Facial Liveness Testing: an approach for the web

Student Name: Ryan Collins

Supervisor Name: Prof A. Krokhin

Submitted as part of the degree of MEng Computer Science to the

Board of Examiners in the Department of Computer Sciences, Durham University

May 1, 2019

Abstract —

Context With password based authentication methods being subject to many attacks, facial recognition is an alternative, relying on biometrics rather than remembering an easily stolen string of characters. While secure facial recognition methods have been developed, spoofing is still a concern. Spoofing is where someone tries to pretend they are someone else to gain access. In order for facial recognition to become more prevalent on the web, facial liveness is needed. Facial liveness methods don't check identity, but aim to detect these impersonation methods. The next step towards secure facial recognition systems are facial liveness services, which provide an indication of liveness and can be integrated into applications, through APIs on the internet. This paper aims to understand which liveness tests are suitable for deploying in such a service, and explains next steps and architectures to the development of a facial liveness service for the modern web.

Aims

- To design, implement, train, and test a new quality based facial liveness test for classifying a whole image.
- To design, implement, train, and test a new facial feature based liveness test, using Residual Networks.
- To design, implement, train and test a new 3D based mask attack detection liveness test using 2D to 3D reconstruction methods.
- To design and test a method for combining multiple liveness tests into one single metric to measure liveness, that can be used to include further liveness tests.
- To only consider liveness tests that can be used on a mobile device, containing a camera of varying quality, and other IoT devices. No special hardware must be necessary to collect the required data.
- To develop liveness tests that can classify liveness for a single image in a maximum of 2 seconds.

Method Design, train and test three different innovative liveness tests. Train using the NUAA and Replay-Attack devel datasets.

The first is the W-IQA liveness test, which analyses image quality across the entire image. This is achieved by calculating 24 different quality metrics for each image, and using output obtained using a Linear Discriminant Analysis classifier (LDA).

The second is a CNN based 2D liveness test. This model used a CNN-based face extraction process to obtain the input face, before feeding this through a ResNet50 based model to classify whether a specific face is real or spoofed.

The third is a 3D based liveness test to detect mask attacks. This process combined existing techniques for reconstructing a face in 3D from a 2D image, and fed this reconstruction through a VoxNet classifier to detect whether the 3D data contains spoofed input or real input.

Using the classification probabilities from liveness tests, an LDA based classifier was then trained to fuse the results of the above metrics together, to improve accuracy and allow for future expansion with further liveness tests.

Results All tests for 2D attacks were carried out over the Replay-Attack test dataset. Top-1 accuracy was measured to determine how accurate the predictions were, and a confusion matrix was produced to analyse what errors the models give.

The W-IQA test performed well, yielding 87% top-1 accuracy. The levels of true positives and true negatives were very high. However, when this test made incorrect predictions, it would often yield false positives which in a real world liveness system would lead to spoofed images yielding a rating of 'real', which isn't ideal.

The CNN based 2D liveness test performed adequately, yielding a 71% accuracy. The model yielded several false negatives, but very few false positives, which is more desirable since false positives lead to a security concern, while false negatives are annoying for the end user, could be potentially filtered out by using a sample of images, and don't yield any security concern.

The 3D based liveness test for detecting mask attacks performed poorly and had various performance issues due to the amount of resources needed to train, and the time taken for a prediction to be made. As such, the accuracy of this model cannot be obtained, but the performance in production can be explained.

Combining the results from the W-IQA and CNN liveness tests yielded a partially linearly separable graph. The LDA used for combining the two metrics worked reasonably well, but more liveness test data is required to assertain the accuracy in production.

Conclusions Overall, both the Whole Image Quality assessment and CNN based 2D test are metrics to include in a liveness-as-a-service system. The W-IQA test individually yields impressive results, but the CNN based metric would perform well when working together with other metrics. The CNN based metric could be further improved by adjusting preprocessing, and using a deeper ResNet pretrained model. The VoxNet model has also shown that this method is a poor liveness test for the existing use case, due to high computational requirements. In addition, the speed at which queries can be answered shows that these can reasonably be used in a web system without extensive delays in processing, or without requiring any additional hardware (aside from a camera).

Keywords — Facial liveness, convolutional neural networks, image quality metrics, residual networks, anti-spoofing

I INTRODUCTION

Currently, username and password authentication is commonplace throughout the web. However, this method has serious security concerns, being easily broken through dictionary attacks, or shoulder surfing. Facial recognition authentication doesn't have either of these problems, but introduces the risk of stolen identity through face spoofing, more specifically expressed in a quotation from Adam Schwartz, a lawyer with the Electronic Frontier Foundation speaking to NPR: "We can change our bank account numbers, we can even change our names, but we cannot change our faces. And once the information is out there, it could be misused". [13] In order to reduce the risk of stolen identities, a system to detect potential spoofing attempts is needed. Providing facial liveness systems work correctly, the ability for criminals to steal people's faces would be reduced, making facial recognition more ideal for authentication on the web. The next step to making facial recognition more secure and mainstream, the implementation of a facial liveness service is necessary: that way developers won't need to worry about implementing their own liveness tests and can instead focus on the process of building their software.

While different methods of detecting liveness are available, these are specialized towards defending against a different type of attack. Some existing liveness tests require use of special hardware or sensor configurations that are not available in most smartphones/computers. The aim of this work is to design and highlight liveness tests that only require a camera, can yield an output in near-real time, and give adequate accuracy. Once these methods are developed, future works can further develop these methods, or build a facial liveness system designed for the web.

In this context, we propose a novel new 3D-based liveness test, based on a two part approach: (i) VRN based 3D reconstruction (ii) VoxNet based 3D classification. We provide a promising new liveness test using ResNet based networks, used to analyse facial structure. We also adapt an existing Image Quality Assessment test for use with whole image classification, rather than a face based method, to classify an image regardless of whether it contains facial information.

II RELATED WORK

As defined in [15], the types of face spoofing attacks can be described under three sections: Photo Attack, Video Attack and Mask Attack. Photo and Video attacks are both classed as 2D spoofing attacks, while mask attacks are 3D attacks.

A 2D Spoofing Attacks

Photo and Video Attacks are both 2D spoofing attacks, which involve using a previously retrieved photo/video, and holding it in front of a camera. In the case of photo attacks, a single photo is used, where in video, some video would be played back on a screen. [15].

With video-based facial recognition systems, user motion is used to determine whether the person is real or spoofed, such as blinking, head movement and others.

In the method defined in [3], structure from motion (SFM) was used on a video to produce a 3D model of a user, with the depth channel being used to determine whether a person is real, or whether it's simply an image. This was extended further by fusing the SFM based method with audio verification. The fusion of multiple methods provides greater reliability. However, while SFM works with video, it doesn't work with a single image, and it also doesn't work if a video with little motion is provided. This fusion was completed using a Bayesian Network.

While motion based methods require a video input, quality based methods can be used for both video and image input (either by extracting key video frames or using all video frames and combining the results).

Since image quality is a major factor in detecting 2D replay attacks, understanding the image quality with respect to different image quality metrics can yield a rough liveness estimate. An extension of this basic method was discussed in [7] by combining 25 different image quality metrics into a vector, and using this as input to a classifier (an LDA). As shown from the work, the accuracy of such a metric is high, and the metric calculations aren't very computationally intensive. [7].

This is an example of combining many items to yield better results. While each metric on its own isn't that great, using them all together yields better results.

Recently, deep learning based approaches have been applied to the task of facial liveness classification.

A.1 Neural Networks, and their structure

Deep learning approaches are focused around the concept of neural networks. Neural networks consist of several layers, each of which are linked by weights. Traditional neural networks have very few layers (shallow), but recently the trend of using a large number of layers (deep) has become popular, yielding better performing classifiers as a result.

There are many types of layers, but the two most commonly used are outlined below:

Fully connected layers Fully connected (or dense) layers consist of neurones. Given two layers l_1 and l_2, each neuron n_i in l_1 is linked to each neuron n_j in l_2 with a weight $w_{i,j}$. For each neuron in the previous layer, the output of that neuron is multiplied by a weight, the total is summed, and the sum is put through a function $t(x)$ which is the activation function. Examples of activation functions include *reLU*, *sigmoid*, and *hyperbolic tangent*.

$$n_j = t(\sum_{i \in PrevLayer} (n_i * w_{i,j}))$$

Convolutional layers Designed primarily for images, these convolve over an image with a kernel. For each location over an image, the weights are multiplied by the image input at that particular location. These are then fed through an activation function. The overall output of a convolutional layer would be a 2D image (in the case of images). While 2D images will be primarily used for image input, 3D convolutional layers follow this same process for 3D representations. Convolutional layers are better used for images compared to fully connected layers, since fewer weights (also known as parameters) are required, thus reducing the required computation and storage.

A note about performance metrics Throughout this paper, the number of parameters acts as a metric of performance. The more parameters (weights) that a model contains, the more computation that needs to be completed, thus requiring more time/better performing hardware. While operations per second is possibly the best metric, number of parameters gives a good indication without the requirement to count the number of operations that occur. Furthermore, counting the number of operations per second also depends on the hardware being used, and as such it's more difficult to compare unless benchmarking is conducted on a single type of machine.

I believe Convolutional Neural Networks are a key approach to facial liveness, since they can be used to learn a variety of features including face structure, texture, and other potential suspicious visual glitches which would be obtained with face spoofing. Convolutional Neural Networks The method proposed in [20] uses Caffe-Net, inputting both the full image along with the isolated face. The output yielded general texture differences, as well as specific facial texture differences. Another interesting idea proposed in this paper is the fusion of two algorithms together to produce an outcome, therefore reducing the false reject rate.

A.2 General 2D image classification models

One major problem with training CNNs in the past has been the risk of over-fitting due to lack of data when training from scratch, but utilizing pretrained models would help reduce this risk. Facial liveness is an image classification problem, and therefore adapting an existing image classification model to the job of facial liveness might yield good results.

One major research area has been the classification of objects based on a 2D image, which uses the well-known ImageNet dataset. The models produced for this purpose have improved in accuracy, while also being reasonably efficient in terms of computation time to be deployable on the web. Since these models have already been deployed in the real world, and the goal of image classification is common with the goal of liveness detection, these models could be a good fit for adapting to liveness applications. Since facial liveness testing is an image classification problem, these models are of importance.

AlexNet One of the initial models was AlexNet, which has 5 convolutional layers (with some max pooling layers), and two globally connected layers. This model was used to classify 1.3 million high resolution images into 1000 classes. [14] Overall deployability with Alex net is fairly good due to a fairly low number of compute operations (meaning faster compute time). [1] However, the accuracy of AlexNet isn't as good as newer methods (such as the ones shown below), which all perform better in terms of accuracy.

VGG16 Network The VGG16 model improved AlexNet by replacing larger filters by more smaller filters one after another. However, VGG requires a high amount of computational power, something that's not easily deployable due to the large number of parameters (128 million), which requires a high amount of memory and compute power compared to other models. [1] In our application, high accuracy is important, but the high memory and computational requirements mean this model isn't the most suitable.

