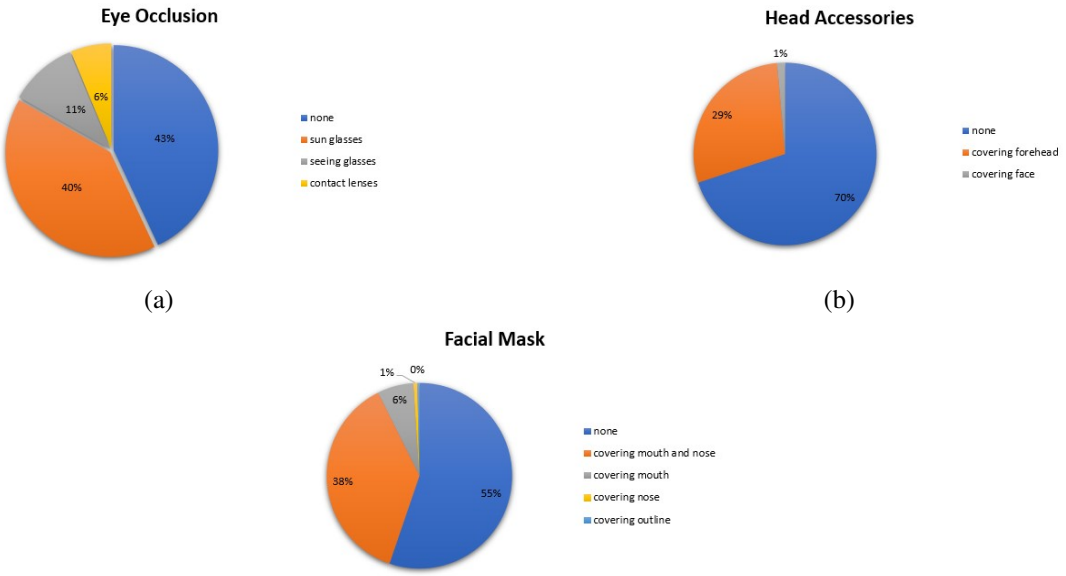


Figure 3.6: CES dataset statistics charts.

Figure 3.7: CES dataset images with ground-truth annotations.

3.2 Metrics

In this Section, the metrics used in this project to evaluate the performance and accuracy of the models, both in face detection tasks and in keypoint detection tasks are presented.

Table 3.1: Dataset Summary

Name	Image Type	Number of Images	Image Size
CelebA	RGB	202,599	Variable
DrivFace	RGB	606	640×480
Tufts	Thermal	1,557	336×256
CES RGB	RGB	1,429	2032×1534
CES Thermal	Thermal	2,858	120×160 206×156
CES NIR	NIR	1,429	1280×960

3.2.1 Object Detection Metrics

Commonly, the output of an object detector consists of a list of bounding boxes, the confidence/score levels and the predicted classes for each bounding box.

To assess the performance of these object detectors, different object detection challenges provide their evaluation source code, allowing participants to evaluate their algorithm before submitting their results [45].

In most of these competitions, the common metric adopted to assess the detections made by participants is the average precision (AP) and its variants [45]. In this project, variants of the AP were also used to evaluate the performance of the different detection models.

The AP metric is calculated by extracting information from the precision $\times$ recall curve. The precision P and recall R concepts are the concepts from which object detection methods mostly base their predictions, and are calculated based on the number of true positives (TP), false positives (FP) and false negatives (FN) obtained during a detection.

The precision is defined as:

$$P = \frac{\text{TP}}{\text{TP}+\text{FP}}, \tag{3.1}$$

and the recall is defined as:

$$R = \frac{\text{TP}}{\text{TP}+\text{FN}}, \tag{3.2}$$

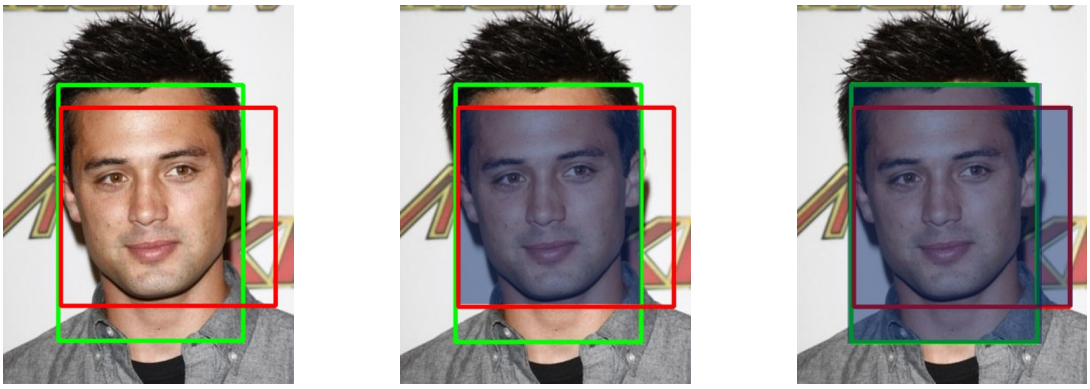
where the concepts of TP, FP and FN, associated with the detections, are defined as [45]:

- True positive: a correct detection of a ground-truth box;
- False positive: an incorrect detection of a nonexistent object;
- False negative: an undetected ground-truth bounding box.

To calculate the number of TP and FP detections, a correct detection and an incorrect detection need to be defined. The intersection over union (IoU) measurement is the most common way to define what a correct detection and an incorrect detection are. The IoU measures the overlapping (intersecting) area between the predicted bounding box (BOX_P) and the ground-truth bounding box (BOX_{GT}), divided by the area of union between them:

$$\text{IoU} = \frac{area(BOX_P \cap BOX_{GT})}{area(BOX_P \cup BOX_{GT})}. \tag{3.3}$$

Figure 3.8 illustrates an example of the calculation of the IoU between a ground-truth bounding box and a detection bounding box with an example image from the CelebA dataset.



(a) (b) (c)

Figure 3.8: IoU measurement: (a) shows an image with a ground-truth box (green) and a predicted bounding box (red). (b) shows the area of the intersection between the two boxes (blue). (c) shows the area of the union between the two boxes (blue). In this example the area of intersection between the two boxes has a value of 44,473 while the area of union between the two boxes has a value of 70,515, resulting in an IoU value of 0.673.

To define correct and incorrect detections, the IoU is compared with a given threshold t. If the IoU between the predicted box and the ground-truth box is greater than or equal to t, then the detection is considered a TP. On the other hand, if the IoU is smaller than t, then the detection is considered an FP [45]. For example, in Figure 3.8 the IoU value was equal to 0.673. For a threshold $t = 0.5$, this detection would be considered a TP. However, for a threshold $t = 0.75$ this detection would be considered an FP.

A goal of training an object detector is to reduce the number of FP and FN, achieving high precision ($FP = 0 \equiv P = 1$) and high recall ($FN = 0 \equiv R = 1$). Therefore, analysing the precision $\times$ recall curve, more specifically the area under the curve (AUC), inferences can be made regarding the precision and recall values, which allow the calculation of the AP. A high area under the curve indicates both high precision and high recall [45].

However, in practical cases, an accurate measurement of the AUC of the precision $\times$ recall curve can sometimes be challenging, as the precision $\times$ recall plot is often a "zigzag-like" curve [45]. To overcome these challenges, the common approach is interpolating the precision, applying the 11-point interpolation or the all-point interpolation methods.

In this project, to calculate the AUC and consequently the AP, the all-point interpolation method was used. In all-point interpolation, the shape of the precision $\times$ recall curve is summarised by interpolating the precision at each level [45]:

$$\text{AP} = \sum_{n}(R_{n+1} - R_n)P_{interp}(R_{n+1}), \tag{3.4}$$

$$P_{interp}(R_{n+1}) = \max_{\tilde{R}:\tilde{R}\geq R_{n+1}} P(\tilde{R}), \tag{3.5}$$

where $P_{interp}(\tilde{R})$ is the maximum precision whose recall value is is greater than or equal to R_{n+1} and n represents all points used for interpolation.

The AP is therefore obtained taking the maximum precision whose recall value is greater than or equal to R_{n+1} [45]. Figure 3.9 illustrates all-point interpolation for precision values, showing that the AUC is simpler to calculate with interpolated precision.

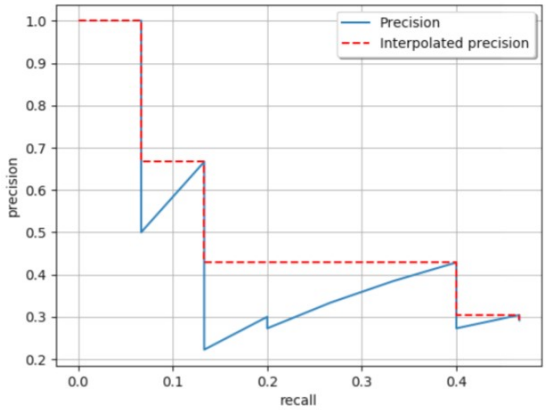


Figure 3.9: Precision $\times$ Recall curves with interpolated precision, applying all-point interpolation, from [45]. In this example, the AP is equal to the AUC of the interpolated precision line: $AP = 1*(0.066-0)+0.66*(0.133-0.066)+0.43*(0.4-0.133)+0.3*(0.466-0.4) = 0.245$

The AP returns a value between 0 and 1, and can be evaluated with different IoU thresholds. In this project, the most common values were used, as the performances of the models were evaluated by analysing the AP with IoU threshold equal to 0.5 (AP50) and equal to 0.75 (AP75).

In detecting models with more than one class, the mean AP (mAP) was calculated, defined as the average of the AP over all N classes [45]:

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N}\text{AP}_i \tag{3.6}$$

3.2.2 Keypoint Detection Metrics

The most common metric to evaluate the performance of keypoint detectors is calculating the MSE between the predicted keypoint coordinates and the ground-truth coordinates. However, this

metric only returns a pixel-wise error for the predictions of the model and not a normalised value.

The Common Objects in Context (COCO) [46] dataset makes available individual challenge tasks, with one task being related to keypoint detection that requires localisation of person keypoints in different conditions. To evaluate the accuracy of keypoint detection, COCO establishes an evaluation metric, the Object Keypoint Similarity (OKS) score, calculated from the distance between the predicted points and the ground-truth points normalised by the scale of the person.

The OKS metric is defined as:

$$OKS = \frac{1}{N}\sum_{i}^{N} exp(-\frac{d_i^2}{2s^2k_i^2}) \tag{3.7}$$

where N is the number of keypoints to detect, d_i the euclidean distance the ground-truth keypoint and the predicted keypoint and s the square root of the object area or the area of the ground-truth bounding box. k_i represents a keypoint constant that controls falloff and is defined depending on the keypoint.

Table 3.2 shows the values for k_i used in the calculating of the OKS metric in this project. These values were adapted from the values calculated by COCO researchers to fit the keypoints required.

Table 3.2: k_i constant for OKS score

Keypoint	k_i
Eyes	0.025
Nose	0.026
Mouth	0.03
Forehead	0.04

The OKS metric returns a normalised value between 0 and 1, depending on how close the predicted keypoint is to the ground-truth keypoint. Perfect predictions will return a value of $OKS = 1$ while poor predictions will return a value of $OKS \approx 0$. This metric was used in this project to evaluate the keypoint detection predictions of the models.

