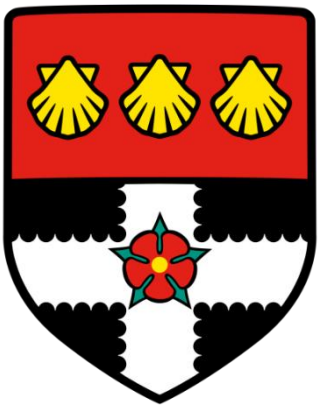


University of Reading
Department of Computer Science

Morph Attack Detection using a Machine Learning Classifier

Hari Gupta

Supervisor: James Ferryman

A report submitted in partial fulfilment of the requirements of the University of Reading for the degree of Bachelor of Science in Computer Science

May 14, 2024

Declaration

I, Hari Gupta, of the Department of Computer Science, University of Reading, confirm that this is my own work and figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

I give consent for my work to be made available more widely to members of UoR and public with interest in teaching, learning and research.

Hari Gupta
April 16, 2024

Acknowledgements

A very special thank you to Dr James Ferryman and Dr Jonathan Boyle
for their continued and extended, support and guidance throughout this project.

Contents

List of Abbreviations

- SOTA - State of the Art
- ML - Machine Learning
- NN - Neural Network
- CNN - Convolutional Neural Network
- MAD - Morph Attack Detection
- PAD - Presentation Attack Detection
- ABC - Automatic Border Control
- S-MAD - Single morph attack detection
- D-MAD - Differential morph attack detection
- GAN - Generative Adversarial Networks
- ReLU - Rectified Linear Unit
- SDG - Stochastic Gradient Descent
- PID - Project Initiation Document

Abstract

This project aims to provide an exploration into morph attacks, by reviewing existing research in the field, and detailing the processes involved in morph creation & detection. The context surrounding our solution, alongside our results visualisations and evaluation methods, inform us that creating a morph attack detection algorithm, requires a very high level of specialised education, and thus puts up a barrier of entry to anyone outside the field, stifling innovation.

Keywords: Morph Attack Detection, Facial Recognition, Literature Review, Machine Learning, PyTorch

Report's total word count ~16,000

Chapter 1 - Introduction

1.1 Background

Facial Recognition has integrated into many areas of society within the past decade, trusted by many as a well reputed identity-verification method for secure applications. However, in the past few year's attackers have been using increasingly complex methods to spoof and bypass these algorithms using Presentation Attacks. This means that the reliability of any facial recognition solution is dependent on the robustness on its anti-spoofing methods. One such presentation attack that is steadily on the rise, and a current hot topic amongst lawmakers and academics, is face morphing attacks. On a very basic level, these involve merging two or more faces into a singular image that can be used to represent either of the original faces. This is currently causing ongoing issues with passports/identification documents, especially at border control.

This project is an exploration into morph attacks, their countermeasures, and the processes involved in developing morph detection algorithms. Providing, an overview of how MAD fits into the wider scope of current SOTA solutions, and an introduction into important concepts involved in PAD and Facial Recognition at large. The motivation behind this project was the distinct lack of in-depth intermediate literature on this topic; we aim to bridge the gap between oversimplified news articles and hyper-specific research papers that require a high level of application-specific knowledge to understand; Making it easier for more people to learn about PAD concepts, and attempt creating their own algorithms.

1.2 Aims and Objectives

The aims of this project are to:

- Develop an understanding of the current state of research in the field.
- Look towards current real-world solutions for inspiration and comparison.
- Create a MAD solution that can outperform humans at face-morph detection.
- Provide in depth guidance on how the solution was formulated and implemented.

The goals of this project are to:

- Analyse various pieces of literature from differing sources on relevant topics and conduct a thorough literature review to better understand the wider context, within which our project exists.
- Consider the requirements of our project and social, legal, ethical implications, so that we are well informed on how to approach the development of our solution.
- To create a large, suitable, and normalised dataset of face-morphs and bona-fide faces to act as our training and testing data.
- Build a convolutional neural network that can classify face images to detect morphs.
- Test and iterate on our algorithm to optimise it and make it perform better.
- Evaluate the final performance of our MAD algorithm using ACPER and BCPER metrics.

1.3 Summary of contributions and achievements

In the duration of this project, we have created a highly automated data pipeline, that streamlines the processes involved.

Starting from raw facial image data, selecting suitable faces, normalising, and aligning them, creating randomised morphs, and then using these images to create the final dataset of over 1.4k suitable images for training and validation.

The MAD algorithm script, loads in the dataset, uses custom data loaders to transform the images and label them, and randomises them in preparation for the model. It continues by iterating the data through the model, backpropagating to update weights / perform learning, it repeats this for every batch of every epoch. Followed by a validation pass once per epoch to test the model performance on unseen data and displays the associated data visualisations created by our custom functions, allowing us to evaluate its performance over time.

The overall summary of our results is as follows: Our model and any model of this complexity, will not be able to sufficiently compute the relevant patterns and learn the differences between convincing face-morphs and actual bona-fide images. This implies we would need to create a more complex solution, with more specific knowledge about ML in facial biometrics, and more compute power that could handle training these solutions.

1.4 Organisation of the report

The following report is organised into four main sections.

Literature review: where we discuss the current literature, SOTA solutions in the field of research and how these pertain to our project.

Methodology: where we delve in-depth into all the processes of developing our MAD solution, explaining our approaches and decisions made along the way.

Results, Discussion and Analysis; in this area we describe the results of our MAD algorithm. Followed by an extensive evaluation, where we analyse and discuss the results.

Conclusion, Future work, and Reflection; in this area we consider the scope of our entire report and project. Synthesising and re-instating major points and findings, outlining future work to be done, and reflecting on the effectiveness of our solution.

1.5 Solution Approach

We need to create a custom dataset, fit for purpose to train our ML solution. For this, we used a publicly available repository from GitLab to perform the face morphing (*written primarily in python*). However, since it was quite outdated, we had to alter its libraries, adjust existing scripts, and write custom scripts to automate our dataset creation.

Our ML solution was implemented in the python language, using a local environment that managed a plethora of libraries relevant to the project. The main libraries of note required for our solution were, Pandas, NumPy, PyTorch, and Matplotlib, these were primarily used for data handling, machine learning, and results evaluation.

We also implemented CUDA into our ML process, so that the model training and validation cycles were accelerated by performing computations on our GPU instead of our CPU. GPUs can break down massively complex problems into multiple smaller simultaneous calculations and solve them quicker, thus speeding up the training process.

Chapter 2 - Literature Review

2.1 A review of current 'state-of-the-art' theories and solutions in the field of research

2.1.1 Facial Recognition

In previous decades, landscape of biometric verification has been rapidly changing, Facial Recognition in particular, has consistently seen major advancements every few years. As more research is conducted in the field, the concepts, approaches, and solutions increase in their complexity and effectiveness. [c1]

With facial recognition recently becoming a more viable security solution, it has been introduced across various aspects of public life. In technologically developed countries, most people are regularly using facial recognition systems for a whole host of applications. For example, CCTV monitoring, Border Control, Entering workplaces/campuses, Banking, and most commonly Personal device security.

2.1.1.1 How does Modern Facial Recognition work

The basic concept is as follows, first you must have a legitimate and high-quality dataset of people's facial images stored in your database, then when anyone tries to verify themselves as one of said people you take an image from a capture device and compare the two biometric samples for similarities and differences.

However, since its initial conception the science has advanced a lot further, and 'Modern' facial recognition uses several key steps to achieve this goal of verifying a person's identity. [c3] [a4] Here are a few examples of the most important steps:

- Face detection
    - First and foremost, the facial regions in an image must be determined, and distinguished against any background/irrelevant image information; then subsequently acquire the alignment of the face, i.e. position, size, rotation.
- Face Alignment (Lighting and pose correction)
    - With the Information provided at the previous stage, the system must correct for the lighting conditions (if reasonable) and manipulate the detected face so that it is aligned correct with regards to pose and position.
    - This 'normalisation' of biometric samples is important for the effectiveness of the next steps in facial recognition.
- Feature extraction
    - Using the correctly aligned face images, the face recognition algorithm will detect the location of facial components like eyes, nose, mouth etc. It will then extract landmarks such as the nose bridge, eyebrow contours, chin etc.
    - These extracted features will be converted into a representation that is suitable for comparison, i.e. by using techniques such as vector normalisation to reduce dimensionality.

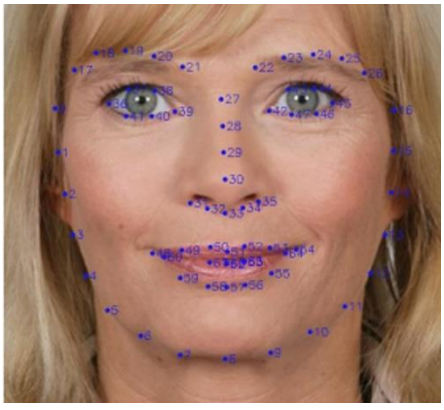


Figure 1 - [a4] Landmarks for 2D Facial Recognition

- Face matching
    - Once the features of our biometric sample are represented, the algorithm iterates through a database of known faces represented in the same way, using various similarity measures (i.e. Euclidean distance) to judge the differences between the two images.

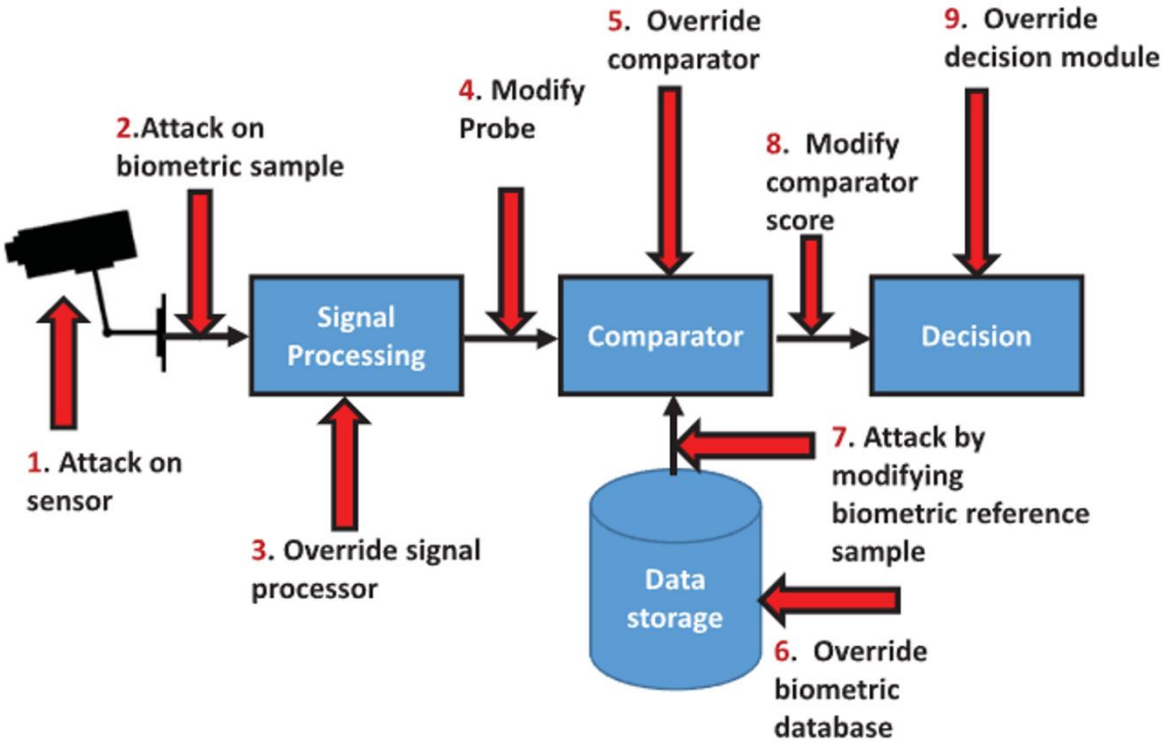
- Based on the similarity scores obtained, a decision is made on whether or not the detected face is matches that of a known person in the database, there is usually a threshold value here that can be adjusted. You don't want the threshold to be too high because otherwise even a small change in photo angle could make it impossible for a person to get identified, but for applications where the data being secured is highly sensitive, it is reasonable to air on the side of caution and get some false negatives in exchange for increased security.

2.1.1.2 Recent Improvements in SOTA Facial Recognition

In SOTA facial recognition solutions, CNNs are used to do the face matching, often also implementing multiple various NNs to do other steps. CNNs have proven very powerful and robust for facial recognition as they can very quickly learn discriminative features, from raw data without the need for handcrafted-features traditional methods relied on. CNNs are also able to perform better under challenging pose and lighting conditions within an image, reducing the need for data cleaning and preparation.

In addition, some newer SOTA solution use Multimodal face recognition to greatly improve the accuracy and performance of their algorithms. The multi-modal approach in this context entails combining 2D and 3D face data for biometric recognition [c4]. In the feature extraction stage in addition to the standard 2D feature extraction, a 3D mesh of the persons face is also constructed. A sort of 'grid' of landmarks are projected by an infrared emitter and the reflections are captured by a depth sensor; that uses reflection travel time to determine distance. An algorithm then uses this data to fill in the gaps and create an appropriately complex mesh of the persons face.

2.1.1.3 Attack vectors on Facial Recognition Systems



[b1] Looking at this diagram, we can see that, there are several attack vectors that a bad actor may use to disrupt and manipulate a standard facial recognition system; However, within the scope of this literature review and the remaining report we will focus solely on the first attack vector, the sensor where biometric samples are captured.

In the next section we shall discuss in great detail, the theories behind ensuring the biometric samples (facial images) captured are of sufficient quality. As this is incredibly important to the efficacy and performance of any facial recognition system. 'Poor' biometric data will at best be images of low visual

quality and create annoying false negatives for legitimate users, or at worst be a manipulated biometric sample from an attacker, yielding a false positive and bypassing the security measures in place.

2.1.2 Face Image Quality Assessment

The quality of the image provided for Facial Biometric applications is integral to the effectiveness of the entire biometric verification pipeline. A poor biometric sample can make it hard for anti-spoofing measures to accurately flag and block tampered samples, and hard for identity verification to perform feature matching to the best of its ability; Ultimately allowing for both false positives and false negatives, at best making it annoying for legitimate users and at worse compromising the integrity & security of the whole system.

Facial Images in particular, have various specific aspects that need to be checked.

The first step is pre-processing the data, this entails finding faces in the image and restricting computations to the relevant face region. After obtaining the face image, you compute the facial landmarks, by determining the exact position of important features, (such as eyes, mouth corners, nose bridge, cheekbones, etc. Lastly you perform subject segmentation, by determining specific regions in the image that represent certain parts of the subject, (i.e. hair, jawline, forehead, etc.)

There are a few capture related aspects to consider, Illumination, Natural Colours, Sharpness, Compression, and Radial distortions. [a4]

Illumination issues include the being too bright or dark, narrow frequency in brightness distribution, and uneven spatial brightness distribution (shadows, hotspots). These are usually caused by unsuitable lighting, under/over-exposure and post-processing done to the image. Illumination problems can cause reduced visibility of texture details, however SOTA facial recognition algorithm CNNs are robust with regards to illumination.

In terms of sharpness & spatial resolution, finer details and textures of the face should be clearly visible. The different image quality issues that can arise are, de-focus of the capture system, motion blur due to movement of the subject or capture system, and post-processing like resizing or printing/scanning. These have potential to negatively impact the effectiveness of facial recognition algorithms, however some SOTA CNNs can take 112x122 size inputs. Therefor it may be more important for other scenarios such as manual identity verification, and manipulation detection.

There are algorithms that can be used for the assessment of illumination and sharpness, the results of these algorithms can then in turn be fed into AI neural nets to classify whether an image should be accepted or rejected at this stage in the pipeline.