GoogLeNet Inception GoogLeNet is an improved module that approximates a space Convolutional Network with a normal dense construction. One of the major features of GoogLeNet is the Inception module. The naive approach to this is to take the input from the previous layer, calculate a 1x1, 3x3 and 5x5 convolution (all at the same time), along with a 3x3 max pooling before feeding this into an output, the output being the filter concatenation step. However, to reduce the dimensionallity, and therefore improve performance, a 1x1 convolution is applied before each 3x3 and 5x5 convolution, and after the max pooling output. These inception modules can be used and stacked to improve performance without a huge increase in computation. [22]

Therefore, this model has improved computational performance over VGG, making it a more suitable model for the web compared to VGG. [1]

Residual Networks Another key problem with deep convolutional networks on when used for image classification is the vanishing gradient problem. This occurs when early layers have very small gradients during the training process and making the model very difficult to train. Residual Networks avoid this problem by allowing a direct path in links between the input and output of a building block. The overall outcome is far better accuracy than VGG and GoogLeNet while being more efficient than VGG in terms of computational power needed. [9] While these aren't directly associated with facial liveness, the nature of image classification is fairly similar to facial liveness (since the image is simply a classifier with two outputs instead of 1000). This model is very ideal for our application, as it acts as an ideal compromise between accuracy and computational requirements, yielding fairly good accuracy for our security focused perspective, while also ensuring that the operational cost (in terms of time, and cost for hardware) isn't too high, compared with other models such as VGG.

Architecturally, Residual Networks consist of residual blocks, which can be seen in Figure 1. This solves the vanishing gradient problem by utilizing shortcut connections, which skip one or more layers. The outputs of these shortcut connections are added to the outputs of the stacked layers.

One important metric to analyze the performance of a model is the Top-1 accuracy figure. This metric represents the class with the highest probability in the model must be exactly the expected answer. The Top-1 accuracy scores for the models outlined above are shown in Figure 2a. These results show that a newer version of the Inception model perform the best, with ResNet based classifiers performing fairly well too. AlexNet performance on the other hand was fairly poor. While accuracy was important, the number of operations is also important, as this will affect how easily the model can be used in a real-world system. Based on the results shown in Figure 2b, AlexNet and GoogLeNet are shown to require very few operations (< 10 G-Ops), with a similar result being yielded

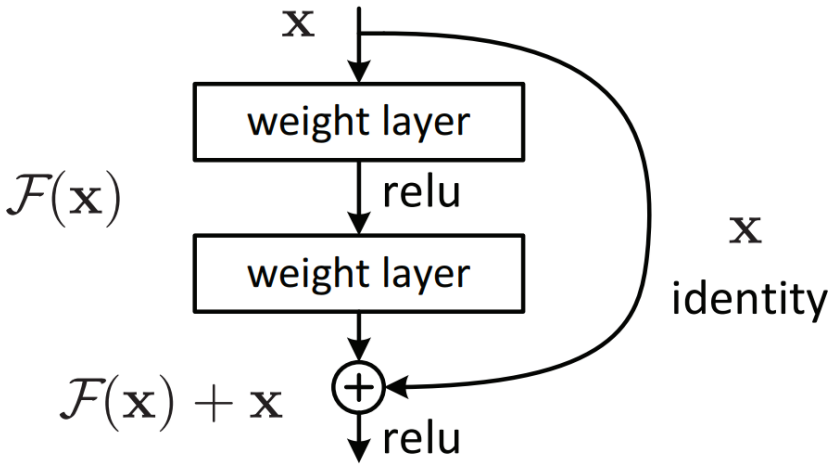
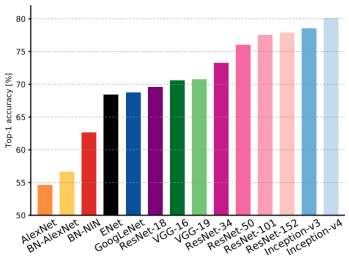
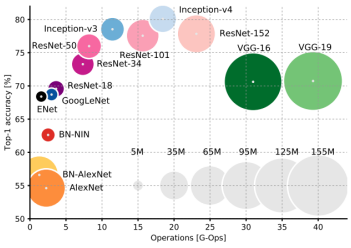


Figure 1: *A Residual Block: this is the building block of the ResNet architecture. Source: [9]*



(a) *Top 1 accuracy vs network. Chart and results from [1]*



(b) *Top 1 accuracy vs operations, where network size is proportional to number of parameters. Operations figures are for a single pass (e.g. predicting given a specific input). Chart and results from [1]*

Figure 2: *These two charts, from [1], show the accuracy and computational requirements of each type of classifier.*

for ResNet 50 and ResNet 34. On the other hand, VGG models require a large number of operations for a similar accuracy, requiring >30 G-Ops. This fact is also true for Inception models.

A.3 Datasets

One of the most common datasets, and one of the earliest created, is the NUAA dataset. The dataset contains photos of 15 subjects, with faked photos (both flat and warped) being placed in front of the camera. Each photo is stored in JPEG format, with a hierarchical structure to show whether the image was real or fake, and what type of attack was used (in the case of faked images). One drawback with the NUAA dataset is with regards to licensing: the dataset is only available for non-commercial use (i.e. research purposes). This implies that this dataset, and models trained with this dataset, would not themselves be able to be deployed [23] However, as this project is only researching into the feasibility of a web based liveness testing system, this project is for research purposes and hence meeting the licensing objectives.

In 2012, the Replay-Attack dataset was first released, which consists of 1,300 video clips of both photo and video attacks. Each set of videos/images are taken under different illumination conditions, and various different attack methods were collected: printed photo, low resolution and high resolution screens with both photos and videos being displayed. The entire dataset consists of several 'sub-datasets': the 'devel' dataset is designed as a validation set/training set, while the 'test' dataset is designed as a test dataset (and therefore must not be used in the training/validation process). [2]

These two datasets are ideal for testing 2D based liveness systems, since they contain enough samples to reasonable train/test our models. They shall be used throughout this project for both training and testing the 2D based liveness models.

A.4 Temporal-based Liveness Tests

These liveness tests require a video input, rather than an image. Rather than looking directly at an image, they mostly look at the differences between the images in a video.

Blink Detection The blink detection liveness test analyses eye blink behavior and determines whether the person is real/fake. This liveness test meets our criteria for a web-based liveness service, as the test doesn't require any extra hardware. Furthermore, a large amount of conscious activity from the user isn't necessary, as the video analyzed is natural behavior. One drawback of this method is testing: this model requires a video input, which isn't available with the NUAA dataset, and therefore testing this method might be more difficult. [19] Furthermore, this method would be susceptible to a print-based attack where eye holes were cut, since natural eye movement is analyzed.

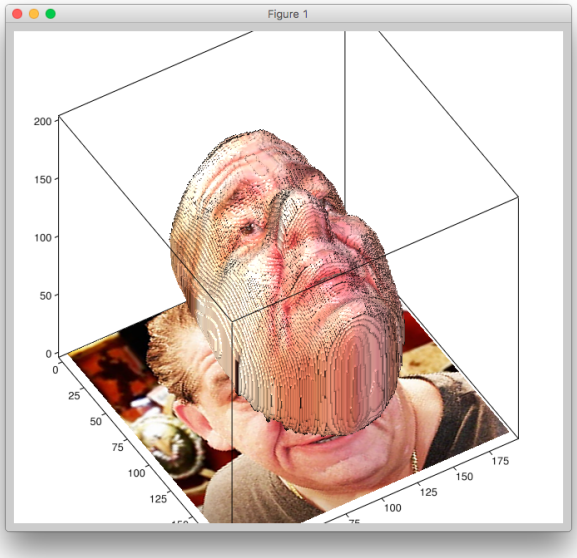


Figure 3: *A screenshot from the VRN Torch to Keras version (a conversion of the VRN model to Keras). This screenshot is from [17]*

Face Flashing One method, known as the "Face Flashing" liveness detection method, uses the light diffusing off a screen to determine that the input is in fact from the user, rather than from a spoofed recorded/simulated input. The method also considers how the light is diffused, as light would diffuse differently based on a face compared to a piece of paper/a screen. [24] While this metrics seems very promising at avoiding replay attacks, testing it to a larger degree is more difficult due to the lack of data available for testing, and therefore not implemented for this project. One potential drawback though would be with devices that don't have screens but do have cameras, such as with IoT devices. Since a screen isn't present on these, the liveness test wouldn't work. However, for traditional mobile phones/computers this would work better.

Overall, these metrics would be fairly useful on video input, since they take into account the movement/differences between frames, but for implementation purposes they are difficult to test due to lack of datasets available for them (since they require special information that isn't available from existing sets).

In a web-based approach, the face flashing approach would be particularly useful at determining whether a source-quality image was being fed into the algorithm, since the random face flashing colors would be able to be detected.

B 3D Spoofing Attacks

Mask Attacks are a 3D spoofing attack, which involve creating a 3D mask of someone and wearing it. [15] These are much less prevalent, but with 3D printing becoming more mainstream, this could potentially get more prevalent in the future.

In 2013, the 3D Mask Attack Dataset (MAD dataset) was released. This dataset contains 76500 frames of 17 people, recorded using a Kinect camera. Each frame contains a depth image, an 2D RGB image with 8 bit color and a size of 640x480 pixels. Each frame also contains eye positions. Real Access samples are contained within the first and second session folders, while the third session contains the frames of mask attacks. The subjects contained within each frame are denoted within the filename.

[6]

B.1 Obtaining 3D data from 2D images/video

One method of obtaining 3D data from a video is to use Structure From Motion (SFM). This method was previously utilized in [3] to consider the depth data as a 2D liveness test, but this method could potentially work as a solution for obtaining 3D data. However, SFM performs poorly when given a video with a small amount of motion. Furthermore, the 3D mask attack datasets available for this project do not contain video but single images, which means such a method would be more difficult to test.

However, with improved machine learning models, there exist models that can reconstruct facial structure based on a single 2D image. One such model is the VRN model, proposed in [11]. The paper, and the associated source code, contains a model that takes a 2D image as input (of size 192x192), and produces a voxel representation of the 3D face structure. This model is more suitable for our application, since the use of a 2D image rather than a video works well with the datasets available. Furthermore, use of a single 2D image would require less user input as a SFM based method, since the user isn't required to move in order to produce an adequate 3D reconstruction. An example of the 3D reconstruction obtained from this model can be seen in Figure 3. The screenshot is obtained from the GitHub repository of a Keras-based implementation of this method (since the initial implementation of the paper used the Torch library).

