Facial Liveness Testing: 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

April 19, 2019

***Abstract* —**

Context With password based authentication methods being subject to many attacks, facial recognition is an alternative, not relying on memory but instead on biometrics. However, with facial recognition comes face spoofing: methods to fool the algorithms into thinking one is someone different to who they are. In order for facial recognition to become more prevalent on the web, facial liveness is needed. With client side code comes the risk that the input could be tampered with, so server side services are ideal for security, and this leaves a few questions: what metrics are suitable for deploying in such a service, and how feasible is the construction of such a service?

Aims
- Verify the results of the Image Quality Assessment test.
- Assess the outcome Convolutional Neural Networks on classifying real/spoofed images.
- Design and implement a new 3D based liveness test, aimed to prevent mask attacks.
- Determine the outcome of fusing the three above methods together, and how successful this is.

Method
- The image quality assessment test was implemented in Python to consider the image as a whole
- A CNN based 2D liveness test was implemented in Python to classify facial structure.
- A 3D based liveness test was proposed and investigated as to its usefulness.

Results
- Image Quality Assessment test performed well, being in the 90% accuracy range over Replay-Attack test.
- CNN based 2D test performed adequately, yielding 76% accuracy over the Replay-Attack test dataset.
- The VoxNet based 3D liveness test performed poorly, and had various performance issues that means it's not currently practical to deploy.

Conclusions Overall, both the Image Quality assessment and CNN based 2D test are ideal in a web-based liveness test as a service system. Image Quality based metric individually yields impressive results, but the CNN based metric would perform well when working together with other metrics. 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

I INTRODUCTION

Currently, username and password authentication is commonplace throughout the web. However, username and password based authentication systems have a number of problems. Some common passwords can be broken using dictionary attacks, especially if they consist partially or entirely of a word in a standard dictionary. Furthermore, the process of shoulder surfing is possible (watching out for someone's password, and how they type it).

While there are different measures of detecting liveness, each method is specialized towards defending against a given attack. The aim of this project is to understand the existing liveness detection methods, which type of attack they aim to prevent, and how effective they are. Once this has been achieved, the aim shall be to bring each of these methods together, hopefully improving the effectiveness of such a system by incorporating multiple methods.

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 also confirm the success of the Image Quality Assessment method for Facial Liveness, and provide an improve

II RELATED WORK

As defined in [10], the types of face spoofing attacks can be described under three sections: Photo Attack, Video Attack and Mask Attack.

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. [10].

With video-based facial recognition systems, motions of some form can be used to determine whether the person is real or spoofed, such as blinking, head movement and others. In the method defined in [4], structure from motion was used on the 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. They also extended this by fusing this 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 are video-only, quality based methods are useful for both videos and images (either by extracting key video frames or using all video frames and combining the results).

While there are various quality metrics that have been used, combining a large number of them can yield some increased accuracy. By combining 25 different metrics, , yielding the resulting metric values into a large vector, and using that as input to a classifier (an LDA), this yields fairly high accuracy. [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 facial liveness (both video and image based).

In particular, Convolutional Neural Networks are a key approach to this to learn features (e.g. texture based methods). Due to the existing datasets available, training CNNs has been difficult due to lack of data where over-fitting has been common. The method proposed in [13] 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.1 General 2D image classification models

Outside of the facial liveness field, image classification on the ImageNet dataset has proven popular and yielded some fairly good results.

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. [9] Overall deployability with Alex net is fairly good due to a fairly low number of compute operations (meaning faster compute time). [2] 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. [2]

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. [14] This is shown in Figure 1.

