Master's thesis

Espen Taftø Vestad

Exploring siamese neural networks for similarity detection in fraudulent websites

Master's thesis in Information Security
Supervisor: Jan William Johnsen
Co-supervisor: An Thi Nguyen
June 2023

Espen Taftø Vestad

Exploring siamese neural networks for similarity detection in fraudulent websites

Master's thesis in Information Security
Supervisor: Jan William Johnsen
Co-supervisor: An Thi Nguyen
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Dept. of Information Security and Communication Technology

Preface

I started my journey in cybersecurity in the autumn of 2018 when I applied for a bachelor's degree in *IT-operations and Information Security* at the Norwegian University of Science and Technology in Gjøvik. Although I had no previous experience with cybersecurity, my interest started to grow during my bachelor's period. Ethical hacking, digital forensics, and cyber threat intelligence became the most interesting fields of profession. The growing interest led me to approach the digital forensic track at NTNU´s master's degree in *Information Security*.

This thesis concludes my master's degree. It was completed during the spring semester of 2023. The chosen topic branched off an initial topic proposal regarding the detection of replicated criminal websites, during a project planning course in autumn 2022. Approaching the task with a machine learning methodology was not initially the plan, as I had no experience within this field on beforehand. After assessing existing work and feasible methods, it became clear that using neural networks was indeed an interesting approach for this type of topic.

The thesis gives a very general and friendly introduction to machine learning, neural networks, siamese nets, and other related concepts. Especially those interested in digital forensics and criminality disruption may find the thesis interesting. Although not required, the reader would benefit from being familiar with concepts of neural networks before reading.

Espen Taftø Vestad
29.05.23

Acknowledgements

Firstly, I would like to thank my supervisor Jan William Johnsen for excellent support, feedback, and a critical mindset during the semester. He provided great reflection and input which have been very valuable for my thesis completion. I would also like to thank my co-supervisor An Thi Nguyen for sharing valuable feedback and knowledge about neural networks and PyTorch. The technical achievements would not have been significant without her.

Furthermore, I would like to thank my classmate Lars for his assistance in complex debugging problems. He, along with my classmates Eirik and Ludvig, also provided great company and fun times throughout the semester. Lastly, I would thank the people maintaining NTNU SkyHigh for providing an efficient and powerful infrastructure, that made implementation swift and joyful.

Abstract

Growing digitization and communication across international borders has allowed cybercriminals to operate more advantageously than law enforcement on the internet. Law enforcement suffers from a complex and time-consuming process to take down criminal website campaigns. In contrast, criminals scale up their operations by replicating instances of fraudulent websites with minor effort. When replicated, similarities often persist between the original website and the replicated copy. Digital forensic investigations could exploit these persisting similarities to detect replicated copies and further cause disruption.

Through this thesis, we propose a method that can contribute to the detection and disruption of criminal website campaigns. By using Siamese Neural Networks, we train a distance metric to compute and compare the similarity between screenshots of fraudulent websites. Previously developed machine learning methods within the field have mostly relied on text-based features, such as website source code, sentences, or HTML structure. The proposed method contributes to existing research by using deep learning to process screenshots as the input medium, without manually determining relevant features upfront. The thesis also evaluates how the network captures features through pixel attribution techniques.

The final result showcases a siamese neural network applicable for website similarity detection when trained in a bigger-scale scenario. In digital forensics, the approach could be utilized in automated solutions to enhance the detection and disruption of replicated criminal websites.

Sammendrag

Økende digitalisering og kommunikasjonstilgang på tvers av landegrenser har tillat cyber-kriminelle å operere med flere fordeler enn myndighetene på internett. En av myndighetenes hovedutfordringer er den tidkrevende prosessen tilknyttet nedstenging av kriminelle nettsteder. De kriminelle på sin side kan skalere sine operasjoner ved å kopiere eksisterende svindelnettsteder. Dette med minimal innsats. Når kriminelle nettsteder dupliseres, vil det ofte være likheter mellom den nye og den originale kopien. Digital etterforskning kan utnytte disse likhetene for å detektere gjennoppstående kopier av nettsider, og videre forårsake forstyrrelser i deres operasjoner.

Gjennom denne oppgaven foreslås metodikk som kan benyttes innen deteksjon og forstyrrelse for kriminelle nettsteder i drift. Ved bruk av siamesiske nettverk trente vi en distanse algoritme for å kalkulere likheter mellom skjermbilder av svindelnettsteder. Tidligere maskinlæringsmetoder innenfor feltet har i hovedsak tatt utgangspunkt i tekstbaserte egenskaper, slik som kildekode, setninger eller HTML-struktur. Vår metode bidrar til eksisterende forskning ved å prosessere skjermbilder av nettsider som input til *deep learning*. Dette uten å manuelt bestemme relevante egenskaper på forhånd. Oppgaven utforsker også hvordan nettverket lærer relevante egenskaper gjennom pikselattribusjon.

Gjennomførte eksperimenter resulterte i et siamesisk nettverk i stand til å detektere likheter mellom nettsteder dersom modellen trenes i et større scenario. Innen digital etterforskning kan modellen eksempelvis benyttes i automatiserte løsninger for tidligere deteksjon og forstyrring av dupliserte kriminelle nettsteder.

