UNIVERSIDADE
BEIRA INTERIOR

Deep Adversarial Frameworks for Visually Interpretable Periocular Recognition

João Pedro da Cruz Brito

Dissertation for Master's Degree in
Computer Science and Engineering
(2º cycle of studies)

Supervisor: Prof. Dr. Hugo Proença
Co-supervisor: Prof. Dr. Christoph Busch

Covilhã, june 2021

Acknowledgements

[WORK IN PROGRESS]

Abstract

[WORK IN PROGRESS]

Keywords

Artificial Intelligence, Convolutional Neural Networks, Deep Learning, Generative Adversarial Networks, Instance Segmentation, Image Synthesis, Interpretability, Latent Space, Machine Learning, Periocular Recognition

Contents

List of Figures

Acronyms List

ALE Accumulated Local Effects

AdaIN Adaptive Instance Normalisation

API Application Programming Interface

AI Artificial Intelligence

ANN Artificial Neural Network

BN Batch Normalisation

COCO Common Objects in COntext

CNN Convolutional Neural Network

DL Deep Learning

EU European Union

FFHQ Flickr Faces High Quality

FC Fully Connected

FCN Fully Convolutional Network

GDPR General Data Protection Regulation

GAN Generative Adversarial Network

GPU Graphics Processing Unit

ILSVRC ImageNet Large Scale Visual Recognition Challenge

IoU Intersection Over Union

LIME Local Interpretable Model-agnostic Explanations

ML Machine Learning

MSE Mean Squared Error

MLP MultiLayer Perceptron

PDP Partial Dependence Plot

ReLU Rectified Linear Unit

RNN Recurrent Neural Network

RGB Red Green Blue

RoI Region(s) of Interest

RoIAlign Region(s) of Interest Align

RoIPool Region(s) of Interest Pooling

RPN Region Proposal Network

R-CNN Region based Convolutional Neural Network

SHAP SHapley Additive exPlanations

SVM Support Vector Machine

UBIPr University of Beira Interior Periocular

VOC Visual Object Classes

Chapter 1

Introduction

In recent years, few areas of research have enjoyed similar manifestations of interest and enthusiasm as Deep Learning (DL). Staggering amounts of progress have been made, culminating in models that can do seemingly everything: image classification, object detection, image synthesis, amongst many others. As the remainder of this document will make clear, DL models (and other Machine Learning (ML) techniques) can be used in creative pipelines, incorporating the advantages provided by the plethora of different architectures. Such wide availability proves how accessible the field has become.

Fortunately for the research community, within the field of DL, there is still some ground to cover in emerging areas. Interpretability is one of those areas, with active lines of research. Most of the techniques found in literature find their way into existing systems due to a need for explanations. Not by design, but by requirement. Usually, this need comes after the system's deployment, as a quick fix. We argue that this mindset is no longer realistic, especially in areas where decisions need to be accurate and frequently audited. Even if the system's environment does not require intensive validation, ensuring some degree of interpretability should always be desirable (it could, perhaps, be considered a "good practice" in the industry).

Based on what was stated above, in this work we are interested in incorporating interpretable components into a system, that given a pair of images from the periocular region, is capable of delivering a twofold output: a ("yes" or "no") decision, supported by an easily (and visually) interpretable explanation. The former is similar to a classical approach, while the latter serves the emerging need for decision validation.

1.1 Motivations and Objectives

The present document aims at presenting a view of the state of ML Interpretability, as well as, describing ways to incorporate such techniques with standard DL methods. Thus, the goals of this work are, in general, an interpretable framework with solid performance and, in particular, one that, given two images, is able to, not only, determine if they belong to the same subject, but also, provide reasons that support such decision.
By extension, the motivations lie with the recent explosion of interest in interpretable Artificial Intelligence (AI) [LHGK19] and how it enforces a different mindset from the developer. Thus, given its context, the work described here culminated in the following realisation: the end result should not be measured in accuracy alone, but also in how interpretable it is.

1.2 Document Organisation

In order to provide an intuitive reading experience, this document is divided in the following chapters:

1. **Introduction** - provides a general overview of this work's motivations and objectives, as well as, the structure by which this document is organised.
2. **Related Work** - presents an extensive look at widely used algorithms and techniques in the fields of interest.
3. **Proposed Methods** - showcases the methods and intuitions developed to tackle the problem at hand.
4. **Results and Discussion** - contains the experiments performed to validate the proposed methods, with accompanying results and discussions.
5. **Conclusion** - concludes the present document with both a review of the work and its potential improvements in the future.

1.3 Dissertation Outline

In order to visualise and have a sense of task completion, the following figure is, essentially, a simple Gantt diagram with the tasks that this work comprised and corresponding time intervals:

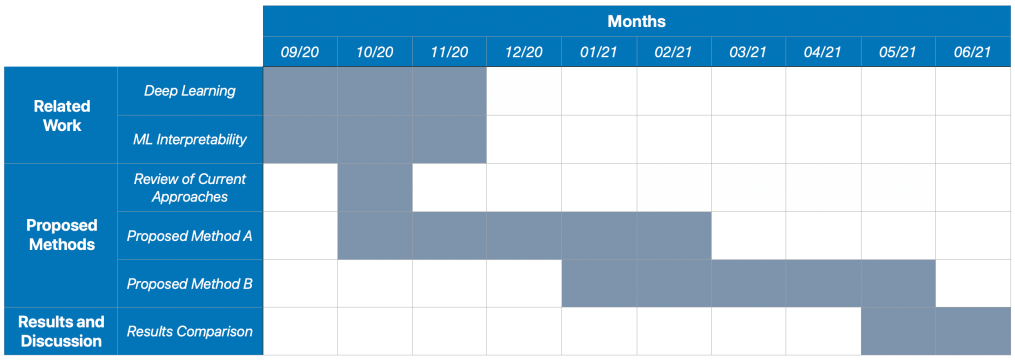

Figure 1.1: Gantt diagram with the dissertation outline. The two lengthiest tasks were, naturally, the ones that contemplated the development of the proposed methods.

Chapter 2

Related Work

The goal of the present chapter is to provide a review of the most relevant ideas and techniques in the areas of interest. To that end, the following sections are organised as follows: section 2.1 starts by describing DL, its main concepts and going over some popular architectures (like CNNs and GANs); section 2.2 follows up by compiling a list of techniques that are commonly cited when a need for interpretability arises; section 2.3 summarises the key aspects of this chapter.

2.1 Deep Learning

As a subordinate area to ML, DL focuses primarily on algorithms that mimic the human brain and how its fundamental units (i.e. neurons) receive, process, store and release information in a coordinated manner. Fortunately, the field has progressed immensely throughout the years, updating and rethinking its archetype (i.e. Artificial Neural Network (ANN)s) to perform a wide range of tasks. However, despite the groundbreaking success of these approaches, the field went through some really unremarkable periods before becoming so coveted nowadays.

The first efforts in the field of DL can be traced back to 1943, with the work of neuroscientist Walter Pitts and mathematician Warren McCulloch. Together, they proposed the simplest of elements, one that did so little but, when combined with more of its kind, could achieve a whole much greater than the sum of its parts. The artificial neuron was capable of taking inputs, assigning weights to them and lightly process them to reach an output.
More than a decade after the first steps, the next major innovation came in the form of the Perceptron [Ros57], so called by Frank Rosenblatt, its inventor. This algorithm could be used to solve linearly separable binary problems, in a supervised manner (i.e. the training samples possess ground-truth labels, against which the model's predictions are to be compared). Some relevant concepts were established, like the use of activation functions to give more expressive power. Moreover, this work started with a single layer Perceptron and naturally extended to an MultiLayer Perceptron (MLP) (i.e. multiple layers of these units). A formulation similar to this one is still used in modern architectures, whether in a feed-forward style, where connections between nodes do not form cycles, or otherwise (i.e. Recurrent Neural Network (RNN)s).
The next couple of decades marked the first really low point in the field's research interest, except for the continued work upon the ideas behind Perceptrons and the discovery of the backpropagation algorithm in 1974 (the default procedure for weight optimisation, based on the derivative of a loss function with respect to a given weight).

After this period, research saw a rejuvenated wave of interest, with breakthroughs like the formalisation of the CNN's precursor, the Neocognitron [Fuk04]. Kunihiko Fukushima described such framework, in 1980, as having two basic types of layers: convolutional and downsampling/pooling layers. Convolutional layers would serve the purpose of applying a sliding window (i.e. a kernel with learnable weights) over the input to extract features. Downsampling layers, usually in between convolutional layers, could reduce the spatial complexity of the input and, as a side benefit, give invariance to a feature's occurrence.
In 1989, Yann LeCun, one of the better known individuals in the DL community, combined a CNN with backpropagation to classify handwritten digits. Subsequent work led to the establishment of one of the most relevant ANN designs, LeNet5 [LBBH98].
Several years later, the next DL revolution was made possible by the advent of Graphics Processing Unit (GPU)s, reducing training times to much more realistic amounts. Based on that, Alex Krizhevsky, under the supervision of another DL great, Geoffrey Hinton, proposed AlexNet [KSH12] and achieved a record top-5 error in ImageNet Large Scale Visual Recognition Challenge (ILSVRC)-2012, proving that deep CNNs were feasible and well suited to the image classification task, Moreover, it showed that Rectified Linear Unit (ReLU) could be amongst the *de facto* activation functions.
Whilst the DL revolution was put in motion, areas like Image Segmentation saw an uprising of novel methods, diverging from the classical clustering or colour/edge based segmentation methods. Many of those methods became obsolete, as the state-of-the-art moved on to fully connected and region-based approaches [GLGSL18].
Another recent line of work has to do with generative algorithms, brought forward in the original GAN paper [IJGB14], co-authored by Ian Goodfellow and Yoshua Bengio, the third pioneer of DL.

Outside of the theoretical strides, practitioners and newcomers have enjoyed the success of specialised frameworks that grant more mainstream appeal to DL as a whole. Tensorflow [MAZ15] and PyTorch [PML+19] are two of those tools, with similarities and disparities that make them more suited to different target audiences. Nonetheless, both include Application Programming Interface (API)s that make the coding more user friendly.

For the purposes of this work, CNNs, GANs and Segmentation algorithms are of particular interest. Therefore, the following subsections describe those architectures both in general and, through the actual models that were chosen, in particular.

2.1.1 Convolutional Neural Networks

As mentioned above, CNNs have the ability to take a fixed-size image as input, apply complex operations and determine the corresponding class (i.e. the group it belongs to). A typical CNN architecture includes a feature extraction stage and a classification stage. The first analyses the image to keep the core aspects, while the second condensates that information into the predicted class. The image below is a visual representation of what was described in this paragraph:

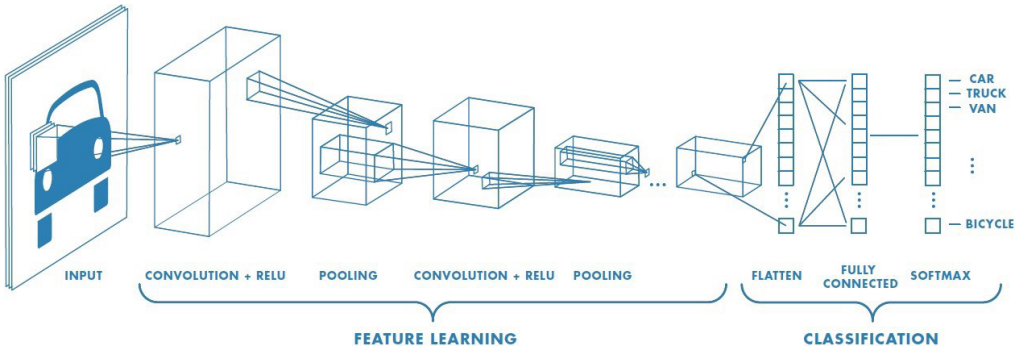

Figure 2.1: Typical CNN architecture [Pra18]. The feature extraction stage extracts as many characteristics as possible from the input image, while the classification stage gives them meaning (i.e. a class).

During the feature extraction stage, the image goes through a series of convolutional and pooling layers. A convolutional layer contains a certain number of filters/kernels of small size (e.g. 3x3 or 5x5 pixels). These filters are responsible for learning both low and high level features from the source image. The first layers usually capture lines, curves and general shapes, while the deeper ones are able to capture more and more abstract concepts. As figure 2.1 shows, a filter is a small window that slides over the input, multiplying its values (learned weights) by the ones from the input.