Therefore, this model has improved computational performance over VGG, making it a more suitable model for the web compared to VGG. [2]

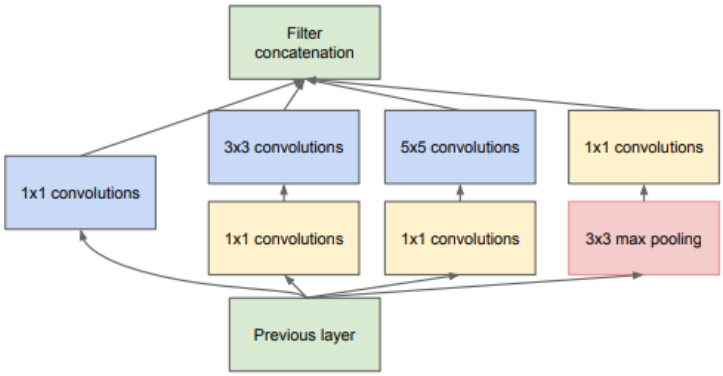

Figure 1: The Inception module with dimensionality reduction. Diagram taken from [14].

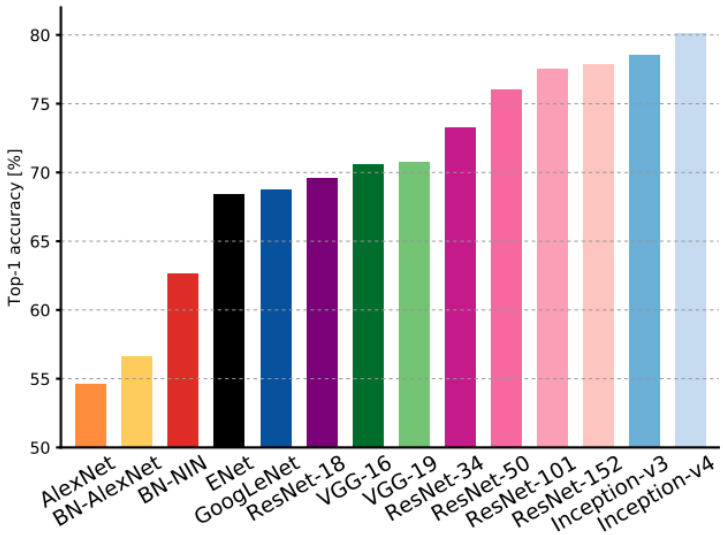

Figure 2: Top 1 accuracy vs network. Chart and results from [2]

Residual Networks Another key problem with deep convolutional networks on Image Classification is the vanishing gradient problem, where early layers have very small gradients during the training process and are therefore much more 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 efficent than VGG in terms of computational power needed. 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).

Explain Resnets better

Cite Residual Networks

In terms of Top-1% accuracy (the accuracy with the top output in a multiclass problem) shown in Figure A.1, newer version of the Inception model perform the best, with ResNet based classifiers not too far behind. AlexNet performance was fairly poor. Now compare this with the operations (the required computational power for each model), which is shown in Figure A.1. AlexNet and GoogLeNet require fairly few operations, in the $< 10 GOps$ range. The same is true for smaller ResNet based models (e.g. ResNet 50 and 34). Meanwhile, VGG requires a fairly large amount of operations ($> 30 GOps$), as do later versions of Inception.

A.2 Datasets

While models exist, in order to test these models data is needed. One of the most common and earliest dataset for facial liveness is the NUAA dataset, which consists of photos of 15 subjects, with faked photos (both flat and warped) being placed in front of the camera. [15]

In 2012, the Replay-Attack dataset was first released, which consists of 1,300 video clips of both photo and

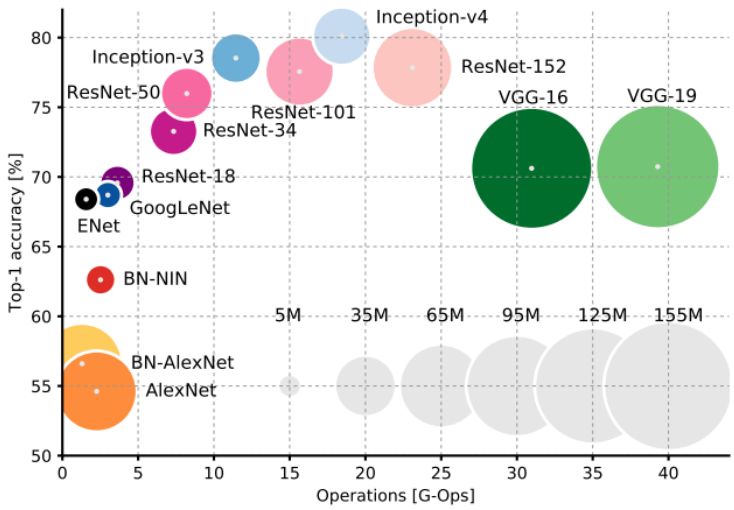

Figure 3: Top 1 accuracy vs operations, where network size is proportional to parameters. Operations figures are for a single pass (e.g. predicting given a specific input). Chart and results from [2]

