THE UNIVERSITY OF QUEENSLAND

AUSTRALIA

Remote Camera Identification Using Deep Models

Omar Bassem Jarkas

Masters of Software Engineering, UQ

s4598179

A thesis submitted for the degree of Masters of Cyber Security at The University of Queensland in 2021

CYBR7902

Abstract

Remote Camera Identification using Deep Models

Abstract

Omar Jarkas, The University of Queensland, 2021

Remote camera identification is the ability to authenticate images by identifying their source. Such identification falls under passive camera security. Passive security is the ability to ensure some kind of feed assurance without involving active security. Active security is the process of securing the different layers of the camera. The problem with active security is that it often requires heavy security requirements. Such requirements can be infeasible or impractical. Remote camera identification relies on exploiting unique features to fingerprint and identify images. With remote camera identification, we can achieve some degree of feed assurance via passive security. This is since if we can identify the camera source based on the images it produces, we can authenticate images. This process can aid the process of remote camera attestation, hence passive security. Passive security can be done remotely and doesn't assume any hardware and software level security. In this work, we present an automated end-to-end solution for remote camera identification. Our solution is a significant performance enhancement over other techniques. The solution achieves scalability by implementing a design that learns noise patterns of images. Based on such noise the model is able to distinguish the source. The solution achieves scalability using the following constraints. The entire system is automated into one pipeline. The system attempts to remove many biases that hinder identification and fingerprinting. The system combines effective methods together for fingerprinting and identification. The technique extracts photo-response non-uniformity noise from raw images using the wavelet transform. Then the design uses a deep learning model, Convolutional Neural Network, to learn image noise patterns. The model achieves a 99% identification accuracy with a maximum of 5 epochs.

Declaration by author

This thesis is composed of my original work, and contains no material previously published or written by another person except where due reference has been made in the text. I have clearly stated the contribution by others to jointly-authored works that I have included in my thesis.

I have clearly stated the contribution of others to my thesis as a whole, including statistical assistance, survey design, data analysis, significant technical procedures, professional editorial advice, financial support and any other original research work used or reported in my thesis. The content of my thesis is the result of work I have carried out since the commencement of my higher degree by research candidature and does not include a substantial part of work that has been submitted to qualify for the award of any other degree or diploma in any university or other tertiary institution. I have clearly stated which parts of my thesis, if any, have been submitted to qualify for another award.

I acknowledge that an electronic copy of my thesis must be lodged with the University Library and, subject to the policy and procedures of The University of Queensland, the thesis be made available for research and study in accordance with the Copyright Act 1968 unless a period of embargo has been approved by the Dean of the Graduate School.

I acknowledge that copyright of all material contained in my thesis resides with the copyright holder(s) of that material. Where appropriate I have obtained copyright permission from the copyright holder to reproduce material in this thesis and have sought permission from co-authors for any jointly authored works included in the thesis.

Acknowledgments

The thesis couldn't have been possible without the guidance and mentorship of Joshua Scarsbrook, and Doctor Joshua Hall. They provided encouragement, feedback, guidance, and expertise.
The research required experimenting with different machine learning domains. A big thank you to Doctor Parham Kojesteh for guidance and consultation in the AI field. A big thank you the Jeremy Hamilton for his infographics and infographic consultation.
A big thank you to Bondi Labs' team for their support. Special thank you to a great boss who always inspires me and the team, Jonathan Marshall, Bondi Labs.
A big thank you to Professor Ryan Ko and Doctor Guangdong Bai for the supervision, guidance, and support.

Financial support

The research is sponsored under the Innovation Connections. The work was a collaboration between Bondi Labs and the University of Queensland to deliver a more secure proprietary industry-grade software.

“Innovation Connections is a service under the AusIndustry Entrepreneurs’ Programme that provides you with advice and funding to get research projects underway in your business. ”

Keywords

Security, CNN, PRNU, Camera, Noise

This thesis is dedicated to nerds everywhere.

Contents

List of Figures

List of Abbreviations and Symbols

Abbreviations	
ISCI	Individual Source Camera Identification
SCMI	Source Camera Model Identification
ANN	Artificial Neural Networks
CNN	Convolutional Neural Networks
SVM	Support Vector Machine
SPN	Sensor Pattern Noise
PRNU	Photo Response Non Uniformity
IP-Cameras	Internet-Protocol Cameras
IoT	Internet of Things
SDK	Software Development Kit
PSO	Particle Swarm Optimization
FFT	Fast Fourier Transform
STFT	Short Time Fourier transform
DSP	Digital Signal Processing
WT	Wavelet Transform

Symbols	
Δt	Time Bound
Δf	Frequency Bound
τ	Time Localizing
ω	Frequency Domain

Chapter 1

Introduction

This section introduces source camera identification using deep models. Following the introduction, the section covers the overview and background of the work done in this research.

1.1 Remote Camera Identification

Remote camera identification is the process of distinguishing the feed or camera source. Such identification can provide some feed assurance and attestation. With such technique camera feed can be further secured. Remote camera identification is a means for such security.

The problem with camera security is a problem for many industries that rely on cameras as a vital part of their technology or service. This problem is relayed further when dealing with IP cameras and web technologies. The problem with modern cameras is that they are ill-equipped to secure against the web's threat surface. Modern IP cameras, like many IoT devices, are commodity products with cheap manufacturing and low security. This is not to say that there hasn't been active research in IoT and hardware security [3]. Yet, the problem with such research is that often it requires heavy software and hardware overhead on the part of the manufacturer or the customer [4]. Such hardware and software constraints for security might be infeasible or impractical [5, 6].

Another approach for feed security assurance is passive identification via remote camera identification. Remote camera identification is the process of exploiting camera characteristics for fingerprinting [7]. If we can fingerprint frames and images we can identify their source thus authenticating it. Such identification can go further to provide integrity checks.

1.1.1 Fingerprinting and Identification

The problem with remote camera identification is that the fingerprinting and identification processes are not trivial [7]. To fingerprint an image, we need to exploit a unique dominant characteristic of the camera evident in the final image [7]. Furthermore, assuming fingerprinting is successful, we also need an identification scheme. With that said, one of the camera features that fit the fingerprinting constraints is image noise generated by the camera. While there are many types of errors generated by the camera

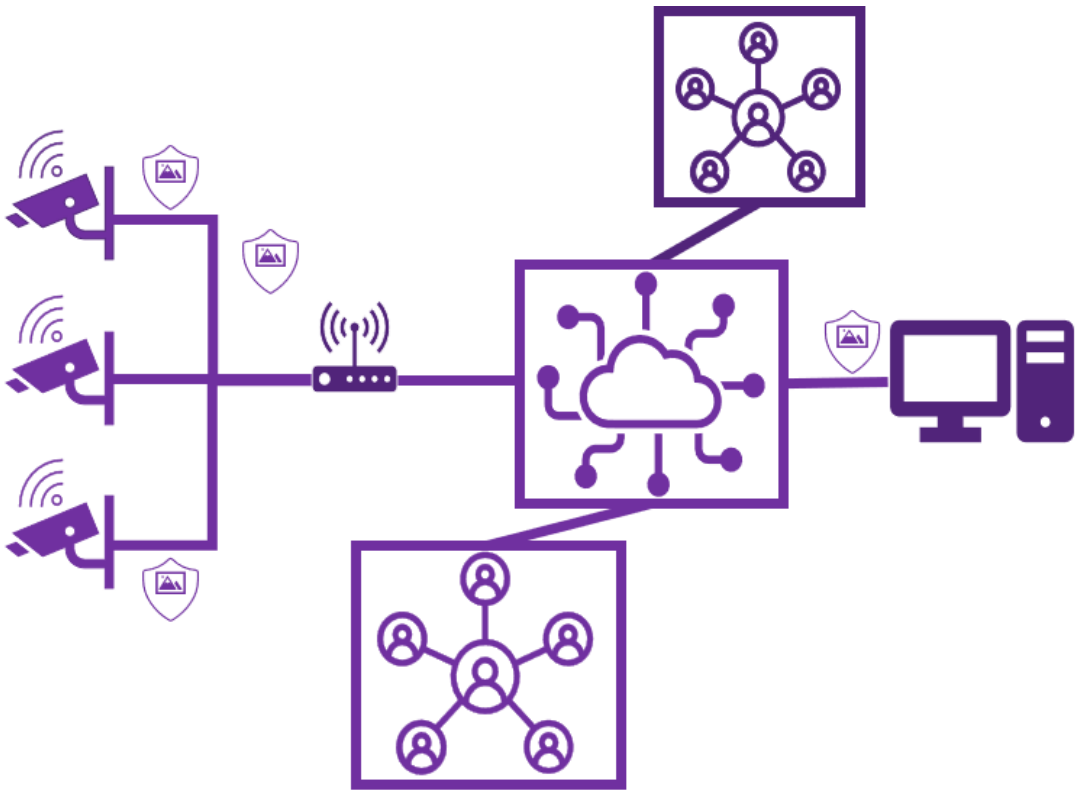

Figure 1.1: IP camera architecture

that degrade the image, many of them are unique to the camera. Such unique noise, assuming we can identify the noise and extract it for fingerprinting, can serve as the basis for passive security. In this work, we discuss the fingerprinting techniques performed to achieve remote identification. Such technique of fingerprinting noise and identifying its source is coined Hardwaremetry [8, 9].

Hardwaremetry and passive identification techniques have been active research fields in digital-cyber forensics [10]. Such techniques rely on extracting noise characteristics from hardware devices, fingerprinting them, and identifying their source. While the idea of fingerprint and identification is simple, the process of extraction and execution is not. To start, choosing and identifying a unique noise is challenging since the noise must fit different criteria. Furthermore, extracting it is also non-trivial. Additionally, many obstacles hinder the identification process. Biases can be introduced at each stage of the remote identification process resulting. Such biases can result in identification inaccuracies and scalability issues.

This said, in this thesis, we present a remote camera identification solution. This solution attempts to extract a unique dominant image noise for different cameras. Furthermore the work builds an AI-based scheme that aims to identify cameras' fingerprints. Further, our aim is to deliver a novel end-to-end solution for passive security via remote camera identification. With that said, such a solution must be automated and scalable. For that, we use open source software development kits (SDKs) and hardware to automate with entire solution. Moreover, we aim to identify and remove any biases that hinder performance and accuracy. The thesis covers and justified some optimization decisions done for efficiency and scalability.

The following sections make up the publication. Section 1.2 covers the background of the work. The background discusses the entire section before diving into the implementation and design. Section 6.1 covers the implementation of the solution before chapter 7, evaluation. Chapter 2 and 8 , include the discussion and related works. The paper concludes in section 9.

1.1.2 Overview

With the explosion of connectivity, cameras, and privacy concerns, camera security is essential. Many industries use cameras as a vital part of their technology, yet, cameras, especially, internet-connected ones, are insecure [11]. Like other IoT devices, cameras are commodity devices manufactured in bulk with little security consideration. Such security is not ideal, especially with industries such as remote inspection and surveillance. Often such industries have to rely on special industry-grade cameras for security. But, such techniques need heavy hardware and software overhead and come with different sets of challenges. Challenges such as key storage, root-of-trust, vulnerability patching, and physical access can be hard to solve. Such challenges, if unmitigated, can expose the cameras to different threat vectors. This is since vulnerabilities in the hardware and software may lead to information disclosure via private key leakage.

For such reasons, there has been active security research into passive security consideration as a practical means for camera security. Such techniques bypass active authentication and integrity measure by using passive techniques to identify and authenticate cameras. Active security is the means of securing the hardware, software of the camera along with providing strong authentication and encryption mechanism for secure image transmission and remote authentication. Passive techniques assume no camera side active security. The technique relies on exploiting unique features produced by cameras and persist in the final image or feed of a camera to map images to cameras. Such techniques can distinguish and identify cameras by fingerprinting such unique features to provide authentication by identification. With such passive identification, no active security assumptions are made on the camera. While passive techniques are unable to provide encryption and information protection, they provide a certain level of feed assurance that is necessary for authentication, integrity, and repudiation.

Remote camera identification is the process of providing remote feed assurance by distinguishing image sources passively to provide authentication via identification [7]. To start, we need to find a camera unique feature that is found in the final image that can be fingerprinted for identification. Following the identification, we need a way to fingerprint it so that an image coming from a camera can be identification [7]. Fortunately, cameras rely on camera sensors for image acquisition. While there are many types of camera sensors, the two dominant ones are CCD and CMOS sensor [12]. Both sensors have a dominant noise pattern that can be identified and efficiently extracted or fingerprinted from images. Such noise patterns are called photoresponse nonuniformity, they are formed due to the manufacturing imperfections of the silicons wafers that capture light. Following PRNU identification, many fingerprint techniques are present. In fact, there is a field named hardwaremetry for fingerprinting and identification. Hardwaremetry is the study of fingerprint sensors based on their intrinsic features

[8, 9].

Hardwaremetry is an active area of research in cyber-forensic science, over the years, the research has investigated many cameras fingerprinting techniques [10]. The fingerprinting techniques are not limited to the camera sensor, other research has attempted to exploit features such as dust and lens shape and distortion to extract unique camera features [10]. Moreover, fingerprint methods range from statistical and mathematical models to using machine learning and deep models [10]. Yet, while there have been many valid proposed techniques, the most prominent feature used for fingerprint in dominant noise left by the camera at acquisition and processing steps. The technique relies on extracting or identifying a unique intrinsic noise pattern and attempting to fingerprint it using statistical and mathematical models to establish an identification reference. Moreover, another prominent method of fingerprinting that yields good results are using machine learning, especially deep models, Convolutional Neural Network.

Yet, while those different methods succeed, they are most inefficient and not automated for production. Moreover, such techniques are often riddled with biases that can hinder the identification accuracy in different ways. Biases such as image content while fingerprint images and other noise introduced in the acquisition, processing, storage, and transmission process can overwrite or interfere with the noise needed for extraction thus degrading identification Process step that includes compression and feature enhancement can introduce new unintended noises that influence the identification process.

With that said, we proposed an end-to-end solution for camera identification that relies on the combination of efficient well-known noise extraction and fingerprinting techniques to deliver a scalable solution for passive remote camera identification. The solution achieves scalability by combining noise extraction and fingerprinting in a novel way that removes many biases. The model collects a data set of raw unprocessed images coming from different sources to build a new noise dataset made up of the extracted PRNU noise. Following the completion of the noise data set, it is used to train a CNN to fingerprint images. We identify and attempt to remove biases along the way. Once training finishes, the model can identify. Moreover, we optimize the CNN model's architecture using Particle Spawn Optimization to achieve better identification accuracy and performance.

How do we verify images without a digital signature?

Cameras, like everything, are not perfect, there are many errors or noise patterns that are unique to a camera and its camera sensor or the type of camera. Such noise that can degrade an image can serve as an important artifact for image identification and verification. Similar to the idea of "bullet scratches" to identify the gun used, camera noise can be used to fingerprint images can identify their source, hence their authenticity and integrity with no camera security and hardware assumptions. Fortunately, there can be noise that is unique to cameras like bullet scratches, such noise can be extracted and used in an approximation or learning model as a reference to identify images coming from the same camera. Moreover, such identification relies on a model-driven or data-driven model for fingerprinting.