The algorithms used for Illumination assessment are as follows:

- Analysis of luminance histograms (e.g. WD5 of ISO/IEC 29794-5)
    - Statistical moments (mean, variance, skewness, kurtosis)
    - Count pixels with very low / very high value.
    - Dynamic Range
        - Entropy of luminance histogram
    - Illumination uniformity
        - Measure intersection of luminance histograms in zones on left and right cheek
- Assessment by CNN: $FIIQA_{DCNN}$

The algorithms used for Sharpness assessment are as follows:

- Features for edge or texture detection (e.g. Laplace, Sobel, LBP)

- Statistical approaches (eigenvalues of image's covariance matrix)
- Comparison with blurred version of image
- Distribution of local intensity variation / gradients
- Analysis in frequency domain
- Classification by CNN

There are also many subject-related aspects that are necessary to consider when determining image quality, we will focus on three main ones, Occlusions of Face, Head Pose and Expression Neutrality. [a4]

Occlusions of face refers to objects obscuring parts of the face during capture, e.g. sunglasses, hair, facemasks, this obviously has negative impacts on the accuracy of biometric algorithms and thus we want to detect when it occurs. The QA algorithm should indicate which parts of the face are not visible (eyes, nose, mouth) and provide actionable feedback. Many CNNs can detect standard/common occlusions (glasses, face coverings, etc), but only a few can detect arbitrary occlusions (i.e. objects in the foreground, or hands on face)

By ISO/IEC standards 19794-5:2011 and 39794-5:2019 Frontal Pose is required for reference images of Face data. Some modern face recognition algorithms can handle variations in pitch / yaw, but direct frontal pose is much preferred for most CNNs and manual inspection. SOTA algorithms use deep learning methods, and multi-task with different CNNs to calculate facial landmarks and pose estimation, creating a morphable 3D model that can be pose adjusted. However, the freely available CNNs are relatively small and unable to handle this.

Images with neutral facial expressions are required by ISO 19794-5 and ISO 39794-5. Same as with the previous subject-related factors, SOTA face recognition systems are robust against slight variations in expression; However, extreme expressions can lead to false rejection; most algorithms and manual inspections greatly benefit from neutral expression.

2.1.2.1 ICAO Photograph Standards

The (International Civil Aviation Organisation) ICAO passport photograph standards are a set of guidelines that all internationally recognised passports must adhere to, for identifying persons. Many of these ICAO guidelines overlap/coincide with common metrics of face image quality assessment. Here are the most relevant examples:

- *(ii)* The image must have adequate brightness and contrast
- *(iii)* The skin tone should be natural. In case of over-exposure or under-exposure of the photo, the skin is either too dark or too light, photo will not be acceptable;
- *(v)* The image should be straight looking, centred with neutral expression;
- *(vi)* Face should be in sharp focus and clear with no ink marks/creases/lines;
- *(viii)* The eyes must be open and no hair obscuring the face;
- *(ix)* Prescription glasses if worn should be clear and thin framed and should not have flash reflection or obscure the eyes;
- *(x)* Head coverings, hair, head-dress or facial ornaments should not obscure the face;
- *(xii)* There must be no other people or object in the photo;
- *(xiii)* The lighting must be uniform with no shadows on the face or behind;

This is highly relevant as borders switch from manual passport inspection to automatic border control gates, using facial recognition algorithms prone to higher level spoofing attacks. We must use current SOTA knowledge and tools to write robust face-image quality assessment algorithms, ensuring ICAO guidelines are met and spoofed images are rejected.

2.1.2.2 Liveness detection

Liveness detection can be used to help determine Facial-Image Quality, as it allows us to have greater trust in the facial images provided. It has some weakness to deepfakes however, and therefor is not recommended as a standalone solution; it's better suited to be used as part of a larger, more comprehensive biometric verification process.

Within liveness detection there are three main approaches, Passive, Active, and Hybrid. In the passive approach, liveness checks occur silently in the background, without the need for user interaction. In the context of facial recognition for example, this could be a front facing phone camera checking for natural movements, like blinking to verify authenticity. The Active approach on the other hand, requires some form of intentional user participation, such as following on-screen directions like tilting their head and looking side to side. The Hybrid approach combines the two methods for a visible but less intrusive solution, ensuring liveness detection accuracy remains uncompromised whilst making the process more streamlined.

Within these approaches there are different methods that may be used for implementation, Challenge & response, Depth & Motion perception, Algorithms & AI, and Multi-modality. [a1]

Challenge & Response is the most ubiquitous Active liveness check, it asks the user to respond to prompts like moving their head, smiling etc. This is intended to beat false representations like videos or images by having the user prove they can respond to specific prompts.

Depth & motion perception, in this method the authentication device can use complex motion data capturing algorithms and/or depth sensors to create a 3D map of the users face and collect more information about subtle facial expressions. This makes Liveness detection very hard for fraudsters to defeat and almost impossible for any 2D replay attacks.

Algorithms & AI are standard place in most biometric systems to verify identities, however, also using it to recognise changes to a authorised user's face and mannerisms, (i.e. facial-hair, glasses, facial-expressions, etc) can also improve the accuracy of biometric authentication.

Multi-modality requires multiple biometric inputs, (i.e. any combination of facial, retinal, vocal and thumbprint), making it one of the most secure approaches to biometric authentication. And greatly increasing the level of complexity an attack would need to have, especially when more than two modalities are being used, making it incredibly unlikely to be bypassed.

Depth-Data for Liveness-Detection in particular has had a lot of academic research and commercial interest in the past few years, leading to recent innovations in consumer electronics, pushing the technology to become cheaper to produce and more widely available to the public. This is expanded upon further in section 2.2.1.

2.1.3 Presentation Attacks

A Presentation attack is a spoofing-attack on biometric verification service, where a bad-actor presents someone else's physical characteristics or biometric data "spoofs", to impersonate them and bypass security measures. [b2]

Some examples of presentation attacks against facial biometric systems are, replay attacks, hyper-realistic masks, and morph attacks. [b3] Replay attacks are the most common and the easiest to use, but are equally easy to detect. These can range from a printed image of someone's face, to a video recording of a persons head (moving, blinking, etc) to emulate liveness. Due to its simplicity and ubiquitous nature, most liveness detection solutions can detect these with a great degree of accuracy. Masks can range in complexity and materials. The simplest is a 2D paper mask but it's ultimately unconvincing to any human

or half-decent facial recognition algorithm. On the other end of the scale, highly detailed silicone masks have been created, emulating skin texture, facial features and signs of aging to a very high standard; these could easily fool older, lower risk facial recognition systems like late 2010's Samsung phones, however in places/systems where security is important, SOTA facial recognition algorithms will be used, which have been robust to these attacks for the past few years.

Figure 2 - https://news.sky.com/story/hyper-realistic-masks-fool-a-fifth-of-people-say-researchers-11865827

In recent years as AI technology has advanced, Morphing attacks have exploded in popularity due to increasing complexity, usage, and research into creating morphs & their counter measures. This topic is explored in further detail in a following section of the literature review.

2.1.4 Morphing Attacks

Morphing attacks are a type of presentation attack where two (or more) *bona-fide* face images are morphed into a singular convincing looking image of a person; the goal of this image is to look like a close enough average, that either person can be identified with the morphed picture.

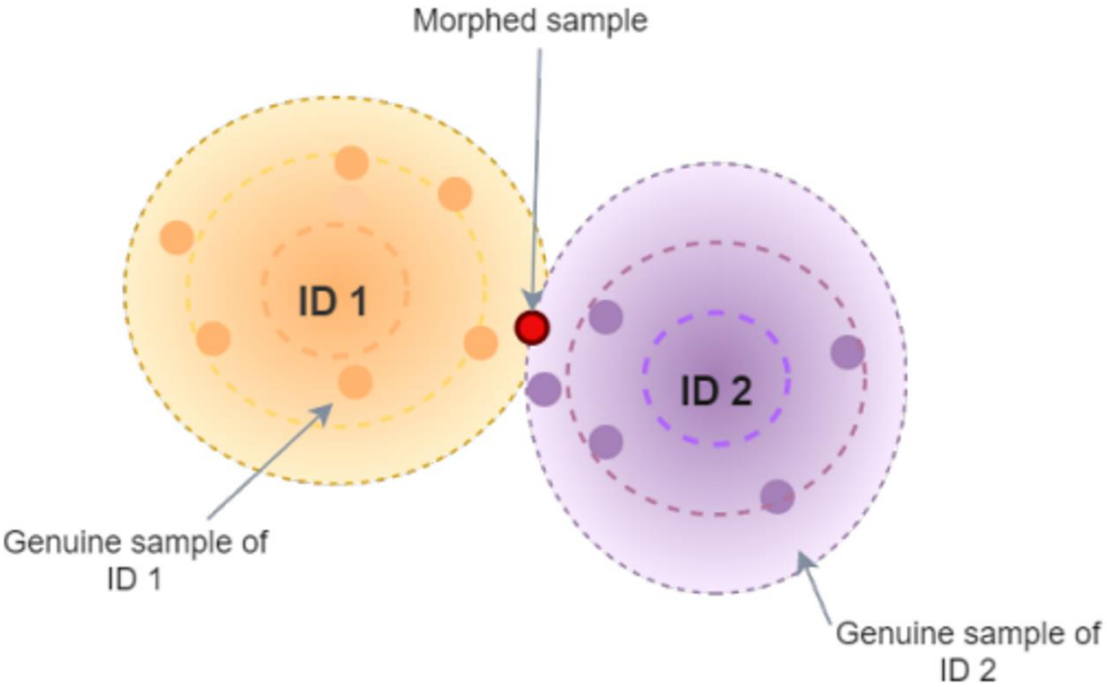


Figure 3 - [d4] Abstract representation of two similar identities in the embedding space. The morph is designed to be within the decision boundary for the two identities.

Image morphing has been an active area of research for over 40 years, with various applications such as film & entertainment [d1]. However, in recent years it has once again piqued the interest of researchers; As these algorithms have improved enough to create convincing morphs, bad actors have started using them to forge identity documents.

There is currently an arms race, between bad-actors creating face morphs and governments/security-contractors creating systems to detect them. Spurred on by the exponential growth in the field of machine learning, both sides are constantly playing cat and mouse trying to build undetectable morphs and un-spoof-able systems.

2.1.4.1 Classical Morphing Techniques vs Generative AI Morphing

As part of the effort to create more convincing morphs, researchers have tried to create new approaches to rival and outperform the status quo.

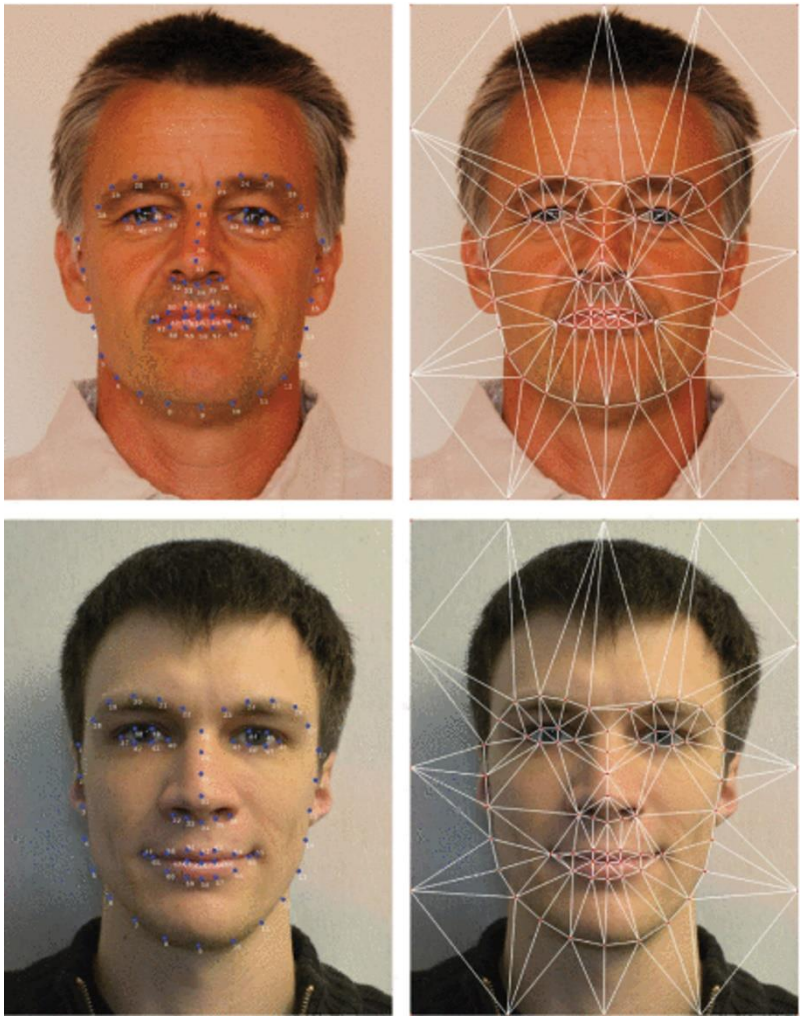


Figure 4 - [d1] Automatically Detected Facial Landmarks and Geometry

The conventional approach to morphing involves blending 2+ static images using automatically detected landmarks, warping to align the landmarks between the images, then finally blending the two aligned images together (*usually using linear blending, which simply averages colour pixel values from the images*). *This is a massive over-simplification and generalisation, but it does a sufficient job of demonstrating the basic concept.*

A newer approach, facilitated by recent improvements in deep-learning techniques, and likely inspired by the growing prominence of generative AI, is morph creation based on Generative Adversarial Networks (GANs). [d2] GANs are a class of deep learning models that have shown remarkable success in generating high-quality, realistic data in various domains.

GAN-based methods generate an entirely new face image that possess characteristics from both the original images. This is done by sampling the facial images in the latent space of the deep learning network, and synthesising the resulting morph. The resulting images are often more realistic and of higher quality than classical morphing. [d6]

Because of these differences, there was great hope that GAN morphs could be the next SOTA solution for face morphing. Unfortunately, however, it proved to be ineffective in real world trails. GAN Face Morphs whilst solving a lot of issues with classical face morphs, also face their own unique set of challenges and tell-tale giveaways, such as over smoothing of the skin. The 'more realistic' images may fool untrained individuals, however due to the limitations of generative image creation, factors such as lack of small details/imperfections, odd lighting and general uncanny feeling make it very easy for a trained person to spot them. And when it comes to MAD algorithms, they have no more trouble detecting GAN morphs from classical morphs, making GAN morphs ineffective and overly complicated.

There has been some preliminary research [d5] into using Diffusion Models for face morphing, since they're the current SOTA generative model used for Image generation, with applications such as Stable Diffusion gaining increasing popularity outside of traditionally 'techy' circles. Whilst the results look promising thus far, with clear improvements over GAN morphs, the number and quality of the studies is yet too low to reach a definitive conclusion. And there is speculation that diffusion models may need a sizably larger dataset for training as compared to GANs.

2.1.4.2 Morphing Attack Detection

Biometric systems are very prone to spoofing attacks, which is why several anti-spoofing methods are usually implemented to prevent such attacks. MAD, however, is a highly specialised method that targets

only this particular type of presentation attack, and must therefore be used in conjunction with other methods for a comprehensive and robust anti-spoofing system.

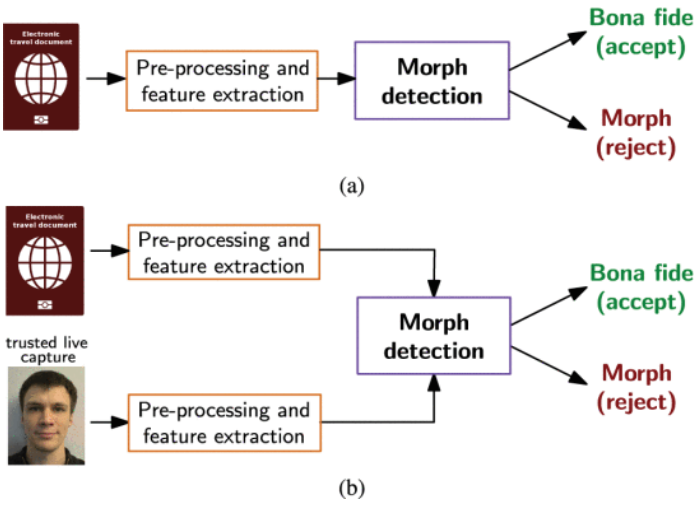
MAD usually involves using various techniques to identify signs of image manipulation and morphing, here are a few examples:

1. Feature analysis - This involves analysing facial features extracted from images. Morphed images may show inconsistencies in facial geometry, such as mismatched eye positions or unnatural blending at the edges of facial features. By analysing these features, detection algorithms can identify anomalies caused by morphing.
2. Texture analysis - Analysing texture patterns within the image. Morphed images show inconsistencies in texture, such as abrupt changes in skin texture or unnatural transitions between different facial regions. Algorithms can be implemented detect such irregularities by analysing texture patterns across the face.
3. Frequency analysis - Due to the nature of blending features together from multiple sources, morphs often introduce high-frequency noise and artifacts into the image. Detection algorithms can search for irregularities in the frequency domain indicative of morphing. [d7]
4. Machine Learning - Using machine learning algorithms such as deep neural networks is currently the SOTA for MAD. If configured and trained correctly, they can learn discriminative features imperceptible to even the most trained human observer, and also outperform other MAD techniques. This approach is highly accurate and shows great promise in lab settings. However due to some limitations with currently existing studies, it is currently still a topic of continuous research and debate.