In-between convolutional layers, pooling layers are responsible for reducing the dimensionality of the previous convolutional layer's output (in figure 2.1, every time a pooling operation is performed, the input's height and width are reduced - the depth remains the same, being solely determined by the number of filters). This step also ensures that a feature's occurrence is spatially invariant. Just like convolutional layers, pooling layers use a kernel of size NxN, but unlike convolutional layers, they doesn't possess learnable weights. Two commonly used forms of pooling are max-pooling and average-pooling: the former only keeps the biggest value inside the chosen kernel, while the latter computes the mean of all the values captured by the kernel.

After extracting features from the input image, the final feature maps are flattened (i.e. stretched to a vector like shape) and fed to a regular MLP. An MLP, as described earlier, is a sequence of Fully Connected (FC) layers of neurons, meaning that a neuron receives the output of every neuron, in the previous layer, and its output is sent to every neuron in the next layer.

Finally, the output layer (last layer of neurons) has, typically, a neuron for each class and the values given by those neurons are the probability of the input image belonging to the classes associated with those neurons (i.e. in figure 2.1, the first neuron outputs the probability of the image belonging to the "car" class).

Within convolutions and neurons, there are special functions (called "activation functions" in the literature) that usually perform non-linear operations.

In convolutions, an example could be the ReLU function, which simply maps negative inputs to zero and behaves like the identity function for positive ones.

Neurons, on the other hand, receive several inputs and perform a weighted sum with them (once again, these weights are the parameters learned by the network). Next, the result is given to an activation function (like ReLU, Sigmoid or Softmax, as shown in figure 2.1), which determines the neuron's activation value.

These kinds of functions are crucial, without them a neuron could only process simple data, and not complex (non-linear) data, like images.

Understandably, these methods have been at the core of research efforts over the past years, not only for their ability to learn meaningful information but also due to the democratisation of GPUs. Naturally, many deep designs were proposed, including the Inception [CSR14], VGG [SZ15], ResNet [KHS15] and DenseNet [GHW18] architectures (which subsection 2.1.1.1 covers in more detail).

2.1.1.1 DenseNet

Throughout the years, CNNs got deeper and deeper, in a move that the research community saw as somewhat natural. In spite of this, the gradient based training seemed to halt this trend. That was until ResNet introduced residual connections to create information pathways between more layers, thus controlling the vanishing gradient problem. Additionally, the work with such deep networks showed how some layers could be dropped during training while not seriously affecting the overall performance. Hence, there seems to be a great deal of redundancy in deep residual networks [GHW18, HSL+16]. Based on this knowledge, the authors of DenseNet proposed an architecture similar to the following:

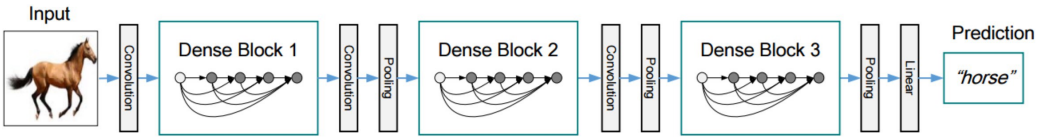

Figure 2.2: Deep DenseNet with three dense blocks [GHW18]. Each one of those blocks contains a series of convolutions, pooling and/or BN layers.

Figure 2.2 contains an example of a deep DenseNet with three blocks, each containing 5 layers. The circles represent the feature maps and the arrows send them through composite functions of operations (including BN, convolutions or pooling), which output more feature maps. Due to the many connections, each composition of operations receives a concatenation of every feature map from the previous layers. In between dense blocks, we have the so called "transition layers", which change the feature maps' sizes via convolution and pooling operations.

With this approach to network design, the authors gave the model a set of densely packed blocks to perform the feature extraction process. Naturally, many versions of the same design were created, like DenseNet-121, DenseNet-161 or DenseNet-201 (where the number is analogous to the number of layers that were included).

2.1.2 Region-based Instance Segmentation

The basic idea behind image segmentation comes from the definition of the word ”segmentation”: division into separate parts/sections. With effect, when an algorithm attempts to segment an image, it tries to determine the object that the pixels belong to. Notice the wording here, because, unlike object detection, a segmentation algorithm does not need to know what the objects and visual elements represent (i.e. the classes).

But before we discuss instance segmentation, we should create a notion for object detection first.

2.1.2.1 R-CNN

The region-based family of object detection algorithms relies on the proposal of Region(s) of Interest (RoI) in any given image. A naive approach to this task would be to consider all possible bounding boxes, with varying dimensions, aspect ratios and positions. Unfortunately, this algorithm would rapidly become prohibitive in terms of the computational budget. So, to keep the same idea but make it more efficient, the R-CNN paper [GDDM14] used selective search [USGS13] to consider just a few thousand region proposals. The following list and figure 2.3 provide more details on how the proposals are derived:

1. Start with an initial set of seemingly random segmentation regions.
2. Group neighbouring regions together, based on a number of similarity metrics (colour, texture or size).
3. Repeat the previous step until the stopping criterion is reached (e.g. maximum number of iterations).

Figure 2.3: Visualisation of how the selective search algorithm progresses [USGS13]. The top row contains the segmented regions, while the bottom row contains the corresponding bounding boxes. The algorithm works by joining similar regions together.

Next, the region proposals are resized to a square aspect ratio and fed to a CNN that acts as a feature extractor. Finally, for each region proposal, a trained SVM receives the corresponding feature maps (which are flattened to become a 4096-dimensional vector) and detects the presence of any object of the class it was trained on:

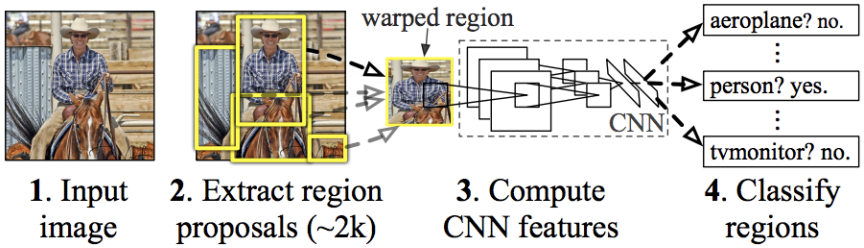

Figure 2.4: R-CNN pipeline [GDDM14]. The framework uses selective search to create region proposals, which are later fed to a CNN that extracts features. Then, a trained SVM classifies the feature maps as describing a known object or not.

2.1.2.2 Fast R-CNN

After the publication of the R-CNN paper, one of the authors improved the architecture with Fast R-CNN [Gir15], a much faster and accurate approach. This new version still relies on selective search to generate region proposals but includes some key improvements.

Once the region proposals (i.e. RoI) are generated, the image is fed to a CNN responsible for extracting a single set of feature maps. Then, each RoI is projected in the common feature maps, effectively slicing it into smaller feature maps. Next, a new layer called Region(s) of Interest Pooling (RoIPool) computes a fixed-length vector for each RoI: it applies max-pooling to each slice of feature maps, ensuring a pre-defined HxW size, thus creating a small feature map for every RoI that still relies on shared computations. Here we start to see the gains in speed and space by only extracting features once. The pipeline continues by mapping the smaller, RoI specific, feature maps to a feature vector, using FC layers. Finally, the architecture bifurcates into two branches with additional FC layers: one serves the classification purposes and has a "$K + 1$"-way Softmax, with probabilities for each of the K classes and the "background" class; and another performs regression by outputting four values for each class, representing the bounding box corners:

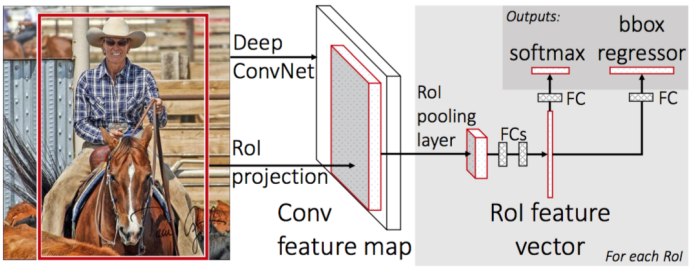

Figure 2.5: Fast R-CNN pipeline [Gir15]. The revision of R-CNN uses a common feature map, shared by all region proposals, instead of extracting features for each and every proposal.

2.1.2.3 Faster R-CNN

Making further improvements, the Faster R-CNN [RHGS16] architecture extends the Fast R-CNN predecessor by replacing the mechanism that proposes the object regions. Faster inference times and accuracy are, once again, made possible by some rethinking of the core aspects. Two modules serve as the basis for this unified framework: RPN and Fast R-CNN detector. The following figure contains a good visual approximation of the Faster R-CNN architecture:

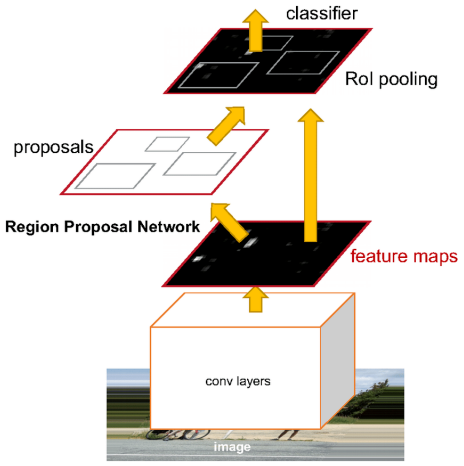

Figure 2.6: Faster R-CNN pipeline [RHGS16]. The two main modules are the RPN, to predict where an object might be, and the R-CNN detector, to classify the proposed regions.

The advantages brought forward by the RPN include its speed and the ability to be tuned to a specific task (i.e. because it is a trainable component, it can become better suited to the types of images and biases in the training set, unlike generic algorithms like selective search). The RPN works by using k anchor boxes (which are, essentially, matrices of varying aspect ratios) that slide over the image's feature maps, followed by an intermediate layer that produces a fixed-length vector (e.g. 256-dimensional) for each anchor. This vector is then given to two separate FC branches. On one hand, the first branch outputs $2k$ scores (two for each of the k anchors) which are the probabilities of an object's presence in each proposal. On the other hand, the second branch outputs four bounding box coordinates for each of those k anchors:

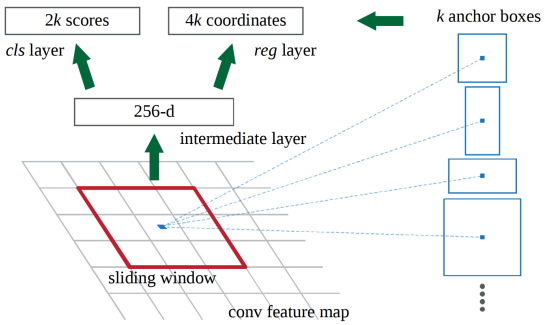

Figure 2.7: Visualisation of the proposed RPN [RHGS16]. By using multiple anchors with different aspect ratios and sliding them over the received feature maps, the network is able to extract several fixed-length vectors, which are later used for classification and regression.

With the region proposals and the image feature maps (the same used by the RPN module), the Fast R-CNN detector can find the slice of feature maps that correspond to each region proposal and pass it through the RoIPool layer. Then, just like before, a RoI feature vector is computed using a couple of FC layers and given to two sibling branches, which output a class and bounding box.

With the architecture assembled, all that is needed is a way to train it. To do so, the authors propose a 4-step training procedure:

1. The RPN is initialised with ImageNet weights and fine-tuned to perform the region proposal task.
2. With the generated proposals from step 1, the Fast R-CNN detector is also fine-tuned with ImageNet weights.
3. The convolutional layers that extract features from the original image are initialised with the Fast R-CNN weights (remember that these layers are shared across RPN and Fast R-CNN). The remaining unique layers of the RPN are fine-tuned.
4. Finally, keeping the shared convolutional layers fixed (from step 3), the unique layers of Fast R-CNN are fine-tuned, resulting in a trained and unified framework.

Faster R-CNN achieved state-of-the-art performance in popular datasets like Pascal Visual Object Classes (VOC) and Common Objects in COntext (COCO), remaining to this day as a very influential design.

2.1.2.4 Mask R-CNN

Specifically designed to produce segmentation masks, the Mask R-CNN framework [LSD15] is an extension of the previously discussed Faster R-CNN. It adds a third branch to the already existing classification and bounding box regression ones. The mask branch is a small Fully Convolutional Network (FCN) applied to each RoI, predicting a binary mask:

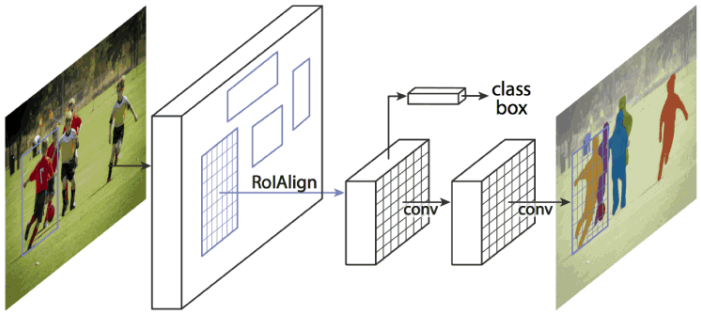

Figure 2.8: Mask R-CNN architecture [LSD15]. Built on top of Faster R-CNN, this architecture favours segmentation and includes alignment techniques to ensure maximum mask quality.