Contents

Figures

Tables

Code Listings

Acronyms

ANN artificial neural network. 8, 12–16

CNN convolutional neural network. 4, 7, 8, 14–18, 21, 22, 25, 35, 37, 42, 56, 66, 69, 72

CSS cascading style sheets. 7, 24

DL deep learning. 7, 8, 10, 11, 14, 21, 24–26, 44

DOM document object model. 24

HTML Hypertext Markup Language. 24

HYIP high-yield investment program. 1, 23, 27–30, 35, 47, 48, 71

JS Javascript. 7

NN neural network. 2, 4, 7, 8, 11, 12, 14, 17, 21, 22, 26, 35, 41, 43, 65–67, 70

SNN siamese neural network. 2–5, 7, 11, 13, 17, 18, 20, 24–27, 29, 31, 36, 37, 39, 41–45, 56, 65–67, 69–71, 73

TLDH top-level domain hopping. 1–3, 70

WCE web-content extraction. 24

Glossary

anomaly detection Identification of patterns that deviates from normal behaviour. 9

batch size Determines how many samples there should be within one batch of data passed through the network. 44, 45

big data Increased volume of data with high variety and velocity. 2, 11

classification Machine learning method for labelling unknown data to predefined classes. 9, 65, 70

clustering Algorithm type for grouping similar datapoints together. 8, 9, 68, 70, 72

digital forensics The process of identifying, collection, analysing and presenting digital evidence. 2–4, 69–71

epoch The amount of epochs determines how many times the network should iterate over the dataset. 44, 45

escrow Third-party contractor responsible for handling high-value transactions and transportation between buyer and seller. 1, 23, 24, 27, 30, 47, 71

gamma Influences the learning rate by multiplying it with a fixed value for each iteration. 44, 45

hyperparameter Values, variables or parameters that control the learning process in machine learning. 45, 63

k-fold cross validation Splitting training and testing datasets into *k* number of *groups* to evaluate models against new data.. 72

learning rate Determines step size for each iteration (epoch) and how fast weights are updated in the network. 44, 45

machine learning A method that allows machines to automatically solve problems, make predictions or find patterns by learning from data. 2, 4, 7–12, 21, 23, 26, 65, 69, 70

optimizer Optimizes network performance by taking weights, learning step and attributes into consideration. 45

overfitting Happens when a model has learned the pattern of the data too well, resulting in low performance for unseen samples.. 11, 38, 53, 55, 67, 72, 74

regression Machine learning method for predicting continuous values. 9

reinforcement learning A machine learning method where the model is trained to make decisions in a specific environment. 9, 10

saliency map A visualization technique in computer visioning showcasing how different parts of the image contribute.. 42, 56, 58, 59, 66, 71, 72

scheduler Adjusts learning rate as the amount of epochs increases (network progression). 45

spider Similar to web crawler. Automatically visits and gathers data on websites of interest.. 7

supervised learning A machine learning method where properties of the training data is known on beforehand. 9, 69

synapse The connection (vertices) between neurons (nodes) in a neural network. 12

top-level domain The highest type of domain in the DNS hierarchy after the root domain . 1

transfer learning Utilizing a pre-trained model as an entry point for solving a new machine learning problem. 22, 35, 60, 67, 69–72

underfitting Happens when a model does not have enough baseline data to generalize well, resulting in struggles to improve in performance.. 11, 38, 54, 55, 71, 72

unsupervised learning A machine learning method where properties of the training data is new/unknown on beforehand. 9

web crawling Automatically visiting webpages of interest to gather data or content. 7

Chapter 1

Introduction

1.1 Topics covered by the project

The utilization of websites and web applications to host fraudulent campaigns has become a way for criminal operations to reach their target audiences and for ease of operation [1]. Being able to host these operations digitally, managing and distributing criminal activity in size thus becomes more feasible. Law enforcement and other authorized entities have for a long time practiced website take-downs to disrupt criminal activity online [2].

As a countermeasure, criminals may utilize techniques for their operations to remain persistent, as for instance *top-level domain hopping (TLDH)* [3]. This technique implies that the criminals publish near-duplicates of their disrupted websites under a different top-level domain. These replicated services may have new hosting details while retaining their branding and reputation [3]. Moreover, replication allows criminals to efficiently spawn new services or run multiple campaigns simultaneously.

high-yield investment programs (HYIPs) and fake escrow-services are some of the common campaigns where criminals practice website replication [4]. By creating fake HYIP websites, scammers can lure victims to invest, further "guaranteeing" high-yield returns within a short time frame. Similarly, fake escrow agencies could defraud both buyers and sellers of lucrative products by stealing their transaction deposits.

Digital forensics is the process of identifying, collecting, analyzing, and presenting digital evidence in a forensically sound manner [5]. Proper seizure of digital evidence is necessary for law enforcement in order to issue legal action against criminal websites [2]. The process of detecting a criminal website's existence, followed up by issuing legal actions, is a time-consuming process [2]. Time is hence of the essence in order to detect and fully remediate the operations of these criminals.

The scale and velocity at which replicated websites appear makes manual intervention infeasible. Machine learning could aid investigators in the digital forensics process. Rapid growth of data gathering and transportation ("big data") has also increased the complexity in machine learning problems [6]. The high processing speeds and learning ability of neural networks (NNs) [7] may especially be useful to solve these bigger scale problems. One example is siamese neural networks (SNNs), which are commonly used to measure similarities in computer vision tasks. With SNNs, both the website detection and intelligence gathering process could perhaps increase investigation efficiency and evidence confidence.

Since certain types of fraudulent websites tend to be replicated, similarities between these often persist. It may be possible to exploit persisting website similarities to strengthen the detection of similar and duplicated websites. Thus, through this thesis, we propose a SNN machine learning method to compute similarities between fraudulent websites.