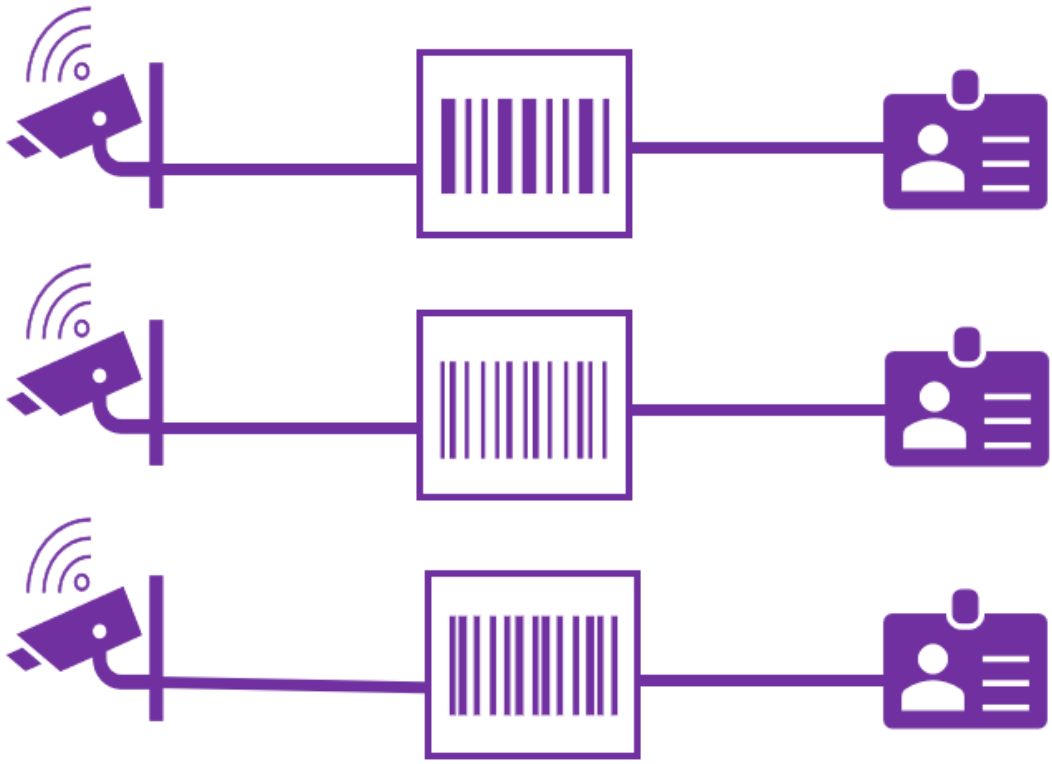

Figure 1.2: Authentication via identification

1.2 Background

1.2.1 Motivation

A chain is only as secure as its weakest link. The same holds true for any system. The problem with trusted feed has been a security problem for many industries that rely on cameras for their technology or service. The problem with cameras is, like any other IoT device, they are generally very insecure [3]. Due to the exposed environments, short time to market, and commoditization of cameras, hardware and software security considerations are generally an afterthought in production [13]. Yet, cameras, especially IP ones, are becoming more and more complex and ubiquitous [14]. Moreover, with the advent of the IoT and AI era, they are being integrated into industries to solve problems [15].

Such interconnection, complexity, and reliance introduce a new attack surface for industries. A chain is only as strong as its weakest chain [16] meaning a comprise the camera can compromise the entire technology stack. With that said, fortunately, there has been an active community of research focusing on camera security. The main two areas of research is active [14] and passive camera security techniques [7].

Active Security

To start with active security. Active security implies the active hardware, software, and encryption security consideration. It relies on continuous security software patching along with implementing replicable secure hardware. Also, it relies on industries standard encryption infrastructure. Such

Figure 1.3: Passive vs Active Security

security considerations can ensure confidentiality, integrity, and authentication via secure encryption.

While such active security is the ultimate security guarantee, it comes at a significant cost. To start such cameras might not fit the industry's technological constraints. For example, technologies such as computer vision may need special types of cameras that don't include such security. Moreover, often such cameras come with heavy hardware and software overhead. Such overhead can amplify if a technology relies on many cameras to scale. Such cost can be more than industries' comfort level, resorting them to rely on commodity cameras intrinsic security.

As mentioned, commodity cameras suffer from key security challenges that make them an insecure solution for trusted feed. One of the key issues is key storage. Due to weak software and hardware consideration, key storage and management are a problem for cameras. This is due to the fact that any compromise and physically exposing cameras can potentially lead to key leakage. Active security deals, with such weakness by implementing strong hardware and software capabilities. An example of such capabilities is secure secure hardware enclave and trust execution environments.

Passive Security

Given such heavy requirements for active security, passive security might be a solution for trusted feed. Passive techniques attempt to bypass active security requirements while ensuring some security properties. It achieves that by exploiting some intrinsic unique traces found in the image to fingerprint the camera. By fingerprinting the camera we can distinguish its source. By distinguishing its source we can provide camera attestation and authentication. While passive techniques are useful, they can only provide some security properties, they can but, ensure confidentiality and availability. Another important feature of passive techniques is that they are practical. They don't assume any camera requirements, hence they can be applied to almost any camera.

Chapter 2 covers the different passive techniques for camera identification. While such techniques are practical, they are proposed in the context of cyber-forensic not trusted feed. The cyber-forensic

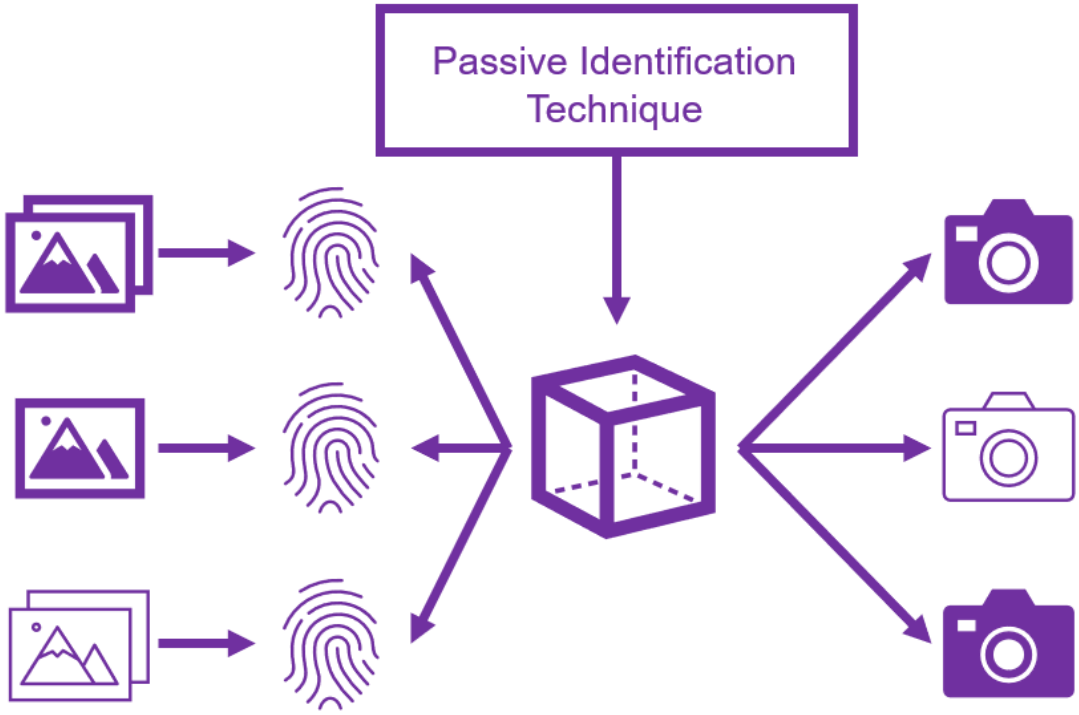

Figure 1.4: Passive Security

context for camera identification is the ability to bind an image to the suspect camera. In this work, we proposed a passive security techniques design around camera identification for a secure feed. We proposed an end-to-end automated solution for camera attestation and authentication. This solution utilizes camera identification as a passive identification technique.

1.2.2 Image Noise

Passive techniques rely on unique distinctive features produced by the camera and found in the final image for identification [7]. One feature that fits such a description is camera noise. Cameras produce noise at acquisition or processing to are present in the final image. Moreover, such noise often leaves a unique noise pattern that can be distinguished using proper techniques.

The advancement of modern cameras implies the addition of more image acquisition, storage, and enhancement processing stages. This means that there are different kinds of noise that can be exploited for different functions. In fact, in his survey [10] Bernacki et. al defined two types of noise types for passive camera identification based on the noise extracted. These are Individual Source Camera Identification (ISCI) and Source Camera Model Identification (SCMI). ISCI, based on the noise fingerprinted, aims to distinguish between individual camera sources. ISCI method aims to fingerprint noise that is unique to the camera itself rather than ones unique to the brand or model. On the other hand, SCMI aims to identify images by fingerprinting the noise that is common across different cameras of the same brand or model. Survey [10], summarizes the different techniques used to identify and fingerprint each type of identification.

As mentioned, this work will focus on remote camera identification of individual cameras, SCMI.

For SCMI, the work can be divided into two basic processes, fingerprinting and identification. Fingerprinting refers to the process of identifying necessary noise, while identification is the process of exploiting such noise to identify the camera using the noise found in the image. The identification process is done by analyzing the fingerprinted noise and making inferences. In most works covered in [10], the fingerprint was based on finding means to extract specific noise, and identification was based on exploiting such noise to make inference on its origin. For fingerprinting SMCI, most work revolves around a noise coined as Sensor Pattern Noise (SPN). SPN is a dominant extractable camera-unique noise pattern. SPN noise is introduced at acquisition time and made up of a combination of sub noises [17]. The most dominant noise from the SPN is noise caused by the imperfection in the silicon wafers used for image acquisition [18]. Such imperfections, caused by manufacturing errors [18], upon illumination leave a distinct error trace across different regions of the image [12, 19]. This noise is coined as photo response non-uniformity (PRNU) noise. ISCI fingerprint is mostly based on extracting such noise or designing an identification technique to automatically fingerprints use noise [10].

For identification, however, two techniques are most prominent [10]. These are statistical analyses using correlation and CNN. The statistical analysis aims at extracting fingerprinting images by extracting PRNU noise patterns to apply mathematical correlation to establish a reference pattern in which images are identified according to it [12, 17, 19]. On the other hand, CNN models aim to automatically identify PRNU images by feeding a large dataset on images containing such noise and designing the cost function according to such specification [8, 20–23]. Identification is CNN is the classification of images based on their different noise pattern.

1.2.3 Noise Extraction

Fingerprinting noise, as mentioned, can take many forms. In this work, we extract the PRNU noise pattern from images. The noise instead of the entire image is used for the identification process. For that, noise extraction is a considerable part of this work.

Since we are dealing with images, noise extraction can be achieved via denoising filters. Given a good PRNU denoising filter, the denoising filter can be used to generate a new image. This new image will have the PRNU noise pattern suppressed. Since now we have two images, the original image with the noise, and the denoise image, we can extract the noise by subtracting the image pixel from each other. This can be summarized in the following formula to obtain noise 1.1.

$$N = I - F(I) \tag{1.1}$$

The formula presented the noise, N, extracted from a given image I. $F(I)$ presented the denoising filter F as a function of the original image I.

1.2.4 Denoising Filters

Formula 1.1 establishes a mechanism for extraction, yet, a mechanism for denoising the specific noise pattern is needed. PRNU noise is the noise caused by inconsistencies in the silicon wafers that capture

light in the noise sensor [17]. Such imperfections cause dominant smoothness inconsistencies in the final image. This means that the denoising process is the process of smoothening the image surface to suppress such inconsistency. If we can suppress them, then we can denoise the image. With the successful denoising of the image, we can apply formula 1.1 with $F(I)$ being the denoised image the extract the noise and F being the denoised filter.

Yet, while PRNU noise causes evident image smoothness inconsistency that we can exploit to fingerprint images, there are other noises that have the same inconsistency effects. Moreover, while denoising the PRNU noise, other noise will be denoised too and part of the noise after extraction using formula 1.1.

While fingerprinting can still be performed in the presence of such noise as demonstrated in [24], such added noise can interfere with the image identification process causing biases and hindering accuracy and performance.

1.2.5 Camera Pipeline

Any modern digital camera today has a pipeline of image processing steps starting from acquisition to storage. Such pipelines enable feature and color enhancement and manipulation giving the user more flexibility and control over the final image. However, the same pipeline introduces different types of noise along its different stages. Extracting a specific noise among the different ones is impossible.

As mentioned, the noise extracted process target PRNU can have a much different noise pattern included. With that said and since extracting PRNU alone is impossible, and other noise can introduce bias to the identification process, the aim is to remove as many biases as possible.

1.2.6 Biases and Raw Images

As mentioned, biases caused by noise interference can hinder performance and identification accuracy. Yet, since much of the added noise is generated due to the camera pipeline, if we can bypass it we can remove many biases present in the final image. Moreover, since the PRNU noise is the noise generated at acquisition time by the camera sensor before image processing, we can ensure the PRNU is not affected if we bypass the other processing stages. With that said, the best means to bypass the processing stage is to capture raw images. Raw images are images that are taken directly from the camera sensor with no post-processing enhancement and modification.

Cameras usually store images in a compressed manner using JPEG and other types of lossy compression [24]. Such compression can modify the original image to a large extent which can interfere with the noise pattern present [24]. For that, we also need to store images uncompressed to guarantee no noise tampering.

Finally, the content of the image, even if it's a black field can interfere with the classification process of the CNN model. For that, we attempt to remove such bias by having the CNN train on only the noise instead of the image themselves.

1.2.7 Automation

Extracting raw images is not enough for our design introduced in chapter 6. In this thesis, we propose an end-to-end automated solution for passive identification. With that said, we need a camera that enables the capture and storage automation of raw uncompressed images. For this solution, we use identical Intel® RealSense™ Depth D455 Cameras for image capture, automation, and data collection. Intel® RealSense™ Cameras offer an open-source SDK available on GitHub [25] that enables the programmatical configuration of cameras. With such SDK we can configure that cameras to capture raw-16 bit unprocessed images and store them uncompressed. The SDK enables automation of the process and the integration of this part of the design with is (data collection) to other parts.

Chapter 2

Literature Review

This chapter covers the previous and related work done on remote camera identification. The chapter covers the proposed techniques and solutions related to ISCI. While most of the previous work has been in the context of forensic, their proof of concept forms the basis for our solution.

As mentioned, the two most studied techniques are using statistical modeling of PRNU noise or AI. If we are able to fingerprint them identify images using such techniques, we can ensure camera identification. In his survey, Bernacki et. al [10] cover more means for passive authentication. Yet, for the literature review, the main focus is on the technique involving PRNU and AI. This is because our solution is a combination of those two methods.

2.1 Introduction

The aim of this work is the achieve passive security using fingerprinting as a means for camera feed attestation. Our work differs from previous contexts. While most of the work focused on fingerprinting images for forensic purposes i.e. to bind an image to an actor, our work uses it to provide passive feed security [10]. Our work uses similar fingerprinting techniques used below, yet, our identification is to authentication its source.

