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 22, 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 [11], 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. [11].

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 [15] 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. [10] 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. [16] 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]

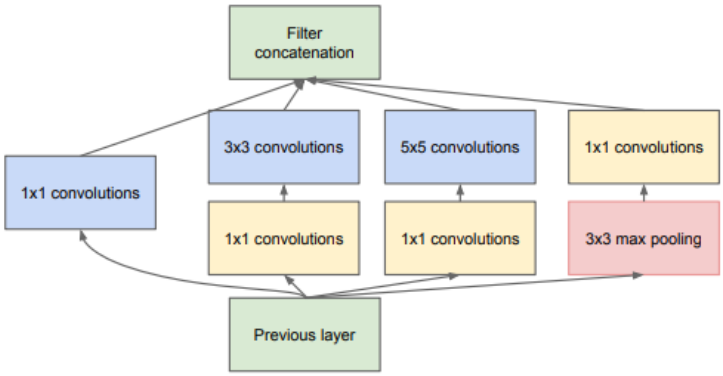

Figure 1: The Inception module with dimensionality reduction. Diagram taken from [16].

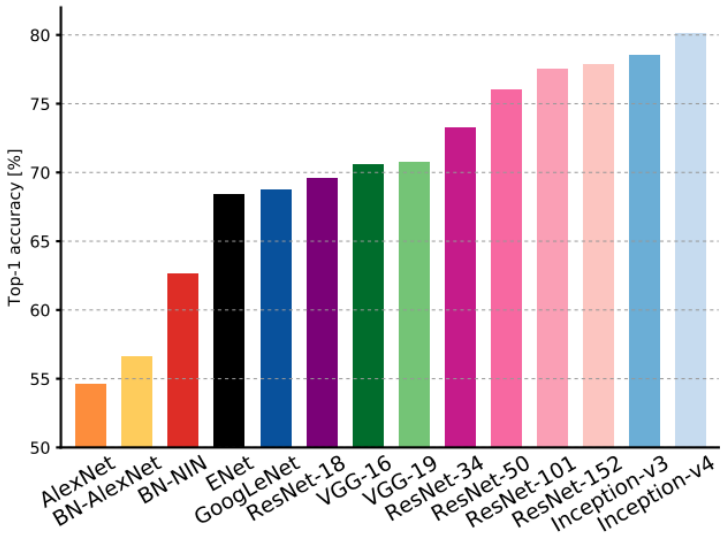

Figure 2: Top 1 accuracy vs network. Chart and results from [2]

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. [17]

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

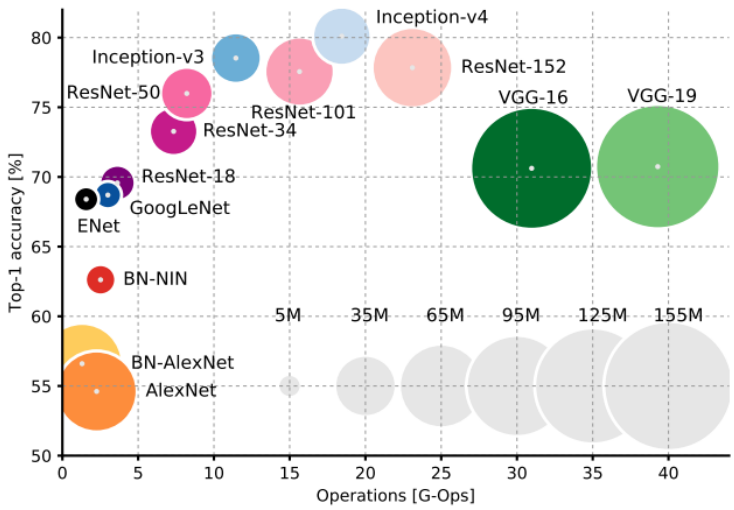

Figure 3: Top 1 accuracy vs operations, where network size is proportional to parameters. Operations figures are for a single pass (e.g. predicting given a specific input). Chart and results from [2]

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. [18] 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. [11] 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. [14]

