CHALMERS
UNIVERSITY OF TECHNOLOGY

Anti-Spoofing for Facial Recognition-based Identification System

Using Convolutional Neural Networks to Detect Fake Faces

Master's thesis in Complex Adaptive Systems

Simon Liljestrand

Department of Electrical Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020

Master's thesis 2020:NN

Anti-Spoofing for Facial Recognition-based Identification System

Using Convolutional Neural Networks to Detect Fake Faces

Simon Liljestrand

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2020

Anti-Spoofing for Facial Recognition-based Identification System
Using Convolutional Neural Networks to Detect Fake Faces
Simon Liljestrand

Supervisor: Kenneth Jonsson, Smart Eye AB
Supervisor: John Finér, Smart Eye AB
Supervisor: Ahmet Oguz Kislal, Department of Electrical Engineering
Examiner: Marija Furdek Prekratic, Department of Electrical Engineering

Master's Thesis 2020:NN
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2020

Anti-Spoofing for Facial Recognition-based Identification System
Using Convolutional Neural Networks to Detect Fake Faces
Simon Liljestrand
Department of Electrical Engineering
Chalmers University of Technology

Abstract

Biometrics are becoming increasingly popular for use as authentication factors, owing to their high accuracy, and uniqueness to the individual, as well the convenience gained when not needing any imposed burdens such as carried tokens, or remembered passwords. However, such authentication factors need to be secured against new attack vectors. When facial recognition is used for authentication purposes, one potential attack vector is spoofing, that is, presenting some form of false imitation of the face of an authorized user. This thesis presents a means of detecting fake faces, intended to be integrated in an authentication system on a lightweight platform deployed as a component of a driver monitoring system. The solution presented uses image frames under two different active light conditions from a NIR-camera (Near Infrared) to classify the face as real or fake. Spoof attacks consist of faces printed on paper and held up in front of the camera, as well as subjects wearing latex masks. Classification of a face as live (real) or spoof (fake) is done using a lightweight neural network, to allow deployment on the sort of lightweight system suited for the setting of driver monitoring. On a small-scale test set, preliminary results indicate a classification accuracy of 100 % for live faces, 99.7 % for printed faces and 94.1 % for latex masks. The overall accuracy is 98.9 %. These results indicate that the solution presented could be successfully used to detect spoofing attempts in the context of authentication based on facial recognition, but further investigations with larger and more diverse data sets are encouraged.

Keywords: convolutional neural network, machine learning, image processing, computer vision, lightweight software, spoofing, security, mask, face, facial recognition, detection, authentication, attacks

Acknowledgements

I am very grateful to Smart Eye AB, for giving me the opportunity of working on this exciting project. I would like to thank Kenneth Jonsson for valuable guidance, discussions, reflections and input with regards to all kinds of technical details throughout the project. I would also like to thank John Finér for supervision of the project. I want to thank Oguz Kislal for guidance with regards to administrative and academic details during the project, as well as good feedback and guidance with regards to the final report and presentation of the project. Finally, I would like to thank my examiner Marija Furdek Prekratic for taking on this project, and for the feedback received to improve the thesis.

Simon Liljestrand, Gothenburg, May 2020

Contents

List of Figures

List of Tables

1
Introduction

1.1 Problem description

As technology has increased our ability to accurately determine personal identity based on various biometrics, they have become increasingly popular as authentication factors. The accuracy of identification using biometrics means that it can ensure authentification and non-repudiation, while the fact that biometrics are intrinsically tied to the person ensures availability without imposed burdens such as remembering passwords, or carrying physical tokens, such as keys. However, new authentication factors come with new potential attack vectors that need consideration for security to be guaranteed. One attack vector could be spoofing, which is presenting falsified biometric data that mimics that of an authorized user. For example, if visual facial features are used as an authentication factor, then one could spoof facial features by presenting a picture of the authorized person, or by wearing a mask that looks like them. Thus, for biometric authentication factors to be secure, robust methods for dealing with these attacks need to be developed.
It is hard to overestimate how important and efficient Convolutional Neural Networks (CNNs) have become for classification tasks in image analysis over the last decade, so given their performance on similar tasks, using a CNN-based approach for spoof-detection is a natural choice.

1.2 Previous work

Different approaches for detecting spoof attacks on biometric authentication systems, including facial recognition, have been presented in a number of previous studies [8–14, 17–21]. Some of these have investigated spoof detection based in part on imagery in the NIR (Near Infrared) spectrum. In [10], multi-spectral solutions to anti-spoofing are investigated, using both NIR-imagery and visible-light imagery. They tried different approaches, using Convolutional Neural Networks (CNNs) as well as more classical, Local Binary Pattern-based (LBP) methods, on visual and NIR-data either separately or in conjunction. The best overall results were found when using a CNN on visual and NIR-data in conjunction, while the worst results were found using LBP-methods on visual and NIR-data in conjunction. In [17], an NIR-based solution was investigated. Spoof detection was based on differential images between active and non-active lighting (a differential image is one where pixel values from the non-actively lit image were subtracted from the actively lit one). Classical statistical methods were used, rather than machine learning. More

specifically, images were analyzed using consistency measures, which easily pick up on anomalies in the background if a 2D-printed image is held up in front of the camera.
However, approaches that do not use NIR-based facial recognition, but rather visible-based, are the most common. [13] proposes a spoof-detection approach based on analyzing blinking behavior of the subject. In [21], focus is not only on the model used to detect spoofing, but also on the means of data collection. To produce a model that is robust to variations found in real-world applications, they downloaded a large number of videos with live faces from the web. They then extracted relevant portions with face-detection software. Spoof videos were produced in two ways. Some conventional recordings were made by replaying live videos on smartphone, tablet, laptop and desktop screens held in front of the recording device. Other recordings were made by applying distortions to live footage in order to mimic the distortions typically produced in spoof attacks. The latter of the two was done to more easily scale the spoof data-base, since making actual spoof recordings is slow and costly by comparison. Finally, they proposed a three-module model with a module for detecting temporal anomalies, another for finding which regions are relevant and a third for detecting spatial anomalies in the regions detected as relevant. In [9], focus is on zero-shot face anti-spoofing, meaning that the model proposed is meant to handle attacks with previously unseen types of spoof media. 13 different spoof media were used, and the model was trained to partition spoof-footage into subcategories without explicit supervision, and then make the binary classification based on which subcategory it corresponded to the most. In [8], the aim was to improve the performance of a recurrent CNN-based classifier by supervising the learning of two auxiliary signals in addition to classification as spoof or live: the scene depth map and the remote photoplethysmography-signal of the subject. This is meant to make learned information more generalizable, and the classification mechanism more transparent.
A special family of spoofing is where fake face imagery is generated using deep learning, so called deep fakes, and in particular, spoofs produced by Generative Adversarial Networks (GANs) [18]. Several approaches to detect this category of spoof attacks have been developed. In [20], a method was proposed where deep fakes superimposed over real footage could be exposed by finding errors in head pose reconstruction of the spoofed face. In [11], GAN-generated faces were detected by training a neural classifier on co-occurence matrices extracted from the imagery. However, the need for better detection of faces synthesized using GANs was highlighted in [12]. They showed that many of the state-of-the art classifiers for detecting faces that were generated with GANs, are susceptible to relatively simple removal of the sort of fingerprints that would normally aid their classification.
Anti spoofing in a multi-factor authentication system, where one of the factors was visible spectrum-based facial recognition, is investigated in [14]. This paper doesn't just look at spoof-detection primitives in isolation, but the overall authentication system they fit into, proposing a two-tier authentication system with three factors, where active spoof detection is only done for factors where the identification stage has already passed. The spoof detection primitive used for the face specifically, is a CNN-based analysis of a feature vector-representation of the face. In [19], another

multi-factor authentication system is investigated. It proposes an authentication system that fuses the factors using parallel support vector machines (SVMs). Different SVMs were trained to cover different combinations of available factors, to make the system more robust to situations where not all biometric factors are available.

1.3 Aim of the project

The aim of this project is to implement a spoof detection solution for distinguishing real faces from fake ones (such as photographs printed on paper, or latex masks). The solution is to be suited for deployment in an automotive setting, as an additional component of what is primarily a driver monitoring system using an NIR-camera. One of the constraints on the project is that any software used needs to be lightweight enough to be feasible to run on a small-scale platform. The main software component of the solution is a neural classifier, which to this end will be restricted to model sizes using fewer than 100 000 trainable parameters. In addition to the software component of the project, there is also a hardware component, in that active NIR-light sources can be used to aid spoof detection, under the constraint that the camera and NIR source setup may not be wider than 20 cm.
The setup used is an NIR-camera with three active NIR sources. The NIR-sources are alternated between using a programmed exponator, giving an image stream with a period of four frames with each light condition (including one lacking active NIR-light). Pairs of consecutive frames are concatenated depthwise, scaled, rotated and cropped to only include relevant information, and finally fed to a neural network for binary classification as either spoof or live.

2
Theory

2.1 Artificial Neural Networks

Artificial Neural Networks are highly adaptable computing systems, inspired by biological neural networks such as animal brains, which are becoming increasingly important for a multitude of tasks. Their adaptability lets them find solutions to highly non-linear multivariable problems that are hard to solve using explicitly designed rules. Most notably, they are used very successfully for classification problems in image analysis.

2.1.1 Neurons and activation functions

The computational element of a neural network is the neuron. The neuron receives a number of signals, x_i, typically of numerical value, and evaluates a function of these inputs to produce an output signal. Generally, this is done by first calculating a weighted and biased sum,

$$z = -b + \sum_i w_i x_i, \tag{2.1}$$

where b is the bias of the neuron, and w_i are the signal weights. After z is computed, the output signal of the neuron is found by evaluating a single-parameter function, σ at z,

$$x_{\text{out}} = \sigma(z), \tag{2.2}$$

where σ is called the activation function of the neuron. The most basic activation function is the linear activation function, which is simply

$$\sigma(z) = z = -b + \sum_i w_i x_i. \tag{2.3}$$

It is called the linear activation function, since it is a linear function of the incoming signals x_i. Another basic activation function, which is generally only used as a conceptual example, is the step function,

$$\sigma_{\text{step}}(z) = \begin{cases} 0, \text{ for } z < 0 \\ 1, \text{ for } z \geq 0 \end{cases}, \tag{2.4}$$

or equivalently,

$$\sigma_{\text{step}}(z) = \begin{cases} 0, \text{ for } \sum_i w_i x_i. < b \\ 1, \text{ for } \sum_i w_i x_i. \geq b \end{cases}, \tag{2.5}$$

which gives a neuron that sends an output signal if the weighted signals sum to a value greater than the bias, and not otherwise. Intuitively, such neurons could be viewed as performing a logical evaluation of a complex statement which is either true or false, conditioned on which input signals are active. However, conventional methods of training neural networks, such as stochastic gradient descent, only work with functions that have a defined gradient. Thus, since the gradient of σ_{step} is always either 0 or undefined, it is generally not suited for most applications. A continuous correspondent to the step function function that is commonly used in practice is the sigmoid function, given as

$$\sigma_{\text{sigmoid}}(z) = \frac{1}{1 + e^{-z}}. \tag{2.6}$$

σ_{sigmoid} is continuous, with continuous gradient. It rapidly approaches 1 as the weighted sum of signals gets larger than the bias, and rapidly approaches 0 as the weighted sum gets smaller than the bias, making it similar to the binary on-off-characteristics of σ_{step}. Since its gradient is always defined, it can be tuned using conventional methods. However, as z gets far away from 0, the gradient will rapidly vanish, making the network susceptible to getting stuck during training. To cope with this problem, one of the most common activation functions used is the rectifier function, which is given as

$$\sigma_{\text{rectifier}}(z) = \max(0, z) = \begin{cases} 0, \text{ for } z < 0 \\ z, \text{ for } \geq 0 \end{cases}. \tag{2.7}$$

Units using $\sigma_{\text{rectifier}}$ are called ReLU-units, and often the activation is simply refered to as the ReLU-function. $\sigma_{\text{rectifier}}$ is less likely to have a vanishing gradient, and more computationally efficient, than σ_{sigmoid}. These advantages have also been observed empirically [6].

