FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Pacient Validation through Facial Recognition

Gustavo Fernando Marques Duarte de Faria

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Jorge Alves da Silva

July 22, 2019

Pacient Validation through Facial Recognition

Gustavo Fernando Marques Duarte de Faria

Mestrado Integrado em Engenharia Informática e Computação

July 22, 2019

Abstract

Hospitals today label their patients the same way a warehouse marks its inventory: with a bar code. This dehumanization makes it easier to mix up the intended recipient of treatment or to mislabel a blood sample. Performing this kind of validation through facial recognition would alleviate these concerns because the medical professional in charge of it would be able to tell if a false positive had occurred. The same is not possible when looking at a bar code. That being said, this dissertation's final objective is to reduce human error when it comes to medical treatment/lab analysis performed in hospitals by implementing a functional prototype that demonstrates the feasibility of performing patient validation through facial recognition instead of the current method of reading a bar code from the patient's bracelet.

A pre-trained deep learning model was fine-tuned to tackle this specific problem in this particular context. In a hospital environment, there will inevitably be cases where someone's face is bandaged or disfigured, and that makes facial occlusion the most pressing difficulty to overcome.

With the concern of facial occlusion, some random, but restrict, data augmentation will be performed on the training dataset to simulate occlusion in several portions of the face, thus gearing the resulting model to be less sensitive to partial occlusions by being trained not to need the totality of a face to return a positive match.

The results obtained show a highest value of 92% accuracy for when identifying one hundred different people, which makes this to still be a very error prone approach to minimizing clinical errors. When measuring the effectiveness of training a face recognition model that focuses on facial occlusion, the results are positive, with an increase of 14% in accuracy when comparing a model trained without facial occlusion to a model trained with facial occlusion.

Resumo

Hospitais hoje em dia identificam os seus pacientes da mesma forma que um armazém controla o seu inventário: com um código de barras. Esta desumanização faz com que seja mais fácil que haja confusões com quem é suposto receber certo tratamento ou de quem é certa amostra de sangue. Fazer este tipo de validação através de reconhecimento facial iria aliviar estas preocupações porque o médico ou enfermeiro encarregado de a fazer seria capaz de reparar num falso positivo caso um ocorresse, o mesmo não é possível fazer quando se olha para um código de barras. Assim sendo, o objetivo final desta dissertação é reduzir o erro humano quando se trata de tratamento médico/análises laboratoriais realizadas em hospitais ao construir um protótipo funcional que demonstre a exequibilidade de realizar a validação de pacientes por reconhecimento facial em vez do método actual que consiste em ler um código de barras no pulso do paciente.

Um modelo pré-treinado de *deep learning* foi afinado para atacar este problema em específico neste contexto em específico. Num ambiente hospitalar, é inevitável a ocorrência de casos onde a cara do paciente está coberta por ligaduras ou desfigurada e isso faz com que a oclusão de partes da face seja a dificuldade mais urgente a ultrapassar.

Com oclusão facial em mente, o conjunto de dados usado para treino foi expandido com aleatoriedade restritiva, simulando a oclusão de diversas regiões da face, assim preparando o modelo final para ser capaz de retornar uma identificação correta, não necessitando que a totalidade da face esteja descoberta para o fazer, tornando o modelo menos sensível a oclusões faciais.

Os resultados obtidos mostram um valor máximo de exatidão de 92% quando se identificam 100 pessoas diferentes, o que faz com que este seja uma tentativa de redução de erros clínicos ainda bastante susceptível a falhas. Quando se mede a relevância de treinar um modelo de reconhecimento facial com foco na ocultação facial, os resultados são positivos, com um aumento de 14% em exactidão quando se compara um modelo treinado sem foco na ocultação facial a um modelo treinado sem foco na ocultação facial.

Acknowledgements

I would like to thank Glintt Healthcare Solutions S.A. for providing me with this opportunity of furthering my education while at the same time working on something that I feel might be of use to society, especially Pedro Rocha and José Melo. I would also like to thank my supervisor, professor Jorge Silva for pointing me in the right direction and giving me the tools that I needed in order to start this endeavor.

I will always be thankful for everything my parents have done for me throughout my whole life, but it feels more appropriate to thank them for giving me ready to eat meals, or even my sister for doing my grocery shopping, whenever I was overflowing with deadlines, which felt like always. Lastly, I'd like to thank my friends for motivating me when I felt like I was in way over my head and for distracting me when I needed a break.

Gustavo Faria

"If We Want Machines to Think, We Need to Teach Them to See"

Fei-Fei Li

Contents

List of Figures

List of Tables

Abbreviations

CNN	Convolutional Neural Network
CNTK	Microsoft Cognitive Toolkit
DAR	Detection-Alignment-Recognition
GAN	Generative Adversarial Network
LFW	Labeled Faces in the Wild
MIEIC	Mestrado Integrado em Engenharia Informática e Computação
NAG	Nesterov Accelerated Gradient
NTE	Normal Training Examples
NVE	Normal Validation Examples
ONNX	Open Neural Network Exchange
OpenCV	Open Computer Vision Library
OTE	Occluded Training Examples
OVE	Occluded Validation Examples
ReLU	Rectified Linear Unit
SGD	Stochastic Gradient Descent
SoC	System on Chip
SSD	Single Shot MultiBox Detection

Chapter 1

Introduction

Facial recognition is a popular form of biometrics. Granted, it is most often thought of as a way of performing authentication, but authentication implies identification, and validation in a hospital environment is just identification performed by a third party. As one of the least intrusive forms of biometrics, and requiring the least specialized hardware, it is the perfect contender for performing patient validation.

The main challenge is correctly identifying a subject when part of his/her face is occluded. That is a very likely occurrence inside a hospital because there will be cases where the patient has bandages, or eye-patches, or facial injuries and such occurrences will make it so parts of a face are not visible, making the task of facial recognition harder.

The latest boom in deep learning advances is linked to the recent advances in data gathering. Because of that, it is no surprise that the two biggest tech giants that, in one way or another, make money out of user data, are also the two leading forces when it comes to deep learning architectures geared towards facial recognition: namely Facebook [TYRW14] and Google [SKP15].

Both of the previously mentioned implementations are well suited for pose and illumination variance, and of these two, only illumination variance is decisively crucial in this context. A frontal, real-time image of the subject is not hard to come by when pointing a camera at the said subject with the purpose of identification, so pose variance is of little importance.

In order to overcome occlusion as a barrier for proper identification, different deep learning models were fine-tuned with data generated with the specific purpose of occluded face recognition in mind. A fraction of the training dataset was altered, introducing some blocked out sections to the face image, applied to random parts of the face, in a way that simulates the natural occlusion that a bandage would apply.

1.1 Context

In the field of Artificial Intelligence, this dissertation is mostly, in a practical sense, about Computer Vision. It tries to describe a solution to the problem of face recognition using Deep Learning and Neural Networks as its tool.

This work will be done in a bi-partisan collaboration between University of Porto's Faculty of Engineering and Glintt Healthcare Solutions S.A.

1.2 Motivation and Objectives

Unfortunately, nowadays, there are many errors in hospitals because of clinical observations or medical actions performed on the wrong patient. Most hospitals use barcode bracelets to carry out this validation. However, seeing as it is not always functional, and even impossible in many cases, medical professionals neglect this validation, which would ensure the procedure's correct execution. A paper bracelet can become worn out and unreadable, there could even be no ink in the printer and inside an emergency room, understaffed and overcrowded, maintaining this validation system operational is a non-priority.

Trying to bypass these inevitable complications, a patient validation system through facial recognition would be less of an inconvenience and therefore more easily carried out, resulting in fewer medical errors, which is this dissertation's final objective.

The minimization of medical errors, however, could only be measured after such a system is put in place, which is why it is referred to as the "final objective". This works to communicate the hopefulness of putting such a system in place. The practical objective, however, lays on proving the usefulness of implementing a facial recognition system in a hospital environment. In order to do so, a proposed system architecture is designed, and experiments are conducted to attest for its feasibility.

1.3 Document Structure

Besides the introductory chapter, this dissertation contains six other chapters. In chapter 2, some background into the field of Deep Learning is presented. In chapter 3, the state of the art is described and some relevant work is analyzed. In chapter 4, the research procedure is presented, ranging from how the data was prepared and transformed, to how the results are obtained and organized. In chapter 5, the architecture for the proposed system to implement is specified as well as the procedure implemented during the development of this dissertation for the purpose of research. In chapter 6, the experiments conducted are defined and their respective results are posted, organized and analyzed accordingly. Finally, in chapter 7, the overall outcome is contemplated on, and general conclusions are drawn.

Chapter 2

Deep Learning

Often used interchangeably, Deep Learning, and Neural Networks are very interconnected. Deep Learning, somewhat synonymous to Machine Learning in the sense that it is a subset of it, is the process of training a neural network with a significant amount of layers. The name is given as an analogy to the way the human brain processes information, naming its nodes as neurons, and the connections between them, or edges, as synapses. The similarities, however, stop there. The reason for the name is more of a public relations move than an accurate comparison to its biological counterpart, which is far more contrived than the simple abstraction designed by this particular field of Artificial Intelligence.

2.1 Neural Networks

Borrowing some similarities from the abstract representation of how a biological brain processes information, neural networks are made up of neurons and their respective connections. The neurons are organized in layers: one input layer, followed by all the middle, or hidden, layers and one output layer, as can be seen in figure 2.1, representing a 5-layer [1] fully-connected network.

A neuron holds a number comprised, most often, between 0 and 1, its activation value. Each neuron of a given layer is connected to every neuron of the subsequent layer, and each edge has a weight. The activation value of the neurons of the next layer is the result of the weighted sum of all the activations of the neurons of the previous layer with the weight of each respective edge, plus some other parameter called the bias. However, since the result is a number between 0 and 1, it still has to be normalized, for that, an activation function is used, such as the Sigmoid or ReLU functions. These will be explained further in section 2.2.1.

The matrix equation for calculating each neuron's activation, using the Sigmoid function, is the following:

[1] Naming conventions state that the input layer is not to be considered when describing the network in terms of the number of layers

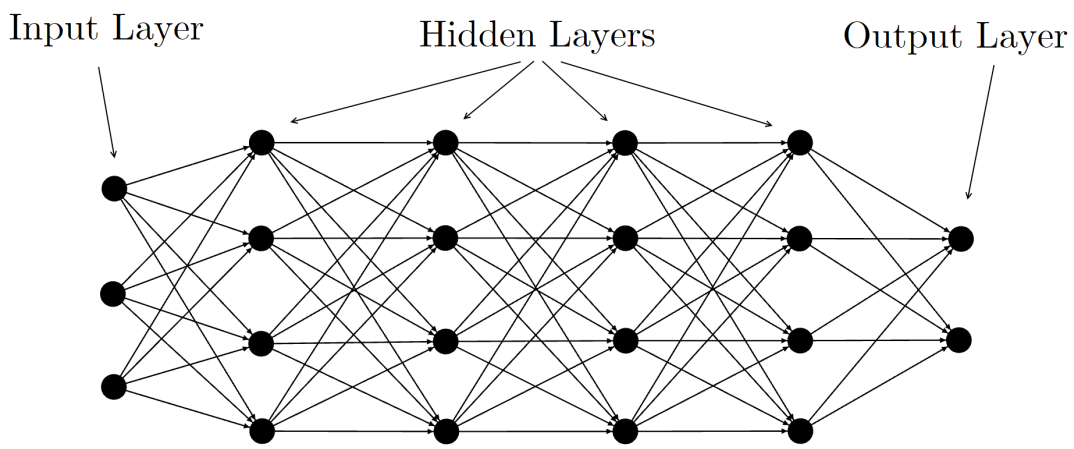

Figure 2.1: Fully-Connected, 5-Layer Neural Network

$$a^n = \sigma(Wa^{n-1} + b) \tag{2.1}$$

Where **a** represents the activation value of a neuron, **n** is the layer where said neuron is inserted, σ is the sigmoid function, pictured here only as an example, **W** is the matrix containing the weights of all the edges and **b** is the bias. Equation 2.1 is shown to reflect the importance that each layer has over the activation values of the next layer.
The differences between the desired output values and the actual result from feeding data forward are the measure of how wrong the output is. Training a neural network is, therefore, nothing but iteratively minimizing the output of the cost function. This way, thinking of the neural network itself as a function, this action can be simplified as finding the values for each variable that result in the lowest cost.

All the weights and biases are parameters of a neural network. That means they are the variables that have control over the output. Given the magnitude that a neural network can have in terms of the number of parameters, often reaching tens of millions, the process of tweaking these values is not manual. There is a special algorithm designed to adjust the values of all weights and biases so that they converge towards a minimum error, and that process is called backpropagation.

2.2 Hyperparameters and Optimization

Weights and biases, due to the enormous magnitude of their supply, are set through backpropagation, but there are ways in which the neural network architect can influence the overall result of the endeavor that is training a deep learning model. Those parameters that are set by the design of the developer are called hyperparameters. Some are set and reset by trial and error, some are

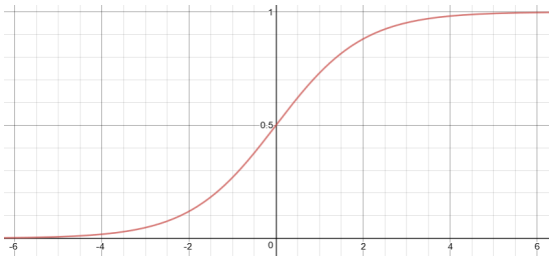

Figure 2.2: Sigmoid Function Plot

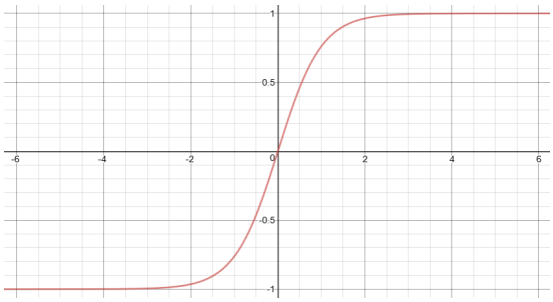

Figure 2.3: TanH Function Plot

chosen because of personal preference and intuition, and some fall out of use through research fueled innovation.