Humans have been proven to be very bad at morph attack detection, study after study has proven this to be true. *"The observers' success rate in detecting morphing was only 58.3%"* [d8]. As morphs become increasingly complex and convincing it is only going to get considerably harder for manual detection. Therefor it is very clear that using just a human for MAD and by extension, identity verification is unreliable and unsecure.

In the real world, hybrid methods perform better than any singular technique alone. And thus, for applications where security is important, it is standard practice to use a hybrid approach, combining multiple algorithms, machine learning and manual analysis, to ensure the authenticity of a digital image.

2.1.4.3 Single vs Differential Morph Attack Detection [d4]



There are two scenarios of morph attack detection usually considered by research, single (no refence) and differential morph detection. S-MAD algorithms base their classification result solely on the morphed image, e.g. morph detection at the time of passport enrolment when the passport photo is submitted. D-MAD algorithms build upon this by requiring an additional trusted image in its decision-making process, e.g. a live capture at border control to compare to the potential morphed image. D-MAD utilizes the extra information present in the bona-fide sample, allowing it to make a more informed decision, and hence be more successful

Figure 5 - [d1] Morphing attack detection scenarios.
(a) No-reference morphing attack detection.
(b) Differential morphing attack detection.

2.2 A description of our project in the context of existing literature, products, & systems

Morphing Detection is still a highly relevant topic of research, with ongoing discussion and new approaches / ideas constantly being created and studied. Machine Learning using deep neural networks is the current SOTA solution for MAD, and therefor is the approach used in this project. Our project aims to highlight the importance of data quality and suitability in the role of training a morph attack detection classifier.

For the project section of this assignment, we will be exploring S-MAD and not D-MAD. This is primarily due to the incredible level of hands-on experience and time needed to build, train and optimize a D-MAD algorithm in comparison to S-MAD. Despite D-MAD being the SOTA and having more interesting topics to explore, it was simply not feasible within the scope of our undergraduate research.

2.2.1 Existing literature, products, & systems

2.2.1.1 Commercial Facial Recognition / Apple ID

Facial recognition for personal device security has become such an industry standard that even Windows has integrated their own touchless solution “Windows Hello” which automatically detects the users face on wake using the webcam, to sign in without any active user input. A first for consumer PCs which have always historically lagged behind on adopting biometric support.

Currently the most prominent example of commercial facial recognition is Apple’s ‘Face ID’, most people in the west use Apple devices daily, using some version of Face ID to secure said devices. However, facial recognition was not always so ubiquitous and error-free.

Face ID development history summary

Back in 2013 Apple acquired a company called PrimeSense, which had previously worked on Kinect motion sensors for Microsoft. Building upon their prior work, developing low-cost infrared depth perception systems, Apple spent several years conducting R&D before releasing Face ID to the public in 2017.

Competitors like Samsung had the jump start on mobile facial recognition, dating back as far as 2012, 5 whole years before apple. Unfortunately, however, the technology at this stage was still in its infancy, and performance was notoriously bad, with false negatives and false positives commonplace. It was not secure enough to protect a device, and the feature acted as a marketing gimmick.

Apples patience paid off though, when they announced Face ID alongside the iPhone X, it blew its competition out of the water. Instead of taking a simple front facing camera picture of your face, it used an infrared emitter and sensor system to project a large array of dots onto the user’s faces, that could then be read back to accurately map facial geometry. This proved incredibly more accurate than existing solutions, and often even ran faster.

Current latest version of Face ID and how it works.

The current version of Face ID uses a ‘True Depth’ camera alongside a neural engine running locally on the device, this allows it to perform identification better than ever before, automatically adjusting to changes in appearance (scarves, hair, beard, glasses, etc.), head pose and even extreme lighting conditions. It now also has built in anti-spoofing neural networks to process captured biometric samples, and advanced liveness detection which can even check if the person is awake and directing attention towards the device. [a3] [a2]

Innovations Apple helped drive forward.

With the innovations that apple brought, it changed the landscape of personal device, facial recognition, great performance and reliable security became the goal everyone strived for. All major manufacturers soon followed suit and implemented their own high-fidelity high-security biometric measures. Some simply stuck with the formula of Face ID, creating their own versions, whilst others switched to unique and even multi modal biometric solutions.

Apples Face ID solution is ultimately the SOTA when it comes to commercial facial recognition, it easily deals with most presentation attacks thrown its way, works in most conditions, is very convenient, fast, and user friendly. We should look to it as an example, It provides excellently performing biometric evaluation, whilst prioritising and maintaining the user experience.

This is particularly useful as inspiration/guidance to our project, as one of the initial ideas was to implement our MAD solution as part of a comprehensive plug-n-play solution further down the line, providing a seamless experience for companies and their users.

2.2.1.2 Facial Recognition at the Border - Automatic Border Security vs Human Officers

Facial Recognition at the border is currently at the centre of many discussions regarding PAD, MAD, facial recognition algorithms' effectiveness etc. It is the current 'big issue' at hand, attackers are finding ways to create more convincing morphs, whilst also getting better at hiding small signs of morphing like artifacts and textural irregularities. This is allowing them to obtain legally verified documents through the official channels but containing falsified data. Creating a set of documents that can be used by multiple people.

If a criminal on the run finds a roughly similar looking accomplice, they can use a face morphed image of the two on the accomplice's passport, allowing the criminal to roam freely and even flee the country using the accomplice's documents and identity. Whilst this is an extreme example, it's a good demonstration to show how detrimental and far reaching the impact of morph attacks can be, allowing people to circumvent laws and security measures by committing virtually unnoticeable identity fraud.

In 2018, members from an anti-surveillance activist collective in Germany entered a local government office and applied for a passport using a morphed image, and very easily obtained an official German passport without anyone batting an eye. Ironically enough the image was morphed with Federica Mogherini, the High Representative of the European Union for Foreign Affairs and Security Policy. This was just a small part of a larger artwork called "Mask ID" which encouraged ordinary citizens to flood government databases with misinformation to disrupt mass surveillance systems. [d9] In 2020 German officials finally announced they will outlaw the morphing of passport photos.

It has been extensively proven study after study, that human observers cannot effectively distinguish between morphs and real faces, no matter their level of training, be it layperson or experienced border control officer. Automatic Border Control (ABC) gates on the other hand might be the solution, as they already use algorithms to perform identity verification and liveness detection at the border. Some PAD methods are already in place, so implementing a MAD algorithm in these gates should be relatively straightforward, given we create a robust model with good performance.

Within the context of these real-world events, we can inform the direction in which we develop our MAD solution, and what aspects of it we choose to prioritise and why so.

2.2.2 Challenges / Limitations

Large-scale face morph datasets are not publicly available, however having this data is vital for our project, to perform model training and evaluation. To overcome this challenge, we had to create our own large, high-quality dataset of suitable images consisting of both bona-fide and morphed faces.

Decent Liveness detection and Facial recognition algorithms would require depth data alongside RGB data to perform properly. Depth sensors have been SOTA in these solutions for many years now and without having access to depth data, they are a lot easier to spoof. Our dataset unfortunately does not include depth data, therefor we had to acknowledge the limitations of our project and focus solely on MAD rather than build a whole PAD pipeline and facial recognition solution that is sub-par and a decade out of date.

2.3 Critique of the review / Summary

Despite there being numerous papers focused on specific aspects of presentation attacks, and comparing different SOTA methods of morph attack detection, there is little to no literature on how to approach creating a simple morph attack detection algorithm.

This glaring gap in freely available knowledge, acts as a barrier to entry for people without an extensive background in computer vision and AI, such as recent computer science graduates. Having more publicly available, entry level knowledge for beginners drives forward innovation. By allowing more people, from differing backgrounds to contribute unique perspective and ideas, we might find new, outside the box solutions to morph attack detection, fuelling progress in this domain.

This research paper could help bridge that gap, and whilst it’s not fully extensive & all-encompassing, it is a much needed first step towards the right direction for MAD literature.

Chapter 3 - Methodology

all the associated code for this project is included in Appendix 3 at the end of this document.

3.1 Requirements and analysis

For this project we needed to create a machine learning classifier that could accurately detect morphed and bona-fide face images from a large dataset. For this purpose, building a Convolutional Neural Network (CNN) would be ideal, as they are particularly suited for finding underlying & subtle patterns in images. Python is the obvious choice of programming language, as it's the industry standard for ML and data science. It's a high-level language with extensive library ecosystem, and intuitive ETL processes, this lends itself to quicker development and testing cycles, which are key to the iterative nature of ML projects.

There are a quite a few popular open-source ML libraries in python: Keras, TensorFlow, Scikit-learn, and PyTorch, despite having prior experience with the first three, we choose the latter for the purposes of this project. Sci-Kit Learn is very limited in its functionality, it includes some basic NN types but does not allow for the complexity we require. Keras is ultimately a high-level library built upon a TensorFlow backend to provide a better interface, its more user friendly but less capable. TensorFlow and PyTorch are the most powerful libraries, they are both fast, high performance and can handle large datasets.

Ideally, we want to train on a very big dataset, this alongside the large amount of information in image data means model training and testing are likely to take a very long time. This is not ideal since we will need to adjust *parameters, data, etc,* and restart the training process many times throughout our development. Slow training times will stifle the flow of development. To speed this up we need to leverage more compute power, for this we can use CUDA, a software framework developed by NVDIA to expand the capabilities of GPU acceleration [e1]. Due to a range of factors such as the capabilities for parallel processing and faster technical calculations, "GPUs are the dominant computing platform for accelerating machine learning workloads" [e2]. To put it simply, GPUs are faster at training neural networks than CPUs. During early stages of the project, whilst configuring the data pipeline and using a test dataset, switching from CPU to GPU reduced epoch time from 5m30s to a mere 20seconds. This demonstrated how crucial CUDA was for this project.

On paper both TF and PyTorch have CUDA support. However, the latest TF versions (>2.10) do not have GPU support on native windows and the older versions of TensorFlow have dependencies issues with other important libraries such as NumPy. TF is also more suited to production grade projects and large-scale applications. Whereas PyTorch lends itself more towards research and academia, due to its flexible framework which allows experimentation with novel ideas, and rapid prototyping [e3]. This makes PyTorch the only choice that meets our requirements, and as a bonus, its unique strengths are perfectly suited for our project.

3.2 Social Legal and Ethical Considerations

3.2.1 Privacy and Oppressive Authority Concerns [c2]

There are growing concerns about mass surveillance and breach of privacy, whilst a decade years ago most people would've chalked this up to conspiracy theories, recent developments have made it increasingly clear that it could become a problem.

Facial recognition technology has become increasingly better at identifying multiple individuals from a live feed in real time, even when they are facing different directions, have slight obfuscations and non-ideal lighting. When implemented across a dense network of CCTV cameras, it can to a certain degree track and

keep tabs on people. An example of this is the London MET police who use live-recognition CCTV cameras to catch criminals.

Fear of surveillance can cause people out in public to constantly feel watched, and thus feel a pressure to act and behave a certain way. Taken to an extreme, it could heavily impact social dynamics. Instead of genuinely enjoying being outside and living in the moment, people would be doing performative normalcy because they're acutely aware that a camera is capturing their every move, an algorithm is identifying them in it and a large data centre is storing this information long term. It would be a nation-wide panopticon and for many causes anxiety about going outside.

An unfortunate example of this was during the Hong Kong protests in 2019, after months of pro-democracy and anti-authoritarian protests. The oppressive Chinese government bypassed normal legislative procedure to, outlaw facial coverings in public spaces to 'control civil unrest'. However, in hindsight it's become clear it was to identify and punish protestors for speaking out against the government. Protestors found to be wearing facial coverings were subject to attacks from police such as tear gas, rubber bullets and physical violence. In recent years the Chinese government is increasingly developing and implementing facial recognition to track and identify citizens as they move around in public spaces. In the hands powerful people, even with the 'right intentions', if used incorrectly, it can be a dangerous tool.

Imagine if a slightly faulty PAD algorithm running on a certain district's CTTV system incorrectly identified an individual multiple times as wearing a silicone mask in public, if they are lucky, they get unfairly stopped and questioned by the police, if they are unlucky however, and police believe the algorithm without question, an innocent person could be sent to jail or issued a hefty fine.

3.2.2 Bias and Discrimination [f1]

Facial recognition algorithms have been proven to discriminate against certain groups and demographic, raising concerns about ethics and fairness [f2]. They have historically been trained on predominantly white, male and middle-aged facial data. Despite boasting high accuracy rates, these rates drop significantly based on the characteristics/demographics of the user [f1]. This has been a continuous topic of discussion in academia for quite a while, and companies have taken steps to strive for better and more inclusive facial recognition, that doesn't marginalise any demographics.

However, there is little to no research on about the fairness of face presentation attacks, to avoid the same issues that early facial recognition was plagued by it is imperative to ensure our PAD models do not discriminate against any group of individuals by their appearance. There would be massive social backlash if, for example, the ABC gates at an airport turned away 10% of mixed-race people and 20% of people with acne scarring. Not to mention the ethical implications of causing people undue stress and wasting their time.

3.2.3 Security Risks and Legal Compliance

In general, biometric technology continues to function creep into more situations where it just isn't warranted, not only do some people dislike the idea, but it might be becoming a genuine security concern. As more apps pop up that show novel AR filters tracking in real time, it is just a matter of time before a sketchy one is inevitably uploaded to the app store; Downloaded by thousands, then one day they switch on a part of the code that steals everyone's facial biometric data, with a comprehensive 3D mesh of their face using Depth and RGB sensor data from multiple pose angles.

Even if no such app is created, all it takes is one company to practice sloppy data handling or have poor security measures, and an attacker can easily gain access to your facial biometrics; And having multiple companies with unnecessary access to your data simply increases the chances. In the event of an attack,

and password or key can be changed, but your face is your face, and it uniquely identifies you, if that gets compromised, there is currently no solution.

3.3 Dataset Creation and Generation

3.3.1 Obtaining suitable data.

For our project we are developing a MAD algorithm, aiming to expand the current body of research, and bring forth new ideas to the discussion, as the landscape of presentation attacks changes. In this context, the most pressing issues are currently in relation to border control and identity fraud. The images used for these documents adhere to the ICAO standard or similar, and thus the data used for training our machine learning classifier should reflect that.

We need to obtain a set of face images that adhere to important characteristics, such as, front facing, neutral expression, no obfuscations, uniform lighting, open eyes, etc.

The Computer Science Department at UOR provided us with a version of the FERET dataset that had already been decompressed and converted into .png image format using a python script. This dataset was large enough to provide ample training data for our CNN model.

3.3.2 Normalising the Data

We want to normalise the data for two main reasons, firstly we need all images to be of the same size to feed into our neural network, and secondly, we want to remove all irrelevant image data.

The more data we have per image, the more information the neural network has to parse for each item, and thus require a higher dimensional space to store the data points. [g3] Having an unnecessarily high dimensional feature space can cause the curse of dimensionality, this can cause a whole range of side effects.

> Primarily and most concerningly, it leads to data becoming sparser, this makes it challenging to extract meaningful patterns, leading to decreased performance and longer training times.
>
> It can also result in overfitted models which simply learn the data by heart and not spotting underlying patterns, meaning they struggle to generalise and accurately classify data from new sources.

On top of this, having data in our images, that's irrelevant to our classification problem increases the chances of our model identifying unrelated patterns. It could end up training on the wrong features in the image and learn to use the wrong information to make the right classifications. A model like this will perform quite poorly in real world scenarios.