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 'subdatasets': 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). [3]

A.3 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

Eye Tracking

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. [16] 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 colours 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. [10] These are much less prevalent, but with 3D printing becoming more mainstream, this could potentially get more prevalent in the future.

In 2013, the Mask Attack Dataset (MAD dataset) was released. [6]

Improve this section, explaining 3D based measures, more about the MAD dataset and attacks, etc.

III SOLUTION

A system deployed to the Cloud would require a few important criteria be met. The liveness methods used should be output a liveness score after no more than 2 seconds to ensure that such a system doesn't become painful to use by users. The liveness methods should not require any additional hardware (only a normal camera) so that the number of devices it can be used by can be maximised. Furthermore, each liveness test must have a reasonable accuracy of over 70%, to ensure that liveness predictions are fairly reliable.

Refactor into functional and non functional requirements.

The solution built should be focused on the metrics to be used, and fusion of these metrics. While speed is important, network latency and transmission time shouldn't be factored in, since these are incredibly dependent on a client's internet speed/image sizes, and therefore cannot be reliably tested. For small images and reasonably fast internet, latency would be fairly small.

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 classifer 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.

insert diagram of the overall system design (e.g. data, consolidation layer, etc)

A Shared Services

A.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, ReplayAttack and MaskAttack 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 customisation, to load specific subdatasets (such as 'devel' or 'test' in the case of the ReplayAttack dataset), or load data regarding specific subjects (in the case of the Mask Attack Dataset). The role of the H5Py dataset is to normalise the datasets into a specific format, to allow for easier dataset processing.

In addition to data normalisation, 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.

Diagram of Dataset management?

A.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 classifer.

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.

A.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.

B Image Quality Assessment Liveness Test

B.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 noticable 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, making it ideal for our use-case.

Reference the specification above

The classifier being used for our implementation is Linear Discriminant Analysis (LDA).

A visual explanation of the method can be seen in Figure B.1.

B.2 The Metrics

The original paper proposed that 25 metrics were used. However, our implementation used only 24, due to some implementation problems (which shall be detailed below). There are various different classes of metrics, but each metric is either a full reference metric, or a no reference metric. With a full reference metric, image I, and image I' are needed, where image I' is the gaussian blur of the original image I, with kernel size of $(5,5)$ and $\sigma = 0$. No reference metrics simply require a single image I.

Under the full reference metrics, some methods are based on error sensitivity (image differences, image correlations, image edges, image gradients, and frequency based comparisons), structural similarity between the images, or information theoretic based measures.

Under the no reference metrics, some methods are based on training a classifier (more specifically, a support vector machine), some consider distortion, and the final metric considers the natural scene.

Each of these metrics outputs a single floating point number. These numbers are assembled together into a vector. This vector is then later fed into the next stage (the classifier).

Most of the error sensitivity metrics were implemented manually using a mixture of Numpy and OpenCV. These were implemented manually based on the implementation guidelines available in the paper.

For the remaining metrics, some library implementations were used from scikit-image where possible, but in a majority of cases no existing library implementation existed, and as such a new library was needed. These methods are outlined below.

The custom PyVideoQuality library For the SSIM, VIF, and other metrics, no existing implementation existed. Looking back at the original papers for these metrics, a single matlab based implementation existed, which wasn't well documented (due to no comments or explanation). This was a problem, since without these metrics it would be difficult to implement this metric.