3.3 Models

In accordance with the proposed objectives, different object detection methods were implemented, capable of detecting the region of interest of the face and/or respective keypoints. In parallel with the detection task, the deep learning detectors were also able to correctly classify in which type of image the detection was being performed, with each image type being a class that the model identifies.

First, a HOG+SVM face detector was implemented and its performance was tested with the final test subset, to compare the results with the Deep Learning methods.

To compare the performance of two-stage detectors with one-stage detectors, both Faster R-CNN, described in Section 2.3.2.1), and RetinaNet, described in Section 2.3.2.2, were implemented and trained to detect faces from the different image types.

To achieve the second goal of facial keypoint detection, a variant of Mask R-CNN [47] was implemented. Mask R-CNN is a method used for object instance segmentation, which extends the Faster R-CNN method by adding a new branch, parallel to the existing branches used to obtain the bounding boxes and classifications, as illustrated in Figure 3.10. One minor change, when compared to the Faster R-CNN, is that instead of an RoI pooling layer, a new RoI*Align* layer is implemented. The RoI*Pool* layer performs quantizations on each RoI in order to extract small feature maps. These quantizations do not impact classification, but have a large impact on predicting pixel-accurate values, such as masks and keypoints [47]. The RoI*Align* layer removes the quantizations and extracts the small feature maps by dividing the original RoI into equally sized boxes and applying bilinear interpolation for each one. This removal of the quantizations greatly improves mask accuracy and is better detailed in [47].

This framework can be extended to detect singular keypoints [47]. By defining a keypoint as a one-hot binary mask, the Mask R-CNN can be adopted to predict K keypoints. This variant of the Mask R-CNN is named Keypoint R-CNN. For this project, the output of the network was modified to receive different keypoints.

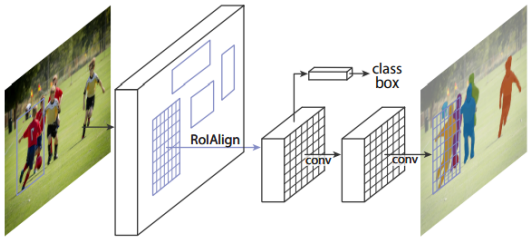


Figure 3.10: Mask R-CNN framework (from [47])

Table 3.3 summarises the models implemented in this project. The backbone architecture used for feature extraction in all the models was the ResNet [26] network. ResNet-18 and ResNet-50 were evaluated. Deeper backbones were not implemented, due to the fact that driver monitoring systems are required to work in real-time and backbones with substantial depths would prove too computationally costly and would increase the time taken to perform a detection.

Another approach for keypoint detection was also tested, modifying the RetinaNet to also detect keypoints. This approach consisted in the addition of a keypoint subnet, similar in structure to the classification and the regression subnets, that returned $3 \times N \times A$ keypoints, with N being the number of keypoints, each consisting of 3 values $(x_i, y_i, visibility)$, and A the number of anchors per spacial location, as illustrated in Figure 3.11. This approach was intuitively named KeyRetina.

Table 3.3: Model Summary

Task	Model	Backbone	Number of Parameters
Face Detection	Faster R-CNN	ResNet18 w/ FPN	28M
		ResNet50 w/ FPN	41M
	RetinaNet	ResNet18 w/ FPN	18M
		ResNet50 w/ FPN	32M
Keypoint Detection	Keypoint R-CNN	ResNet18 w/ FPN	46M
		ResNet50 w/ FPN	59M

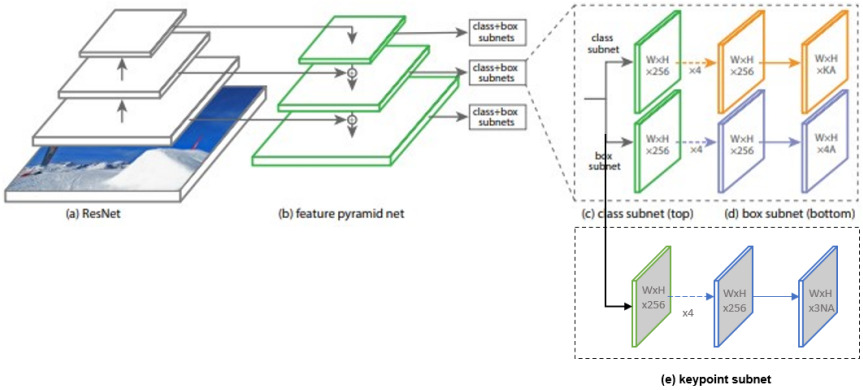


Figure 3.11: KeyRetina framework. The structure of the subnet (e) is similar to (c) and (d).

3.4 Transfer Learning

Transfer learning is a machine learning method that aims to improve the performance of a model by transferring knowledge obtained from previous tasks into a new similar task. In the deep learning scope, this approach can be implemented by first training a network on a large base dataset, and then take the learned features and parameters and retrain them with the target dataset [48]. This method is extremely useful when the target dataset is a small-scale dataset when compared to the base dataset, as it can be a powerful tool to train the network without overfitting [48].

Transfer learning was used in this project due to the availability of a large-scale dataset, to compare the performance of a pretrained network with a non-trained network and evaluate the impact transfer learning has on this project.

The main task of the project was divided into three related tasks: face detection using only RGB images; face detection using both RGB and FIR images; face detection using RGB, FIR and NIR images. Figure 3.12 illustrates how transfer learning techniques were used for these purposes.

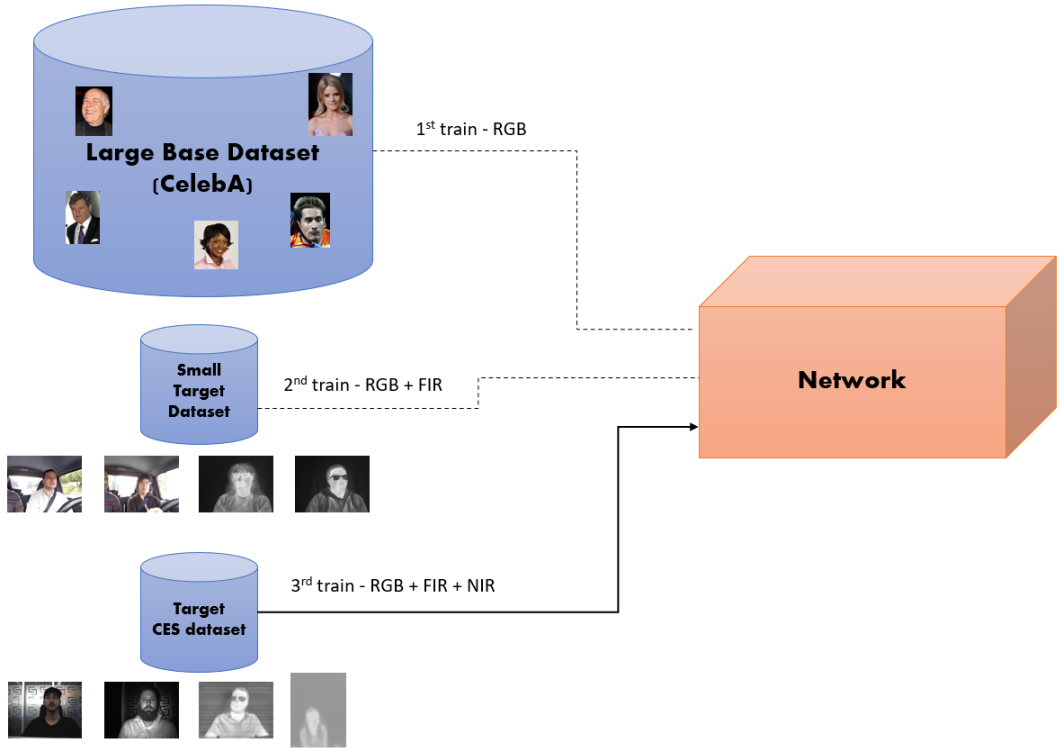


Figure 3.12: Transfer learning approach for model training. First, the networks are trained using the large-scale CelebA dataset; afterwards, the already trained networks are trained using a combination of DrivFace, CelebA and Tufts datasets; lastly, the networks are trained using the CES dataset with a combination of all 3 types of images.

3.5 Training and Testing

Before training and testing the networks, the datasets used for each scenario were first split into 3 subsets:

- Training set: the sample of data used to train the models. It is the data that is fed to the network during forward propagation, from which the cost value is extracted and the parameters are updated and optimised;
- Validation set: the sample of data used to validate the models during training. At the end of each epoch, the models are evaluated using this data. The use of a validation subset is very common in machine learning, as it helps prevent overfitting while allowing the tuning of hyperparameters;
- Test set: the sample of data used to perform an unbiased evaluation of the trained and validated models. The model's performance will be evaluated using this data subset, containing images that were not used in the training phase.

Figure 3.12 illustrates the steps for which the networks were trained applying transfer learning techniques.

The typical distribution applied when splitting the original dataset is 60%-20%-20% of the data for training, validation and test sets respectively. In Figure 3.12, the first train is implemented using the large-scale CelebA, where 200,000 images were used. Out of these images, 120,000 were used for training while 40,000 were used for validation. The remaining 40,000 were used in the final evaluation of the model.

Table 3.4 displays some information regarding training and testing scenarios, common to each network used.

Table 3.4: Train and Test Scenarios

	Scenario			
	1	2	3	4
Dataset	CelebA	CelebA DrivFace Tufts	CES	CES
Number of Training Images	120,000	1,870	3,375	3,375
Number of Validation Images	40,000	622	375	375
Number of Test Images	40,000	622	535	535
Number of epochs	10	15	15	50
Batch size	10	10	10	20

Object detectors usually output several overlapping proposals or candidate regions for detected objects, many of which are imprecise and not the ideal solution. To obtain a single bounding box for each detected object, a post-processing algorithm known as non-maximum suppression (NMS) is applied [49]. This algorithm takes all proposals given by the network (with a higher confidence score than a defined threshold t_{score}) and filters them by selecting the proposal with the best confidence score and suppressing the proposals that overlap it by comparing their IoU with a given threshold t_{nms}. The remaining windows are then organised, again, by their confidence score and the next top-scoring one is selected as being the correct detection [49].

In both validation and testing phases, NMS was implemented with a $t_{score} = 0.5$ and a $t_{nms} = 0.5$ before the calculation of the performance metrics. For validation, AP_{75} was the metric used to evaluate the performance of the model during training. At the end of each epoch, if the AP_{75} value was superior to the previous maximum value, the parameters of the model would be saved.

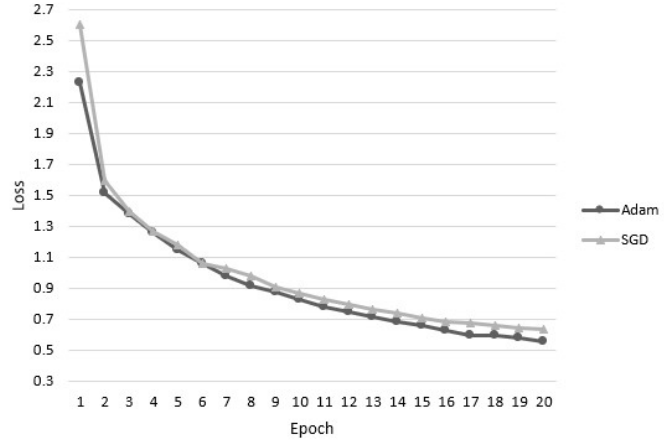
For testing, both AP_{50} and AP_{75} as well as Precision and Recall values were used to assess the models' accuracy. All tests were performed with a batch size of 1.