To perform the image normalisation we wrote a script, to perform facial alignment based on facial landmarks using the OpenCV library, and crop all the images to a standard size.

```python
def cutout(ipath):
    global fa
    if isinstance(ipath, str):
        original255 = skimage.io.imread(ipath).astype(np.ubyte)
    else:
        original255 = ipath.copy()
        ipath = '_'
    # original255 = imread(ipath)

    # print(original255.shape)
    if original255.shape[-1] == 4:
        original255 = original255[:, :, :3]

    frame = original255

    print('frame',frame)

    # Run the face alignment tracker on the webcam image
    imagePoints = fa.get_landmarks_from_image(frame)

    print('')
    print('imagePoints',imagePoints)
    print('')

    chipSize = 512
    chipCorners = np.float32([[0,0],
                              [chipSize,0],
                              [0,chipSize],
                              [chipSize,chipSize]])


    if(imagePoints is not None):
        print('processing points')

        imagePoints = imagePoints[0]

        # Compute the Anchor Landmarks
        # This ensures the eyes and chin will not move within the chip
        rightEyeMean = np.mean(imagePoints[36:42], axis=0)
        leftEyeMean  = np.mean(imagePoints[42:47], axis=0)
        middleEye    = (rightEyeMean + leftEyeMean) * 0.5
        chin         = imagePoints[8]
        #cv2.circle(frame, tuple(rightEyeMean[:2].astype(int)), 30, (255,255,0))
        #cv2.circle(frame, tuple(leftEyeMean [:2].astype(int)), 30, (255,0,255))

        # Compute the chip center and up/side vectors
        mean = middleEye[:2] #((middleEye * 3) + chin) * 0.25
        # centered = imagePoints - mean

        upVector    = (chin        - middleEye)[:2] * 1.2

        rightVector =   np.array([ upVector[1], -upVector[0] ])

        # Compute the corners of the facial chip
        imageCorners = np.float32([(mean + ((-rightVector - upVector))),
                                   (mean + (( rightVector - upVector))),
                                   (mean + ((-rightVector + upVector))),
                                   (mean + (( rightVector + upVector)))])

        # Compute the Perspective Homography and Extract the chip from the
        chipMatrix = cv2.getPerspectiveTransform(imageCorners, chipCorners
        chip = cv2.warpPerspective(frame, chipMatrix, (chipSize, chipSize)

    return chip
```

Executed at 2024.03.08 01:10:15 in 1s 997ms

The script then iterates through the FERET dataset file structure, searching for images names ending with '_fa' or '_fb' as these indicate the front facing images in the dataset. After creating a list of suitable images, it iterates through them, loading them in, performing the alignment and cropping, then saving them to the appropriate destination with a suitable file name.

```python
from pathlib import Path
imread = cv2.imread
imwrite = cv2.imwrite

# Set up directory for datasets
src_dir = Path("C:/Users/harip/desktop/facetest/dataset/FERET-dataset/")
dst_dir = Path("C:/Users/harip/desktop/facetest/dataset/FERET-dataset-alig

# Get all image paths (* means "any combination")
image_path_list = list(src_dir.glob("*/*/*_fa*.png"))
image_path_list += list(src_dir.glob("*/*/*_fb*.png"))
print(len(image_path_list))

i = 0

for i in range(len(image_path_list)):

    # load in the unaligned image
    src_img_path = r'{}'.format(image_path_list[i])
    img  = imread(src_img_path)
    print (src_img_path)

    # perform the alignment
    crop = cutout(img)

    # write out the aligned image
    dst_img_path = (str(dst_dir) + '/Aligned_Face_{}'.format(i) + '.png')
    imwrite(dst_img_path, crop)
    print (dst_img_path)

print('All {} images have been aligned.'.format(i))
```

Executed at 2024.03.08 16:14:36 in 7m 50s 897ms

3.3.3 Creating Morphs

We used an off the shelf solution to create the actual face morphs, called "face_morpher", a modified fork of which was provided to us by Dr Jonathan Boyle of our Computer Science department. We made alterations to the scripts to allow them to run properly with updated python libraries, on our system, and with our data.

We then created a "main.py" script to automate the process of creating our morphed images *(code located in Appendix 3)*. It iterates through directory of the aligned face images, morphing together "random" images *(line 35, 38),* and creating "unmorphed" images *(line 44).*

The averager function used to carry out the morphing has its own built-in alignment and cropping functionality, we noticed that all morphs created were more zoomed in on the face area than the bona-fide images from the last step and did not want the model to accidentally learn to differentiate them by that factor, since it's irrelevant to our project.

By running two randomly chosen bona-fide images AND two of the same bona-fide images through the averager function, we can ensure that both the morphed and "unmorphed" images created at each iteration are affected equally by alignment/cropping, thus leaving behind only the image data necessary for training.

3.3.4 Creating the Final Dataset

We wrote a script to automate the creation of our final dataset.

```
import shutil

src_dir_1 = Path("C:/Users/harip/desktop/facetest/dataset/FERET-dataset-morphed/")
src_dir_2 = Path("C:/Users/harip/desktop/facetest/dataset/FERET-dataset-unmorphed/")
dst_dir = Path("C:/Users/harip/desktop/facetest/dataset/finalised-dataset/")

# Get all image paths (* means "any combination")
image_path_list_1 = list(src_dir_1.glob("*.png"))
image_path_list_2 = list(src_dir_2.glob("*.png"))
print(len(image_path_list_1), ' | ', len(image_path_list_2))

# iterates through the list of morphed faces and puts every 5th image into the test set
# all other images go into the train set
a, a1, a2 = 0, 0, 0
for a in range((len(image_path_list_1))):
    src_img_path = image_path_list_1[a]
    dst_img = '/morphed/morphed_face_{}'.format(a) + '.png'
    if (a % 5) == 0:
        shutil.copy2(src_img_path, (str(dst_dir) + '/test/' + dst_img)); a1 += 1
    else:
        shutil.copy2(src_img_path, (str(dst_dir) + '/train/' + dst_img)); a2 += 1
print ('Put {} images in the train set and {} images in the test set'.format(a2, a1))

# iterates through the list of unmorphed faces and puts every 5th image into the test set
# all other images go into the train set
b, b1, b2 = 0, 0, 0
for b in range((len(image_path_list_2))):
    src_img_path = image_path_list_2[b]
    dst_img = '/unmorphed/unmorphed_face_{}'.format(b) + '.png'
    if (b % 5) == 0:
        shutil.copy2(src_img_path, (str(dst_dir) + '/test/' + dst_img)); b1 += 1
    else:
        shutil.copy2(src_img_path, (str(dst_dir) + '/train/' + dst_img)); b2 += 1
print ('Put {} images in the train set and {} images in the test set'.format(b2, b1))
Executed at 2024.03.08 15:54:55 in 1s 630ms
```

It takes the morphed and unmorphed image set and creates a fair train/test split, with every 5th image from both going into the test folder. Within both the train and the test folders there is a folder for the morphs and one for the unmorphed images. This file structure was intentionally set out this way so that later it can be used to infer labels instead of using a dedicated. json label file.

3.3.5 Limitations of our Data

3.3.5.1 Alignment / ICAO compliance

Some of the face images do not come properly aligned, are a few degrees away from “forward facing”, have a slight expression, or wireframe glasses.

Our data normalising steps make sure to align the faces in regard to the facial landmarks, orientation, pitch & yaw, but this does cause some slight warping which is not ideal for preserving the actual facial structure data and reduces image quality.

There is no simple solution to filter out images with non-neutral expression and glasses, however this isn’t a major issue as they comprise a very small fraction of our images.

Whilst the wireframe glasses do not violate ICAO standards and don’t really obfuscate the face for facial recognition, when going through the morphing process they may get unnaturally warped and be an obvious giveaway.

3.3.5.2 Image Quality

The quality of the images also leaves something to be desired, the FERET dataset we are using consists of facial images collected between 1993-1996, they were captured using 30 year old camera technology. Comparing camera sensors from the 1990’s and modern day, it’s clear to see that they are on completely different levels of being capable to capture fine textural details such are pores, which could prove important for training our neural network.

3.3.5.3 Biases in Data / Homogeneous sources

FERET is a large dataset full of diverse faces and collected over 15 sessions over a period of 3 years, taking multiple images of the same people on different days [g1]. This is a better attempt at diversifying factors than most other face databases and was pioneering at the time of its creation. However, in hindsight, we can see there is still some bias present.

There are some demographics that are poorly represented, there are no children and very few young adults or elderly people present, the vast majority of faces are of middle age.

Looking into the sources that make up the FERET dataset, all of them come from research departments of universities in highly wealthy major US cities, this makes it likely to bias the dataset towards people who are us citizens, academically inclined, and are financially well off. [g2]

Whilst on the surface it seems this shouldn’t affect our algorithm, as they still have human faces the same we all do, it ultimately does due factors such as socially acceptable fashion and grooming habits within specific groups, *(for example: most men in this have very short hair and are clean shaven wearing formal or semi-formal clothing)*.

Ultimately this will be the case with almost all face datasets as they are a singular source created by a singular central entity and bias is innately present in all human made things. The only approach to mitigate this problem is to license multiple facial image databases from different sources.

3.3.5.4 Morphing Algorithm Limitations

A singular morphing algorithm is used to create the entire dataset, in the real-world attackers aren’t just going to stick to one solution but rather use different morphing techniques/algorithms, possibly even applying postprocessing to mask signs of morphing. A secure MAD system should be able to detect a morph regardless of what program was used to generate it. Unfortunately, however, we can only evaluate how our model performs with this specific morphing program. It could be possible that the model is overfitted to the specific signs of morphing produced by this algorithm, and would perform more poorly

generalising against other morphing algorithms that cause different artifacts; But, ultimately it is not possible for us to know with our current training dataset.

3.3.5.5 Morph Quality Limitations

The morphs we created are not carefully hand selected. In the real-world attackers will choose similar looking individuals and generate a set of morphs for each chosen pair, allowing them to manually choose the most convincing morphed image. For the purposes of this study, it was not feasible for us to do this, as we needed a large dataset for machine learning, automating this process would be quite extensive, and manually curating each morph would be very time inefficient. There are 1208 people in our FERET dataset, and each person has multiple suitable face images we can use. Due to this our morphs may not accurately reflect the quality of morphs used by attackers, and by extension this impacts our model training.

3.3.5.6 Concluding remarks

From the above-mentioned factors, we can speculate that our model may perform worse against real world morphs than our curated training dataset which has inherent biases & limitations.

3.4 Building the MAD algorithm

3.4.1 Setting up CUDA enabled ML

Our machine learning algorithm had to process a large amount of data, every time we made changes to the model or training parameters, this was very time and computationally expensive. To solve this problem, we used NVDIAs CUDA software. It allowed us to use our NVDIA GPU to perform complex calculations faster and dramatically speed up the training process.

We implemented this by installing the relevant version of CUDA on our pc, downloading CUDA-enabled PyTorch and writing the following lines of code.

```
# making sure cuda support is set up and working correctly
device = torch.device("cuda") if torch.cuda.is_available() else torch.device("cpu")
print(d)
print(f'Using {device} for inference')
print(torch.cuda.get_device_name(0), d)
```

It checks if a CUDA device is present in the system, and if so, sets it as the default. On average, this reduced the inter epoch time from over 3 hours down to just ~6 mins.

```
--------------------------------------------------------------------------------

Using cuda for inference
NVIDIA GeForce GTX 1080

--------------------------------------------------------------------------------
```

Figure 6 - console showing which device is being used for calculations.

```
EPOCH 1:

 2%|          | 1/42 [04:43<3:13:51, 283.70s/it]
```

Figure 7 - non CUDA (CPU only) model-training epoch time

```
EPOCH 3:

100%|██████████| 42/42 [06:45<00:00,  9.66s/it]
```

Figure 8 - CUDA (GPU accelerated) model-training epoch time

3.4.2 Custom Data-Loaders

We wrote Custom Data-Loaders to control how data flows into our neural network. This allows us to pre-process, transform and batch our data so that it's optimal for our task.

3.4.2.1 Checking the Data

The first step before we even begin creating the dataset, is to check our data and that the algorithm is parsing it correctly. These lines of code, allow us to declare the directory location of our dataset, and use the written 'walk_through_dir' function to recursively go through the file structure, then in console return the contents of each folder.

```
There are 2 directories and 0 images in
'C:\Users\harip\desktop\facetest\dataset\finalised-dataset'.
There are 2 directories and 0 images in
'C:\Users\harip\desktop\facetest\dataset\finalised-dataset\test'.
There are 0 directories and 221 images in
'C:\Users\harip\desktop\facetest\dataset\finalised-dataset\test\morphed'.
There are 0 directories and 114 images in
'C:\Users\harip\desktop\facetest\dataset\finalised-dataset\test\unmorphed'.
There are 2 directories and 0 images in
'C:\Users\harip\desktop\facetest\dataset\finalised-dataset\train'.
There are 0 directories and 882 images in
'C:\Users\harip\desktop\facetest\dataset\finalised-dataset\train\morphed'.
There are 0 directories and 452 images in
'C:\Users\harip\desktop\facetest\dataset\finalised-dataset\train\unmorphed'.
```

```python
# Set up directory for dataset and walk through its folders to check
directory = Path("C:/Users/harip/desktop/facetest/dataset/finalised-dataset/")


1 usage  Hari Gupta
def walk_through_dir(dir_path):
    for dirpath, dirnames, filenames in os.walk(dir_path):
        print(f"There are {len(dirnames)} directories and {len(filenames)} images in '{dirpath}'.")


walk_through_dir(directory)
print(d)
```

3.4.2.2 Image/Data transform

Next, we wrote the image transform, which is used later for the generation of the dataset. This resizes all the images to a 300 x 300 standard, randomly flips half of them, and converts them all to a tensor.

Sizing down the images reduces the amount of data per image, and consequently the dimensionality,

```python
# 6. Write transform for image
data_transform = transforms.Compose([
    # Resize the images to 300x300
    transforms.Resize(size=(300, 300)),
    # Flip the images randomly on the horizontal
    transforms.RandomHorizontalFlip(p=0.5),  # p = probability of flip, 0.5 = 50% chance
    # Turn the image into a torch.Tensor
    transforms.ToTensor()  # this also converts all pixel values from 0 to 255 to be between 0.0 and 1.0
])
```

without sacrificing too much quality. The size is also a nicely divisible number allowing us to create appropriately sized kernels, input, and output layers for our model.

Each Image loaded in has 0.5 chance to be flipped, this increases the robustness of our model by teaching it to be able to perform classification even when images are flipped.

In CNNs images are typically represented are tensors rather than raw pixel values. Tensors are used for standardisation *(e.g. differing image file types),* computational efficiency, and batch processing. Therefore, converting our images to tensors is necessary so that they can be processed correctly.

3.4.2.3 Creating Datasets

First, we defined the locations of the train test folders.

```python
# Setup train and testing paths
train_dir = directory / "train"
test_dir = directory / "test"
print(train_dir, "\n", test_dir, d)
```

Then we used the 'ImageFolder' function with the relevant directory locations and *(image)* data transform to create our train and test datasets.

```
# 7. Use ImageFolder function to create dataset(s)
train_data = datasets.ImageFolder(root=train_dir,  # target folder of images
                                  transform=data_transform,  # transforms to perform on data (images)
                                  target_transform=None)  # transforms to perform on labels (if necessary)

test_data = datasets.ImageFolder(root=test_dir,
                                 transform=data_transform)
```

After this, next few lines check that the train and test datasets have been created properly, with correct dataset length and class labels.

```
print(f"Train data:\n{train_data}\nTest data:\n{test_data}", d)

# little check the dataset classes and length is correct
# Get class names as a list
class_names = train_data.classes
print(class_names)
# Check the lengths
print("traindata:", (len(train_data)), "\ntestdata:", (len(test_data)), d)

# 8. index our train and test datasets to find samples and their target labels
img, label = train_data[0][0], train_data[0][1]
```

```
['morphed', 'unmorphed']
traindata: 1334
testdata: 335
```

The final step of creating the dataset, is to index the train and test datasets, so that later we can easily find samples and their corresponding class labels.

```
# 8. index our train and test datasets to find samples and their target labels
img, label = train_data[0][0], train_data[0][1]
```

3.4.2.4 Creating Data Loaders

Creating a data loader is a necessary step when working with a custom dataset, which we are. It helps manage and efficiently load data into our neural network during training and validation, adhering to rules we specify.

```
# 9. Turn train and test Datasets into DataLoaders
BATCH_SIZE = 32  # how many samples per batch?
NUM_WORKERS = 6  # how many subprocesses to use for data loading? (higher = more)
print(f"Creating DataLoader's with batch size {BATCH_SIZE} and {NUM_WORKERS} workers.", d)

train_dataloader = DataLoader(dataset=train_data,
                              batch_size=BATCH_SIZE,
                              num_workers=NUM_WORKERS,
                              shuffle=True)  # shuffle the data?

test_dataloader = DataLoader(dataset=test_data,
                             batch_size=BATCH_SIZE,
                             num_workers=NUM_WORKERS,
                             shuffle=False)  # don't usually need to shuffle testing data
```

In the above lines, we specify the batch size (how many images are in each batch) and the number of workers (subprocesses to load in the batches). The data loaders are then defined using these variables, and the datasets; The train data-loader is also shuffled to remove any unintended patterns in the data order.