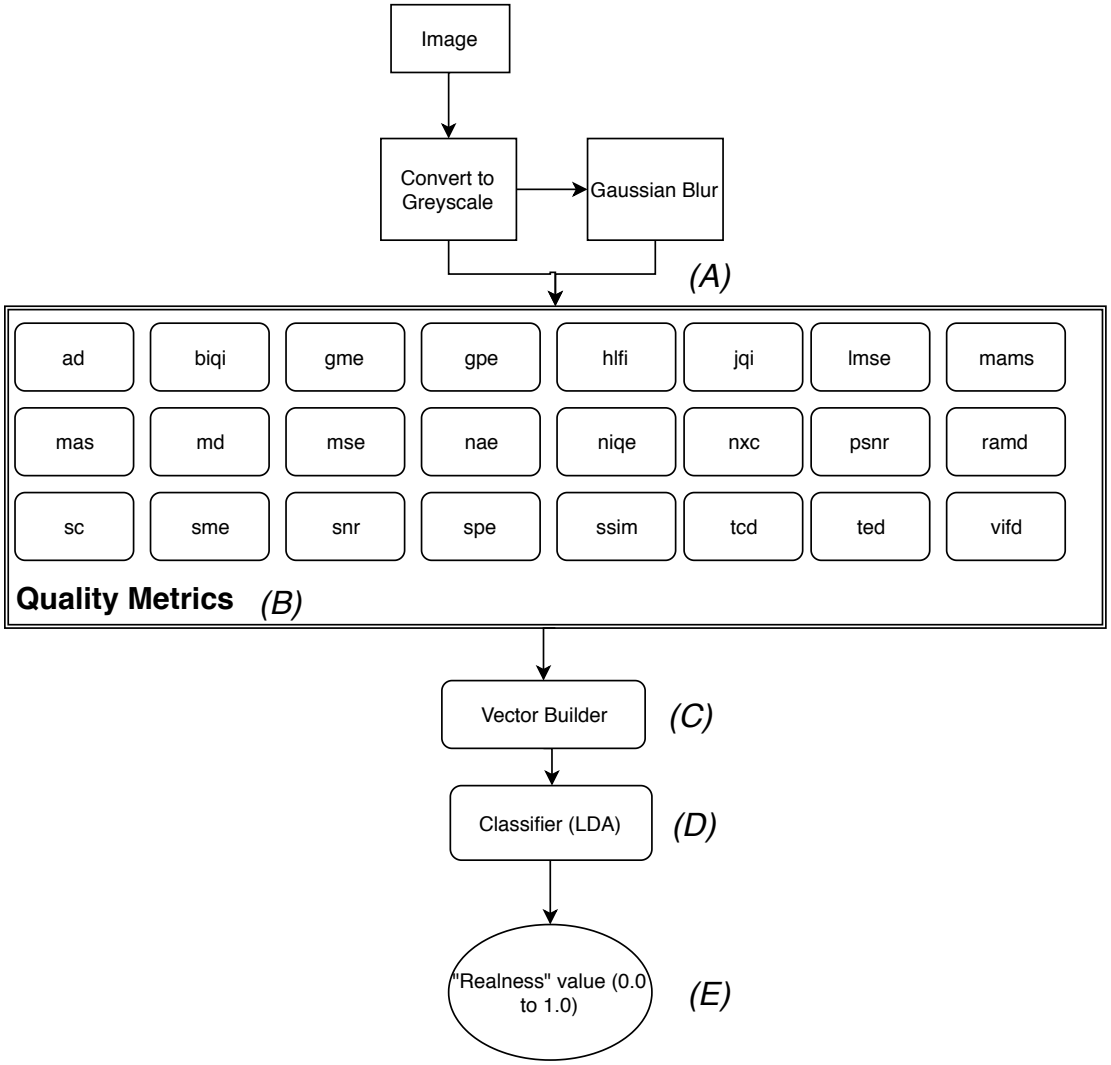

Figure 4: The architecture of the image quality liveness test. (A) The greyscale copy of the image, and a blurred copy of the image are input into each of the metrics. (B) The metrics are individually calculated, and a single value output from them. (C) These values build a 1D vector. (D) They are classified 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 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).

C.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.

C.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.

C.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.