Our solution can be divided into two parts, identification and fingerprinting. For identification, we use PRNU fingerprinting and for identification we use AI. Fingerprinting is the extraction of feature that is able to identify the camera and identification is verifying that such fingerprint belongs to a specific camera. With that said, the literature review focuses on remote camera identification using PRNU analysis and AI techniques.

2.2 Remote Camera Identification

Source Camera Model Identification or SCMI are techniques in which specific camera patterns and characteristic that are unique to a camera model are examined and used to identify. SCMI method can distinquish between two camera of different models or brands and fail the identify identifical cameras.

While less effective than their counterpart, ISCI, since they are less reliable in camera identification and can't be used for court scenario, they are still useful in some enviroment.

2.2.1 Analysis of photo response non uniformity (PRNU)

In his seminal work [12, 19], Lukas et. al demonstrated the possibility of passive remote camera identification and verification. The papers were able to fingerprint images by extracting noise from images. For identification, the paper, used the extracted fingerprint together with satistical correlation.

The technique is as follows:

1. PRNU extraction using formula 2.1
2. Esbalishing a PRNU reference pattern for each camera fingerprinted using statistical correlation
3. Identifying new images by comparing their PRNU pattern to the reference patterns establish
    - The noise pattern is the average of all the PRNU pattern extraction and analyzed
    - Each camera must have its unique reference pattern
4. Using formula, the closest reference pattern implies the image source

By fingerprint images using PRNU and using statistical analysis, identification was successful. The evaluation of this method demonstrated the ability to identify camera sources with a high degree of reliability.

Moreover, the paper attempts to justify its choice of the following question.

Why PRNU noise?

The authors explain the use of PRNU by outlining the distinct feature that makes it perfect for fingerprint. To start, the authors claim that PRNU noise is relatively stable over the camera's lifespan, making it a persistent and viable solution for a long period of time. Moreover, according to the paper, PRNU noise is unique and dominant across images is not affected by frame averaging. Finally, PRNU noise can be transformed into a high spatial frequency signal that can be easily extracted using the appropriate denoising filter [17].

How was extraction performed?

Formula 1 is the formula for noise extraction. The formula is a simple subtraction expression where the noise is the difference between the original image and the denoise one.

$$N = I - F(I) \tag{2.1}$$

Figure 2.1 present the noise extraction formula. I symbolized the original image subtracted from $F(I)$, the denoised image. This subtraction yields the PRNU noise given the appropriate denoising filter.

Why statistical correlation for identification?

Correlation gave the best accuracy results. The paper discusses different methods for calculating the reference patterns. Those methods include median filtering analysis and flat-fielding. Following the comparison of the result obtained from the different identification techniques, statistical Correlation was the most accurate in identifying the image's source.

$$\rho_c(Y) = corr(Y - F_\sigma(Y), P_c) = \frac{(Y - F_c(Y) - E[Y - F_c(Y)]) \cdot (P_c - E[P_c])}{\|Y - F_c(Y) - E[Y - F_c(Y)]\| \|P_c - E[P_c]\|} \tag{2.2}$$

The equation for identification via correlation is demonstrated in 2.2 taken from [12, 19]. The equation denotes that P_c as the noise reference and $\rho_c(Y)$ as the identification process in of image Y. Y is the spatial representation of the image and $F_c(Y)$ is the wavelet denoised version of Y making $Y - F_c(Y)$ the noise extracted from Y. Let $E[Y - F_c(Y)]$ be the mean of the noise extraction from all the images from the same camera. With the said the formula demonstrated is simply the correlation, ρ_c, between an image noise $Y - F_c(Y)$ and P_c.

The results show that the experiment was successful in getting a high degree of accuracy for classification, around 90%. However, such a technique has some drawbacks with is that noise extraction can time up to 1 minute. This is due to the denoising transformation used to denoise image I, $F(I)$.

Following this work, Goljan et. al [26], attempted similar pattern noise extraction. However, instead of using correlation to identify the camera, cross-correlation analysis and peak-to-correlation energy ratio (PCE) were used. The experiments also proved successful, however, with slight accuracy improvement. However, a considerable improvement in the accuracy came in 2012 with the use of the circular correlation norm to identify cameras.

2.2.2 AI Techniques for Remote Camera Identification

Machine Learning

In the machine learning doman, there has been many work that relies on it for camera identification. [27] Moreover, beyond correlation and PCE, PRNU for identification was also examined in other works that relied on different identification mechanisms. Baar et. al [28] introduce a database grouping concepts for images so that images coming from the same camera can be automatically identified and grouped in a database. The work examined different approximation methods for grouping images based on origin and used the K-means algorithm to achieve such grouping. The technique used the K-means algorithm to automate the process, replacing correlation in the process and using MATLAB for more efficient denoising and clustering.

Beyond ISCI, in 2015, an Support Vector Machine (SVM) classifier was used for SCMI. The SVM was used for camera identification for model-attribution [29].

Deep Models

Camera identification has taken a great leap with the improvements of deep methods [20]. With the improvements and optimization of machine learning approaches, camera identification can be

considered a deep learning problem.

In 2016, Baroffio et.al proposed a CNN deep model for camera identification for both individual device-attribution and model-attribution [30]. In the work, the author postulates that deep models are able to extract such noise features automatically and learn to distinguish between cameras by training and reducing the cost function. The paper goes through the architecture of the convolutional models and explains each step and layer. The model was applied on the well-known Dresden data set [31] in which two experiments were carried out. The first is on individual camera identification and the second is for model identification. The work use training and testing accuracy to prove the results. The results produce a high degree of accuracy for such identification in both model and device verification.

After Baroffio et.al, the same concept was applied using a different deep model. In 2017, Chen et. al [20] investigated the use on residual Neural Networks (ResNet) in three concepts for camera identification, brand-attribution, model-attribution, and device-attribution. This is an important distinction since it allows the identification of a mobile's brand, model, and sensor. The work was based on ResNet deep model implemented in the identification. The data set, composed of more the 15,000 images captured from 13 different phones, was taken from the Dresden Database [31]. The Deep model was implemented on the data set to train, classify, and identify 13 different phones and brands of phones. The author claim that ResNet, for such classification, produced a much more effective approach to image forensics than [30]. This is due to been accuracy results than [30], moreover, it outperformed well-known CNN models such as AlexNet [32] and GoogLeNet [33] in terms of accuracy. Following this work, another technique for mobile source camera identification was proposed by [23]. Both methodologies are similar, however, the latter work relied on a different dataset, mainly MICHE-I Dataset [34]. Moreover, it implemented its own custom CNN architecture model for the learning and testing and images by tuning the hyperparameters. The result also claims a higher accuracy than [30].

Based on the power of CNN to automatically learn on classify image sources, Tuama et. al, [21] add a high pass filter is applied to the input image to optimize learning via noise extraction. However, while this work uses noise extraction layers, the filter used is to reduce the image content rather than using the noise in the training [35]. While this work includes feature extraction filters and a deep model, it is significantly different than the work done in the thesis. To start the work doesn't acknowledge the impact of compression on such identification. Unlike the work included in this thesis, the noise is not fed to the model, instead, the noise is merely used the reduce the impact of image content on learning. Moreover, the thesis primary work relies on camera identification using PRNU noise which is not the case in the paper. After hypertuning the CNN model and examining different layers, the model proposed in the paper produced a high degree of accuracy that is comparable with mentioned deep models [20, 23, 30]. The final model which includes 3 convolutional layer and max pooing achieve approximately 98% accuracy.

Chapter 3

Data Collection

3.1 Introduction

As mentioned, camera identification starts with fingerprinting noise found in the final image. Identification is fingerprinting enough images to establish some kind of reference pattern. After establishing a reference, identification occurs by comparing an image's noise to it. In our case, we use CNNs to do this comparison in the form of classification. With that said, the CNNs network usually needs a considerable volume of data to achieve acceptable results. This is due to the nature of CNNs networks which include the ability to learn with each new image, each new image improves learning.

Yet, while the size of data is important, data quality plays a large part in the accuracy and performance of the model. It also dictates the size of data required for accurate classification. As mentioned, one of the factors of quality is the number of biases and variances in the data [36]. Such factors degrade the data and hinder identification. This is because, from a machine learning perspective at least, such factors are considered image features [37]. Machine learning models can't tell the difference between noise features and other features of the image, thus with too many biases, the identification can result in inaccuracies. Such biases can result in identification accuracy and performance and need a large data set. In this section, we examine the different biases we can remove and cover the data collection process.

3.1.1 Biases

The advancement of modern cameras made taking real-world resembling images much more common. Such advancement implies more internal processing stages that modify and enhance the image for the human eye. Yet, the same enhancement introduces new errors and biases to the final image that can hinder identification. Issues such as the scalability and efficiency of passive identification techniques usually stem from poor identification performance. Hence, if we can improve performance by removing such biases we can put in place a much more efficient. In this subsection, we describe the different biases found in the final image along with the ones we managed to remove to enable a more effective identification.

Biases play a big role in the identification process, the simple solution is to remove them. Yet, such solutions might not be easy or possible in some cases [37]. This is because some biases are hard to remove without degrading the entire image. Moreover, some biases are rooted in the image formation and acquisition process. While biases left at image acquisition caused by the image sensors or lens distortion are unavoidable, we can remove others. Image acquisition noise includes PRNU, and CFA interpolation, and demosaicing [38]. Stages that involve image enhancement and storage are further down the pipeline. Such stages include color correction, tone mapping, compression which introduce their own errors. If we can safely remove such stages we can remove a significant amount of bias. Raw images are the version of the final image that bypasses image enhance and unnecessary stages. If we can extract raw images from the camera without affecting the PRNU noise we can remove many biases.

PRNU is a bias, but, it is one that we want evident in the final image for fingerprinting and identification. Since the camera sensor, which handles image acquisition, generates the PRNU noise, we can use raw images to bypass other biases.

Moreover, images after acquisition and processing are compressed and stored using a compression algorithm. Such compression algorithms can inference with PRNU noise. For that, we need an image acquisition process that extracts raw images and saves them in an uncompressed manner.

3.1.2 Raw Image Acquisition and Automation

For raw image acquisition and data collection, we utilize Intel® RealSense™ cross-platform SDK [25] to configure and automate the image acquisition process. The SDK is an open-source framework for Intel® RealSense™ [25] cameras that allows manipulation and automation of the image acquisition and storage pipeline. With that said, a script is developed for the image acquisition process that connects to each camera, configures the image stream to 16-bit raw unprocessed each 30 seconds frame, and stores them as raw data. Raw images are single-channel intensity images produced directly by the image sensor and passed through and CFA to produce a single dimension image with 16-bits of data [39]. In this case, since we are using 16-bit raw images, each image is stored as an array.

3.1.3 Data Collection

Using the camera and the framework, we captured over 4,000 1280 x 800 16-bit minimally processed raw image streams from each camera and stored them as arrays. The process of acquisition and storage is automated and integrated into the system as the first stage of the pipeline.

Chapter 4

Noise Extraction

This section introduces noise identify and used for identification. Moreover, it covers the mechanism in which noise identification and extraction occurs. The section goes over the history and evaluation of the technique used to build to second data set along with the extraction mechanism.

4.1 Introduction

The second stage of the pipeline is concerned with utilizing a PRNU noise extraction mechanism to denoise the image data set and form a new noise data set. Nothing is perfect, like everything in the world, images are not perfect. Due to many factors, events contaminate images with different kinds of acquisition, transmission, compression, and storage noise. Yet, while noises are inevitable, over the years, there have been many successful denoising techniques. Techniques of which can extract much noise [40]. As mention, for this work, PRNU is the main noise of interest. For PRNU noise extraction, the experiment relies on the same techniques used in prior well-known work [17, 19, 24]. In this section, we examine the noise extraction technique and build the noise data set.

The goal is to extract noise that can serve as a feature pattern for identification. PRNU noise, as mentioned, is a unique noise pattern that can distinguish different cameras. By extracting PRNU noise and training a CNN model, we can perform camera identification.

For building the new noise data set the main problem is being able to extract the specific noise pattern required. If we are able to find a filter that is able to suppress PRNU noise from images, we can use formula (1) to extract PRNU noise. The formula works by subtracting the denoised image from the original noisy image to end up with the noise.

4.2 PRNU Noise

PRNU noise a noise pattern trace present in the final image. This noise is the main noise used for fingerprinting and identification due to its following unique properties:

- **PRNU is the not reduced by frame averaging** Frame averaging is a technique used by the digital camera processor to eliminate electronic noise, since PRNU is an acquisition noise generated by the silicon wafers, it is not affect by such a process.

- **Dominant noise present in the final image** The more dominant an noise is in the final image, the easier and faster the identification process. Since PRNU is a dominant noise pattern, its extraction and fingerprint is easier.

- **Unique cause by light refraction** PRNU noise is unique to each camera and image. Only of the most important characteristic of an fingerprint is it uniqueness. Since PRNU noise is manufacturing imperfect of individual cameras, we can use it for ISCI identification. Moreover, since it is impossible for two image to have the same light intensity, and PRNU is the result of such illumination, PRNU noise is not fixed. It forms a pattern of identification with makes it perfect machine learning problem.

4.3 Dataset

The second stage of the pipeline is concerned with utilizing a PRNU noise extraction mechanism to denoise the image data set and form a new noise data set. Nothing is perfect, like everything in the world, images are not perfect. Due to many factors, events contaminate images with different kinds of acquisition, transmission, compression, and storage noise. Yet, while noises are inevitable, over the years, there have been many successful denoising techniques. Techniques of which can extract much noise [40]. As mention, for this work, PRNU is the main noise of interest. For PRNU noise extraction, the experiment relies on the same techniques used in prior well-known work [17, 19, 24]. In this section, we examine the noise extraction technique and build the noise data set.

The goal is to extract noise that can serve as a feature pattern for identification. PRNU noise, as mentioned, is a unique noise pattern that can distinguish different cameras. By extracting PRNU noise and training a CNN model, we can perform camera identification.

For building the new noise data set the main problem is being able to extract the specific noise pattern required. If we are able to find a filter that is able to suppress PRNU noise from images, we can use formula (1) to extract PRNU noise. The formula works by subtracting the denoised image from the original noisy image to end up with the noise.

4.4 Wavelet Transformation

As mentioned, the goal is to extract the specific unique noise pattern, PRNU. Image denoise is still an active research area and a fundamental problem in digital signal processing (DSP). The extraction of single noise is impossible. Yet, since PRNU noise appears in the final image as a smoothness inconsistency, smooth noising removal methods can work [1]. Denoising images by smoothing

methods [1] extracts PRNU noise along with other noise. But, since raw image avoids many other processing noises, PRNU is the dominant noise that remains.

Spatial-Frequency Filtering is one of the popular filters used for smoothness denoising. The filtering method transforms the digital image into a frequency signal. Following the transformation into the frequency domain, a function extracts the noise frequency. For stationary and periodic signals, the Fast Fourier Transform (FFT) enables effective extraction. Given a stationary signal and a cut-off frequency, FFT smoothens the signal by removing frequency size above the threshold.

$$f(\omega) = \int_{-\infty}^{+\infty} f(t)e^{-iwt}dt, \tag{4.1}$$

FT is illustrated in 4.1, were an image in transfered into a non stationary image signal. Following transformations are built on-top of FT.