However, after searching online, a Python 2 implementation was found on GitHub. [1] Therefore, this was forked, and code was converted to Python 3 ready for integration into the project, which included added docstrings to assist the development process.

In order to allow separation of concerns, it was decided to build a seperate Python package for this. A release on GitHub was created to allow for easy importing from GitHub using pip. The final library implementation can be found on GitHub. [12]

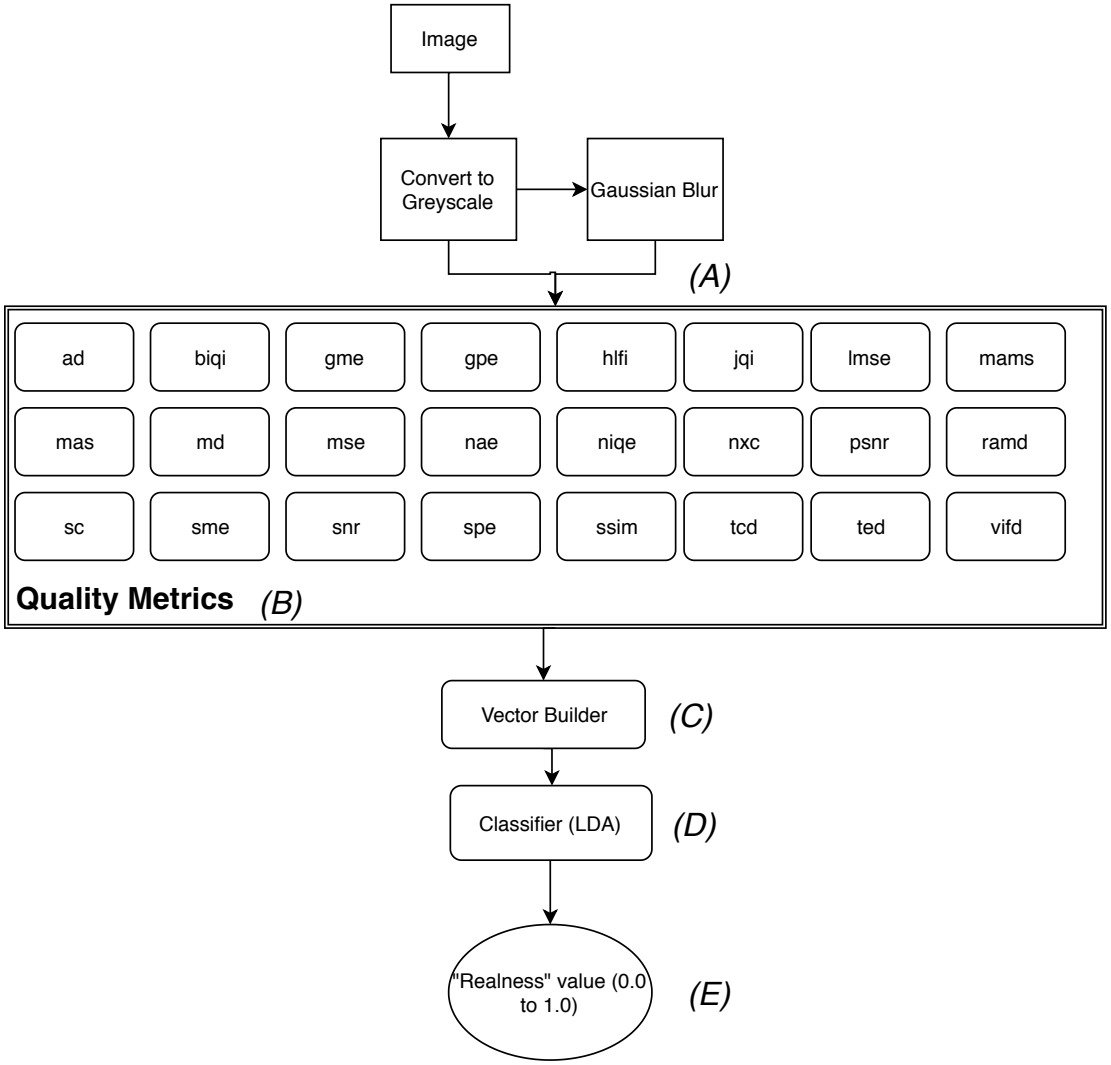

Figure 4: The architecture of the image quality liveness test. (A) The greyscale copy of the image, and a blurred copy of the image are input into each of the metrics. (B) The metrics are individually calculated, and a single value output from them. (C) These values build a 1D vector. (D) They are classified using an LDA classifier. (E) The realness value is 1.0 for real, and 0.0 for fake, or in between.