3.4.3 Building a Convolutional Neural Network

A CNN is ideal for performing image classification. Compared to other neural networks they have superior performance with image, speech, and audio signal inputs. The connection pattern in a CNN is based on the human brain's visual cortex, with certain receptive fields in visual space causing specific neurons to respond. [e1] This allows them to, **"automatically learn the spatial hierarchies of features, such as edges, textures, and shapes**" [e2]

```
# create / configure the image classifier neural network model
1 usage   Hari Gupta +1
class MorphDetector(nn.Module):
    Hari Gupta +1
    def __init__(self):
        super().__init__()  # instantiate our nn.Module
        self.model = nn.Sequential(
            nn.Conv2d( in_channels: 3,  out_channels: 32,  kernel_size: (30, 30),  stride: 1),  # input layer
            nn.ReLU(),
            nn.Conv2d( in_channels: 32,  out_channels: 64,  kernel_size: (30, 30),  stride: 1),  # first hidden layer
            nn.ReLU(),
            nn.Conv2d( in_channels: 64,  out_channels: 64,  kernel_size: (30, 30),  stride: 1),  # second hidden layer
            nn.ReLU(),
            nn.Flatten(),
            # THE SHAPE COMING OUT OF FLATTEN ALIGNS WITH THE SIZE OF SHAPE EXPECTED IN LINEAR:)
            nn.Linear(64 * (273 - 60) * (273 - 60),  out_features: 1)  # output layer "morph" or "not morph"
        )

    # function to move data between layers
    Hari Gupta
    def forward(self, x):
        return self.model(x)
```

Figure 9 - After conducting research into other PyTorch image classification approaches and many iterations, this is the structure we arrived at for our neural network and were ready to start training.

In a CNN the input layer is always a convolution layer, for this purpose we used Conv2d, which performs the most popular type of convolution. Convolution is the building block of all CNNs and where most of the computation occurs. The input layer accepts images that are 300 x 300 dimensions with a 3-bit colour channel, using a kernel of size 30x30 and stride of 1. The kernel is a filter that passes over the input image, applying a convolution product to output a 'filtered image' [h3].

The next two layers are hidden layers, these are also Conv2D layers with the same kernel size as before. This makes the CNN hierarchical as each corresponding convolution layer breaks down the features further than the layer before it. Making it easier for the NN to interpret and extract relevant patterns.

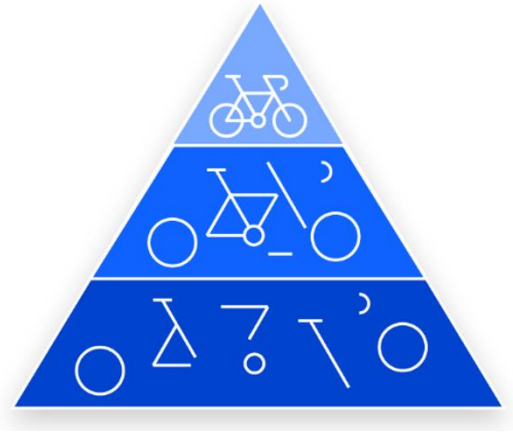


Figure 20 - Visual Example of Feature Hierarchy

The number of out channels is also greater than before, these represent the number of features that the layer can learn. Whilst this does affect the complexity and capacity of these layers, unlike other neural networks, in a CNN it does not directly correlate to the number of neurons present.

In-between all these layers there are ReLU functions, these increase the complexity of the neural network by introducing non-linearity, allowing it to learn more complex representations of the data [h4].

This is important as a linear classifier can only draw a straight line for its decision boundary, whilst this is fine for simple classification problems, our task is actually very complex. Use of a nonlinear activation function allows our NN to draw curved decision boundaries and separate our two classes more accurately.

The ‘nn.flatten()' function takes the shape from the previous layer and flattens it, to align with what the linear layer accepts. The output ‘linear’ layer has a singular neuron, the ‘out_feature’, this is the class prediction.

```
# instantiate our neural network, optimizer, and loss function
md = MorphDetector().to(device)
opt = Adam(md.parameters(), lr=1e-3)  # adam gets trained very quickly, so we can use fewer epochs, approx 10 not 1000
loss_fn = nn.MSELoss()
```

We use MSE loss to calculate the loss values, and the Adam loss optimiser as it reduces our model’s loss very quickly, and we don’t have enough computational power to run 10s or 100s of epochs within a reasonable time frame. The model is also sent to our CUDA device so that it can handle all the calculations faster.

```
# training loop
1 usage   Hari Gupta +1 *
def train_one_epoch(epoch_index, tb_writer):
    running_loss = 0.
    last_loss = 0.

    # Here, we use enumerate(training_loader) instead of iter(training_loader)
    # so that we can track the batch index and do some intra-epoch reporting
    for i, data in tqdm(enumerate(train_dataloader), total=len(train_dataloader)):

        # Every data instance is an input + label pair
        inputs, labels = data
        labels = (labels.to(device).float())  # fixed some cuda stuff

        # Zero your gradients for every batch!
        opt.zero_grad()

        # Make predictions for this batch
        output = md(inputs.to(device).float())
        outputs = torch.squeeze(output)
        loss = loss_fn(outputs, labels)
        loss.backward()

        # Adjust learning weights
        opt.step()

        # Gather data and report
        running_loss += loss.to(torch.device('cpu')).item()
        if i % 1000 == 999:
            last_loss = running_loss / 1000  # loss per batch
            print('  batch {} loss: {}'.format( *args: i + 1, last_loss))
            tb_x = epoch_index * len(train_dataloader) + i + 1
            tb_writer.add_scalar('Loss/train', last_loss, tb_x)
            running_loss = 0.
    return last_loss
```

3.4.4 Training and Testing Loop

This is our training loop, it creates two variables ‘running_loss’ and ‘last_loss’ to store continuous loss values along the duration of the epoch, then return the averaged loss values once the epoch ends. It enumerates through the data loader and for each batch it, creates a data instance of inputs and their labels, zeros optimisation gradients, make predictions, and adjust learning weights accordingly based on performance.

```
for epoch in range(EPOCHS):
    print('EPOCH {}:'.format(epoch_number + 1))

    # Make sure gradient tracking is on, and do a pass over the data
    md.train(True)
    avg_loss = train_one_epoch(epoch_number, writer)

    running_vloss = 0.0
    # Set the model to evaluation mode, disabling dropout
    # and using population statistics for batch normalization.

    # Disable gradient computation and reduce memory consumption.
    # Test the model on the validation dataset
    with torch.no_grad():
        for i, vdata in enumerate(test_dataloader):
            vinputs, vlabels = vdata
            voutputs = md(vinputs.to(device))
            vloss = loss_fn(voutputs, vlabels.to(device))
            running_vloss += vloss
            #print('THIS IS THE VLOSS: ', vloss.item())

    # print out the avg loss for the training and test dataset in this epoch
    avg_vloss = running_vloss / (i + 1)
    print('LOSS : train {} valid {}'.format( *args: avg_loss, avg_vloss))

    # Log the running loss averaged per batch
    # for both training and validation
    writer.add_scalars( main_tag: 'Training vs. Validation Loss',
                        tag_scalar_dict: {'Training': avg_loss, 'Validation': avg_vloss},
                       epoch_number + 1)
    writer.flush()

    # Track the best performance, and save the model's state
    if avg_vloss < best_vloss:
        best_vloss = avg_vloss
        model_path = 'model_{}_{}'.format( *args: timestamp, epoch_number)
        torch.save(md.state_dict(), model_path)

    train_loss.append(avg_loss)
    val_loss.append(avg_vloss)
    train_accuracy.append(avg_loss)
    val_accuracy.append(avg_vloss)

    epoch_number += 1
```

This is the main function of our MAD pipeline, for the number of epochs it loops, calling the training loop, then turning off learning computation and testing the model against the validation data. It logs the average loss per batch for both training and validation data then at the end of each epoch displays them side by side in the console. It also saves the models state when it had the best performance, so that we can use it again for validation and related visualisations in the future without having to re-train it.

3.5 Summary

We took into consideration our requirements, and our social, legal, ethical concerns to inform how we were going to proceed with the implementation of our project, from the dataset creation to the MAD algorithm.

It was important for us to automate as many stages of our project as possible, namely dataset creation, training of our model, and results gathering. Having these functions scripted and compartmentalised allowed us to waste as little time as possible between iterations of our solution.

During the implementation of our CNN, we experimented with various configurations of the network layers, comparing their performance in regard to aspects such as training time and compute resources, proceeding with the model configuration that seemed the most suited to our purposes and computer system.

Chapter 4 - Testing & Results

4.1 Testing our Morphing *(dataset creation)* Processes

Before we implemented the automated pipeline to create our train/test dataset, we needed to manually implement the face_morpher scripts and tinker with them, to understand their functionality and evaluate their effectiveness.

We switched to a different branch of face_morpher that uses dlib instead of stasm for face landmark detection, because the stasm library was causing some dependency conflicts. We also had to use an older version of OpenCV (3.4.18) as the newer versions of the library (4+) were incompatible with the face_morpher scripts. Finally, we also changed the way these python scripts were imported and called from within one another.

After taking these steps the commands to run these scripts were actually starting to work; morpher.py was working using the '—out_frames' command flag, however the quality of morphs produced were less than ideal, there was very noticeable artefacts and ghostly outlines of [head shapes / ears / hair]. The morphs at this stage would easily be detected, even by untrained humans, thereby defeating the whole purpose of creating a NN to detect signs of morphing imperceptible to trained humans.

After reading through the morpher.py script to find the source of these issues, we realised it ultimately calls the averager.py script to perform the bulk of the morphing. Therefor we switched our focus on trying to get the averager.py script working to see if it was the cause of our ghosting issues. Various attempts at getting it to run from the command line didn't work, and despite extensive debugging, the issue was not fixable without rewriting the whole script, so instead we attempted to call it in a similar fashion to how it was in morpher.py.

We created a main.py file that called the averager script directly in python instead of command line, executing it this way showed that the imgpaths were working correctly and loading in the relevant files properly, solving our previous issues. Going from a undefined generator object to the actual image locations stored in the generator object.

```
This is the imgpaths generator
 <generator object list_imgpaths at 0x0000014C7D46EEA0>

This is the imgpaths list
 []
```

Figure 11 - erroneous generator object when running averager.py from command line

```
This is the imgpaths generator
 ['examples\\source\\aligned2.jpg', 'examples\\source\\aligned3.jpg']

This is the imgpaths list
 ['examples\\source\\aligned2.jpg', 'examples\\source\\aligned3.jpg']
```

Figure 13 - working generator object when running averager.py from within another python script

The morphs generated at this stage looked like a convincing middle ground between the two provided faces. We deduced the earlier issues with morpher.py were due to the hair, ears, face edges being deemed as 'background' and occluded from the face averaging, just to be added back in later, and cause those ghostly outlines.

Whilst calling averager.py in main.py directly resolved this issue, it required one of the two images to be assigned a 'background' otherwise it would output a cropped face in a black background, cutting off the ears, hair, etc. The problem with this is, if the two heads look quite different, the face ends up looking more like the one which is chosen for background, likely because hair and head shape are important parts of how human brains perform identification, it is unclear if this will affect facial recognition algorithms in the same way.

For the purposes and scope of our project, the quality of these morphs was quite suitable, and therefore we did not pursue attempts to generate more sophisticated ones.

4.2 Model Efficiency / Training Time Optimisations

Even after using CUDA and our GPU to optimize the model training calculations, it still took a considerable amount of time to complete the full set of epochs. Having slower training would increase the time between iterations, especially at later stages of development; Where we need to wait for the entire program to run before we can evaluate accuracy through data visualisations and improve model performance with hyperparameter tuning.

To solve this issue, we conducted tests where we adjusted batch size and Num workers to observe their effect on performance, focusing primarily on intra-epoch time. Every NN model is unique, and our dataset is novel, because of this there is no standardized method to follow, we simply tried different values, until we found the best performing options.

To create these progress bars, we utilized the tqdm library, which prints a dynamically updating progress bar in real time for our iterate-able function *(every batch in epoch)*.

From this testing we can see that 'num workers 6' & 'batch size 32' yields the lowest training time in our particular scenario. A theoretical explanation is, too little workers result in data taking longer to be loaded, and too many workers mean they can't be executed simultaneously, causing a queue build-up *'traffic'* on the CPU's cores. This is supported by the fact our CPU has 6 cores, and 6 workers seem to be optimal.

4.3 Model Evaluation / Performance Metrics

4.3.1 Confusion Matrix [k1]

The first step in evaluating our model is to check its ability to accurately perform predictions; Since that is the overall goal for this algorithm, and our model is the only part of it performing said predictions. To this end, we created a custom function which took in image label & prediction data, computed a confusion matrix, and then created the corresponding visualisation.

```
198  def conf_matrix(pred, true, num, suffix):
199      # creates a confusion matrix using skikit learn
200      cm = confusion_matrix(true, pred)
```

Figure14 - The function input parameters and computation of confusion matrix

```
202      # creates a pandas dataframe of the confusion matrix
203      df_cm = pd.DataFrame(cm, index=[i for i in class_names], columns=[i for i in class_names])
204
205      # creates a plot for visualisations + uses seaborn to create heatmap on plot
206      plt.figure(figsize=(12, 7))
207      cm_plot = plt.subplot()
208      sn.heatmap(df_cm, annot=True, cmap='Greens', fmt='d', ax=cm_plot)
```

Figure 15 - lines that directly create the confusion matrix visualisation.

To create a visualisation, we created a plot using matplotlib, data-frame to hold the confusion matrix data using pandas; and then used these in conjunction with seaborn's heatmap function to create a colour coded visualisation of the confusion matrix.

```
210      # set label axis
211      cm_plot.set_xlabel('Predicted labels')
212      cm_plot.set_ylabel('True labels')
213
214      # creates name for visualisation and assigns it as plot title
215      cm_title = ('Confusion Matrix - ' + suffix + ' ' + str(num))
216      cm_plot.set_title(cm_title)
```

We added lables along the axes and a title to make it easier to understand what the graph is about. The title name is calculated using the input parameters declared when calling this function.

Figure 16 - code to increase the ease with which a person may understand the visualisation.

```
    # display the heatmap matrix in console and save it as an image in our project files
    plt.show()
    plt.savefig(cm_title+'.png')
```

After this, we display the visualisation in the console, and save it with the same name as the plot title calculated earlier. This makes it easier for us to identify, to which set of predictions *(epoch)* each confusion matrix corresponds, when performing comparisons.

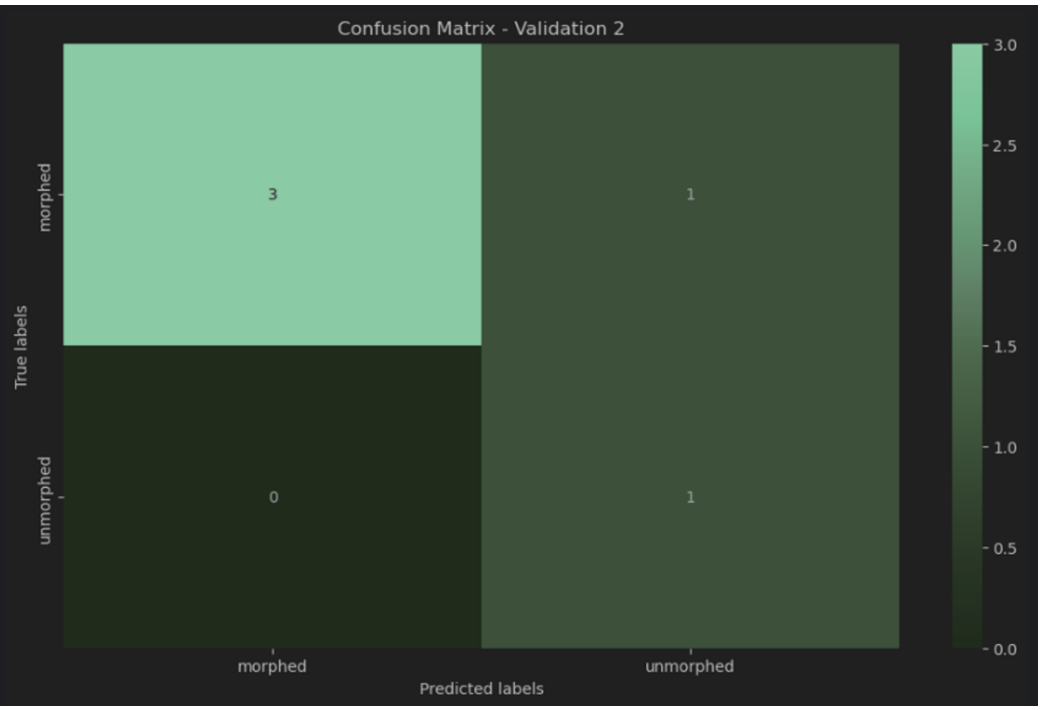


Figure 17 - Confusion matrix visualisation test, performed by training our model on our sample face-morph dataset.