2.2.1 Activation Function

The dot product that is calculated to give the output of a neuron can range between very different values, so in order to normalize and to be able to tell how "active" a neuron is compared to all others in the same layer, that output value is mapped to a more restricting range. There are several examples of activation functions, and the following are the most commonly used.

- **Sigmoid**: Now in disuse [DSH13], the sigmoid function had been the go-to activation function because of its easy visual translation of inputs.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.2}$$

  Very negative numbers get ever closer to 0, and their positive counterpart edges closely to 1. But since those very negative or positive numbers have too low of a gradient, it is possible that during backpropagation parameters tied to that neuron see very slow evolution.

- **Tanh**: Very similar to the sigmoid function, but zero centered, its range goes from -1 to 1. The disadvantages remain the same.

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.3}$$

- **ReLU**: Recently found popularity [LBH15], the Rectified Linear Unit is computationally cheap given its non-complex mathematical form:

$$ReLU(x) = max(0, x) \tag{2.4}$$

Figure 2.4: ReLU Function Plot

ReLU has shown to significantly decrease the time needed to converge on a minimum when calculating Stochastic Gradient Descent when comparing to the previously mentioned activation functions. [KSH12].

2.2.2 Gradient Descent

Minimizing the loss of every forward pass is how a neural network is trained, or in a more colloquial sense, is how a neural network learns. It is an iterative action, and there are several ways of computing the next step towards finding the minimum of the cost function. These are called update functions, and all are variations of plain gradient descent. The most notable ones are listed below:

- **Stochastic Gradient Descent (SGD)**: SGD introduces some randomness to the calculation of the gradient vector by grouping the training data into mini-batches and by not using the calculated loss [2] of one single training example but averaging the loss of every sample inside one batch to then attain the vector that translates into a more direct "route" towards a local minimum of the cost function. SGD is almost guaranteed to make positive progress during training, but it is a very cautious and slow approach.

- **Momentum**: Takes a physics-inspired approach to SGD by adding a variable for velocity. That variable is increased in any direction where the gradient is of consistent value, similar to how a ball would roll faster as it progressed going down a hill.

[2] Loss and cost can be used interchangeably when describing the correctness of the classification results of a single forward pass through the network

Slight variations of this approach exist, namely Nesterov Accelerated Gradient or NAG. NAG overcomes some of plain Momentum's weaknesses by implementing a lookahead step which prevents it from overshooting as far as it would have otherwise [BBLP13].

2.2.3 Dropout Rate

The dropout rate is a probability. More specifically, the probability of randomly dropping out nodes, and their connected edges, from the neural network when training.

> "By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections" [SHK+14]

The point of doing so is to prevent the network from overfitting.

The optimal value, according to the original investigators, seems to be around 0.5, for the input layer, however, this probability strafes closer to 1.

2.2.4 Learning Rate

The learning rate variable as a number is a hyperparameter on itself. It dictates how big of a step the update function should take. If it takes big steps, it will often overshoot the minimum and, in a sense, zig-zag around it. If it takes small steps, it will generally evolve towards the minimum but at a slow pace. Finding the right learning rate is an essential part of debugging a neural network and is usually done by trial and error.

There are more dynamic approaches to finding the correct learning rate, and they come directly tangled with the update function. These are called per-parameter adaptive learning rate methods. A list of the most important ones is compiled below:

- **Adagrad**: As a per-parameter method, Adagrad lowers the learning rate for the most occurring features while increasing it for parameters with less regular features [DHS11].
- **RMSprop**: In an attempt to improve on Adagrad's performance, which linearly adapts the learning rate, RMSprop performs this update using a moving average of squared gradients [TH12].
- **Adam**: Similarly, Adam is an attempt to improve RMSprop. It can be quickly described as RMSprop with momentum.

2.2.5 Batch Size

2.3 Convolutional Neural Networks

Similarly to ordinary neural networks, Convolutional Neural Networks still transform input from the first layer into class scores in the final layer. The difference is that CNN architectures explicitly expect the input to be an image (three-dimensional, instead of a one-dimensional vector, or two-dimensional, if it is the case of a gray-scale image).

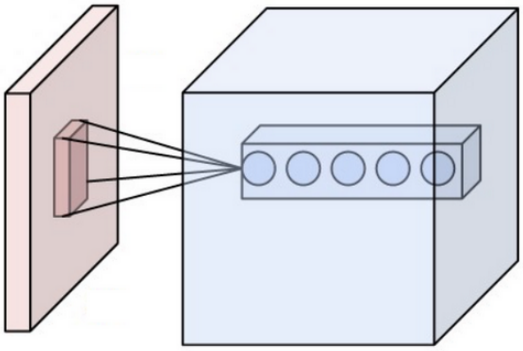

Figure 2.5: Visual Representation of a Convolutional Layer [Kar19]

2.3.1 Layers

Aside from fully-connected layers, the ones present in ordinary neural networks, as described in chapter 2.1, convolutional neural networks can be made up of several layer types, each with its own very distinct purpose:

- **Convolutional Layer**:

  The convolutional layer is the heart of the CNN architecture.

  Unlike fully-connected layers, convolutional neural networks are organized into three dimensions, and each layer has a set of small filters that will convolve across its two spatial dimensions (the same way our eyes would when reading text from a piece of paper) as if scanning a picture. These filters are usually small in size when compared to the layer they are working on, typically $5\times5\times3$ or somewhere around that. They are meant to be smaller than the layer they are convolving, so that local features can be extracted and so those features can be found in any part of the picture, effectively reusing an activation map for different parts of the input picture, something that is not possible with fully-connected layers. This local connectivity allows the total number of parameters in a neural network to be reduced, directly minimizing the risk of overfitting. The only dimension of the filters that has a fixed size is its depth, seeing as it is supposed to be the same as the depth of the layer it is convolving over. Each convolutional layer has a set of filters, and every one of them will return a separate two-dimensional activation map, which will then be stacked across the depth dimension and result in the output volume of that layer

- **Pooling Layer**:

  Often placed in between convolutional layers, pooling layers are used to progressively reduce the width and depth of the representation and hence reduce the number of parameters of the network, preventing overfitting.

> There are several pooling functions available such as max-pooling and average-pooling and depending on the size of the layer's filters, the subsequent representation's number of parameters can be reduced to 25% of the size of the previous layer.

2.3.2 Architecture

Typical CNN architecture is not too hard to understand. There are a lot of possible combinations, but the building blocks are somewhat constrained:

> "Most modern convolutional neural networks (CNNs) used for object recognition are built using the same principles: Alternating convolution and max-pooling layers, followed by a small number of fully connected layers." [SDBR14]

CNN architecture mostly differs on the depth of the neural network and frequency of pooling layers.

Chapter 3

State of the Art

In this chapter some of the most notable research papers in the field are studied, along with some less known but more similar to the scope of this dissertation.

3.1 Facial Recognition

The problem that is facial recognition can be subdivided into two: face verification and face identification. The first can be described as comparing two faces and simply deciding if they belong to the same subject or different subjects; The latter involves choosing, from a set of previously known subjects, to whom that face belongs.

3.1.1 Deep Learning and Face Recognition

Greater accessibility to training data has brought neural networks back to life, and the field of face recognition took advantage of this revival [PVZ+15, WZLQ16]. Before such occurrence, less data-hungry models were the norm, and as such, LFW [1] accuracy could not get past 95% [CCWS13].

After 2012, following AlexNet's [KSH12] overwhelming victory of the ImageNet competition, the benefit of adding more layers to a neural network became apparent.

The following subsections enumerate the most well-known algorithms of Deep Face Recognition.

3.1.2 Occlusion and Face Recognition

Over the years there have been approaches aimed at dealing with occlusion, some more and some less realistic as to what kind of augmentations were used to simulate this problem.

[1] LFW or Labeled Faces in the Wild is a face recognition dataset used by most researchers to compare different face recognition architectures fairly

The researchers responsible for [LLF07] took a simple, and almost static approach to the kind of augmentation used to simulate occlusion, as seen in figure 3.1, where they systematically cover one third or one quarter of the face picture.

Figure 3.1: Static Geometric Augmentation

On the other hand, the researchers behind [JM08] and [WLY+16] took and even more limited, but more realistic approach to their augmentation method. They superimpose the silhouette of a pair of sunglasses in the general position of the eyes, and/or a scarf over the lower half of the face picture, as shown in figure 3.2.

Figure 3.2: Static Realistic Augmentation

3.1.3 DeepFace

The first facial recognition neural network technology to achieve near human performance was DeepFace [TYRW14] in 2014.

Despite the innovative use of deep learning to achieve near human face verification accuracy, DeepFace also implemented a form of face processing, where 3d face alignment was computed to the input samples before being fed to the neural network.

DeepFace exploited Facebook's private face recognition dataset, one that contains 4.4 million faces of 4030 unique identities. They were able to achieve **97.35%** accuracy on the LFW benchmark using an ensemble of deep neural networks.

3.1.4 FaceNet

FaceNet took advantage of Google's private face dataset (100M-200M face images) and coupled their image classification architecture, the Inception network [SLJ+15], with a new loss function

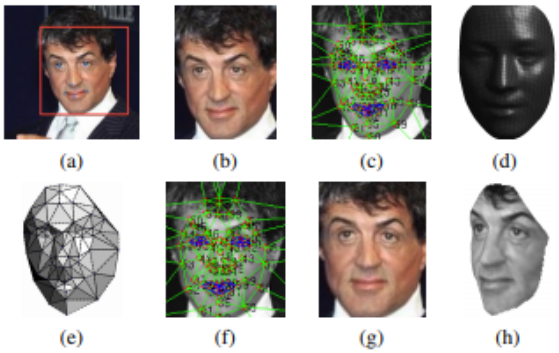

Figure 3.3: DeepFace 3D Face Alignment [TYRW14]

called triplet loss [SKP15] with which they claim to achieve faster convergence while minimizing intra-personal differences and maximizing inter-personal differences, represented in figure 3.4. The input images used were of 220×220 resolution and the network achieved an accuracy of **99.63%** on the LFW benchmark.

3.1.5 VGGface

Similarly to how FaceNet incorporates a more general image classification architecture into their face recognition model, so can the VGGnet [SZ14], be used for that same purpose. Can and has, and its pre-trained model is publicly available and is one of the most commonly used.

3.1.6 DeepID series

The DeepID series is composed of:

- **DeepID**

  Published in the same year as DeepFace, DeepID [SWT14] managed to overcome DeepFace's already impressive LFW benchmark accuracy, from 97.35% to **97.45%**. That might not look like much, but bearing in mind that its researchers lacked the resources made available by Facebook, it suddenly becomes a lot more impressive. DeepID was trained on the CelebFaces+ [LLWT15] dataset, compiled by the same researchers, but only posteriorly made public. While, on the one hand, DeepID was trained on 200,000 pictures, DeepFace was trained on 4.4 million.

- **DeepID2**

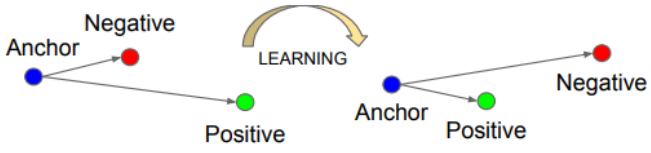

Figure 3.4: Triplet Loss Learning Visualized [SKP15]

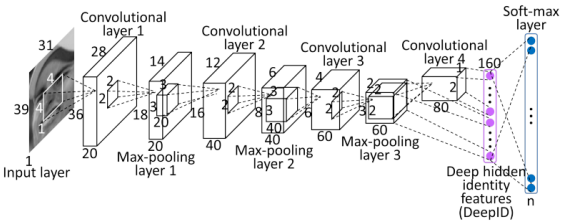

Figure 3.5: Original DeepID Architecture [SWT14]

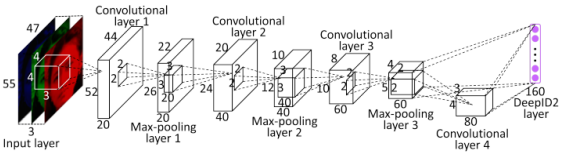

Figure 3.6: DeepID2 Architecture [SCWT14]

DeepID and DeepID2 [SCWT14] share most of their architecture, as is shown in figures 3.5 and 3.6. They are both composed of 4 convolutional layers, interposed by max-pooling layers while ending with a fully-connected layer to return the classification scores. The differences lie mostly on the input used: DeepID uses as input images of 39×31 pixels of a single gray-scale color channel, while DeepID2 performs its computations on images with a resolution of 55×47 pixels with three color channels, holding RGB information.

- **DeepID2+**

  DeepID2+ [SWT15] improved upon the previous state-of-the-art architecture, coincidently, DeepID2, by:

  > "increasing the dimension of hidden representations and adding supervision to initial convolutional layers."

  By doing so, DeepID2+ was able to increase the state-of-the-art of verification on the LFW dataset from 99.15% to **99.47%**.

- **DeepID3**

  In a bit of a miss-step, DeepID3 [SLWT] was initially regarded as having improved on DeepID2+'s accuracy performance on the LFW benchmark, from 99.47% to **99.53%**. Those new numbers were correct, however. But since then LFW detected a few mislabeled examples in its dataset, and those incorrectly labeled pictures had, therefore, been incorrectly assigned as an error by DeepID2+ at the time of publishing. Accounting for those errors, its accuracy then improved to **99.63%**, the same as DeepID3.

3.2 Training Data

As neural networks architectures increase in depth, so does the need for training data. Without a massive amount of training examples, and with a considerable number of parameters, it is difficult for a network to learn how to generalize. Without generalization, a deep learning model is poorly fit to identify any example that was not used in training correctly. This phenomenon is called over-fitting.