Blind Image Quality Index Code The Blind Image Quality index implementation followed a similar story to the PyVideoQuality library, as no library implementation existed. A manual implementation was considered, but in order to save time an existing Python 2 implementation was found as a gist [5]. This was converted to Python 3 manually. A manual implementation would have taken more time as this model uses a pretrained classifier, so it would have been required to implement the required preprocessing, train a classifier to an appropriate level, and then include this as part of a wider system. Therefore, using this existing implementation seemed the most suitable option due to time available. This implementation required the use of the *pywavelet* library, along with *libsvm* tooling from the command line to train and get the output of the metric.

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.

a big table showing each metric, and implementation details (library vs custom code)

B.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 pooly, 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.

C Residual Network based 2D liveness test

Recently, 2D convolutional neural networks have had great success in image classification tasks. Therefore, it might be possible to train a residual neural network (resnet) to classify for facial liveness tasks. The final architecture that was reached can be seen in Figure C. These sections outline specific areas, namely preprocessing, the model, and training, in terms of the implementation detail.

C.1 Preprocessing

Initially, the entire image was fed directly into the Residual Network, but this yielded fairly poor performance and generalisation. As a result, a HoG based face detector was used to find the largest bounding box in an image (of a person's face), and crop the image around this face structure. The HoG detector was initially used due to performance benefits, since a neural network based face detector would require slightly more processing power, and therefore time, to both train and predict with our model.

The image is then resized to the expected input size (required for the Keras image data generator), before again being resized to an image of shape (224, 224). While this worked, the bounding box width and height ratio did differ a large amount, which could potentially have yielded slightly poorer performance. Given a bounding box $B = (top, bottom, left, right)$, we can create a new bounding box B' which retains square dimensions, by first finding the square side width s. Mathematically, this is defined as:

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

Now we create a new bounding box, defined as:

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

.

By following this method, the model appeared to perform better overall, as rather than focusing on the overall image quality (which the previous model did), it would focus on the facial region.

During the process of training, it was noted that the HoG detector was missing approximately one eighth of the faces, therefore outputting the entire image, which could potentially have an impact on training and the accuracy of the model. Changing this to the CNN based classifier had surprising results: the computational performance increased, due to the underlying GPU acceleration, and the accuracy also improved due to the lack of random noise in the training set (through the generators). One thing to note with the CNN based face detector was to ensure upsampling was set to zero, otherwise memory issues would result (since the model and face detector model all need to be stored in GPU memory).

C.2 The Model

In order to simplify the process of training, an existing resnet model (ResNet50) was used, with only the final convolutional layer being set to trainable. This is because the initial convolutional layers contain the standard features contained within images, while the final one learns bundles of features. Internal feed forward activations use relu, while the external output uses the sigmoid activation function

Batch Normalization was therefore added to help make the make the model generalise further.