The benefit of creating this visualisation is that, at a quick glance we can see how well our model is making predictions via the heatmap; Whilst also retaining a higher level of detail with the actual prediction numbers, informative plot title, and labelled axes, allowing us to accurately compare confusion matrixes in-depth, at different epochs/stages of training.

4.3.2 TP, FN, FP, TN

The values generated by the confusion matrix, are in actuality the amount of (True Positive, False Negative, False Positive, and True Negative) predictions made by our model. Every time the custom confusion matrix creation function is called in our algorithm, these values are calculated and stored in a local pandas data frame. The values are then extracted from the data frame and appended from here into global variable lists for TP, FN, FP, TN respectively.

```
    # Get the current TP, FP, FN, TN values from the confusion matrix
    # Then add them all into the relevant global list objects,
    TP_temp = (df_cm.iat[0, 0]); TP.append(TP_temp)  # morphs detected as such
    FP_temp = (df_cm.iat[0, 1]); FP.append(FP_temp)  # bonafide detected as morphs
    FN_temp = (df_cm.iat[1, 0]); FN.append(FN_temp)  # morphs detected as bonafide
    TN_temp = (df_cm.iat[1, 1]); TN.append(TN_temp)  # bonafide detected as such
```

Figure 18 - code at the end of the confusion matrix function, that extracts local confusion matrix values into global [TP/FN/FP/TN] arrays.

Each of these variable lists store an array of relevant values, one value for every time the confusion matrix was calculated during the algorithm. This means at the end of training; we can easily access the historical data of our model predictions.

Since we decide to call it during the validation data pass at the end of every epoch, the values we collected are a direct result of the predictions on that specific data, however this is supposed to emulate how our model may perform on unseen data in the real world.

4.3.3 APCER / BPCER [k2] [k3]

APCER and BPCER are evaluation metrics used specifically in the field of biometric identification, to determine the effectiveness of any system related to biometrics. They are computed based on the number of False Positives (FP), True Negatives (TN), True Positives (TP) and False Negatives (FN), as outlined in the ISO standards: **ISO/IEC 30107-3:2023**. [k4]

Since these metrics are standardised, it makes it a lot easier for people developing different solutions, using a variety of libraries, datasets, languages, and models to all directly compare and evaluate model

performance against one other without having to do perform complicated calculations or make too many assumptions.

Since our algorithm is designed to specifically detect face morphs, a “true” or “positive” value indicates a presentation attack is labelled or predicted; Whereas a “false” or “negative” value indicates a bona-fide sample is labelled or predicted, respectively.

ACPER stands for Attack Presentation Classification Error rate, this is essentially the false positive rate. The equation for calculating this is, APCER = FP / (TN + FP). [k5]
This rate illustrates the percentage of presentation attacks which are classified as bona-fide images. A very low rate of this is of utmost importance, because if attackers can easily bypass this MAD and pose as legitimate users, then the security of the whole biometrics system is greatly compromised.

BCPER stands for Bona-fide Presentation Classification Error rate, this is essentially the false negative rate. The equation for calculating this is, BPCER = FN / (TP + FN). [k5]
This rate illustrates the percentage of bona-fide images which are predicted as presentation attacks. A lower rate of this is ideal as having many genuine users unable to prove their identity, may cause frustration and annoyance. However, having lower ACPER is significantly more important than having a low BCPER.

ACER stands for Average Classification Error Rate and is essentially an average of BCPER and ACPER. The equation for calculating ACER is $(ACPER+BCPER)/2$, and it just gives us the overall classification error rate of the model.

We created a function that calculated the APCER, BPCER, and ACER evaluation metrics, using data from the global arrays of TP, FP, FN, TN.
Firstly, the aforementioned arrays are printed out into the console, so that we can check if these values are suitable. This function then iteratively goes through each value in the said arrays, and calculates the evaluation metrics in accordance with the ISO standard, creating a placeholder for null values, and appending these metrics back into global arrays representing their respective metrics. After that, it prints out the lists of evaluation metrics into the console, so we can inspect them for any issues/errors.

```
# function to calculate PAD evaluation metrics
1 usage  new *
def eval_metrics():
    for i in range(len(TP)):
        APCER.append(FP[i] / (TN[i] + FP[i]))
        BPCER.append(FN[i] / (TP[i] + FN[i]))
        # patch to replace values are being divided by 0, with 0 instead of null
        # to make sure metrics still show up on the graph if value is null
        if (TN[i] + FP[i]) == 0: APCER[i] = 0
        if (TP[i] + FN[i]) == 0: BPCER[i] = 0
        # calculate ACER by averaging APCER and BPCER
        ACER.append((APCER[i] + BPCER[i]) / 2)
```

Figure 19 - function to calculate evaluation metrics

We also created an accompanying function that takes these evaluation metrics and creates a visualisation for them, in the form of a graph that shows our model’s classification error rate over time. It plots our evaluation values as lines of differing colours, with a legend, labelled axes, and descriptive title, for ease of understanding.

```python
# create data visualisations to evaluate the model performance / show effect of model training
1 usage  new *
def eval_metrics_graph():

    # plotting evaluation metrics over time (epochs)
    plt.figure(figsize=(12, 7))
    eval_plot = plt.subplot()

    # Assigning the first subplot to graph training loss and validation loss
    eval_plot.plot( *args: APCER, color='b', label='APCER')
    eval_plot.plot( *args: BPCER, color='r', label='BPCER')
    eval_plot.plot( *args: ACER, color='g', label='ACER')
    eval_plot.legend(loc="upper right")

    # set label axis
    eval_plot.set_xlabel('Epochs', fontsize=15)
    eval_plot.set_ylabel('Classification Error Rate', fontsize=15)

    # giving the graph a title
    eval_plot.set_title( label: 'Visualisation of model error rate, over time',
                        fontdict={'fontsize': 24}, pad=12)

    # display the heatmap matrix in console and save it as an image in our project files
    plt.show()
    eval_plot.savefig("model_performance.png")
```

Figure 20 - function to create graph of evaluation metrics changing over time.

This is produced once at the end of our algorithm, evaluating and summarising all of the model's performance. This allows us as the creator of this CNN to spot patterns in the way it learns, equipping us with necessary knowledge to make adjustments and improve the model so that it can perform better.

Figure 10 - Graph visualising our model's classification error over successive epochs, using APCER, BPCER, ACER values derived from model predictions made on the validation data in the sample dataset

Ideally, we would like to see a general trend of the error rates decreasing over time, as this would indicate that our model was making less mistakes as time went on; From this we could deduce it was iteratively learning how to better differentiate between morphed and bona-fide facial images, therefor achieving our goal of MAD.

4.4 Model Adjustments & Hyperparameter tuning

Once the confusion matrix visualisations were working correctly, we were getting accurate TP, TN, FP, FN values on our model's predictions, and visualising the evaluation metrics on our sample dataset; We switched back to using the actual dataset *(see section 3.3.4)* to see the performance of our model in action. The confusion matrixes and performance over time graphs didn't show a general trend over time, which indicated a lack of effective learning occurring within our model. Because of this we investigated aspects of our model structure and configuration, focusing primarily on the effects these had on our model's predictions.

At first, the direct prediction values from the NN's output were largely nonsensical, and varied wildly in their range, from values like 79324.13032 to -0.003145.

Since we are performing a binary classification task, we added a sigmoid activation function onto the model output [j1]. This returns a value from 0 to 1 and is ideal for our needs as we are predicting a binary outcome. With 1 representing true (morphed face), 0 representing false (bona-fide face), and every float value in between being a probability between the two.

After that step, the predictions were more consistent, however after a few batches, the list of predicted values per batch would consistently start converging towards 0, these batch predictions below are just from the first epoch; the subsequent epochs only carried on getting worse.

```
EPOCH 1:
batch  0 : [0.49774292 0.49754837 0.4968505  0.4987098  0.49793723 0.49768323
 0.4985155  0.49734095 0.49810782 0.4972531  0.49709085 0.4971966
 0.49773207 0.49799967 0.49537903 0.49645624 0.4961793  0.49753055
 0.49675417 0.49615067 0.4968679  0.49937102 0.49676457 0.49652153
 0.49698466 0.49500442 0.49810025 0.49908602 0.49741456 0.49961114
 0.49709517 0.4981303 ]
batch  1 : [0.06729472 0.10845639 0.03443147 0.10902245 0.071468   0.08321501
 0.0709294  0.06598989 0.13399388 0.05294151 0.07550569 0.12990376
 0.07569423 0.08289456 0.0549831  0.05734738 0.06896412 0.08237242
 0.10192448 0.09115548 0.06633251 0.06890321 0.13560334 0.14089592
 0.04477395 0.08201232 0.06194538 0.11668953 0.05518637 0.1209356
 0.04989243 0.06460515]
batch  2 : [0.5036235  0.5059551  0.5051016  0.50898594 0.5047537  0.51011646
 0.5066442  0.508791   0.504514   0.5087393  0.5053921  0.507909
 0.5077496  0.50865203 0.50941205 0.5066905  0.50712955 0.509513
 0.5072465  0.5072219  0.5070046  0.50843763 0.50752515 0.50585186
 0.5050399  0.505945   0.5082934  0.50908995 0.50628096 0.5108043
 0.50493157 0.50318724]
batch  3 : [3.2542464e-03 7.8150420e-04 8.9925400e-04 5.9705675e-05 2.0722744e-04
 2.8666845e-04 5.1024828e-05 2.1427271e-03 9.0324909e-05 1.3431406e-03
 1.3422276e-04 1.3266994e-03 1.6771637e-04 2.5273217e-05 2.9814205e-04
 1.6023855e-03 1.1725719e-04 7.8671379e-05 3.8178216e-05 7.0044096e-03
 3.7241072e-05 1.2152339e-03 2.0262820e-05 3.3233548e-04 5.4271663e-05
 4.9610401e-04 1.8128926e-04 3.0575966e-04 1.5759592e-04 2.8021286e-05
 3.3731434e-05 3.6286324e-04]
batch  4 : [0.77047217 0.7741103  0.7801023  0.8187425  0.76766646 0.77993613
 0.7991986  0.774919   0.79232246 0.7652234  0.8035704  0.77597445
 0.8083853  0.77063906 0.804898   0.769138   0.77814245 0.7714044
 0.8598953  0.76568484 0.79355913 0.76855916 0.79457086 0.75563467
 0.7621145  0.76962143 0.7839299  0.82203186 0.76302975 0.7685509
 0.8077828  0.78616065]
```

```
batch  37 : [8.909628e-25 9.029727e-25 9.000290e-25 9.022634e-25 8.749741e-25
 9.055113e-25 9.006987e-25 9.062371e-25 9.063684e-25 9.069425e-25
 9.053940e-25 9.051971e-25 9.047000e-25 8.987115e-25 9.060089e-25
 9.059537e-25 9.057187e-25 9.062302e-25 9.053663e-25 9.016853e-25
 9.021774e-25 9.019399e-25 9.060539e-25 8.848931e-25 9.068284e-25
 8.819914e-25 9.020018e-25 8.596396e-25 9.003758e-25 9.047277e-25
 9.046344e-25 9.062821e-25]
batch  38 : [9.069425e-25 9.038928e-25 9.064757e-25 9.043964e-25 8.890649e-25
 9.068527e-25 9.003723e-25 9.057394e-25 8.905517e-25 8.995038e-25
 9.018539e-25 9.057808e-25 9.012004e-25 8.783584e-25 9.008430e-25
 9.032482e-25 9.064134e-25 9.042204e-25 9.031726e-25 9.063650e-25
 9.055598e-25 9.003827e-25 9.029899e-25 8.848761e-25 9.056081e-25
 9.068457e-25 8.998194e-25 8.946172e-25 9.007983e-25 8.822202e-25
 9.021050e-25 8.758190e-25]
batch  39 : [9.042343e-25 9.041031e-25 8.983584e-25 8.935701e-25 9.016853e-25
 8.946854e-25 9.013551e-25 9.067177e-25 8.942930e-25 8.956928e-25
 9.068423e-25 9.012692e-25 9.049312e-25 9.066866e-25 8.993975e-25
 8.484941e-25 9.065690e-25 9.057532e-25 9.032448e-25 9.066866e-25
 9.017025e-25 9.056979e-25 9.046206e-25 9.066520e-25 9.057463e-25
 9.002006e-25 8.847074e-25 9.059744e-25 9.045343e-25 9.023357e-25
 9.058465e-25 8.937610e-25]
batch  40 : [9.0538700e-25 9.0343773e-25 9.0352746e-25 9.0172314e-25 9.0624755e-25
 9.0031739e-25 8.9426909e-25 8.7449688e-25 9.0270408e-25 8.7108762e-25
 9.0515567e-25 9.0121758e-25 9.0685616e-25 9.0057840e-25 8.8978757e-25
 9.0508319e-25 9.0434117e-25 9.0453434e-25 9.0560128e-25 9.0080855e-25
 8.9871837e-25 8.7875052e-25 9.0366876e-25 8.9961707e-25 9.0334819e-25
 9.0219113e-25 9.0695299e-25 8.9444649e-25 9.0141361e-25 9.0572907e-25
 9.0462762e-25 8.9859836e-25]
batch  41 : [9.050418e-25 8.926400e-25 9.059225e-25 8.873099e-25 8.946854e-25
 9.008980e-25 8.893533e-25 9.066381e-25 8.939894e-25 9.010045e-25
 9.067489e-25 8.960003e-25 9.051626e-25 8.847378e-25 9.028383e-25
 9.037859e-25 9.055806e-25 8.928069e-25 9.062924e-25 9.011455e-25
 8.909255e-25 8.883293e-25]
```

To deal with this prediction drift, we examined the processes of training loop ‘train_one_epoch’, we can see that for each batch, it is passed forward through the NN, losses calculated, then propagated back through the network to adjust the weights using our optimization function SDG. [j3]

Since the weights are optimised after each batch, based on the error values, the loss function was the first place we investigated as potential cause of the problem. Previously, we were using a MSE loss function, but after some research it became clear that using Binary Cross Entropy loss was the suitable choice for our task of binary classification [j1]. We switched to BCELoss and normalised the prediction shapes for it to perform the calculations correctly [j2]. This had the effect of slowing the rate of convergence, meaning that there was a more even spread of predictions for morphed and ‘unmorphed’ *(bona-fide)* than before. Considering the confusion matrixes & our prior knowledge of the dataset, we can observe that whilst this does not completely solve the problem of convergence, it is a step in the right direction.