Following, the relevant training specifications of each CNN in this work are detailed:

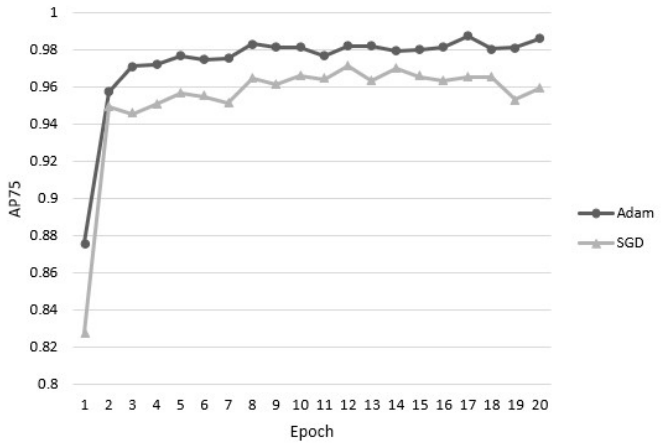
- Faster R-CNN was trained using Adam as an optimisation method, as detailed in Section 2.1.2.6. Initial tests showed that the default learning rate value of $lr = 0.001$ caused the model to diverge at a certain point during training. Therefore, a lower learning rate of $lr = 0.0001$ was defined, with the decay rates and numerical stabilisation variables being the default value (see Algorithm 2). Experiments using SGD as an optimisation method were

also conducted. However, results concluded that using Adam as an optimiser lead to faster convergence and better performance results, although by a small margin. This difference is better illustrated when analysing Keypoint R-CNN, shown in Figure 3.13. A cross-entropy loss is used to calculate the classification losses while a smooth-L1 loss is used to calculate the bounding box regression losses. The total loss is the sum of both classification and regression losses from the final predictor and the RPN. Before feeding the ground-truth data to the network, images are normalised and both images and boxes/keypoints are resized to fit between a minimum and maximum size with a range between 100 and 700 pixels. These transformations allowed faster convergence and training;

- RetinaNet was also trained using Adam as an optimisation method. The total loss was calculated as the sum of RetinaNet's Focal Loss for classification with smooth-L1 loss for bounding box regressions. As before, normalisation and resize transformations are also applied to the ground-truth data before being fed to the network;
- Keypoint R-CNN was trained in the same conditions as Faster R-CNN. The only difference was the addition of cross-entropy loss for the calculation of the loss associated with keypoint detection. To illustrate the difference in performance using different optimisers, initial trains were conducted, with the results show in Figure 3.13.



(a)



(b)

Figure 3.13: Adam vs. SGD performance differences. These results were achieved by training a Keypoint R-CNN model with 4500 CelebA training images and 500 validation images for 20 epochs. (a) shows the loss value through each epoch while (b) shows the AP_{75} for the validation phase in each epoch. Analysing the results, using Adam as an optimiser allowed for lower loss with greater accuracy.

Chapter 4

Results and Discussion

In this chapter, the results of this project are presented and discussed.

The training and testing algorithms were executed with access to a workstation with 64GB of RAM, an Intel(R) Xeon(R) W-2133 @3.60 CPU and an NVIDIA's Quadro RTX 6000 with 24GB VRAM GPU.

All algorithms used in this project were developed using Python, while the neural networks were developed using the PyTorch [50] deep learning framework, a framework commonly used in research for machine learning and deep learning tasks.

First, the results of the face detection models are presented. Both training and testing performances are detailed and analysed in Section 4.1.1. Next, the keypoint detection models' results are shown. As in the previous section, both training and testing performances are analysed. These results of the keypoint detection models are presented in Section 4.1.2. Finally, in Section 4.2 discussion of the obtained results is shown.

In all, the models were trained and tested for four scenarios, as detailed in Table 3.4:

- Scenario 1: trained and tested using the large-scale CelebA dataset, for 10 epochs;
- Scenario 2: using a mixed set containing images from CelebA, DrivFace and Tufts datasets, for 15 epochs, with pretrained networks from scenario 1;
- Scenario 3: using CES dataset containing images from all types, for 15 epochs, with pretrained networks from scenario 2;
- Scenario 4: using CES dataset containing images from all types, for 50 epochs, with networks that were not pretrained.

4.1 Results

4.1.1 Face Detection

4.1.1.1 Classical Computer Vision

The results from a testing phase of an HOG+SVM face detector with the test subset from scenario 3 were not positive. As this detector is only capable of face detection and is not capable of performing classification, it was tested using only one type of image at a time. The normalised confusion matrices shown in Figure 4.1 illustrate the results obtained. Evaluating the HOG+SVM detector, a false positive detection was defined as a detection with an IoU below 0.5 and a false negative detection occurred when no prediction was made (when a prediction is negative, but the actual is positive), meaning that the detector was not able to detect the face in the image.

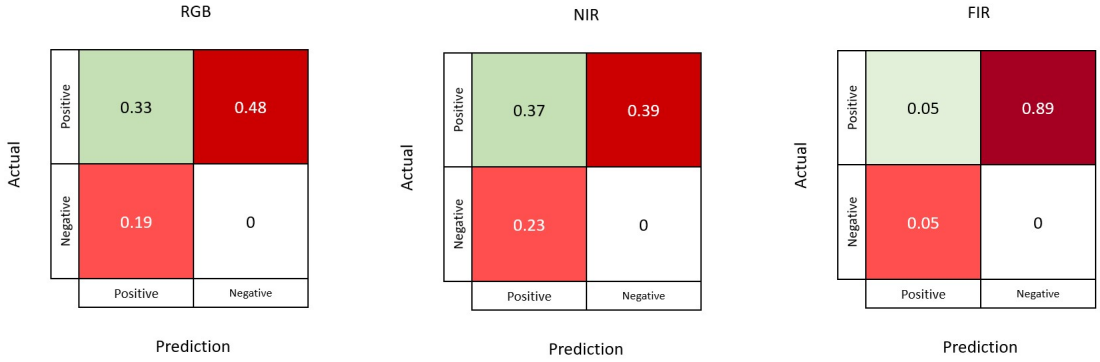


Figure 4.1: HOG+SVM test results with CES test subset for three image types.

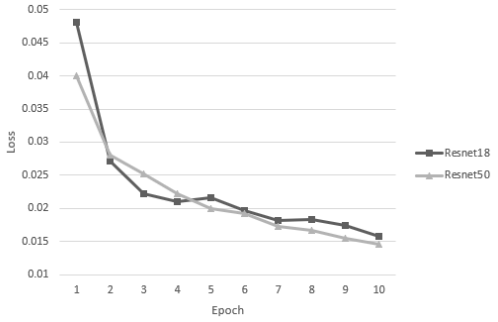
Figure 4.1 shows that the detector performed poorly for FIR images, failing to detect the face in nearly 90% of these images. Results with RGB images are more positive, but still far from ideal, only correctly detecting the face in 33% and failing to detect the face in 48% of these images. The same can be said for NIR images, with which the detector achieved better results, correctly detecting the face in 37% of these images.

4.1.1.2 Training Results

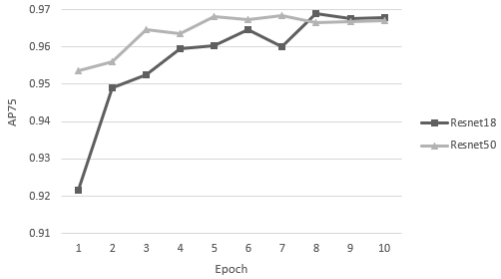
During training, information on the loss value as well as information on the accuracy of the validation phase was stored. Visualisation of the evolution of both loss and AP values allowed for an analysis of the progress of the training and verification if the training was advancing adequately.

As mentioned in the previous chapter, Faster R-CNN and RetinaNet were trained with ResNet as a backbone, implemented with depths of 18 and 50. Figure 4.2 illustrates training results for Faster R-CNN networks in all scenarios while Figure 4.3 shows training results for RetinaNet networks.

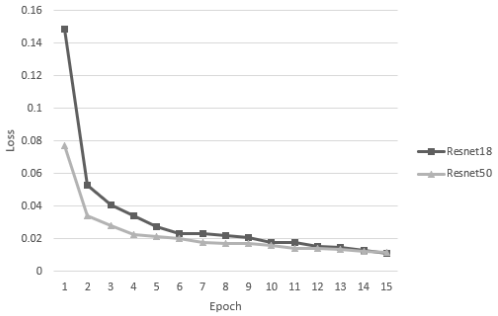
For Scenario 1, the location task is dominant in these detection models, since this problem contains only one class (RGB face). Figures 4.2a and 4.2b present the loss and the AP_{75} values per