To that extent, it is of the utmost priority that there exists enough training data. While publicly available datasets are nowhere near the numbers that models such as DeepFace and FaceNet have at their reach, there are a few of sufficient scale.

Furthermore, increasing the size of the training dataset is not as simple as combining multiple datasets. The problem is them not being disjoint, or in other words, one dataset containing pictures of identities also present in another dataset. That is easily the case when it is taken into account the fact that most face recognition geared datasets are made up of images taken from the internet. Which is not a problem when training the model, the occasional repeated image has little effect in the final performance of the network. However, in order to obtain accurate, non-manipulated accuracy results when testing, it is necessary to assure that no identity used for training is present in the validation dataset. To test the model's ability to generalize, it has to identify images of previously unknown identities correctly.

3.2.1 Labeled Faces In The Wild

Acquired with the sole purpose of furthering the research in the field of face recognition in an unconstrained environment, Labeled Faces in the Wild [HMBLM08], or LFW, is the most common benchmark for measuring a model's accuracy in face verification.

It gives researchers access to:

- 13233 images
- of 5749 people

3.2.2 VGGFace2

A more recent dataset, compiled to provide a bigger training ground where every single identity had multiple examples, is VGGFace2.

> "The dataset contains 3.31 million images of 9131 subjects, with an average of 362.6 images for each subject." [CSX+18]

Every identity in this dataset has between 80 and 800 examples. Those examples were also treated to several stages of noise filtering.

3.2.3 MS-Celeb-1M

Compiled with the specific task of face identification (as opposed to face verification, such as LFW) MS-Celeb-1M [GZH+16] was also designed to be a benchmark.

> "recognize one million celebrities from their face images and identify them by linking to the unique entity keys in a knowledge base"

Alongside this challenge, train and test splices are also specified in this database of celebrity face pictures.

3.2.4 CelebA

Most commonly known by its shorter name, CelebA stands for CelebFaces Attributes Dataset [LLWT15].

Within it is a more modest number of face images: 202599, of 10177 different identities, averaging just under 20 face images per identity. Although mostly irrelevant for this dissertation's goal, it has a plethora of different annotations per image, such as whether that person is bald, or if that person is wearing eyeglasses or a hat, among many others. In total, there are 40 binary attributes and five landmark locations.

Given its size, and it not being excessive to handle on a typical laptop, it is on this dataset that the experimentations performed on this dissertation most heavily lean on.

3.3 Deep Learning Tools

Tensorflow

TensorFlow is Google's open-source software platform for deep learning [ABC+16]. According to the TensorFlow team, their framework facilitates:

> "Easy model building", "Robust ML production anywhere " and "Powerful experimentation for research" [Ten19a]

TensorFlow's biggest strength, however, lays in their large user base and online community [Dan19], an aspect that is appealing to beginners and experts alike.

TensorFlow also allows easy deploying of deep learning models to mobile [2] thanks to TensorFlow Lite, a tool that can convert any TensorFlow or Keras model into a compressed version of itself, optimized to run on mobile processors.

PyTorch

PyTorch [PGC+17] is Facebook's open-source deep learning library. It is more recent than TensorFlow and currently does not have a stable release, thus making its user base naturally smaller in comparison. Despite that, it is still the second largest deep learning framework [Dan19].

It offers higher abstraction and a gentler learning curve. However, said higher abstraction level comes with a catch, it means it allows for less control over the minutia of the development.

CNTK

CNTK [SA16] is the chosen initialism for Microsoft Cognitive Toolkit, thus revealing the entity responsible for its maintenance.

The size of its community is comparable to PyTorch's [Dan19], and similarly to PyTorch, it was a pioneer in adopting the Open Neural Network Exchange, or ONNX, format. By doing this,

[2] Android and iOS

they are facilitating greater research collaboration, by allowing its trained models to be imported, fine-tuned, and deployed by other deep learning frameworks, and vice-versa.

Caffe

Berkeley AI Research department [UC 19] developed Caffe [JSD+14]. Caffe is quoted as being the most well-optimized approach into Convolutional Neural Networks and image processing. It is, however, not nearly as well documented as TensorFlow, which is one of the reasons for its smaller user base.

Theano

Theano [The16] [3] is not a Deep Learning specific library. It is

> "a Python library and optimizing compiler for manipulating and evaluating expressions, especially matrix-valued ones" [The19]

which is to say it is a low level, speed optimized, version of Numpy. Still worthy of mention because it is one of three back-ends available to use with Keras.

Keras

Keras [C+15] is a research focused deep learning API, also written in Python. Its focus is on

> "easy and fast prototyping (through user friendliness, modularity, and extensibility)".

It is a high-level wrapper of other popular deep learning libraries, namely TensorFlow, CNTK and Theano.

[3] As of 28 September 2017 MILA has announced their plans to stop development for Theano

Chapter 4

Methodology

In this chapter the methodology used to transform raw image data, grouped together to simulate a patient dataset, into a face recognition model, capable of calculating the most probable identity of the person represented in each picture is described.

4.1 Pre-Processing

4.1.1 Detection and Recognition Pipeline

Face Recognition is a tiered task that entails some pre-processing of data beforehand in order to achieve the most accurate results possible. As such, this pipeline of data fitting is commonly referred to as Detection-Alignment-Recognition or DAR [HMBLM08].

As the name suggests, in order to allow the face recognition model to focus on that specific task, because most datasets are made up of pictures of faces with unconstrained environments, and so are most real-life cases, the first step is to detect the face, or faces, in the picture. Once the facial landmarks are identified, the picture can then be cropped, so only the relevant information is left. One algorithm capable of doing this is the Viola-Jones face detection [VJ04].

Besides simple cropping of a photo, it might also be necessary to align the face with what would be the camera's point of view, and this way, minimizing the effect that pose-variance has on the final result. This step, however, is optional. A researcher might need to faithfully recreate real-world conditions, to which end, face alignment might hinder the realization of that objective. Not only that, in this particular use-case, alignment constitutes an additional operation to perform when running in a mobile environment, and since the detection step is already expensive enough, computationally speaking, adding an alignment phase would further deviate this whole process from being able to run in real-time.

The final step on this pipeline is indeed the task at hand, and having followed those previous two steps, there should be a significant reduction in the final result's error percentage.

Detection

Two different methods of face detection were used during the development of this dissertation, one for each of the environments in focus, namely: research and production, or mobile, environments.

The research environment, as it is built in python, took advantage of OpenCV's [Bra00] pre-built face detection framework. Modifications were made on top of their usage example in order to bypass the alignment phase, and also so different margin values could be used to experiment with. Different pre-trained face recognition deep learning models operate better with different values for the proportion of face relative to the background, i.e., the size that the face occupies can be changed and is also a value to optimize.

The production environment has the constraint of needing to run in a mobile setting. That required the detection model used to be lightweight and, because of the chosen deep learning framework, convertible into tflite. With that in mind, the chosen approach was the SSD, or Single Shot MultiBox Detection [LAE+16] algorithm running on top of a mobilenet [HZC+17] architecture, as taken from [Hu,19].

Recognition

The recognition step can also be split into two. Most often, those two steps are stacked and operate as one. They are:

- Representing a face picture as a feature vector;
- Assigning a probability score to a feature vector to each of the classes in question, i.e. each person to identify.

The first step is where the pre-trained neural networks become indispensable: By removing the top, or classification layer, it is possible to extract an n-dimensional[1] vector that is, in a sense, a compressed version of a face picture that can be more efficiently compared to other faces, in order to assign it to one of the known classes.

The second step is where the actual classification occurs. It can be done by implementing a softmax classifier or by manually performing the arithmetic calculations a softmax classifier would intuitively learn. This second step, however, will only be described in the next chapter and referred to as classification.

As previously mentioned, pre-trained neural-networks are indispensable in order to obtain good results. It is severely computationally heavy to backpropagate through a neural-network with tens of millions of parameters and would be unfeasible to do without proper hardware. Having access to such a resource is a huge time saver and makes this type of research available to many more people. By resource what is meant is a pre-trained neural network.

Two neural network architectures were used to perform this study on face recognition focusing on facial occlusions and class set evolution:

[1] The size of the feature vector, i.e. the number of dimensions, is tied to the chosen neural network architecture, as it was explicitly implemented by the network's designer

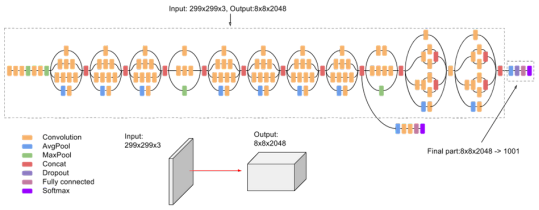

Figure 4.1: Inception-v3 Architecture [Goo19]

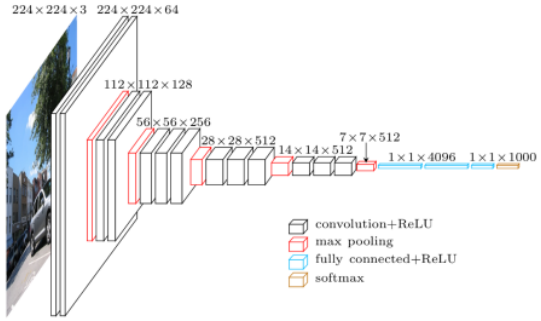

Figure 4.2: VGG16 Architecture [Fro19]

- InceptionV3 [SVI+16], as in FaceNet [SKP15]
- and VGG16 [SZ14]

By class set evolution, two factors are involved: variable number of pictures per person, and variable number of people to recognize.

Two other networks were considered, but their inherent architecture proved too complex for the problem at hand. They were ResNet50 [HZRS16] and SENet50 [HSS18]. The feature vector generated by those networks had 2048 dimensions, four times as much as VGG16's 512 feature vector, and such complexity requires a more substantial amount of information to train correctly, and the number of training examples necessary to reach a redeemable accuracy using these networks would not fit with the case at hand.

There is much focus dedicated to the choice of feature vector because its generation is part of the data preprocessing procedure. Not when working in the mobile environment, but in training, the vectors are predicted and then kept in storage. Fine-tuning a neural-network with a custom final layer is the same as training a softmax classifier that takes a feature vector as input, only faster since it bypasses all of the layers that belong to the pre-trained model. The two models can be stacked together later for real-time inference.

4.1.2 Data Augmentation

Data augmentation was a great advancement in Deep Learning. Properly utilized, it can take a deep learning model from adequate to good, or from good to great [PW17]. It is a means of artificially expanding a dataset by introducing some slight variations or noise to the input data in a way that its label continues to hold true but also so that bitwise, it is an entirely different file. Different, more sophisticated methods for data augmentation exist, such as using GANs to change the setting of a picture from day to night, or from summer to winter [ZPIE17], or, in the context of face recognition, to combine two faces of different identities, in order to generate a realistic, third identity that does not exist in the real world [KALL17].

Both of the techniques described in the following sections, namely geometric transformation, and random face patches, were used to increase the number of available training samples, each by

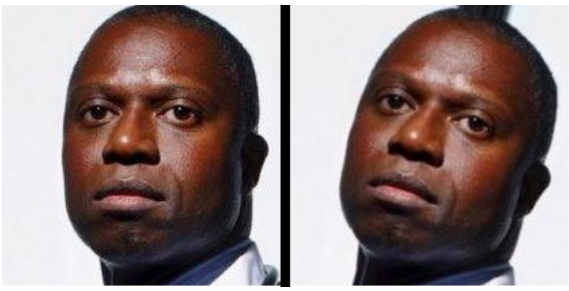

Figure 4.3: Rotation vs Original

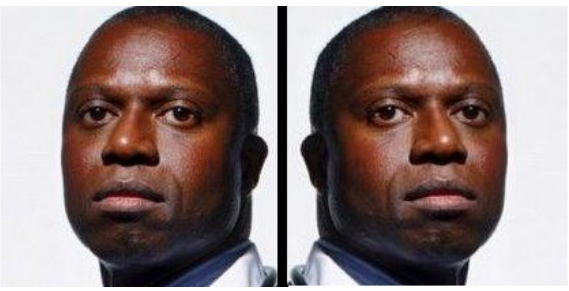

Figure 4.4: Horizontal Flip vs Original

a factor of twenty. This multiplication means that when using all available augmented data, plus the original examples, the size of the training dataset becomes

$$D' = D + 20D \tag{4.1}$$

where **D'** is the number of total training examples, and **D** is the number of original training examples.

There is no overlap between them. An augmented example was generated either by geometric transformation or by superimposing with a random patch. To assess the relevance of the face patches, mimicking natural facial occlusions, is one of the objectives of this study, and because of that, its usage must be isolated so that one can correctly assess the benefits or disadvantages that it might bring.

Geometric Transformations

Even though plenty more could have been used, only three types of geometric transformations were used, as to not deviate too far from the original so that the ground truth no longer holds. These were:

- Rotation (figure 4.3),
- Vertical or Horizontal Shift (figures 4.5 and 4.6),
- Horizontal Flip (figure 4.4).

All three are applied simultaneously, except for the horizontal flip, which for any given picture, has a 25% chance of being applied. Every time a new augmented example is created a random rotate or shift is applied, though it can be of zero degrees or zero pixels, respectively. The rotation angle can go from -15 degrees to 15 degrees and the shift range can reach up to 15% of the height or width of the picture in question.

Figure 4.5: Horizontal Shift vs Original

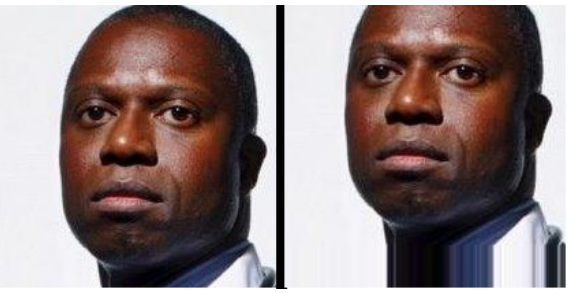

Figure 4.6: Vertical Shift vs Original

Random Face Patches

The proposed solution described in this work will focus on the problem of occlusion. Despite a deep neural network being able, to some degree, to intuitively learn to overcome this problem, as was shown by DeepID's research team:

> "[DeepID2+] is much more robust to occlusions, although occlusion patterns are not included in the training set." [SWT15]

It could still benefit from extra attention. Moreover, given the importance that occlusion robustness would have in a hospital environment, it justifies this extra attention given in training to this problem.