By building on top of the Faster R-CNN architecture, Mask R-CNN adds very little extra overhead and keeps the speed that the former enjoyed. The Region(s) of Interest Align (RoIAlign) operation (depicted in figure 2.8), which fixes the RoI misalignment

present in the previous R-CNN based methods, also helped the object detection performance (i.e. if the mask prediction branch is disabled, the resulting architecture behaves better than the baseline Faster R-CNN). The reason for the misalignment in the original frameworks has to do with the approximations that are made with RoIPool, resulting in the loss of some spatial information. RoIAlign, on the other hand, does not use the exact values in the feature maps, but rather, interpolations that take into account the surrounding values. By doing so, we are using some of the information that would have been discarded by a quantisation based approach like RoIPool. Only after doing this, can the max-pooling operation be used to obtain a fixed-size feature map (the maximum value still has some information about its surroundings).

Mask R-CNN achieved, at the time, state-of-the-art results on the COCO dataset and was even used to perform pose estimation, despite not being intentionally trained to do so.

2.1.3 Generative Adversarial Networks

As generative algorithms, GANs possess the ability to create new data, unlocking many interesting possibilities. Although seemingly complex, the motivation behind these frameworks is really intuitive: two networks with competing goals are put against each other to make for a better learning process (if one gets better, the other has to keep up).

Formally, a generator G tries to generate new data from random noise. Then, synthetic and real samples are given to a discriminator D, whose task is to distinguish their sources:

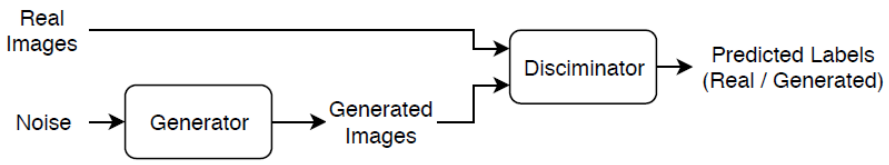

Figure 2.9: Typical GAN architecture [Mat20]. The generator takes random noise as input and outputs fake, but realistic images. The discriminator should not be able to distinguish those images from the real ones.

Analysing figure 2.9, one can understand the training process that much better: G will learn to sample a random input vector and produce a synthetic image, so that it can fool D into thinking it came from the real distribution. Conversely, D will learn to distinguish fake samples from real ones. At first, G's images will look nothing like the real ones, but the feedback loop will drive the two adversaries to a converged state: G will be updated by how far it was from fooling D and D's update will be based on how it fared at discriminating between real and fake samples. After training, the discriminator is discarded and we are left with a generator that can produce realistic images.

Equations 2.1 and 2.2 present the formal loss functions that enable the training of both D and G. It should be noted that, in the original description, D will try to maximise the average of the log probability for real images, as well as, the log of the inverse probability for fake images. G, on the other hand, should seek to minimise the log of the inverse

probability predicted by the discriminator for fake images:

$$\mathcal{L}_D = \frac{1}{m}\sum_{i=1}^{m}[\log(D(x_i)) + \log(1 - D(G(z_i)))] \quad \textbf{(2.1)}$$

$$\mathcal{L}_G = \frac{1}{m}\sum_{i=1}^{m}\log(1 - D(G(z_i))) \quad \textbf{(2.2)}$$

Since the original publication, the loss terms have been update, whether by making G maximise the log of the discriminator probabilities for fake images or rethinking core aspects of the training procedure (as in Wassertein GANs [MAB17], where the discriminator is replaced with a critic that scores how realistic the given samples look). Due to the high interest shown by the research community, many different designs were proposed, including AttGAN [HZK+18], for image manipulation, and StyleGAN2 [KLA+20], for high quality image generation. Given its use in this work, StyleGAN2 is highlighted below.

2.1.3.1 StyleGAN2

The StyleGAN2 team started experimenting with a concept called "progressive growing". In the paper where they introduce this idea [KALL18], the authors postulate that the discriminator D and generator G could start working with low resolution images (e.g. 4x4 pixels) and then get progressively upgraded to deal with higher and higher resolutions. Both D and G would get new layers, at each resolution change, to deal with the higher dimensionality. To avoid the risk of sudden shocks or instability, the new layers are faded in smoothly and every layer remains trainable throughout the entire training process. This methodology allows both the D and G to first learn the large-scale aspects of the images and then, progressively, shift their attention to finer details, which usually come with higher resolutions. The resulting network, called ProgGAN, was able to generate high quality images at higher than usual resolutions (e.g. 1024x1024 pixels).

Following the work above, the same authors proposed an architecture called StyleGAN. The main differences between this architecture and the baseline ProgGAN are the use of an additional mapping network to map latent codes to a new latent space, the use of style vectors and the introduction of a technique called Adaptive Instance Normalisation (AdaIN). The mapping network opposes itself to the traditional GAN formulation, where we map a point in the latent space to the image space. In this new configuration, we first map the latent points to an intermediate space, through an 8-layer MLP. This intermediate space W is, according to the authors, more disentangled than the former space Z, enabling better manipulation of attributes. Once we have the new latent codes, we apply an affine transformation to create style vectors, which condition the image generation process. As for the AdaIN technique, it helps to incorporate the aforementioned styles with the feature maps that come from the successive convolution layers. There is also the use of noise vectors, which provide more variation in finer details (i.e. less smoothing):

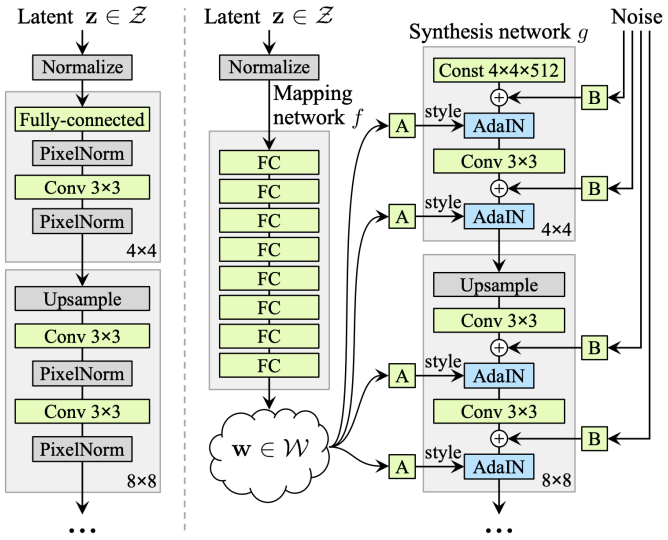

Figure 2.10: StyleGAN architecture [KLA19]. On the left, we have the ProgGAN architecture, while on the right we have the proposed StyleGAN version with the mapping network and style injection.

After the release of the original StyleGAN, the same team corrected some of the visual artefacts that it tended to include in the generated images (i.e. "water droplets" and misaligned eyes or teeth). The StyleGAN2 framework [KLA+20], as it ended up being called, redesigned the way style vectors are incorporated into the generation process, due to a belief that the AdaIN operation limited the generator's ability to use the maximum amount of information. In fact, the "water droplet" effect results from the generator attempting to sneak some information past the normalisation step. Furthermore, the revised architecture replaced progressive growing with input/output skip connections (for the generator) and residual connections (for the discriminator):

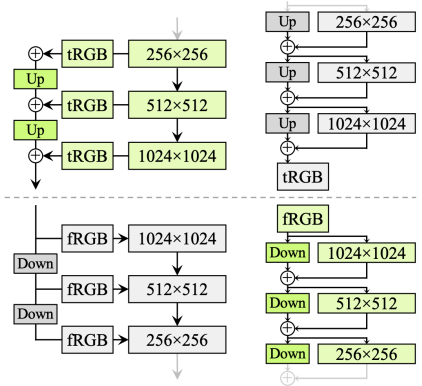

Figure 2.11: Proposed revisions of the networks' architectures [KLA+20]. On the left, we have input/output skip connections and, on the right, residual style connections. After some experiments, the authors settled on a skip generator (top-left) combined with a residual discriminator (bottom-right).

The improved architecture allowed both networks to have access to every resolution layer from the beginning, but only make use of them if it proved beneficial during training. This revision, along with several others, improved what was already a state-of-the-art design, elevating the quality of synthetic imagery even further.

2.2 Machine Learning Interpretability

As an example of what this topic is all about, let us imagine a binary classifier whose task would be to distinguish between wolves and huskies [Mol19]. After training, it ends up misclassifying a number of huskies as wolves. Interpretability techniques could, perhaps, show a tendency for the classifier to use snow, in some of the images, as a feature for the class ”wolf”. Without these techniques, it would be decidedly harder to reach the same conclusion (it could perhaps become an untraceable behaviour). Thus, the adoption of interpretable components can bridge the gap between humans and ML models, increasing the level of trust between the two parties.

In [Lip17], the author highlights the different degrees of interpretability in ML models. We might link the word ”interpretable” to the model's architecture, meaning that it can be easily understood, even if that simple nature leads to a deficiency in capacity. We might also consider the model's parameters as an interpretable component. There seem to be many possible (and equally valid) forms of interpretability. Therefore, authors in this field have debated on the most accurate taxonomy to describe the scope and depth of what it means to be an interpretable system.

The proposed criteria [Mol19, Lip17] classifies ML Interpretability in terms of depth, scope and model applicability:

- **Intrinsic or Post hoc**: are we reducing the model's complexity to make it more interpretable? Or are we allowing the model to be complex and only explaining its output?
- **Local or Global**: are we explaining individual predictions? Or the behaviour of the entire model?
- **Model-specific or Model-agnostic**: are the interpretation techniques specific to a limited range of models (i.e. only work with them)? Or are they generic enough to be paired with practically any kind of ML model?

Following the above criteria, we argue that our approaches (chapter 3) are Post hoc, Local and Model-agnostic. In other words, they allow model complexity, explain single instances of data and can be paired with several types of ML and/or DL models (in specific, different types of classifiers, generative models and instance segmentation architectures).

As a natural extension to this introduction, the next subsections attempt to showcase the most common techniques found in the literature. These techniques have ranging levels of complexity and applicability to the problem of periocular recognition. Some of them rely on images, while other make use of plots. Nevertheless, they share a common goal: remove the mist that conceals the reasoning of complex ML models.

2.2.1 Partial Dependence Plot

A PDP is, as the name suggests, a plot relating one or two features with a target variable [Fri01]. Ideally, the features have to be independent in order to properly explain the behaviour of another, dependent variable (i.e. target). The main purpose of such technique is to determine the nature of feature-target relationships (e.g. linear or more complex). Additionally, PDP is a global interpretability method, meaning that it can capture the behaviour of the entire dataset to produce a global relationship between features and targets. From this point forward (including subsection 2.2.2), the features we want to explain will be represented by x_S and the remaining ones by x_C.

As an example, let us consider a simple, linear regression model. In this setting, the Partial Dependence Function, that would allow us to create a plot, has the form:

$$\hat{f}_{x_S}(x_S) = \int_{x_C} f(x_S, x_C)\, \mathbb{P}(x_C) dx_C \quad (2.3)$$

The formulation above shows that the PDP values for the features in x_S are the result of marginalising the model f over the distribution of the features present in x_C. By doing so, a new model $\hat{f}$ is obtained, depending exclusively on the features in x_S. To approximate the indefinite integral, one can take a Monte Carlo approach and average the prediction of instances in the training set (with length N), while fixating the values of x_S:

$$\hat{f}_{x_S}(x_S) = \frac{1}{N} \sum_{i=1}^{N} f(x_S, x_C^{(i)}) \quad (2.4)$$

It becomes evident, then, that the basic principle of PDP is the lack of correlation between the features in x_S and the ones in x_C, which is usually not the case.
Suppose we only have weight (x_S) and height (x_C) as features. If a given value of x_S is fixated, all the other possible values for x_C, observed in the training data, will be paired with that specific value of x_S. In many cases, unlikely combinations will appear, like an instance where a really small weight will be paired with a really high height. It simply comes down to the fact that these two features are heavily dependent on each other. Unfortunately, such limitation limits PDP's applicability in a plethora of cases.

In spite of this, in situations where the chosen features are independent, this technique can even be used for classification problems, in which case the ML model outputs class probabilities. Consequently, a PDP would display the class, as the dependent variable, and one or two features as the independent ones. To get a broader perspective, one could simply create a PDP per class.