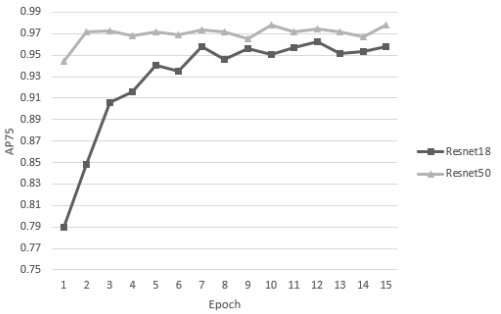
(a) Training Loss - Scenario 1



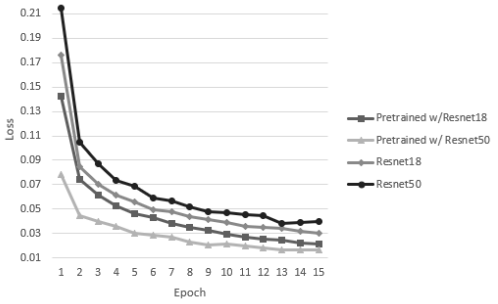
(b) Training AP_{75} - Scenario 1



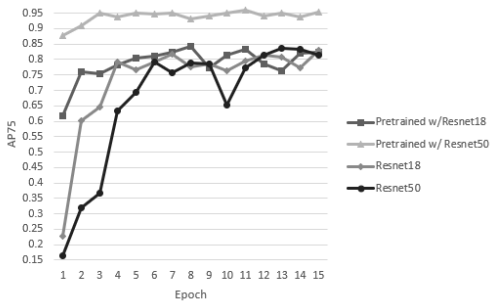
(c) Training Loss - Scenario 2



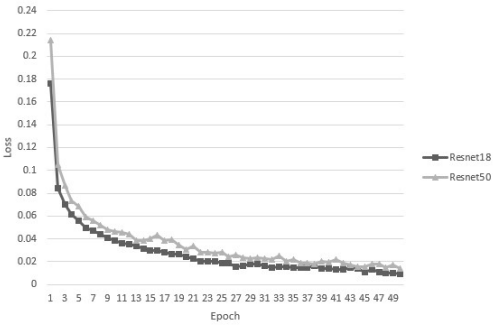
(d) Training AP_{75} - Scenario 2



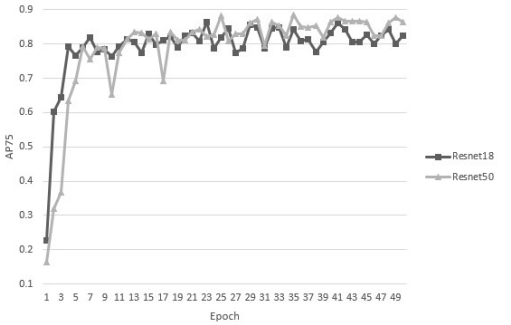
(e) Training Loss- Scenario 3+4



(f) Training AP_{75}- Scenario 3+4



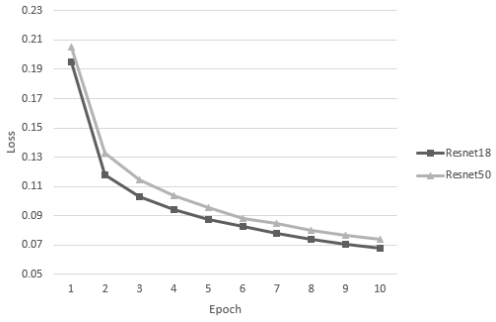
(g) Training Loss - Scenario 4



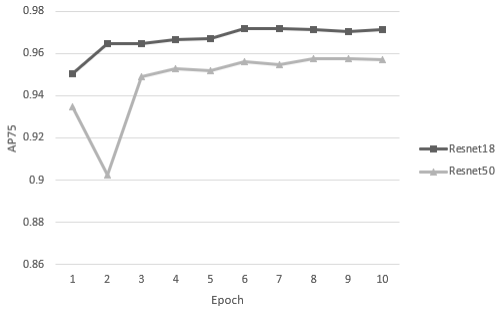
(h) Training AP_{75} - Scenario 4

Figure 4.2: Faster R-CNN training results by epoch

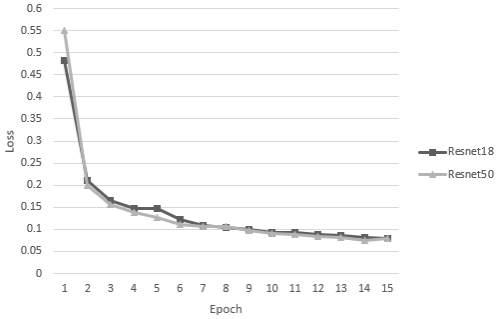
epoch during Faster R-CNN training for Scenario 1. Despite greater loss values, the model with a lower-depth backbone (ResNet18) achieved better validation accuracy results, although by a small margin (+0.0005). These first results suggest that both backbones, despite the depth difference,



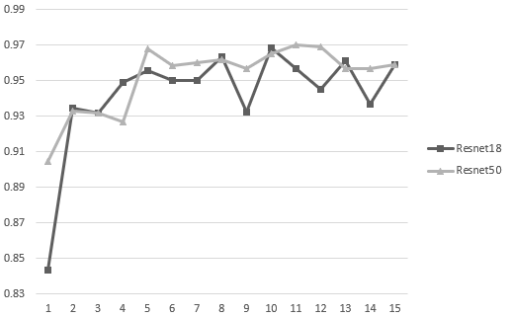
(a) Training Loss - Scenario 1



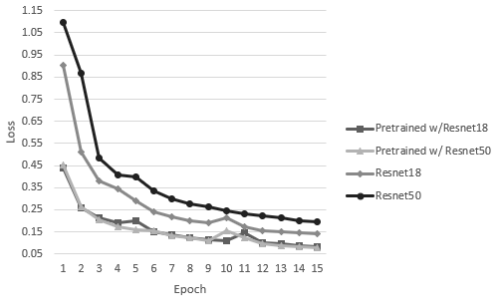
(b) Training AP_{75} - Scenario 1



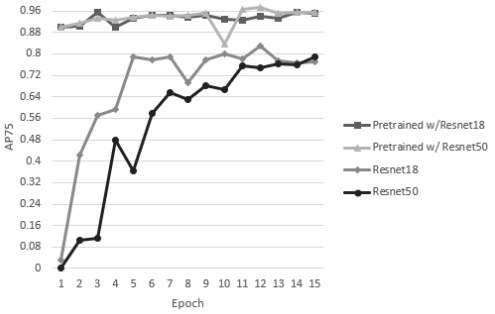
(c) Training Loss - Scenario 2



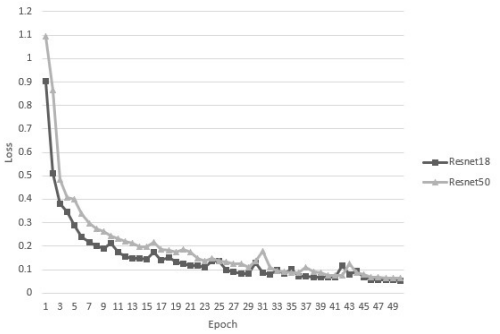
(d) Training AP_{75} - Scenario 2



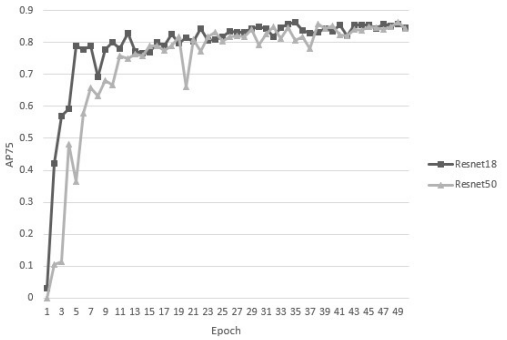
(e) Training Loss - Scenario 3+4



(f) Training AP_{75}- Scenario 3+4



(g) Training Loss - Scenario 4



(h) Training AP_{75} - Scenario 4

Figure 4.3: RetinaNet training results by epoch

can extract the relevant features required to detect faces from RGB images. Furthermore, Faster R-CNN with ResNet50 converges very rapidly while the model with a ResNet18 backbone, despite

starting with lower accuracy and higher loss values, also converges rapidly to achieve very good results.

RetinaNet with a lower-depth backbone achieved better training results for Scenario 1, as shown in Figures 4.3a and 4.3b. Not only did it obtain lower loss values but the RetinaNet with ResNet18 achieved the maximum validation AP_{75} out of all face detection networks for RGB images (0.972: +0.014 when compared with RetinaNet with ResNet50; +0.0029 when compared with Faster R-CNN with ResNet18).

Figure 4.2c and 4.2d show the training results of Faster R-CNN for Scenario 2, now with two classes: RGB and FIR. The trains were executed using the networks trained in the prior task, but with the addition of the new class. Faces in different types of images were defined as different classes, allowing the models to not only detect the face but also correctly classify the type of image in which the face was being detected. Faster R-CNN with ResNet50 showed better results at correctly classifying and detecting the different types of faces from the beginning of the training, while Faster R-CNN with ResNet18 took more epochs to learn how to correctly distinguish between the two types of images, starting with a greater loss value (+0.0714) and a much lower accuracy (-0.155) at the first epoch. However, both networks achieve good results, with the Faster R-CNN with ResNet50 achieving better validation accuracy (+0.0156).

RetinaNet with ResNet18 starts with a much lower AP_{75} when compared with the bigger-depth RetinaNet, as occurs with the Faster R-CNN networks. Despite this early difference, both networks also achieve good results, with a small gap between their validation performance (difference of 0.0017). These training results are illustrated in Figures 4.3c and 4.3d. However, for overall training validation results in Scenario 2, Faster R-CNN with ResNet50 achieved the best results.

Training with Scenarios 3 and 4 was performed using the same target dataset (CES dataset). Trains for Scenario 3 were initialised with pretrained networks from the previous scenarios, while training in Scenario 4 was initialised using models with randomly initialised weights.

Figures 4.2e and 4.2f show training results for the first 15 epochs, comparing the results using pretrained models and non-pretrained Faster R-CNN models.

The pretrained Faster R-CNN with ResNet50 as a feature extractor achieves the best validation results by a large margin when compared with Faster R-CNN with ResNet18 and the non-pretrained networks. As expected, applying transfer learning techniques allows for faster training, with better validation accuracy, as the amount of epochs needed to achieve better results is greatly reduced.

However, using ResNet18 as a feature extractor, the pretrained model achieves similar validation results when compared with both non-pretrained models, which may show that the features extracted from the lower-depth backbone may not be sufficient for good detection and classification using all image types.

Figures 4.3e and 4.3f show the training results obtained from the first 15 epochs from scenarios 3 and 4 using pretrained and non-pretrained RetinaNet networks. As with Faster R-CNN, the pretrained models achieved far better accuracy all while being trained for less epochs. RetinaNet

with ResNet-50 achieved the best validation performance, with an AP_{75} of 0.9747 (+0.0149 than Faster R-CNN with ResNet-50)

Unexpectedly, using a ResNet18 backbone with this architecture, the results are very positive and are way superior to the ones achieved with a Faster R-CNN architecture with the same backbone. These results may infer that, when using a RetinaNet architecture, features extracted from the ResNet-18 backbone can be used for correct detection and classification regardless of image type, contrary to Faster R-CNN.

For Scenario 4, the non-pretrained models were trained for 50 epochs. Figures 4.2g and 4.2h show Faster R-CNN training results for scenario 4 while Figures 4.3g and 4.3h show RetinaNet training results. Despite the constant reduction in the loss values, the accuracy values remained similar throughout the train, with small variations from epoch to epoch. These results can indicate that overfitting occurred, as the variation in the loss value did not correspond to a positive variation in the AP. For that reason, the train was not continued past further 50 epochs.

Table 4.1 summarises training results with all scenarios for all networks, showing the highest validation AP_{75}, the epoch in which that value was obtained, and the loss value at that epoch.

Table 4.1: Face Detection Training results by epochs

Scenario	Image Type	Network	Epoch	Loss	AP_{75}
#1	RGB	Faster R-CNN ResNet18	8	0.018	**0.969**
		Faster R-CNN ResNet50	7	0.017	0.968
		RetinaNet ResNet18	7	0.078	**0.972**
		RetinaNet ResNet50	8	0.079	0.958
#2	RGB + FIR	Faster R-CNN ResNet18	12	0.016	0.963
		Faster R-CNN ResNet50	15	0.012	**0.978**
		RetinaNet ResNet18	10	0.092	0.968
		RetinaNet ResNet50	11	0.089	**0.970**
#3	RGB + FIR + NIR	Faster R-CNN ResNet18	8	0.035	0.842
		Faster R-CNN ResNet50	11	0.020	**0.960**
		RetinaNet ResNet18	14	0.088	0.956
		RetinaNet ResNet50	12	0.097	**0.975**
#4	RGB + FIR + NIR	Faster R-CNN ResNet18	23	0.020	0.863
		Faster R-CNN ResNet50	35	0.021	**0.885**
		RetinaNet ResNet18	35	0.101	0.862
		RetinaNet ResNet50	49	0.063	**0.864**

The results presented in this section allowed for the inference of some performance results. However, the main focus of validation accuracy is to analyse if no overfitting occurs while training the models and if the model performs well with images not used in the train set, since validation results should not be considered for real evaluation of the models.

4.1.1.3 Testing Results

The final models were evaluated using the pre-defined test sets, calculating the precision, recall and AP values for IoU thresholds of 50% and 75%. These thresholds are used to reduce the impact of the annotation error, present in the ground-truth of all datasets used in this work. Apart from the human annotation error, there is no pattern in the annotations of the public datasets used. These different annotations sometimes lead to lower results with high IoU thresholds ($> 75\%$), which resulted in the decision to use the previously mentioned IoU threshold values for evaluation.

The average result from all test sessions for the face detection task is shown in Table 4.2.