B.2 3D Classifiers

In a similar approach to the 2D methods researched above, there are a variety of existing 3D classifiers that could be re-purposed for the liveness classification approach. While these methods don't yield the same accuracy figures as with ImageNet, further improvements

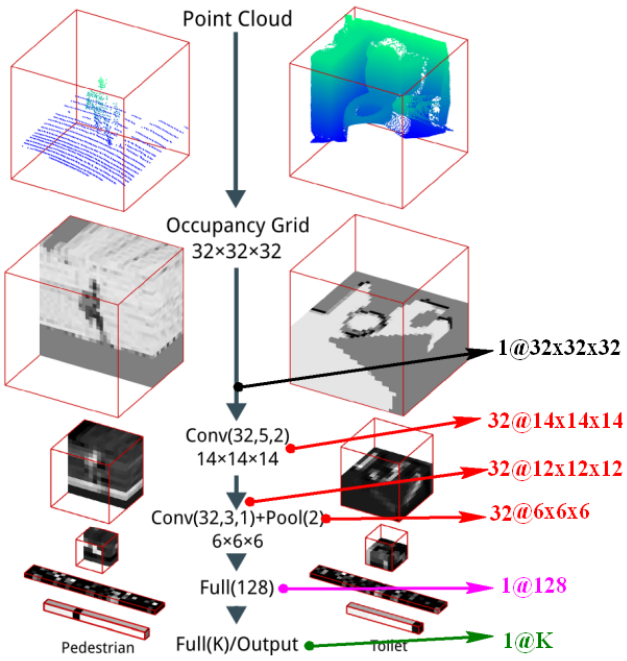


Figure 4: *The original VoxNet architecture. While point cloud input is available, voxel based representation can also be used. Diagram obtained from [18]*

in the field of 3D classification could lead to better liveness methods in the future.

PointNet PointNet is a model for classifying point clouds, represented as an unordered set of points. PointNet in total has 3.5 million parameters, with 440 million FLOPs per sample, which is a reasonable number considering other classifiers, but is on the upper end of what would be expected for a web based system. The classifier used within PointNet is a simple feed forward network. [21] While this model has performance benefits over CNN based models (since 3D convolution is computationally expensive), the input is a point cloud structure which might not be the most ideal solution (since sparse data might not be captured, dependent on which reconstruction model is used).

VoxNet Unlike PointNet, this model uses a voxel representation. This is ideal due to the nature of existing 3D reconstruction models from a 2D image. The accuracy of this classifier isn't the most ideal, but improvements could be made such as implementing a Residual network approach (something that worked well for 2D classifiers). As a proof of concept, this model would be ideal to test whether this model would work in the real world. This model was developed to identify objects based on their 3D representation.[18] This is very similar to the process of identifying liveness based on the 3D face structure input. The overall architecture of the classifier can be seen in Figure 4. Convolutional layers are used to learn the 3D feature space, while a dense layer is used to produce the output.

III SOLUTION

A Specification, and design requirements

1. Any liveness test produced must be able to predict a single image in under 2 seconds on a standard i7-7700k powered machine
2. Any liveness test must only require the use of a 2D image input from a standard built in camera. No other external hardware is allowed.
3. The Top-1 accuracy figure of each liveness test should be greater than 70%.
4. The liveness test consolidation layer must allow for future inclusion of further liveness tests,.
5. A liveness test should be developed to classify the entire image, to detect spoofing given a specific input.
6. A liveness test should be developed to detect liveness based on the 2D input of a facial area from a camera.
7. A liveness test should be developed to specifically detect 3D based attacks (mask attacks).

1 is important to ensure that the liveness tests that are built don't take too long to execute and yield a result. While in practice an Intel i7-7700k machine might have a slightly higher clock speed than a standard cloud-based processor, the results would still be reasonable in practice.

Some of the liveness tests proposed in previous works require the use of specialised hardware which won't be available on many devices. The most common device capable of taking in facial input is a webcam, and since these are found in many laptops, IoT devices, and smartphones, this is an ideal limiting factor. While screens could also be considered useful, certain IoT devices might not necessarily have them either. This is what 2 aims to address.

The results of the liveness tests produced are also important, since a random prediction of liveness yields a 50% accuracy, so therefore a 70% accuracy yields a reasonable performance over this, and might require extension to improve it further, while a higher accuracy indicates that the liveness test performs better and might have been refined more. This is what 4 addresses.

5, 6 and 7 propose the method which each liveness test uses and how they work functionally. Whole image detection is necessary due to the potentially poor performance of a model to extract a face, since it might be ideal to still be able to obtain a liveness value based on the estimate of the image quality itself. Furthermore, while quality is one factor, analysing the facial structure and texture to determine whether spoofing has occured is also an important factor. Finally, mask attacks are still not that common, but with the rise in 3D printing technology we could see an increase in them in the future. As such, a method of preventing 3D based mask attacks is necessary.

With these specifications in mind, the design focused on three different liveness tests: an Image Quality assessment based liveness test, a ResNet 50 based classifier (based on a pretrained ImageNet model), and a novel 3D based classifier using 3D facial reconstruction models interlinked with the VoxNet 3D classification model. Each of these liveness tests have some common services that are needed; these include reading large datasets without causing resource availability problems.

B Shared Services

B.1 Dataset Managers

In order to assess our liveness tests and train them, dataset managers are needed.

A generic implementation of a dataset was created as an Abstract Class, which was then extended by the NUAA, Replay-Attack and Mask-Attack dataset managers. This generic implementation allows for future datasets to be easily added, and also provides the class definition of what's needed, to help improve the software engineering process.

The role of a dataset manager is to load in a dataset from a folder structure (which varies between dataset), conduct any basic preprocessing to convert the files into OpenCV images, and produce two H5Py Datasets, one for real images, and one for fake images. The dataset manager also allows further customization, to load specific sub-datasets (such as 'devel' or 'test' in the case of the Replay-Attack dataset), or load data regarding specific subjects (in the case of the Mask Attack Dataset). The role of the H5Py dataset is to normalize the datasets into a specific format, to allow for easier dataset processing.

In addition to data normalization, it also provided a method of reducing RAM usage, therefore allowing larger liveness test models to be created. This is because data is only fetched when needed from the hard drive, rather than loading the entire dataset into RAM at once.

B.2 Neural Network Infrastructure

Neural Network Framework For the 2D and 3D based classifiers, neural networks were to be used. It was decided to use Keras, with the Tensorflow backend, as Keras provides a high level interface to Tensorflow and also allows for other backends to be used in the future (which could potentially perform better in different scenarios). The Tensorflow backend was used because of the easy configuration with both GPUs and without (simply install tensorflow-gpu for GPU processing, and tensorflow for CPU only processing), depending on the machine being used. A high level interface was necessary to avoid boilerplate too, since models would be regularly changed to find the best outcome for each classifier.

Hardware for training Training the neural network required more processing power than was directly available. Google Cloud Compute Engine was used to provide this processing power, since it's easily accessible and has existing deep learning environments with Intel-accelerated mathematics libraries. The Google Cloud instance also provided easy extensibility, since some of the liveness tests required additional tools that could not be easily installed on NCC/Hamilton clusters at Durham University. Training was conducted on a VM with 64GB RAM, 8 virtual CPUs with an NVIDIA Tesla P100 GPU being used to accelerate the training process (through parallelism).

Hardware for testing GPUs are expensive, and therefore if such a system were deployed the cost of GPUs would provide expensive to run. Therefore, a CPU only implementation for the testing was followed, using an Intel i7-7700K (at 4.2 GHz), with 16GB RAM. Performance metrics were yielded using this hardware to emulate the expected performance and understand which metrics performed best and whether they performed adequately.

B.3 Image Processing and Computation Management

OpenCV was used to manage the processing of images, including the image loading components. This library was chosen for the wide support available, and for the large feature set that it provides (including Gaussian Blurring, some image metrics, and some fourier analysis). For the rare occasions where OpenCV didn't have the required functionality, scikit-image provided some operations that were necessary.

Numpy was instrumental in most other computation operations (including image preprocessing, mathematical calculations), due to the large speed improvements provided by the C implementation (compared to Python's slower math libraries). Since OpenCV interfaces well with Numpy, no conversion was needed which improved the ease of implementation.

C Image Quality Assessment Liveness Test

C.1 Overview

One common way of detecting liveness is to consider the image quality of the camera input. When a printout/screen is held up to the camera, the facial image will have some noticeable differences, specifically in the high frequencies. There might also be some image compression visible in fake images, compared to real ones. This is the basis for this method.

More specifically, the implementation was based on the work contained in [7]. 24 different image quality metrics were implemented, with metric values being used as an input to a classifier. From previous work, it has been shown that this metric is accurate (therefore detecting spoofing well), while also being fairly fast to compute in terms of time, meeting 3 and 1.

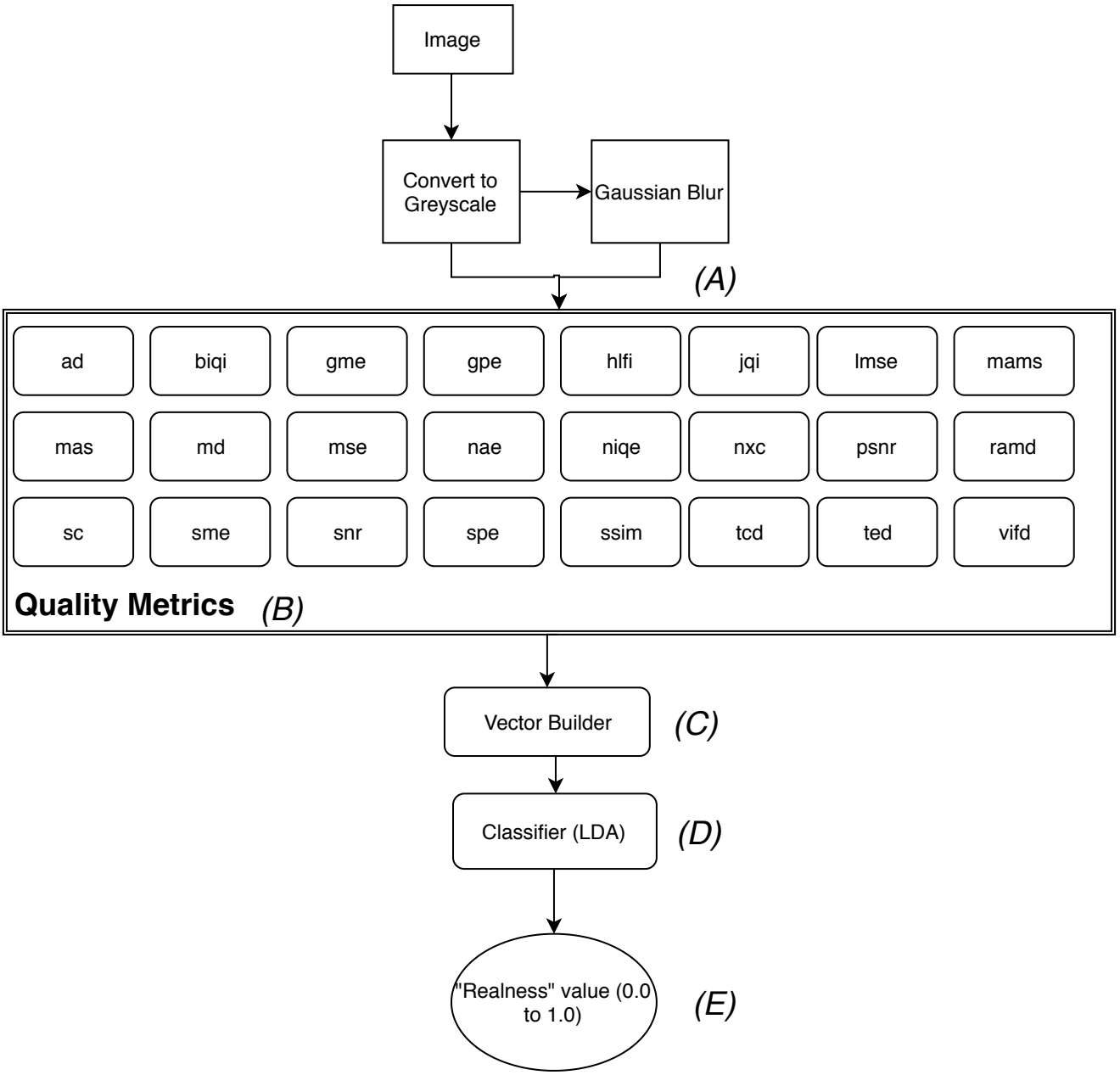


Figure 5: *The architecture of the image quality liveness test. (A) The greyscale copy of the image, and a blurred copy of the image are input into each of the metrics. (B) The metrics are individually calculated, and a single value output from them. (C) These values build a 1D vector. (D) They are classified using an LDA classifier. (E) The realness value is 1.0 for real, and 0.0 for fake, or in between.*