To better understand and visualise this concept, figure 2.12 shows three different variables (features) being individually plotted against the target variable (in this case, the number of bike rentals on any given day). Note that here we are using the model $\hat{f}$, which only requires one input variable at a time, given that the remaining variables were marginalised:

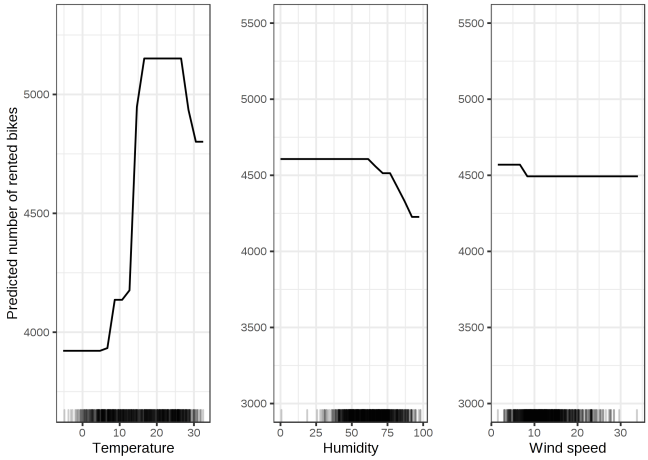

Figure 2.12: Three PDPs from a regression model with three independent variables and one, dependent variable [Mol19]. The predicted values come from a marginalised model ($\hat{f}$) that only relies on one feature.

As an interpretable technique, PDP works well if its conditions are met (variable independence and need for only one or two features per plot). However, when features start getting dependent on each other, PDP can produce unreliable results [Mol19].

Considering the purposes of this work, this method will not suffice for two main reasons. Firstly, our attributes are, intuitively, entangled: blue eyed people tend to have lighter skin tones, as opposed to darker skin tones, which often accompany darker iris shades. Feature independence is much more theoretical and usually present in toy problems. Secondly, it is simply not enough to display just one or two features at a time.

2.2.2 Accumulated Local Effects

Another technique that produces plots is ALE, commonly cited as an alternative to PDP, particularly due to the latter's unreliability when the features are correlated [AZ19]. Before getting into the formulas, it is important to visualise ALE's reasoning:

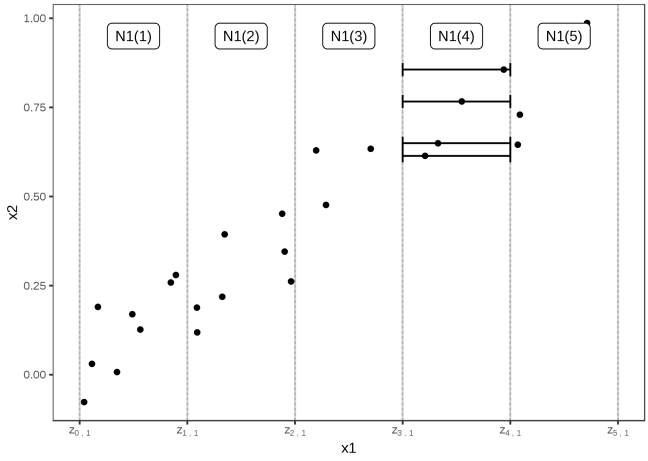

Figure 2.13: Calculation of ALE for feature x_1, strongly correlated with feature x_2 [Mol19]. The distribution in divided into intervals and, for each one, we determine the difference in predictions when feature x_1 takes on the values of the lower and upper limits. These results are later accumulated (i.e. summed) and centred.

From the figure above, we get the idea that ALE evaluates differences in predictions at a smaller scale (i.e. the intervals) and then sums everything to get a broader perspective on how the model f behaves. Indeed, that is what the next formula describes:

$$\hat{f}_{x_S}(x_S) = \int_{z_{0,1}}^{x_S} \int_{x_C} \frac{\partial f(z_S, x_C)}{\partial z_S} \mathbb{P}(x_C|z_S) dx_C dz_C - constant \quad (2.5)$$

In equation 2.5, ALE also tries to produce a new model that only requires the features in x_S. The difference with regards to PDP comes from the fact that we do not use all the possible values for the features in x_C, but rather, the values that are possible in the range that our specific feature values belong to ($\mathbb{P}(x_C|z_S)$). By using a conditional distribution, we eliminate the unlikely combinations that PDP would have considered. Then, the leftmost integral applies the same reasoning to the entire range of possible values for the features that x_S contains. Finally, a constant is subtracted to ensure that, in the ALE plot, the average effect is zero.

As an example, the next couple of formulas are going to describe how the ALE values are calculated when we want to use just one feature (this method is broad enough to work for two features at a time and even for categorical features).
Just like with PDP, in practice we approximate the theoretical definition, and, in this case, that process is done with the following formula:

$$\hat{f}_j(x) = \sum_{k=1}^{k_j(x)} \frac{1}{|N_j(k)|} \sum_{i:x_j^{(i)} \in N_j(k)} [f(z_{k,j}, x_{\setminus j}^{(i)}) - f(z_{k-1,j}, x_{\setminus j}^{(i)})] \quad (2.6)$$

Starting from the right, in $[f(z_{k,j}, x_{\setminus j}^{(i)}) - f(z_{k-1,j}, x_{\setminus j}^{(i)})]$, we are interested in determining the difference in predictions when we replace the value of the feature j we want to explain with the values from the upper ($z_{k,j}$) and lower ($z_{k-1,j}$) limits of a given interval. The summation $\sum_{i:x_j^{(i)} \in N_j(k)}$ allows us to repeat the aforementioned process for every instance whose feature j belongs to a neighbourhood given by $N_j(k)$. If we recall figure 2.13, these neighbourhoods are the vertical slices that divide feature j's distribution. Then, to average the differences we divide the summation by the number of instances in the given interval ($|N_j(k)|$). This part of the formula computes the local effect of feature j.
Following the above steps, the leftmost summation is responsible for computing and, more importantly, accumulating the local effects of every interval (thus giving meaning to the acronym ALE). Finally, we subtract the average to the resulting values, ensuring that they stay centred around zero:

$$\hat{f}_j(x) = \hat{f}_j(x) - \frac{1}{n} \sum_{i=1}^{n} \hat{f}_j(x_j^{(i)}) \quad (2.7)$$

Figure 2.14 depicts an ALE plot of the problem seen in subsection 2.2.1. This example shows that the plots are indeed centred around zero, with positive values indicating an increase in the model's prediction and negative values indicating a decrease, when com-

pared to the average prediction. Additionally, it becomes clear how shaky ALE plots can be, especially when compared with the PDP version (as a side note, the number of intervals contributes to this effect):

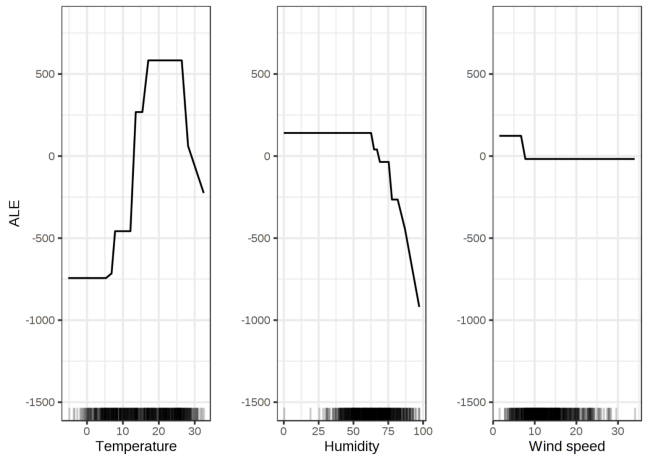

Figure 2.14: Three ALE plots from a regression model with three independent variables and one, dependent variable [Mol19]. As per the trained model, the temperature and humidity's influence outweights that of the wind speed.

Overall, ALE plots are considered a superior technique to PDPs, despite having some drawbacks too: as mentioned before, the shaky lines can become undesirable and the implementation of this method is more complex. In spite of this, it is generally accepted that, when we are working with independent variables and require fast computing times, PDP is the preferred option. For virtually any other scenario, ALE is regarded as the superior solution.

Just like PDP, and despite being better for the specific use case described in the present document, the visualisation of only a couple features limits ALE's applicability. The following subsections will describe other methods that come closer to what is expected from a framework that can, visually, explain its decisions.

2.2.3 Occlusion Map

Being one of the most straightforward techniques, Occlusion Maps try to make perturbations on certain areas of an input image and register how the model (usually a classifier, like a CNN) reacts to those changes.

In practice, this method involves taking a square, of fixed size and colour, and sliding it over the image, iteratively. Then, the perturbed images are fed to the classifier and its scores are noted. Finally, the scores are translated into a heat map, indicating the portions of the image where an occlusion had the most impact (i.e. the areas most used by the model to predict the correct class).

Figure 2.15 shows a clear example of what kind of results can be expected with this tech-

nique. Another example can be found in [ZF13] and figure 2.16, where the authors went even further by visualising feature maps and noticing how the classifier would attribute different (wrong) classes when certain parts of the image get occluded:

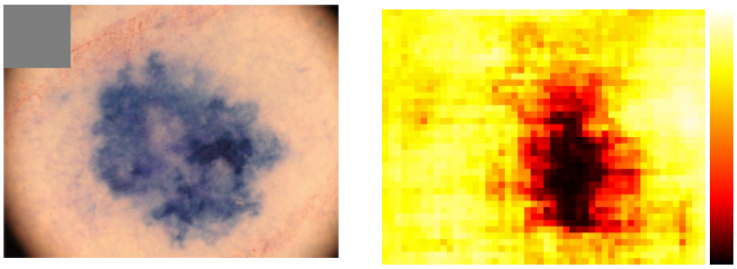

Figure 2.15: Example of a heat map generated with the Occlusion Map technique [She18]. Here, the model clearly identified the melanoma as a crucial element.

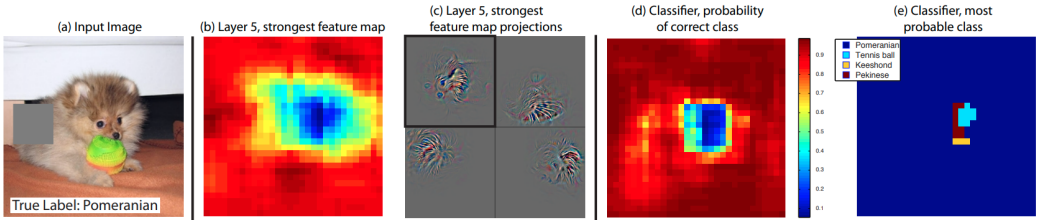

Figure 2.16: Another example of Occlusion Maps being used to interpret a model's decisions [ZF13]. The rightmost figure shows how the predicted class varies with respect to occlusions.

As seen above, Occlusion Maps are able to locate areas of an image that contribute the most to a classifier's decision. As an additional benefit, they are also easy to implement and require almost no redesign or retraining of the classifier.

The applicability of this technique, however, is somewhat limited, given the need for a higher level explanation (i.e. the ability to highlight well defined areas of the periocular region, like the iris or the eyebrow).