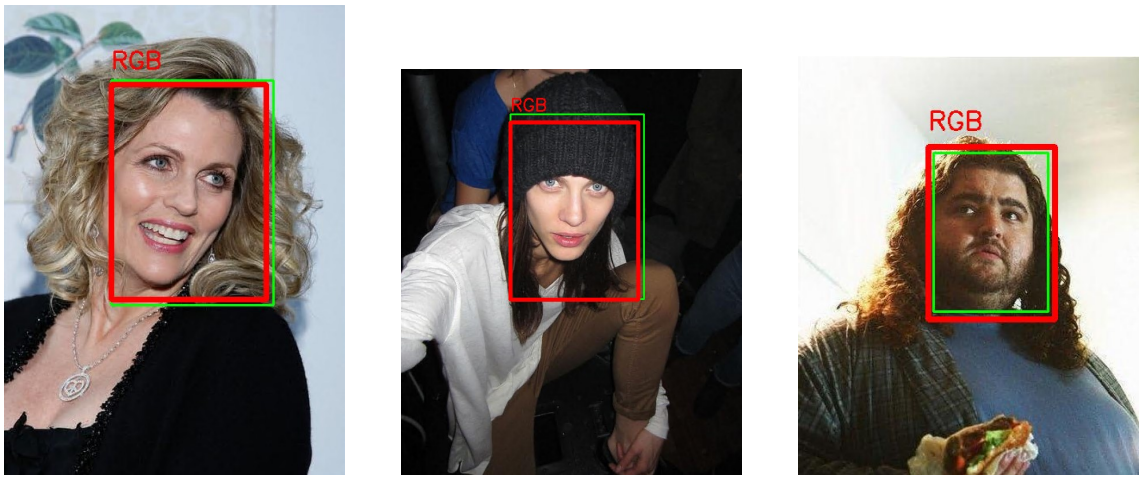
Table 4.2: Face Detection Testing results

Scenario	Network	P_{50}	P_{75}	R_{50}	R_{75}	AP_{50}	AP_{75}
#1	Faster R-CNN ResNet18	0.989	0.966	0.995	0.972	0.995	0.971
	Faster R-CNN ResNet50	0.994	0.971	**0.997**	0.972	**0.996**	0.971
	RetinaNet ResNet18	0.995	**0.976**	0.994	**0.975**	0.994	**0.974**
	RetinaNet ResNet50	**0.996**	0.964	0.993	0.961	0.993	0.958
#2	Faster R-CNN ResNet18	0.995	0.884	0.997	0.886	0.996	0.799
	Faster R-CNN ResNet50	**0.997**	0.902	**0.998**	0.904	**0.998**	0.888
	RetinaNet ResNet18	0.984	0.899	0.993	0.904	0.993	0.890
	RetinaNet ResNet50	0.993	**0.940**	0.993	**0.940**	0.993	**0.925**
#3	Faster R-CNN ResNet18	0.972	0.869	0.989	0.884	0.987	0.810
	Faster R-CNN ResNet50	**0.994**	**0.932**	0.996	0.934	0.996	0.898
	RetinaNet ResNet18	0.992	0.927	**1.0**	0.934	**0.999**	**0.916**
	RetinaNet ResNet50	0.981	0.9245	0.998	**0.940**	0.998	0.907
#4	Faster R-CNN ResNet18	0.946	0.826	0.989	0.862	0.988	0.816
	Faster R-CNN ResNet50	**0.989**	**0.891**	0.991	**0.892**	0.990	**0.856**
	RetinaNet ResNet18	0.911	0.805	0.991	0.874	0.987	0.837
	RetinaNet ResNet50	0.921	0.814	**0.994**	0.878	**0.992**	0.824

As expected, pretrained models reached higher AP values when compared to the non-pretrained models. An exception was the Faster R-CNN with ResNet18, where the non-pretrained network achieved higher AP_{75} than the pretrained network (+0.006). However, the results demonstrate that applying transfer learning before training improves the network's performance.

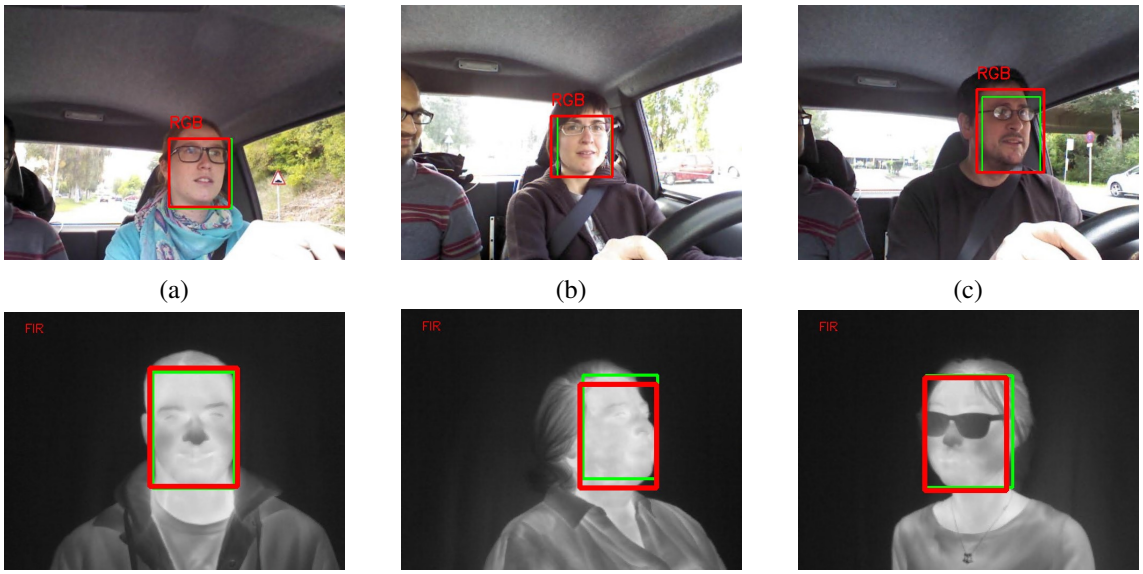
Results for Scenario 1 show that the networks perform extremely well for face detection in RGB images, with all networks achieving near-optimal results ($AP_{50} > 0.99; AP_{75} > 0.95$). Figure 4.4 illustrates some detections performed by the network with the best results for this scenario (RetinaNet with ResNet18), with the green bounding box being the ground-truth and the red bounding box the prediction.

Implementing face detection with both RGB and FIR images resulted in a slight reduction of the AP, as shown in Scenario 2 testing results from Table 4.2. Figure 4.5 illustrates some detections using the RetinaNet with ResNet50, which was the model with the best results for this scenario.



(a) (b) (c)

Figure 4.4: Face detections for Scenario 1 with CelebA dataset



(a) (b) (c)

(d) (e) (f)

Figure 4.5: Face detections for Scenario 2 with DrivFace and Tufts datasets

RetinaNet with ResNet18 obtained the best results out of the pretrained models for Scenario 3, achieving an AP_{75} of 0.9159, outscoring the best performing Faster R-CNN network by 0.0179 and RetinaNet with ResNet50 by a minimal margin of 0.0088, as shown in Table 4.2.

A confusion matrix can better illustrate the predictions made by each network for Scenario 3. Figure 4.6 shows normalised confusion matrices that illustrate each network's predictions. Each row of the matrix represents instances for true classes while each column represents instances for predicted classes. Predicted class detections with an IoU smaller than 0.75 were defined as false positives. These confusion matrices demonstrate that the networks achieved worse results when detecting faces in FIR images, and achieved better results when detecting faces in RGB

images. Furthermore, they show that the Faster R-CNN with ResNet18 achieves poorer results when compared with the other models, as also shown in Table 4.2.

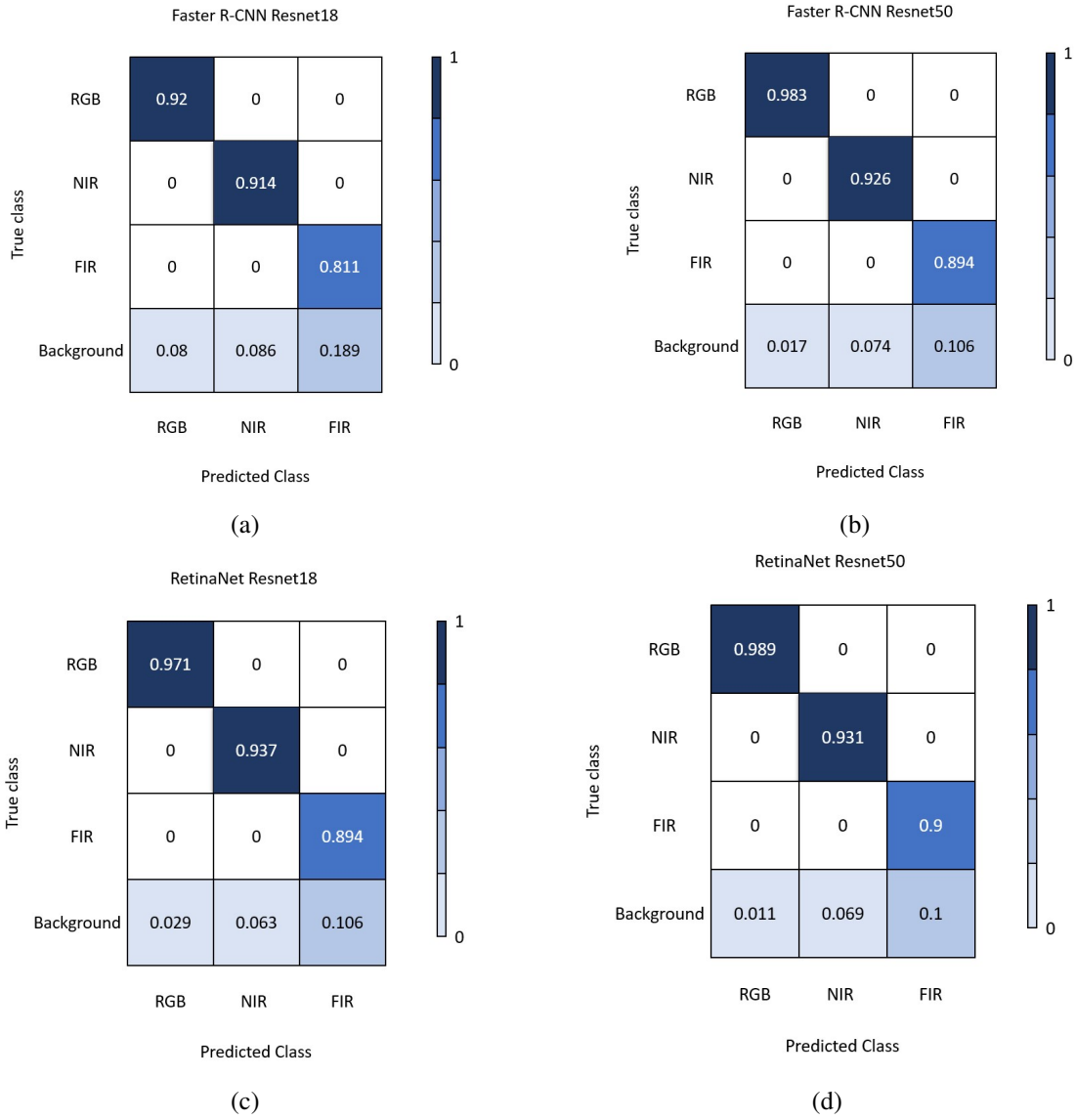


Figure 4.6: Confusion matrices illustrating testing results from Scenario 3

Figure 4.7 illustrates some detections performed by RetinaNet with ResNet18. These figures demonstrate the overall good results obtained in the face detection task, proving that a single process can extract the features from different types of images and accurately detect a face present in them. The green bounding box represents the ground-truth while the red bounding box refers to the model's prediction. In some cases, the detections could even be considered an improvement over the human annotations, demonstrating the overall good performance of the face detection networks.