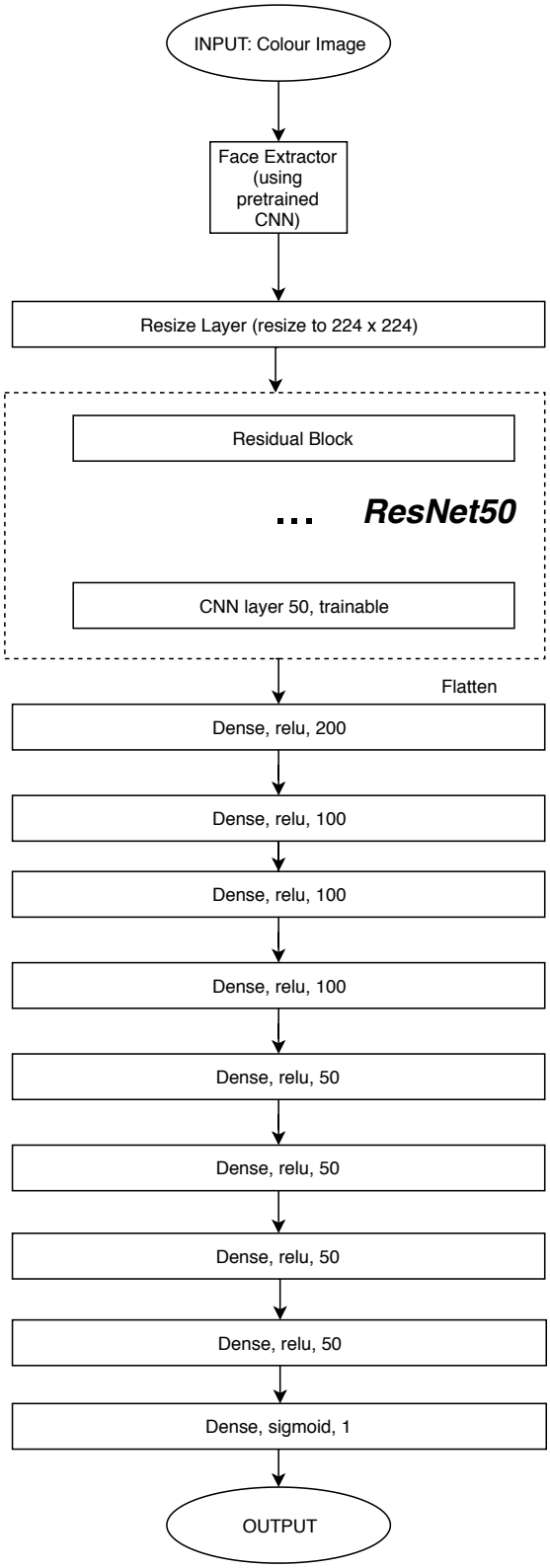

Figure 5: The 2D CNN test architecture. We take the face image, resize to a fixed size, and put through ResNet50. The two last CNN layers of this ResNet are trainable. The output of this network is flattened and fed through a deep feed forward network, yielding one output (which is the liveness score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

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.

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 crossentropy 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.

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. For this model, $\alpha = 0.3$. The activation function is defined formally as:

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

Find source for this, boht Relu and LeakyRELu

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 neurones 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.

ONe hot encoding source needed.

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 C.3. The normalization and dropout are not visible in the diagram, since they are only used for training.

C.4 The Training Process

Optimizer Initially, the Standard Gradient Descent was used as the optimizer. This was chosen due to the findings of [19], which advised that SGD was better than the Adam optimiser for producing models that generalise. However, after experimenting further the Adam optimiser appeared to generalise 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.

Cite categorical cross entropy better for one hot encoding.

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 optimisation.

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.

D 3D Face Reconstruction Liveness Test

D.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.

D.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 colour. This size was necessary due to the requirements of the pretrained model.

D.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. [8] While the original work was built using the Torch deep learning framework, a pretrained model with appropriate source code exists for Keras. [12] This pretrained model takes an image of size (192, 192) in, and produces a 3D voxel based representation.

FIX BIBTEX citation for Torch to Keras. It currently points to ResNets which is WRONG!!!

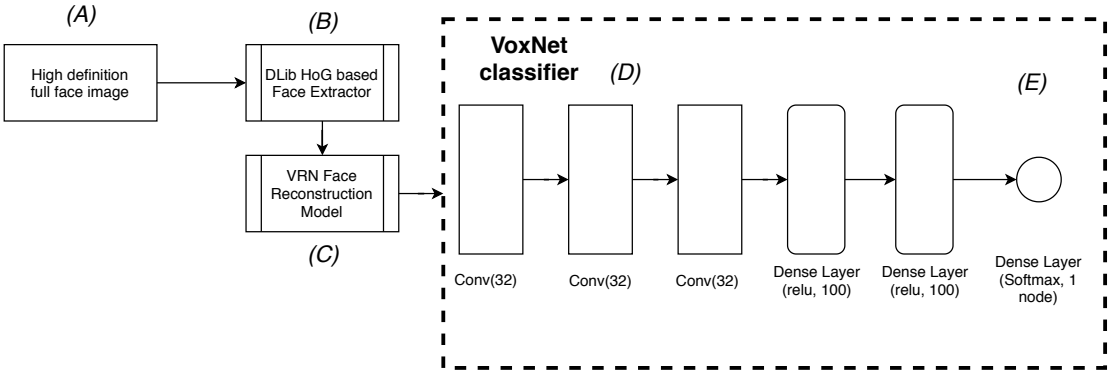

Figure 6: This is an overview of the 3D classifier. **(A)** a high resolution image is input into the classifier. **(B)** The image goes through a HoG based face detector. The bounding box of the face is extracted, and the image is cropped. The image is then resized to be 192x192 pixels, which is what's 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.

In order to correcly 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. [9]

D.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. [13] 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 D.4.

D.5 The Training Process

Training was conducted using a binary crossentropy 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.

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).

D.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 repesentative of the problem than the ReplayAttack 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.

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.

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.

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 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).

However, in order to be deployable, the processing time needs to be manageable. The processing time was measured on a standard desktop computer (the specifications of which were specified in the Solution section).

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

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

Insert time here to isolate face/preprocess

C 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 [13], 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.

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.

Figures for time to reconstruct a single image to 3D

V EVALUATION

A Usefulness of our system

Quality Test

2D CNN

Fill this in evaluating our system.

Fill in this evaluating our system.

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

[10] 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.

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

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

[13] 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.

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

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

[16] 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.

[17] 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.

[18] 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.

[19] 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.