2.1.2 Network Architecture

A neural network is a system of neurons that receive and transmit signals between each other, resulting in complex computations in aggregate. Network architecture concerns the macro-scale design of such systems; how signals flow through the network, which weights are dependent on each other, etc. Networks that have neural connections forming cycles are referred to as recurrent networks, whereas networks whose connections have a strict chronology are referred to as feed forward networks. Typically, neural networks order neurons into consecutive layers, where signals are passed between layers but not within them. In the final layer, the network output (for example a vector of class probabilities for a classification network) is found.
The following sections will introduce some important concepts for neural network architecture that have been used in this project. These include the convolution, which is an operation that can be used to identify particular features in a tensor. Depthwise separable convolution, which is a more computationally efficient version of the convolution, will also be introduced as an alternative. Two other concepts that will be introduced are bottlenecks and expansions, which can be used to increase or decrease tensor depth. We will also talk about skip connections, which allow signals

in a neural network to skip several layers. Finally, the role of these modules will be explained in the context of MobileNetV2, which is the neural network architecture used in this thesis.

2.1.2.1 Convolution

The discrete 2D-convolution $g(x, y)$ between a kernel ω and a function $f(x, y)$ is defined by the following operation:

$$g(x, y) = \omega * f(x, y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx, dy) f(x + dx, y + dy), \tag{2.8}$$

where a and b are constants that define the width and height of the kernel as $(2a+1)$ and $(2b + 1)$, respectively. Such operations are used in image processing for many purposes. The intuition is that the kernel is a matrix that corresponds to some feature. By sliding the kernel over regions of the input function, taking the dot product of the kernel with these regions, we produce an output matrix which highlights how present this feature is in different parts of the picture. This has several uses. For example, by choosing kernel values that resemble a Gaussian in 2D-space, one could perform a blurring operation. Other kernel choices may cause edges, corners, intersections, etc, to be highlighted.
So far, we have treated 2D-images as tensors with two spatial dimensions representing the width and height. However, in image processing, it is common to define a third dimension of the tensor, which represents channels rather than spatial dimensions. For example, an RGB-image will have three channels, representing the intensity of the colors red, green and blue in each pixel. Since the channel dimension does not represent a spatial dimension, such images are still referred to as a 2D-images, and the kernel will generally be implemented with the same depth as the image, so that it only slides in the two spatial dimensions. The output from applying a single such kernel will only have a depth of one. The height and width of the output image will depend, among other things, on the size of the kernel, how big steps it uses to slide over the image, and whether padding is used at the image boundary.

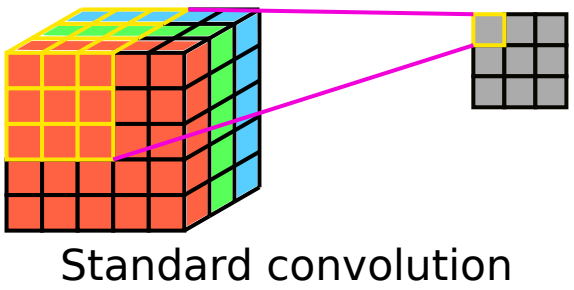

Figure 2.1: A 3×3 kernel applied without padding to a 5×5 image with depth 3, producing a 3×3 output image of depth 1.

A graphical representation of a simple standard convolution is illustrated in figure 2.1, where the input image is 5×5, with a depth of three, while the output is 3×3

with a depth of one after applying a 3×3 kernel. By applying several kernels of the same size and stacking their outputs, one can produce an output with an arbitrary number of channels.

2.1.2.2 Depthwise Separable Convolution

When increasing the number of output channels, which is done by increasing the number of kernels convolved with an input image, the number of required multiplications increases rapidly. If an input image with M channels is convolved with N kernels of size $K \times K$, producing N output channels of size $L \times L$, this requires $K^2 \times M \times N \times L^2$ multiplications. Depthwise separable convolution is an operation inspired by the standard convolution described above, that reduces the scalability factor between the desired number of output channels and the number of multiplications [2, 5, 15, 22]. Depthwise separable convolution is done in two steps. In the first step, a depthwise convolution is applied, wherein a kernel of depth 1 is applied to each channel of the input, producing an intermediary output with the same depth as the input. In the second step, N different 1×1 kernels (with the same depth as the input) are applied to combine the intermediary outputs from the different channels, producing an $L \times L$ image with N channels.

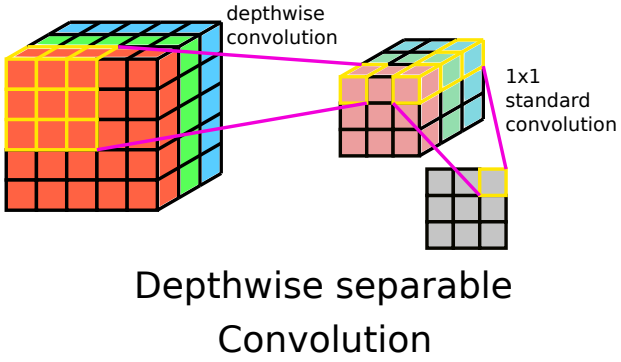

Figure 2.2: Illustration of depthwise separable counterpart to figure 2.1: First, depthwise separable convolution is done, where 3×3 kernels of depth 1 are applied to each channel to produce an intermediary output of depth 3. Second, a 1×1 kernel of depth 3 is applied to the intermediary output, yielding a final, single channel output.

A graphical representation of this operation is presented in figure 2.2. We see that the number of multiplications needed is $K^2 \times M \times L^2 + M \times L^2 \times N = M \times L^2(K^2+N)$. Dividing this with the number of multiplications needed using standard convolution, we get the ratio $\frac{1}{N} + \frac{1}{K^2}$, which decreases with N, telling us that the depthwise separable convolution scales the number of multiplications slower than the standard convolution when increasing the number of channels in the output. Since multiplication is an expensive operation, this means that the depthwise separable convolution can be an appropriate alternative to standard convolution for use on systems with limited hardware, such as mobile and embedded systems.

2.1.2.3 Convolutional Neural Networks

Two of the most common types of layers in neural networks are fully connected layers and convolutional layers. In a fully connected layer, each neuron receives the signal from every neuron in the previous layer. A convolutional layer, on the other hand, is a bit more selective about how signals are received. Neural signals from the previous layer are ordered in a tensor. This tensor is typically a 1-or 2-dimensional array, which is often extended with additional channels; an RGB-picture would, for example, generally be conceptualized as a 2D-array with three channels - one for each base color. Signals are then passed on to channels of the convolutional layer using a number of convolutional kernels before adding biases and applying the neural activation function. In effect, the values of the kernels used for convolution correspond to neural weights that are shared between neurons in the convolutional layer. This can give a number of potential advantages; for example, since each signal is passed to fewer neurons, this decreases computational costs, and since weights are essentially shared between several neural connections (since the kernel doesn't change as it slides over the input), training of weights can be done more efficiently. It's also intuitively sound that convolutions, which are very useful for feature identification in image processing, should be useful for similar functions in neural networks. And of course, in addition to the theoretical motivations, convolutional neural networks (CNNs), that is neural networks that incorparate convolutional layers, have been successfully implemented for a plethora of applications in practice [1, 3, 6, 7, 16].

2.1.2.4 Bottlenecks and expansions

In convolutional neural networks, some layers will use many kernels, resulting in very deep signal tensors. These can be expensive to store in memory and to operate on, and it's generally assumed that only a manifold in the space spanned by the channels contains relevant information [15].

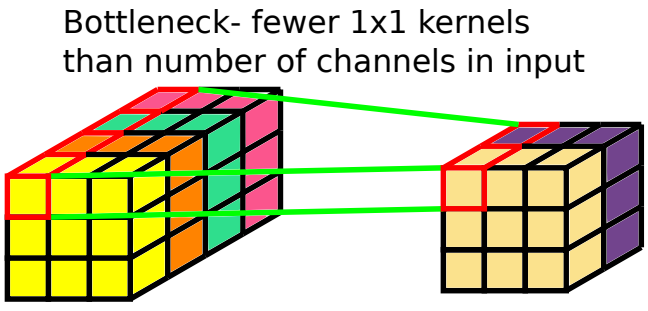

Figure 2.3: Graphical depiction of a bottleneck.

As seen in figure 2.3, this lets us reduce the number of channels, with little to no loss of relevant information, by applying a layer with a number of 1×1 convolutional kernels that is smaller than the number of channels in the layer input [4, 5, 15, 22].

Such layers are often called bottleneck layers. After decreasing tensor depth before applying a convolution, it may be desirable to expand the depth back again.

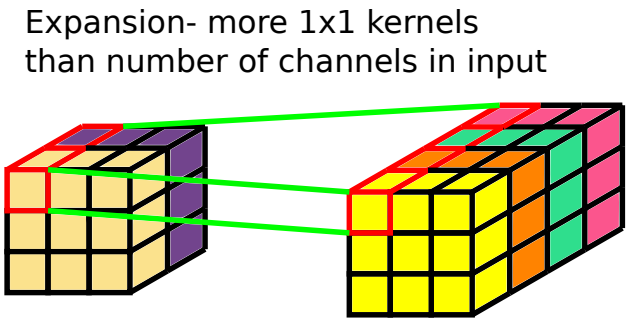

Figure 2.4: Graphical depiction of an expansion.