To simulate occlusion, random patches were overlayed on top of the face pictures, in the manner represented by figure 4.7.

Ideally, more realistic examples should have been used, ones that more closely resemble eyeglasses, or eye patches, or any type of bandage. The chosen training dataset already includes natural examples of facial occlusions, such as hair over the face, or sunglasses, but they are relatively sparse. That is why this specific configuration for the face patches was chosen. Three parameters can describe this configuration in specific:

- shape,
- size,

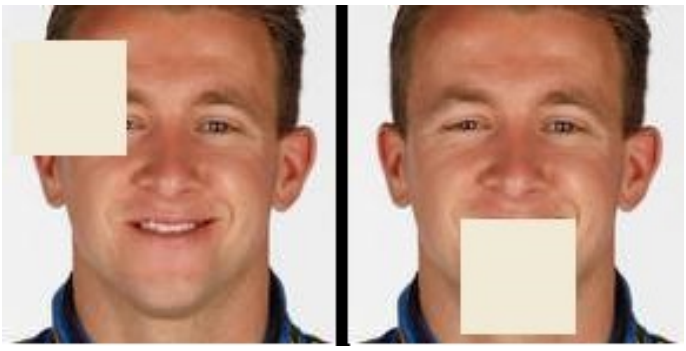

Figure 4.7: Examples of Pictures Covered by Face Patches

- color.

The square shape was chosen for its simple and efficient implementation. With limited hardware resources, it was indeed a factor to consider.

The length of the square patch is one-third of the picture length, an arbitrarily chosen value. This choice was made just so the area covered by the obstruction is neither too small so that it has little effect, nor too big so that it risks covering the whole face and make it so difficult to identify that even a human would be unable to do it. The color, with RGB values of 240, 234, 214, respectively, is colloquially referred to as eggshell white, another arbitrary choice, meant to mimic the color of most hospital bandages. This is, however, the least important of all the parameters for the face patch.

4.2 Data Processing

The scientific problem to tackle in this dissertation is 1-n classification of human faces, focusing on occluded faces, ease of attainment of representational face pictures, i.e. populating a database, and flexibility towards additions and subtractions regarding said database.

Several options are presented in order to choose the one alternative that best solves all of the problems mentioned above.

4.2.1 Hyper-Parameter Optimization

The process of minimizing loss and increasing accuracy on a neural network is mostly a trial and error approach. A data scientist would try some initial values for the network's hyper-parameters, analyze the results, adjust the values based on the results, and repeat until no gain is achievable.

The hyper-parameters tweaked in order to optimize the network were the dropout rate, learning rate coupled with the optimizer, the batch size and, in a way, the number of epochs.

Their relaxed result will be posted in this chapter, as opposed to the actual parameters of investigation of this dissertation, which will be presented in chapter 6. These hyper-parameters, despite crucial, are not the main focus of this work, and therefore, in order to minimize the number of variables presented ahead, the results shown in chapter 6 will assume the best possible combination of these hyper-parameters.

Dropout Rate

The dropout rate is a unit interval, and in most experiments conducted, the best results were attained with a dropout rate of 0.75, although any value between 0.6 and 0.8 shows little difference in performance. This statement agrees with the original dropout research where it is claimed that the best dropout rate for input layers is between 0.5 and 1 [SHK+14].

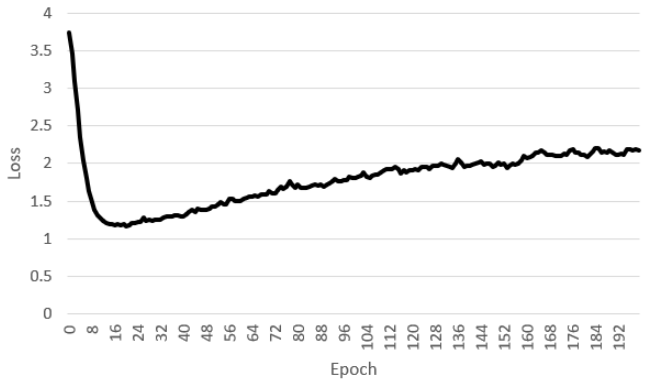

Figure 4.8: Possible Influence The Number of Epochs Has Over The Performance of The Network

Learning Rate and Optimizer

The value for the learning rate is tightly coupled with the choice for optimizer, as the same value can have drastically different effects on finding the global minimum on different optimizers, such as Adagrad or RMSProp. That said, the learning rates used range from 0.00001 to 0.001, depending on how big or small the amount of training data is for any given trial.

Batch Size

The batch size is another superficially simple, but in reality relatively deep parameter. If it is too big, it converges more quickly but generally not towards the global minimum; if it is too small, the final value for the loss is improved but it takes more time to converge. Furthermore, as the different pre-trained models are of different sizes, it is also necessary to adjust to the model used to generate the feature vectors.

The values can range from 32 to 512.

Number of Epochs

The number of epochs is not a great concern. The most important aspect to assure is that it is not too small as to not give the network enough iterations to converge. If the value is too big two concerns are raised: It takes longer than necessary to train, and it risks seeing the loss regress and the performance of the network actually worsening. It becomes a bigger concern when speed of deployment becomes an issue, an even while not giving it attention it is possible to find its optimal value: logs of training are generated and it is possible to know to which epoch corresponds the best performance, as shown in figure 4.8. Also, periodic checkpoints are saved so the theoretical best epoch is always closely available.

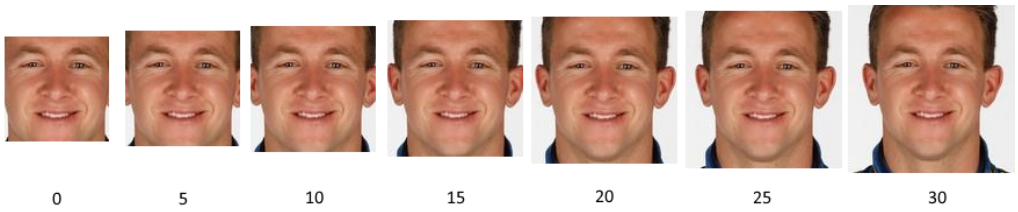

Figure 4.9: Visualizing Different Face-to-Background Ratios

Face-to-background Ratio

While not a hyper-parameter in the context of deep learning, I chose to describe it as such in this work because it influences the final results, but it is not complex nor relevant enough to justify making it a dependant variable. Its optimal value is still achieved as opposed to set, but it is only related to the pre-trained model used in each trial.

This parameter is treated as a percentage, but only seven, arbitrarily chosen, discrete values are available: 0, 5, 10, 15, 20, 25 and 30. Zero corresponds to the raw output given by the face detection algorithm, all the subsequent values are a percentage of the height and width of the 0% crop and are used to calculate the additional margin to be given to the face picture as shown in figure 4.9. This margin can be calculated as:

$$\frac{W \times P}{100} \tag{4.2}$$

Where **W** is the width or height of the detected face and **P** is the percentage.

4.3 Classification Methods

Two methods for classification are compared in this work. In order to test their relevance the metrics are:

- accuracy and
- time to update.

What is being measured is how well the model can classify the identities it is prepared to identity, and the amount of time it takes to redeploy with an updated list of identifiable people. The list of identities to classify is a dynamic one: identities can be removed and identities can be added. When such an event occurs, the system in place needs to be capable of responding to these changes.

Accuracy is measured in three different ways: the maximum achieved value, the value when the loss is minimum and the average accuracy after there is a decrease in the evolution of the model. This third measure stems from the normal behaviour of a neural network when it is being trained: the increase in accuracy a network experiences over time starts high and gradually levels

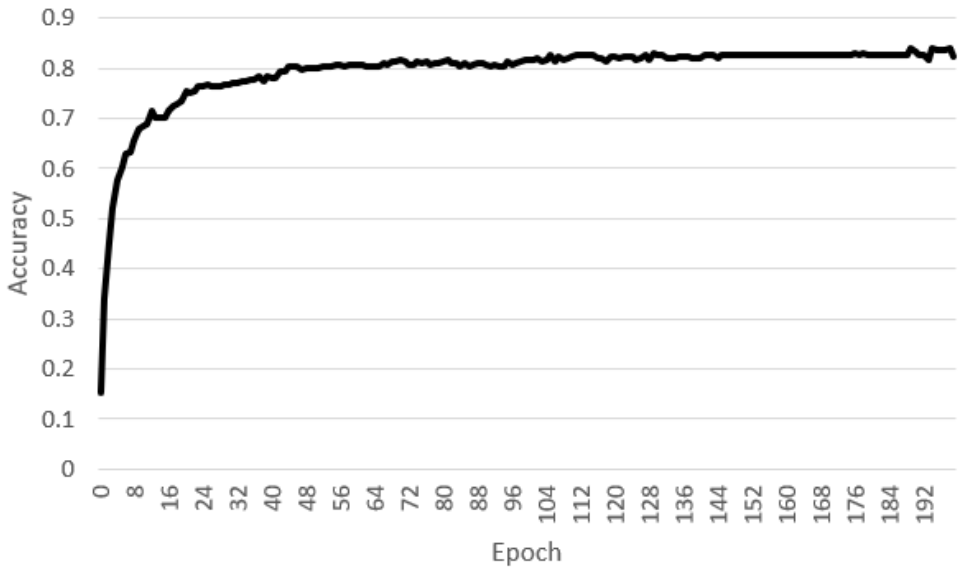

Figure 4.10: Generic Training Session

off, as shown in figure 4.10. This levelling off can be thought of as a plateau and it is after the network goes into this phase that average accuracy is calculated. In order to calculate the plateau in an objective manner, the average accuracy for every tenth epoch is computed using the accuracy values for the previous ten epochs. Each of those values is then compared to the aforegoing value, the first tenth epoch were this ten-epoch average had an increase in accuracy of less than 5% is considered the start of the plateau and every epoch that follows is used when calculating the post-plateau average.

Two options for classification arise, as described in the following sections:

Softmax Classifier

The Softmax Classifier is a simple two-layer fully-connected network, with dropout in between those two layers. As there are only two, they are, respectively, the input and output layers. The input layer has as many nodes as the feature vectors have dimensions. That means they are dependant on the pre-trained model used to generate the feature vectors. The output layer, however, is not dependant on the length of the feature vectors. Its size is dictated by the number of identities to classify. Its actual value is equal to the number of identities, plus one. This additional node represents the **unknown** class. This node is added as a precaution meant to reduce the risk of a false-positive.

There is the risk of trying to identify a person that is not currently in the set of identifiable people. If such was the case, the softmax classifier would classify that person as being someone else, and would thus provide an incorrect guess. To counter this, another class is added, one that represents every person not in the set of identifiable people. This class of unknown people is handled by adding an supplementary set of pictures, of disjoint identities, to the training dataset.

Those added pictures are all labeled as the same identity, despite being as sparse as possible. The weight given to this class was 10% of the training dataset, as to provide enough training examples to make the class of unknown people generic enough to distinguish from all other identifiable classes, but not too big as to make the network unbiased enough to over-identify the validation examples as unknown.

Vector Distance

Despite the number of dimensions, these feature vectors can be treated as a simple n-dimensional vector, or a one-column matrix. Arithmetic operations can be done on those matrices, such as addition, subtraction or multiplication.

The training of the model, when working with vector distance and not softmax classification, is not training, per se. It is only described as such in order to make an analogy to what is done when in softmax classification mode. What is being done, in fact, is calculating the mean vector for each identity, i.e. all the available feature vectors for each identity on the training split are grouped together and their mean is calculated, resulting in one additional feature vector per identity.

The validation phase iteratively calculates the distance between each validation example and every mean feature vector by subtracting one against the other. The distances are then sorted and the lowest distance is treated as being the identity of the specific validation example being tested.

Similarly to softmax classification, there is a risk of identifying a person that has not yet been added to this database as someone else. In order to prevent this from happening, i.e. the return of a false positive, a value for a bias is specified. This bias acts as a threshold between **sure** and **unsure** identifications. Distances that are above this bias value are treated as an unsure identification and are therefore discarded, or in other words, classified as a person outside of the list of known people.

The matches via vector distance are calculated via methods: **euclidean distance** and **cosine similarity**.

Chapter 5

Architecture

In this chapter the proposed system is described. It outlines the architecture of the system that would be put in place in a hospital environment, focusing mostly on how the model can be deployed into a mobile device.

The first section begins by progressively narrowing the focus of the system, starting with the overall context in order to provide a step by step description of each container's interactions.

The second section of this chapter illustrates how the information is handled in order to find the best combination of parameters prior to streamlining the deployment of the model described in section 5.1.

5.1 System Description

Three levels were considered when describing this system's architecture, different from each other by level of abstraction:

- **Context**, with the highest level of abstraction.
- **Container**, where inter-component interactions are layed out.
- **Component**, where the inner-workings of each component is described.

5.1.1 Context

Figure 5.1 provides an overview of the proposed software system. A simple division into three parties where at the center is the final product:, i.e. the mobile application. Both of the other parties interact with it. The remote system, which will be expanded on in the following sections, interacts with the mobile application only when transferring the finalized face recognition model. The mobile application has one use case: perform patient validation through face recognition, represented here by the connection between the user and the mobile application.