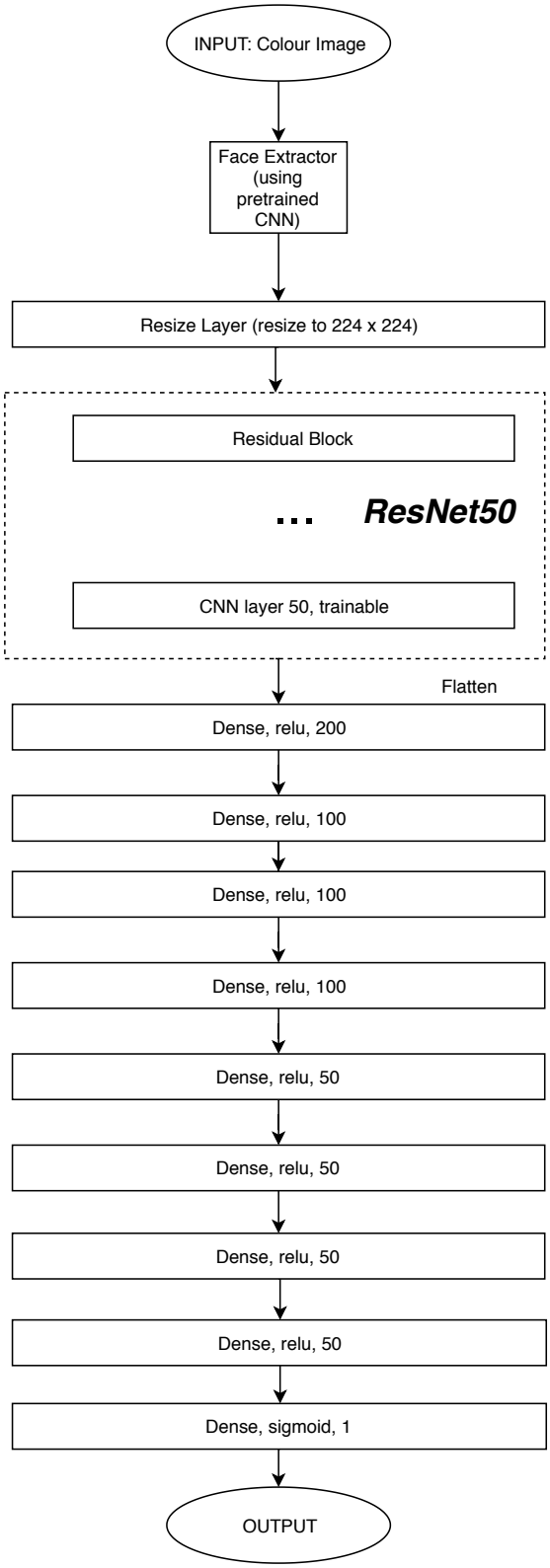

Figure 5: The 2D CNN test architecture. We take the face image, resize to a fixed size, and put through ResNet50. The two last CNN layers of this ResNet are trainable. The output of this network is flattened and fed through a deep feed forward network, yielding one output (which is the liveness score as before).

Throughout the training process, a binary crossentropy loss function was used, since there are only two possible classes, being predicted with a single number output.

The final architecture can be seen in Figure C. While normalization layers aren't visible, they are located in between each dense layer. The residual network is also simplified.

C.3 The Optimiser/Training

Initially, I was using the Standard Gradient Descent optimiser, due to the findings of [17] which stated that SGD was better than Adam for generalisation. However, after experimenting further I found that using an Adam optimiser with a learning rate of 0.001 led to an improved validation accuracy. With SGD the validation accuracy fluctuated, peaking at around 0.75 without increasing further. With Adam, the validation accuracy obtained in the results section was yielded, which was quite a large improvement.

D A system for preventing 3D spoofing attacks

While the systems before might go partially towards preventing 3D spoofing attacks, though primarily considering the 2D image, we now propose a method that is designed for classifying facial liveness based on a 3D point cloud. There are two distinct parts here: 2D to 3D, and 3D classifier. Each part shall be addressed individually, and then the two components combined.

D.1 2D to 3D Conversion

In order to classify an image/video, the 2D image needs to be converted to a 3D representation of a user's face. While 3d reconstruction is easier with videos (using structure from motion or other multiview based methods), there also exist image-based reconstruction methods such as vrn (**(author?)** 8) which are more specific and designed for reconstructing faces based on images. This also has benefits, as structure from motion is unable to reconstruct 3D from a single image, or from videos with very little motion.

The image was converted by first applying a facial detection algorithm on the image, and cropping the image down to provide only the face. This cropped image was then resized to be of size $(192,192)$, still in colour. This cropped and resized image was then fed through the VRN network. After this, the network output was filtered and stacked to provide the voxel input required.