Expansion layers do just this by applying more 1×1 kernels than the number of channels in the input tensor, as seen in figure 2.4. In short; bottleneck layers and expansion layers respectively serve to decrease and increase signal tensor depth.

2.1.2.5 Residuals

As neural networks become deeper, they have a tendency to become more difficult to train, due to exploding or vanishing gradients. A way to mitigate this problem is to insert skip connections in the neural network, where neural signals from an earlier layer are added to the signals of a later layer before the activation function is applied, allowing information to travel unadulterated further down the network. Blocks that incorporate skip connections are often called residual blocks [4].

2.1.2.6 MobileNetV2

MobileNetV2 is a neural network architecture intended for mobile and embedded platforms [15]. What mainly distinguishes it from preceding architectures is the use of a bottleneck residual block. This is a module that incorporates many of the submodules discussed above.

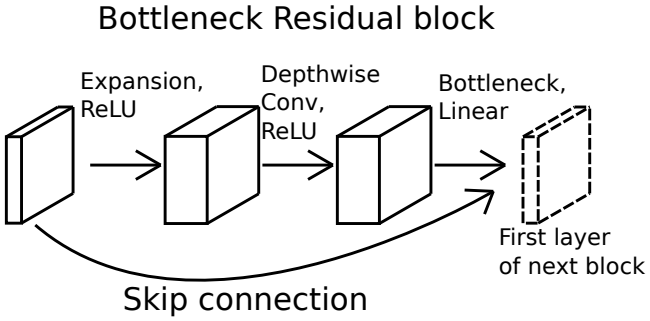

Figure 2.5: Graphical depiction of the bottleneck residual block.

As seen in figure 2.5, it first uses an expansion layer. The expansion layer is followed by a depthwise convolution. Finally a linear bottleneck layer is used (that is, a bottleneck layer using linear activation function). The depthwise convolution and the linear bottleneck collectively constitute a special case of a depthwise separable convolution. The module also has skip connections going from before expansion to after the bottleneck; thus the skip connections in a chain of bottleneck residual blocks go from bottleneck to bottleneck. The reason for this is that the tensor that is skip connected must be stored in memory through the rest of the block, and by reducing the size of the tensor, its load on memory is also significantly reduced [15]. The overall architecture uses a standard convolution followed by 17 bottleneck residual blocks, a big expansion and average pooling before a final bottleneck leading to the output neurons.

3

Method

3.1 Hardware setup

The hardware used in the project is set up to simulate conditions at a driver's seat, with an NIR-camera approximately at a position that would correspond to the steering wheel, or instrument panel. The subject sits in a chair in front of a desk where the camera is fixated, viewing the subject about 35° from below.

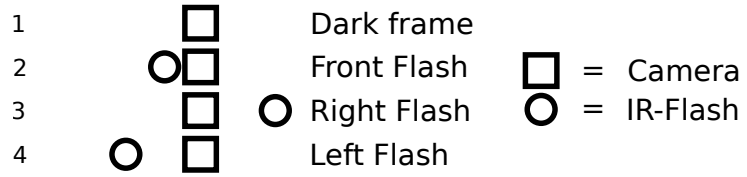

Figure 3.1: Flash configuration for 4 consecutive frames. The viewing direction of the camera is towards the reader.

Footage is collected using a 120 Hz NIR, camera which uses an exponator programmed to produce a stream of consecutive image frames with flash activation alternating periodically as in figure 3.1. That is, each period consists of a dark frame (without active light), a frame taken with a front flash, a frame taken with a flash 10 cm to the right of the camera, and a frame taken with a flash 10 cm to the left.

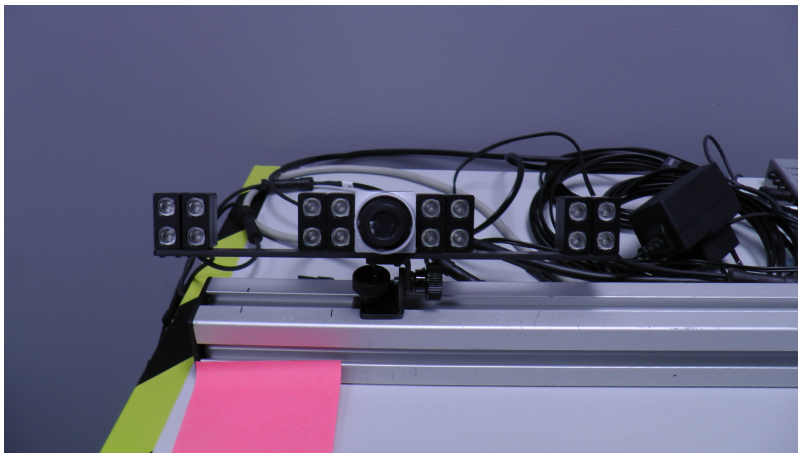

Figure 3.2: Photograph of the camera setup that was used.

A photograph of the camera setup can be seen in figure 3.2. The camera used is a Basler acA1920-155um. In addition, a halogen lamp is part of the experimental setup to simulate an ambient NIR-light source, such as the sun, however, it is important to note that this is not a part of the anti-spoofing system, but rather used to produce training data that will allow the system to compensate for external light conditions. The halogen lamp is placed about 1 meter straight to the left of the subject, providing asymmetric illumination of the subject when active. The setup is located in a small room with no other ambient NIR-light sources. The room had a curtained window facing an indoor corridor, which let through some visible light, but negligibly in the NIR-spectrum.

3.2 Data collection

Recordings are divided into two main categories; live recordings, depicting real faces of living subjects, and spoof recordings, depicting fake faces, e.g. photographs printed on paper, or subjects wearing a mask.

3.2.1 Live test subjects

A total of 37 live recordings were made with 11 different test subjects. All subjects had signed an agreement to the use of recordings in the project, and storage of the data in compliance with relevant legislation such as the General Data Protection Regulation (GDPR). All data is stored with anonymized indexation and metadata. In each recording, the subject was first instructed to look at a marked point on the wall in front of them, then another one, about 45 ° to the right, followed by a point about 45° to the left, and finally straight into the camera. This way, footage of several head poses would be collected. Such a recording would be about 32 seconds in total - 8 seconds for each pose. Each recording session consisted of three such recordings; first, a recording would be made in office lighting, then another one in near-darkness, and finally one with a halogen lamp illuminating the subject from the side. The dark recording and the one in office lighting would be similar in terms

Figure 3.3: An example of a frame from a live recording.

of illumination perceived by the NIR-camera, but due to the expansion of the pupil in dark conditions, clues such as corneal reflection differ in clarity between the two. The recording using illumination from a halogen lamp would significantly alter the ambient NIR-lighting, contributing valuable information about how to account for such lighting, which is usually present in real-world application. In figure 3.3, a frame from a live recording can be seen. There were a couple of inconsistencies during data gathering accounting for the anomaly in the number of recordings compared to the number of subjects; with two of the subjects, a recording session of three recordings each were made with glasses on, and another without. With one subject early in the project, two recordings were unfortunately made incorrectly and not usable.

3.2.2 Spoof media

Two kinds of spoof media were used in the project: faces printed on paper, and masks.

3.2.2.1 Prints

For faces printed on paper, there were three kinds of prints: photographs that were taken with a high quality visual spectrum camera, printed in color, photographs that were taken with the same camera, but printed in black and white, and finally, NIR-frames from the live recordings which were extracted and printed in black and white. All prints were made in landscape orientation on A3-papers. Prints were fixated to a board of rigid, flat shape, so that the prints would not bend, held up in front of the NIR- camera in a position corresponding to the facial position of subjects in live recordings, and recorded for 10 seconds each. The orientation of the print would be slightly manipulated during the recording to give some slight variety

in angles in the footage. Each print was used in one recording with office lighting. No recording was made in darkness, since there is no chance of corneal reflection in a printed pupil regardless. The motivation for shorter recordings than for live subjects is that only one pose is depicted in each print, meaning that additional recording time would contribute less information.

Figure 3.4: An example frame from a spoof recording in which a NIR-frame from a live recording was printed on paper and held up in front of the camera.

In figure 3.4, a frame from a print attack can be seen. In total, 30 NIR-frames were printed, 25 visual spectrum-photographs were printed in black and white, and 8 visual spectrum-photographs were printed in color. The reason for the slim amount of color-prints was that such photographs looked very obviously different from real faces in the the attack footage they produced, and thus would contribute less important information. All in all, 63 prints were used in one attack recording each. Note that the higher number of recordings compared to live faces does not correspond to a larger amount of data, as these recordings were shorter.

3.2.2.2 Masks

Three masks were used for spoof attacks in the project. A total of 12 mask attacks were recorded. In the attacks, they were worn by a subject who would be recorded according to the same protocol as in the live recordings; each recording was made with the same four poses, and each mask was recorded under three different lighting conditions. Unlike with printed faces, the eyes visible under this attack are real eyes, which means pupil dilation and corneal reflection could be observed, so all three lighting conditions were relevant for these attacks.

Figure 3.5: An example frame from a spoof recording in which the subject wore a mask.

In figure 3.5, a frame from a mask attack can be seen. The reason for the number of attacks being 12 rather than 9, is that one of the masks was used for an additional recording session, in which a hood was worn over the head, to further expand the database and introduce some diversity.

3.3 Main Algorithm

The spoof-detection algorithm can be divided into three main steps. The first step is pairing of consecutive frames which differ in active light conditions. The second step is frame preparation, in which consecutive frames are rotated, scaled, translated and cropped to only include the region of interest in a frame of specified size, and finally appended together depthwise to create a two-channel image frame. In the third step, classification of the face is done using a neural network.

3.3.1 Pairing of Consecutive Frames

The algorithm always operates on two consecutive frames which, as described previously, have differing active light sources, since it is plausible that the response to differing light conditions could be correlated to reveal more information about geometry and object texture than a solitary frame would. Following the same naming convention as in figure 3.1, we see that we can pair up consecutive frames in each four-frame period as in table 3.1.

Frame pair	Frame A	Frame B
P_1	Dark frame	Front Flash
P_2	Front Flash	Right Flash
P_3	Right Flash	Left Flash

Table 3.1: Pairing of frames with different active lighting.