The classifier being used for our implementation is Linear Discriminant Analysis (LDA).
A visual explanation of the method can be seen in Figure 5.

C.2 The Metrics

The original paper proposed that 25 metrics were used. However, our implementation used only 24, due to some implementation problems. Most of the metrics were implemented manually as defined either in [7], or utilizing further sources to understand how they worked. In addition, the Matlab source code that was released with [7] was referred to, but written manually in Python using Numpy/OpenCV or in some cases other libraries.

Each of these metrics requires an input image I. Non-Reference metrics only require I, while Full-Reference metrics require an additional image I'. $I' = Gaussian(I)$ in our implementation.

PyVideoQuality Library As part of the implementation process, the NIQE metric required the use of a pretrained classifier which didn't exist for Python. As such, it was necessary to modify existing code. The only Python source code available was [10], which was written for Python 2 and consisted of single Python files that would be difficult to implement into the system. As such, the Python 2 code was converted to Python 3 manually, and refactored to follow a python package by myself. This new library was shared-alike, and is now available on GitHub. [4]

Absolute Difference (AD) This is a full reference metric, defined as $AD(I,I') = Mean(I - I')$. This calculation was implemented using Numpy manually.

Blind Image Quality Index (BIQI) BIQI is a no-reference metric. Rather like NIQE, there was no existing BIQI library implementation. It also consists of a pretrained classifier element, which would be hard to replicate without the correct processing or training data. The training set used is the LIVE IQA dataset, which is public domain but requires permission to access which would take quite a bit of time for a small component of the project.

BIQI consists of two steps: the first is to assess image distortion, and the second is to analyze the quality.

As such, an existing implementation was found on GitHub but for Python 2. [5] This was refactored to work with Python 3, and the code included as part of the existing GitHub repository.

Gradient Magnitude Error (GME) GME is a full reference metric, and it calculates the error corresponding to the gradient magnitude between the two images. The gradients ΔI and $\Delta I'$ are calculated using a Sobel filter, kernel size 5 in both X and Y directions. The Sobel filter is calculated using OpenCV within our implementation.

Using these gradients, the metric is calculated as $GME(I,I') = Mean((\Delta I - \Delta I')^2)$. In our implementation, Numpy is again used to calculate both the mean and also the squared difference.

Gradient Phase Error (GPE) GPE is a full reference metric, which like GME uses the Sobel filter as a basis. Using four different Sobel filters, gradients in the x and y direction for both I and I' are calculated. These are ΔI_x, ΔI_y, and $\Delta I'_x$, $\Delta I'_y$.

look into this because I think GME a GPE m have be switche during imple- mentati for som reason.

Using these direction gradients, $Mag_I = Magnitude(\Delta I_x, \Delta I_y)$ and $Mag_{I'} = Magnitude(\Delta I'_x, \Delta I'_y)$ is then calculated using the gradients shown above.

Using this, the metric is defined as: $GPE(I,I') = Mean((Mag_I - Mag_{I'})^2)$.

High Low Frequency Index (HLFI) This considers the frequency of specific low and high frequencies within a single image. It is therefore classed as a no-reference metric. First, the image is put through a discrete fourier transform process using OpenCV. The frequency spectrum is then shifted and a final magnitude spectrum is calculated using Numpy. This magnitude spectrum is defined as $M(i,j)$, where the shape of M corresponds to the image I.

Using this spectrum, a sum of the low frequencies, from $(1,1)$ up to (i_l, j_l) is calculated by cycling through each pixel in the spectrum. These two upper constants for the low frequency band are defined with respect to the shape of M. Given the shape of M is (X,Y), $i_l = 0.15 \cdot X$ and $j_l = 0.15 \cdot Y$. For each location (i, j) within the limits, the magnitude spectrum is summed to produce a final value called lfs which is the low frequency sum.

A similar process is followed for the high frequencies. However, instead of starting at 1, we start from i_h, j_h and finish at (X,Y). The constants i_h and i_l are calculated in a similar manner to before, only adding one to the result. Therefore $i_h = Ceiling(0.15 \cdot X + 1)$ and $j_h = Ceiling(0.15 \cdot Y + 1)$, where $Ceiling(x)$ rounds a value x up to an integer. The high frequency sum hfs is calculated through this process.

Based on this, the metric can be defined as $HLFI(I,I') = \frac{lfs - hfs}{\sum_{i,j=1}^{i,j=(X,Y)} |M|}$.

JPEG Quality Index (JQI) The JQI quality index is used to understand the nature of JPEG compression in an image, specifically with regards to JPEG images being blocky (a nature of the DCT based compression), and blurring components (due to high frequency DCT coefficients being lost).

First the blockiness feature B is calculated as average differences across block boundaries.

EXPLA MORE

Two metrics are then used to define the activity of the image signal. The first is A, which calculates the average absolute difference between in block image samples. Then, Z is the zero crossing rate. .

EXPLA MORE WHAT THESE

As these above features were calculated individually for horizontal and vertical features, they can now be fused together. This is done simply by averaging them.

$$B = \frac{B_h + B_v}{2}$$
$$A = \frac{A_h + A_v}{2}$$
$$Z = \frac{Z_h + Z_v}{2}$$

The final calculation uses a number of constants which are defined as: $\alpha = -245.8909$, $\beta = 261.9373$, $\gamma_1 = -239.8886$, $\gamma_2 = 160.1664$, $\gamma_3 = 64.2859$.

Bringing the constants, and the above calculations together, this yields the overall definition of the metric:

$$JQI(I,I') = \alpha + \beta \cdot B^{\frac{\gamma_1}{10000}} \cdot A^{\frac{\gamma_2}{10000}} \cdot Z^{\frac{\gamma_3}{10000}}$$

While other definitions of this can be utilized, this yielded good prediction performance in the paper [25]. In our implementation, this was implemented using Numpy.

Laplacian Mean Squared Metric (LMSE) LMSE is a full reference metric that considers the edge differences to measure quality. A high LMSE value implies that the input image is of poor quality.

This requires a Laplacian function, which is defined in [8] as:

$$Laplacian(I(m,n)) = I(m+1,n)+I(m-1,n)+I(m,n+1)+I(m,n-1)-4\cdot I(m,n)$$

This function is calculated for each (m,n) in an image. The current implementation uses the above function written in Numpy to produce the Laplacian output. [8]

Once this operator has been defined, it can then be used to calculate the overall metric:

$$LMSE(I,I') = \frac{\sum(Laplacian(I)-Laplacian(I'))^2}{\sum Laplacian(I)^2}$$

Mean Angle Magnitude Similarity (MAMS) MAMS is a full reference metric that considers the similarity between magnitudes of the angles within the image. First, a matrix of scalar products are calculated, such that $S = I \cdot I'$.

Then, a matrix of magnitude products are calculated $M = |I| \cdot |I'|$. All zero values within M are then set to 1, in order to avoid divide by zero errors in a later step.

Using both S and M, the angles at each position on the image were calculated. Each angle produced from the arccos operation is then subtracted from 1. The operation yields a matrix α of angles in radians:

$$\alpha = 1-((\frac{2}{\pi})*\arccos(S/M))$$

While the angles have been produced, the magnitudes are now needed. These magnitudes N are calculated as follows:

$$N = 1-\frac{|I-I'|}{255}$$

Using this, the mean angle similarity for each part of the image can be calculated: $T = 1-\alpha \cdot N$.

Using the values from T, the overall metric output is the mean value from T. $MAMS(I,I') = Mean(T)$.

This implementation was done using Numpy functions only.

Mean Angle Similarity (MAS) This metric is almost identical to MAMS, being a full reference metric that considers the similarity of the angles. Unlike the above, we don't consider the magnitude of these similar angles.

Using the calculations from the MAMS metric, the steps up to the calculation of α are needed only.

Then, MAMS can be defined as $MAMS(I,I') = 1.0 - Mean(\alpha)$.

Maximum Difference (MD) The maximum difference metric takes the maximum difference between I and I' (making it a full reference metric). It's defined as:

$$MD(I,I') = Max(|I-I'|)$$

This was implemented using Numpy functions.

Mean Squared Error (MSE) MSE is a reference metric, calculating the mean squared error between I and I'. More formally:

$MSE(I,I') = Mean((I-I')^2)$

Implementation was done using Numpy.

Normalized Absolute Error (NAE) NAE is a full reference metric, calculating the absolute error and normalizing it with respect to I. It's defined as $NAE(I,I') = \frac{\sum|I-I'|}{\sum|I|}$. [7] Implementation was done using Numpy.

Naturalness Estimator (NIQE) NIQE is a no-reference metric.

Explain the libra implem tation a roughly how it works.

Normalized Cross Correlation (NXC) NXC is a full reference metric. It is defined as $NXC(I,I') = \frac{\sum(I \cdot I'_{transposed})}{\sum(I^2)}$. [7]

Peak Signal to Noise Ratio (PSNR) PSNR is the peak signal to noise ratio achieved between I and I'. It's classed as a full reference metric. First, the mean squared error is calculated (as before). Using this, the metric is defined as: $PSNR(I,I') = 10*\log(\frac{\max(I^2)}{MSE(I,I')})$. [7]

R-Averaged Metric (RAMD) RAMD is a full reference metric, which calculates the average of all differences that are smaller than a specific value R. In our implementation, $R=10$.

We create a matrix $D=|I-I'|$.