1.2 Keywords

Siamese neural networks, machine learning, deep learning, website replication, website takedown, digital forensics.

1.3 Problem Description

Website replication combined with other techniques such as TLDH allows for cybercriminals to maintain their operations more efficiently over longer time periods [3]. By persisting their campaigns on several top-level domains, a single takedown inquiry may only disrupt a portion of their operations. Disruption can also be influenced by the methodology used for hosting criminal websites. Hosting can for instance be possible through compromise of a legitimate website, through free and less regulated hosting providers, or hidden services only accessible in anonymity networks [8]. Community-specific keywords, actions, or restricted paywalls are also utilized by criminals to maintain anonymity [9], and may further make disruption harder.

Moore and Clayton examined the effects of taking down phishing websites [8]. Their findings described how takedown is helpful for disrupting criminal operations, but it will not be possible to achieve complete mitigation due to the process' time consumption [8]. Despite that great efforts can be made to shut down these services, cybercriminals may operate more advantageous than law enforcement in terms of resources required to spawn near-duplicate websites. A law enforcement takedown process involves requesting evidence preservation, obtaining a warrant, and seizure conducted by the local police [2]. Cross-jurisdictional delays in the process increase the time consumption, allowing criminals to scale operations. Foreign takedown requests may require further action by local authorities, which may experience shortages in available resources, lack of expertise, and high costs tied to the investigation [2]. The takedown process from start to finish is thus more tedious and demanding compared to the effort needed by cybercriminals to replicate a campaign.

1.4 Justification, Motivation, and Benefits

The fact that cybercriminals operate more advantageously than law enforcement motives the need for further disruption activities. For earlier detection and faster response against the spawning of criminal, replicated websites, the utilization of automated methods can aid in the process. Furthermore, this can contribute to enhancing the ability to detect TLDH and other persistent techniques at an earlier stage. This could for instance be achieved by calculating website fingerprints or measuring similarities between TLDH websites.

As mentioned previously in Section 1.3, the process for law enforcement to inquire takedown requests depends on several steps and cross-jurisdictions which increases complexity. Since criminals profit from replicating content across domains [8], detection of duplicated content could be used against them to disrupt their operations. Thus, criminals have to put more resources into developing new websites. By slowing down the criminals, they may operate less advantageous to law enforcement compared to what they do as of today.

Some website elements can change during website replication, such as color, structure, images, or fonts. For criminals to achieve the maximum level of up-time in their operations, however, developing unique websites for each takedown is not feasible or scaleable. Thus, several website features, for instance, those mentioned above, will often remain throughout replication [4]. Research within the field of criminal website replication could benefit several types of entities or corporations. Corporations could find the research relevant to discover fake competitive website clones, while organizations or law enforcement could be more motivated by justice, ethics, or human rights [2]. Security researchers and digital forensics investigators could benefit from the intelligence gathered from detection of these websites.

1.5 Research Questions

The main focus area of this thesis is motivated by the detection of criminal websites based on similarities in their characteristics. This further emphasizes evaluating the use of siamese neural networks for this purpose, in addition to interpreting how features are captured. More specifically, the project aim seeks to answer the following research questions through conducted experiments:

- **RQ1:** How precise are siamese neural networks in identifying similarities between screenshots of fraudulent websites?
- **RQ2:** How can the trained model be interpreted to analyze relevant features learned from fraudulent website screenshots?
- **RQ3:** What are possible similarity thresholds for determining whether pairs of websites are dissimilar, similar, or near-duplicates?

RQ1 will explore whether siamese neural networks are appropriate in wesbite similarity detection. Screenshots of fraudulent websites will be used as the input medium to explore this. **RQ2** evolves around interpreting the trained SNN model to assess its ability of capturing features. Lastly, **RQ3** will consider potential threshold values for determining how the SNN should define similarities.

1.6 Contribution

By exploring the above research questions, the thesis aims to contribute with input in the field of detection and disruption of fraudulent website campaigns. The use of SNNs for this purpose has, to our knowledge, not been explored previously. This thesis will contribute towards determining the usefulness of SNNs for this purpose. Our contribution could be useful for digital forensics investigators and security researchers to strengthen the disruption of criminal websites in further work.

1.7 Thesis Outline

This thesis is divided into several main chapters with appropriate sections. Fundamental concepts related to the thesis will be discussed in the background chapter. This aims to ensure a proper understanding of the underlying concepts used in the project's methodology. Previous related research will be briefly discussed to assess already explored topics in the field. The developed methodology is further presented, along with conducted experiments and achieved results. These are lastly discussed before the thesis is concluded. Suggestions for further work are also presented.

- **Chapter 2 - Background:** Covers fundamental topics relevant to the thesis. Concept explanations are supported by illustrations and external sources. Covered topics include machine learning theory, neural networks, and convolutional neural networks.
- **Chapter 3 - Relevant Work:** Assesses existing research with respect to this thesis' focus area. This includes juridical challenges tied to criminal website disruption and previously explored machine learning methods.
- **Chapter 4 - Methodology:** Describes methods used to implement the thesis' technicalities. This includes a description of data gathering, population, and

SNN development.

- **Chapter 5 - Experiments:** Presents achieved results from conducted experiments. Figures and tables will be utilized to support these descriptions.
- **Chapter 6 - Discussion, conclusion, and further work:** Obtained results and experiments conducted in Chapter 5 will be further discussed. Through discussion, this chapter aims to describe results on a higher level along with their meaning. Research questions will also be addressed in this chapter with respect to achieved results. Lastly, the thesis is concluded along with suggestions for further work.