The code to operate this can be found under the *liveness.vox.reconstruction* namespace within the code.

D.2 3D point cloud classification

Once the 3D reconstruction is obtained, one can then classify this using some model to produce the fake/real metric.

VoxNet takes in a point cloud and converts this to an occupancy grid. This is then fed through two convolutional layers, pooled, and then goes through a dense layer before reaching the classifier output (a dense layer with the k outcomes).

As a pretrained version of VoxNet wasn't readily available, the whole system was trained together from scratch.

D.3 Linking everything together

While each system is self-contained, linking them together took a little bit more work than expected. The models themselves couldn't be directly joined together, as VRN required extra postprocessing steps which couldn't be implemented using tensors within tensorflow. As such, the initial 2D to 3D conversion was required to be run as a preprocessing step.

To assist in the training phase, a generator was written in Python to conduct the postprocessing on the fly for each batch, which didn't require the entire preprocessing step to be done before training, thus reducing the peak memory usage problems. While an ImageDataGenerator was used previously, this isn't compatible with 3D, and therefore a custom module needed to be written.

Once the preprocessing had been completed, the preprocessed image was fed to the VoxNet.

E Datasets, and data management

F Training/Test Split

Using a single dataset is problematic due to the potential for overfitting, coupled with the risk that the classifier wouldn't generalise on the entire problem set, but rather the specific classifier.

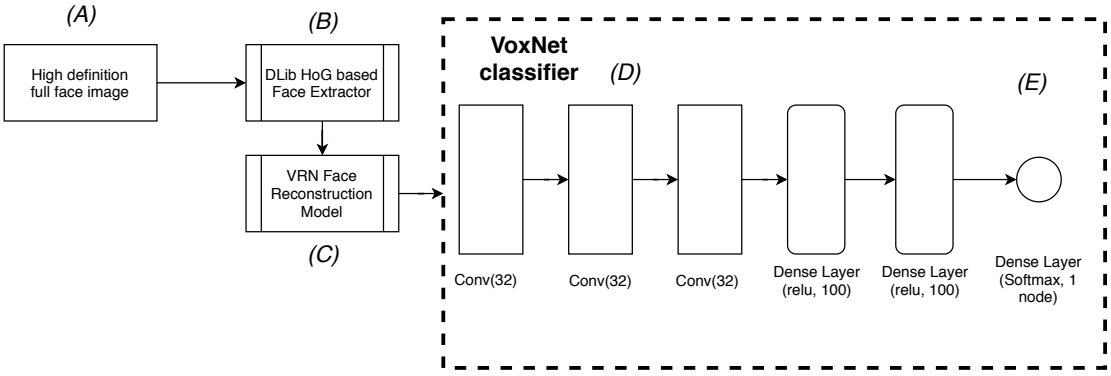

Figure 6: This is an overview of the 3D classifier. **(A)** a high resolution image is input into the classifier. **(B)** The image goes through a HoG based face detector. The bounding box of the face is extracted, and the image is cropped. The image is then resized to be 192x192 pixels, which is what's required by the VRN process. **(C)** The pretrained VRN face reconstruction model takes an image input, and outputs a voxel representation. Some postprocessing 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.

Therefore, it was decided to use one dataset for training, and one for testing. With each of the above classifiers proposed, the NUAA dataset (the entire dataset) was used for training, and the testing was carried out using the Replay-Attack dataset. If one was to use the ReplayAttack dataset for training, results were poor due to the lack of variation between the images (since we have a limited number of videos, with several frames from each video being similar, with a smaller number of subjects).

With this approach, the overfitting was fixed but the model would now underfit due to lack of data. Furthermore, using NUAA just for training and Replay-Attack for testing had poor performance on its own, due to differences between the overall datasets (e.g. resolution issues, as well as subject specific attribute differences). Therefore, cross-dataset validation was carried out. The entire NUAA dataset acted as training data, with the Replay-Attack devel dataset being used as the validation set. The test set was still seperate, as the Replay-Attack test set was used (which is seperate from the devel set, designed for testing). This yielded optimum performance for the 2D CNN test.