2.2.4 Saliency Map

Described in [SVZ13], this technique also attempts to highlight certain parts of the input image with brighter or darker shades, based on their importance to the model's score. The authors propose a simple example to start with, similar to the following:

$$S_c(I) = w_c^T I + b_c \quad (2.8)$$

Equation 2.8 comes from a linear score model for class c and, as standard, there is a weight vector w, which is later transposed, and a bias b. In this setup, each pixel of image I is weighted according to its importance. Translating such formulation to a CNN becomes more complicated. In spite of this, one can approximate that value by computing the first-order Taylor expansion:

$$S_c(I) \approx w^T I + b \quad (2.9)$$

In this case, w is the derivative of S_c with respect to a specific image I_0:

$$w = \frac{\partial S_c}{\partial I}\bigg|_{I_0} \quad (2.10)$$

One can interpret equation 2.10 as obtaining an image-specific class saliency where the magnitude of the derivative indicates the pixels that need to be changed the least to change the final score the most.
Having this foundation, the authors continued the Saliency Map extraction process. Remembering that any image (i.e. I_0) has m rows and n columns, a class Saliency Map belongs to R^{m*n}. The first step is to find w, as per equation 2.10 through back-propagation. Then, if I_0 is a greyscale image, w has exactly one element for each pixel, meaning the Saliency Map could be calculated as $M_{ij} = |w_{h(i,j)}|$, where $h(i,j)$ is the index of the element in w directly corresponding to I_0's pixel in row i and column j. As for Red Green Blue (RGB) images, with multiple depth channels, w has more elements to account for the added depth (in this case, an index takes the form $h(i,j,c)$). Thus, to ensure a single saliency value is derived, only the maximum magnitude across all channels is kept: $M_{ij} = max_c|w_{h(i,j,c)}|$.
Figure 2.17 has some examples extracted directly from the original source. It should be note that the three Saliency Maps shown are from the highest scoring class on pseudo-randomly selected ILSVRC-2013 test images:

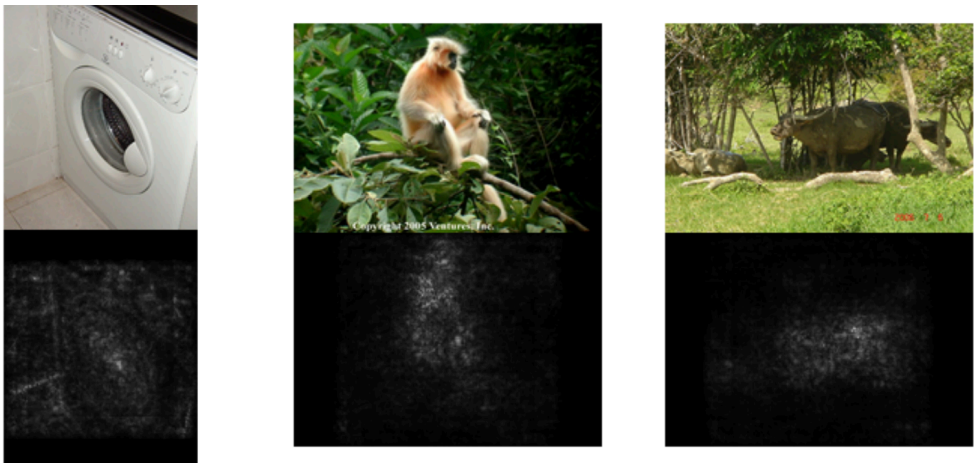

Figure 2.17: Three Saliency Maps (below) extracted from each of the input images (above) [SVZ13]. The pixels in whiter tones are the most significant to the classes that were predicted, meaning that changes to them could impact the output class.

One advantage of Saliency Maps is that they are not expensive to compute, only requiring a single back-propagation pass, and do not assume the existence of any additional annotations (apart from the labels used when training the original model).

The ability to locate the most relevant components of an image fits perfectly into the desirable output of a periocular recognition system (not to mention the higher finesse, when compared with Occlusion Maps). Therefore, section 3.1 presents some results that make use of Saliency Maps to achieve satisfactory levels of interpretability.

2.2.5 Local Interpretable Model-Agnostic Explanations

One of the most widely adopted techniques is LIME, originally proposed in [RSG16]. This method relies on surrogate (i.e. auxiliary) models to, locally, explain the behaviour of a much more complex, black box model.
In order to create a basic understanding of how LIME works, the authors propose we forget about the training data and assume we only have access to the trained black box model itself, to which samples can be fed and predictions can be withdrawn. LIME's approach starts by taking a sample and creating variations from it. With this process, a new dataset is created and these samples can be fed to the original, black-box model. Having the dataset and the corresponding predictions, LIME moves on to the next phase, where a much simpler model (often linear) is trained with the aforementioned data. The linear model is, however, weighted, meaning that it gives more importance to samples that are closer to the one being explained (thus forcing locality). The learned model should be a good approximation of the original one, albeit locally (the so called local fidelity [Mol19]).

In broader terms, an explanation can be defined as follows:

$$explanation(x) = \arg\min_{g \in G} L(f, g, \pi_x) + \Omega(g) \quad \textbf{(2.11)}$$

Equation 2.11 shows how an explanation for an instance x is the summation of a loss component L (e.g. Mean Squared Error (MSE)) and a complexity component Ω. L measures how the predictions from the surrogate model g and the original model f compare, considering the neighbourhood given by π_x. Ω controls the degree of complexity of the surrogate model (preferably low). It should be noted that, in practice, LIME optimises the former term, with the latter being left to the user's responsibility.

One interesting characteristic of this method is that it works for tabular, text or image data. For each of them, LIME tries to create the previously mentioned variations by changing certain aspects of the raw data. With tabular data, it varies each feature individually, helped by a normal distribution with mean and standard deviation in accordance to that feature. As for text and images, LIME turns words or super-pixels (contiguous patches of neighbouring pixels) on or off.
Figure 2.18 presents an example on how LIME obtains interpretable explanations:

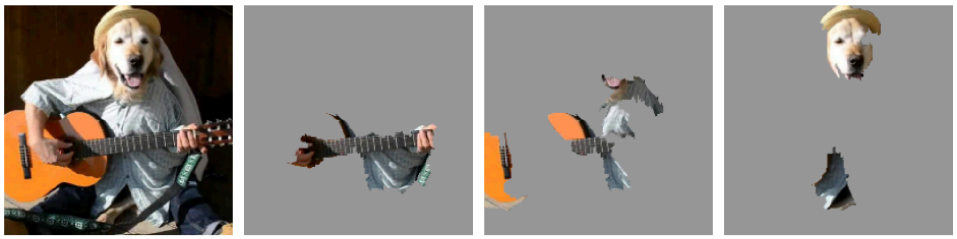

Figure 2.18: Inception's top-3 prediction ("electric guitar", "acoustic guitar" and "labrador" respectively) explained using LIME [RSG16]. The highest contributing super-pixels were kept, while the remaining ones were greyed out.

LIME's disadvantages include the inherent limitations of linear models, which can be incapable of modelling a complex decision boundary, even if it is local and at a smaller scale. Despite the drawbacks, this technique is certainly capable of providing a hint of interpretability to otherwise opaque models.

Considering LIME is popular and effective, section 3.1 includes a perspective on how it could be applied to the present work.

2.2.6 Anchors

Following their work with LIME, the authors proposed another approach in [RSG18], whose goal is to create decision rules in the form IF-THEN, such that a prediction is sufficiently anchored by some features (i.e. if changes in other feature values do not change the prediction). The techniques employed to that end include reinforcement learning and graph search.

Just like LIME, this method generates perturbations on real instances to create local explanations. Simple IF-THEN rules are used to explain local behaviours, instead of surrogate models. Additionally, the notion of coverage is introduced to specify the amount of instances a rule applies to (i.e. they are expected to be reusable). Figure 2.19 provides a visualisation of both anchors and LIME:

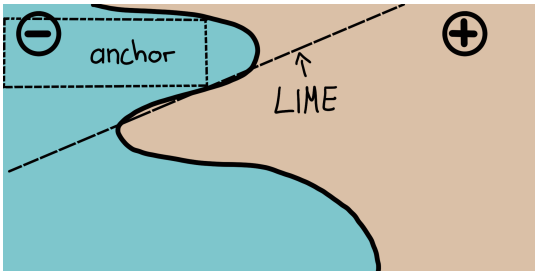

Figure 2.19: A comparison between LIME and anchors [RSG18]. One can see that, while LIME tries its best to approximate the decision boundary's behaviour, an anchor provides a more realistic result.

The form of an anchor is usually:

IF (feature$_1$ == value$_1$ AND/OR feature$_2$ == value$_2$ AND/OR ...) THEN
PREDICT target = value$_3$
WITH PRECISION ...% AND COVERAGE ...%

The example above shows the readability provided by an anchor, which specifies the features that contributed the most to a prediction, the original model's prediction, the level of precision (i.e. accuracy) shown by the anchor and how applicable it is to the perturbation space's instances.
In more formal terms, an anchor A must satisfy a given level of precision (τ) such that

$precision(A) \geqslant \tau$. To determine an anchor's precision, we can use the following equation:

$$precision(A) = \mathbb{E}_{D(z|A)}[\mathbb{1}_{f(x)=f(z)}] \quad (2.12)$$

Analysing equation 2.12, z stands for the perturbed neighbours of x to which A is applicable, D is the distribution of perturbed instances and $\mathbb{1}_{f(x)=f(z)}$ denotes the black box model's predictions with respect to x and z (expectably, the same). In practice, it is intractable to determine adequate anchors using equation 2.12. To solve such issue, the authors propose the introduction of a new parameter (referred to as δ) such that $0 \leqslant \delta \leqslant 1$, effectively creating a probabilistic definition:

$$P(precision(A) \geqslant \tau) \geqslant 1 - \delta \quad (2.13)$$

Furthermore, the notion of coverage, intuitively explained as the need for rules that are applicable to a large portion of D, can also be described with an equation:

$$coverage(A) = \mathbb{E}_{D_{(z)}}[A(z)] \quad (2.14)$$

Maximising coverage is desirable, given that the generated anchors should be reusable on a decently sized portion of the perturbation space:

$$\max_{As.t.P(precision(A)\geqslant\tau)\geqslant 1-\delta} coverage(A) \quad (2.15)$$

From all the equations shown, it becomes clear that this process tries to find anchors with the highest coverage, assuming they satisfy the precision constraint (2.13). One interesting aspect is that, rules with more predicates (i.e. conditions in the IF branch) have a tendency for higher precision. However, such characteristic is not to be pushed to extreme cases. A rule that has too many predicates may be overly tuned to predict the instance given (x) and none other (or a really small amount of similar instances). There is, then, a trade-off between precision and coverage.

In addition to the formulations described above, this technique relies on the following steps:

1. Candidate anchors are generated in rounds. The first round is responsible for creating one candidate per feature of x.
2. The best performing candidates move to the next round, where they are expanded to explain yet another feature of x.
3. This iterative process eventually stops when another routine determines we have an anchor good enough to satisfy equation 2.13.

When an anchor is said to perform better than the remaining ones in any given round, a step was used involving the Multi-Armed-Bandit formulation, where each arm is a candidate rule and a classical problem of exploration and exploitation arises. This type of

setting helps speed up the search for an optimal rule, at any given moment.

In terms of examples, the original paper describes the process of obtaining anchors for text and image classification purposes (among other tasks), showing the versatility of this method. Text data is perturbed by omitting certain words and replacing them with pseudo-random ones (following some rules that guarantee coherent replacement words). As for images, instead of turning super-pixels on or off (as was the case with LIME), the active ones are kept and superimposed over an unrelated image, to determine how the black box model handles them:

Figure 2.20: Anchor explanations for the class "beagle" [RSG18]. In the rightmost figures, we can visualise how this technique superimposes the active super-pixels over unrelated samples to mislead a CNN.

Anchors are intuitive and easy to understand, delivering a performance level close to LIME's. Section 3.1 includes some preliminary results with LIME, instead of anchors, mainly due to the fact that there is an official implementation of LIME for image data, but not for anchors.

2.2.7 SHapley Additive exPlanations

SHAP, first introduced in [LL17], has its foundation laid upon Shapley values. Those values come from cooperative game theory, in which features are seen as players inside a cooperative setting, where they can choose to form coalitions (cooperative parties) to maximise future gains.
As seen in the original source, to calculate the Shapley value for feature/player i, one can use the following formula:

$$\phi_i = \sum_{S \subseteq F \setminus \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} [f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)] \quad (2.16)$$

Breaking down equation 2.16 step by step, we consider F to be the set containing every feature in our problem and S is some subset of F that does not contain i. Additionally, x is the instance we seek to explain and f is a version of the black box model that considers n features (for example, $f_{S \cup \{i\}}$ is a model that was trained to used all features in S and i, but f_S was trained exclusively with S). Hence, this method requires retraining f for every possible coalition of the features present in F.

Starting from the right, with $[f_{S\cup\{i\}}(x_{S\cup\{i\}})-f_S(x_S)]$ we are computing the marginal value of adding player i to subset S. In other words, we have a coalition where i was not originally present (S), allowing us to perform a prediction using f_S on x_S. The obtained prediction is our base value (i.e. one where i has no influence whatsoever). Then, if we train a version of the black box model f that uses i ($f_{S\cup\{i\}}$) and give it an instance $x_{S\cup\{i\}}$, we will get a prediction where feature i was considered. Therefore, by subtracting this latter value by the former base value, we determine the marginal influence of feature i in the game and feature subset S.
Moving to the left, the fraction $\frac{|S|!(|F|-|S|-1)!}{|F|!}$ is responsible for averaging out the effect of every other feature in S that is not i. Imagining that $|S| = 3$ and $|F| = 5$, we would have $\frac{3!(5-3-1)!}{5!} = \frac{1}{20}$. Hence, every possible subset S with three elements, derived from a set F with five elements, contributes by $\frac{1}{20}$. This factor will then rescale the influence of i in a specific subset S. By doing so, we are marginalising the actual composition of S, thus isolating the effect of feature i.
Finally, the summation repeats the same process for every possible subset S, essentially approximating the real Shapley value of feature i.