Chapter 2

Background

The following Chapter will address theoretical material for underlying concepts relevant to the thesis. Initially, concepts within the field of website crawling and machine learning will be described. Then, the chapter proceeds with describing the workings of deep learning (DL) and neural networks (NNs). convolutional neural networks (CNN) will further be introduced. Lastly, the concept of siamese neural networks (SNN) will be described along with common parameters. The machine learning framework utilized in the thesis methodology will also be briefly touched upon.

2.1 Website crawling

Web crawling is an automated process that systematically visits or downloads contents from webpages [10]. These crawlers are also referred to as spiders, and is often used within web search engines and website archiving technology. In addition, web crawling can be convenient to automate data gathering from a bigger set of websites. This data could for instance be website content, images, screenshots of web pages etc.

Some crawlers are also designed to simulate human interaction with websites [11], presenting crawled content as it would appear to human beings rather than machines. One of the most common examples of this is *Selenium* [1], which is a web driver that can automate human-like interactions with websites. Selenium can be utilized by many programming languages, including Python and JavaScript. Selenium can also perform other actions on websites, such as modifying local CSS and Javascript, input assertion, and capturing screenshots. These features will be relevant for the data collection process as described in Section 4.2.1.

[1]`https://www.selenium.dev/`

2.2 Machine Learning

Machine learning is a method that allows a system to automatically make decisions while improving performance through experience [6]. The use of machine learning is often preferred when you process bigger amounts of data when it is infeasible to conduct manual decision-making. Outsourcing the task of decision-making and solving complex problems to the machine can hence improve efficiency with minimal human intervention [12][13].

In practice, machine learning is conducted by utilizing an algorithm to train a model. Based on how well this model is trained, it will be able to conduct predictions and problem-solving of unknown data [6]. One of the most important goals of a machine learning algorithm is to achieve as high accuracy as possible while maintaining feasible computational costs.

As the use of machine learning has progressed dramatically during the past two decades [6], several areas of and subsets machine learning have been developed accordingly. Figure 2.1 shows a conceptual distinction between the different areas in the machine learning field. The more traditional machine learning methods consist of techniques such as decision trees and clustering methods. artificial neural networks (ANNs) borders between traditional learning and deep learning (DL). This is because ANNs can be both deep fully-connected networks or have a shallow architecture. DL refers to NN algorithms having *deeper* architectures with many layers in the network [14]. Commonly, CNNs are referred to as *deep NNs*.

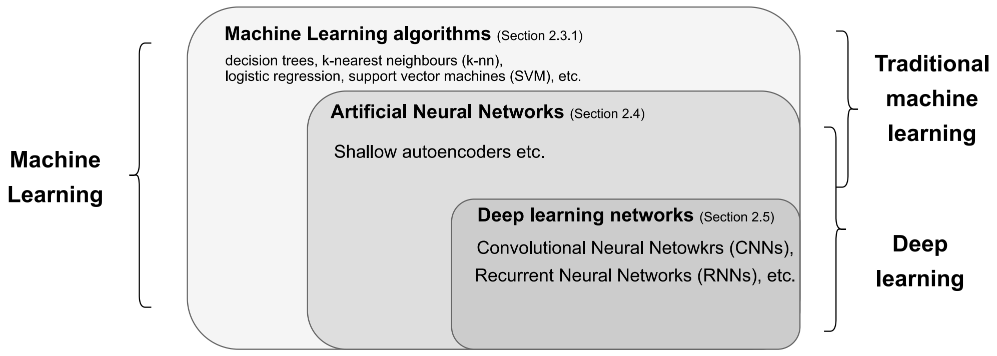

Machine Learning algorithms (Section 2.3.1)
decision trees, k-nearest neighbours (k-nn),
logistic regression, support vector machines (SVM), etc.
Artificial Neural Networks (Section 2.4)
Shallow autoencoders etc.
Deep learning networks (Section 2.5)
Convolutional Neural Netowkrs (CNNs),
Recurrent Neural Networks (RNNs), etc.
Machine Learning
Traditional machine learning
Deep learning

Figure 2.1: An abstraction of layers in the field of machine learning, inspired by C. Janiesch & P. Zschech [12].

2.2.1 Traditional machine learning

Traditional machine learning models typically rely on simple algorithms that are easy to interpret and train. The way these algorithms learn from and adapt to new data is different based on the type of learning. Machine learning models are commonly divided into three main types of learning scenarios [15]: Supervised learning, unsupervised learning and reinforcement learning.

Supervised learning is the most widely used category [6] and refers to the scenario where a model is trained based on a dataset where the data's features are known beforehand [15]. This requires more preliminary work, as each entry in a dataset must be associated with relevant features and classes before training can start. If the training data is biased or not properly representative of the population, the model may perform badly towards unseen data [16]. On the other hand, supervised machine learning may perform very well when built upon reliable, high volumes of training data. Performance and accuracy are also easy to evaluate as they can be linked to already labeled training data. Common supervised learning methods include classification and regression [6]. Figure 2.2 illustrates how classification and regression works in practice. Classification aims to distinguish and label samples based on their belonging. Regression is utilized for predicting continuous values.