With this, this function can be expressed:

$$RAMD(I,I')=\frac{\sum_{D(x,y)<=R}^{(M,N)}(D(x,y))}{R}$$

In the above definition, (M,N) is the shape of both I and I'. (x,y) is defined as the coordinates of each pixel. For this calculation, the difference is calculated for each pixel location, with any value that's greater than R being set to zero. Then, the remaining values are divided by R.

Structural Content (SC) SC is a full reference metric. It's calculated using the following equation: $SC(I,I')=\frac{\sum(I^2)}{\sum I'^2}$. Since I' is a Gaussian blurred version of I, this will yield a sum of the high frequencies.

Spectral Magnitude Error (SME) SME metric is a full reference metric that utilizes Discrete Fourier Transform (DFT) to analyze the frequency. We define a function $FourierMag(img)$ which calculates the DFT of the image img, shifts it, and produces the magnitude of the transform. The magnitude image produced is a 2D array of shape (X, Y), where X represents the real plane and Y represents the complex plane.

For image I and I' respectively, the transformed magnitude M, and M' is calculated, such that: $M=20*\log(FourierMag(I))$, and $M'=20*\log(FourierMag(I'))$.

Using this newly defined variable, the function definition can be easily defined as:

$$SME(I,I')=Mean((M-M')^2)$$

In the implementation, OpenCV was used to conduct the DFT process, with Numpy providing additional fourier transform functionality to calculate the magnitude. In addition, Numpy was utilized to easily conduct the remaining matrix operations (such as $Mean$, log, and subtractions).

Signal to Noise Ratio (SNR) SNR is a fairly standard reference metric. It requires the calculation of the Mean Squared Error, which has been defined above. This shall be referenced as a function $MSE(I,I')$. In addition to this function, the shape dimensions of the images are defined as (M, N). It's assumed that both images have the same shape.

Therefore, the definition follows:

$$SNR(I,I')=10*\log(\frac{\sum I^2}{N*M*MSE(I,I')})$$

Spectral Phase Error (SPE) The SPE metric utilizes DFT based transformations which were implemented using OpenCV, with Numpy being used to provide the fourier transform shift functionality.

While the SME metric defined above uses magnitude, this metric utilizes the phase. We define a function $FFTShift(I,plane)$. When carrying out the DFT transform, two planes are produced: the real plane, and the complex plane. The real plane is the x component of the image, while the complex plane is the y component of the image. As such, we carry out 4 different calculations:

$I_{shiftX}=FFTShift(I,real)$ $I_{shiftY}=FFTShift(I,complex)$

$I'_{shiftX}=FFTShift(I',real)$ $I'_{shiftY}=FFTShift(I',complex)$.

These shift components can be used to produce the magnitude for both I and I'. This can be considered a function $Magnitude(I)$, which yields the magnitude based on the x and y components for each element in the matrix.

Using this function, the metric can be easily defined as:

$$SPE(I,I')=Mean((Magnitude(I)-Magnitude(I'))^2)$$

.

Structural Similarity (SIM) SSIM is a full-reference metric. This metric was implemented using the *skimage* library. The library implementation can be found as a function called *compare_ssim*. Rather than implementing this from scratch, the library implementation was used. Into this function, I and I' are passed in as arguments.

SSIM is a perception based model, based on how a human would perceive a given image. One of the important factors in perceived human image quality is the structure. The method proposed considers three elements of similarity between the two images: luminance similarity, contrast similarity, and structure similarity. [26]

Total Corner Difference (TCD) TCD is a full reference based metric.

A function called $GetNumberOfCorners(I)$ was created, which counts the number of corners found in an input image. This function applies a corner Harris detector, and counts the number of corners yielded from this detector. The Harris Detector used was the OpenCV *cornerHarris* detector.

Using this function, we can define the final metric as:

$$TCD(I,I')=\frac{|GetNumberOfCorners(I)-GetNumberOfCorners(I')|}{max(GetNumberOfCorners(I),GetNumberOfCorners(I'))}$$

More al
how we
used Co
Harris
to make
it work
(becaus
this fun
is weird

Total Edge Difference (TED) TED is a full reference based metric. It's similar to the TCD metric, but instead considers the total edges. Since edges aren't easily countable, it's easier to consider the edges which exist in I but not in I'. Edges are isolated using a Sobel function as previously used. This yields the following definition for the metric:

$$TED(I,I') = Mean(|I - I'|)$$

.

The Sobel filter is calculated using an OpenCV function, with the remaining operations using Numpy backed calculations.

Visual Information Fidelity (VIFD) VIFD is a full reference based metric that considers the quality at 5 different scales. The number of scales can be variable, but in our instance 5 was chosen.

Explain why we chose 5 vs other e.g. loo the Mat implem tion of

The overall definition of this metric is:

$$VIFD(I,I') = \sum_{s=1}^{s=5} \frac{\log_{10} \frac{1+g^2 * \sigma_1^2}{sv_{sq}+\sigma_{nsq}}}{\log_{10} \frac{1+\sigma_1^2}{\sigma_{nsq}}}$$

Here, s is the scale with $s=1$ being the input images, while $s=2$ means that each image has been Gaussian blurred $s-1$ times before having this calculation being done. When this compounding Gaussian process is carried out on I, the result at a specific time step is called ref. With I', it's called $dist$. A few other values are necessary:

$$g = \frac{\sigma_{12}}{\sigma_1^2 + \epsilon}$$

$$sv_{sq} = \sigma_2^2 - g * \sigma_{12}$$

$$\sigma_1^2 = Gaussian(ref * ref) - (GaussianBlur(ref))^2$$
$$\sigma_2^2 = Gaussian(dist * dist) - (GaussianBlur(dist))^2$$
$$\sigma_{12} = Gaussian(ref * dist) - (GaussianBlur(ref) * GaussianBlur(dist))$$

There are also several constants, which are:

$$\epsilon = 1 \times 10^{-10}$$
$$\sigma_{nsq} = 2$$

The RRED Problem One of the metrics proposed in [7], Reduced Reference Entropic Distance metric (RRED), was problematic to implement due to the lack of Python implementation available. The only implementation available was written in Matlab, and relied on Matlab only libraries. While some had Python equivalents, the steerable pyramid functions did not have a Python equivalent, therefore increasing the workload massively for this single metric. Therefore, the metric was ignored for the WIQA test, since the test performance without the RRED metric was within the expected range.

Testing these metrics Once these metrics were produced, testing needed to be done to ensure that these metrics were outputting sensible values that could be relied on. Two images were selected at random from the NUAA dataset (using the full image path, rather than the dataset manager created beforehand). These metrics were then carried out individually on these two images. The values were then compared, to see if they differed, and to detect any potential image quality difference. Testing each of the metrics was slightly challenging as no ground truth data for each metric existed.

C.3 Classifier

Initially, a support vector machine was trialed to test whether a different classifier from the paper [7] would work best. However, the performance was fairly unreliable. Adding a grid search to find the optimal parameters still performed fairly poorly, yielding only 70% accuracy when training and testing on the same dataset. This was poor compared to the expected results from the original paper. Therefore, the classifier was quickly switched to use Linear Discriminant Analysis (LDA), which provided far better performance.

For both classifiers, existing SVM and LDA implementations from the *sklearn* library were used for their performance.

The default sklearn LDA settings were initially used, but the results produced still weren't ideal. By using an eigenvector based solver, and automatic shrinkage, the LDA model performed far better, yielding the results shown in the results section.

Once a model is trained, it needs to be saved to be used for the testing process. Saving was achieved using the Python *pickle* package. For small models, pickle is ideal since it easily creates a Python object that can be loaded/written without additional boilerplate. Since the sklearn classifier models aren't too deep, pickle works. If the models were deeper, then Python's recursion depth limit would hit, causing errors.

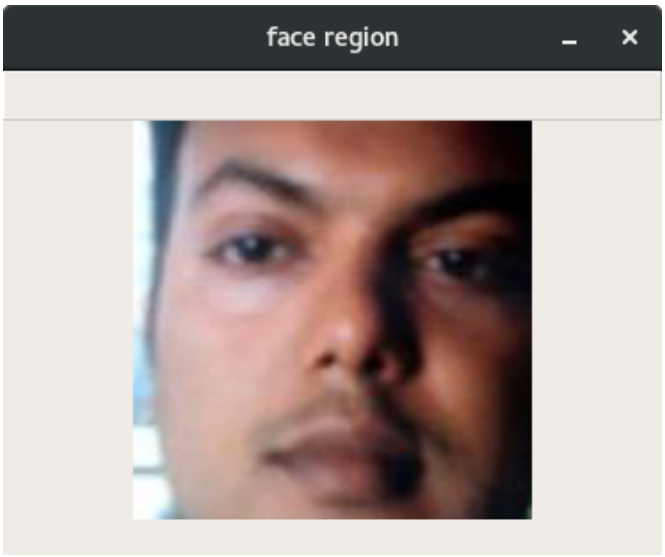


Figure 6: *This is the output produced from the preprocessing step of the ResNet based method.*

D 2D based CNN Liveness Test

Recently, 2D convolutional neural networks have had great success in image classification tasks. Systems that use these technologies are currently deployed in real world systems. Therefore, by adapting these models to classify facial liveness, it might be possible to obtain a new model, designed for the web (with both low latency and good accuracy).

D.1 Overview

As discussed earlier in the paper, there are several different models available for image classification. Unlike ImageNet based classifiers, we only have 2 output classes, which are *real* and *fake*. Therefore, the final layer of the model would be different, but the remaining details would be identical. Unlike the image quality metric, this metric concerns itself with the facial data, to understand which faces are real, and which ones are fake.

D.2 Data Preprocessing

As this metric concerns itself with the facial structure, it became necessary to isolate the face from the input image. Furthermore, as part of the CNN model a required input size was needed to be specified. The preprocessing step therefore needed to produce a facial image that had fixed dimensions.

This was achieved using the *face_recongition* package. Using this package, an image is input and a set of bounding boxes is yielded. The largest bounding box (by area) is found, yielding the face. Initially, the bounding box was simply cropped and resized to be a square (even if it wasn't initially a square), before being passed to the model. This yielded very poor results, due to the difficulty in learning the constantly changing dimensions. As a result, a new method of face isolation based on bounding boxes was designed:

Given a bounding box $B = (top, bottom, left, right)$, a new bounding box B' which has square dimensions can be created by finding the square side width s. Mathematically, this is defined as:

$$s = Max(bottom - top, right - left)$$

Using this, the new bounding box is defined as:

$$B' = (top, top + s, left, right + s)$$

By following this method, the model appeared to perform better overall, and produced non-skewed images.

However, simply isolating the face isn't enough. A fixed size image was needed, in order to be fed into the network. After much experimentation, an image size of $(224, 224)$ was decided, as smaller dimensions failed to produce adequate results. The resizing was completed using OpenCV. The output of this step is shown in Figure 6.

D.3 The Model

AlexNet Initially, an AlexNet based classifier was designed. While AlexNet performance on the ImageNet dataset isn't as high as other models, it provided an ideal starting point to understand the difficulty of the classification. It was eventually found that AlexNet performed relatively poorly on facial liveness classification, and therefore a switch was made to a more complex and better performing model.