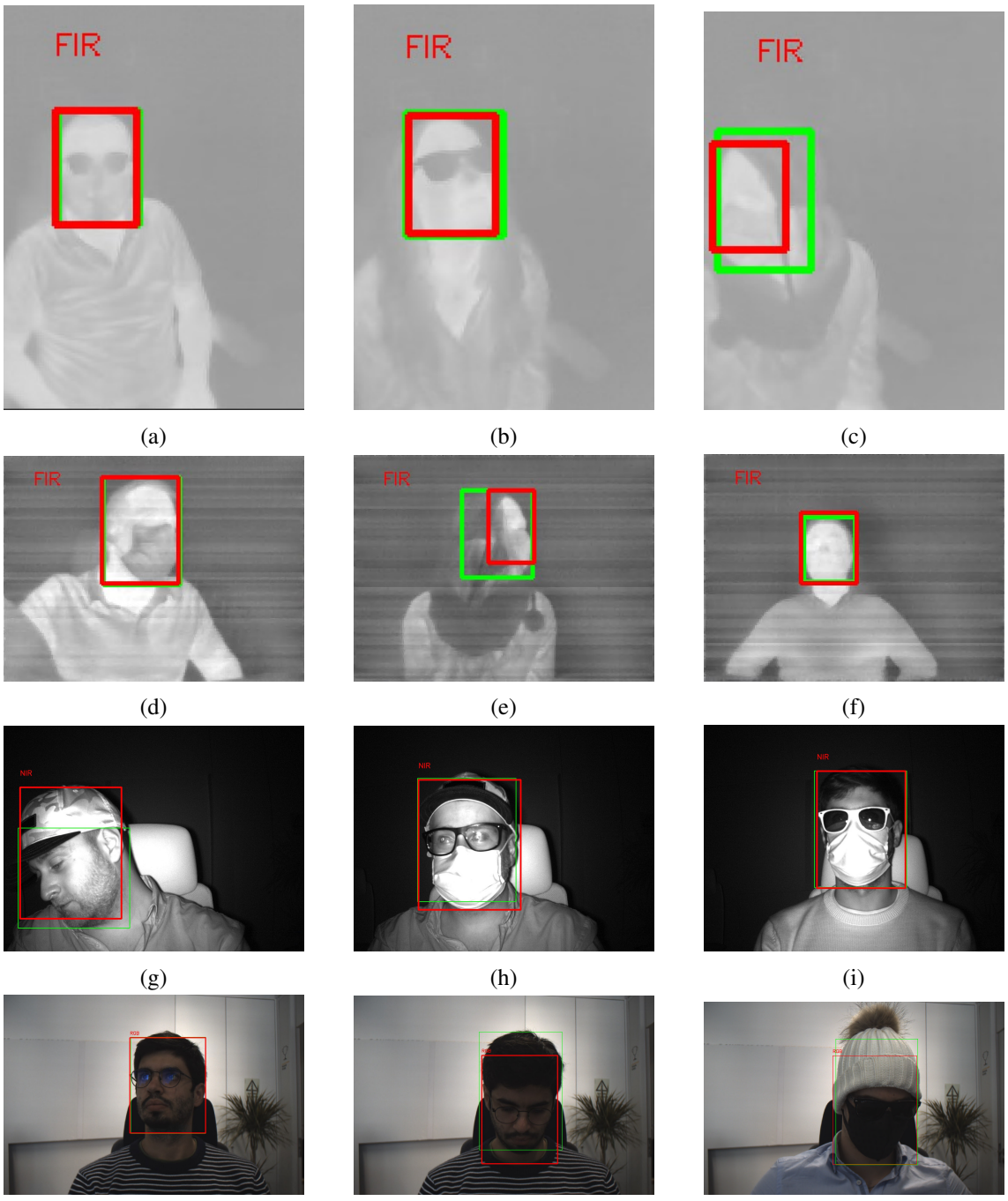


Figure 4.7: Face detections for Scenario 3 images with RetinaNet with ResNet18. The green bounding box represents the ground-truth box, the red bounding box represents the network's prediction.

4.1.2 Keypoint Detection

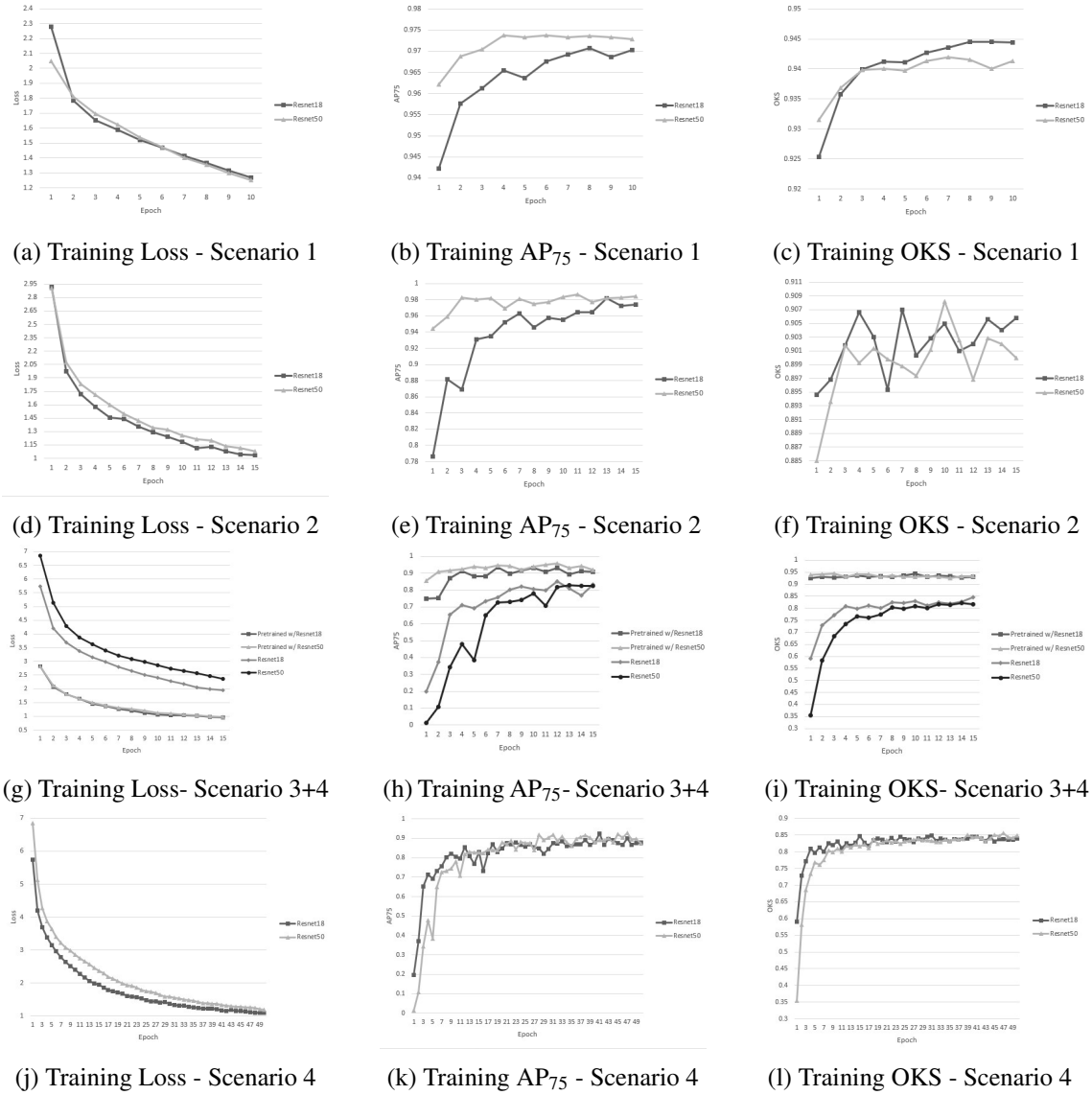
Similar to Section 4.1.1, this section will present training and testing results for keypoint detection using the Keypoint R-CNN network. This network performs not only keypoint detection but

also face detection. To evaluate the performance of the models, both AP and OKS metrics were calculated.

4.1.2.1 Training Results

During training, validation was applied to ensure the model was converging and that no overfitting would happen. In this case, to evaluate the performance in the keypoint detection task, the OKS metric was calculated during validation, along with the AP_{75}.

The training sessions are the same as described in the previous section and Table 3.4. Figure 4.8 shows the training results by epoch for all scenarios.



(a) Training Loss - Scenario 1 (b) Training AP_{75} - Scenario 1 (c) Training OKS - Scenario 1

(d) Training Loss - Scenario 2 (e) Training AP_{75} - Scenario 2 (f) Training OKS - Scenario 2

(g) Training Loss- Scenario 3+4 (h) Training AP_{75}- Scenario 3+4 (i) Training OKS- Scenario 3+4

(j) Training Loss - Scenario 4 (k) Training AP_{75} - Scenario 4 (l) Training OKS - Scenario 4

Figure 4.8: Keypoint R-CNN training results by epoch.

Validation results from Figures 4.8b and 4.8c show that Keypoint R-CNN with a ResNet50 backbone achieves slightly better accuracy for face detection when compared with the network

with a ResNet18 backbone, although the opposite occurs for keypoint detection. However, these performance differences are very small (+0.0029 AP_{75}; -0.0025 OKS), indicating similar performance for both networks.

Validation results for Scenario 2 with Keypoint R-CNN were very similar, in terms of face detection accuracy, to the Faster R-CNN results, as illustrated in Figures 4.8e and 4.2d. The pretrained model with the ResNet18 backbone needs more epochs to achieve AP values similar to its higher-depth counterpart. Regarding face detection and keypoint detection tasks, both models achieve similar validation accuracy values, with Keypoint R-CNN with ResNet50 achieving slightly better results (+0.005 AP_{75}; +0.0026 OKS).

Comparing training performances of pretrained models with non-pretrained models show that pretrained models achieve higher results, both for face detection and keypoint detection. However, the non-pretrained networks, trained for 50 epochs in scenario 4, achieve good results in terms of AP_{75} (> 0.92) and OKS (> 0.84). Keypoint R-CNN with ResNet50 reached better performance, with an AP_{75} of 0.956 and an OKS of 0.930.

Table 4.3 summarises training results of all scenarios and networks, showing the highest validation OKS, the highest validation AP_{75}, the epoch in which that value was obtained, and the loss value at that epoch.