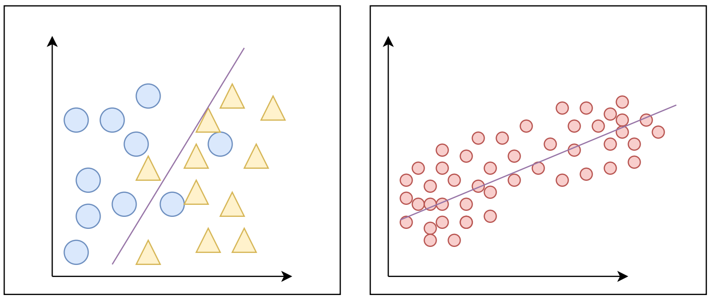


Figure 2.2: Supervised learning methods. The example to the left illustrates classification, while the rightmost example illustrates regression.

In unsupervised learning, the model is trained on unlabelled (unknown/unseen) data [6], with the goal of finding new patterns or structures [17]. Unsupervised learning has an advantage considering that preliminary knowledge about the training data (features) is not a requirement. This makes the method suitable to detect anomalies in unknown data or discovering unknown patterns not possible to identify through supervised learning [18]. It can be harder to determine an unsupervised model's accuracy since data is not labeled before training [15]. Common unsupervised learning methods include clustering and anomaly detection. Clustering in order to detect potentially criminal websites has been researched previously and is further mentioned in Chapter 3. Figure 2.3 visualizes clustering through an illustrated example.

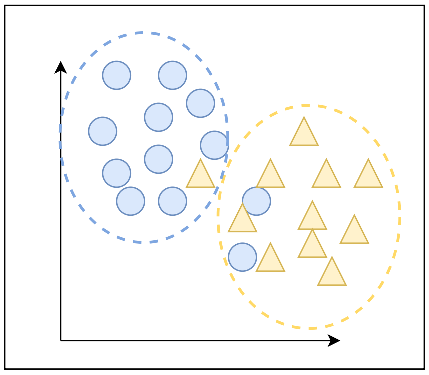


Figure 2.3: Illustrative example of clustering.

Machine learning may also utilize reinforcement learning techniques. This method is often used in circumstances where decision-making within a specific environment is desired [15]. The goal is to maximize some reward, where the model learns through trial and error [6]. Thus, the model is dependent on positive feedback (rewards) or negative feedback (penalties) as input for learning how to interact with its environment [17]. Figure 2.4 illustrates the overall workings of reinforcement learning.

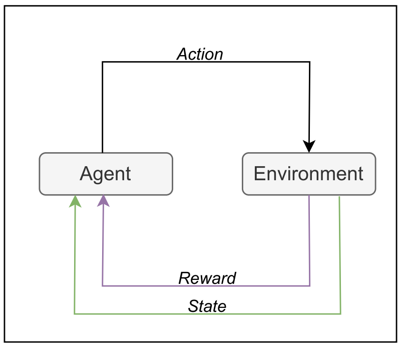


Figure 2.4: Illustrative example of re-enforcement learning. The agent (model) performs some action and receives rewards or penalties as a result.

What mainly characterizes classic machine learning is the use of low complexity algorithms which makes the model and result easy to interpret [19]. In addition, classic machine learning models require fewer amounts of data in order to be trained, compared to complex DL architectures. These factors also make classic machine learning models cheaper to train in terms of computational consumption. On the other hand, classic machine learning often rely on high-quality datasets which can be intimidating to create, while DL are able to automatically learn features without knowing about these beforehand [18].

2.2.2 Deep learning

As the concept of machine learning and its use cases have progressed dramatically during the past two decades [6], so has the amount of data to process by the machine learning models [19]. The growth of networked mobile and computer systems and visual mediums has increased the production and transport of data, further introducing the term *big data* [6]. Bigger amounts of data require more complex algorithms with more parameters in order to maintain performance [19] in contrast to conventional machine learning methods. DL models can be utilized to cope with complex and bigger amounts of data. They are built on a concept known as neural network (NN). DL models are especially useful in situations where relevant features are not known beforehand, as neural networks are able to process features from raw data on their own [19].

2.2.3 Metric Learning

Metric learning is a machine learning technique for learning distance functions [20]. This is achieved by comparing samples to each other. By measuring the distance between two computed points (embeddings) using this metric, the resulting output represents the semantic similarity between them [21]. In other words, metric learning can be used to identify the relationship between the samples. A siamese neural network is a metric learning algorithm, computing the similarity between two embeddings. The workings of SNNs are further described in Section 2.5.

2.2.4 Overfitting and underfitting

A machine learning algorithm is trained to fit predictions of input data as well as possible [16]. Sometimes, depending on how it is trained, the model may struggle to conduct accurate predictions. Two terms are commonly used to evaluate the model´s fitness: Overfitting and underfitting. Overfitting happens when a model has learned the underlying patterns of the training data too well. It makes the model too dependent on the training data [17], causing it to struggle with predicting unseen data accurately. Underfitting happens when the model has not been able to generalize well enough to the training data. Training data is thus not representative enough to make the model learn, which could lead to higher bias [16]. A model is fit when neither overfitting nor underfitting is significant.

2.3 Neural Networks

Neural Networks (NNs) are a type of machine learning method that is inspired by the workings of the human brain [7]. The neurons are where the learning takes place. However, one neuron on its own is not particularly powerful [7]. When more neurons are fully connected, we have a better ability to learn. Neurons are thus connected to each other through dendrites and axons. The dendrites work as receivers for signals from other neurons, while the axons work to transmit signals to other neurons. The connections between the neurons (dendrites and axons) are also referred to as synapses, which is a more commonly used term in machine learning. Figure 2.5 shows a representation of how neurons are connected in the human brain. artificial neural networks (ANNs), the machine learning implementation of a NNs, are illustrated in Figure 2.7 and further described in Section 2.3.2.