The three frame pair arrangements, P_i, are primarily investigated as separate solutions with differing priorities, rather than intended to be used in conjunction with each other in the same solution.
P_1 represents the most compact setup, where differential lighting is achieved using a frame with only ambient NIR-light, and a frame lit using a NIR-flash very close to the camera. This setup will give the least information about scene geometry and texture, especially when ambient NIR-light level is low, but it was still considered, due to the advantage of it being so compact, which is important for the intended deployment of the system. However, it turned out empirically, that the information contributed by the dark frame, at least under the investigated conditions, is largely negligible.
P_2 represents the second most compact setup, using a frame lit by a flash close to the camera, and a frame lit by a flash 10 cm away. This would be less sensitive to ambient NIR-light levels than P_1.
P_3 represents the least compact setup, using two flashes, 10 cm in each direction from the camera. This means the setup occupies more space, but it also has the greatest potential for giving information about scene geometry, due to it having the largest angle difference between NIR-rays originating from the two flashes.
In addition to information about scene geometry, it is interesting to note that the front flash, which is used in both P_1 and P_2, potentially causes a distinct corneal reflection to be caught in footage, due to the very small angle it has to the viewing direction of the camera; a phenomenon known as *bright pupil*. If bright pupil is induced in neither or both frames, this could be an indication of the face being spoofed. It should be noted that the eye visible in a mask attack is a genuine eye, thus, while lack of corneal reflection could be a useful clue for rejection of print attacks, it's not necessarily a useful clue for for acceptance of genuine faces. Since this information would not be present in P_3, this is a factor that might benefit the approach using P_1 or P_2 despite the more economic setup. However, it should be noted that induction of bright pupil is hard to guarantee for a genuine pupil, especially in bright conditions, which cause the pupil to contract, so the importance of this factor should not be overestimated.
An additional experiment was run, in which the same frame was appended to itself. This would not give the neural network information from more than one light condition. This was done to control to what extent the classifier actually benefitted from the additional light condition.

3.3.2 Frame preparation

The frame preparation procedure serves to present relevant information from the NIR-camera in a way that suits classification by a neural network. As we have seen,

we take an approach operating on two consecutive frames with differing light conditions. Frame preparation is done as in figure 3.6. In short, we accomplish the task

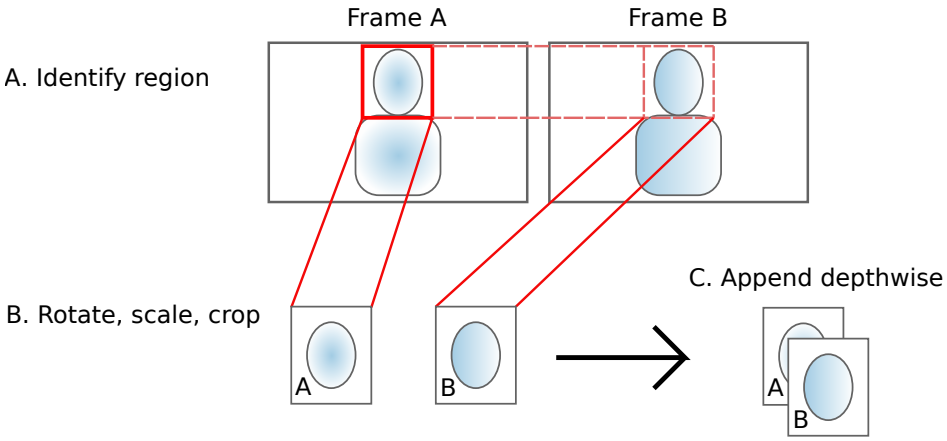

Figure 3.6: A brief description of the frame preparation procedure that constitutes the first step of the algorithm.

by identifying a region of interest in one frame, extracting this region from both frames and appending them together.
Since the number of mathematical operations and neural network parameters grows quickly with the size of neural network input, it is preferable to only pass on relevant information. This is done by rotating, scaling and cropping the image to include the most important region of the image. Two different regions of interest were investigated separately in this project: the full face, scaled to 200×250 pixels, and an eye region, scaled to 48×48 pixels. The motivation for investigating the face-approach was that the face should intuitively be the most interesting region of a frame for determining whether the face is real. The idea for investigating the eye-approach is that it would be computationally cheaper to use a smaller region. The latter approach was also investigated due to the speculation that relevant discrepancies between fake faces and real faces should be relatively big in that region. Masks have openings at the eyes where a sharp contrast might be seen against the real face right at the opening, and printed eyes don't react to active NIR-Light in the same way as real eyes.
A mapping from two points in an input image to two output points is sufficient to determine a transformation that rotates, scales and translates a 2D image without warping it. This is because such a transformation is determined by four parameters: one rotation angle, one scale factor and one translation distance for each of the two dimensions. Each of the two point mappings contributes one relationship per each of its two coordinates, giving four relationships, meaning the problem is determined. To extract the face region, the image coordinates of the outer corners of the eyelids are identified using existing functionality in SmartEye's proprietary software. They are then always mapped to the pixels fixed at $x_1 = 50, y_1 = 100$ and $x_2 = 150, y_2 = 100$, respectively (the positive x-direction being rightwards and the positive y-direction being downwards).

Figure 3.7: To the left is shown a raw frame. To the right is the same frame after being scaled, rotated, translated and cropped to include only the face.

This empirically tends to give the full face, with some margin to the frame boundary, as seen in figure 3.7. To extract the eye region, the coordinates of the outer corner and inner corner of the eye region are identified and mapped to $x'_1 = 7, y'_1 = 24$ and $x'_2 = 41, y'_2 = 24$, respectively. This can be done for either eye (note that the right eye then is mirrored), and both were used in the project.

Figure 3.8: To the left is shown a raw frame. To the right is the same frame after being scaled, rotated, translated and cropped to include only an eye.

Just as for the face, we see in figure 3.8 that this tends to give the full eye region with some margin. As illustrated in figure 3.1, only one frame is used for the identification of relevant points, and treated as if they are located in the same location in the other frame, since little time passes in between. In fact, for each period of four frames, the frame using front flash is used for identification of the relevant region, and reused in the remaining frames.
This way, the relevant region has been identified and extracted, corresponding to steps A and B in figure 3.6. Finally, they are appended depthwise, giving a final tensor representing the region of interest with two channels, corresponding to lighting from different active NIR-sources.

3.3.3 Classification

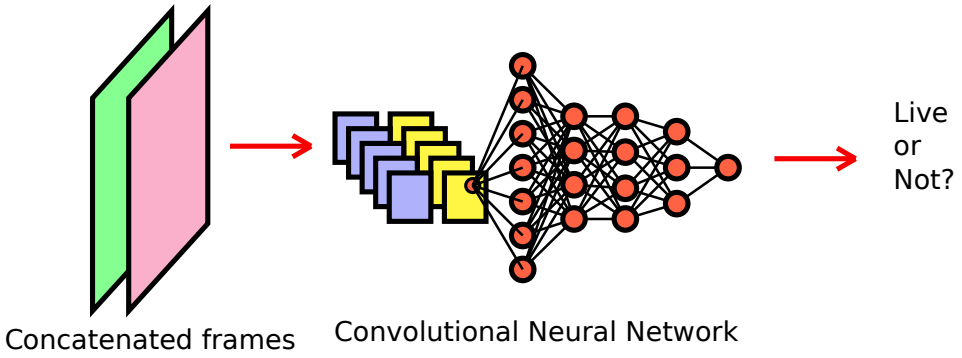

Figure 3.9: Classification is done using a neural network of MobileNetV2 architecture [15], taking concatenated frames as input

As depicted in figure 3.9, once two frames have been prepared and appended together depthwise into a two-channel image, this image is sent as input to a binary neural classification network, which has one output with sigmoid activation to give values from 0 (input classified as spoof) to 1 (input classified as real). The network architecture used is MobileNetV2, with width multiplier $\alpha = 0.1$, as defined in [15]. It was implemented using Keras. Detailed descriptions of the resulting architectures can be found in the Appendix, section A.1 for the face region classifier, and section A.2 for the eye classifier. The neural network had a total of 87,289 trainable parameters, regardless of whether the full face is used for classification, or whether only the eye region is used. However, when the full face is used, it requires 50,640,990 floating point operations to process the network input, whereas if the eye region is used, it only requires 2,635,686 floating point operations. Thus, the computational cost required to process a full face is about 20 times greater than that required to process an eye region.

3.4 Neural Network Training

To train the neural classifier, frames were extracted from the recordings made in data collection, and prepared according to the method described above. Frames extracted from spoof recordings were labeled as 0, while frames extracted from live recordings were labeled as 1. The frames were then divided so that all recordings of 8 randomly chosen subjects were used as training data, and the rest as validation data. Frames from mask recordings were divided so that recordings with two of the masks were used for training, and the other mask was used for validation. Both training and validation data was normalized samplewise based on the statistics from the training data set. Due to the relatively small training data set, variation was introduced by augmentation of the training data. Specifically, during training of the network, input frames would be rotated within 30°, shifted within 10 % height -and widthwise, and zoomed between 0 and 15%. Augmentation was not done during

validation, as augmentation doesn't serve a diagnostic purpose so much as a way of getting a more robust model fit from a smaller data set. The optimizer used was Adam. Binary cross-entropy was used as the cost function. The Keras callback for reducing the learning rate as accuracy improvement halts was also used.

4
Results

Following the same notation for the different frames and frame pairs as introduced in figure 3.1 and table 3.1, we see in table 4.1 how accurately the neural network performed on the different recording classes. The results are for recordings of previously unseen faces. We also see the results for networks trained for all the different types of input. Note, however, that while we make a distinction in this table between masks and prints to gain a deeper understanding of how the network performs, the neural network is just supposed to classify both as not-live.

Frame/frame pair	Input region	Live acc.	Print acc.	Mask acc.	Overall acc.
P_1	Full face	99.9%	98.3%	88.7%	97.6%
P_2	Full face	100.0%	99.7%	94.1%	98.9%
P_3	Full face	100.0%	98.5%	88.0%	97.6%
Front flash	Full face	100.0%	95.6%	94.4%	97.6%
Right flash	Full face	100.0%	91.6%	99.8%	97.1%
Left flash	Full face	99.9%	99.3%	93.4%	98.7%
P_1	Eye region	96.8%	94.4%	56.4%	89.7%
P_2	Eye region	99.8%	98.3%	40.9%	90.2%
P_3	Eye region	99.1%	94.5%	68.5%	93.5%

Table 4.1: The final accuracies of the neural classifier on previously unseen faces, depending on which type of input it was trained on.

Overall, the neural classifier shows clear signs of learning how to distinguish recordings of fake faces from those of live faces, provided that it is given the full face region as input. When only given the eye region, it could still classify better than random chance, but considerably worse than when given the full face region, especially for masks.

There is not an extreme difference in overall performance between networks trained on frame-pairs, compared to single frames, but there is one interesting difference: the networks trained on frame-pairs can classify print attacks nearly perfectly, while having slightly more difficulty with masks, whereas those trained on single frames perform slightly worse on print-attacks and better on mask attacks. In order to draw any certain conclusions, one would need a bigger data set, but this might have something to do with the fact that geometric anomalies, as opposed to textural ones, could be more easily identified when working with frame-pairs. Clearly, the geometric anomalies are greater with a printed face (which is entirely flat).