While FFT does a good job for periodic signals, images transformed into the frequency domain are nonperiodic. Images are complex frequencies with varying magnitudes and need to be model as a function of time. Since FFT cannot achieve good time resolution, the Short-time Fourier transform (STFT).

$$f(\tau, \omega) = \int_{-\infty}^{+\infty} f(t)w(t-\tau)e^{-iwt}dt, \tag{4.2}$$

The above formula 4.2 illustrates STFT. The formula is based on FT with the addition of a moving window function w. τ presents the time localizing translation parameter, the translation of the function across the signal.

STFT came as a solution to handle the non-periodic signals, especially images signals. STFT attempts to divide the non-stationary signals into discrete stationary ones. Given a window function of fixed length w, w divides the signal into the small stationary signal. After obtaining a set of stationary signals that make up the image signal, an FFT transformation on each signal can occur. The shortening of the time resolution in small windows W results in a much better frequency resolution. This flow from the inescapable law of physics [41] which states:

> We cannot know what frequency exists at what time instance, but we can know what frequency bands exist at what time intervals.

The law states the higher the time resolution, the greater the frequency uncertainty. For high-frequency resolution, short time windows yield less uncertainty and vice versa. The law is presented in the following equation.

$$\Delta t \Delta f \geq \frac{1}{4\pi} \tag{4.3}$$

Δt and Δf are reciprocal and bounded by 4π.

While STFT outperforms FFT, it still has some major limitations that influence certainty and noise extraction. One major drawback is the fixed length of the window function. A fixed window cannot distinguish between high-frequency areas and low ones. High-frequency areas need shortened time

frames and low-frequency areas need a widened time frame according to formula 4.3. Thus, a fixed windows function results in degraded noise suppression and smoothness.

We cannot use property 4.3, but, we can improve on it. If we can have a scalable window function, optimal noise suppression is possible. We can guarantee optimal noise suppression if we have an adjustable window function that can shorten and widen depending on the frequency-time resolution. With that said, the Wavelet Transform introduced the concept of wavelets. Wavelets are adjustable time frames that yield an optimal frequency-time resolution.

$$W\{f(\tau,s)\} = \int_{-\infty}^{+\infty} f(t)\frac{1}{s}\psi(\frac{t-\tau}{s})dt, \tag{4.4}$$

The above equation 4.4 represents the Wavelet Transform. Instead of the window function of equation 4.2, we have a complex conjugate known as the wavelet ψ. The wavelet represent a modified window function that is scalable depending on the frequency region on the signal represented by s. s is the inverse of frequency and determine the window in time.

Amongst the mentioned techniques, the Wavelet Transform is the most effective at PRNU extraction. Since WT can ensure a high degree of frequency-time resolution, better PRNU denoising is possible.

4.5 Extraction

Given the data set made up of raw unprocessed data, we can extract the PRNU image noise to form a new noise data set. The first step in the noise data set process is denoising each image. Following that, noise extraction occurs by applying equation 2.1. For each image in the original data set, noise extract is the difference between its pixel values and its denoised version. WT denoises the image, and noise is then extracted.

4.6 Noise Data Set

We use the noise data set for training for the following reason. Plotting the new noise data set for visual representation results in a homogeneous field color. Such consistent color across all the different images taken removes image content bias. Since there is no image content in the noise image arrays, there is no bias introduced by image content features. Performance and accuracy improve with the reduction of biases.

Chapter 5

Identification

This section introduces Neural Networks and gives an overview of their most relevant topics and background. The section covers the idea covers in the development of the identification models and well. The following sections introduce the model's identification scheme using CNN. The Chapter also covers the preprocessing, training, and optimization step used.

5.1 Introduction

The process of remote camera identification is two-fold fingerprinting and identification. As mentioned, the fingerprint process is based on the PRNU noise extraction with WT. For the identification process, however, a deep model is developed to recognize different cameras' fingerprints. As bio-metric fingerprints, identifying them and binding them to an individual is not trivial. Often many factors influence the fingerprint such as light, position, and time. The same is true for cameras, we cannot identify a camera from one fingerprint, rather we need to rely on a noise pattern inside. Only by learning the fingerprint features can one accurately identify the image source. Learning such patterns and features is uniquely suited for deep models.

5.2 Background

Deep models are the term coin of a specific domain of AI-related to machine learning. It refers to a supervised learning model made up of multi-layer perceptron [42]. Such algorithms are a powerful tool for learning specific learning, such as fingerprints, a classifying them based on their annotation. A good example of such classification is a classifier that learns dogs' and cats' features and classifies them according. The same process can be mimicked for the noise extracted for the images. The model will then learn the different fingerprint features associated with each camera to be able to identify its images in the future.

5.2.1 Neurons

The term neurons in a neural network is a term inspired by the brain. Like the brain's neurons, neural network's neural are the fundamental building blocks of a model. Neurons are nothing more than organized entities that hold a specific piece of the puzzle's information. Typically they hold a number between 1 and 0.

5.2.2 Layers

Neurons are only effective when they are grouped and work together to represent an idea. Such grouping of neurons forms a level of neurons. A neural network is a combination of different layers containing different numbers of neural. An example of a layer of neurons is an input layer. Given an image of 100 x 100 dimensions, the input layer might be made up of 1000 neurons each representing an individual pixel in the image. Neurons are adjusted in the learning processing. The value of each neuron is called its activation. For the input layer of an image, the pixel corresponds to the activation of each neuron.

Layers form each stage of the neural network, from the input layer to the output one. The output layer in the case of the neural network represents the final classification output. Consider a neural network classifier that can distinguish between numerical digits. The output layer in this case will be the grouping of 10 neurons each presenting the activation of one digit from 0 to 9. With enough training, the aim is to get the model to generate the correct output for a given input. For this example, given an image of the digit 9, the model output layer would have the highest activation of the 9th neuron.

Hidden layers compose the inner layer of a neural network. The number of layers along with the neuron they hold vary with each model. Generally, each layer aims to produce the specialized defined output.

5.2.3 Learning

A neural network is a sequential stack of layers made up of many neurons that learn by example. Neurons are presented as activation functions. Activation in one layer determines the activation in other layers. This is done via the neurons themselves. Each neuron is connected to all or most of the neurons in the previous layers. With that said, the heart of the network is the mechanism in which such activation influence one another in the learning process to the output layer. The hope is to design a model that with me able to adjust its activation along with the network the learn specific features that make up any pattern. Once able to pick up on features and pattern the neural network outputs that in the output layer as an activation.

The question at hand is what is the mechanism that manipulates such activation to learn patterns? What parameter should the model have to express such patterns and features and learn them? To start, the first parameters in the model are related to neurons. For each neuron in the model, the activation number it holds is the result of the following parameters. The activation number of neurons in the

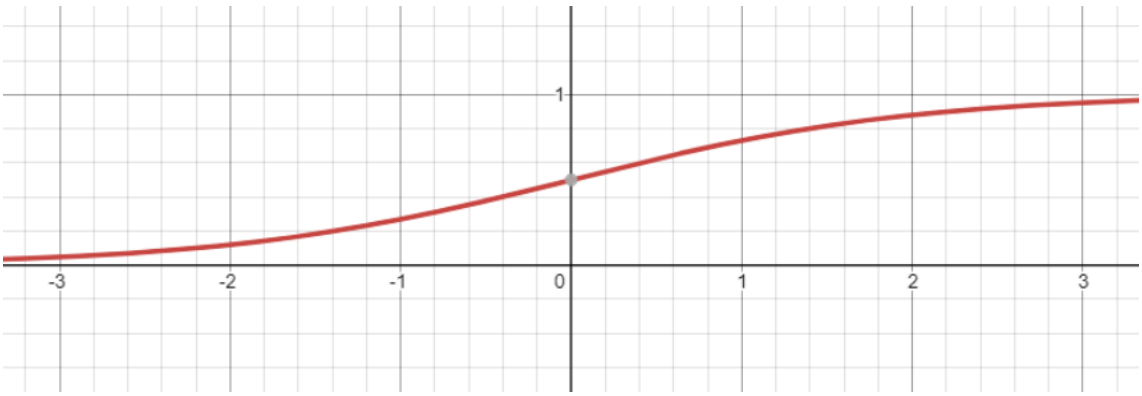

Figure 5.1: Sigmoid Function

previous layers the connected to a specific neuron. a_j^l presents the symbol for the activation of neuron j in layer l. The weights of the connection between each neuron of the previous neuron. The connections between each neuron are also presented as weight which is the strength of the connection. w_{ij}^{l-1} presents the connection between the ith neuron in layer $l-1$ with the ith neuron in layer l. Moreover, there is an additional parameter in the form of bias where the results are skewed to a certain threshold. The is called a bias, bias b_j^l represent the bias of neuron j in layer l. Learning is only finding a valid combination of these parameters in each neuron to solve a certain pattern problem.

$$a^1 = \sum_{n}^{i=1} a_i^{l-1} * w_{ij}^{l-1} + b_j^l \tag{5.1}$$

With that said, the activation function of one neuron can be demonstrated in equation 5.1. a^1, the activation value of one neuron, is the summation of all the activation values, a_i^{l-1}, multiplied by each corresponding weight of a_i^{l-1} the previous layer. Also, bias is presented as b_j^l.

5.2.4 Activation functions

We compute the activation of a neuron, we might end up with a number. Since we want our output layer to present a number from 0 to 1, we can use an activation function to limit the range of output.

Sigmoid

A common function that compresses the activation value into this range is called the Sigmoid function. The Sigmoid function is used in our classification model.

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{5.2}$$

The above formula 5.2 present the Sigmund function. The function is plot in graph 5.2. The graph shows that negative values tend toward 0 and positive values tend toward 1 give a range of 0 to 1.

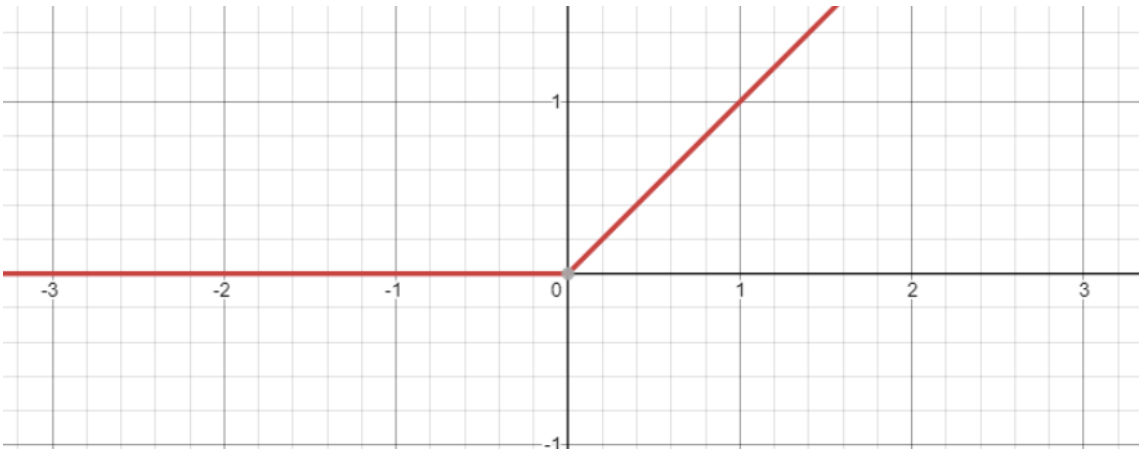

Figure 5.2: ReLU Function

Rectified Linear Unit (ReLU)

A modern approach for activation ranging is optimized for learning the ReLU function. The real function is also we for our training and is presented in the following function 5.3.

$$ReLU = max(0, x) \tag{5.3}$$

ReLU function is very simple, the activation value is limited from 0 to infinity by taking the max between 0 and the activation value. This will convert all negative numbers into 0. The graph 5.3 show the RelU function plotted.

5.2.5 Cost Function

As mentioned, each neuron in any neural network is connected to almost all the neurons of the previous layer yielding the activation of the neuron. A neural network initially has a random assignment to the weights corresponding to each neuron. Adjusting such weights affect the activation and the neuron, thus the learning process. The adjustment of such weights for learning is done with the help of a cost function. A cost function loss values between the output and the desired output; the squared difference between them. The desired output is determined by the label of the training sample presented as a neuron in the output layer. With each training sample, the cost function is calculated.

In simple terms, the cost function is a function with all the weight of all the networks (optionally the biases) as input. It outputs the cost of training one sample of input. The higher the cost function, the more incompetent the network.

5.2.6 Gradient Descent

With that said, the main goal of learning is to have a minimum cost function as much as possible. The loss the cost of training the higher the accuracy of the model. Given the cost function, we need a way to minimize it but adjust the weight and biases of the network so that the cost function is a small as possible. Since the cost function is a function, we can think of it in terms of calculus. We can minimize

the loss of the cost function by adjusting the parameter using the function gradient. The gradient of a function gives the path of greatest descent to reach a local minimum. Simply the gradient tells how to adjust the weight to reach the minimum of the function (cost function) most quickly. This idea is known as back-propagation using gradient descent.

With that said, a neural network is merely a combination of layers made up of neurons that have random weights initially. Based on the weight, the cost function is calculated and adjusted implicitly with each training sample using gradient descent. Each training sample help reduce the cost function and improve learning.

5.3 Convolutional Neural Networks CNNs

ConvNets or CNNs are special types of neural networks that have gained wide interest. CNNs, like all neural network, is made up of neurons and stacked sequential layers that are tuned using backpropagation and a cost function. Unlike other neural networks, however, CNNs have the following criteria in which they mainly consist of convolutional layers, a non-linear activation function, thresholding, local pooling, and intensity normalization. Such characteristic enables CNNs to achieve the best result between AI algorithms for problems relating to computer vision. Problems such as image classification, object detection, and segmentation have been a specialty for CNNs.

Since we are working with camera identification, the use of CNN made sense for trying to classify and distinguish noise patterns for identification. However, unlike previous work that involved CNN, this work attempts to train only on the noise that that on the entire image content. Using formula 1.1, we extract a new data set made up of only the noise of each image. Using CNN, such noise can also be treated as a computer vision problem since the result of the extraction is a noise array with the same dimensions as the images themselves.

5.3.1 Convolutional layers

Convolutional layers are similar to any neural network layer, however, they operate differently. Convolutional layers have some unique characteristic that makes them optimized for pixel inputs [23, 35, 43].

To start, unlike the weight and biases mentioned prior of neurons in neural network layers, the convolution layer employs a convolution filter that is the sum of the weights of all the neural connections of a layer to produce a feature map [43].

5.3.2 Fully connected (Dense layers)

As mentioned, almost all neurons can connect from the previous layers. However, they might not all be connected in which each neuron of a layer is connected to each neuron and the next layer. Fully connected layers are layers where all the neurons of the last layer are connected to all the neurons of the next layer, hence fully connected.