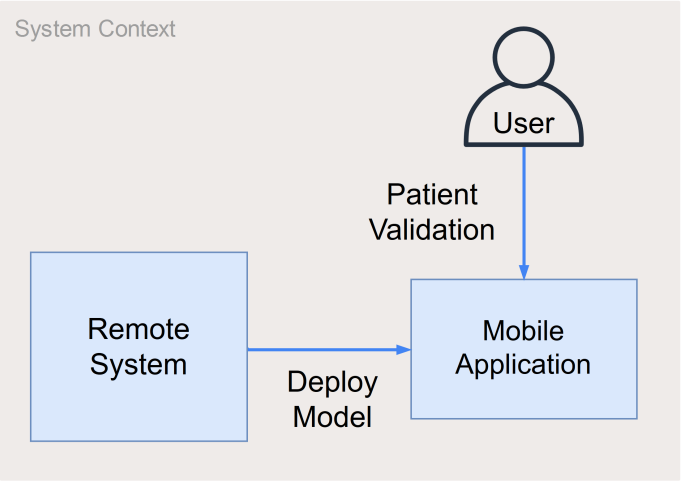

Figure 5.1: System Context Diagram

5.1.2 Containers

Remote System

The container diagram aims to expand on the remote system. When observing the software system through the lens of the overall context (figure 5.1), the user is not a piece of software, i.e. not expandable, and the mobile application should be considered as a lower level component. Therefore the only element left to expand is indeed the remote system, pictured in figure 5.2, which can then be detailed further. Its components are: a worker responsible for the pre-processing of data; a worker responsible for using that data to train and deploy a finalized face recognition model; and a database capable of holding all the files and information necessary to train said model.

The remote system carries the bulk of the workload. The reason for that is that the face recognition model needs to be updated every time a new patient is admitted. Whenever this event occurs, the entire process within the so-called remote system needs to be executed. And although not requiring the training worker to run, the pre-processing worker must also be activated anytime a

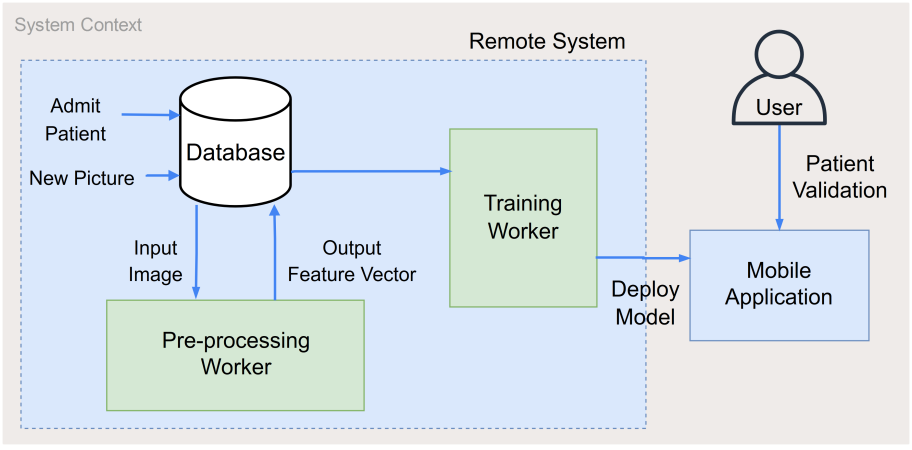

Figure 5.2: Container Diagram

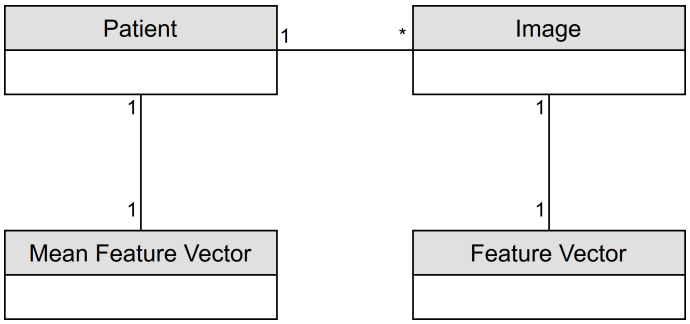

Figure 5.3: Database Specification

new face picture of an existing patient is introduced. This routine will be expanded in section 5.1.3, it describes how to obtain said face picture's corresponding feature vectors. These will be stored in the database and, together with the information pertaining to the already existing identities, fed to the worker responsible for training and deploying the final face recognition model.

Database

Despite being represented within a container, the database itself is a container as well, with it being a data host.

Its specification, as shown in figure 5.3, is a simple one, with the class attributes being omitted, allowing the concepts to remain somewhat abstract.

The patient is the core class within this schema. Each image has only one identity assigned to it, however, each identity, i.e. patient, can be represented by multiple images, which is why the multiplicity between patient and image is of one-to-many.

Both the original face pictures and their corresponding augmentations, described in section 4.1.2, are here treated simply as images, seeing as the augmented examples share the same identity as the original ones. This serves to specify that each image, augmented or otherwise, must have a corresponding feature vector, one that will be generated by the pre-processing worker.

The mean feature vector has a multiplicity of one-to-one with the patient because it represents the aggregation of all generated feature vectors for each identity.

5.1.3 Components

As stated above, the components featured in this specification are three: the worker responsible for data pre-processing, the worker responsible for training and deploying the face recognition model, and the mobile application. Each of these and their inner-workings is described below.

Pre-Processing Component

As can be seen by figure 5.4, the pre-processing worker interacts only with the database, receiving an image and returning one feature vector for each augmented example of the image received as input, plus the feature vector of the original image.

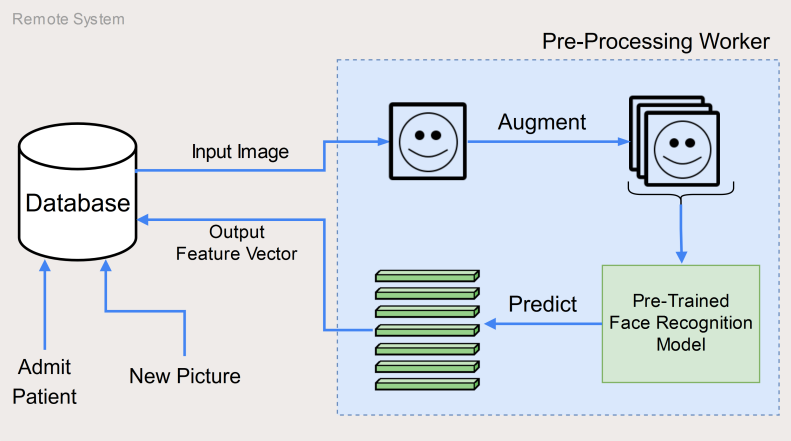

Figure 5.4: Pre-Processing Component Diagram

This process is triggered by a new patient being inserted into the database. With each new patient, one or more face images are also added, and for each of those face images the same process must be followed: firstly, the picture is augmented, creating copies of it, with some slight variations such that the picture is still recognizable as being the same person, but ensuring it is a different data pattern for the neural network to analyze; secondly, each of the newly generated augmented examples is fed to a pre-trained deep learning face recognition model in order to generate its prediction; finally, those predictions are inserted into the database.

Training Component

The training worker, as shown in figure 5.5, runs everytime an updated list of identifiable patients is generated. This generally means a patient has been admitted or released, however, since a patient being released does not constitute a big enough change, the new model is only generated when the updated list includes a new patient. This is the case because if a patient is released, it is still possible to use the model in place to correctly identify all the other patients. However, if a new patient is introduced, that patient is not identifiable until a new model is generated, one that included this new patient.

In order for a new model to be generated it first needs to be trained and with that purpose it

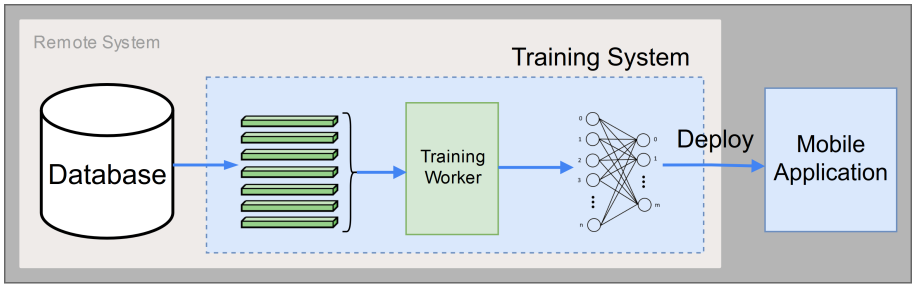

Figure 5.5: Training Component Diagram

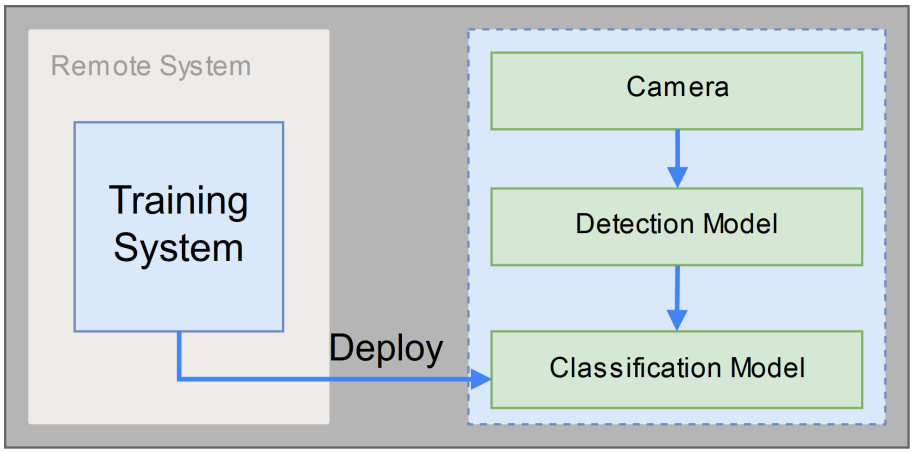

Figure 5.6: Mobile Application Component Diagram

is fed the feature vectors generated in the pre-processing phase. This training is a two-part fine-tuning of the pre-trained face recognition model used to generate the feature vectors. To lower the necessary computational power required to train a neural network with a small window of time to deploy, the final softmax classification layer, i.e. the softmax classifier described in section 4.3, is trained independently of the convolutional pre-trained model, and posteriorly stacked, to allow for the input to be raw images and not feature vectors. The stacking of the pre-trained model with the softmax classifier results in adding one final softmax layer to the pre-trained model. The only downside with this approach is the increase in necessary disk space.

Mobile Component

The mobile application, as shown in figure 5.6, has three main modules: the camera, the detection model, and the classification model.

The camera is the source of input. Low-resolution pictures are taken and passed through to the detection model, prior to being classified.

The detection model is an essential part of this process, as it is necessary to ensure that the data fed to the classification model is similar to the data used to train it. The detection model serves essentially for real-time data pre-processing.

The classification model is the construct of the remote system and the only module of the mobile application that needs to be updated. As previously mentioned, it is the result of adding a final softmax layer to the pre-trained face recognition model so that it can receive image data as input, and output class probability scores.

Both the detection and the classification models mentioned in figure 5.6 are, in reality, tflite files. These files are compressed, mobile optimized versions of tensorflow deep learning models. However, despite this intrinsic optimization provided by Tensorflow Lite to run models in SoC processors, the presence of a second deep learning model can be felt during inference because of the increase in execution time.

The mobile application, to be used as a proof of concept, is Tensorflow's demo application [Ten19b], with some slight modifications.

5.2 Research Pipeline

All the computations are guided by python scripts, with recourse to third-party libraries such as Tensorflow, Keras and NumPy. The whole training procedure has three distinct steps:

- dataset filtering and pre-processing,
- feature vector extraction,
- and fine-tuning.

The dataset used in this dissertation was CelebA [LLWT15]. The first task is to reduce the number of identities. This dataset has information pertaining to 10177 distinct identities. However, due to time and technical constraints, the decision to use only the first 500 identities provided was made. Ideally, the number would be closer to 1000, the approximate amount of occupied beds at any given moment in Porto's main hospital, Hospital de São João [Min19]. The amount of data to store and iterate through would be too large, hence why this decision was made. Most hospitals' occupancy, however, is closer to the smaller value used, if not smaller. The cited hospital would serve mostly as an extreme case.

After the filtering is done, the information is then split into two separate directories, one for training and another for validation, with a proportion of 90 to 10 percent, respectively. This is not the exact proportion, however. It is guaranteed that each person has at least one validation example, so, if the number of pictures for any given identity is less than 10, one of those pictures would necessarily be used for validation, even if it means half of the available examples would be used for validation.

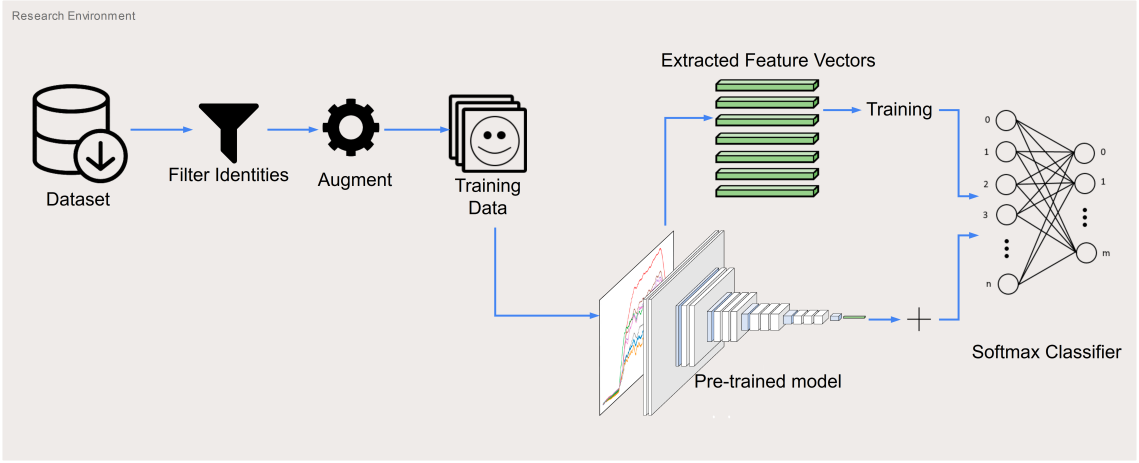

Figure 5.7: Research Pipeline

Afterward is the pre-processing phase. The description of this step is skipped here as it is better described in chapter 4.