However, since an LDA doesn't have a validation set, the training set for the image quality metric was a merge between the NUAA and Replay-Attack devel datasets to improve performance.

F.1 Optimising Dataset IO

Due to the large dataset sizes, memory is a limit to the performance of the system. As such, optimisation needs to be achieved to allow working with large datasets within the specified memory constraints. This was achieved using *h5py*, which caches the dataset to a file, and the associated fetch statements are called when necessary. By doing this, the large dataset doesn't need to be entirely loaded into memory, but can instead use the hard disk drive. Some performance was sacrificed (due to the reduced read speeds of SSDs/HDDs compared to RAM), but this allowed for larger and better models to be produced on a smaller (and less expensive) machine.

Dataset io using h5py

Mention more detail here, and cite more information about our implementation e.g. with H5Py for caching, the use of generators for testing/training

G Visualisation and Demonstration

In order to visualise the overall outcome of facial liveness, a generic model

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

Liveness Test	Accuracy (%)	True Negatives (%)	False Positives (%)	False Negatives (%)	True Positives (%)
Image Quality	87	37.5	12.5	0.5	49.5
2D CNN	71.2	71.5	1.37	22.5	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).

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.

A 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 IV show fairly good performace on the ReplayAttack 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 concious case isn't wanted (since inconvenience isn't as much of a problem as security). However, the overall performance was better than expected.

Look into false positives/negatives, to see if we can get an outcome that performs better

B 2D Convolutional Neural Network Liveness Test

The overall performance of our liveness test can be seen in Table IV. While not as accurate as the previous model, the model still detects true negatives with a fairly high percentage While these results might not appear to be as ideal, this is partly due to the nature of the Replay-Attack dataset as each 20th frame was taken from the dataset. Due to this, when there is movement and where the face isn't visible by the camera, the image can't be correctly processed and therefore the entire image is used as the input to the network, thus yielding poorer performance than expected. This is only encountered with this metric due to the requirement that the facial extraction is successful.

C 3D VoxNet Liveness Test

As discussed in the method, this metric had several performance challenges. Applying VoxNet caused memory issues, in addition to yielding very poor performance (50% accuracy with a single dense layer output), indicating that the features weren't being learnt correctly. 69% accuracy was expected, as specified in the VoxNet paper [11]. There are a few reasons why this wasn't matched: firstly, the voxnet model we needed required large inputs: (192 x 192 x 192 x 3), which is far too high for easy and real-time computation.

Add time to conduct computation here (without multi-threading) - for predictions only, not training

V EVALUATION

A Usefulness of our system

show an example image of the system at work with image quality liveness

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.

Insert time here to isolate face/preprocess

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

Insert time here to conduct neural network prediction

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 colour 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 favourable.

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.

ADD Calculations here for speed with hard drive, SSD, and memory

Privacy/Legal Considerations While the security of a facial liveness cloud service is reasonable, there would be some legal concerns to consider in the deployment of an actual application. Face data, as with all biometric data, is classed as sensitive personal data under GDPR . As such, care would need to be taken in the development of such a system to ensure the data is transferred back and forth with encryption, and if data is at rest that it is encrypted (although storing a large amount of data wouldn't necessarily be ideal).

Find source

Furthermore, before deploying our current system, one major problem would be dataset licenses. The Replay-Attack dataset is licensed on condition of research, rather than for commercial use. As such, the deployed version of such a liveness based system would need to use a network trained with a new dataset that has the appropriate licensing (either a bespoke one, or one that is public domain).

Mention compliance of our current system (to some extent), and how it can be improved

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. The consolidation layer provides an ideal point of extension, allowing for multiple tests to work together and allow confirmation to prevent false positives (which would lead to security problems).

References

[1] Aizvorski. aizvorski/video-quality, Mar 2015.

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

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

[4] 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.

[5] Cvley. 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] 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.

[9] 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.

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

[11] 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.

[12] OhmGeek. Ohmgeek/video-quality, Dec 2018.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.