The neural network that unequivocally performed the best, and (as is seen in figure 4.2) had the most stable training history, was the one trained on P_2, that is

front flash+right flash. While intuition would have P_3 give the most information about scene depth and geometry (due to the larger angle between the flashes), it performed worse than P_2. Perhaps the qualitative difference in textural response between a flash to the front and one a bit to the side, is more useful for classification than just geometric information from shades. Also, corneal reflections could give some part of the explanation, since they are only induced in frames using front flash. One should however be careful when drawing conclusions, seeing as the dataset is relatively small. These hypotheses should be investigated in a larger dataset, with more live subjects, more print attacks and more masks used.
The attentive reader will notice that the dark frame is not present in the table. This is because it turned out that the classifier would learn absolutely nothing from training on the dark frame alone, indicating that the information contained in it is negligible. Thus, one can conclude that active NIR-lighting is essential, since ambient lighting cannot be relied on to get enough information to classify faces as real or fake.

4.1 Training history

In this section, we will see how training progressed depending on what information was provided to the neural classifier.

4.1.1 Full Face

When trained on full face, the neural network achieved much higher accuracy than when trained on just the eye region. Peak accuracy was also reached much faster, and more stable. This was the single most important factor in training; much more important than which, or how many, light conditions were provided.

4.1.1.1 Using Frame Pairs

The network trained on frame pairs tend to perform better on print attacks compared to mask attacks. Among all networks in the study, the one which gave the best performance, is the one trained on front flash and right flash (P_2), as seen in figure 4.2. It landed on an accuracy of 98.9 % overall. It also shows the most stable training history, as it had converged after about 40 epochs, . In figures 4.1 and 4.3, we see that the other frame pairs, P_1 and P_3, gave slightly worse performance; both reached an accuracy of about 97.6 %, and they had converged after about 65 and 60 epochs respectively.

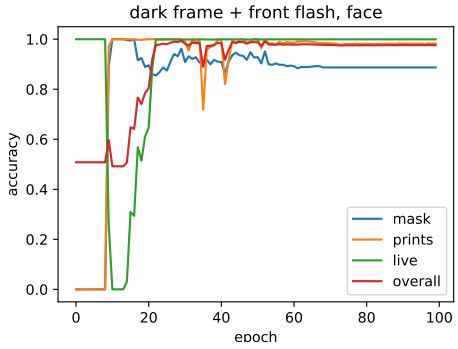

Figure 4.1: Accuracy of correct classification on validation data during training history, on the pair of frames using dark frame and right flash (P_1).

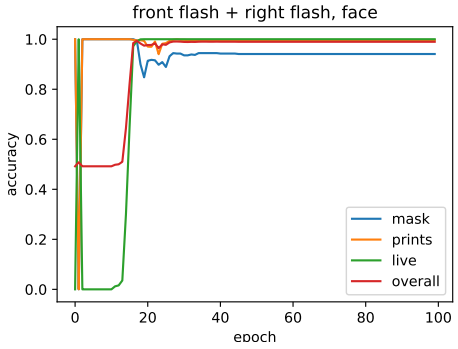

Figure 4.2: Accuracy of correct classification on validation data during training history, on the pair of frames using front flash and right flash (P_2).

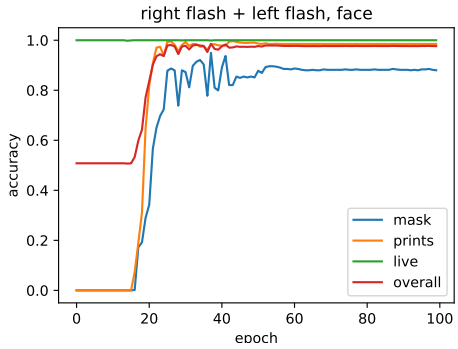

Figure 4.3: Accuracy of correct classification on validation data during training history, on the pair of frames using right flash and left flash (P_3). (Note that the neural network classification is binary - real face or fake; the three categories are for the aid of the reader)

4.1.1.2 Using Single Frames

Overall, the networks trained on single flashes perform on a similar level of accuracy to those trained on two flashes, although, as seen in figures 4.4, 4.5 and 4.6, none of them perform as well or with the same stability as the network trained on front flash and right flash (P_2). An interesting fact is that these networks don't show the same tendency to categorize print attacks more accurately than mask attacks. In particular, as seen in figure 4.5, the network trained on just right flash, ended up much more accurately detecting mask attacks than print attacks. It is hard to say why this was the case, and to draw any certain conclusions, one would need to use a bigger data set.

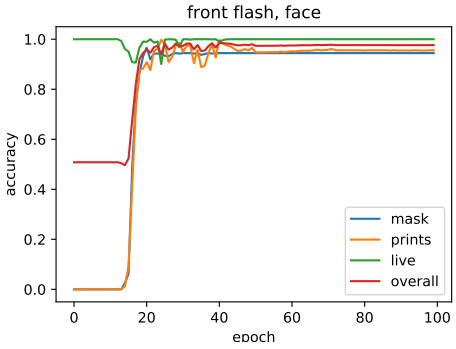

Figure 4.4: Accuracy of correct classification on validation data during training history, on the pair of frames using only front flash.

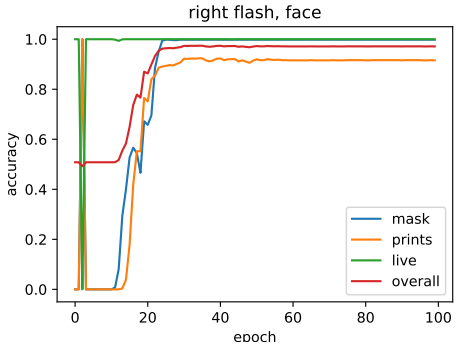

Figure 4.5: Accuracy of correct classification on validation data during training history, on the pair of frames using only right flash.

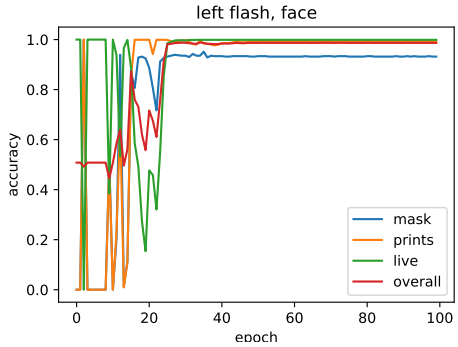

Figure 4.6: Accuracy of correct classification on validation data during training history, on the pair of frames using only left flash. (Note that the neural network classification is binary - real face or fake; the three categories are for the aid of the reader)

4.1.2 Eye Region

While the networks did get some accuracy classifying prints from just the eye region (94.4-98.3 % depending on the frame pair), this was not nearly as accurate as for the full face region, as can be seen in figures 4.7, 4.8 and 4.9. Its accuracy on masks was very low. Accuracy fluctuates with greater amplitude throughout training history, although the networks seem to converge after 40-65 epochs, which is comparable to the networks trained on the face region. The hypothesis that this region would give important clues about masks, due to having an opening with a border against real face in the eye region, thus seems to be incorrect as far as these results indicate. This means that while this approach would be much more computationally cheap, it doesn't seem to be a viable option for spoof detection.

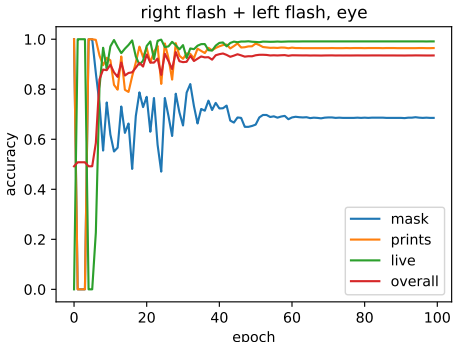

Figure 4.7: Accuracy of correct classification on validation data during training history, on the pair of frames using dark frame and right flash (P_1).

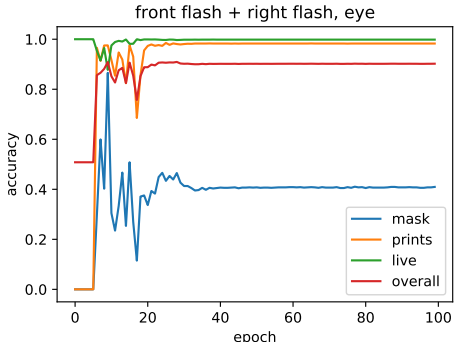

Figure 4.8: Accuracy of correct classification on validation data during training history, on the pair of frames using front flash and right flash (P_2).

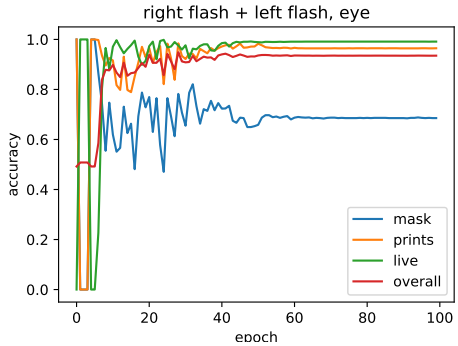

Figure 4.9: Accuracy of correct classification on validation data during training history, on the pair of frames using right flash and left flash (P_3). (Note that the neural network classification is binary - real face or fake; the three categories are for the aid of the reader)

5
Conclusion

This thesis investigated a spoof detection solution that used a lightweight convolutional neural network to classify faces as live or spoof based on footage from a NIR-camera with multiple active NIR-sources. We can conclude that classification of faces as spoof or live can be done with high accuracy, using footage from a NIR-camera and the lightweight neural network architecture MobileNetV2 (originally presented in [15]). To achieve these results, one needs to feed the full face region to the classifer, as just feeding the eye region results in classification with considerably lower accuracy, especially for masks. The best approach found was to use one flash very close to the camera, and a second flash further to the side of the camera. This resulted in a classifier that could correctly classify live frames of previously unseen faces as live 100 % of the time, print frames as spoof 99.7 % of the time and mask frames 94.1 % of the time. However, while the best accuracy was achieved when using two flash configurations, results using only a single flash were not dramatically worse.

To make a more robst classifier, and to enable more accurate evaluation of performance, future work should aim to scale up the database used, preferably including at least 100 subjects of different age, sex and appearance. Data should be gathered in conditions that more closely resemble the intended automotive application, and should include more variation in terms of subject pose as well as light conditions. This could be done by making recordings of subjects in an actual vehicle, which naturally provides a variety of light conditions and head poses in the relevant environment. More masks need to be used for attack recordings as well, and it could be good idea to include more attack categories, such as replay attacks on screens of smartphones and tablets. Another area that could be investigated further is the architecture and training of the classifier. For example, if a recurrent neural network were used, it might be able to take dynamical information into account.

Bibliography

[1] Quantum error correction for the toric code using deep reinforcement learning. *Quantum*, 3:183, September 2019.

[2] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[3] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML '08, page 160–167, New York, NY, USA, 2008. Association for Computing Machinery.