Every usable picture is then fed into a pre-trained neural network, and its prediction[1] is saved into a parallel directory where all the feature vectors are stored. Because different versions of the same picture, even crops of the same photo, produce different feature vectors, some redundancy is present when storing the prediction results, as one input is replicated 14 times: for each of the two pre-trained models used in the experiments, there are seven different values for the cropping percentage. This constitutes the prediction step.

A pre-trained model is inherently more accurate when the data passed to it for prediction or validation is similar to the data it used when being trained. The aspect that I found influences this accuracy the most is how aggressively the original picture is cropped. A value of 0% margin would crop out the chin and half of the forehead, while a value of 30% would still leave a significant area of the background visible.

[1] Prediction is the name given to the output of a neural network.

Chapter 6

Results and Analysis

6.1 Experiments

The variables studied in this work are described in this section while the results of the experiments are posted in section 6.2.

The methodology, so to say, was to test all combinations of these variables, within realistic limits.

It must be said, however, that the first and second round of tests do not use the same amount of information. Due to an error when generating the results, only detected after analysis, the first round of tests[1] does not take advantage of dataset augmentation, i.e. only the original examples were used for training and validation, making it so those models were trained on 20 times less information.

Table 6.1: Parameter Combinations for First Round of Tests

Model	Softmax	N Identities	Exs/Identity
VGG ǀ Inception	True ǀ False	50 ǀ 100 ǀ 250 ǀ 500	1 ǀ 2ǀ 3 ǀ 4 ǀ 5 ǀ ∞

Table 6.2: Parameter Combinations for Second Round of Tests

Model	Softmax	Trained w/Occlusion	Validated w/Occlusion
VGG ǀ Inception	True ǀ False	True ǀ False	True ǀ False

6.1.1 Pre-Trained Model and Softmax Classifier vs Vector Distance

The choice of pre-trained model and whether to use a softmax classifier or perform classification by calculating the vector distance are the only two variables common to all tests. Their only

[1] The one aimed at studying the best choice of pre-trained model architecture, classification mode, number of identities and examples per identity

metric will be the difference between their final validation accuracy by comparing every test case where the model chosen is VGG versus when the model used is Inception, or Softmax at true versus Softmax at false The two specific test cases compared only differ on that one variable. An example is presented in tables 6.3 and 6.4.

Table 6.3: Example Test Case for Model Comparison

Model	Softmax	N Identities	Exs/Identity	Accuracy
VGG	True	100	1	vgg_accuracy
Inception	True	100	1	inception_accuracy

Table 6.4: Example Test Case to Compare Softmax Classifier vs Vector Distance

Model	Softmax	N Identities	Exs/Identity	Accuracy
VGG	True	100	1	softmax_accuracy
VGG	False	100	1	v_distance_accuracy

6.1.2 Number of Identities

The number of committed patients in a hospital will inevitably change, even throughout the day. Not only that, different hospitals have different capacities, so it is crucial to know how well these facial recognition algorithms scale in that regard, in order to figure out up until what value they are feasible to implement, if at all.

As previously mentioned, the maximum number of patients to be tested is 500; for the minimum, however, the choice was 50. A value too small would lose its relevance, so 50 was chosen as the starting point. The intermediary values serve to assure that the progression of the metrics is not linear.

Table 6.5: Example Test Case for Metric Evolution with Varying Number of Identities

Model	Softmax	N Identities	Exs/Identity	Accuracy	Training Duration
VGG	True	50	1	accuracy_50	dur_50
VGG	True	100	1	accuracy_100	dur_100
VGG	True	250	1	accuracy_250	dur_250
VGG	True	500	1	accuracy_500	dur_500

6.1.3 Examples per Identity

The metrics for the number of examples per identity are the same as for the number of identities, i.e. accuracy and training duration, because it is necessary to perform the comparison with both values in tandem. Common sense dictates accuracy will increase with the size of the dataset, provided the number of classes to identify remains the same, while training duration increases. The goal is to find the best ratio of accuracy to training duration, as a good accuracy is required, but

rendered useless by exaggerated training times. It is still necessary to keep in mind that in a real-world scenario multiple pictures of a person might be hard to acquire. In a hospital context, these pictures might be taken at the moment of triage, and it might be inconvenient, if not impossible, for time reasons, to get any number of pictures.

The values tested are: 1, 2, 3, 4, 5 and infinity. By infinity what is meant is there is no restriction to the number of examples per identity. The real value for infinity is the total amount of pictures provided by the dataset. In most cases the actual number of examples per identity varies from identity to identity, and is listed in table 6.6.

Table 6.6: Average number of pictures per person in various dataset splits

Num Identities	Train Exs. Total	Exs./Identity
50	941	18.82
100	1983	19.83
250	4816	19.26
500	9738	19.48

Table 6.7: Example Test Case for Metric Evolution with Varying Number of Identities

Model	Softmax	N Identities	Exs/Identity	Accuracy	Training Duration
VGG	True	100	1	accuracy_1	dur_1
VGG	True	100	2	accuracy_2	dur_2
VGG	True	100	3	accuracy_3	dur_3
VGG	True	100	4	accuracy_4	dur_4
VGG	True	100	5	accuracy_5	dur_5
VGG	True	100	∞	accuracy_inf	dur_inf

6.1.4 Occlusion Analysis

The objectives with occlusion analysis are to measure how much influence the proposed method for increasing occlusion robustness has over the final result, and to measure any possible increase in real-life use cases, where the face is indeed occluded, and not only artificially occluded, as is the case.

Table 6.8: Test Cases for Occlusion Analysis
(N=Normal, O=Occluded; TE=Training Examples, VE=Validation Examples)

...	NTE	OTE	NVE	OVE
	True	False	True	False
	True	False	False	True
	False	True	True	False
	False	True	False	True

Even though better results can be achieved by combining both groups of training examples, i.e. with and without artificial occlusion, the main goal is to assess said artificial occlusion's relevance,

therefore it was deemed necessary to isolate the different groups of input pictures. The increase in accuracy gained by combining them would be due to the increase in size of the training dataset, and not in the combination of the input types per se. Simply put, in order to present reliable results it is necessary to maintain control over as many variables as possible, and the size of the dataset is indeed a very important one, which is why table 6.8 features no test cases exist where both normal and occluded training examples are fed to the classifier.

6.2 Results

In this section the results for the aforementioned experiments are published and analyzed. Analysis is made measuring both the accuracy of the face recognition models and the time it takes to obtain the results.

The raw results for every test case are available in Appendix A

6.2.1 Pre-trained Model

In figures 6.1 and 6.2, positive values for difference in accuracy, pictured in the vertical axis, mean that the architecture VGG16 obtained a higher accuracy, while negative values mean that InceptionV3 had the best results.

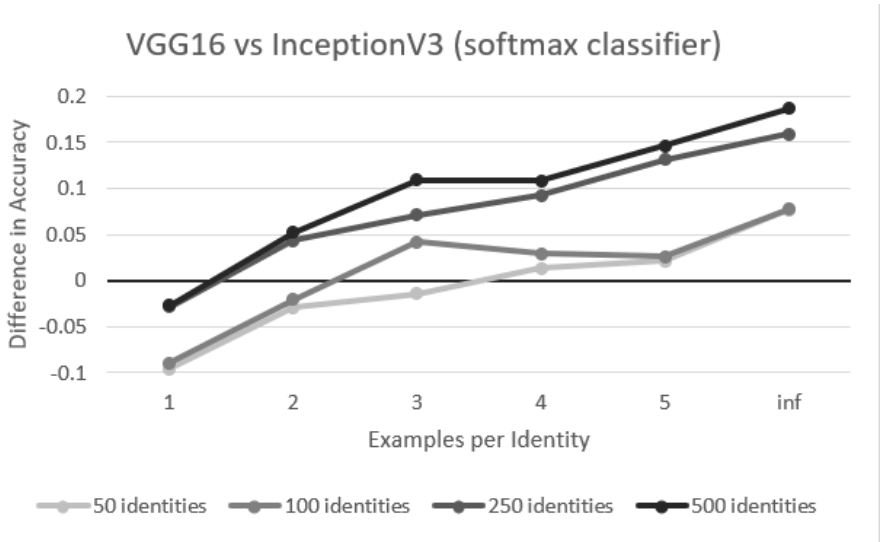

Figure 6.1: Comparison Between Pre-Trained Models

As shown in figure 6.1, the cases where Inception performs better are the ones where there is less information per class, i.e. fewer face pictures per identity.

This goes according to the expected seeing that the feature vectors produced by the InceptionV3 have fewer dimensions and are therefore more resistant to overfitting in the presence of smaller datasets, although making it less flexible when more information is available for training.

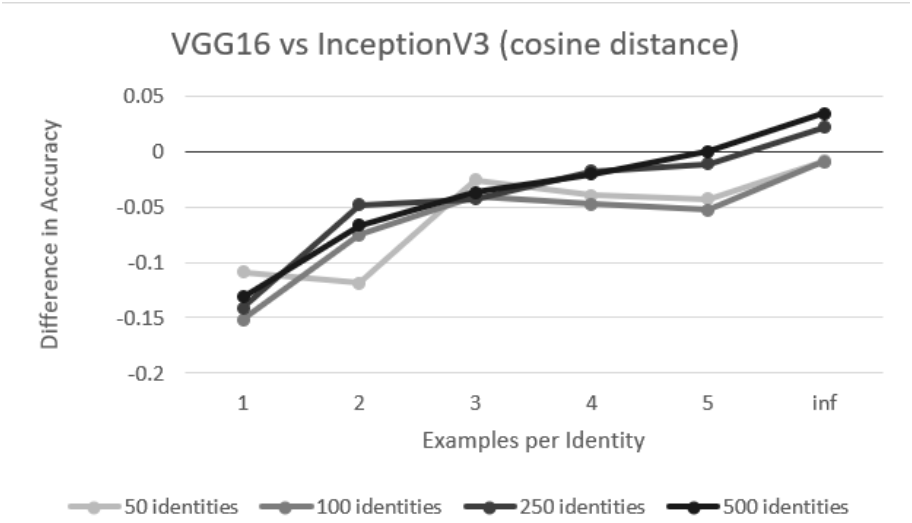

Figure 6.2: Comparison Between Pre-Trained Models

The same behavior is observable when comparing results obtained not with a softmax classifier but by cosine distance, as in figure 6.2. The main difference between the two is the position of the tipping point which is further to the right, meaning that using VGG16 generated feature vectors over InceptionV3 ones only becomes preferable with a higher number of examples per identity.

6.2.2 Softmax Classifier vs Vector Distance

In figures 6.3 and 6.4, positive values for difference in accuracy, pictured in the vertical axis, mean that the softmax classifier obtained a better accuracy, while negative values favor vector distance as the best performing classification method in terms of accuracy.

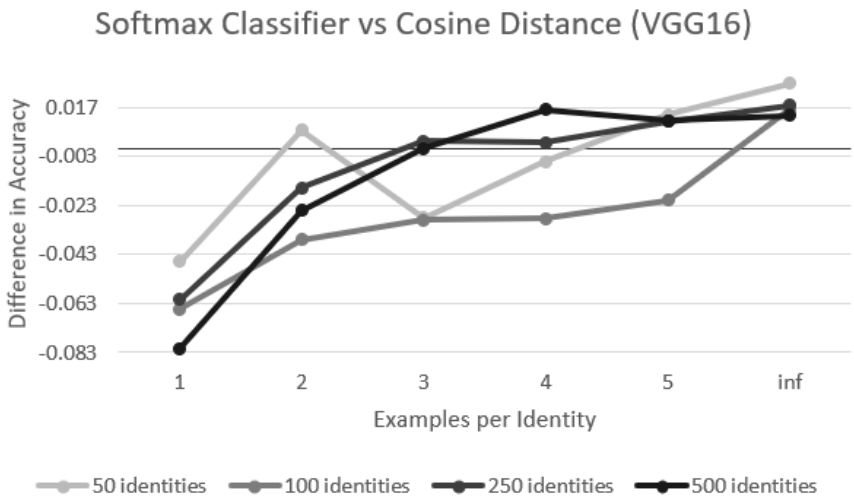

Figure 6.3: Comparison Between Classification Methods

From the graph represented in figure 6.3, one can witness the disadvantage felt by the softmax classifier. There is a clear evolution towards the positive range with an increase in number of face pictures per identity. This disadvantage is felt more in cases where there are fewer identities to classify. Both of these factors, examples per identity and number of identities, are revealing of the importance for a high amount of data when discussing the accuracy of a softmax classifier.

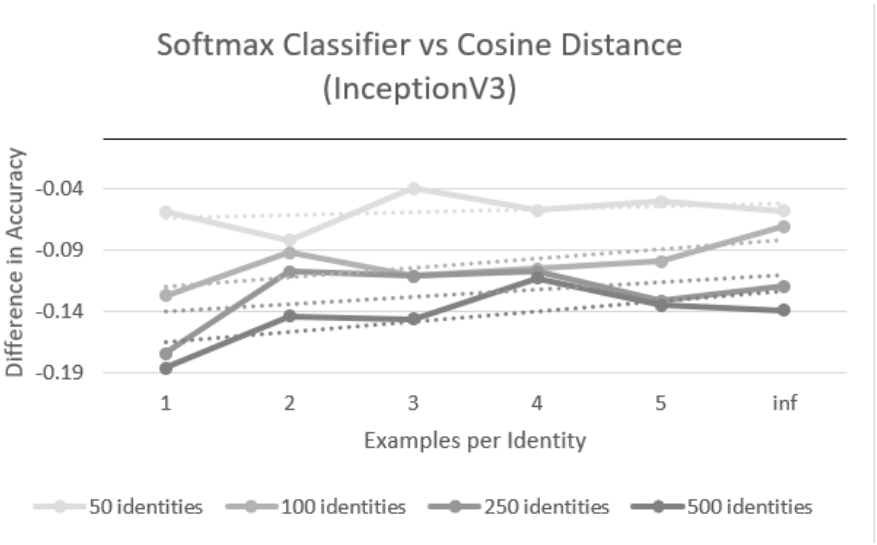

Figure 6.4: Comparison Between Classification Methods

Although showing a higher degree of entropy, figure 6.4 still shows a slight improvement in accuracy difference when increasing the number of examples per identity. However, the number of examples per identity where this value becomes positive is not contained within the range of realistic values for number of face pictures a hospital might hold of any given patient, which makes cosine distance the clear choice of classification method, provided the architecture used is not VGG16.