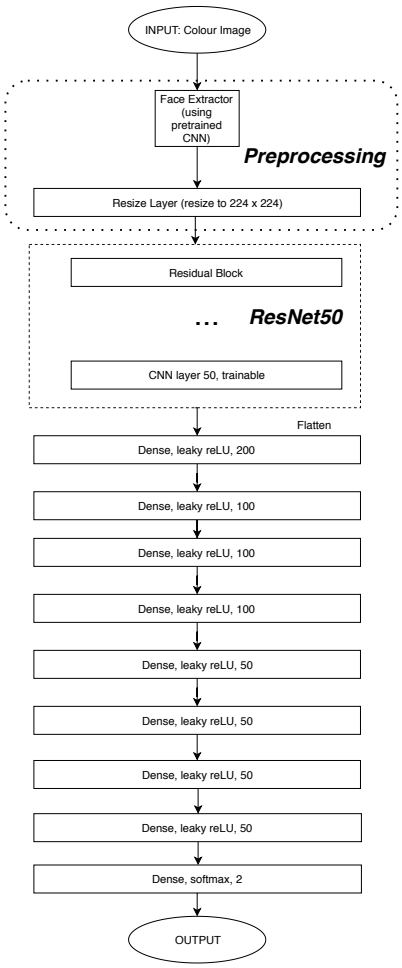


Figure 7: *The 2D CNN test architecture. We take the face image, resize to a fixed size, and put through ResNet50. The two last CNN layers of this ResNet are trainable. The output of this network is flattened and fed through a deep feed forward network, yielding one output (which is the liveness score as before).*

Residual Networks Residual Networks were chosen over VGG and Inception due to their better performance on ImageNet, coupled with their ability to be easily deployed without excessive memory/computation requirements. A ResNet 50 architecture was decided upon, as 50 layers seemed reasonable in terms of the available computational power that was available for this proof of concept. It was deemed that if ResNet 50 works, then ResNet 101 might perform equally, if not slightly better. A ResNet 50 architecture is an ideal proof of concept.

While training a model from scratch might have advantages for some applications, a pretrained ResNet50 model from the Keras standard library was used. This was done due to the fairly small amount of data available (and therefore using a pretrained model can avoid overfitting), and also to save time learning the basic features that are present in all generic image classification models. Training the entire ResNet50 model would require a large number of parameters to be adjusted, so to further reduce the complexity of the training process, only the last layer was set to trainable, with the other layers remaining static.

Feed Forward Classifier While a CNN is ideal for processing images, a fully convolutional approach to classification didn't seem appropriate. The output from the ResNet 50 model was flattened, and fed directly into a feed forward neural network. While pooling layers were considered, these only take the maximum/minimum/average of specific sections, reducing the dimensionality, and the loss of data means the feed forward layer has less information to act upon. Therefore, it was instead decided to use no pooling layers. The output from the ResNet is flattened, and fed into an 8 layer feed forward neural network directly. The first layer of this network has a very large number of nodes, while the number of nodes is reduced towards the network output.

With the initial experiment, the output layer consisted of a single node, with a sigmoid activation function. The entire network was trained using a binary cross-entropy loss function. The network accuracy yielded was fairly reasonable, but the confusion matrix showed that the model was outputting *0.0* (fake) for each value, no matter the input. This was a problem, as the network wasn't suitably learning the dataset. As a result, three changes were made to solve the problem, which are outlined below.

The Problem with Activation Functions Initially, the *reLU* activation function was used on all internal nodes within the feed forward network component. While reLU is very popular for improved speed over traditional activation functions such as tanh and sigmoid, for negative values it has a problem. ReLU is positive for all positive values, but 0 for all negative values.

While all inputs were non-negative, weights could lead to these values becoming negative. There is a known problem called "dying ReLU", where some neurons output 0 due to having a negative input (the weights and inputs for all input neurons sum to a non-positive value). Therefore, the model was able to learn 0 (fake) easily, but couldn't learn real (1.0) at all. [16]

This problem was counteracted by changing activation function from reLU to Leaky ReLU. Unlike reLU, leaky reLU isn't zero when negative. It requires an input value α, which is used as the gradient of the line when less than zero. [16] For this model, $\alpha = 0.3$. The activation function is defined formally as:

$$Activation(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha \cdot x, & \text{otherwise} \end{cases} \tag{1}$$

The Problem with a Single Neuron Output In order to mitigate the problem of outputting all zeros, the output method of the network was changed. Instead of a single neuron, giving a realness value, two neurons were used. These neurons would use the softmax activation function, rather than the sigmoid activation function, to ensure the sum of the neuron outputs would be zero (therefore giving a probability that can be used). The use of two neurons to encode the realness means that the nature for a network to learn 0 is slightly removed, as there will always be a non-zero output for one of the neurons expected. The encoding method used here is called one-hot encoding in the literature.

Normalization and Dropout Batch Normalization and dropout were both used within the feed forward network component to improve learning and reduce the risk of overfitting. Overfitting was a concern due to the small dataset that was used for training. Batch Normalization layers were added between each dense layer, which resulted in increased accuracy of the model.

Throughout the experimentation process, the model was found to be overfitting. Therefore dropout was added (with a dropout level of 0.3) between each dense layer. This reduced the effects of overfitting.

The overall outcome is shown in Figure 7. The normalization and dropout are not visible in the diagram, since they are only used for training.

D.4 The Training Process

Optimizer Initially, the Standard Gradient Descent was used as the optimizer. This was chosen due to the findings of [27], which advised that SGD was better than the Adam optimizer for producing models that generalize. However, after experimenting further the Adam optimizer appeared to generalize better for this project. Using the SGD optimizer, the validation accuracy fluctuated with fairly poor accuracy results overall. The Adam optimizer didn't fluctuate, and the accuracy yielded was much higher.

It's possible that SGD might perform better over longer periods of time with a very small learning rate, but due to time constraints and the need to experiment and show a proof of concept, Adam proved better for the purposes of this project.

Loss Function Initially, while the model had a single neuron output, the binary cross entropy loss function was used as it's suitable for a single binary output. However, when the two neuron output model was introduced, the loss function was changed to use categorical cross entropy as this was the most suitable for a categorical one hot encoding method.

Data Generators Due to the large amount of data being processed, it was necessary to use a data generator to conduct the preprocessing on the fly. While the preprocessing could have been saved to disc, the preprocessing steps were changed throughout the project, therefore defeating the benefits of such an optimization.

Initially, Keras' built-in ImageDataGenerator was used, as it allowed for a preprocess function to be passed in. While this works for a preprocess function that yields the same shape image as was input, Keras' ImageDataGenerator does not support resizing images within the preprocess function (e.g. for cropping).

Initially, a Lambda expression within the network was used to resize the image, but this led to problems with saving/loading models (due to Tensorflow being necessary). Therefore, a custom data generator was written from scratch. With this new generator, it wasn't necessary to follow this size constraint within the preprocess function.

E 3D Face Reconstruction Liveness Test

E.1 Overview

While 2D methods work well for traditional paper/screen based attacks, they are not designed for detecting the wearing of 3D masks. With 3D printing becoming more common, and automatic mask generators also becoming available online, this type of attack is becoming more common.

The method proposed merged recent developments in facial reconstruction with recent deep learning models for classifying 3D data, to investigate a proposed architecture for a 3D classifier. Unlike previous methods, face data is captured in 2D using a standard device camera, before being reconstructed and then classified.

E.2 2D Image Preprocessing

The 2D preprocessing step followed a similar process to the previous liveness test proposed. A CNN based face detector produced a set of bounding boxes, and a square face image was produced following the same method as was proposed previously. Unlike the previous method, the image was resized to a fixed size of (192, 192) in color. This size was necessary due to the requirements of the pretrained model.

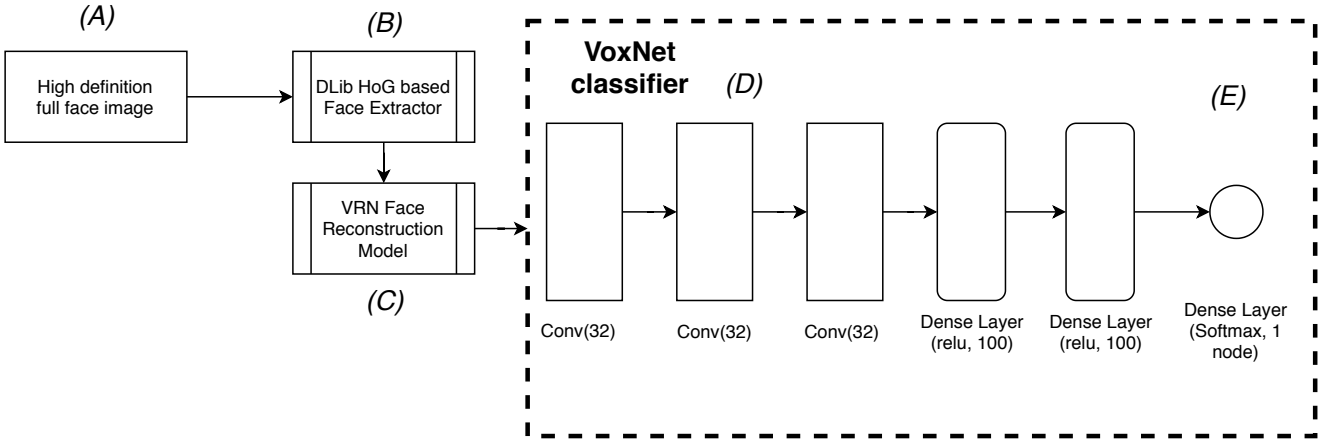


Figure 8: *This is an overview of the 3D classifier.* ***(A)*** *a high resolution image is input into the classifier.* ***(B)*** *The image goes through a HoG based face detector. The bounding box of the face is extracted, and the image is cropped. The image is then resized to be 192x192 pixels, which is what's required by the VRN process.* ***(C)*** *The pretrained VRN face reconstruction model takes an image input, and outputs a voxel representation. Some preprocessing from the VRN model is necessary to convert an occupancy grid into a voxel representation (this is done here rather than in the VoxNet model).* ***(D)*** *The VoxNet classifier uses several 3D convolutional layers, along with a couple of Dense layers to classify.* ***(E)*** *The output of the last dense layer is simply a single number defined as the certainty of realness. 1.0 implies the model is certain that the input is real, while 0.0 implies the model is certain that the input is faked.*

E.3 3D Facial Reconstruction

A method of obtaining facial structure from a 2D image was required. While video based methods such as Structure From Motion exist, they require a video input which has a large variety in motion, which might not be available/obtainable. However, recent research has developed a method known as VRN, which is a neural network designed to produce 3D facial structure from a single 2D image. [11] While the original work was built using the Torch deep learning framework, a pretrained model with appropriate source code exists for Keras. [17] This pretrained model takes an image of size (192, 192) in, and produces a 3D voxel based representation.