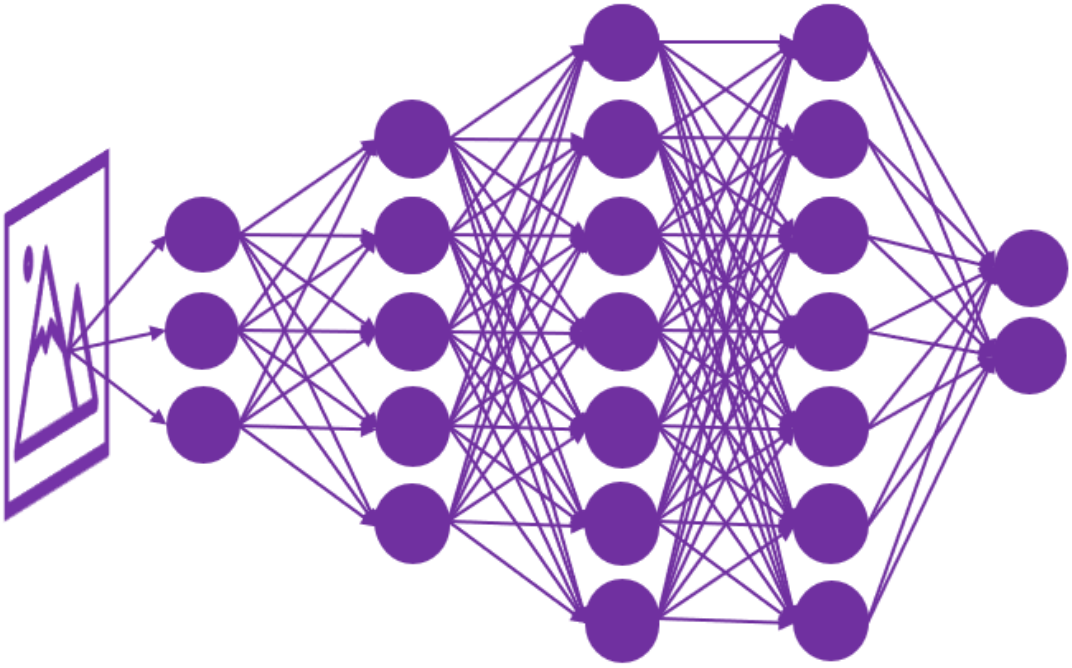

Figure 5.3: CNN Architecture Example

This is useful, especially for computer vision problems where the model is required to learn complex image features. While it increases computational complexity, it leads to more efficient loss and learns optimization.

5.4 Camera Identification

Following the denoising and the noise extraction process, we need an efficient CNN model for learning against the noise data set. For that, this section covers and justifies the model and its architecture. Yet, before any training can take place, preparing the noise data set for training is necessary. This section covers all necessary steps required for identification. The section starts by covering the preprocessing step before describing the model and the training process. Following that, the section covers model optimization before evaluating and testing the results.

5.4.1 Preprocessing

The preprocessing stage is an essential step in the identification process. The first step is data annotation. We annotated and shuffle the data for binary CNN classification. Before training the model the noise data set must for formated and spit in training and testing sets. The training set if necessary for learning PRNU patterns found in the noise data set. The training data set forms 70% of the whole noise data set. While the evaluation of the model uses the testing data set. After the data annotation, shuffling, and segregation, the model is ready for training.

5.4.2 Training

For training, we train a simple CNN model derived from the documentation on the noise data. We derive the CNN model from simple binary cats and dogs classification built by [44] which can produce a 90%+ testing accuracy. We chose this model due to its simplicity that can aid identification performance.

The model is a linear stack of layers made of three components, an input layer, hidden layers, and an output layer. The input layer accepts the 1280 x 800 16-bit noise pixels as input for each noise image array. The hidden layers are 3 layers with 2 convolutional layers and 1 fully connected dense layer. The final layer, the output layer, displays the probability results of the classification.

Further dive into each layer, the input layer consists of 1280 x 800 input nodes, each presenting a pixel value. The input layer depends on the dimension of the image, a cropped image yield a reduced number of nodes. The input layer connects to the first layer in the hidden layers. The first layer is, as mentioned, a convolutional layer that receives input from prior-layer neurons. A fixed number of neurons make up the layer, for the first model tested, we chose an arbitrary number of neurons, 256. This number is common and used in the simple dogs and cat classification [44].

Going more in-depth, 3 components make up a convolutional layer. The first component is the weight and biases associated with each neuron or node. For this, as mentioned 256 neurons make up the convolutional layer in the model. Neurons represent the different weights and biases of a convolutional layer required to yield a feature map. The second component is the activation function necessary to formalize the weights and biases of the layer to yield a feature map. The convolutional layers in the model used Rectified Linear Unit Function (ReLU) as the activation function. ReLU reduces the learning time by efficient gradient decent adjustments. The last component in the convolutional layer is an operation used to extract features, smooth and sharp. Pooling is the operation of downsampling the features to increase performance and extracting edges and distinct features. 2 adjacently connected convolutional layers make up 2 out of 3 layers in the hidden one both using 2-dimensional max pooling.

The final layer in the hidden layer is 64 nodes fully connected dense layer. Unlike convolutional layers, a dense layer is one in which each neuron connects to each one in the previous layer. While dense layer increase model complexity, it enhances feature extraction and classification. The 2 components that make up the layer are the Sigmoid activation function and the 64 nodes. The dense layer is then finally connected to the output layer, the output layer is then given by two nodes that result in the last feature map. The output is a probability, the higher probability is the classification decision.

5.4.3 Particle Swarm Optimization

Particle swarm optimization (PSO), first introduced in 1995 by Kennedy and Eberhart [45], is an algorithm to find the maximum of a multidimensional vector. It relies on the exhaustive search of the maximum value by evaluating all possible function parameters [46].

Following the simple architecture of chapter 6, we optimize the model using PSO [47] and data

set 1. As mentioned, PSO relies on the iterative tuning and model evaluation of the different hyper-parameters to obtain optimal results. By tuning and evaluating the model on different hyper-parameter we can select the optimal accuracy model.

CNN relies on loss function for training, the change of the loss function following each input decides classification. Minimizing the loss or cost function summarizes CNN classification. Gradient descent is loss function reduction by the architectural and hyper-parameter tuning [48]. For this model we tuning the following parameters:

- Number of convolutional layers;
- Number of neurons in the convolutional layers;
- Number of dense layers;

Following the optimization, we generated 27 models presented in 7.3.

Following optimization, the optimized model is integrated and automated into the system's pipeline. With that, we have an end-to-end solution for camera identification. Starting from image extraction to identification, the model achieves high accuracy and performance. We automate the model by developing an algorithm that can run in the background. The algorithm starts by extracting images from each camera at random intervals. After reaching a specific threshold, the algorithm starts PRNU noise extraction. Following building the second data set, automatic PSO optimization and model selection occurs. After selecting the optimal model, identification is the classification of new images. This is also automated by taking and evaluating a random image every specified time.

Chapter 6

System Design and Implementation

In this chapter, we summarize the end-to-end solution for the trusted feed. The identification process is made up of a pipeline of processes that are connected and integrated together to form one cohesive scalable solution. In this chapter, we cover all the stages and design choices for such a solution. Further, we elaborate on how the different parts of the pipeline are automated into one end-to-end solution. We also justify optimization and scalability choices.

6.1 Introduction

Passive security requires no assumption on the hardware and software security. Instead, the aim is to have a remote solution, that sits on trusted hardware, that is able to analyze and identify the source based on the feed. With that said, the solution starts with that camera and noise it generates all the way to the identification process on the other end.

6.2 System Design and Implementation

This section covers all the stages of the pipeline for our end-to-end solution. We also justify scalability and design decisions.

The pipeline can be divided into 6 connected stages. The first stage is the raw image acquisition process presented by the two cameras. The second stage is data collection and storage. Following data collection, stages three and 4 are noise extraction and noise collection. The fifth stage is annotation and data preprocessing before the sixth stage which is training. Figure 6.1 presents the system design presented as a 6 stages pipeline.

6.2.1 Image Acquisition, Storage, and Data Collection

The cameras are responsible for image acquisition and storage. As mentioned, all modern camera has an image enhancement pipeline to introduce errors that interfere with PRNU [37]. Moreover,

compression algorithms are known to destroy and recreate the image with every use [24]. Such processes can reduce PRNU noise dominance in images. Can we bypass such biases affecting the PRNU noise extraction process?

PRNU is a dominant unique artifact generated at acquisition time. The fact that it is generated at acquisition time implies that we can bypass many of the post-acquisition processing stages [38]. One of the ways to bypass such stages and compression is by extracting raw images that contain minimal processing and saving them in uncompressed format. However, such a process needs automation to be a fast and scalable solution. The machine learning model might need a large dataset of images before accuracy identification is possible. Manual data collection and storage can be suitable for identification, but not for a scalable security solution. With that said, we use cameras for image acquisition that have an open-source SDK. The SDK is important for two reasons. It allows us to automate the data collection process of the model by enabling programmatic image acquisitions, storage, and annotation. The second reason is that it allows us to change the configuration of the camera data stream and storage to extract raw images and store them in arrays. The cameras used are two Intel Real Sense Cameras [25]. The SDK can be found on GitHub [25].

The cameras are configured to extract 600 images from each camera. The images then are stored for noise extraction.

6.2.2 Noise Extraction

As mentioned in chapter 2 section 2.2.2, given a large data set, there have been attempts to bypass the noise extraction process and feed the images to the classifier directly. While such techniques were successful in previous research by using GoogleNet and AlexNet, such techniques require a very large dataset. Our design allows the collection of a very large dataset, yet, a large data set can cause scalability and performance issues.

The main reason why such previous work requires a large data set is because of the image content and biases. Since the images are fed in the CNN directly, the CNN can't distinguish between the noise patterns and image content. Hence, while the CNN model can eventually learn such noise patterns, it requires a large data-set to do so.

Image content can be a big source of bias. Hence, an optimization for image content is to extract a new data set made of only the PRNU noise patterns and use only the PRNU noise of each camera for the identification. Since we have already extracted the image dataset, we can use the wavelet transform 4.4 along with formula 2.1 to extract a new data set of image noise. This is demonstrated in figure 6.1 where the camera is used to extract the first data set. Then the second data set is the result of filtering the original one before annotating the noise based on its source.

6.2.3 Preprocessing

The security of the model stems from the basis that if we fingerprint and learn an image source we can identify it in the future. The concept can be modeled using machine learning terms are training and

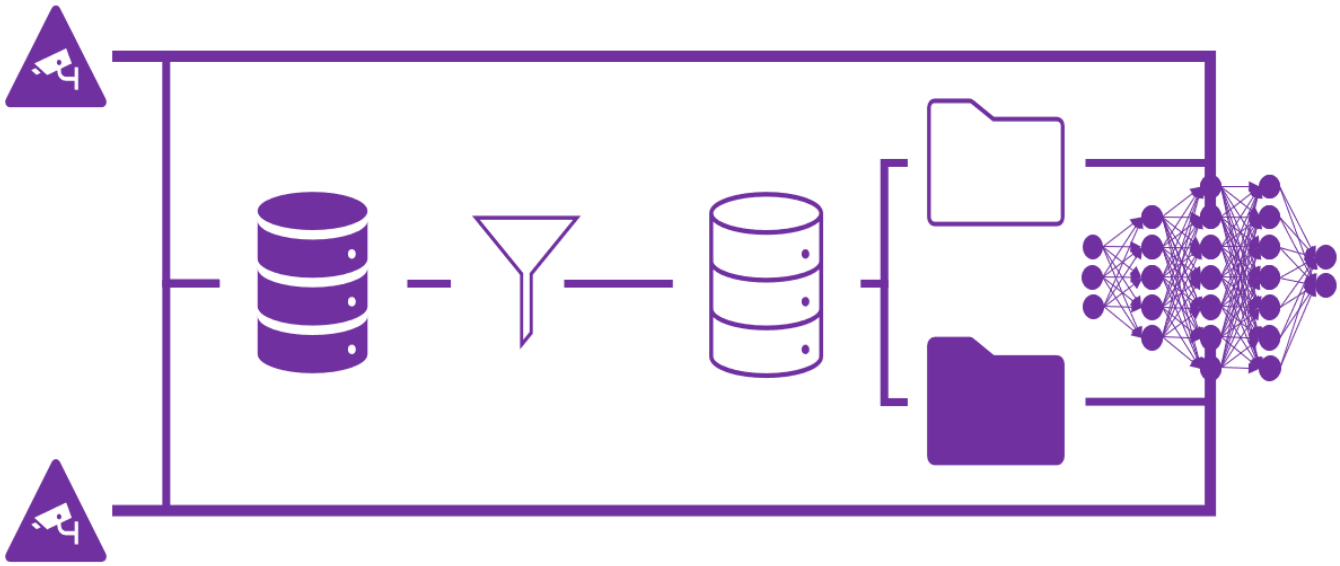

Figure 6.1: System Pipeline

testing data. The training data is the data collected, fingerprinted, and analyzed. The testing data, is used for evaluating the model. With that said future identified images can be also considered testing data and their identification metrics, such as testing accuracy, can be thought of as the identification accuracy. In the preprocessing stage, we annotate the noise based on their source. We then place them in appropriate data structures and dimensions. Then, we split the dataset into training, validation, and testing sets. Moreover, we sacrifice accuracy for scalability by cropping the noise dimensions into 256 * 256 pixels. The original dimension is 1280 * 980 pixels. The evaluation shows that the model was still able to generate a high degree of accuracy on cropped images.

6.2.4 Training and Optimization

Training is part of the identification process which is the final stage of the model. Following preprocessing the noise into machine learning data set, the fingerprinting and identification process uses CNN classification. The final CNN network devised for identification was inspired by a simple binary classification model [44]. The reason for such a network is its simplicity. This means that the model doesn't need great computation which aids scalability and performance. The second reason is that since we only have two cameras, we can only make use of binary classification models.

The model, made up of dense and convolutional layers, was able to learn such noise patterns in a short manner, a maximum of 5 training epochs. Chapter 7 shows the evaluation of the CNN network and the solution. Following the initial model, we aimed to optimize the model by using PSO. PSO aimed to find the highest accuracy model that yields optimal performance. The PSO process was the iterative evaluation of 27 different binary CNN models. The process tuned different hyperparameters including convolutional layers, neurons, and dense layers. Following such optimization, the model was able to achieve optimal testing accuracy using 1 convolutional layer, 32 nodes, and 0 dense layers.

Chapter 7 table 7.3 presents the different accuracy for the different models trained and evaluated.

Chapter 7

Evaluation

The chapter evaluates the identification process of our solution. The valuation was done using the standard metrics used for testing a machine learning model. Moreover, we include the result of the optimization on performance and accuracy.

7.1 Introduction

While there are many stages involved in the system, eventually the identification process is done by a CNN classifier. For that, we can use the standard metric used for classification the machine learning model to evaluate the accuracy of the model in distinguishing the camera source based on the images. The evaluation is done over more than 27 different CNN models with two different data sets.

7.2 Evaluation

The evaluation is done on more than one data set. The first data set is made up of images taken in a changing environment in terms of display content and light. The second is controlled where the images are taken following the method devised in the original PRNU paper [17]. This section will cover the result of both data sets. Since both results gave a high degree of accuracy, we optimized the first data set using PSO to get on a final model that achieve the highest identification and performance accuracy.