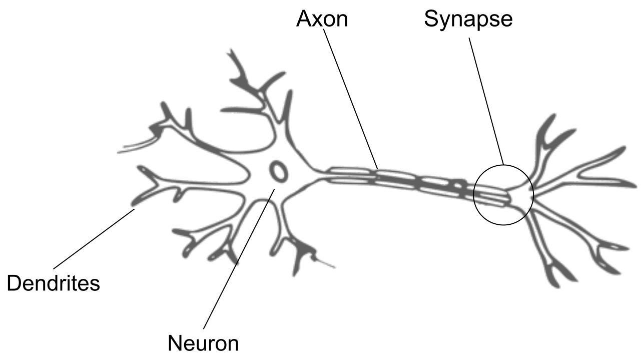


Figure 2.5: Illustration of how neurons in the human brain are connected.[2]

In machine learning neurons may also referred to as *nodes*. These nodes receive multiple input signals from n amount of dependant variables [7]. The signals are thus processed in their respective receiver nodes, which output a value sent further down the chain. This value will then be passed on as input to other nodes in the network. The inputs in which the node processes will also have weights. These weights help the neural network learn [7]. It is accomplished by adjusting the weights along the way based on how relevant the input is. Each node will consider the sum of all input weights and apply the *activation function*, which will determine the output that is passed further [22]. The results of this function determine the relevance of the output that is passed on to receiving neurons.

[2]Original image by OpenClipart-Vectors. License: Copyright-Only Dedication

2.3.1 Activation function

The activation functions determine the value to pass further down in the network, which can also influence *if* the output should be passed at all [22]. The type of activation function chosen can greatly affect a neuron's output. For instance, using a simple *linear* activation function given by $F(x) = ax$ will provide an output that is directly proportional to the input [22]. The main drawback of the linear activation function is improving from error since the gradient remains constant through each iteration. It also struggles with complex data patterns, making it more suitable for simple problems [22].

Another common activation function used in ANNs is the *ReLU function*, given by $F(x) = max(0, x)$. ReLU is efficient in terms of low computational complexity, as only a certain number of neurons is activated at the same time in the network [22]. A neuron will thus be deactivated only when a linear transformation outputs zero.

The *sigmoid function* is another activation function that allows for smoother output values. The function, given by $F(x) = \frac{1}{1+e^{-x}}$, progresses more gradually compared to the prior examples, providing output values between 0 and 1 [22]. This makes it more applicable to probability calculation. *Sigmoid* is the activation function used in this thesis' SNN implementation.

Figure 2.6 illustrates an example for each of the activation functions discussed above. There also exist other types of activation functions for various purposes, for instance *threshold activation*, *Binary Step*, and *Tanh*.

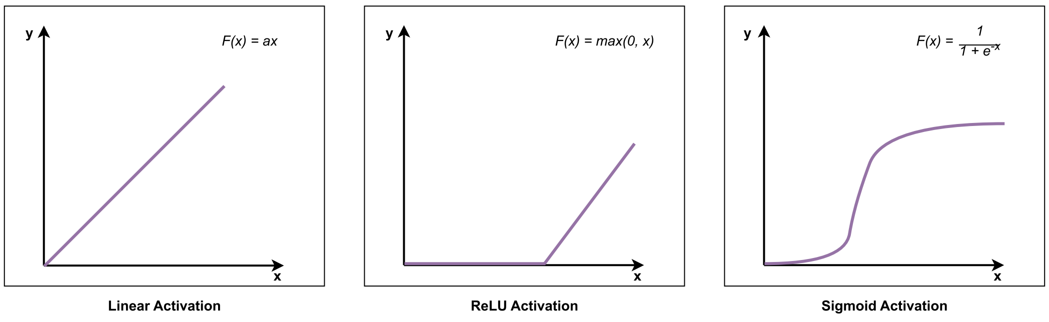


Figure 2.6: Some common neural network activation functions: Linear, ReLU, and Sigmoid.

2.3.2 Hidden layers

Fully-connected ANNs can be layered into three types of layers: The input layer containing the dependant input variables, the hidden layers, and the output layer. Several hidden layers are usually present in the network, where the neurons are connected to each other through weighted vertices [7]. Figure 2.7 shows an example of an ANN with three hidden layers, having four neurons in each layer. The weighting of an input towards a neuron determines whether the input is relevant for the neuron in question [22]. The chosen activation function will be applied to each neuron, which all considers different input values. Together, the neurons in the hidden layers will provide relevant output to the output layer.

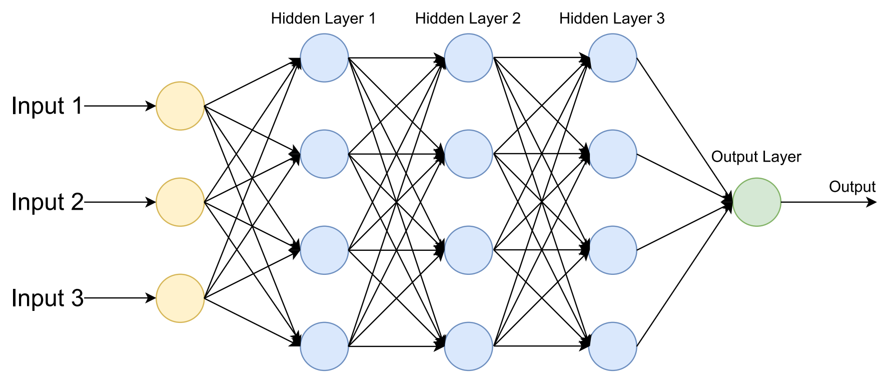


Figure 2.7: A fully-connected ANN with an input layer, 3 hidden layers, and an output layer.