Table 4.3: Keypoint Detection Training results by epochs

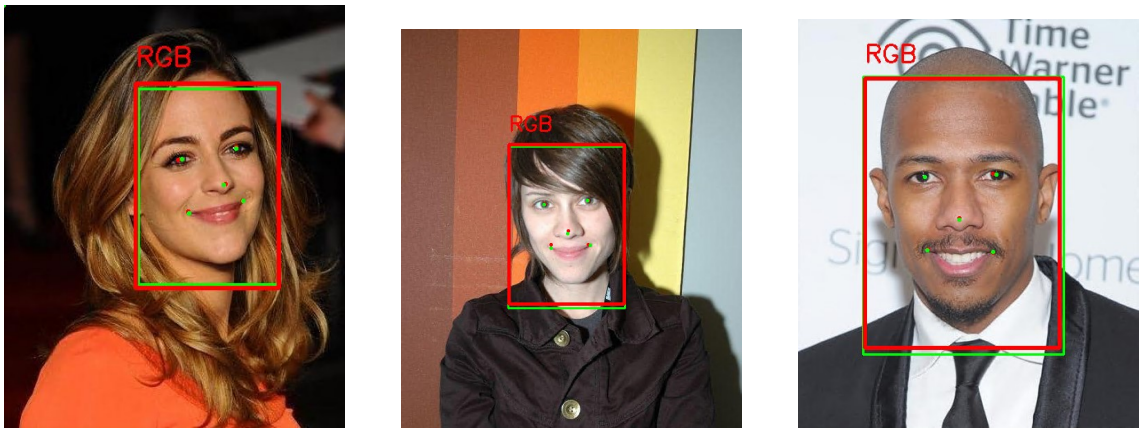
Scenario	Image Type	Network	Epoch	Loss	AP_{75}	OKS
#1	RGB	Keypoint R-CNN ResNet18	8	1.367	0.971	**0.944**
		Keypoint R-CNN ResNet50	4	1.624	**0.974**	0.942
#2	RGB + FIR	Keypoint R-CNN ResNet18	13	1.082	0.981	0.906
		Keypoint R-CNN ResNet50	11	1.216	**0.986**	**0.908**
#3	RGB + FIR + NIR	Keypoint R-CNN ResNet18	7	1.272	0.934	**0.93**
		Keypoint R-CNN ResNet50	12	1.052	**0.956**	0.930
#4	RGB + FIR + NIR	Keypoint R-CNN ResNet18	41	1.168	0.925	**0.844**
		Keypoint R-CNN ResNet50	49	1.279	**0.928**	0.840

4.1.2.2 Testing Results

Expectedly, the pretrained networks performed better than the non-pretrained ones, both for face detection and keypoint detection.

For Scenario 1, with only RGB images from the CelebA dataset, both lower-depth and higher-depth networks achieved similar results, with excellent AP and OKS results ($AP_{75} > 0.97$ OKS $>$ 0.94). Figure 4.9 illustrates some face and keypoint detections performed by the network with the best results for this scenario.

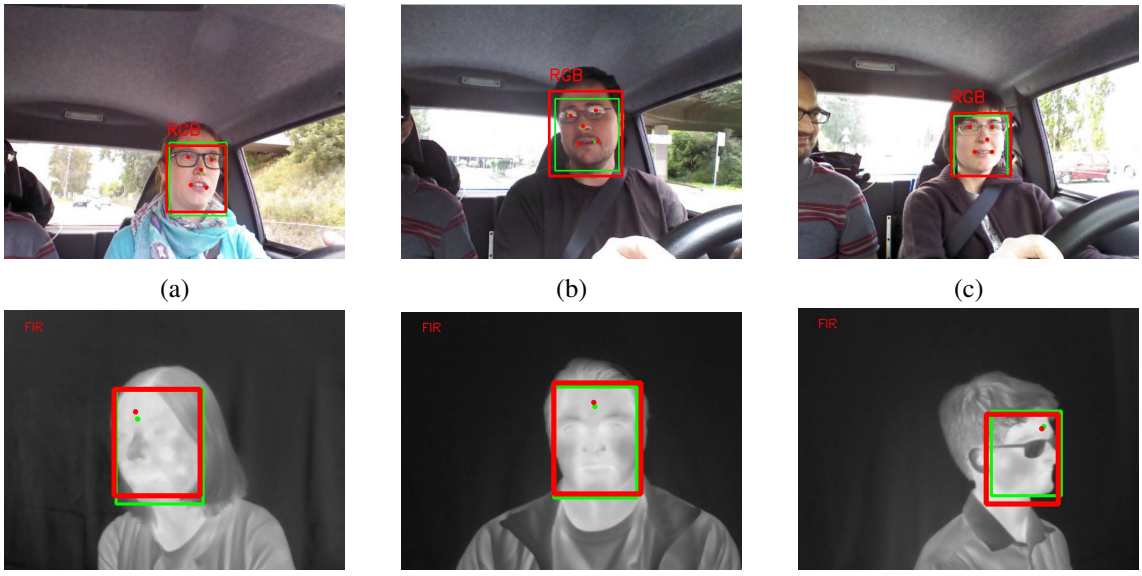
Regarding Scenario 2 with RGB and FIR images, a more challenging scenario than the first, test results regarding OKS were inferior when compared with Scenario 1 OKS results(≈ 0.88).



(a) (b) (c)

Figure 4.9: Face and keypoints detections for Scenario 1 with CelebA dataset

However, Keypoint R-CNN with ResNet18 reached higher AP (+0.0512 AP_{75}). Figure 4.10 illustrates some face and keypoint detections performed by the network with the best results for this scenario.



(a) (b) (c)

(d) (e) (f)

Figure 4.10: Face and keypoint detections for Scenario 2 with DrivFace and Tufts datasets

Moving on to Scenario 3 with RGB, NIR and FIR images, results show a large decrease in OKS values. This decrease can be explained from the fact that the CES dataset contains images with non-visible keypoints, discarded during training. However, results achieved by the best performing model are positive, with an AP_{75} of 0.897 and an OKS of 0.777.

Table 4.4 summarises testing results obtained from the keypoint detection networks while Figure 4.11 illustrates some detections performed by the pretrained Keypoint R-CNN with ResNet50 network.

Table 4.4: Keypoint Detection Testing results

Scenario	Network	P_{50}	P_{75}	R_{50}	R_{75}	AP_{50}	AP_{75}	OKS
#1	Keypoint R-CNN ResNet18	0.968	0.947	0.996	0.974	0.996	0.972	**0.945**
	Keypoint R-CNN ResNet50	**0.989**	**0.969**	**0.998**	**0.978**	**0.997**	**0.976**	0.941
#2	Keypoint R-CNN ResNet18	0.957	**0.894**	0.997	**0.931**	0.996	**0.918**	0.880
	Keypoint R-CNN ResNet50	**0.987**	0.877	**1.0**	0.887	**0.999**	0.866	**0.881**
#3	Keypoint R-CNN ResNet18	0.947	0.823	0.992	0.863	0.992	0.805	0.767
	Keypoint R-CNN ResNet50	**0.976**	**0.901**	**0.996**	**0.919**	**0.996**	**0.897**	**0.777**
#4	Keypoint R-CNN ResNet18	0.889	0.796	**0.988**	**0.884**	**0.987**	**0.858**	**0.689**
	Keypoint R-CNN ResNet50	**0.941**	**0.844**	0.985	0.883	0.984	0.857	0.681

Regarding the other tested architecture, KeyRetina, results showed inaccuracy in keypoint detection. In this approach, the keypoints are selected from the correspondent anchor with the best bounding box prediction, and not from a keypoint confidence score. During training, the remaining keypoints detected are ignored, and only those associated with the anchor of the bounding box are used in the loss calculation. This parallel dependency approach was not enough to optimise the regression across all network outputs, resulting in a small shift in the predicted keypoints.

Table 4.5 compares the keypoint detection performance difference between KeyRetina and Keypoint R-CNN. These results were obtained by calculating the OKS of the detection for 500 CelebA test images that contain fully visible keypoints. As observed, the KeyRetina achieved much poorer results in a task where Keypoint R-CNN was extremely accurate.

Table 4.5: KeyRetina performance results

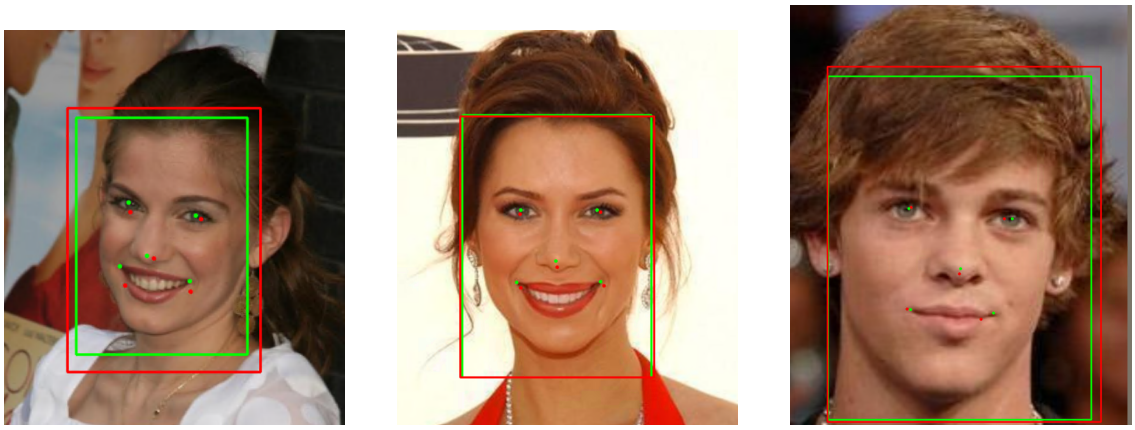
Model	OKS
KeyRetina ResNet50	0.787
Keypoint R-CNN ResNet50	**0.951**

Figure 4.12 shows some keypoint detection results. Although the results showed promise, the lack of pixel-wise precision made this network unfit for the goal of accurate keypoint detection. Despite the fact that the small error between the ground-truth (green) bounding box and the predicted (red) bounding box is acceptable (high IoU), the accuracy of keypoint predictions (red) is not the desired when compared to the ground-truth keypoints (red). Figure 4.12a better illustrates this problem, as all predicted keypoints deviate from their ground-truth counter-parts.

Figure 4.11: Face and keypoint detections for CES images with Keypoint R-CNN ResNet50.

4.2 Discussion

The results from Scenarios 3 and 4 show that applying transfer learning increases the AP of the detections by an average of 0.049 (for AP_{75}). Also, the pretrained models need to be trained for only 15 epochs while the non-pretrained models demanded 50 epochs for better convergence.



(a) (b) (c)

Figure 4.12: Examples of KeyRetina detections.

For Scenario 1, face detection in RGB images, RetinaNet with ResNet18 was the best performing network with an AP_{75} of 0.9742. Results from this first evaluation show that the depth of the backbone was not relevant for this task, as less deeper networks were able to perform similarly to deeper networks. These results can be justified by the fact that a ResNet backbone features skip-connections that overlook irrelevant layers.

For Scenario 2, as expected, performance values dropped with the addition of a new class (FIR face) and image type. In this case, deeper networks performed better, with RetinaNet with ResNet50 achieving the best AP_{75} (0.925). On the opposite end, Faster R-CNN with ResNet18 achieved the worst results, with an AP_{75} value of 0.799. All networks performed well with an IoU threshold of 50%, achieving AP_{50} values above 0.99. These values indicate that the models were able to correctly classify whether the detected face came from an RGB or an FIR image.