7.2.1 Data set 1

For the first experiment, we collected a data set of 2000+ noise images coming from each camera. The cameras were put in a stationary constantly changing environment. They were automated to capture 4000 images over a period of 2 hours. This allowed the final image to be diverse with different lighting and image content. Following the collection of the image data set, the denoising process resulted in a field of semi-homogeneous colors with no image content present.

But, due to automation, we designed the system in such a way that the camera takes and evaluates 600 images for their identification results. If the testing accuracy of the model achieves more than

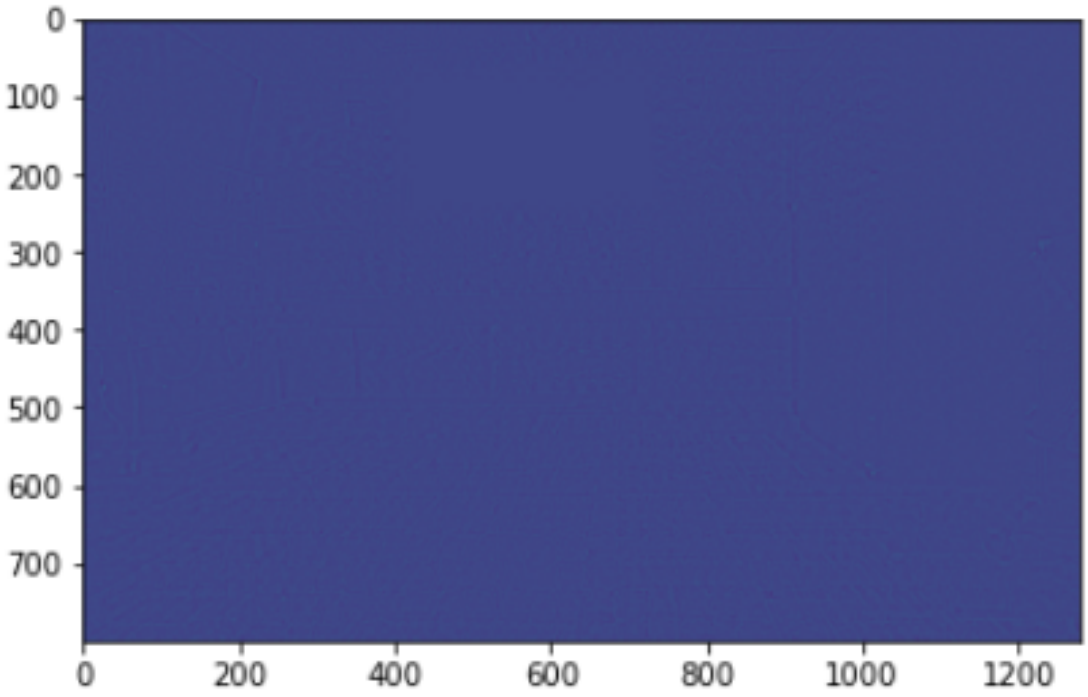

Figure 7.1: Sample Raw Noise Image

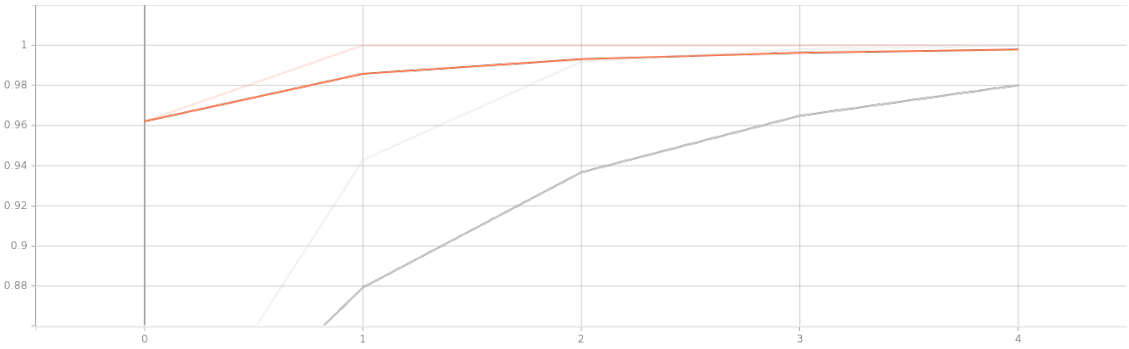

Figure 7.2: Optimized model (1 convolutional layer, 64 nodes, and 0 dense layers) training and validation accuracy

95% testing accuracy, the model stops training and starts testing new images. The model was able to achieve a result of around 96% to 98% training, testing, and validation accuracy with different epochs on the first try. The model was able to achieve such accuracy after 3 to 4 epochs with a batch size of 32. An epoch is a full round of training over the entire dataset. Batch size is the number of examples, noise samples, used in one training iteration. With that said, the model didn't need more than 600 images for each camera. Moreover, the original size of the image is 920 * 1280 pixels wide, such dimensions can make the denoising process slow. With that said, we attempted to crop the images into square sizes of 256 * 256 pixels. That model also achieved a high identification accuracy from the first try, 95%. Figure 7.2 shows the training and validation accuracy of the model as a function of epochs. We can see the model's identification accuracy continued to increase with each epoch to reach 96% after 4 epochs.

Figure 7.2 shows the accuracy a model made up of 1 convolutional layer, 64 nodes, and 9 dense layers. With model was chosen from the PSO models generated.

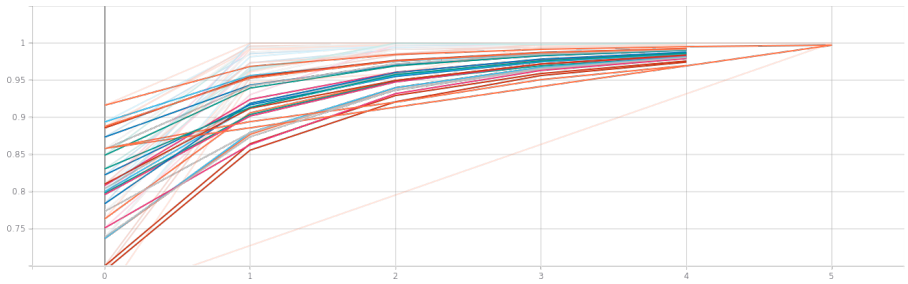

Figure 7.3: Optimized model (1 convolutional layer, 32 nodes, and 0 dense layers) testing and validation accuracy

7.2.2 Data set 2

For the second data set, we needed to make sure that no other biases might be involved in the learning process. Moreover, we needed to make sure that the pattern recognition is occurring on the PRNU noise. For such reasons, unlike the first data set, we need to have more control over the environment. For that, we used the method implemented in [17] and block the camera lens from any visible light. This means that all the images that are generated are dark fields. After extracting the noise data set, the result was also very similar to data set 1 7.2.1. With a training accuracy and validation accuracy of 95%+, and a testing accuracy of 96%. The model was able to achieve such accuracy after 3 to 4 epochs with a batch size of 32.

7.3 Optimized Training

Following the architecture of chapter 6, we optimize the model using PSO [47] and data set 1. As mentioned, PSO relies on the iterative tuning and model evaluation of the different hyper-parameters to obtain optimal results. By tuning and evaluating the model on different hyper-parameter we can select the optimal accuracy model. The following table 7.3 show the different testing accuracies obtained after PSO. The table shows that all the different architecture achieved high identification accuracy. The model that was able to achieve the highest result was the one with 1 convolutional layer, 32 neurons, and 0 dense layers.

Figure 7.3 show training accuracies for the 27 different models used in PSO as a function epoch. We can see the all of them converge over 95%+ training accuracy.

Figure 7.4 shows the validation loss for the 27 different models used in PSO as a function epoch. We can see the all model start with high (high cost function value) and decrease towards a very small value.

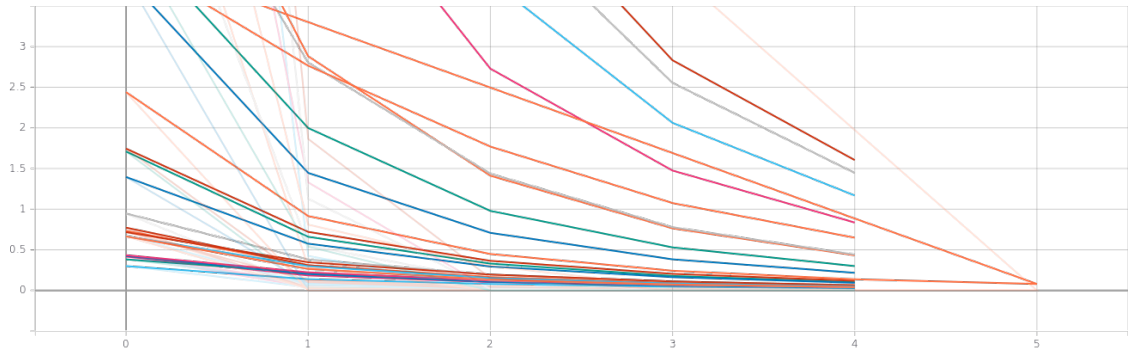

Figure 7.4: Optimized model training loss

Conv Layers	Nodes	Dense	Testing Accuracy
1	128	0	98.22%
1	128	1	97.83%
1	128	2	97.52%
1	32	0	99.42%
1	32	1	97.27%
1	32	2	98.5%
1	64	0	97.32%
1	64	1	97.64%
1	64	2	97.64%
2	128	0	99.05%
2	128	1	98.28%
2	32	0	96.88%
2	32	1	98.92%
2	32	2	98.2%
2	64	0	98.08%
2	64	1	98.81%
2	64	2	99.1%
2	128	0	96.69%
3	32	0	97.89%
3	32	1	98.71%
3	32	2	97.61%
3	64	0	98.91%
3	64	1	97.92%
3	64	2	98.37%
3	128	0	96.69%
3	128	1	98.04%

Chapter 8

Discussion

This chapter is a discussion on the solution presented. The chapter covers the general and specific weaknesses of the solution and passive security. It also covers the possible attack scenarios and security challenges. Following that, the chapter also discusses the solution's specific application and where can it be effective. The last section discusses the future work for improving and evaluating the system.

8.1 Introduction

Today, more and more services are being automated and done remotely. Entire industries have emerged to automate manual tasks and ease communication and management. Many of such communication and management rely on internet-connected cameras as a vital part of their service. Yet, when it comes to cameras, privacy and security are almost as important as the service itself. This is true especially in industries where security and privacy are crucial. Industries such as remote inspection, conferencing, and auditing need a high level of assurance to offer their service.

With the ever-increasing reliance on IoT devices in industries, the attack vectors increase. Industries are looking for security solutions for feed assurance. But, such security is not trivial. The problem with internet-connected cameras is the many attack vectors. Attack vectors include levels such as hardware, software, operating system, network, and encryption. Yet, with all such attack vectors, modern cameras are ill-equipped to deal with it.

8.1.1 Passive Security

As mentioned, active security measures can be an infeasible and impractical solution. This is true for industries that have large amounts of cameras or provide software as a service (SaaS). A large amount of cameras requires a big investment in camera security. SaaS technologies can't assume any hardware security for the camera. This is because the camera can be a webcam or phone camera. An example of SaaS is remote conferencing software like Zoom. Yet, while there are many struggles with active security, passive security cannot be a solution for inclusive security. This is since passive security, at least our solution, only can guarantee to a degree of certainty only some security properties.

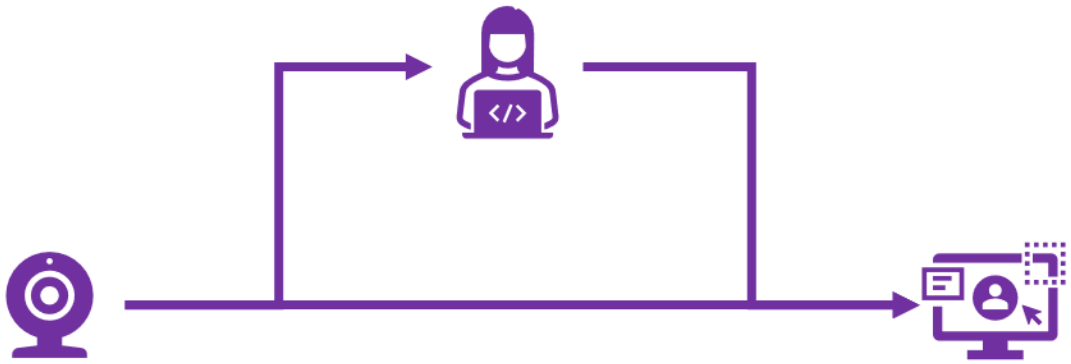

Figure 8.1: Man-in-the-middle Attack against Passive Security

Passive security can only guarantee authentication and integrity. It has no answer for attacks targeting availability and confidentiality.

8.1.2 Attack Scenario

Another challenge for our solution is that it is vulnerable to replay and man-in-the middle attack for some scenarios. Scenarios involve the adversary having access to the camera acquisition and knowledge of the passive security process. Assume the following scenario, attacker Bob wants to bypass the identification process to send false images. Assuming secure storage of image and noise data sets, yet Bob has access to the camera, Bob can capture an image. After Bob has an image, he can extract the PRNU noise of the image, using formulas 2.1 and 4.4, and inject it into a false image. The new image might pass the identification process since it contains a similar PRNU pattern learned by the model. Bob can also enhance classification by denoising the false image before adding the PRNU noise. Thus making the PRNU the only dominant noise pattern in the image. Other attack scenarios against deep learning techniques are identified in [49,50]. Such attacks relies on the adversary knowing different properties of the model and the system to bypass identification.

Unfortunately, there is no effective solution to such attacks. Yet, due to the complexity and technicality of such an attack, it is unlikely and involves access to images or cameras.

8.1.3 Scalability

Furthermore, one scalability issue that the model needs to contend with is the time required for PRNU extraction. PRNU extraction can take a relatively long time with respect to other processes in the model. It can take up to 20 seconds to denoise 1280 x 800 images [51]. But, in this solution, noise extraction happens once at the beginning, moreover, it is automated in the background. After, noise extraction and training the identification process happens instantaneously. We tested the model on cropped images of 256 x 256 dimensions with significantly less extraction time; the model still achieves a high degree of testing accuracy.