2.3.3 Backpropagation and cost calculation

When the output value is calculated in the network's output layer, the model keeps track of the output value's relevance. The output value is thus compared to an absolute value, which is the desired outcome [17]. Comparison is conducted through a *loss function*, which calculates a loss (cost) value. A low loss represents an accurate value, whereas a high loss is inaccurate. There exist several loss functions, and which one to choose depends on the application.

When loss is calculated, the network initializes *backpropagation of errors* [17]. The goal is to keep the loss as low as possible. Thus, the NN is propagated backward and the weights between the neurons are updated accordingly. During the next iteration, the NN will process the next entry of data, but with updated weights in the network. At the end, a new output and loss value will be calculated and weights are updated. This process is repeated until the achieved accuracy is as high as possible.

2.4 Convolutional neural networks

convolutional neural networks (CNNs) are another type of deep learning neural networks commonly used to train models for image and video recognition. A CNN model focuses on extracting features from visual input, for instance, edges, colors, patterns, or textures [23]. Figure 2.8 illustrates a simplified CNN model, consisting of three types of layers [23]: The Convolution layer, pooling layer, and fully-connected layer. Moreover, the CNN architecture can be sorted into four phases:

1. **Input layer:** Represents the input image(s) in pixel value.
2. **Convolution:** Where the CNN learns features of the input data using *feature maps*.
3. **Pooling:** Downsampling of *feature vectors* received as input from the convolution layer.
4. **Fully-connected layers:** A fully-connected ANN that can compute probabilities, classifications etc.

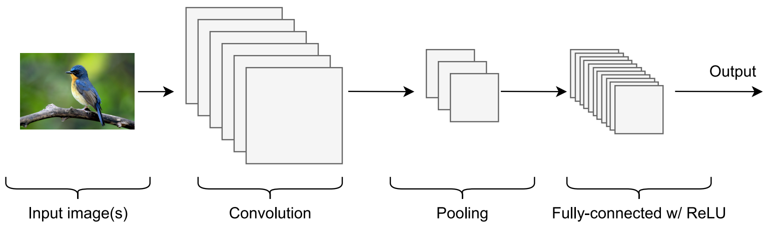


Figure 2.8: Illustration of a simplified CNN architecture[3], inspired by K. Shea & R. Nash [23].

2.4.1 Convolution

During convolution, the image is processed through one or more convolutional layers. Convolution layers produce a 2D activation map by convolving each filter across the spatial dimensionality of the input [23].

Feature filters are applied to the layer(s), which is used for extracting features from the input image. The *feature filter* is a $n \times n$ grid of random numbers. A *feature map* will be calculated using the input vector and the feature filter. This is done by calculating the dot product of a block from the input vector and the feature filter [24]. The result further produces the feature map. This continues until every block of the feature filter's size in the input vector has been considered. The process is further illustrated in Figure 2.9. It is important for the CNN to create smaller *feature maps* from the input images to improve the efficiency of the process, as convolution reduces complexity through output optimization [23]. *Feature maps* produced in one layer will be passed on as input to the next layer in the process.

[3]Bird photo by Karen Arnold, CC0 1.0 Universal

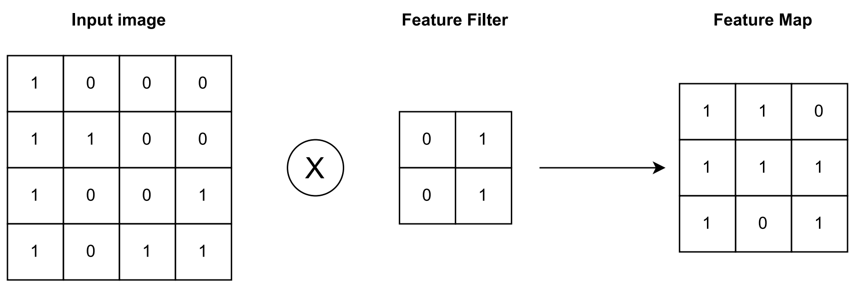


Figure 2.9: Simple, illustrative example of how a feature map is produced from an input vector through convolution.

2.4.2 Pooling

To reduce dimensionality of the data, the feature map is down-sampled (pooled) [23]. Several pooling methods exist, for instance, *max pooling* or *average pooling*. Perhaps the most common type of pooling is max pooling, which looks at a $n \times n$ block of pixels in the inputted feature map and only considers the biggest pixel value in that block. This value is inputted towards a new, *pooled* feature map. Figure 2.10 illustrates an example of how pooling is conducted. With the pooling technique, the CNN is able to greatly reduce computational complexity while still preserving the most relevant features in input data [23]. This makes it easier to pick up variance in the input since the pooled feature map has less noise.