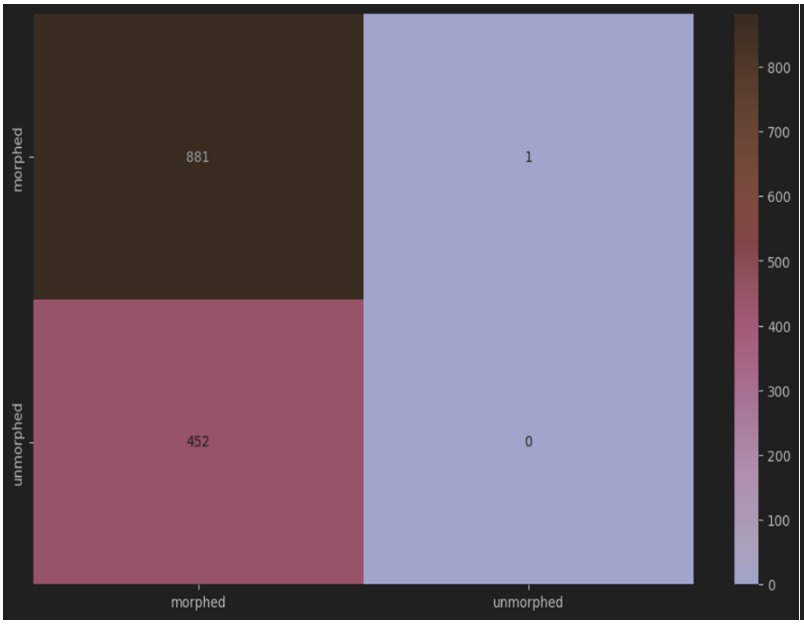


Figure 22 - Epoch 1 heatmap when using MSE loss function

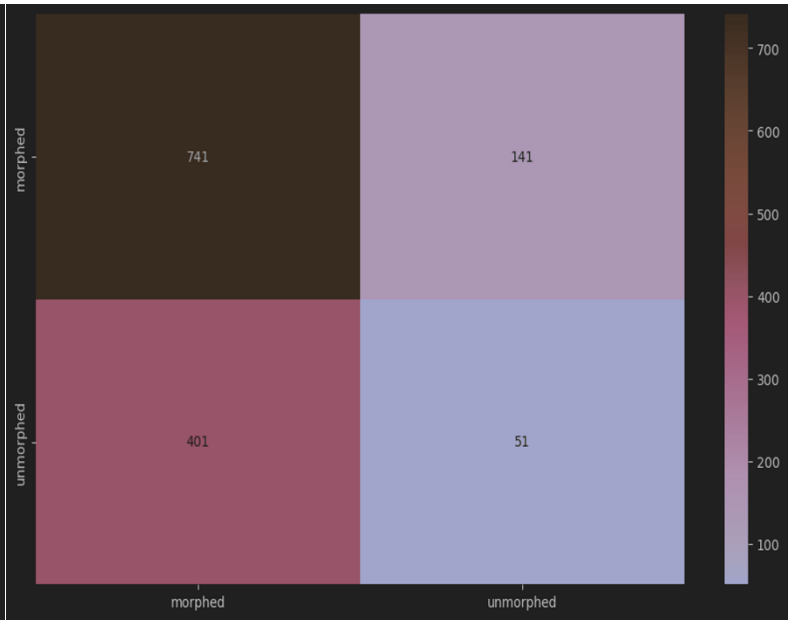


Figure 23 - Epoch 1 heatmap when using BCE loss function

The next aspects we reviewed, were the activation functions in-between the hidden layers of our neural network. [j4] We experimented with many different combinations of activation layers, such as pooling layers, softmax, tahn, etc. however these did not make any noticeable positive changes to way our model trained. Because of this we reverted back to our original implementation of using ReLU functions between each layer to produce non-linear decision boundaries.

As a final attempt to fix our model's performance, we implemented changes to our dataset and the way it was being used to train the model.

By looking at patterns in confusion matrixes produced by previous models, we saw that as training went on, models tended to guess an increasing number of samples as being morphed. We deduced that even though it seemed unable to learn features of the images to perform our classification as intended, it did somehow learn that there were more morphed than bona-fide face images present in our dataset. By the final epoch in a training cycle, It always predicted every image as morphed since that was the only pattern it was able to learn about the data, which made it more accurate than random guessing simply due to the unbalanced nature of our dataset.

This lack of balance in our dataset was an unintended consequence of the script we created to generate the dataset, *(see section 3.3.3)*, it created two morphs for every bona-fide image. To stop the number of samples in each class having an impact on the model's learning, we manually curated a new 'balanced dataset' that had an equal number of samples per class, 100 each for training and 50 each for validation.

We altered the model structure and image transforms used in the data loader so that our images were no longer being down sampled from 512x512 to 300x300. We did this to keep the full quality of the original images, to retain small details that the model may use to learn, such as artifacts and textural irregularities caused by morphing. This greatly increased our training time, however it was still worth attempting, as in a real-world implementation, this won't be an issue. The MAD model will already be trained before deployment and can be used countless times without needing to be trained again, if at all.

```python
# 6. Write transform for image
data_transform = transforms.Compose([
    # Resize the images to 300x300
    #transforms.Resize(size=(300, 300)),
    transforms.Resize(size=(512, 512)),
    # Flip the images randomly on the horizontal
    transforms.RandomHorizontalFlip(p=0.5),  # p = probability of flip, 0.5 = 50% chance
    # Turn the image into a torch.Tensor
    transforms.ToTensor()  # this also converts all pixel values from 0 to 255 to be between 0.0 and 1.0
])
```

Figure 11 - changed image transform resizing

```python
1 usage  Hari Gupta *
class MorphDetector(nn.Module):
    Hari Gupta *
    def __init__(self):
        super().__init__()  # instantiate our nn.Module
        self.model = nn.Sequential(
            nn.Conv2d( in_channels: 3,  out_channels: 32,  kernel_size: (32, 32),  stride: 1),  # input layer
            nn.ReLU(),
            nn.Conv2d( in_channels: 32,  out_channels: 64,  kernel_size: (32, 32),  stride: 1),  # first hidden layer
            nn.ReLU(),
            nn.Conv2d( in_channels: 64,  out_channels: 64,  kernel_size: (32, 32),  stride: 1),  # second hidden layer
            nn.ReLU(),
            nn.Flatten(),  # Shape coming out from flatten, aligns with shape expected in Linear Layer
            # parameters of linear layer is (in_neurons * (image height or width * 2), out_neurons)
            nn.Linear(64 * (419**2),  out_features: 1),  # output layer "morph" or "not morph"
            nn.Sigmoid()  # Sigmoid activation function to get an output between 0 and 1
        )

    # function to move data between layers
    Hari Gupta
    def forward(self, x):
        return self.model(x)
```

Figure 12 - altered model input and output shape to accommodate the larger image sizes

Ultimately, despite all the different avenues which we explored, nothing was able to fix the model's ability to learn, and therefore we must accept that this model may not be the right one for our classification task.

4.5 Results Summary

The face morphing processes were tested in depth, to understand how they functioned. Ensuring that we were aware of the limitations of our data, and that high quality, normalised data was being produced for our dataset.

Trial and error testing was conducted on how data was batched up and loaded into our neural network, to find the optimal settings of batch size 32 with 6 workers.

Relevant performance / evaluation metrics were created using TP, FN, FP, TN prediction values at each epoch, allowing us to understand how the model was learning over time, and inform adjustments made to increase model performance.

After several iterations of our model, it became clear that our approach to creating and training a model capable of MAD was unsuccessful. There was either a problem with our data or the model configuration itself, however we dove deep into checking and adjusting both of these aspects and conducted extensive testing; *see Appendix 1 for some examples*. This leads us to believe that a successful implementation would require immense prior knowledge and expertise, which is out of our grasp as undergraduate students.

Chapter 5 - Discussion and Analysis

5.1 Discussion and Analysis of the results

From our findings we discovered that a neural network of this complexity is simply not sufficiently capable of performing the task of face-morph attack detection.

We conducted extensive testing and attempted various fixes on a variety of factors, such as activation functions, loss functions & learning rates, data handling & balancing, model structure & conv2D kernel parameters. However, our model was still plagued with prediction drift, often causing prediction values to converge towards one extreme of the sigmoid function. Despite our best efforts, and weeks of trial and error, our model behaved in unpredictable ways at best, and was completely unable to learn / train in most cases.

These findings indicate that extensive expertise in the fields of machine learning, computer vision, and biometric systems are all required to build a comprehensive morph attack detection system. And a relatively high level of prior experience / knowledge is required to even create an algorithm that can make predictions better than random chance.

5.2 Significance of the findings

These findings are important because they relate back to the section 1.1 of our introduction; Highlighting and further proving the point that, a lack of intermediate literature on PAD methods results in a massive knowledge gap, which acts a barrier of entry to those without prior master's level education or research experience in the field.

This is especially significant for people like us, with computer science undergraduate degrees, who may be interested in these specialised fields, but cannot justify / afford getting a master's degree. Having free access and publicly available knowledge about this topic, *(and other topics in similar situations)*, will ultimately lower the barrier to entry and drive forward innovation, speeding up technological progress, and benefitting humanity in the long term.

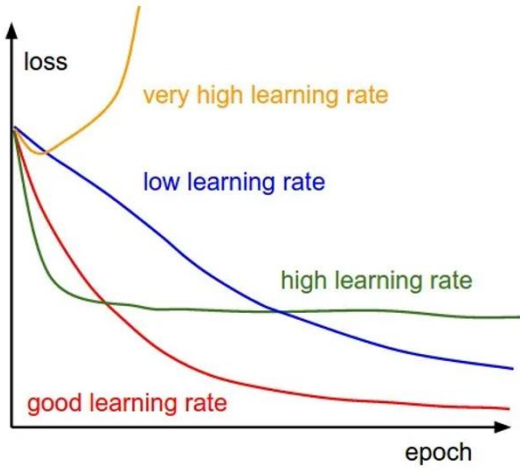
By extension these points are also relevant for the average layperson in public, as they are participants of society, and would benefit from a safer society. Many criminals use face morphing to evade being identified, to escape prosecution and bypass border security to flee the country so they can't be found and rehabilitated. Having better PAD detection allows the authorities to catch people committing identity fraud, keeping them from committing further crimes and holding them accountable for their actions.

5.3 Limitations

Our findings and results were greatly limited by our model's inability to learn during training, listed below are some of the potential 'limitations' that caused these issues, and their implications:

- Data set related:
    - Our facial image data may not have been of high enough quality (i.e. resolution, lighting), especially since it was captured over 20 years ago using outdated equipment.
    - Perhaps the facial alignment algorithm performed on every image, leaves more artifacts than the morphing algorithm, making the image data noisy enough to mask the effects of morphing.
    - Maybe our dataset is not large / complex enough to properly train the dataset, SOTA facial recognition neural networks are trained on tens of thousands of face images with varying angles for each face, our dataset on the other hand after balancing has only 300. Whilst facial recognition and MAD are not the same, it still serves as a meaningful comparison since they are usually used in conjunction.

- Model configuration related:
    - Perhaps our model is not complex enough, lacking enough layers or neurons, to learn the hierarchical image features that it could use to perform the classification. Therefore, it struggles to find adequate patterns in the data.
    - Our loss function in conjunction with its learning rate could be causing our model to learn in the wrong way. [l1] Having too high a learning rate can cause the weights to either overshoot the optimal point or diverge and produce large errors. Having too low a learning rate means more time is needed to train the model. Either way both extremes result in wasted time and compute power, especially for us since we are training using a mid-range pc from 5 years ago, not a supercomputer with tens of GPU blades.



        - From our current testing and understanding of our model, this is one of the most likely culprits behind our issues; Exacerbated by a smaller dataset, and slow learning times on our hardware, which make it unfeasible to iteratively tweak and re-run the model with more than 5 epochs at a time.

Chapter 6 - Conclusions and Future Work

6.1 Conclusions

The main objective of this project and report was to create a detailed exploration into morph attack detection and discuss the wider context surrounding it. To this end, we had four main aims, to write a detailed literature review on the subject matter, create our own face morphs dataset to understand the process of attackers, create a MAD algorithm detailing & justifying the choices made along the way, and evaluate our project via the appropriate metrics & discussing our overall approach.

Our literature review painted a broad picture of the landscape of modern facial recognition, it also successfully detailed the issues that these systems face from attackers, particularly focusing on morph attack detection, and the real-world scenarios in which it is currently being exploited / fought against.

Creating the face morphs for our training validation datasets, allowed us to gain insight into how exactly attackers may create morphs, what tools are required, and what the limitations of publicly available morph creation software are. We evaluated every stage of the process to ensure our face morphs were of high quality. The bona-fide images were also normalised in the same fashion, ensuring there were no extra image features the model may accidentally use to learn.

We attempted to create a morph attack detection algorithm that would outperform humans in identifying morphs, however getting our model to learn relevant image features during training proved unsuccessful. We attempted various methods, fixes, and overhauls in an attempt to improve the model's performance, but it was fruitless. Whilst the data pipeline and the rest of the algorithm worked perfectly, our neural network was simply not suitable for this classification task.

For the evaluation we created easy to understand, detailed visualisations, showing the confusion matrixes on validation data at every epoch, as well as a graph showing the classification error rates over time. This data was what ultimately allowed us to spot and diagnose the problems with our model's training. We also concisely discussed the factors surrounding our overall approach and the impacts of our results, outside of just the technical scope. Namely, that the knowledge gap in this field is halting potential progress / innovation, and why this impacts our lives as people living in technological societies.

6.2 Future Work

Initially one of the main aims of this project, was to create a MAD algorithm that would outperform human prediction, however as this project progressed, it became clear that this was significantly more complex than we had anticipated, and we were ultimately unable to create a model capable of learning the difference.

In the future, given more time to iterate and test, we could try building many differing approaches to a MAD CNN. Comparing them alongside one another on the same data, to determine how each solution performs and why. And use that information gathered to combine the most successful aspects of each approach and create an effective solution moving forward. We have learned however, to experiment with the more complex neural networks needed for this, we would require access to significantly more powerful hardware, as iterating and re-training our models would be prohibitively slow otherwise.

During the conception of this project, we played around with the idea of creating a plug-and-play solution for facial recognition that focused on PAD. The plan was to integrate existing solutions for liveness detection and face recognition with our own MAD algorithm to create a comprehensive and robust solution to tackle vision-based presentation attacks. However, it was not feasible to spend our limited project time, on a lower priority, optional function. Once a successful MAD algorithm is created however, it would be relatively simple to implement the other components into a singular solution, since they have already been created by third parties and we just simply need to integrate them together; using a easy to use library OpenCV, to handle capturing and processing of image data.

Chapter 7 – Reflection

The biggest challenge we faced during this research project, was getting our MAD solution to accurately make predictions on which images were morphs or bona-fide. Despite exploring multiple avenues and delving deep into the potential causes behind this, we were unable to find a singular straightforward solution. We kept running into the same conclusion, our CNN was unfit for the purpose, and we didn't have the expertise required to make a NN that worked.

In response to this, we were forced to change the narrative around our results and conclusion, as we were no longer able to evaluate how our basic MAD algorithm compared to humans in other similar trails.

This taught me a very important lesson, the importance of accepting when research deviates from the intended outcomes. And how we must take our findings as they are, presenting them in an unbiased light, without letting the original aims cloud our judgement and subsequent conclusions we draw.

In the future if a similar problem where to present itself, we would react differently in how we handle it. Instead of fixating on the expected outcome and spending too much time and resources trying to brute force a solution; We would be more vigilant to notice when project progress is slowing down, take a step back, and compare the intended flow of the project to the current findings/narrative. This would allow us to adjust our expectations, and plan how we will proceed moving forward, making sure we take both the bigger picture and the active task into reasonable consideration.

7.1 Project deviation from initial PID

In our PID we outlined a few key features that we wanted our project to contain, these could be boiled down to, Live identity verification, presentation attack detection, and Image quality checks. For this we intended to use off the shelf solutions for, facial recognition, image quality assessment, and liveness detection, implemented together with our own original PAD algorithm. To form one comprehensive solution, that could capture biometrics samples from a live webcam and use that data to perform a comprehensive and robust identity verification.

However, as the research was being carried out, it became clear that the scope of our project was too large, and we had to trim the parts less integral to our core research. Ultimately focusing solely on building a comprehensive MAD pipeline, and thoroughly documenting it in every stage, designating the integration of other related solutions as optional extras.

We focused in on details involved in morph attacks, and even went as so far as to create our own face morphs through python scripts, allowing us to better understand the steps behind an attack.

The algorithm was also unable to perform the image classification task required from us, so we considered out goals again. And linked it back to one of the original motivations of our project, the lack of intermediate knowledge available on morph attack detection, and how it stifles progress / innovation by acting as a barrier of entry to those without a specific education background.