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

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017, 1704.04861.

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

[7] S. Lawrence, C. L. Giles, A. C. Tsoi, and A. D. Back. Face recognition: a convolutional neural-network approach. *IEEE Transactions on Neural Networks*, 8(1):98–113, 1997.

[8] Y. Liu, A. Jourabloo, and X. Liu. Learning deep models for face anti-spoofing: Binary or auxiliary supervision. *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 389–398, 2018.

[9] Y. Liu, J. Stehouwer, A. Jourabloo, and X. Liu. Deep tree learning for zero-shot face anti-spoofing. *The IEEE Conference on Computer Vision and Pattern Recognition*, pages 4680–4689, 2019.

[10] S. Mohamed, A. Ghoneim, and A. Youssif. Visible/infrared face spoofing detection using texture descriptors. *International Conference on Circuits, Systems, Communications and Computers*, 23, September 2019.

[11] L. Nataraj, T. M. Mohammed, B. S. Manjunath, S. C., A. F., J. H. Bappy, and A. K. Roy-Chowdhury. Detecting GAN generated fake images using co-occurrence matrices. *CoRR*, abs/1903.06836, 2019, 1903.06836.

[12] J. C. Neves, R. Tolosana, R. Vera-Rodriguez, V. Lopes, and H. Proenca. Real or fake? Spoofing state-of-the-art face synthesis detection systems. *arXiv preprint arXiv:1911.05351v1*, 2019.

[13] G. Pan, L. Sun, and Z. Wu. Eyeblink-based anti-spoofing in face recognition from a generic webcamera. *IEEE International Conference on Computer Vision*, 11, October 2007.

[14] M. Sajjad, S. Khan, T. Hussain, K. Muhammad, K. A. Sangaiah, A. Castiglione, C. Esposito, , and S. W. Baik. Cnn-based anti-spoofing two-tier multi-factor authentication system. *Pattern Recognition Letters*, 126:123–131, February 2019.

[15] M. Sandler, A.G. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. *CoRR*, abs/1801.04381, 2018, 1801.04381.

[16] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550, October 2017.

[17] X. Sun, L. Huang, and C. Liu. Context based face spoofing detection using active near-infrared images. *International Conference on Pattern Recognition*, 23:4262–4267, December 2016.

[18] Ruben Tolosana, Ruben Vera-Rodriguez, Julian Fierrez, Aythami Morales, and Javier Ortega-Garcia. Deepfakes and beyond: A survey of face manipulation and fake detection. *arXiv preprint arXiv:2001.00179*, 2020.

[19] F. Wang and J. Han. Robust multimodal biometric authentication integrating iris, face and palmprint. *Information Technology And Control*, 37:326–332, April 2008.

[20] X. Yang, Y. Li, and S. Lyu. Exposing deep fakes using inconsistent head poses. *CoRR*, abs/1811.00661, 2018, 1811.00661.

[21] X. Yang, W. Luo, L. Bao, Y. Gao, D. Gong, S. Zheng, Z. Li, , and W. Liu. Face anti-spoofing: Model matters, so does data. *Conference on Computer Vision and Pattern Recognition*, pages 3507–3516, June 2019.

[22] X. Zhang, X. Zhou, M. Lin, and J. Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017, 1707.01083.

A

appendix 1

A.1 Neural Network for Face Region

This section contains a tabular representation of the detailed neural network architecture used for evaluating a face region. Each entry represents a network layer. Layer types are are denoted by their name in the neural network library Keras, for python. Note that some of the layers are divided by name into numbered blocks. These are the residual bottleneck modules, which are central to this neural network architecture as seen in section 2.1.2.6. Tensor shapes are given as (height, width, depth).

Layer (type)	Out Shape	Connected to
input_1 (InputLayer)	(250, 200, 2)	
Conv1_pad (ZeroPadding2D)	(251, 201, 2)	input_1
Conv1 (Conv2D)	(125, 100, 8)	Conv1_pad
bn_Conv1 (BatchNormalization)	(125, 100, 8)	Conv1
Conv1_relu (ReLU)	(125, 100, 8)	bn_Conv1
expanded_conv_depthwise (DepthwiseConv2D)	(125, 100, 8)	Conv1_relu
expanded_conv_depthwise_BN (BatchNormalization)	(125, 100, 8)	expanded_conv_depthwise
expanded_conv_depthwise_relu (ReLU)	(125, 100, 8)	expanded_conv_depthwise_BN
expanded_conv_project (Conv2D)	(125, 100, 8)	expanded_conv_depthwise_relu
expanded_conv_project_BN (BatchNormalization)	(125, 100, 8)	expanded_conv_project
expanded_conv_add (Add)	(125, 100, 8)	Conv1_relu expanded_conv_project_BN
block_1_expand (Conv2D)	(125, 100, 48)	expanded_conv_add
block_1_expand_BN (BatchNormalization)	(125, 100, 48)	block_1_expand
block_1_expand_relu (ReLU)	(125, 100, 48)	block_1_expand_BN
block_1_pad (ZeroPadding2D)	(127, 101, 48)	block_1_expand_relu
block_1_depthwise (DepthwiseConv2D)	(63, 50, 48)	block_1_pad
block_1_depthwise_BN (BatchNormalization)	(63, 50, 48)	block_1_depthwise
block_1_depthwise_relu (ReLU)	(63, 50, 48)	block_1_depthwise_BN
block_1_project (Conv2D)	(63, 50, 8)	block_1_depthwise_relu
block_1_project_BN (BatchNormalization)	(63, 50, 8)	block_1_project
block_2_expand (Conv2D)	(63, 50, 48)	block_1_project_BN
block_2_expand_BN (BatchNormalization)	(63, 50, 48)	block_2_expand
block_2_expand_relu (ReLU)	(63, 50, 48)	block_2_expand_BN
block_2_depthwise (DepthwiseConv2D)	(63, 50, 48)	block_2_expand_relu
block_2_depthwise_BN (BatchNormalization)	(63, 50, 48)	block_2_depthwise
block_2_depthwise_relu (ReLU)	(63, 50, 48)	block_2_depthwise_BN
block_2_project (Conv2D)	(63, 50, 8)	block_2_depthwise_relu
block_2_project_BN (BatchNormalization)	(63, 50, 8)	block_2_project
block_2_add (Add)	(63, 50, 8)	block_1_project_BN block_2_project_BN

block_3_expand (Conv2D)	(63, 50, 48)	block_2_add
block_3_expand_BN (BatchNormalization)	(63, 50, 48)	block_3_expand
block_3_expand_relu (ReLU)	(63, 50, 48)	block_3_expand_BN
block_3_pad (ZeroPadding2D)	(65, 51, 48)	block_3_expand_relu
block_3_depthwise (DepthwiseConv2D)	(32, 25, 48)	block_3_pad
block_3_depthwise_BN (BatchNormalization)	(32, 25, 48)	block_3_depthwise
block_3_depthwise_relu (ReLU)	(32, 25, 48)	block_3_depthwise_BN
block_3_project (Conv2D)	(32, 25, 8)	block_3_depthwise_relu
block_3_project_BN (BatchNormalization)	(32, 25, 8)	block_3_project
block_4_expand (Conv2D)	(32, 25, 48)	block_3_project_BN
block_4_expand_BN (BatchNormalization)	(32, 25, 48)	block_4_expand
block_4_expand_relu (ReLU)	(32, 25, 48)	block_4_expand_BN
block_4_depthwise (DepthwiseConv2D)	(32, 25, 48)	block_4_expand_relu
block_4_depthwise_BN (BatchNormalization)	(32, 25, 48)	block_4_depthwise
block_4_depthwise_relu (ReLU)	(32, 25, 48)	block_4_depthwise_BN
block_4_project (Conv2D)	(32, 25, 8)	block_4_depthwise_relu
block_4_project_BN (BatchNormalization)	(32, 25, 8)	block_4_project
block_4_add (Add)	(32, 25, 8)	block_3_project_BN block_4_project_BN
block_5_expand (Conv2D)	(32, 25, 48)	block_4_add
block_5_expand_BN (BatchNormalization)	(32, 25, 48)	block_5_expand
block_5_expand_relu (ReLU)	(32, 25, 48)	block_5_expand_BN
block_5_depthwise (DepthwiseConv2D)	(32, 25, 48)	block_5_expand_relu
block_5_depthwise_BN (BatchNormalization)	(32, 25, 48)	block_5_depthwise
block_5_depthwise_relu (ReLU)	(32, 25, 48)	block_5_depthwise_BN
block_5_project (Conv2D)	(32, 25, 8)	block_5_depthwise_relu
block_5_project_BN (BatchNormalization)	(32, 25, 8)	block_5_project
block_5_add (Add)	(32, 25, 8)	block_4_add block_5_project_BN

block_6_expand (Conv2D)	(32, 25, 48)	block_5_add
block_6_expand_BN (BatchNormalization)	(32, 25, 48)	block_6_expand
block_6_expand_relu (ReLU)	(32, 25, 48)	block_6_expand_BN
block_6_pad (ZeroPadding2D)	(33, 27, 48)	block_6_expand_relu
block_6_depthwise (DepthwiseConv2D)	(16, 13, 48)	block_6_pad
block_6_depthwise_BN (BatchNormalization)	(16, 13, 48)	block_6_depthwise
block_6_depthwise_relu (ReLU)	(16, 13, 48)	block_6_depthwise_BN
block_6_project (Conv2D)	(16, 13, 8)	block_6_depthwise_relu
block_6_project_BN (BatchNormalization)	(16, 13, 8)	block_6_project
block_7_expand (Conv2D)	(16, 13, 48)	block_6_project_BN
block_7_expand_BN (BatchNormalization)	(16, 13, 48)	block_7_expand
block_7_expand_relu (ReLU)	(16, 13, 48)	block_7_expand_BN
block_7_depthwise (DepthwiseConv2D)	(16, 13, 48)	block_7_expand_relu
block_7_depthwise_BN (BatchNormalization)	(16, 13, 48)	block_7_depthwise
block_7_depthwise_relu (ReLU)	(16, 13, 48)	block_7_depthwise_BN
block_7_project (Conv2D)	(16, 13, 8)	block_7_depthwise_relu
block_7_project_BN (BatchNormalization)	(16, 13, 8)	block_7_project
block_7_add (Add)	(16, 13, 8)	block_6_project_BN block_7_project_BN
block_8_expand (Conv2D)	(16, 13, 48)	block_7_add
block_8_expand_BN (BatchNormalization)	(16, 13, 48)	block_8_expand
block_8_expand_relu (ReLU)	(16, 13, 48)	block_8_expand_BN
block_8_depthwise (DepthwiseConv2D)	(16, 13, 48)	block_8_expand_relu
block_8_depthwise_BN (BatchNormalization)	(16, 13, 48)	block_8_depthwise
block_8_depthwise_relu (ReLU)	(16, 13, 48)	block_8_depthwise_BN
block_8_project (Conv2D)	(16, 13, 8)	block_8_depthwise_relu
block_8_project_BN (BatchNormalization)	(16, 13, 8)	block_8_project
block_8_add (Add)	(16, 13, 8)	block_7_add block_8_project_BN