With the theoretical foundation provided above, one can understand SHAP that much better. As the authors mention, SHAP borrows from LIME and Shapley values to build a better technique. Similarly to LIME, SHAP uses an auxiliary (linear) model to aid in the search for explanations. One interesting aspect of SHAP, are the several branches that build upon it, like KernelSHAP [Mol19], whose details will be described below.

Being based on a linear model that requires a training stage, the first step to understanding KernelSHAP is to analyse its loss function:

$$L(f, g, \pi_x) = \sum_{z'\in Z} [f(h_x(z')) - g(z')]^2 \pi_x(z') \quad (2.17)$$

The main aspects of equation 2.17 are f (the original black box model), g (the surrogate model) and π_x (the SHAP kernel). The model f can, basically, be any ML model (i.e. we are only interested in what comes out of it). The same cannot be said about g, which is much more important for this technique and can be defined as:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \quad (2.18)$$

In the equation above, g is, as we know, the explanation (surrogate) model, z' is a simplified vector of ones and zeros to enable or disable certain features (also known as the coalition vector), M is the maximum coalition size and ϕ_i is the coefficient (i.e. Shapley value) for feature i. For tabular data, the coalition vector would turn on or off certain features and for images it would do the same process but for super-pixels. According to [LL17], formulation 2.18 has certain properties, amongst which, local accuracy (i.e. the explanation model should match the original model in terms of predictions).

The final piece of the puzzle is the SHAP kernel, responsible for attributing more weight to small or large coalitions, as opposed to coalitions that only aggregate half the features (or close to it). We happen to learn more about individual features if we can analyse them in isolation (small coalitions) or if we have almost every feature but one (large coalitions):

$$\pi_x(z') = \frac{(M-1)}{\binom{M}{|z'|}|z'|(M-|z'|)} \quad (2.19)$$

Finally, the five steps that bring everything together are as follows:

1. Sample K coalitions: $z'_k \in \{0,1\}^M$.
2. Obtain predictions for each z'_k by using $f(h_x(z'_k))$. Note that h is a function that enables or disables features to form coalitions. For tabular data, it keeps the features we wish to form a coalition with and, for the deactivated features, it samples values from other instances. As for images, it keeps super-pixels (if they are paired with a 1 in the coalition vector) or replaces them with a solid colour (if paired with a 0).
3. Compute the weight for each z'_k with equation 2.19.
4. Fit the weighted linear model using equation 2.17.
5. Return the coefficients from the linear model, namely, the Shapley values ϕ_i.

After training, the computed coefficients (ϕ_i) are the Shapley values that explain each feature's effect over x's prediction. In order to illustrate the way KernelSHAP works, the following figure contains a visual explanation of a CNN's prediction:

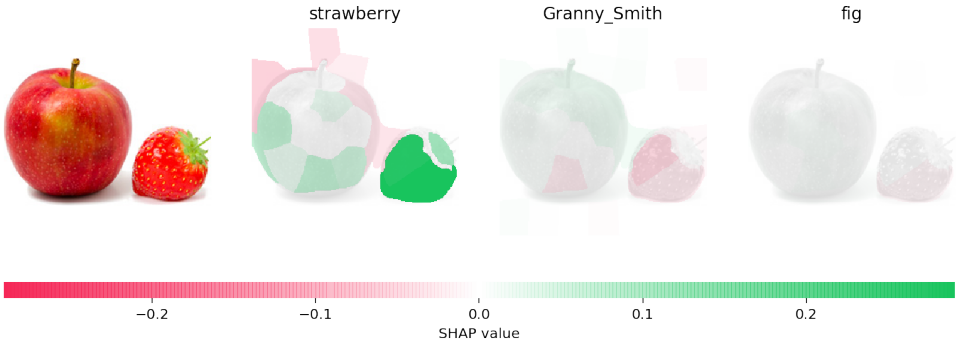

Figure 2.21: KernelSHAP's explanations of a CNN's predictions when given an image from the ImageNet dataset [LL17]. Super-pixels marked with green tones have higher Shapley values and thus contribute more to the predicted class (as opposed to super-pixels marked with red tones).

SHAP is a solid technique with comparable results to LIME's, exceeding it in some cases. Additionally, it relies on grounded and proven concepts, like game theory, Shapley values and LIME's reasoning. However, KernelSHAP suffers much the same problems as other permutation based methods: by replacing omitted features with random ones, unlikely data points may be generated, which can lead to unrealistic explanations [Mol19].

Nonetheless, a possible application of SHAP to the problem of periocular recognition can be found in section 3.1.

2.3 Conclusion

The present chapter focused on describing DL and some of its main architectures, which lay the foundation for the proposed methods. Such description was followed by a compilation of the most relevant techniques in the field of ML Interpretability. Some of them are perfectly valid and interesting, but their applicability is let down by the lack of visual explanations or overly simple nature, while others are better suited to what this work is trying to accomplish. As an attempt to empirically validate this belief, the following chapter contemplates a section solely dedicated to discussing what results can be expected by only using existing techniques, in the search for an adequate level of interpretability.

Chapter 3

Proposed Methods

The present chapter can be seen as a twofold: first, it tries to make a point on whether we can be satisfied with the application of existing techniques to the problem of periocular recognition; second, building on the possible conclusions, it provides an extensive overview of two methods that try to explain why two images from periocular regions belong (or not) to the same person.

To that end, the following sections are organised as follows: section 3.1 describes how Saliency Maps, LIME and SHAP could be used, in the context of this work; sections 3.2 and 3.3 contain an overview of the novel methods developed to tackle the problem at hand; section 3.4 summarises the main takeaways from the present chapter.

Finally, it should be noted that the results seen from this point forward were made possible through the use of a combination of the University of Beira Interior Periocular (UBIPr) [PP12] and Flickr Faces High Quality (FFHQ) [KLA19] datasets. On one hand, the UBIPr dataset is naturally oriented towards periocular recognition problems and contains attribute annotations. On the other hand, the FFHQ dataset contains full face images, requiring some special steps to extract just the periocular region and annotate the resulting images. In practice, just a small portion of the FFHQ samples was used, to balance the attribute distribution of the UBIPr dataset, thus creating a more equitable super set.

3.1 Review of Current Approaches

In order to justify the pursuit of new techniques with interpretability in mind, one must start by using existing methods. To that end, the present subsection is dedicated to describing the experiments conducted using the aforementioned methods and their respective results. Model-specific (Saliency Maps) and model-agnostic techniques (LIME and SHAP) were employed, so as to give a general overview of what is achievable with currently available approaches. LIME and SHAP's results were obtained using their official implementations[1] [2]. All three techniques were paired with the same DenseNet-121 model, whose task was to output the probability of two images belonging to the same person. Note that the samples seen hereafter belong to the disjoint test set, meaning they were never used during training and serve the purpose of simulating real-world scenarios.

The pairs that the CNN classified as "genuine" are shown in figure 3.1. The network seemed really confident in classifying these pairs, going inline with the expected behaviour:

[1] https://github.com/marcotcr/lime
[2] https://github.com/slundberg/shap

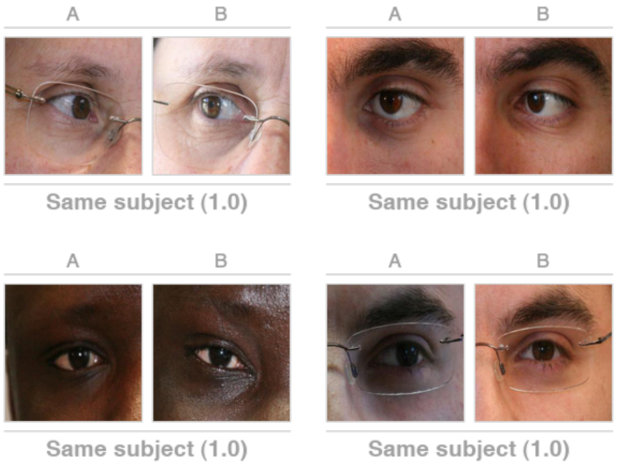

Figure 3.1: Genuine pairs correctly classified by the CNN. Note that, when the network deems the pair to be ”genuine”, it doesn't produce an explanation, reserving that step for ”impostor” pairs (this behaviour was, simply, an implementation decision).

Given that they are more interesting to discuss, the remainder of the present section will analyse how the aforementioned techniques explained ”impostor” pairs. It should be noted that some of this setup's limitations can be attributed to the CNN upon which the explanations are built, and not just the interpretability techniques themselves.

Saliency Maps are, essentially, greyscale images where whiter tones highlight crucial areas used by the CNN to predict a class. Accordingly, the top-right pair shows how the subject A's eyeglasses and subject B's eyebrow contributed the most to an ”impostor” verdict. Continuing a similar trend, the bottom row pairs show the eyebrows as differentiating factors (while erroneously leaving behind important components, like the skin spot in the bottom-left sample):

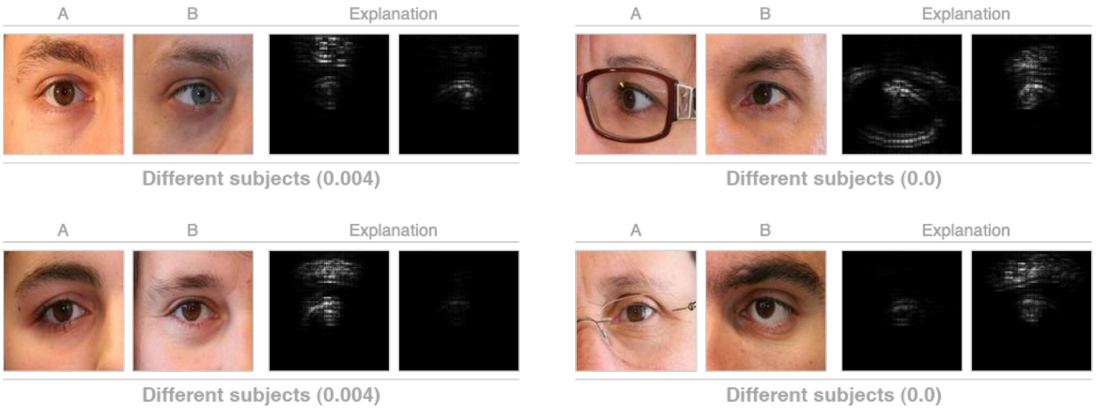

Figure 3.2: Impostor pairs explained using Saliency Maps. Whiter tones highlight areas that justify the predicted class.

LIME, as previously mentioned, divides images into super-pixels that are either kept or disabled. With effect, in figure 3.3, the top-left sample hints at a difference in terms of the eyebrows and a portion of the skin as well (although, the lack of a super-pixel in the iris is not accurate, given the obvious differences). In the top-right example, the highlights

comprise a fraction of the eyeglasses and an area above subject A's eyebrow, which is considerably thinner than subject B's. Next, the lower-left pair also has differences in eyebrow thickness, which LIME appears to capture, and in terms of skin spots (subject A has a spot near the eyebrow that ended up not being highlighted). Finally, the lower-right sample shows that a distinguishing factor between the two subjects has to do with the skins and eyebrows (notice how widespread the highlighted super-pixel is):

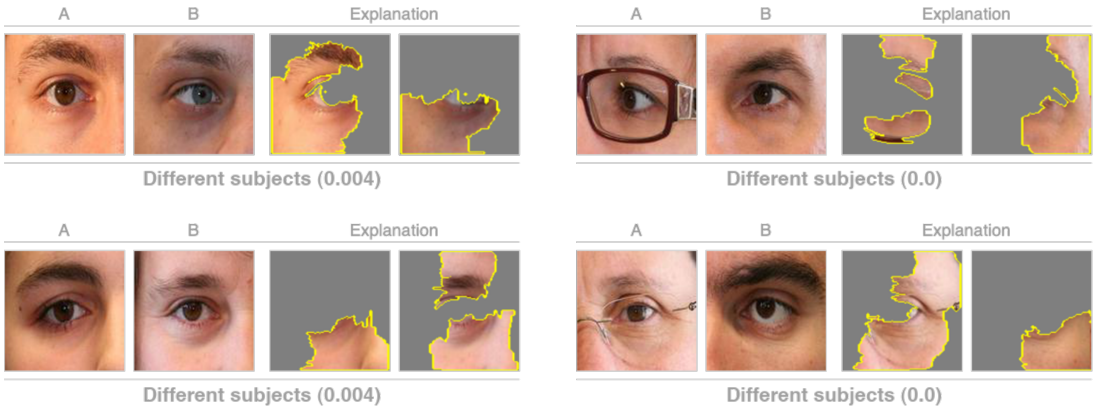

Figure 3.3: Impostor pairs explained with LIME. As mentioned earlier, LIME keeps the most favourable super-pixels and replaces the remaining ones with a solid colour.