References

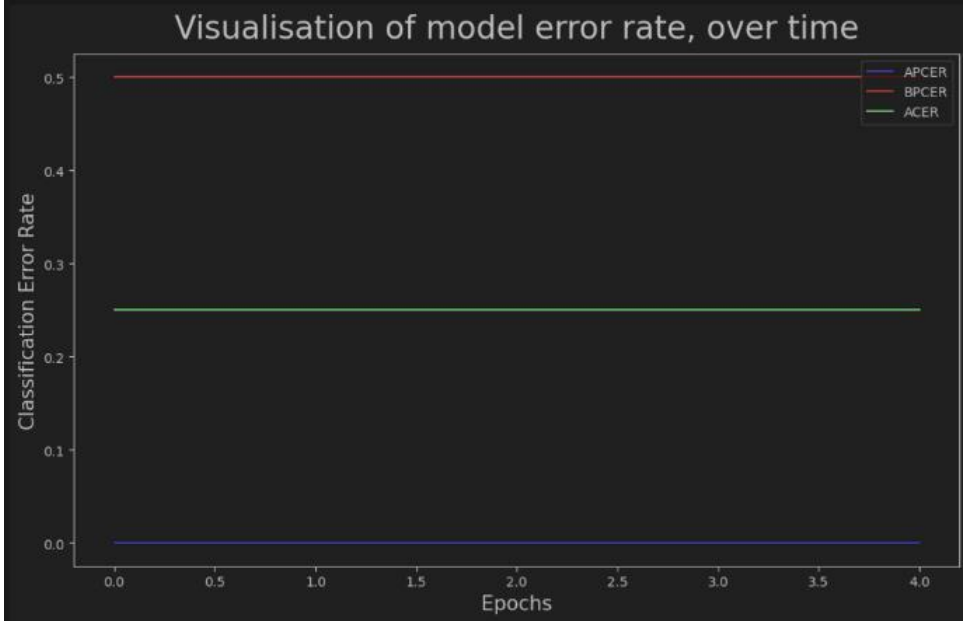
https://www.mdpi.com/1424-8220/20/2/342 [c1]
https://www.tandfonline.com/doi/full/10.1080/17439884.2020.1686014 [c2]
https://www.researchgate.net/publication/351952209_The_future_of_biometrics_technology_from_face_recognition_to_related_applications [c3]
https://ieeexplore.ieee.org/abstract/document/1240842 [c4]
https://www.incognia.com/the-authentication-reference/what-is-liveness-detection [a1]
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6515036/ [a2]
https://support.apple.com/en-gb/102381 [a3]
https://pages.nist.gov/ifpc/2022/presentations/3_2022-11-07_OFIQ_SOTA.pdf [a4]
https://arxiv.org/pdf/2009.01103.pdf [a5]
https://dl.acm.org/doi/abs/10.1145/3038924 [b1]
https://plaid.com/blog/presentation-attack-id-verification/ [b2]
https://antispoofing.org/presentation-attacks-types-instruments-and-detection/ [b3]
https://ieeexplore.ieee.org/abstract/document/8642312 [d1]
https://ieeexplore.ieee.org/abstract/document/9380153 [d2]
https://www.sciencedirect.com/science/article/pii/S2214212619302029 [d3]
https://researchrepository.wvu.edu/cgi/viewcontent.cgi?article=11202&context=etd [d4]
https://arxiv.org/pdf/2301.04218.pdf [d5]
https://arxiv.org/pdf/2205.02496.pdf [d6]
https://dl.acm.org/doi/pdf/10.1145/3335203.3335721 [d7]
https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=9997091 [d8]
https://www.vice.com/en/article/pa9vyb/peng-collective-artists-hack-german-passport [d9]
https://www.turing.com/kb/understanding-nvidia-cuda [e1]
https://blogs.nvidia.com/blog/why-gpus-are-great-for-ai/ [e2]
https://opencv.org/blog/pytorch-vs-tensorflow [e3]
https://sitn.hms.harvard.edu/flash/2020/racial-discrimination-in-face-recognition-technology/ [f1]
https://www.sciencedirect.com/science/article/abs/pii/S0031320323007008 [f2]
https://libguides.princeton.edu/facedatabases [g1]
https://www.nist.gov/programs-projects/face-recognition-technology-feret [g2]
https://www.datacamp.com/blog/curse-of-dimensionality-machine-learning [g3]
https://www.bbc.co.uk/news/world-asia-china-49931598 [g4]
https://theworld.org/stories/2019/10/03/hong-kong-set-ban-face-masks-and-other-enact-emergency-laws-it-struggles-contain [g5]
https://datagen.tech/guides/computer-vision/cnn-convolutional-neural-network/ [h1]
https://datagen.tech/guides/image-classification/image-classification-using-cnn/ [h2]
https://stackoverflow.com/questions/75192505/why-relu-function-after-every-layer-in-cnn [h3]
https://towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148 [h4]
https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8 [j1]
https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html [j2]
https://towardsdatascience.com/optimizers-for-training-neural-network-59450d71caf6 [j3]
https://www.v7labs.com/blog/neural-networks-activation-functions [j4]
https://christianbernecker.medium.com/how-to-create-a-confusion-matrix-in-pytorch-38d06a7f04b7 [k1]
https://www.idrnd.ai/presentation-attack-detection/ [k2]
https://publications.idiap.ch/downloads/papers/2018/Chingovska_SPRINGER_2019.pdf [k3]
https://www.iso.org/standard/79520.html [k4]
https://chalearnlap.cvc.uab.cat/challenge/33/track/33/metrics/ [k5]
https://towardsdatascience.com/https-medium-com-dashingaditya-rakhecha-understanding-learning-rate-dd5da26bb6de [l1]

Appendix 1 - Model Performance at differing model configurations

Model parsing 512x512 image data, using Adam optimiser, with learning rate of 0.01

```
1 usage   Hari Gupta *
class MorphDetector(nn.Module):
    Hari Gupta *
    def __init__(self):
        super().__init__()  # instantiate our nn.Module
        self.model = nn.Sequential(
            nn.Conv2d( in_channels: 3,  out_channels: 32,  kernel_size: (32, 32),  stride: 1),  # input layer
            nn.ReLU(),
            nn.Conv2d( in_channels: 32,  out_channels: 64,  kernel_size: (32, 32),  stride: 1),  # first hidden layer
            nn.ReLU(),
            nn.Conv2d( in_channels: 64,  out_channels: 64,  kernel_size: (32, 32),  stride: 1),  # second hidden layer
            nn.ReLU(),
            nn.Flatten(),  # Shape coming out from flatten, aligns with shape expected in Linear Layer
            # parameters of linear layer is (in_neurons * (image height or width * 2), out_neurons)
            nn.Linear(64 * (419**2),  out_features: 1),  # output layer "morph" or "not morph"
            nn.Sigmoid()  # Sigmoid activation function to get an output between 0 and 1
        )

    # function to move data between layers
    Hari Gupta
    def forward(self, x):
        return self.model(x)


# instantiate our neural network, optimizer, and loss function
md = MorphDetector().to(device)
opt = optim.Adam(md.parameters(), lr=0.01)  # adam gets trained very quickly, so we can use fewer epochs
```

```
The model has finished training!

The model state was saved at the lowest loss rate: 50.0

100%|██████████| 50/50 [08:39<00:00, 10.39s/it]
100%|██████████| 50/50 [08:46<00:00, 10.54s/it]
100%|██████████| 50/50 [08:56<00:00, 10.74s/it]
100%|██████████| 50/50 [08:53<00:00, 10.68s/it]
100%|██████████| 50/50 [08:56<00:00, 10.73s/it]

Here are the conf matrix values:
 [50, 50, 50, 50, 50] [0, 0, 0, 0, 0] [50, 50, 50, 50, 50] [0, 0, 0, 0, 0]
```



Model parsing 512x512 image data, using SGD optimiser, learning rate of 0.0001

```
158
    1 usage   Hari Gupta *
159 class MorphDetector(nn.Module):
        Hari Gupta *
160     def __init__(self):
161         super().__init__()  # instantiate our nn.Module
162         self.model = nn.Sequential(
163             nn.Conv2d( in_channels: 3,  out_channels: 32,  kernel_size: (32, 32),  stride: 1),  # input layer
164             nn.ReLU(),
165             nn.Conv2d( in_channels: 32,  out_channels: 64,  kernel_size: (32, 32),  stride: 1),  # first hidden layer
166             nn.ReLU(),
167             nn.Conv2d( in_channels: 64,  out_channels: 64,  kernel_size: (32, 32),  stride: 1),  # second hidden layer
168             nn.ReLU(),
169             nn.Flatten(),  # Shape coming out from flatten, aligns with shape expected in Linear Layer
170             # parameters of linear layer is (in_neurons * (image height or width * 2), out_neurons)
171             nn.Linear(64 * (419**2),  out_features: 1),  # output layer "morph" or "not morph"
172             nn.Sigmoid()  # Sigmoid activation function to get an output between 0 and 1
173         )
174
175     # function to move data between layers
        Hari Gupta
176     def forward(self, x):
177         return self.model(x)
178
179
180 # instantiate our neural network, optimizer, and loss function
181 md = MorphDetector().to(device)
182 opt = optim.SGD(md.parameters(), lr=0.0001)  # adam gets trained very quickly, so we can use fewer epochs
183 loss_fn = nn.BCELoss()
```

```
--------------------------------------------------------------------------------

The model has finished training!

The model state was saved at the lowest loss rate: 0.69266754

--------------------------------------------------------------------------------

100%|██████████| 50/50 [09:17<00:00, 11.15s/it]
100%|██████████| 50/50 [09:19<00:00, 11.18s/it]
100%|██████████| 50/50 [09:20<00:00, 11.22s/it]
100%|██████████| 50/50 [09:20<00:00, 11.21s/it]
100%|██████████| 50/50 [09:08<00:00, 10.98s/it]
```

```
Here are the conf matrix values:
 [50, 25, 43, 23, 19] [0, 25, 7, 27, 31] [50, 24, 36, 22, 17] [0, 26, 14, 28, 33]

--------------------------------------------------------------------------------
```

Appendix 2 - Project Initiation Document (PID)

SECTION 1 – General Information

Project Identification

1.1	**Project Title**
	Vision Based Presentation Attack Detection
1.2	**Please describe the project with key-phrases (max 5)**
	Machine Learning, Neural Network, Computer Vision, Biometric Facial Recognition, Presentation Attack Detection
1.3	**E-logbook maintenance agreed with supervisor.** *Use Google doc, OneDrive, or any mobile App whereby you will be able to generate a PDF copy*
	The agreed upon E-logbook will be in Google Docs
1.4	**GitLab link for maintain source code and research data.** *Any change in GitLab link and Source code repository MUST be explicitly mentioned in final report*
	https://csgitlab.reading.ac.uk/ur012338/final-year-project

SECTION 2 – Project Description

2.1	**Summarise the project's background in terms of research field /application domain (max 100 words).**
	The primary focus of research in this project is on Computer Vision, Biometrics and Machine Learning. The application domain of this project is for identity verification in sectors where security is paramount, such as the home office and their online passport renewal service.
2.2	**Summarise the project aims, objectives and outputs (max 250 words).** These aims, objectives, and outputs should appear as the tasks, milestones and deliverables in your project plan (fill out Section 3).
	The aim of the project is to create a Biometric Evaluation service, this will be accomplished by developing and training a Machine Learning algorithm in Python. It will put an image through various checks; firstly, image quality checks to ensure the image is of a high enough quality to be evaluated; secondly, a presentation attack detection layer to ensure no malicious attackers can gain unauthorised access; finally, an identity verification layer to validate the user's identity. Objectives: - Conduct research into key areas (e.g., Face-morph detection, ICAO spec, RESNET 50) - Learn how to create a classical machine learning classifier in python - Create a facial recognition prototype, using python and pytorch - Train and test the prototype on specific training and testing datasets - Tweak and improve the algorithm, to achieve better accuracy / results - Quantify and evaluate the results of the algorithm and the overall project - Complete written documentation and hand in completed Final Year Project Milestones: - Initial research into project's background - First Prototype functional - Training and Testing in progress - Programme ready for deployment - Project Documentation written up - Work handed in and FYP completed

2.3	**Initial project specification – roughly indicate key features and functions of your finished program/application. Indicate possible method, data source, technology etc. (max 400 words)** (Sensible and relevant Charts, Table, and Figures can be used)

	Key features / functions of this program include:
	
	- Identity Verification:
	  - Take a live image from a user
	  - verify if the user is who they claim to be
	  - by checking against database using the ML algorithm.
	- Presentation Attack Detection:
	  - Detect Face morphing attacks
	  - Recognise Print attacks
	  - etc.
	- Image Quality Checks
	  - Face Image quality assessment
	
	Data Sources and technology:
	
	- Python and Pytorch - to create a machine learning classifier.
	- Webcam for testing and developing and testing the facial recognition programme.
	- A large dataset of faces, provided by the department of computer science. for training and testing the machine learning algorithm.

	Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval? (If your project requires a questionnaire/interview for conducting research and/or collecting data, you will need to apply for an ethical approval)
	This project will require ethical approval for collecting faces and facial recognition data
2.5	**Identify the items you may need to purchase for your project. A cost up to £200 can be applied (include VAT and shipping if known). You need to have consent of your supervisor. Your request will be assessed by the department.**
	Logitech Webcam – 1080p 60fps: This is needed for testing functionality and debugging the facial recognition algorithm in real time, during the development process, and final deployment.
2.6	**State whether you need access to specific resources within the department or the University e.g. special devices and workshop**
	N/A

SECTION 3 - Project Plan

Please provide your project plan.

Below is an example project plan, you can use any tool or software to generate yours.

START DATE: 29/09/2023

Project Weeks — **Ideal Timeframe: Blue** — **Backup Timeframe: Red/Orange**

Project stage [this is only indicative – write your own stages]	0-3	3-6	6-9	9-12	12-15	15-18	18-21	21-24	24-27	27-30	30-33	33-36	36-39
1 Background Research													
1.1. Create Project Initiation Document													
1.2. Research topics discussed in initial meeting													
2 Analysis/Design													
2.1. Plan out the workflow of the program													
2.2. Design the interface to be used													
3 Develop prototype													
3.1. Create First working Prototype													
3.2. Train Prototype on Training Data													
4 Testing/evaluation/validation													
4.1. Conduct testing and evaluate performance													
4.2. Tweak algorithm to optimise its accuracy													
5 Assessments													
5.1. Visualise Performance Data / Metrics													
5.2. Evaluate the overall success of the project													
5.3. Complete write all up documentation													

Appendix 3 - Code Repositories / Scripts

Morph Attack Detection Algorithm

This is the actual script that performs the morph attack detection on our dataset
https://csgitlab.reading.ac.uk/ur012338/final-year-project

Datasets used for training and validation

This includes every dataset collected, generated, etc throughout our entire research project, some of which are directly called into the above script to train our machine learning classifier
https://livereadingac-my.sharepoint.com/personal/ur012338_student_reading_ac_uk/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Fur012338%5Fstudent%5Freading%5Fac%5Fuk%2FDocuments%2FFYP%2Ddatasets&ga=1

Dataset Creation algorithm

This is the script used to manage and organise the dataset creation
https://csgitlab.reading.ac.uk/ur012338/facemorph_dataset_creator

Face-morphing solution used to perform the morphing

This is used in a certain stage of the above data-set creation algorithm to create face morphs
https://github.com/alyssaq/face_morpher

Custom Scripts written to automate process and perform data handling

This is the custom script that conduct the morphing by calling other scripts in the above repository

```python
from pathlib import Path
import averager
import aligner

# allows us to declare the source and dest directories
src_dir = Path("C:/Users/harip/Desktop/facetest/dataset/FERET-dataset-aligned/")
dst_dir_1 = Path("C:/Users/harip/Desktop/facetest/dataset/FERET-dataset-morphed/")
dst_dir_2 = Path("C:/Users/harip/Desktop/facetest/dataset/FERET-dataset-unmorphed/")

# Get all image paths (* means "any combination")
image_path_list = list(src_dir.glob("*.png"))
print(len(image_path_list))

x = 0
# need to improve the way images are chosen / program looped
# would be ideal to choose to random images from the stack
try:
    for x in range((len(image_path_list)-15)):
        # load in two aligned images - source
        src_img_path_1 = '{}'.format(image_path_list[x])
        print(src_img_path_1) # print out the image being sent to the unmorphed
        y = x + 15 # rudementary fix for having similar images near one another in name order
        src_img_path_2 = '{}'.format(image_path_list[y])
        print (x, y)
        print('Aligning together: ' + src_img_path_1 + ' & ' + src_img_path_2)

        # create the path of the morphed image - destination
        dst_img_path_1 = str(dst_dir_1) + '/morphed_face_{}'.format(x) + '+{}'.format(y)
        # create the path of the unmorphed image - destination
        dst_img_path_2 = str(dst_dir_2) + '/unmorphed_face_{}'.format(x) + '.png'
        # print out path of the two destinations
        print (dst_img_path_1, ' | ', dst_img_path_2)

        # generate morph using the background of the first image
        averager.averager([src_img_path_1, src_img_path_2], width=512, height=512, background=0, out_filename=(dst_img_path_1 + "_a" + '.png'))

        # generate morph using the background of the second image
        averager.averager([src_img_path_1, src_img_path_2], width=512, height=512, background=1, out_filename=(dst_img_path_1 + "_b" + '.png'))

        # generate morph of two faces without the background
        # averager.averager([src_img_path_1, src_img_path_2], width=512, height=512, out_filename=dst_img_path")

        # generate a "morph" formatted image but with no morphing | basically just cropped and aligned
        averager.averager([src_img_path_1], width=512, height=512, background=0, out_filename=(dst_img_path_2))
except Exception as error:
    print ("The morphing has stopped early")

print('{} unmorphs have been created.'.format(x))
print('{} morphs have been created.'.format(x*2))
```