6.2.3 Number of Identities

In figures 6.5 and 6.6, the vertical axis holds the accuracy, represented as a value between 0 and 1, while the horizontal axis holds the number of identities.

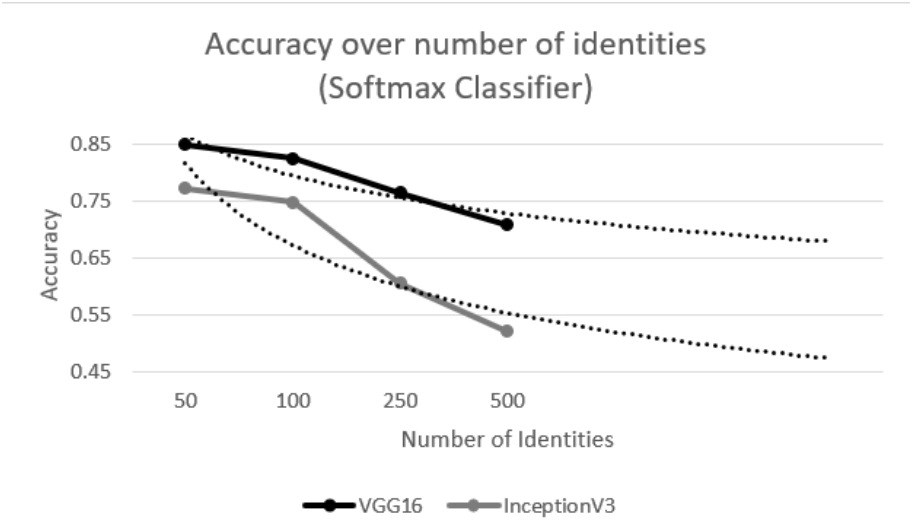

Figure 6.5: Analysis of Number of Identities

The chart in figure 6.5 might make it seem that the higher the number of identities, the higher the cost in accuracy, but such is not the case. Accuracy does decrease, however, the decline is not exponential. It is, in fact, inversely proportional, as can be seen by the dotted trendlines. Table 6.9 concretely displays the values for the slope between every consecutive pair of coordinates for the VGG test case.

Table 6.9: Slope Values for VGG Coordinates in Figure 6.5

Num Identities	Accuracy	Slope
50	0.848780489	
100	0.82510535	-0.000473503
250	0.76503418	-0.000400474
500	0.708060935	-0.000227893

Despite the drop-off not being as agressive as the charts in figures 6.5 and 6.6 make it appear to be, the low accuracy values still suggest that this sort of validation is more relevant towards hospitals with a lower patient volume.

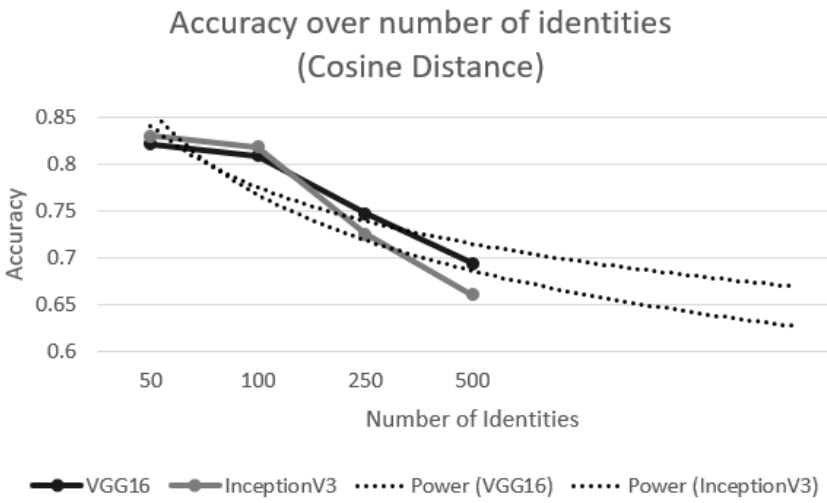

Figure 6.6: Analysis of Number of Identities

6.2.4 Examples per Identity

The charts in figure 6.7 shows the accuracy evolution regarding the provided number of examples per identity.

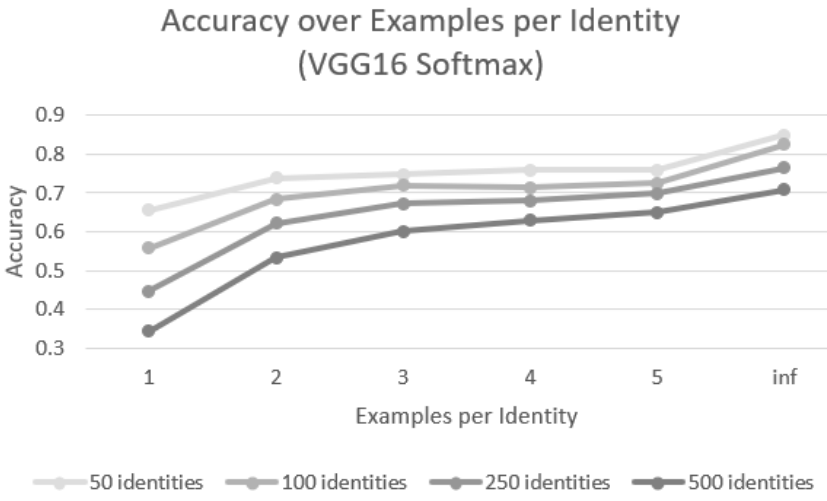

Figure 6.7: Analysis of Examples per Identity

As would be expected, accuracy increases with the number of face pictures per identity. However, a less straightforward conclusion can also be taken, simply from visually analyzing the graph: Its apparent funneling of the four displayed series[2] along the horizontal axis reveals that the more distinct identities are available for identification, the more the classifier benefits from having a greater number of examples per identity. This raises the awareness that if a patient database were

[2] 50, 100, 250 and 500 identities

to increase significantly, perhaps it would become necessary to update already inserted patients with additional face pictures.

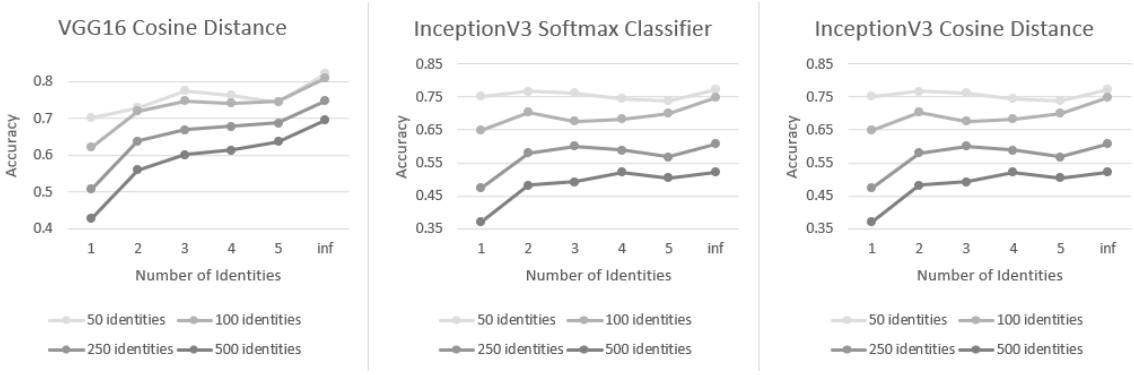

Figure 6.8: Analysis of Examples Per Identity

Aggregated in figure 6.8, in order to avoid redundancy, are the results for Accuracy over Examples per Identity for VGG16 Cosine Distance, InceptionV3 Softmax Classifier and InceptionV3 Cosine Distance. What can be concluded from those results is equal to what can be concluded from figure 6.7

6.2.5 Occlusion Analysis

Figure 6.9 shows values for accuracy when different splits of the dataset are used to train and to validate. The three adjacent bars represent different ways of measuring the softmax classifier's accuracy.

Decoding the chart's legend: NTE + NVE represents a test case where NTE (Normal Training Examples Dataset Split) was used to train the model, and NVE (Normal Validation Examples Dataset Split) was the split used to perform validation.

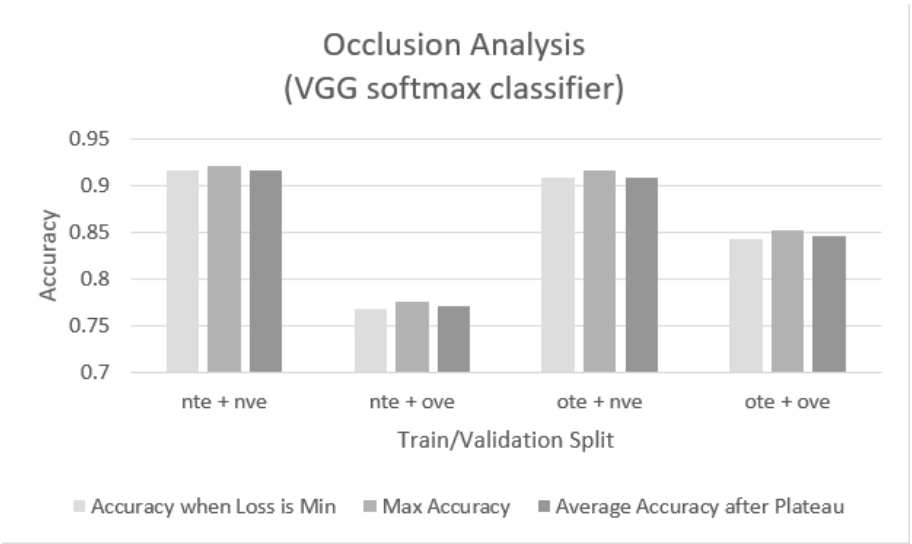

Figure 6.9: Occlusion Analysis

Several observations must be had when analyzing the differences in accuracy between the different train/validation splits.

When comparing the tests for NTE + NVE with OTE + NVE, two cases which differ on the training set, but are validated on the same information, there is a slight drop in accuracy, but barely noticeable (0.9159 for NTE + NVE, and 0.9078 for OTE + NVE). This is expected because normality lies with a neural network faring better while being validated against information that follows the same standard as the information used to train it. Given that the decrease in accuracy is difficult to notice, it can be concluded that the added face patches do not pose a hindrance to a facial recognition model, even when validating against uncovered faces.

Following that same train of thought, and seeing the other pair of test cases that is validated with the same information while being trained with different splits (NTE + OVE and OTE + OVE) suffers a noticeable decrease in accuracy when it is trained with uncovered faces (0.7676 for NTE + OVE, and 0.8430 for OTE + OVE), it leads to the conclusion that the accuracy is indeed influenced by the presence of face patches. While it is still not possible to affirm that training with face patches is an advantage, it is certainly possible to say that training without face patches is a disadvantage.

The most noticeable difference, however, can be seen between the test cases for NTE + OVE and OTE + NVE (0.7676 and 0.9078, respectively), trained and validated with different dataset splits. This comparison aims to measure the capability for generalization with this kind of augmentation, i.e., the face patches. A model trained without occlusion does not fare well when being validated against occluded examples, while the reverse is not true.

Figure 6.10: Occlusion Analysis

Similar tendencies can be observed when calculations are made with a different pre-trained model and with a different classification method, as shown in figure 6.10. One notable detail, however is that the overall values for accuracy are not as good as when obtained by VGG16 Softmax Classifier.

6.2.6 Time analysis

Time to deploy model is also an important factor when choosing a classification method. This factor is not as important as the accuracy, as a fast to deploy, low accuracy model is of little relevance in a practical context.

The timing result is posted in tables 6.10 and 6.11, represented in seconds.

Table 6.10: Timing Results for Variable Examples per Identity

Examples/Identity	VGG Softmax	VGG Cosine	Inception Softmax	Inception Cosine
1	72	58	60	63
2	155	95	101	92
3	193	132	150	127
4	260	175	188	154
5	252	198	231	185
inf	1720	1249	1407	1221

Table 6.11: Timing Results for Variable Number of Identities

Num Identities	VGG Softmax	VGG Cosine	Inception Softmax	Inception Cosine
50	137	11	118	11
100	306	46	306	48
250	810	260	719	273
500	1720	1249	1407	1221

The trend is similar on both variables, and as such it is possible to affirm that time to train increases in a linear fashion, depending on the number of training examples.

Two other conclusions are directly taken: VGG16 is slower than InceptionV3, and Cosine Distance is faster than Softmax Classification. The magnitude of these values, however, has little significance. It can be used as a baseline for possible future implementations of the system described in this dissertation, however, if such were to happen, there would be a need for dedicated hardware to run these operations. That hypothetical hardware would likely be more capable than a laptop with four physical cores running at 1.99GHz, with no GPU. Such a system is not feasible on low-end hardware.

Chapter 7

Conclusions

This dissertation had two main objectives: to assess the feasibility of implementing a face recognition system inside a hospital environment, and to assure such a system's resistance to facial occlusions.

As patients go in and out the list of people to identify mutates and as such the model capable of performing such a task needs to be updated in order to reflect that change. That is the main difficulty when assuring a face recognition system makes sense in such a dynamic environment. A static list of people can be correctly identified fairly easily, but that task takes time and a lot of computing power. This assessment of feasibility lays in measuring how well a fast to update system can recognize a person, and how fast a good, i.e. accurate, face recognition system can be updated.