KernelSHAP, on the other hand, produces results with various shades of green and red, depending on how favourable (green) or not (red) the highlighted super-pixels are to the predicted "impostor" class. Starting with the top-left pair in figure 3.4, we have two correctly highlighted components (iris and eyebrow) and an incorrectly highlighted one (subject B's skin spot). In the top-right pair, a large portion of the eyeglasses is depicted in green tones, whereas one of the irises is shown in red (which is correct, given how similar they are). Moving to the bottom row, the left pair barely includes a skin spot in a green region and hints at differences in eyebrow thickness with a decently sized green super-pixel above subject B's eyebrow. Finally, the bottom-right sample highlights a slight difference in iris colouration and, more prominently, a large green region encompassing subject B's eyebrow (which is positively different when compared to subject A's eyebrow):

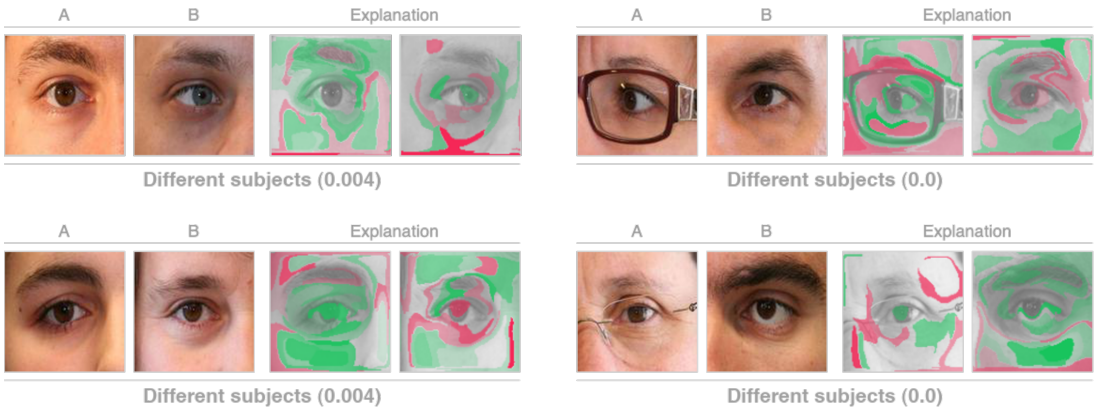

Figure 3.4: Impostor pairs explained with SHAP. SHAP diverges from LIME by highlighting certain areas with green or red tones, depending on whether they increase or decrease the probability of the output class.

These results are already decent and certainly keep interpretability at the forefront. However, there are cases where these techniques miss obvious components, and we argue improvements can be made with approaches that are specifically designed to address this type of problem. Therefore, the following sections will comprehensively describe two such methods, whose aim will be to increase the efficacy of the results shown so far.

3.2 Method A - K-Neighbours on a Big Synthetic Dataset

The first framework, that set out to tackle the problem of periocular recognition, is based on widely used DL architectures: CNNs and GANs. Such models were used as the basis of both parts that make up the final answer: the CNN for the traditional binary part and the GAN for the visual counterpart.

To go over the details of said method, subsection 3.2.1 covers the pre-processing steps that enabled the subsequent stages, which are later described in subsection 3.2.2.

3.2.1 Data Pre-processing

The data pre-processing stage is quite common in pipelines that involve some kind of DL model. These models often demand fixed size inputs and perform optimally with balanced and rich datasets. Naturally, the present method also required some preliminary steps:

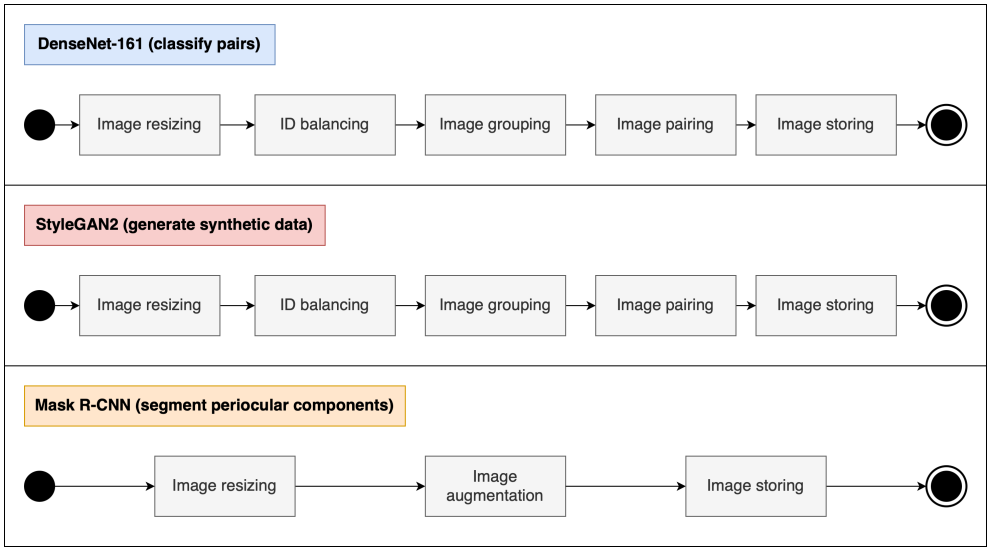

Figure 3.5: Diagram of the data pre-processing pipeline. The images are resized, processed and stored in proper folders.

Figure 3.5 depicts a visual representation of the main steps, divided into 3 sets (one for each of the main components). More details on these components can be found in the next subsection, whereas the pre-processing steps are as follows:

- **Image resizing**: the images are resized to a fixed size (e.g. 256x256 pixels).

- **ID balancing**: considering that, in the unprocessed dataset, some IDs possess more images than others, each ID gets it images either sampled or augmented. On one hand, if the ID has less images than a pre-defined target, the existing ones are augmented to reach said target (with techniques like horizontal flips, rotations, contrast changes or noise additions). On the other hand, if the image count is too great, a simple sampling routine chooses as many images as the target value enforces.
- **Image grouping**: the images are grouped by ID and some of them are reserved just for the test phase (according to a pre-established list).
- **Image pairing**: the images are paired with each other, to form either ”genuine” or ”impostor” pairs.
- **Image storing**: the images are stored in folders targeted towards training, validation and testing (in the case of DenseNet). As obviously required for the CNN and GAN, within each of these main folders there are class sub-folders (”0” for ”impostor” pairs and ”1” for ”genuine” pairs). The Mask R-CNN model just required the images and corresponding masks to be stored in standard folders.

3.2.2 Method Description

In this subsection, the main steps included in the present method will be described, both in visual and textual forms. To that end, figure 3.6 contains the schematics of how the pipeline flows, while a more detailed description is included in the subsequent paragraphs:

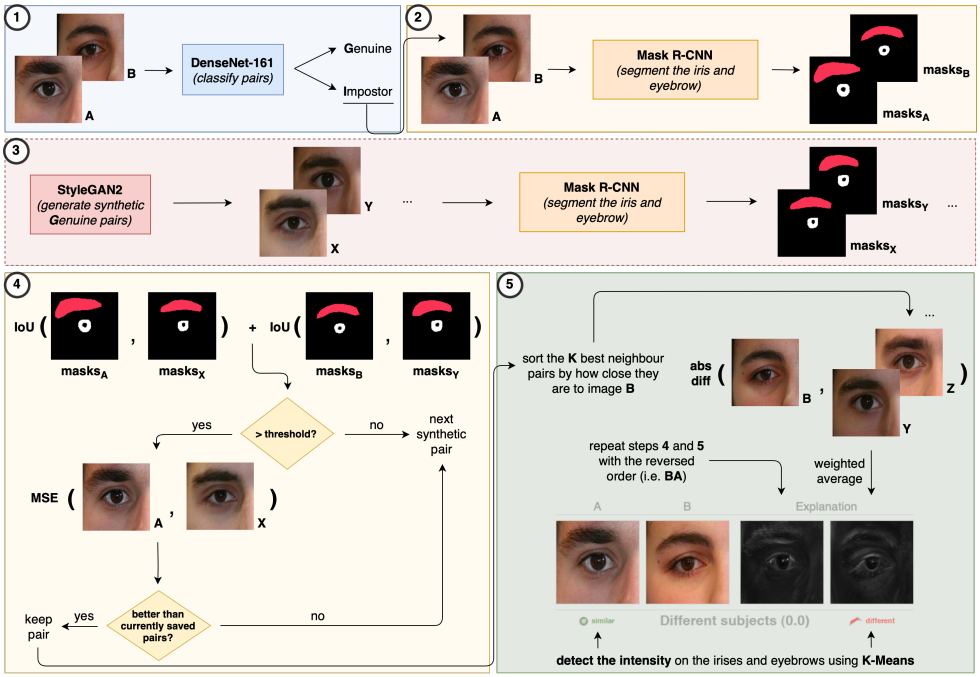

Figure 3.6: Diagram of the main pipeline. Step one constitutes a traditional approach, where a CNN distinguishes between ”genuine” and ”impostor” pairs. The latter are explained with steps two to five, in which we try to find ”genuine” synthetic pairs that closely resemble the test pair. By doing so, and despite looking similar, the test pair will probably contain certain internal differences (i.e. between images A and B) that the synthetic ones do not, thus providing an interpretable explanation.

The first step involves the use of a CNN (e.g. DenseNet-161) to provide the binary part of the final answer (”yes” or ”no”). Upon receiving a pair of images, the model's output consists of a vector of probabilities for the two classes (”genuine” and ”impostor”). Then, the highest value ends up being chosen as the network's prediction.

With the first part taken care of, the remaining ones (two to five, in the diagram) are responsible for creating the visually interpretable explanation.

According to the numbering system used, step two takes the test images and, using an already trained Mask R-CNN model, segments the irises, scleras and eyebrows. Then, step three uses a trained StyleGAN2 generator to create a large, synthetic dataset of genuine pairs (i.e. where both images have differences but belong to the same person). For each of these synthetic pairs, another CNN determines the side configuration (i.e. whether both images come from the left or right side of the face). Next, masks are obtained with the segmentation model in the same fashion as the test pair. It should be noted that this step can be done beforehand and not during the inference stage. This is also the preferred behaviour, given how time consuming step three can be.

With the synthetic images and masks obtained, we can use the latter to determine which generated pairs have the iris, sclera and eyebrow in roughly the same position as their counterparts in the test pair. Such pre-condition is key, given that the pixel differences, which make up the visual explanation, are sensitive to component misalignment. To achieve said condition, a synthetic pair must have a Intersection Over Union (IoU) mask score above a pre-defined threshold. Upon successfully meeting the requirement, the image A from that pair is compared against the test pair's image A (using a metric like MSE). If the score is better than the currently saved pairs, the considered pair is kept as one the best matches, to that point. this iterative process continues until every synthetic pair is analysed. What this process tries to do is to find, amongst the thousands of synthetic pairs, the ones that are closer to the test pair, terms of image A. Therefore, if the test pair's image B is a genuine match, it must be similar to the images B from the chosen synthetic pairs. On the other hand, if image B is not a match, it will most likely be different in some areas, and that is exactly what we want to highlight in the final explanation.

From this point forward, we have selected the K closest pairs (or neighbours) from the synthetic dataset. It becomes possible, then, to sort those neighbour pairs by how close they are to image B, using the MSE metric. In practice, for each image B from the synthetic pairs, we compute pixel differences against the test pair's image B, resulting in a final, overall difference, which is a weighted average of those intermediate differences. In practice, a neighbour's distance is subtracted from the total sum of distances, thus creating an inverted distance. This approach ensures that neighbours with smaller distances receive more weight as opposed to neighbours with bigger distances. Skipping this step would give more weight to neighbours with bigger distances, which is obviously undesirable. Then, the inverted distances are simply divided by the sum of inverted distances.

To complete the explanation, an extra step is responsible for translating some visual cues into text. To do so, the iris and eyebrow masks aid in retrieving the areas of the explanations that are located in the irises and eyebrows. Next, using the clustering algorithm

K-Means, the dominant intensity can be computed for both the iris and eyebrow regions. Because we are working with greyscale values, the whiter tones hint at more differences, while darker ones represent the opposite. Then, based on pre-defined thresholds, the intensities of the iris and eyebrow regions are given a textual representation, ranging from ”very similar” to ”very different”. With this simple, yet effective step, we provide another dimension to explanations that would otherwise be exclusively visual.