For Scenario 3, surprisingly, AP values increased for all networks except RetinaNet with ResNet50, when compared to the previous test where only two types of images were used. Analysing the results from this scenario, Faster R-CNN with ResNet50 achieved higher precision, but both RetinaNet models reached the higher AP values (with a maximum of 0.9159 for RetinaNet with ResNet18). These results are very encouraging, as the network with the lowest number of parameters, and consequently the lowest computational cost of the implemented networks, achieved the best performance. However, this was unexpected since two-stage detectors are usually more accurate because they first perform localisation, followed by classification and regression. One justification for these results can come from the simplicity of the problem, as there are few classes and only one face per image. That being said, these results determine that it is possible for a single deep neural network to accurately detect the face in images from different types and resolutions, thus achieving the main goal of this dissertation.

Regarding keypoint detection, the same approach was followed. First, the networks were trained with only RGB images in Scenario 1. Keypoint R-CNN with ResNet50 performed even better in face detection in RGB images than the best performing network evaluated before, achieving an AP_{75} value of 0.9765 (+0.0023 than RetinaNet with ResNet18) in the corresponding sce-

nario. This improvement may come from the fact that the introduction of more factors in the loss function (keypoint detection loss) results in improved detections. In terms of OKS, both less deeper and deeper networks achieved similar results, with a small advantage to the less deeper network (+0.0045 OKS).

Moving on to Scenario 2, as with face detection networks, AP values decreased as well as OKS values. The addition of the forehead keypoint and the fact that different images contained different keypoints could be seen as the reason for the decrease in keypoint detection performance. Contrary to the previous test, the less deeper network performed better in terms of AP while the deeper network achieved better OKS results.

The final networks for Scenarios 3 and 4 were implemented using the target CES dataset. This dataset contained images of faces in multiple poses and with a variety of accessories, which lead to the non-visibility of some keypoints. These non-visible keypoints were not used for the calculation of the keypoint detection loss during training. However, as the first datasets only contain visible keypoints, the addition of the non-visible keypoints explains the poorer keypoint detection performance, with lower OKS values. Thus, although the results obtained for keypoint detection with the target dataset are good, they are not optimal, with the best network (pretrained Keypoint R-CNN with ResNet50) achieving an OKS value of 0.777. As for face detection performance using this network, similar results were achieved when compared with Faster R-CNN, with an AP_{75} of 0.8972. As before, the pretrained models achieved better results both for face detection and for keypoint detection. The only exception was the non-pretrained Keypoint R-CNN with ResNet18 that showed higher AP_{75}. This result suggest that an overfitting occurred while training the less deeper non-pretrained network, since this network was trained using only one dataset and ground-truth.

Chapter 5

Conclusions

5.1 Final Conclusions

The main goal of this master dissertation was to determine if a single deep neural network can correctly detect the human face from the three different image types. Results shown in Section 4.1 demonstrated that a single CNN is, in fact, capable of processing data from different image sources, such as RGB, NIR and FIR, and detect a face from the data, with the implemented networks presenting accurate results (maximum AP_{75} of 0.9159 for the CES dataset). Not only were the networks able to detect the individual face in all image types, but were also able to correctly classify the type of image analysed. Furthermore, the network with the highest accuracy was surprisingly the one with the lowest number of layers and consequently the lowest number of parameters, which translates to a faster detection in real-time.

The development of a ground-truth dataset containing images from all types was another goal of this project and shown to be strongly relevant to train and evaluate the performance of the models. This goal was achieved with the CES dataset, a dataset composed of images of individual faces, with a total of 5716 images from RGB, FIR and NIR image sources.

A secondary goal was the implementation of keypoint detection in parallel with face detection. Results show that the addition of keypoint detection slightly increases face detection performance. With the introduction of different image types and images featuring realistic scenarios with keypoint occlusion, keypoint detection performance decreases by a large margin, which was somewhat expected. The best performing model achieved an AP_{75} of 0.8972 and an OKS of 0.777. Thus, despite poorer performance in keypoint detection, these results prove that it is possible that a CNN can detect not only the face but also key facial points from images from different sources.

In conclusion, the work developed in this dissertation is very promising and could result in an upgrade for DMS. The verification that a single process can be implemented to correctly extract information from different camera sensors could mean the development of new features and algorithms and the reduction in the computational cost of these systems. As for the proposed goals, it is possible to conclude that these were all achieved.

5.2 Future Work

This project allows for a variety of future works, in special regarding dataset improvement and keypoint detection.

First of all, although the CES dataset was considered to be a good dataset for the development of this project, improvements can still be made. One such improvement revolves around the need for intraobserver analysis, to compare the errors obtained from the networks with human errors. Another improvement can be made by acquiring more images from different subjects with distinct conditions, thus increasing the amount of data of the dataset and the variability, which could lead to better results for both face detection and keypoint detection tasks. One final upgrade to be made to the dataset is to capture new images inside a vehicle, as was initially planned. Due to COVID-19 restrictions, performing image capture inside a vehicle was not possible, as all captures were performed indoors in a scenario that did not represent a real driving scenario.

One other improvement is to train and test the networks with video frames captured from the camera sensors. Driver monitoring systems perform video camera from inside the vehicle, so the next step from this dissertation can be to verify if the network can accurately detect the face from a combination of video inputs from each camera sensor.

Finally, the most important improvement would be to implement some way of improving keypoint detection performance. Other architectures, such as KeyRetina presented in Section 3.3, could be upgraded to achieve accurate results. Some other approaches could be implemented and tested, and modifications regarding the visibility of the keypoints could be experimented with.

References

[1] Andrew Chesterton. How many cars are there in the world?, 2018. Accessed: 2020-10-20. URL: https://www.carsguide.com.au/car-advice/how-many-cars-are-there-in-the-world-70629.

[2] The Wandering RV. 50+ Car Accident Statistics (Aug. 2020) // Auto Accident Deaths, 2020. Accessed: 2020-10-20. URL: https://www.thewanderingrv.com/car-accident-statistics/.

[3] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[4] Md Zahangir Alom, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul A. S. Awwal, and Vijayan K. Asari. A state-of-the-art survey on deep learning theory and architectures. *Electronics*, 8(3), 2019.

[5] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep Learning vs. Traditional Computer Vision. *Advances in Intelligent Systems and Computing*, 943(Cv):128–144, 2020.

[6] Ismail Mebsout. Deep Learning's mathematics, 2020. Accessed: 2020-12-09. URL: https://towardsdatascience.com/deep-learnings-mathematics-f52b3c4d2576.

[7] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[8] Sindhu V. an Empirical Science Research on Bioinformatics in Machine Learning. *Journal of Mechanics of Continua and Mathematical Sciences*, spl7(1):86–94, 2020.

[9] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[10] Varun Basal. The deep history of deep learning. Accessed: 2020-12-28. URL: https://towardsdatascience.com/the-deep-history-of-deep-learning-3bebeb810fb2.

[11] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[12] Roei Bahumi. Cross entropy loss derivative. Accessed: 2021-01-30. URL: http://machinelearningmechanic.com/assets/pdfs/cross_entropy_loss_derivative.pdf.

[13] Yann LeCun, Patrick Haffner, Léon Bottou, and Yoshua Bengio. Object recognition with gradient-based learning. In David A. Forsyth, Joseph L. Mundy, Vito di Gesu, and Roberto Cipolla, editors, *Shape, Contour and Grouping in Computer Vision*, Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), pages 319–345. Springer Verlag, 1999. International Workshop on Shape, Contour and Grouping in Computer Vision ; Conference date: 26-05-1998 Through 29-05-1998.

[14] Ross B. Girshick. Fast r-cnn. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[15] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(2):318–327, 2020.

[16] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[17] Sebastian Ruder. An overview of gradient descent optimization algorithms. Accessed: 2021-01-31. URL: https://ruder.io/optimizing-gradient-descent/index.html#rmsprop.

[18] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 12 2014.

[19] Wikipedia. Convolutional neural network, 2020. Accessed: 2020-12-03. URL: http://en.wikipedia.org/w/index.php?title=Convolutional%20neural%20network&oldid=992073762.

[20] Kunihiko Fukushima. Neocognitron: A hierarchical neural network capable of visual pattern recognition. *Neural Networks*, 1(2):119–130, 1988.

[21] Yann Lecun, Le'on Bottou, Yoshua Bengio, and Parick Haffner. Gradient-Based Learning Applied to Document Recognition. *proc. OF THE IEEE*, 1998.

[22] Ismail Mebsout. Convolutional Neural Networks's mathematics, 2020. Accessed: 2020-12-09. URL: https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0.

[23] Abhishek Kumar Pandey. Convolution, Padding, Stride, and Pooling in CNN, 2020. Accessed: 2021-02-01. URL: https://medium.com/analytics-vidhya/convolution-padding-stride-and-pooling-in-cnn-13dc1f3ada26.

[24] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[26] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

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

[28] Matteo Bodini. A review of facial landmark extraction in 2d images and videos using deep learning. *Big Data and Cognitive Computing*, 3(1), 2019.

[29] 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*, volume 1, pages I–I, 2001.

[30] Cha Zhang and Zhengyou Zhang. A survey of recent advances in face detection. Technical Report MSR-TR-2010-66, June 2010.

[31] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, 2005.

[32] Tommaso Gritti, Caifeng Shan, Vincent Jeanne, and Ralph Braspenning. Local features based facial expression recognition with face registration errors. In *2008 8th IEEE International Conference on Automatic Face Gesture Recognition*, pages 1–8, 2008.

[33] Huachun Yang and Xu An Wang. Cascade classifier for face detection. *Journal of Algorithms & Computational Technology*, 10(3):187–197, 2016.

[34] J. Cruz, E. Shiguemori, and Lamartine Guimarães. A comparison of haar-like, lbp and hog approaches to concrete and asphalt runway detection in high resolution imagery. *Journal of Computational Interdisciplinary Sciences*, 6, 01 2016.

[35] Harihara Dadi and P.G. Mohan. Improved face recognition rate using hog features and svm classifier. *IOSR Journal of Electronics and Communication Engineering(IOSR-JECE)*, 11:34–44, 04 2016.

[36] T. Evgeniou and M. Pontil. Support vector machines: Theory and applications. In *Machine Learning and Its Applications*, 2001.

[37] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[38] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[39] Petru Soviany and Radu Tudor Ionescu. Optimizing the trade-off between single-stage and two-stage deep object detectors using image difficulty prediction. *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pages 209–214, 2018.

[40] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[41] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 936–944, 2017.

[42] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[43] Katerine Diaz-Chito, Aura Hernández-Sabaté, and Antonio M. López. A reduced feature set for driver head pose estimation. *Applied Soft Computing*, 45:98 – 107, 2016.

[44] Karen Panetta, Qianwen Wan, Sos Agaian, Srijith Rajeev, Shreyas Kamath, Rahul Rajendran, Shishir Paramathma Rao, Aleksandra Kaszowska, Holly A. Taylor, Arash Samani, and Xin Yuan. A comprehensive database for benchmarking imaging systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(3):509–520, 2020.

[45] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. A survey on performance metrics for object-detection algorithms. In *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 2020.

[46] Tsung-Yi Lin, M. Maire, Serge J. Belongie, James Hays, P. Perona, D. Ramanan, Piotr Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[47] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[48] Jason Yosinski, Jeff Clune, Y. Bengio, and Hod Lipson. How transferable are features in deep neural networks? *Advances in Neural Information Processing Systems (NIPS)*, 27, 11 2014.

[49] R. Rothe, M. Guillaumin, and L. Gool. Non-maximum suppression for object detection by passing messages between windows. In *ACCV*, 2014.

[50] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.