In order to correctly modularize the system, the *FaceVoxelBuilder* class was produced to correctly load the pretrained model, and build the voxel structure for either 1 image, or multiple images (in the case of batch training).

While implementing this component of the system, there was a problem: Tensorflow didn't build the objects from the pretrained model correctly in some cases, as certain components of the model didn't exist on the graph. This was solved by manually creating the predict function by calling a private helper function (*model._make_predict_function()*). This was a known issue with the version of Keras that was being used, and the solution found was specified by the developers of the framework. [12]

E.4 3D Classification

While 2D image based classifiers exist, and perform well, this isn't the case with 3D image classifiers. A VoxNet based classifier was chosen due to the adequate performance on the SUOD dataset (of 69%), coupled with the relatively small model size. [18] While this model can in future be improved and adjusted, this classifier was believed to produce results that could show a proof of concept, and determine whether this method would be suitable for a web-based liveness system. The final architecture is shown in Figure 8.

E.5 The Training Process

Training was conducted using a binary cross-entropy loss function, with an Adam based optimizer. Binary cross entropy was used since there was only a single output node, compared with a categorical approach.

In order to conduct training, a Data Generator was used to minimize memory constraints, since the entire dataset didn't need to be loaded into memory after being preprocessed. Instead, only a single batch would be preprocessed on the fly, thus reducing memory usage considerably.

While training, a problem with the built in data generators was encountered. While 2D images were being input, a 3D representation was required as output from the data generator, something that the Keras ImageDataGenerator wasn't able to deal with. Initially, the VRN model was included as part of the classification model. This would see a 2D image being input into the model, and the model would reconstruct to 3D and classify all in the same model. While this is a more monolithic approach, it would avoid the need to switch out the ImageDataGenerator. However, when implemented this didn't work: the 3D reconstruction process required non-tensor operations (more specifically, the use of the stack command), which could not be included as a network layer.

Therefore, the final solution was to instead conduct the 3D reconstruction as a preprocessing step, and led to the replacement of the Keras ImageDataGenerator with a custom generator, designed to output 3D from 2D input. This custom generator would resize the images, reconstruct the 3D representation, and return the appropriate batch yielded. This solved the problem.

E.6 The Dataset

Unlike the previous two liveness tests, this liveness test relied on a completely different dataset, the Mask Attack Dataset (MAD). [6] The goal of this liveness test was to detect mask based attacks (and more widely, 3D based attacks), and this dataset was used due to it being more representative of the problem than the Replay-Attack and NUAA datasets (which are designed for 2D based attacks).

Unlike the 2D based methods, two datasets weren't used in a cross validation style assessment of the method, so instead the MAD dataset was split by subject (the person being spoofed): half of the subjects were used for the training/validation sets, and the other half were left for training purposes.

Subjects 1,2,3,4,5,6,7 were used for training purposes, and 8,9,10,11,12, 13 were used for testing purposes. Splitting up the dataset in this way was necessary to correctly assess the classifier, to ensure results give a true indication of how well the model learned the features, rather than how well the model learned the dataset.

F Consolidation Layer

Each liveness test individually might have some degree of error. Merging the results of each metric together into some classifier to produce a more reliable outcome would be ideal. While probability based methods (such as Bayesian calculations) would be feasible, this only works for a larger number of liveness tests. In some cases, very few liveness tests (minimum 2) might be used.

The solution proposed is to use a classifier, pretrained with the results yielded from the previous liveness tests. After training, this consolidation classifier would be able to spot patterns between each liveness test, and hopefully yield a better result by fusing the results of each individual test together.

Implem
DETAI
NEEDE

IV RESULTS

For both liveness tests, cross dataset validation/testing was conducted. Each model was trained using the entire NUAA dataset, and the Replay-Attack test set was used to measure the results shown below. In the case of the 2D Convolutional Neural Network (CNN), a validation set was required to ensure the model performed well, so in this case the Replay-Attack devel set was used. It must be noted that no overlap occurs between the Replay-Attack devel and test sets, to prevent the risk of these results being invalid. In order to visualize the result as a demonstration, a script was written using OpenCV calls to produce an overlay on top of the image. The results of these models in a faux-production environment can be seen in Figure 9.

	Image Quality Liveness Test	**2D CNN Liveness Test**
Accuracy (%)	87.0	71.2
True Negatives (%)	37.5	71.5
False Positives (%)	12.5	1.37
False Negatives (%)	0.5	22.5
True Positives (%)	49.5	4.69

Table 1: *Table of results, showing test accuracy with the percentage of test results falling into the specific category defined in the confusion matrix (obtained using sklearn).*

	Image Quality Liveness Test	**2D CNN Liveness Test**
Time to load (s)	0.00104	7.44
Time to classify 1 image (s)	1.40	1.04

Table 2: *Table of results, showing the wall clock time for the load and predict phases of both liveness tests.*

A Testing Process

A.1 Datasets

The Replay-Attack test dataset was used to access the results of each liveness test. This dataset contains no overlap with the set that was trained on, therefore ensuring that the results are a true indication of the generalised performance.

A.2 Accuracy

Top-1 accuracy is an ideal metric to judge the performance of the system, since a high Top-1 accuracy in the results would show how often the liveness test yields the correct results.

A Top-1 accuracy figure of 50% implies that the model is simply randomly yielding output (since there are two cases), which isn't ideal. A figure of 70% implies that the model is reasonably yielding the correct result and is feasible, but might need minor changes or more training to yield a better result. A figure of over 85% implies that the model is yielding the correct results most of the time. The higher the accuracy figure, the better, when being run on an unseen dataset.

A.3 Confusion Matrix

While accuracy is a reasonable metric, it's also important to visualise how often each model selects each case (fake/real), and whether the selection is correct or incorrect. There are four important terms with the confusion matrix. These are:

Figure 9: *Outputs from the* live_webcam_output.py *file, being run on two different videos from the Replay-Attack test dataset. The model predicted the correct output here.*

(a) *The system correctly predicting a real piece of data. Image is from a video in the Replay-Attack test dataset.*

(b) *The system correctly predicting spoofing in the input video. Image is from a video in the Replay-Attack test dataset.*

True Positives This is the percentage where the model predicts that an input is real, where the input is real. Therefore, the liveness test has identified a user as real correctly. This figure should be above zero, and a reasonable figure.

False Positives This is the percentage where a model predicts an input is real, where the input actually contains a spoofing attack. This is a concern, since for a security focused liveness test, this should be mitigated as much as possible.

True Negatives This is the percentage where a model predicts an input is fake, where the input contains a spoofed image. This is an indication of how often a model predicts that an input is faked. This again should be reasonably high.

False Negatives This is the percentage where a model predicts an input is fake where it's actually real. This isn't ideal, but this is the preferred option when a liveness test performs poorly, since this would only cause inconvenience for the user, rather than a potential security breach.

B Image Quality Liveness Test

Overall, the Image Quality Test performed as expected with reference to the initial paper. Unlike the original paper however, instead of isolating the face from the input image, the entire image was used. While isolating the face might perform well, using the entire image might provide further subtle information about the image quality.

The overall results, shown in Table 1 show fairly good performance on the Replay-Attack test dataset, with an accuracy of 87%. This is in line with what was expected from the paper [7]. While accuracy gives an overall account of the results, it doesn't show the overall performance.

The level of true negatives and true positives respectively are fairly high, but the number of false positives was slightly higher than expected (i.e. where the model predicts someone to be real where they are actually not). What's interesting to note is the low number of false negatives (indicating someone is fake where they are real), which in our security conscious case isn't wanted (since inconvenience isn't as much of a problem as security). However, the overall performance was better than expected. While accuracy is important, the time taken for the model to classify a single image is also important. The LDA method took 1.40 seconds to classify a single 2D image, which is within the specified 2 seconds bound. Also, the model itself is very small and therefore can be loaded in a very minute amount of time (0.001 seconds). The bulk of the time required is in the calculation of the metrics, rather than the classification process, and therefore future speed optimizations could consider improving the calculation of the metrics. These results can be seen in Table 2.

C 2D Convolutional Neural Network Liveness Test

The overall performance of our liveness test can be seen in Table 1. While not as accurate as the previous model, the model still detects true negatives with a fairly high percentage. While the overall accuracy of this model could be improved with further training or by deepening, the results show that this basic architecture is feasible. In the security conscious environment of facial liveness, false negatives, while potentially annoying to the end user, are an ideal result compared to false positives (which would imply that someone is real when they are fake). While false positives still exist, their number is small compared to the number of negatives. Furthermore, the number of positives is small, meaning that this model leans on the side of caution, something that is ideal. Therefore, this model would be secure and accurate enough with further training (and potentially replacing ResNet50 with a deeper ResNet model).

The time taken to load the model from memory was the main bottleneck in terms of computational performance, as loading the model into memory for the first time took 7.44 seconds. However, once the model was loaded, the time taken to predict a single image

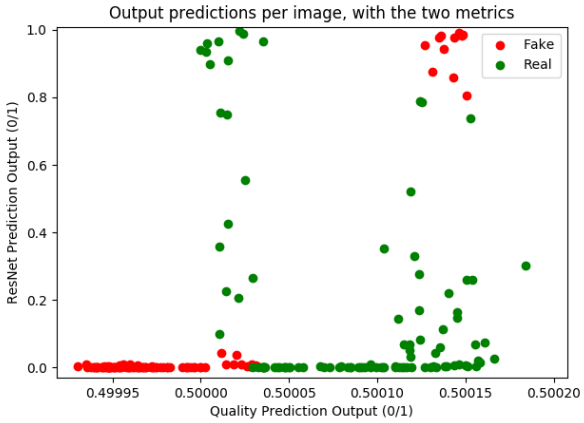


Figure 10: *A plot of ResNet output against WIQA output. As one can see, for two metrics a perceptron could be used due to the linearly separable nature of the data (that is, a single line could be used to separate the real and fake classifications in some cases). Effectively, with two metrics this would be acting as an OR gate (with fuzzy calculations).*

is very small, on average taking 1.04 seconds, which is even smaller than the W-IQA test, and almost half the specified 2 second time, which is ideal. These results can be seen in Table 2.

D 3D VoxNet Liveness Test

This method had some major performance challenges. The 3D facial reconstruction worked well, producing the desired outcome. However, the 3D classifier (VoxNet), had numerous problems. The original VoxNet model, proposed in [18], used 32 filters for the Conv3D layers (with a 32x32x32 input). However, using 32 filters in our model for each Convolutional Layer (alongside the larger data input than expected), led to memory errors and wasn't able to be trained on the training hardware. Reducing the 3D reconstruction resolution wasn't an option, since the entire model would require retraining (which would take lots of time, resources, and wasn't feasible). Furthermore, as seen with the 2D CNN method, reducing the resolution might also reduce the accuracy. Therefore, the remaining solution was to reduce the number of filters down to a small 12 filters per convolution layer.

During the training process, with a variety of learning rates, the accuracy of the model didn't leave 50% with a differing amount of batch sizes. This indicates that the model wasn't learning the features correctly (potentially due to the lack of filters in the Convolutional Layer).