By doing the steps above, we have explained image B and highlighted the areas that motivate a ”no” decision from the CNN. To replicate the same process for image A, we just have to repeat steps four and five in the reversed order (i.e. in four, calculate the MSE differences with images B and, in five, the pixel differences with images A).

3.3 Method B - Latent Space Directions

[**WORK IN PROGRESS**]

3.3.1 Data Preprocessing

[**WORK IN PROGRESS**]

3.3.2 Method Description

[**WORK IN PROGRESS**]

3.4 Conclusion

[**WORK IN PROGRESS**]

Chapter 4

Results and Discussion

The present chapter focuses on the experimental and practical side of this dissertation. In it, one can expect to find results for both methods and the conclusions that can be derived from each. To that end, section 4.1 contains details on specific hyper-parameters, training procedure and other general aspects. Section 4.2 includes results for both methods (some are direct comparisons with those found in section 3.1, while others are unique to this chapter). To finalise chapter 4, section 4.3 presents some final remarks as to how the two methods performed.

4.1 Implementation details

Regarding the first method, the DenseNet-161 model was trained for 15 epochs with a learning rate of 0.0002 and a batch size of 64 image pairs. The Adam optimiser was used to update the weights (with default β_1 and β_2 values). The CNN responsible for determining the side configuration of a given pair (ResNet-18) was trained with almost the same parameters (except for the number of epochs, which was set to 5).
The Mask R-CNN's training process used almost all the default values, translating into a learning rate of 0.001, a batch size of 1 and 30 epochs worth of training (in this case, fine-tuning from the COCO pre-trained weights).
With regards to how StyleGAN2 architecture was trained, it comprised a total of 80000 iterations, a batch size of 8 and an image size of 256x256 pixels. The size of the generated synthetic dataset settled on, roughly, 400000 pairs, a number that, if increased, can very well improve the overall results (more samples enable more variance and possibly better neighbour matching).
Recalling figure 3.6, step four mentions a certain threshold that the synthetic pairs' masks must meet in order to be accepted. In practice, the IoU score was calculated on a per component basis (i.e. first we check if the irises meet and then the eyebrows). The default threshold values were set to 0.45 and 0.25 for the irises and eyebrows, respectively. Moreover, the number K, which determines how many synthetic neighbours should be be kept, received a value of 15.

4.2 Results Comparison

As an attempt to compare the first method's results with those in section 3.1, the following figure shows the same four pairs, as well as, an additional six pairs:

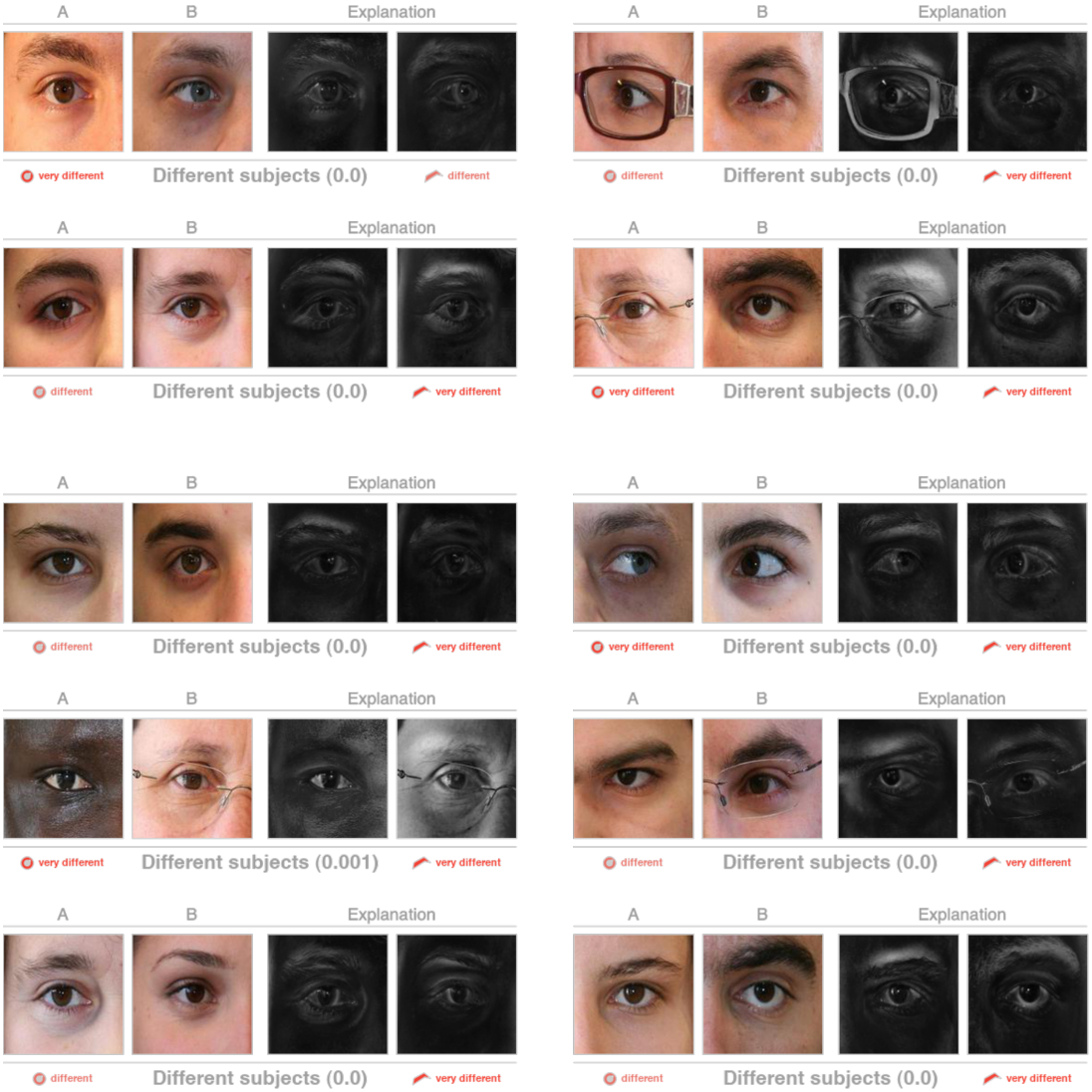

Figure 4.1: Results obtained with the first method. The top four pairs can be directly compared with the results seen in figures 3.3 and 3.4, where as the bottom six pairs are exclusive to this section.

As seen above, the first method is able to highlight interpretable areas like the eyebrow, iris or skin spots. When compared to LIME and KernelSHAP, these results are, arguably, more expressive and interpretable. Proving just that, the third and fourth pairs show prominent white areas around the eyebrows and, in the case of the third pair, it goes a step further by including accurate differences in terms of skin spots and eyelashes.

Moving to the bottom six pairs, which are unique to this section, the first sample highlights, once again, an interpretable difference with regards to eyebrow thickness. Next to it, the second pair shows a slightly whiter tone in one of the irises, showing some differences there. Then, the third sample also stands out from the rest by giving the skins a whiter tone, something that is not seen in any other test pair (this behaviour is perfectly accurate, given the contrasting skin colours). Right next to it, the fourth pair rightfully includes, besides other aspects, white tones in subject B's eyelid and eyelashes. Finally, amongst the last row, the two test pairs show evident disparities in terms of how the eyebrows are composed and the last pair even includes whiter tones in the sclera underneath

subject B's iris (which is, effectively, distinct from subject A's sclera).

The results shown here are decidedly interpretable and, in many cases, fairly accurate. With them, we have justified why the DenseNet model classified all the pairs seen here as coming from different persons. Aiding the visual cues, the text captions help in providing additional sources of information.

4.3 Conclusion

[**WORK IN PROGRESS**]

Chapter 5

Conclusion

The present chapter is a general, high-level overview of the work that was developed in the scope of this dissertation. To that end, subsection 5.1 withdraws the most important conclusions, while subsection 5.2 enumerates the ways in which the work could be improved. Together, these two subsections serve as a succinct collection of words that summarises many months of research and experimentation.

5.1 Main Conclusions

[**WORK IN PROGRESS**]

5.2 Future Work

[**WORK IN PROGRESS**]

Bibliography

[AZ19] Daniel W. Apley and Jingyu Zhu. Visualizing the effects of predictor variables in black box supervised learning models, 2019. 16

[CSR14] Yangqing Jia Pierre Sermanet Scott Reed Dragomir Anguelov-Dumitru Erhan Vincent Vanhoucke Christian Szegedy, Wei Liu and Andrew Rabinovich. Going deeper with convolutions, 2014. 6

[Fri01] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001. Available from: https://doi.org/10.1214/aos/1013203451. 15

[Fuk04] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193–202, 2004. 4

[GDDM14] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014. ix, 7, 8

[GHW18] Laurens van der Maaten Gao Huang, Zhuang Liu and Kilian Q. Weinberger. Densely connected convolutional networks, 2018. ix, 6

[Gir15] Ross Girshick. Fast r-cnn, 2015. ix, 8

[GLGSL18] Yanming Guo, Yu Liu, Theodoros Georgiou, and Michael S. Lew. A review of semantic segmentation using deep neural networks, 2018. Available from: https://doi.org/10.1007/s13735-017-0141-z. 4

[HSL+16] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Weinberger. Deep networks with stochastic depth, 2016. 6

[HZK+18] Zhenliang He, Wangmeng Zuo, Meina Kan, Shiguang Shan, and Xilin Chen. Attgan: Facial attribute editing by only changing what you want, 2018. 12

[IJGB14] Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Ian J. Goodfellow, Jean Pouget-Abadie and Yoshua Bengio. Generative adversarial networks, 2014. 4

[KALL18] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation, 2018. 12

[KHS15] Shaoqing Ren Kaiming He, Xiangyu Zhang and Jian Sun. Deep residual learning for image recognition, 2015. 6

[KLA19] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019. x, 13, 29

[KLA+20] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan, 2020. x, 12, 13

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012. Available from: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf. 4

[LBBH98] Y. LeCun, L. Bottou, Yoshua Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 1998. 4

[LHGK19] Ben Z. Yuan Ayesha Bajwa Michael Specter Leilani H. Gilpin, David Bau and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning, 2019. 1

[Lip17] Zachary C. Lipton. The mythos of model interpretability, 2017. 14

[LL17] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions, 2017. x, 24, 25, 26

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2015. ix, 10

[MAB17] Soumith Chintala Martin Arjovsky and Léon Bottou. Wasserstein gan, 2017. 12

[Mat20] MathWorks. Train generative adversarial network (gan), 2020. Available from: https://www.mathworks.com/help/deeplearning/ug/train-generative-adversarial-network.html. ix, 11

[MAZ15] Paul Barham Eugene Brevdo Zhifeng Chen Craig Citro-Greg S. Corrado Andy Davis Jeffrey Dean Matthieu Devin Sanjay Ghemawat Ian Goodfellow Andrew Harp Geoffrey Irving Michael Isard Yangqing Jia Rafal Jozefowicz Lukasz Kaiser Manjunath Kudlur Josh Levenberg Dandelion Mané Rajat Monga Sherry Moore Derek Murray Chris Olah Mike Schuster Jonathon Shlens Benoit Steiner Ilya Sutskever Kunal Talwar Paul Tucker Vincent Vanhoucke Vijay Vasudevan Fernanda Viégas Oriol Vinyals Pete Warden Martin Wattenberg Martin Wicke Yuan Yu Martín Abadi, Ashish Agarwal and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. Available from: https://www.tensorflow.org/. 4

[Mol19] Christoph Molnar. *Interpretable Machine Learning*. 2019. Available from: https://christophm.github.io/interpretable-ml-book/. x, 14, 16, 18, 21, 25, 26

[PML+19] Gross Sam Paszke, Adam, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 4

[PP12] Chandrashekhar Padole and Hugo Proença. Periocular recognition: Analysis of performance degradation factors in proceedings of the fifth iapr/ieee international conference on biometrics – icb 2012, new delhi, india. 03 2012. 29

[Pra18] Prabhu. Understanding of cnn, 2018. Available from: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. ix, 5

[RHGS16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. ix, 9

[Ros57] F. Rosenblatt. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory, 1957. Available from: https://books.google.pt/books?id=P_XGPgAACAAJ. 3

[RSG16] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should i trust you?: Explaining the predictions of any classifier, 2016. x, 21

[RSG18] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations, 2018. x, 22, 24

[She18] Susan Sheldrick. What does ai know: Model interpretability and occlusion [online]. 2018. Available from: https://www.linkedin.com/pulse/what-does-ai-know-model-interpretability-occlusion-susan-sheldrick/. x, 19

[SVZ13] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013. x, 19, 20

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015. 6

[USGS13] Jasper Uijlings, K. Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104:154–171, 09 2013. ix, 7

[ZF13] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks, 2013. x, 19