A likely cause for the proposed face recognition system to perform poorly is the occurrence of identification cases where the patient's face is deformed or covered in bandages. This problem is commonly referred to as facial occlusion. The high chance of occluded examples makes it so this resistance to facial occlusions is also a main focus of this dissertation.

The presumed knowledge claimed that using deep learning approaches aimed at image classification, or in this specific case, face recognition, requires a large amount of information in order to achieve a decently performing model. While this is in fact true, that truth can be bent slightly. When generating augmented pictures out of one original example, one can successfully multiply the size of one's training dataset and use that increase in size to mitigate against the lack of training information. This is especially useful in a hospital environment, where it might be difficult to come by multiple pictures of one person.

It was also assumed that training a face recognition model while focusing on a specific type of image, in this case, occluded examples, would lower that same module's capabilities against the less specialized examples. This assumption, however technically true, should not be taken for granted. When validating against the same subset of non-occluded images the model trained with non-occluded examples did achieve an accuracy value that was 0.8% higher than the model trained

with the artificially occluded dataset. Although small, there is a discrepancy. The real difference, however, can be noticed when comparing the test cases that train and validate on different datasets. When doing so the advantage stands clearly on the side of the model that is trained with the facial occlusions, a difference of 14% in accuracy, and this value speaks volumes for the generalization ability provided by these training examples. That being said the conclusion that is able to be drawn is that this alternative form of augmentation is in fact of benefit when training a face recognition model and facial occlusions are a concern.

The time to deploy, however, benefits greatly from the usage of the vector distance method, over softmax classification. It is overall not as accurate as with the softmax classifier, especially when validated against occluded examples, however, it is much faster and easier to update. Its dynamic nature make it the safe choice in this regard. That being said, if a hospital has a low rate of admissions, i.e. less than 10 per day, or if this identification can be separated by departments, in order to minimize both the number of times it needs to be updated, the softmax classifier becomes a more than viable option, given its greater overall accuracy.

The face-patch approach to data augmentation was the point that the research conducted in this dissertation mostly focused on, and it is this dissertation main contribution. It objectively poses no disadvantage to a more general face recognition problem, while increasing a model's accuracy when focusing on facial occlusion. This augmentation technique can be improved. The aim was to keep the face-patches as simple as possible in order to lower the number of variables. If one were to make it more complex, changing its shape, size and/or color, the success or failure of this type of image augmentation could be due to any of those factors or their combination of values. This way one can measure the impact that its presence has, or lacks.

As a round up, this became a fairly successful endeavor, delving into the viability of implementing such a system as a hospital's method for patient validation, ending with the proposal for an alternative and computationally cheap data augmentation technique.

References

[ABC+16] Martin Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283, 2016.

[BBLP13] Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. Advances in optimizing recurrent networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8624–8628. IEEE, 2013.

[Bra00] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[C+15] François Chollet et al. Keras. https://keras.io, 2015.

[CCWS13] Dong Chen, Xudong Cao, Fang Wen, and Jian Sun. Blessing of dimensionality: High-dimensional feature and its efficient compression for face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3025–3032, 2013.

[CSX+18] Qiong Cao, Li Shen, Weidi Xie, Omkar M Parkhi, and Andrew Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *Automatic Face & Gesture Recognition (FG 2018), 2018 13th IEEE International Conference on*, pages 67–74. IEEE, 2018.

[Dan19] Dan Clark. Top 13 Python Deep Learning Libraries. https://www.kdnuggets.com/2018/11/top-python-deep-learning-libraries.html, 2019. Online; accessed 23 April 2019.

[DHS11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[DSH13] George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8609–8613. IEEE, 2013.

[Fro19] Frossard, Davi. Model and pre-trained parameters for VGG16 in TensorFlow. https://www.cs.toronto.edu/~frossard/post/vgg16/, 2019. Online; accessed 29 May 2019.

REFERENCES

[Goo19] Google Inc. Advanced Guide to Inception v3 on Cloud TPU. https://cloud.google.com/tpu/docs/inception-v3-advanced, 2019. Online; accessed 29 May 2019.

[GZH+16] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition. In *European Conference on Computer Vision*, pages 87–102. Springer, 2016.

[HMBLM08] Gary B Huang, Marwan Mattar, Tamara Berg, and Eric Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. In *Workshop on faces in'Real-Life'Images: detection, alignment, and recognition*, 2008.

[HSS18] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018.

[Hu,19] Hu, Yixuan. Tensorflow Face Detector. https://github.com/yeephycho/tensorflow-face-detection, 2019. Online; accessed 28 May 2019.

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

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[JM08] Hongjun Jia and Aleix M Martinez. Face recognition with occlusions in the training and testing sets. In *2008 8th IEEE International Conference on Automatic Face & Gesture Recognition*, pages 1–6. IEEE, 2008.

[JSD+14] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[KALL17] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

[Kar19] Karpathy, Andrej. Convolutional Neural Networks: Architectures, Convolution/Pooling Layers. http://cs231n.github.io/convolutional-networks/, 2019. Online; accessed 29 May 2019.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[LAE+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

REFERENCES

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

[LLF07] Yen-Yu Lin, Tyng-Luh Liu, and Chiou-Shann Fuh. Face detection with occlusions. *Images & Recognition*, 13(1):4–21, 2007.

[LLWT15] 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)*, 2015.

[Min19] Ministério da Saúde. Centro Hospitalar de São João, Porto. Relatório e Contas 2017. http://portal-chsj.min-saude.pt/uploads/document/file/658/R_C_2017.pdf, 2019. Online; accessed 3 June 2019.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[PVZ+15] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.

[PW17] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *arXiv preprint arXiv:1712.04621*, 2017.

[SA16] Frank Seide and Amit Agarwal. Cntk: Microsoft's open-source deep-learning toolkit. pages 2135–2135, 08 2016.

[SCWT14] Yi Sun, Yuheng Chen, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems*, pages 1988–1996, 2014.

[SDBR14] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[SHK+14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[SKP15] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SLWT] Y Sun, D Liang, X Wang, and X Tang. Deepid3: Face recognition with very deep neural networks. arxiv 2015. *arXiv preprint arXiv:1502.00873*.

[SVI+16] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

REFERENCES

[SWT14] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deep learning face representation from predicting 10,000 classes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1891–1898, 2014.

[SWT15] Yi Sun, Xiaogang Wang, and Xiaoou Tang. Deeply learned face representations are sparse, selective, and robust. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2892–2900, 2015.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[Ten19a] TensorFlow. TensorFlow. https://www.tensorflow.org, 2019. Online; accessed 22 April 2019.

[Ten19b] Tensorflow. TensorFlow Lite image classification Android example application. https://github.com/tensorflow/examples/blob/master/lite/examples/image_classification/android/README.md, 2019. Online; accessed 4 June 2019.

[TH12] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[The16] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[The19] Theano. Theano at a Glance – Theano 1.0.0 documentation. http://deeplearning.net/software/theano/introduction.html#what-does-it-do-that-they-don-t, 2019. Online; accessed 23 April 2019.

[TYRW14] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[UC 19] UC Berkeley. Berkeley Artificial Inteligence Research. https://bair.berkeley.edu/, 2019. Online; accessed 22 April 2019.

[VJ04] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.

[WLY+16] Yandong Wen, Weiyang Liu, Meng Yang, Yuli Fu, Youjun Xiang, and Rui Hu. Structured occlusion coding for robust face recognition. *Neurocomputing*, 178:11–24, 2016.

[WZLQ16] Yandong Wen, Kaipeng Zhang, Zhifeng Li, and Yu Qiao. A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision*, pages 499–515. Springer, 2016.

[ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.

Appendix A

Test Results

The following tables are divided into sections and subsections as an organizational measure, as the full compiled results were both too wide and too long to fit inside one page.

A.1 Variable Number of Identities and Examples per Identity

The following subsections assume the values for OTE, OVE, NTE and OVE as true, and are as such ommited from the tables.

Softmax Classifier Results

VGG16

n_iden	**ex/iden**	**acc_loss_min**	**max_acc**	**plt_avg_acc**	**sc time** (s)
50	1	0.659864	0.693878	0.654568	20
50	2	0.766187	0.766187	0.737950	20
50	3	0.752604	0.768229	0.748031	21
50	4	0.769710	0.780083	0.758257	25
50	5	0.774138	0.787931	0.758890	29
50	inf	0.856369	0.865312	0.848780	137
100	1	0.585859	0.585859	0.557576	20
100	2	0.704745	0.708260	0.683456	28
100	3	0.723567	0.743949	0.718623	37
100	4	0.720244	0.739573	0.713072	45
100	5	0.727350	0.735817	0.724979	54
100	inf	0.829396	0.834504	0.825105	306
250	1	0.466125	0.466125	0.446758	36
250	2	0.621795	0.650285	0.622379	65
250	3	0.677737	0.692268	0.672263	85
250	4	0.684580	0.690368	0.680666	109
250	5	0.705943	0.712126	0.698924	131
250	inf	0.769942	0.773890	0.765034	810
500	1	0.350410	0.373634	0.344690	72
500	2	0.546432	0.555038	0.533410	155
500	3	0.621509	0.621509	0.601341	193
500	4	0.642397	0.643853	0.629417	260
500	5	0.655542	0.659865	0.649194	252
500	inf	0.711963	0.714178	0.708061	1720

InceptionV3

n_iden	**ex/iden**	**acc_loss_min**	**max_acc**	**plt_avg_acc**	**sc time** (s)
50	1	0.775510	0.775510	0.750295	20
50	2	0.784173	0.802158	0.767033	20
50	3	0.768229	0.815104	0.762071	21
50	4	0.755187	0.773859	0.745202	25
50	5	0.746552	0.767241	0.737543	28
50	inf	0.771274	0.784282	0.772199	118
100	1	0.643098	0.673401	0.647093	20
100	2	0.724077	0.731107	0.704331	29
100	3	0.676433	0.692994	0.677138	35
100	4	0.700916	0.704985	0.684087	38
100	5	0.704488	0.716342	0.699164	47
100	inf	0.750223	0.756098	0.747338	306
250	1	0.495935	0.509485	0.475348	33
250	2	0.586182	0.610399	0.578870	55
250	3	0.607161	0.617021	0.600913	73
250	4	0.602728	0.614303	0.587982	91
250	5	0.577808	0.584679	0.567725	112
250	inf	0.609756	0.622141	0.606018	719
500	1	0.392760	0.407104	0.371708	60
500	2	0.490498	0.500896	0.481596	101
500	3	0.511355	0.514226	0.492207	150
500	4	0.535469	0.540670	0.521229	188
500	5	0.523085	0.524295	0.503205	231
500	inf	0.519407	0.532966	0.521282	1407

Vector Distance Results

VGG16

n_iden	ex/iden	acc euc_d	acc cos_d	vec time (s)
50	1	0.489796	0.700680	2
50	2	0.528777	0.730216	1
50	3	0.755208	0.776042	1
50	4	0.545643	0.763485	1
50	5	0.591379	0.744828	2
50	inf	0.626829	0.821951	11
100	1	0.444444	0.622896	3
100	2	0.553603	0.720562	3
100	3	0.582166	0.747771	5
100	4	0.595117	0.741607	6
100	5	0.610500	0.745978	7
100	inf	0.694547	0.808709	46
250	1	0.342818	0.508130	19
250	2	0.492165	0.638177	21
250	3	0.541775	0.668915	28
250	4	0.558082	0.677966	35
250	5	0.575404	0.687736	44
250	inf	0.639717	0.747282	260
500	1	0.297131	0.426230	58
500	2	0.432054	0.558623	95
500	3	0.494388	0.601149	132
500	4	0.522155	0.613480	175
500	5	0.541069	0.637731	198
500	inf	0.605137	0.694325	1249

InceptionV3

n_iden	ex/iden	acc euc_d	acc cos_d	vec time (s)
50	1	0.619048	0.809524	3
50	2	0.661871	0.848921	1
50	3	0.695313	0.802083	1
50	4	0.684647	0.802905	2
50	5	0.682759	0.787931	2
50	inf	0.700271	0.830352	11
100	1	0.609428	0.774411	5
100	2	0.664323	0.796134	3
100	3	0.672611	0.788535	4
100	4	0.638861	0.789420	5
100	5	0.614733	0.798476	6
100	inf	0.692121	0.818159	48
250	1	0.555556	0.649051	20
250	2	0.614672	0.686610	21
250	3	0.622211	0.711988	27
250	4	0.588673	0.695742	37
250	5	0.581931	0.699072	43
250	inf	0.621600	0.725434	273
500	1	0.489071	0.557377	63
500	2	0.548225	0.625314	92
500	3	0.549465	0.638215	127
500	4	0.534429	0.633867	154
500	5	0.533633	0.638077	185
500	inf	0.558194	0.660238	1221

A.2 Variable Dataset Splits for Training and Validation

The following subsections assume the values for Number of Identities as 100 and for Examples per Identity as infinity (see table 6.6) and are as such ommited from the tables.

Softmax Classifier Results

model	nte	ote	nve	ove	acc_loss_min	max_acc	plt_avg_acc
vgg	t	f	t	f	0.917140	0.920455	0.915982
vgg	t	f	f	t	0.770312	0.776042	0.767665
vgg	f	t	t	f	0.908144	0.916667	0.907877
vgg	f	t	f	t	0.846354	0.851563	0.843066
inception	t	f	t	f	0.880208	0.882102	0.875688
inception	t	f	f	t	0.748958	0.754167	0.746976
inception	f	t	t	f	0.866004	0.866004	0.857434
inception	f	t	f	t	0.784375	0.786458	0.774248

Vector Distance Results

model	nte	ote	nve	ove	acc euc_d	acc cos_d
vgg	t	f	t	f	0.826477	0.885066
vgg	t	f	f	t	0.525654	0.686126
vgg	f	t	t	f	0.354525	0.866367
vgg	f	t	f	t	0.580890	0.774869
inception	t	f	t	f	0.829718	0.856644
inception	t	f	f	t	0.477487	0.741623
inception	f	t	t	f	0.731488	0.832710
inception	f	t	f	t	0.771466	0.813613