While the classifier itself didn't work and yield any meaningful results, it's also important to consider the time taken for a 2D image to be reconstructed. To reconstruct a single image from 2D to a 3D face structure, it took 5.39 seconds, which already exceeds the 2 second constraint set by the specification.

Due to the poor accuracy with a low number of filters, the excessive amount of memory required for unknown accuracy results, and the large time taken to reconstruct the 3D structure (not including the added time to theoretically classify the result), this liveness test clearly isn't feasible for the purpose of a web liveness service.

E Consolidation Layer

As shown in Figure 10, plotting the results together with the probability outcome (rather than the class outcome), yield a plot where the two classes could approximately be divided up with a single line. Therefore, as expected, the relationship between predictions is linearly separable.

While investigating this further, using an LDA as a consolidation layer yielded fairly ideal results: 75% accuracy to be precise.

However, it's important to note that while this works for two liveness tests, we can't test this method for a larger number of liveness tests. While accuracy was fairly reasonable, using more metrics could yield improved accuracy.

F Live Application

While numerical figures are useful to measure the results, it's also important to show performance on a completely unseen subject (through the use of a webcam). An OpenCV based script was written to, in real time, access the output of the liveness tests and overlay the results on the screen with their predictions.

The successful results can be seen in Figure 9. While not numerical, it yielded a large number of useful insights into the real time nature of such a system, and it's practical deployment.

Benefits of CNN facial isolation The CNN facial isolation was found to be very good at selecting the facial area ready for preprocessing. Even with part of the face covered up, it still yielded a reasonable facial crop, which could then be fed into the network. Also, the head being placed at different angles also yielded a reasonable face isolated image that could be used for the CNN metric.

Resolution problems with WIQA test While the test datasets contain a small number of different resolutions, the differences in resolutions are minimal. Therefore, using a completely different image resolution of 1280x720, which was completely unseen by the classifier previously. This led to the model predicting each input as real, which was incorrect.

The solution taken was to resize this image down to a more reasonable 640x480, which yielded far better performance in real time. However, in future iterations, resolution is a factor that needs to be considered, and potentially the resolution should be included in the classified vector. Furthermore, more resolutions should be tested to yield a reasonable classification.

Uncertainty of the CNN liveness test While the CNN liveness test could correctly classify a specific facial input, it would occasionally change the prediction to an incorrect prediction as frames were stepped through. This was even observed with minimal change in the input image. Since the prediction of 1 image takes minimal time, it might be ideal in future to take a set of 8 frames, and output the mode classification from this batch, this way smoothing out the prediction. Alternatively, the CNN liveness test might need more training to assert the difference between real/fake in a more varied set of lighting conditions and resolutions.

Show u
tainty h
someho

Problem in running liveness tests in series Since we have two liveness tests, the script runs the WIQA test first, before running the CNN based liveness test. This is fairly slow, as each liveness test requires their own resources and time to complete. It might be more ideal to parallelise this in future, to yield better performance. While single-computer parallelisation might not be the best option (since each classifier requires some degree of multithreading), multiple systems could be used to speed up this process.

V EVALUATION

A Usefulness of our system

Quality Test The Quality Test performed well, and is suitable for a web based liveness service due to the high 87% accuracy achieved. True positives and true negatives (that is, correct predictions for both real and fake) were high which means that the classifier is classifying correctly. However, the high false positive rate shows a slight cause for concern, since the model in 12.5% of cases classified a fake image as real, which isn't ideal for our security focused solution. This could potentially be solved by adjusting output variable of the classifier (using a figure representing 'fakeness' rather than realness). This would hopefully lead to more false negatives, which are inconvenient but more secure for the system.

In terms of computational performance, a 1.40 second time for classifying a single image is within the limits of the expected 2 seconds, and could be improved further with parallel based methods, and not relying on libsvm for the BIQI metric.

2D CNN The 2D CNN Test performed adequately. While the accuracy was lower than expected (at 71%), the classifier itself still performed better than random. Furthermore, the model was very good at classifying true negatives, with a total percentage of 71.5% being true negatives. While the model itself had a higher than expected false negative percentage, this is just inconvenient for the user rather than a security problem. The model showed it could classify true positives, but this figure was fairly low. This could be improved by improving the training process: ensuring each input image has a correctly identified face (as some images without a detectable face would have been left to classify the entire image, just resized to the input size). The less noise in the input data, the better the potential results in the future. Furthermore, using a larger ResNet model, such as ResNet-101 could lead to better results.

In terms of performance, classifying an image is very quick, but the time taken to load the model is what took the most time (due to memory needing to be allocated and written to). In production, providing a model was preloaded and ready to accept input, the computation time of this metric would be very fast, and therefore be ideal for inclusion in a liveness web service.

3D VoxNet Liveness Test As seen from the results, this method of liveness test isn't feasible. With extra training, while the accuracy of the model could be improved, the real time memory requirements don't seem feasible. Based on the VoxNet paper, the accuracy achieved with a 32x32 VoxNet classifier on the SUOD dataset (a 3D dataset of objects) is 69%. While this accuracy could be justified with reasonable computational and memory characteristics, this further proves that this method isn't feasible in the current state. In the future, a new approach of detecting 3D attacks is necessary.

B Improvements

Representing 3D Attacks While our system works fairly well for 2D based attacks, performance for 3D based attacks definitely needs work. While our VoxNet based method didn't yield any meaningful results, there is a chance that a 2D image could yield results with 3D based attacks (using a residual network on a static image to detect minor mask-based imperfections), or alternatively considering a sequence of images using LSTMs to detect changes in movement. This however would require video input, meaning the NUAA dataset wouldn't be useful.

Preventing Source-Quality Web Browser Attacks Current approaches followed (including ones in this paper) assume that the video capture and liveness processing is conducted securely, without any interference from malicious actors. However, in the case of web browsers, the video capture process might not be secure as the client-side code can be easily tampered with, and therefore shouldn't be trusted fully. The solution to this is a random video input, something requested by the server and required to be contained within the video (similar to a CSRF token in web forms). The answer to this is contained within an existing liveness metric: a motion based one. The process of face flashing would be ideal, as a specific color could be requested, and expected to be visible in the frame. Alternatively, head tracking and expecting a specific set of motions would also be suitable, but as this would require user input might not be favorable.

Existing BIQI implementation Currently, the BIQI implementation requires the use of a subprocess to call libsvm based commands on the system. Instead of using standard output, files are used which cause a reduction in speed. Speed isn't the only problem here, as scaling would become a problem due to the existing implementation.

To fix this problem, a reader would be needed to import a trained libsvm model into sklearn, and the code would then need to be refactored to use the updated sklearn model.

VI CONCLUSIONS

This project showed that creating a facial liveness service for the web is a feasible idea, and performs fairly well for 2D attacks, with adequate accuracy and computational requirements. The image quality liveness test is accurate and fast, while the ResNet based method is a feasible idea and performs adequately, and could be improved further to improve the accuracy. For 3D attacks however, the proposed VoxNet based model performed badly and is not recommended for inclusion in a liveness test web service.

References

[1] A. Canziani, A. Paszke, and E. Culurciello. An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*, 2016.

[2] I. Chingovska, A. Anjos, and S. Marcel. On the effectiveness of local binary patterns in face anti-spoofing. september 2012.

[3] T. Choudhury, B. Clarkson, T. Jebara, and A. Pentland. Multimodal person recognition using unconstrained audio and video. In *Proceedings, International Conference on Audio-and Video-Based Person Authentication*, pages 176–181. Citeseer, March 1999.

[4] R. Collins. Ohmgeek/video-quality. `https://github.com/OhmGeek/video-quality`, Dec 2018.

[5] Cvley. Image quality. `https://github.com/cvley/ImageQuality`, Apr 2016.

[6] N. Erdogmus and S. Marcel. Spoofing in 2d face recognition with 3d masks and anti-spoofing with kinect. september 2013.

[7] J. Galbally, S. Marcel, and J. Fierrez. Image quality assessment for fake biometric detection: Application to iris, fingerprint, and face recognition. *IEEE Transactions on Image Processing*, 23(2):710–724, Feb 2014.

[8] M. Gulame, K. Joshi, and R. Kamthe. A full reference based objective image quality assessment. *Int. J. Adv. Electr. Electron. Eng*, 2(6):13–18, 2013.

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

[10] A. Izvorski. aizvorski/video-quality: Video quality metrics, reference implementation in python. `https://github.com/aizvorski/video-quality`, Mar 2015.

[11] A. S. Jackson, A. Bulat, V. Argyriou, and G. Tzimiropoulos. Large pose 3d face reconstruction from a single image via direct volumetric cnn regression. *International Conference on Computer Vision*, September 2017.

[12] Keras-Team. Tensorflow backend - bug in model._make_predict_function(...), issue #2397, keras-team/keras, Apr 2016.

[13] A. Khalid. Facial recognition may boost airport security but raises privacy worries. `https://www.npr.org/sections/alltechconsidered/2017/06/26/534131967/facial-recognition-may-boost-airport-security-but-raises-privacy-worries?t=1556103517110`, Jun 2017.

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

[15] S. Kumar, S. Singh, and J. Kumar. A comparative study on face spoofing attacks. 05 2017.

[16] D. Liu and D. Liu. A practical guide to relu. `https://medium.com/tinymind/a-practical-guide-to-relu-b83ca804f1f7`, Nov 2017.

[17] P. Lorenz. Vrn torch to keras. `https://github.com/dezmoanded/vrn-torch-to-keras`, October 2018.

[18] D. Maturana and S. Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928, Sep. 2015.

[19] G. Pan, L. Sun, Z. Wu, and S. Lao. Eyeblink-based anti-spoofing in face recognition from a generic webcamera. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8, Oct 2007.

[20] K. Patel, H. Han, and A. K. Jain. Cross-database face antispoofing with robust feature representation. In *CCBR*, October 2016.

[21] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *arXiv preprint arXiv:1612.00593*, 2016.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[23] X. Tan, Y. Li, J. Liu, and L. Jiang. Face liveness detection from a single image with sparse low rank bilinear discriminative model. In *ECCV*, 2010.

[24] D. Tang, Z. Zhou, Y. Zhang, and K. Zhang. Face flashing: a secure liveness detection protocol based on light reflections. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*, August 2018.

[25] Z. Wang, H. R. Sheikh, and A. C. Bovik. No-reference perceptual quality assessment of jpeg compressed images. In *Proceedings. International Conference on Image Processing*, volume 1, pages I–I. IEEE, 2002.

[26] Z. Wang, E. P. Simoncelli, and A. C. Bovik. Multi-scale structural similarity for image quality assessment. In *in Proc. IEEE Asilomar Conf. on Signals, Systems, and Computers, (Asilomar*, pages 1398–1402, 2003.

[27] A. C. Wilson, R. Roelofs, M. Stern, N. Srebro, and B. Recht. The marginal value of adaptive gradient methods in machine learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4148–4158. Curran Associates, Inc., 2017.