block_9_expand (Conv2D)	(16, 13, 48)	block_8_add
block_9_expand_BN (BatchNormalization)	(16, 13, 48)	block_9_expand
block_9_expand_relu (ReLU)	(16, 13, 48)	block_9_expand_BN
block_9_depthwise (DepthwiseConv2D)	(16, 13, 48)	block_9_expand_relu
block_9_depthwise_BN (BatchNormalization)	(16, 13, 48)	block_9_depthwise
block_9_depthwise_relu (ReLU)	(16, 13, 48)	block_9_depthwise_BN
block_9_project (Conv2D)	(16, 13, 8)	block_9_depthwise_relu
block_9_project_BN (BatchNormalization)	(16, 13, 8)	block_9_project
block_9_add (Add)	(16, 13, 8)	block_8_add block_9_project_BN
block_10_expand (Conv2D)	(16, 13, 48)	block_9_add
block_10_expand_BN (BatchNormalization)	(16, 13, 48)	block_10_expand
block_10_expand_relu (ReLU)	(16, 13, 48)	block_10_expand_BN
block_10_depthwise (DepthwiseConv2D)	(16, 13, 48)	block_10_expand_relu
block_10_depthwise_BN (BatchNormalization)	(16, 13, 48)	block_10_depthwise
block_10_depthwise_relu (ReLU)	(16, 13, 48)	block_10_depthwise_BN
block_10_project (Conv2D)	(16, 13, 16)	block_10_depthwise_relu
block_10_project_BN (BatchNormalization)	(16, 13, 16)	block_10_project
block_11_expand (Conv2D)	(16, 13, 96)	block_10_project_BN
block_11_expand_BN (BatchNormalization)	(16, 13, 96)	block_11_expand
block_11_expand_relu (ReLU)	(16, 13, 96)	block_11_expand_BN
block_11_depthwise (DepthwiseConv2D)	(16, 13, 96)	block_11_expand_relu
block_11_depthwise_BN (BatchNormalization)	(16, 13, 96)	block_11_depthwise
block_11_depthwise_relu (ReLU)	(16, 13, 96)	block_11_depthwise_BN
block_11_project (Conv2D)	(16, 13, 16)	block_11_depthwise_relu
block_11_project_BN (BatchNormalization)	(16, 13, 16)	block_11_project
block_11_add (Add)	(16, 13, 16)	block_10_project_BN block_11_project_BN

block_12_expand (Conv2D)	(16, 13, 96)	block_11_add
block_12_expand_BN (BatchNormalization)	(16, 13, 96)	block_12_expand
block_12_expand_relu (ReLU)	(16, 13, 96)	block_12_expand_BN
block_12_depthwise (DepthwiseConv2D)	(16, 13, 96)	block_12_expand_relu
block_12_depthwise_BN (BatchNormalization)	(16, 13, 96)	block_12_depthwise
block_12_depthwise_relu (ReLU)	(16, 13, 96)	block_12_depthwise_BN
block_12_project (Conv2D)	(16, 13, 16)	block_12_depthwise_relu
block_12_project_BN (BatchNormalization)	(16, 13, 16)	block_12_project
block_12_add (Add)	(16, 13, 16)	block_11_add block_12_project_BN
block_13_expand (Conv2D)	(16, 13, 96)	block_12_add
block_13_expand_BN (BatchNormalization)	(16, 13, 96)	block_13_expand
block_13_expand_relu (ReLU)	(16, 13, 96)	block_13_expand_BN
block_13_pad (ZeroPadding2D)	(17, 15, 96)	block_13_expand_relu
block_13_depthwise (DepthwiseConv2D)	(8, 7, 96)	block_13_pad
block_13_depthwise_BN (BatchNormalization)	(8, 7, 96)	block_13_depthwise
block_13_depthwise_relu (ReLU)	(8, 7, 96)	block_13_depthwise_BN
block_13_project (Conv2D)	(8, 7, 16)	block_13_depthwise_relu
block_13_project_BN (BatchNormalization)	(8, 7, 16)	block_13_project
block_14_expand (Conv2D)	(8, 7, 96)	block_13_project_BN
block_14_expand_BN (BatchNormalization)	(8, 7, 96)	block_14_expand
block_14_expand_relu (ReLU)	(8, 7, 96)	block_14_expand_BN
block_14_depthwise (DepthwiseConv2D)	(8, 7, 96)	block_14_expand_relu
block_14_depthwise_BN (BatchNormalization)	(8, 7, 96)	block_14_depthwise
block_14_depthwise_relu (ReLU)	(8, 7, 96)	block_14_depthwise_BN
block_14_project (Conv2D)	(8, 7, 16)	block_14_depthwise_relu
block_14_project_BN (BatchNormalization)	(8, 7, 16)	block_14_project
block_14_add (Add)	(8, 7, 16)	block_13_project_BN block_14_project_BN

block_15_expand (Conv2D)	(8, 7, 96)	block_14_add
block_15_expand_BN (BatchNormalization)	(8, 7, 96)	block_15_expand
block_15_expand_relu (ReLU)	(8, 7, 96)	block_15_expand_BN
block_15_depthwise (DepthwiseConv2D)	(8, 7, 96)	block_15_expand_relu
block_15_depthwise_BN (BatchNormalization)	(8, 7, 96)	block_15_depthwise
block_15_depthwise_relu (ReLU)	(8, 7, 96)	block_15_depthwise_BN
block_15_project (Conv2D)	(8, 7, 16)	block_15_depthwise_relu
block_15_project_BN (BatchNormalization)	(8, 7, 16)	block_15_project
block_15_add (Add)	(8, 7, 16)	block_14_add block_15_project_BN
block_16_expand (Conv2D)	(8, 7, 96)	block_15_add
block_16_expand_BN (BatchNormalization)	(8, 7, 96)	block_16_expand
block_16_expand_relu (ReLU)	(8, 7, 96)	block_16_expand_BN
block_16_depthwise (DepthwiseConv2D)	(8, 7, 96)	block_16_expand_relu
block_16_depthwise_BN (BatchNormalization)	(8, 7, 96)	block_16_depthwise
block_16_depthwise_relu (ReLU)	(8, 7, 96)	block_16_depthwise_BN
block_16_project (Conv2D)	(8, 7, 32)	block_16_depthwise_relu
block_16_project_BN (BatchNormalization)	(8, 7, 32)	block_16_project
Conv_1 (Conv2D)	(8, 7, 1280)	block_16_project_BN
Conv_1_bn (BatchNormalization)	(8, 7, 1280)	Conv_1
out_relu (ReLU)	(8, 7, 1280)	Conv_1_bn
global_average_pooling2d (GlobalAveragePooling2D)	(1280)	out_relu
Logits (Dense)	(1)	global_average_pooling2d

A.2 Neural Network for Eye Region

This section contains a tabular representation of the detailed neural network architecture used for evaluating a face region. Each entry represents a network layer. Layer types are are denoted by their name in the neural network library Keras, for python. Note that some of the layers are divided by name into numbered blocks. These are the residual bottleneck modules, which are central to this neural network architecture as seen in section 2.1.2.6. Tensor shapes are given as (height, width, depth).

layer (type)	out shape	connected to
input_1 (inputlayer)	(48, 48, 2)	
conv1_pad (zeropadding2d)	(49, 49, 2)	input_1
conv1 (conv2d)	(24, 24, 8)	conv1_pad
bn_conv1 (batchnormalization)	(24, 24, 8)	conv1
conv1_relu (relu)	(24, 24, 8)	bn_conv1
expanded_conv_depthwise (DepthwiseConv2D)	(24, 24, 8)	conv1_relu
expanded_conv_depthwise_bn (BatchNormalization)	(24, 24, 8)	expanded_conv_depthwise
expanded_conv_depthwise_relu (ReLU)	(24, 24, 8)	expanded_conv_depthwise_bn
expanded_conv_project (conv2d)	(24, 24, 8)	expanded_conv_depthwise_relu
expanded_conv_project_bn (BatchNormalization)	(24, 24, 8)	expanded_conv_project
expanded_conv_add (add)	(24, 24, 8)	conv1_relu expanded_conv_project_bn
block_1_expand (conv2d)	(24, 24, 48)	expanded_conv_add
block_1_expand_bn (BatchNormalization)	(24, 24, 48)	block_1_expand
block_1_expand_relu (relu)	(24, 24, 48)	block_1_expand_bn
block_1_pad (zeropadding2d)	(25, 25, 48)	block_1_expand_relu
block_1_depthwise (DepthwiseConv2D)	(12, 12, 48)	block_1_pad
block_1_depthwise_bn (BatchNormalization)	(12, 12, 48)	block_1_depthwise
block_1_depthwise_relu (relu)	(12, 12, 48)	block_1_depthwise_bn
block_1_project (conv2d)	(12, 12, 8)	block_1_depthwise_relu
block_1_project_bn (BatchNormalization)	(12, 12, 8)	block_1_project
block_2_expand (conv2d)	(12, 12, 48)	block_1_project_bn
block_2_expand_bn (BatchNormalization)	(12, 12, 48)	block_2_expand
block_2_expand_relu (relu)	(12, 12, 48)	block_2_expand_bn
block_2_depthwise (DepthwiseConv2D)	(12, 12, 48)	block_2_expand_relu
block_2_depthwise_bn (BatchNormalization)	(12, 12, 48)	block_2_depthwise
block_2_depthwise_relu (ReLU)	(12, 12, 48)	block_2_depthwise_BN
block_2_project (Conv2D)	(12, 12, 8)	block_2_depthwise_relu
block_2_project_BN (BatchNormalization)	(12, 12, 8)	block_2_project
block_2_add (Add)	(12, 12, 8)	block_1_project_BN block_2_project_BN