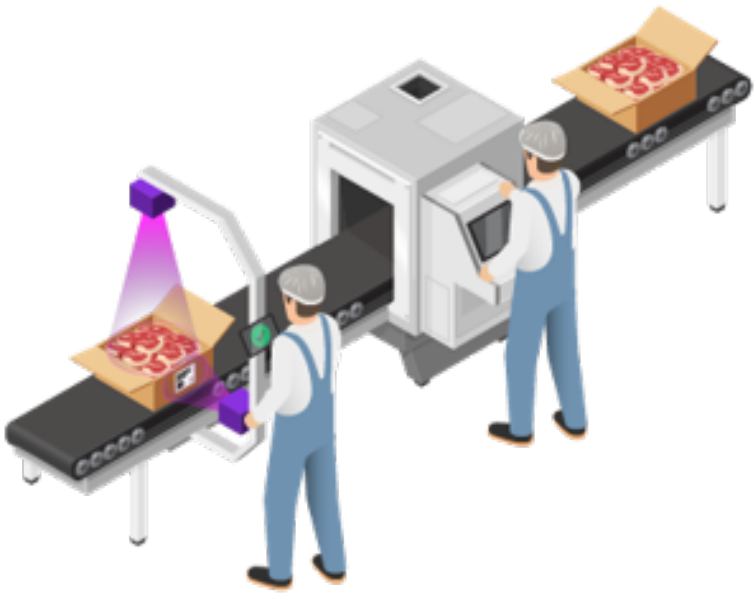

Figure 8.2: Bondi Labs Verification Process taken from [1]

8.2 Application

This work comes to secure Bondi Labs Continuous Verification (BLV) Project using Elixar security stack [52]. BLV is an augmented intelligence project that aims to verify the carton labels ensuring they meet export requirements. Figure 8.2 shows an info-graphic of the BLV project.

The project consists of two stationary cameras that capture conveyor belt meat. The cameras then send the feed to an edge device that does the AI processing, segmentation, and classification. The edge device, responsible for the processing, is also responsible for transmitting the feed over the internet. The feed is then viewed by auditors as part of remote auditing. Since internet transmission is key in this technology, encryption is unavoidable for privacy [53]. With that said, active security is a must.

While securing each camera in the BLV project with hardware and software specifications is a solution, it's an expensive one. A more reasonable approach would be to secure the edge device responsible for transmitting the feed over the internet. If the edge device is secure the communication link between the camera and device needs only to provide a certain degree of certainty. Especially since the camera does not need an internet connection and the communication can be air-gaped. In this case, passive security can be effective in providing authentication and integrity checks.

Diagram 8.3 demonstrates the security stack of the BLV project. The stack starts with the camera capturing the feed and sending it to the edge device. The operation system of the device then intercepts and handles the feed for AI processing. The first security link is between the camera and the edge device. The edge device then encrypts the video and sends it over the internet via a network provider. Since Elixar is a cloud-hosted application, the cloud provider handles transmitting the feed to the auditor.

As mentioned, digital feed assurance is important for any technology that relies on cameras as a vital part of its technology. Yet, while passive security via remote camera identification can solve some security issues, it can't ensure feed privacy and availability. Yet, since the communication link

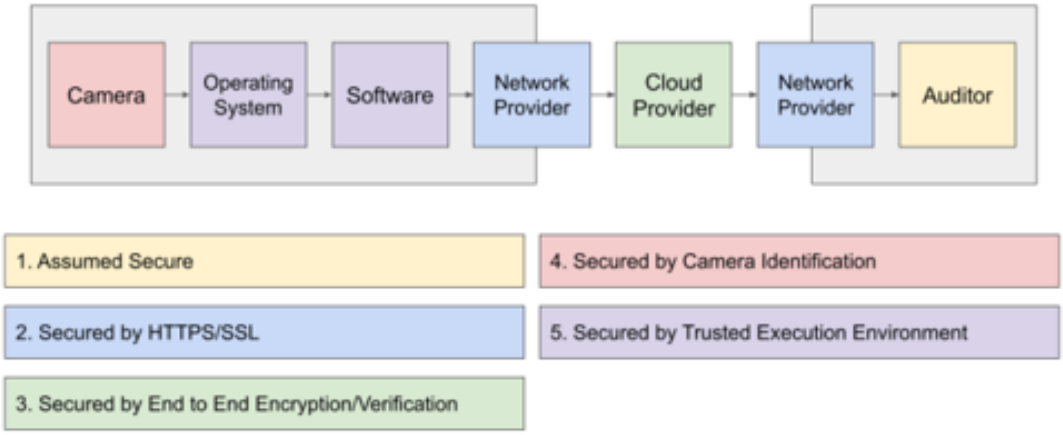

Figure 8.3: Elixar Security Stack taken from [2]

between the camera and the edge device is air-gaped, we can rely on passive security for authentication and integrity checks. Passive security can provide some security checks for the first chain of the BLV technology stack.

8.3 Future Work

The aim of this work is not to present a complete secure passive security solution for feed assurance. Rather, the work aims to provide a degree and certainty and attestation about the feed received. There is no such thing as a fully secure system. With that said, the aim is to make the system as hard as possible for attackers to exploit. Passive security as demonstrated in this work is a good way for such assurance.

Yet, while the solution presented in this thesis works to a degree, its applicability is still young. There still exist many challenges and improvements for it to be a good solution moving forward.

One of the challenges of such a solution is that it needs special cameras for automation. In this work, we use two Intel Real Sense with open-source SDKs [25]. The open-source SDKs allowed us to automate the data collection, configuration, and image extraction process and integrate it into the system as a one-end-to-end solution. Yet, the reliance on such SDK means that we can't use any normal modern camera for this solution. The cameras that can be used must have an SDK that is similar to the Intel Real sense SDK. The SDK should allow the configuration, acquisition, and storage of images as well and the automation and integration with the other stages of the pipeline. This makes our solution impractical since it is limited to special types of cameras. One of the future challenges is to find a way to develop an interface for image acquisition is the camera independent.

Another weakness of the system that stems from the use of special types of cameras is that we

couldn't extend our work further. Since we only have two Intel Real Sense cameras and cannot use and extract raw images from other cameras, we are limited to two cameras for identification. While the initial implementation of the project processed images coming from more than two sources, we can only use two for the identification solution. This is since only two are available. Future work must include the use and automation of more camera sensors. This is to test the model on a non-binary classification and to see the effects of scalability on the solution.

With more camera sensors, identification scalability can grow linearly. Since there is one hardware that handles noise extraction, fingerprinting, and identification, the identification time will increase. With each camera we need a new set of images taken from it to fingerprint and train. As mentioned fingerprint via noise extraction can take a long (up to 1 min per 1 image). Moreover, the training, validation, and identification process will also need more time. Thus, the future will include taking such scalability issues into account and evaluating the model against such constraints.

Chapter 9

Conclusion

The broad aim of the thesis was to provide a workable and practical security mechanism for one link of a trusted video. This link is between the camera and the edge device. In this work, we aimed to use passive security in the form of remote camera identification to provide some camera security guarantees. Based on prior research, we were able to provide a solution that uses a dominant noise found in the image to fingerprint and identify the camera. The work included extracting such noise and using a deep learning model to distinguish the cameras. Following the design of such a model, we aimed to provide an automated end-to-end solution for passive security. By using open-source SDKs and developing our own automation scripts, we were able to achieve our goal. Furthermore, work on the scalability of the model by boosting its performance. By removing different biases that hinder fingerprinting and identification, we were able to achieve an automate scalable solution for passive security. Overall, while the project has some weakness and challenges it needs to address in future work, it was able to achieve the aims.

While the work is still young, we believe it has the potential to be a viable practical solution for passive security in remote cameras. Inside of investing in hardware and software security to provide some security guarantees, passive security aim to achieve such guarantees without any hardware or software assumptions. With that said, while passive security can ensure such guarantees it is not immune from attack, nor can provide all the security guarantees of active security. Aside from attacks mentioned and scalability issues, the passive security doesn't provide any confidentiality and availability guaranteed. The makes active security necessary for privacy in some scenarios. Moreover, due to limited resource we could only implement the design on two open-source SDK cameras the enables automation. While the work aims at scalablitiy by boosting identification performance, we couldn't effectively test such scalability in the model since we only had two cameras.

To our knowledge, this is the first automated end-to-end remote camera identification solution for passive security. Yet, while this work is promising it is very young. There are many areas of further research and improvements. One area of research can be camera noise and extraction process. PRNU noise is not the only noise found in image, the work could experiment with different noise and noise extraction technique that can aid accuracy and scalability. Moreover, on the machine learning side,

there are many application for such fingerprint and identification on other supervised and unsupervised training models. This can be included can further evaluation of the model and it's result. It can also aid scalability if a new model the identify pattern more accurately is devised.

Bibliography

[1] J. Hamilton, Bondi labs continuous verification and remote inspection, Bondi Labs, Augmented Intelligence (2020).

[2] J. Scarsbrook, Elixar security stack, University of Queensland (2021).

[3] L. Tawalbeh, F. Muheidat, M. Tawalbeh, M. Quwaider, Iot privacy and security: Challenges and solutions, Applied Sciences 10 (2020) 4102. doi:10.3390/app10124102.

[4] W. H. Hassan, et al., Current research on internet of things (iot) security: A survey, Computer networks 148 (2019) 283–294.

[5] Y. Seralathan, T. T. Oh, S. Jadhav, J. Myers, J. P. Jeong, Y. H. Kim, J. N. Kim, Iot security vulnerability: A case study of a web camera, in: 2018 20th International Conference on Advanced Communication Technology (ICACT), 2018, pp. 172–177. doi:10.23919/ICACT.2018.8323686.

[6] J. Wurm, K. Hoang, O. Arias, A. Sadeghi, Y. Jin, Security analysis on consumer and industrial iot devices, in: 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), 2016, pp. 519–524. doi:10.1109/ASPDAC.2016.7428064.

[7] X. Lin, J.-H. Li, S.-L. Wang, F. Cheng, X.-S. Huang, et al., Recent advances in passive digital image security forensics: A brief review, Engineering 4 (1) (2018) 29–39.

[8] J. Bernacki, Robustness of digital camera identification with convolutional neural networks, Multimedia Tools and Applications (2021) 1–17.

[9] C. Galdi, M. Nappi, J.-L. Dugelay, Multimodal authentication on smartphones: Combining iris and sensor recognition for a double check of user identity, Pattern Recognition Letters 82 (2016) 144–153.

[10] J. Bernacki, A survey on digital camera identification methods, Forensic Science International: Digital Investigation 34 (2020) 300983.

[11] A. Vulpe, A. Paikan, R. Craciunescu, P. Ziafati, S. Kyriazakos, A. Hemmer, R. Badonnel, Iot security approaches in social robots for ambient assisted living scenarios, in: 2019 22nd International Symposium on Wireless Personal Multimedia Communications (WPMC), IEEE, 2019, pp. 1–6.

[12] J. Lukas, J. Fridrich, M. Goljan, Digital camera identification from sensor pattern noise, IEEE Transactions on Information Forensics and Security 1 (2) (2006) 205–214.

[13] J. Wurm, K. Hoang, O. Arias, A.-R. Sadeghi, Y. Jin, Security analysis on consumer and industrial iot devices, in: 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), IEEE, 2016, pp. 519–524.

[14] A. Banafa, Iot standardization and implementation challenges, IEEE internet of things newsletter (2016) 1–10.

[15] V. Hurtoi, D. Avadanei, Iot project management, Informatica Economica 24 (3) (2020) 75–80.

[16] I. Arce, The weakest link revisited [information security], IEEE Security Privacy 1 (2) (2003) 72–76. doi:10.1109/MSECP.2003.1193216.

[17] M. Goljan, Digital camera identification from images–estimating false acceptance probability, in: International workshop on digital watermarking, Springer, 2008, pp. 454–468.

[18] J. R. Janesick, Scientific charge-coupled devices, Vol. 83, SPIE press, 2001.

[19] J. Lukas, J. Fridrich, M. Goljan, Determining digital image origin using sensor imperfections, in: Image and Video Communications and Processing 2005, Vol. 5685, International Society for Optics and Photonics, 2005, pp. 249–260.

[20] Y. Chen, Y. Huang, X. Ding, Camera model identification with residual neural network, in: 2017 IEEE International Conference on Image Processing (ICIP), IEEE, 2017, pp. 4337–4341.

[21] A. Tuama, F. Comby, M. Chaumont, Camera model identification with the use of deep convolutional neural networks, in: 2016 IEEE International workshop on information forensics and security (WIFS), IEEE, 2016, pp. 1–6.

[22] L. Bondi, L. Baroffio, D. Guera, P. Bestagini, E. J. Delp, S. Tubaro, First steps toward camera model identification with convolutional neural networks, IEEE Signal Processing Letters 24 (3) (2016) 259–263.

[23] D. Freire-Obregón, F. Narducci, S. Barra, M. Castrillón-Santana, Deep learning for source camera identification on mobile devices, Pattern Recognition Letters 126 (2019) 86–91.

[24] W. Van Houten, Z. Geradts, Source video camera identification for multiply compressed videos originating from youtube, Digital Investigation 6 (1-2) (2009) 48–60.

[25] Intel® realsense™ cross-platform sdk (Last update 2021).
URL https://github.com/IntelRealSense

[26] M. Goljan, J. Fridrich, T. Filler, Large scale test of sensor fingerprint camera identification, in: Media forensics and security, Vol. 7254, International Society for Optics and Photonics, 2009, p. 72540I.

[27] Y. Cengel, T. M. Heat, A practical approach, New York, NY, USA: McGraw-Hill, 2003.

[28] T. Baar, W. van Houten, Z. Geradts, Camera identification by grouping images from database, based on shared noise patterns, arXiv preprint arXiv:1207.2641 (2012).

[29] F. Marra, G. Poggi, C. Sansone, L. Verdoliva, Evaluation of residual-based local features for camera model identification, in: International conference on image analysis and processing, Springer, 2015, pp. 11–18.

[30] L. Baroffio, L. Bondi, P. Bestagini, S. Tubaro, Camera identification with deep convolutional networks, arXiv preprint arXiv:1603.01068 (2016).

[31] T. Gloe, R. Böhme, The'dresden image database'for benchmarking digital image forensics, in: Proceedings of the 2010 ACM Symposium on Applied Computing, 2010, pp. 1584–1590.

[32] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems 25 (2012) 1097–1105.

[33] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015, pp. 1–9.

[34] M. De Marsico, M. Nappi, D. Riccio, H. Wechsler, Mobile iris challenge evaluation (miche)-i, biometric iris dataset and protocols, Pattern Recognition Letters 57 (2015) 17–23.

[35] Y. Qian, J. Dong, W. Wang, T. Tan, Deep learning for steganalysis via convolutional neural networks, in: Media Watermarking, Security, and Forensics 2015, Vol. 9409, International Society for Optics and Photonics, 2015, p. 94090J.

[36] V. Gudivada, A. Apon, J. Ding, Data quality considerations for big data and machine learning: Going beyond data cleaning and transformations, International Journal on Advances in Software 10 (1) (2017) 1–20.

[37] T. Gerhard, Bias: considerations for research practice, American Journal of Health-System Pharmacy 65 (22) (2008) 2159–2168.

[38] T. Brooks, B. Mildenhall, T. Xue, J. Chen, D. Sharlet, J. T. Barron, Unprocessing images for learned raw denoising, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 11036–11045.

[39] R. Sumner, Processing raw images in matlab, Department of Electrical Engineering, University of California Sata Cruz (2014).

[40] L. Fan, F. Zhang, H. Fan, C. Zhang, Brief review of image denoising techniques, Visual Computing for Industry, Biomedicine, and Art 2 (1) (2019) 1–12.

[41] R. Polikar, Wavelet tutorial - part 1 (1994).

[42] S. Albawi, T. A. Mohammed, S. Al-Zawi, Understanding of a convolutional neural network, in: 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1–6. doi:10.1109/ICEngTechnol.2017.8308186.