block_3_expand (Conv2D)	(12, 12, 48)	block_2_add
block_3_expand_BN (BatchNormalization)	(12, 12, 48)	block_3_expand
block_3_expand_relu (ReLU)	(12, 12, 48)	block_3_expand_BN
block_3_pad (ZeroPadding2D)	(13, 13, 48)	block_3_expand_relu
block_3_depthwise (DepthwiseConv2D)	(6, 6, 48)	block_3_pad
block_3_depthwise_BN (BatchNormalization)	(6, 6, 48)	block_3_depthwise
block_3_depthwise_relu (ReLU)	(6, 6, 48)	block_3_depthwise_BN
block_3_project (Conv2D)	(6, 6, 8)	block_3_depthwise_relu
block_3_project_BN (BatchNormalization)	(6, 6, 8)	block_3_project
block_4_expand (Conv2D)	(6, 6, 48)	block_3_project_BN
block_4_expand_BN (BatchNormalization)	(6, 6, 48)	block_4_expand
block_4_expand_relu (ReLU)	(6, 6, 48)	block_4_expand_BN
block_4_depthwise (DepthwiseConv2D)	(6, 6, 48)	block_4_expand_relu
block_4_depthwise_BN (BatchNormalization)	(6, 6, 48)	block_4_depthwise
block_4_depthwise_relu (ReLU)	(6, 6, 48)	block_4_depthwise_BN
block_4_project (Conv2D)	(6, 6, 8)	block_4_depthwise_relu
block_4_project_BN (BatchNormalization)	(6, 6, 8)	block_4_project
block_4_add (Add)	(6, 6, 8)	block_3_project_BN block_4_project_BN
block_5_expand (Conv2D)	(6, 6, 48)	block_4_add
block_5_expand_BN (BatchNormalization)	(6, 6, 48)	block_5_expand
block_5_expand_relu (ReLU)	(6, 6, 48)	block_5_expand_BN
block_5_depthwise (DepthwiseConv2D)	(6, 6, 48)	block_5_expand_relu
block_5_depthwise_BN (BatchNormalization)	(6, 6, 48)	block_5_depthwise
block_5_depthwise_relu (ReLU)	(6, 6, 48)	block_5_depthwise_BN
block_5_project (Conv2D)	(6, 6, 8)	block_5_depthwise_relu
block_5_project_BN (BatchNormalization)	(6, 6, 8)	block_5_project
block_5_add (Add)	(6, 6, 8)	block_4_add block_5_project_BN

block_6_expand (Conv2D)	(6, 6, 48)	block_5_add
block_6_expand_BN (BatchNormalization)	(6, 6, 48)	block_6_expand
block_6_expand_relu (ReLU)	(6, 6, 48)	block_6_expand_BN
block_6_pad (ZeroPadding2D)	(7, 7, 48)	block_6_expand_relu
block_6_depthwise (DepthwiseConv2D)	(3, 3, 48)	block_6_pad
block_6_depthwise_BN (BatchNormalization)	(3, 3, 48)	block_6_depthwise
block_6_depthwise_relu (ReLU)	(3, 3, 48)	block_6_depthwise_BN
block_6_project (Conv2D)	(3, 3, 8)	block_6_depthwise_relu
block_6_project_BN (BatchNormalization)	(3, 3, 8)	block_6_project
block_7_expand (Conv2D)	(3, 3, 48)	block_6_project_BN
block_7_expand_BN (BatchNormalization)	(3, 3, 48)	block_7_expand
block_7_expand_relu (ReLU)	(3, 3, 48)	block_7_expand_BN
block_7_depthwise (DepthwiseConv2D)	(3, 3, 48)	block_7_expand_relu
block_7_depthwise_BN (BatchNormalization)	(3, 3, 48)	block_7_depthwise
block_7_depthwise_relu (ReLU)	(3, 3, 48)	block_7_depthwise_BN
block_7_project (Conv2D)	(3, 3, 8)	block_7_depthwise_relu
block_7_project_BN (BatchNormalization)	(3, 3, 8)	block_7_project
block_7_add (Add)	(3, 3, 8)	block_6_project_BN block_7_project_BN
block_8_expand (Conv2D)	(3, 3, 48)	block_7_add
block_8_expand_BN (BatchNormalization)	(3, 3, 48)	block_8_expand
block_8_expand_relu (ReLU)	(3, 3, 48)	block_8_expand_BN
block_8_depthwise (DepthwiseConv2D)	(3, 3, 48)	block_8_expand_relu
block_8_depthwise_BN (BatchNormalization)	(3, 3, 48)	block_8_depthwise
block_8_depthwise_relu (ReLU)	(3, 3, 48)	block_8_depthwise_BN
block_8_project (Conv2D)	(3, 3, 8)	block_8_depthwise_relu
block_8_project_BN (BatchNormalization)	(3, 3, 8)	block_8_project
block_8_add (Add)	(3, 3, 8)	block_7_add block_8_project_BN

block_9_expand (Conv2D)	(3, 3, 48)	block_8_add
block_9_expand_BN (BatchNormalization)	(3, 3, 48)	block_9_expand
block_9_expand_relu (ReLU)	(3, 3, 48)	block_9_expand_BN
block_9_depthwise (DepthwiseConv2D)	(3, 3, 48)	block_9_expand_relu
block_9_depthwise_BN (BatchNormalization)	(3, 3, 48)	block_9_depthwise
block_9_depthwise_relu (ReLU)	(3, 3, 48)	block_9_depthwise_BN
block_9_project (Conv2D)	(3, 3, 8)	block_9_depthwise_relu
block_9_project_BN (BatchNormalization)	(3, 3, 8)	block_9_project
block_9_add (Add)	(3, 3, 8)	block_8_add block_9_project_BN
block_10_expand (Conv2D)	(3, 3, 48)	block_9_add
block_10_expand_BN (BatchNormalization)	(3, 3, 48)	block_10_expand
block_10_expand_relu (ReLU)	(3, 3, 48)	block_10_expand_BN
block_10_depthwise (DepthwiseConv2D)	(3, 3, 48)	block_10_expand_relu
block_10_depthwise_BN (BatchNormalization)	(3, 3, 48)	block_10_depthwise
block_10_depthwise_relu (ReLU)	(3, 3, 48)	block_10_depthwise_BN
block_10_project (Conv2D)	(3, 3, 16)	block_10_depthwise_relu
block_10_project_BN (BatchNormalization)	(3, 3, 16)	block_10_project
block_11_expand (Conv2D)	(3, 3, 96)	block_10_project_BN
block_11_expand_BN (BatchNormalization)	(3, 3, 96)	block_11_expand
block_11_expand_relu (ReLU)	(3, 3, 96)	block_11_expand_BN
block_11_depthwise (DepthwiseConv2D)	(3, 3, 96)	block_11_expand_relu
block_11_depthwise_BN (BatchNormalization)	(3, 3, 96)	block_11_depthwise
block_11_depthwise_relu (ReLU)	(3, 3, 96)	block_11_depthwise_BN
block_11_project (Conv2D)	(3, 3, 16)	block_11_depthwise_relu
block_11_project_BN (BatchNormalization)	(3, 3, 16)	block_11_project
block_11_add (Add)	(3, 3, 16)	block_10_project_BN block_11_project_BN

block_12_expand (Conv2D)	(3, 3, 96)	block_11_add
block_12_expand_BN (BatchNormalization)	(3, 3, 96)	block_12_expand
block_12_expand_relu (ReLU)	(3, 3, 96)	block_12_expand_BN
block_12_depthwise (DepthwiseConv2D)	(3, 3, 96)	block_12_expand_relu
block_12_depthwise_BN (BatchNormalization)	(3, 3, 96)	block_12_depthwise
block_12_depthwise_relu (ReLU)	(3, 3, 96)	block_12_depthwise_BN
block_12_project (Conv2D)	(3, 3, 16)	block_12_depthwise_relu
block_12_project_BN (BatchNormalization)	(3, 3, 16)	block_12_project
block_12_add (Add)	(3, 3, 16)	block_11_add block_12_project_BN
block_13_expand (Conv2D)	(3, 3, 96)	block_12_add
block_13_expand_BN (BatchNormalization)	(3, 3, 96)	block_13_expand
block_13_expand_relu (ReLU)	(3, 3, 96)	block_13_expand_BN
block_13_pad (ZeroPadding2D)	(5, 5, 96)	block_13_expand_relu
block_13_depthwise (DepthwiseConv2D)	(2, 2, 96)	block_13_pad
block_13_depthwise_BN (BatchNormalization)	(2, 2, 96)	block_13_depthwise
block_13_depthwise_relu (ReLU)	(2, 2, 96)	block_13_depthwise_BN
block_13_project (Conv2D)	(2, 2, 16)	block_13_depthwise_relu
block_13_project_BN (BatchNormalization)	(2, 2, 16)	block_13_project
block_14_expand (Conv2D)	(2, 2, 96)	block_13_project_BN
block_14_expand_BN (BatchNormalization)	(2, 2, 96)	block_14_expand
block_14_expand_relu (ReLU)	(2, 2, 96)	block_14_expand_BN
block_14_depthwise (DepthwiseConv2D)	(2, 2, 96)	block_14_expand_relu
block_14_depthwise_BN (BatchNormalization)	(2, 2, 96)	block_14_depthwise
block_14_depthwise_relu (ReLU)	(2, 2, 96)	block_14_depthwise_BN
block_14_project (Conv2D)	(2, 2, 16)	block_14_depthwise_relu
block_14_project_BN (BatchNormalization)	(2, 2, 16)	block_14_project
block_14_add (Add)	(2, 2, 16)	block_13_project_BN block_14_project_BN

block_15_expand (Conv2D)	(2, 2, 96)	block_14_add
block_15_expand_BN (BatchNormalization)	(2, 2, 96)	block_15_expand
block_15_expand_relu (ReLU)	(2, 2, 96)	block_15_expand_BN
block_15_depthwise (DepthwiseConv2D)	(2, 2, 96)	block_15_expand_relu
block_15_depthwise_BN (BatchNormalization)	(2, 2, 96)	block_15_depthwise
block_15_depthwise_relu (ReLU)	(2, 2, 96)	block_15_depthwise_BN
block_15_project (Conv2D)	(2, 2, 16)	block_15_depthwise_relu
block_15_project_BN (BatchNormalization)	(2, 2, 16)	block_15_project
block_15_add (Add)	(2, 2, 16)	block_14_add block_15_project_BN
block_16_expand (Conv2D)	(2, 2, 96)	block_15_add
block_16_expand_BN (BatchNormalization)	(2, 2, 96)	block_16_expand
block_16_expand_relu (ReLU)	(2, 2, 96)	block_16_expand_BN
block_16_depthwise (DepthwiseConv2D)	(2, 2, 96)	block_16_expand_relu
block_16_depthwise_BN (BatchNormalization)	(2, 2, 96)	block_16_depthwise
block_16_depthwise_relu (ReLU)	(2, 2, 96)	block_16_depthwise_BN
block_16_project (Conv2D)	(2, 2, 32)	block_16_depthwise_relu
block_16_project_BN (BatchNormalization)	(2, 2, 32)	block_16_project
Conv_1 (Conv2D)	(2, 2, 1280)	block_16_project_BN
Conv_1_bn (BatchNormalization)	(2, 2, 1280)	Conv_1
out_relu (ReLU)	(2, 2, 1280)	Conv_1_bn
global_average_pooling2d (GlobalAveragePooling2D)	(1280)	out_relu
Logits (Dense)	(1)	global_average_pooling2d