[43] P. Kim, Convolutional Neural Network, Apress, Berkeley, CA, 2017. doi:10.1007/978-1-4842-2845-6_6.
URL https://doi.org/10.1007/978-1-4842-2845-6_6

[44] F. Chollet, Building powerful image classification models using very little data, Keras Blog 5 (2016).

[45] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, Vol. 4, 1995, pp. 1942–1948 vol.4. doi:10.1109/ICNN.1995.488968.

[46] A. Tam, A gentle introduction to particle swarm optimization (2021).

[47] J. Kennedy, R. Eberhart, Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, Vol. 4, 1995, pp. 1942–1948 vol.4. doi:10.1109/ICNN.1995.488968.

[48] S. Ruder, An overview of gradient descent optimization algorithms, arXiv preprint arXiv:1609.04747 (2016).

[49] F. Marra, D. Gragnaniello, L. Verdoliva, On the vulnerability of deep learning to adversarial attacks for camera model identification, Signal Processing: Image Communication 65 (2018) 240–248.

[50] I. J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, arXiv preprint arXiv:1412.6572 (2014).

[51] J. Bernacki, M. Klonowski, P. Syga, Some remarks about tracing digital cameras-faster method and usable countermeasure., in: SECRYPT, 2017, pp. 343–350.

[52] B. Labs, Elixar application (2021).

[53] S. Garfinkel, PGP: pretty good privacy, " O'Reilly Media, Inc.", 1995.

[54] N. B. Luca Bondi, Paolo Bestagini, Python porting of prnu extractor and helper functions (2018).
URL https://github.com/polimi-ispl/prnu-python

Appendix A

Appendix

A.1 Camera Automation

```
import pyrealsense2 as rs

import NumPy as np
import cv2
import time
from PIL import Image

pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.color, 1280, 800, rs.format.raw16, 30)

# Start streaming
pipeline.start(config)

try:
    for x in range(1, 10000):

        # Wait for a coherent pair of frames: depth and color
        frames = pipeline.wait_for_frames()
        color_frame = frames.get_color_frame()

        # Convert images to numpy arrays
        color_image = np.asanyarray(color_frame.get_data())

        np.save('D:/raw-dataset-4/raw-2/raw'+str(x)+'.npy', color_image)
        time.sleep(3)

finally:

    # Stop streaming
```

```
    pipeline.stop()
```

A.2 Noise Extraction

Noise Extraction using the wavelet transform taken from [54].

```
def noise_extract(im: np.ndarray, levels: int = 4, sigma: float = 5) -> np.ndarray:
    """
    NoiseExtract as from Binghamton toolbox.
    :param im: grayscale or color image, np.uint8
    :param levels: number of wavelet decomposition levels
    :param sigma: estimated noise power
    :return: noise residual
    """

    # assert (im.dtype == np.uint8)
    # assert (im.ndim in [2, 3])

    im = im.astype(np.float32)

    noise_var = sigma ** 2

    if im.ndim == 2:
        im.shape += (1,)

    W = np.zeros(im.shape, np.float32)

    for ch in range(im.shape[2]):

        wlet = None
        while wlet is None and levels > 0:
            try:
                wlet = pywt.wavedec2(im[:, :, ch], 'db4', level=levels)
            except ValueError:
                levels -= 1
                wlet = None
        if wlet is None:
            raise ValueError('Impossible to compute Wavelet filtering for input size: {}'.format(im.shape))

        wlet_details = wlet[1:]

        wlet_details_filter = [None] * len(wlet_details)
        # Cycle over Wavelet levels 1:levels-1
        for wlet_level_idx, wlet_level in enumerate(wlet_details):
            # Cycle over H,V,D components
```

```
            level_coeff_filt = [None] * 3
            for wlet_coeff_idx, wlet_coeff in enumerate(wlet_level):
                level_coeff_filt[wlet_coeff_idx] = wiener_adaptive(
    wlet_coeff, noise_var)
            wlet_details_filter[wlet_level_idx] = tuple(level_coeff_filt)

        # Set filtered detail coefficients for Levels > 0 ---
        wlet[1:] = wlet_details_filter

        # Set to 0 all Level 0 approximation coefficients ---
        wlet[0][...] = 0

        # Invert wavelet transform ---
        wrec = pywt.waverec2(wlet, 'db4')
        try:
            W[:, :, ch] = wrec
        except ValueError:
            W = np.zeros(wrec.shape[:2] + (im.shape[2],), np.float32)
            W[:, :, ch] = wrec

    if W.shape[2] == 1:
        W.shape = W.shape[:2]

    W = W[:im.shape[0], :im.shape[1]]

    return W
```

A.3 Preprocessing

```
from absl import app, logging, flags
import numpy as np
import os
import cv2
import random
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D


def crop_image(arr, w, h):
    return arr[0:w, 0:h]

def main(args):
```

```
18
19      datadir = '/mnt/omar2/denoised_jt_dataset_cnn'
20
21      categories = []
22
23      for c in os.listdir(datadir):
24          if "label" not in c:
25              categories.append(c)
26
27      training_data = []
28
29      logging.info("loading")
30
31      for cat in categories:
32          path = os.path.join(datadir, cat)
33          class_num = categories.index(cat)
34          counter = 0
35          for img in os.listdir(path):
36              img_array = np.load(os.path.join(path, img))
37              cropped_img_array = crop_image(img_array, 128, 128)
38              training_data.append([cropped_img_array, class_num])
39              counter = counter + 1
40              if counter > 500:
41                  break
42      logging.info("Training data array done.")
43      X = []
44      y = []
45      random.shuffle(training_data)
46      logging.info("Training data suffled.")
47      for features, label in training_data:
48          X.append(features)
49          y.append(label)
50    logging.info("Convert into numpy done.")
51      X = np.array(X)
52      y = np.array(y)
53      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
        random_state=42)
54      X_train = np.expand_dims(X_train, axis=-1)
55      X_train = X_train.reshape(-1, 128, 128, 1)
56      X_test = np.expand_dims(X_test, axis=-1)
57      X_test = X_test.reshape(-1, 128, 128, 1)
58      # y_train = y_train.reshape(-1, 128, 128, 1)
59      logging.info("Train test slip done.")
60
61      logging.info("Starting with the model")
62      print("X_train.shape", X_train.shape)
```

A.4 Automation

```
#!/usr/bin/env python3
import os
import prnu
import numpy as np


# The following script will capture image and store them in two folders raw
    -1 raw-2
# the dataset is stored in '/media/omar/New Volume/raw-dataset-4'
data_collection_script.py
# general path
path = '/media/omar/New Volume'
# path to the undenoised dataset directory
dataset_path = '/media/omar/New Volume/raw-dataset-4'
# the 3 directory formats required as in https://github.com/dusty-nv/jetson-
    inference/blob/master/docs/pytorch-collect.md#collecting-your-own-
    classification-datasets
sets = ['train','val','test']
# the two classes
classes = ['raw-1','raw-2']
# choose a name for the dataset to be saved
# dataset_name = 'raw_noise_dataset_1'
# ./ape1/py denoised_jt_dataset_1
dataset_name = 'denoised_jt_dataset_2_tiff'
# /home/kali/Desktop/thesis/test1
# the new dataset will be saved under /home/omar/Desktop/project/$1
storing_path = os.path.join(path,dataset_name)
os.mkdir(storing_path)
# creating a label.txt file as in
f = open(os.path.join(storing_path, "label.txt"), "w")
for cl in classes:
    f.write(cl)
    f.write("\n")
f.close()
for s in sets:
    # /home/kali/Desktop/thesis/test1/train
    set_path = os.path.join(storing_path, s)
    os.mkdir(set_path)
    for c in classes:
        # /home/kali/Desktop/thesis/test1/train/class
        classes_path = os.path.join(set_path, c)
        classes_dataset_path = os.path.join(dataset_path, c)
        num_images = len(os.listdir(classes_dataset_path))
        os.mkdir(classes_path)
        counter = 0
```

```
        for img in os.listdir(classes_dataset_path):
            # /home/kali/Desktop/thesis/test1/train/Camera/img
            try:
                img_path = os.path.join(classes_dataset_path,img)
                img_store = os.path.join(classes_path, str(counter)+" "+img)
                img_array = np.load(img_path)
                img_array = img_array.astype(np.float32) / 255
                noise = prnu.noise_extract(img_array)
                if counter <= num_images / 1.3:
                    img_name = os.path.join(storing_path,sets[0],c,img)
                    np.save(img_name, noise)
                    print("Image ",img_name," was stored in ",sets[0]," ",c)
                elif counter <= num_images / 1.23:
                    try:
                        img_name = os.path.join(storing_path,sets[1],c,img)
                        np.save(img_name, noise)
                        print("Image ",img_name," was stored in ",sets[1],"
    ",c)
                    except OSError:
                        print("Directory ",os.path.join(storing_path,sets
    [1],c),"doesn't exist.. BREAKING")
                        break
                else:
                    try:
                        img_name = os.path.join(storing_path,sets[2],c,img)
                        np.save(img_name, noise)
                        print("Image ",img_name," was stored in ",sets[2],"
    ",c)
                    except OSError:
                        print("Directory ",os.path.join(storing_path,sets
    [1],c),"doesn't exist.. BREAKING")
                        break
            except cv2.error as e:
                print("There is a opencv-error, don't panic just skip the
    image")
                pass
            counter=counter + 1
train.py
```

A.5 Tensor Flow CNN model

The CNN Model and PSO is inspired by [44].

A.6 Training

```
import numpy as np
import os
import cv2
import random
from sklearn.model_selection import train_test_split
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D

datadir = '/media/omar/New Volume/denoised_jt_dataset_cnn'

categories = []

for c in os.listdir(datadir):
    if "label" not in c:
        categories.append(c)

training_data = []

for cat in categories:
    path = os.path.join(datadir, cat)
    class_num = categories.index(cat)
    for img in os.listdir(path):
        img_array = np.load(os.path.join(path, img))
        training_data.append([img_array, class_num])

print("Training data array done.")
X = []
y = []

random.shuffle(training_data)

print("Training data suffled.")

for features, label in training_data:
    X.append(features)
    y.append(label)


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
    random_state=42)

print("Train test slip done.")
X_train = np.array(X_train)
y_train = np.array(y_train)
```

```
X_test = np.array(X_test)
y_test = np.array(y_test)


print("Convert into numpy done.")

print("Starting with the model")
model = Sequential()

model.add(Conv2D(256, (3, 3), input_shape=X_train.shape[0:]))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(256, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors

model.add(Dense(64))

model.add(Dense(1))
model.add(Activation('sigmoid'))

print("Compiling the model.")
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

print("Training the model.")
model.fit(X_train, y_train, batch_size=32, epochs=3, validation_split=0.3)

print("Saving the model")

model.save('saved_model/my_model1')

print("Evaluating the model.")
val_loss, val_acc = model.evaluate(X_test, y_test)

print("val loss", val_loss)
print("val accuracy", val_acc)
```

A.7 PSO

```
from absl import app, logging, flags
import numpy as np
```

```
import os
import random
from sklearn.model_selection import train_test_split
import time

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
from tensorflow.keras.callbacks import TensorBoard


def crop_image(arr, w, h):
    return arr[0:w, 0:h]

def main(args):

    datadir = '/mnt/omar2/denoised_raw_dataset_5'

    categories = []

    for c in os.listdir(datadir):
        if "label" not in c:
            categories.append(c)


    training_data = []

    logging.info("loading")

    for cat in categories:
        path = os.path.join(datadir, cat)
        class_num = categories.index(cat)
        counter = 0
        for img in os.listdir(path):
            img_array = np.load(os.path.join(path, img))
            cropped_img_array = crop_image(img_array, 128, 128)
            training_data.append([cropped_img_array, class_num])
            counter = counter + 1
            if counter > 500:
                 break

    logging.info("Training data array done.")
    X = []
    y = []

random.shuffle(training_data)

```

```
50      logging.info("Training data suffled.")
51
52      for features, label in training_data:
53          X.append(features)
54          y.append(label)
55
56
57      logging.info("Convert into numpy done.")
58
59      X = np.array(X)
60      y = np.array(y)
61
62      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
        random_state=42)
63
64      X_train = np.expand_dims(X_train, axis=-1)
65      X_train = X_train.reshape(-1, 128, 128, 1)
66
67      X_test = np.expand_dims(X_test, axis=-1)
68      X_test = X_test.reshape(-1, 128, 128, 1)
69      # y_train = y_train.reshape(-1, 128, 128, 1)
70      logging.info("Train test slip done.")
71
72  X = np.array(X)
73      y = np.array(y)
74
75      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
        random_state=42)
76
77      X_train = np.expand_dims(X_train, axis=-1)
78      X_train = X_train.reshape(-1, 128, 128, 1)
79
80      X_test = np.expand_dims(X_test, axis=-1)
81      X_test = X_test.reshape(-1, 128, 128, 1)
82      # y_train = y_train.reshape(-1, 128, 128, 1)
83      logging.info("Train test slip done.")
84
85      logging.info("Starting with the model")
86      print("X_train.shape", X_train.shape)
87
88      dense_layers = [0, 1, 2]
89      layer_sizes = [32, 64, 128]
90      conv_layers = [1, 2, 3]
91
92      for dense_layer in dense_layers:
93          for layer_size in layer_sizes:
94              for conv_layer in conv_layers:
```

```
95                NAME = "{}-conv-{}-nodes-dense-{}".format(conv_layer,
    layer_size, dense_layer, int(time$
96
97                tensorboard = TensorBoard(log_dir='logs/{}'.format(NAME))
98                model = Sequential()
99
100               model.add(Conv2D(layer_size, (3, 3), input_shape=X_train.
    shape[1:]))
101               model.add(Activation('relu'))
102               model.add(MaxPooling2D(pool_size=(2, 2)))
103
104               for l in range(conv_layer-1):
105                   model.add(Conv2D(layer_size, (3,3)))
106                   model.add(Activation('relu'))
107                   model.add(MaxPooling2D(pool_size=(2,2)))
108
109               for l in range(conv_layer-1):
110                   model.add(Conv2D(layer_size, (3,3)))
111                   model.add(Activation('relu'))
112                   model.add(MaxPooling2D(pool_size=(2,2)))
113
114               model.add(Flatten())
115
116               for l in range(dense_layer):
117                   model.add(Dense(layer_size))
118                   model.add(Activation('relu'))
119
120               model.add(Dense(1))
121               model.add(Activation('sigmoid'))
122
123               model.compile(loss='binary_crossentropy', optimizer='adam',
    metrics=['accuracy'])
124
125               model.fit(X_train, y_train, batch_size=30, epochs=5,
    validation_split=0.3, callbacks=[t$
126
127               logging.info("Saving the model")
128
129               model.save('saved_model/model_'+NAME)
130
131               logging.info("Evaluating the model.")
132               val_loss, val_acc = model.evaluate(X_test, y_test,
    batch_size=30)
133
134               print("val loss ", val_loss)
135               print("val accuracy", val_acc)
136
```

```
137
138
139
140
141
142
143 if __name__ == "__main__":
144     app.run(main)
```

A journey of a thousand miles begins with a single step.

Lao Tzi,
Chinese proverb