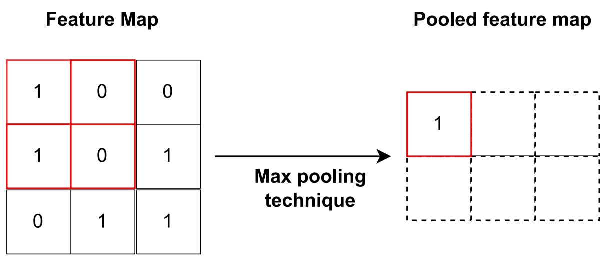


Figure 2.10: Example of how a pooled feature map takes shape from a feature map input using the *max pooling* technique.

When the dimensionality of the feature map is reduced, the pooled feature map(s) is normalized and flattened. Flattening the feature map involves converting the multi-dimensional map into a single row of values.

2.4.3 Fully-connected layer

The fully-connected layer is structured like an ANN [23], receiving input from the previously discussed layers. This is where the network conducts its final prediction. As described in Section 2.3.3, the amount or error (cost) in the prediction will be calculated, followed up by backpropagation which adjusts the weights accordingly.

2.5 Siamese Neural Networks

A siamese neural network is a metric learning algorithm that combines the usage of two or more identical NNs to calculate the similarity between outputs. Although a SNN could be used to calculate similarities for various mediums, the most common use case is perhaps similarity measurements between images using two or more CNNs. These networks share the same weights and parameters [25][26]. A distance value is further calculated using the feature vectors from each CNN. This is achieved using a distance calculation algorithm (i.e. cosine similarity). A sigmoid function is commonly applied to retrieve similarity values between 0 and 1. Figure 2.11 shows a general example of a SNN.

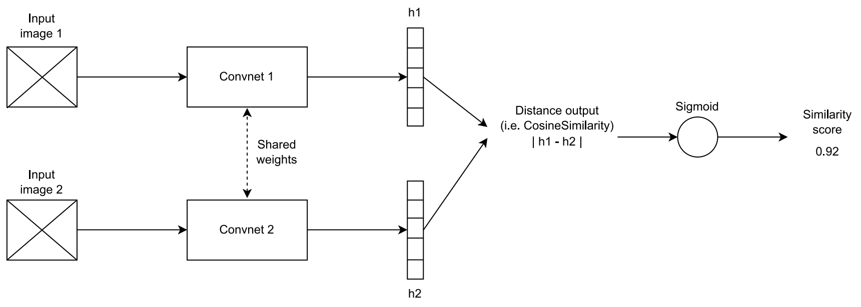


Figure 2.11: Illustration of a Siamese Neural Network using two CNNs as subnetworks. Similarity scores are outputted by calculating cosine similarity and passing results to the sigmoid function.

Although NNs can solve a variety of complex problems, huge amounts of data are often necessary in order to maintain representative models. SNNs are often utilized with methods such as *one-shot* learning, which allows for learning with limited amounts of data available [27]. SNNs is also particularly useful when it is not possible to receive exact classification from unknown data, but assigning the appropriate classification based on how similar the sample in question is [25][27].

2.5.1 Common loss functions

SNNs are often based on pairwise learning. Thus, SNNs and metric learning algorithms often utilize loss functions for pairwise learning [21] rather than traditional classification loss functions (i.e. *CrossEntropyLoss*). Perhaps the most common loss functions used for SNNs are *ContrastiveLoss* and *TripletLoss*. ContrastiveLoss is often utilized to learn embeddings that preserve important properties of the input data [28]. The function can further minimize the distance between similar outputs while maximizing it for dissimilar outputs. This makes the loss function suitable for similarity calculation in SNNs rather than classification loss functions.

TripletLoss is similar to ContrastiveLoss in the way the notion of similarity between inputs is calculated. In TripletLoss however, similarity is considered between triplets of input rather than pairs. This process is further showcased in Figure 2.12. The triplet consists of the *anchor* image, the *positive* image, and the *negative* image. The anchor and the positive are different images from the same class, whilst the negative image is from a separate class. In this case, all three images will be passed through their respective CNN sub-networks before being passed to the TripletLoss calculation. The goal of the TripletLoss function is to minimize the distance between the anchor and the positive while maximizing distance between the anchor and the negative [29][30].

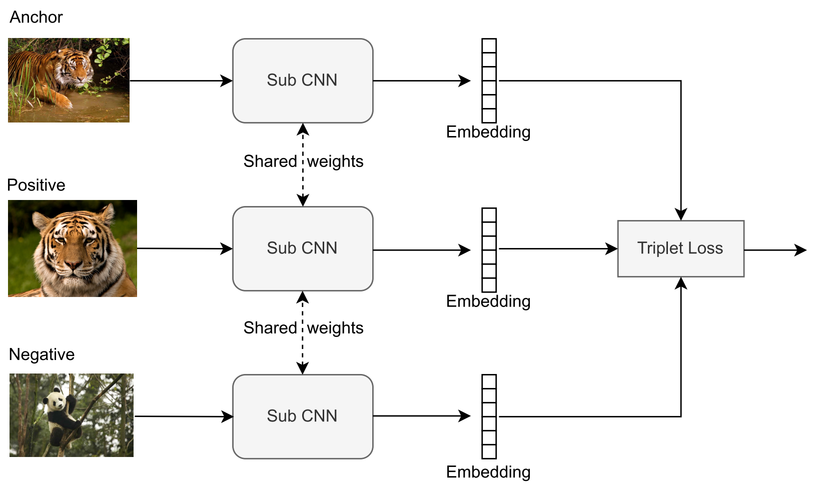


Figure 2.12: Illustrative example[4]of how Siamese networks can utilize triplet loss. The positive and negative are compared to the anchor image.

2.5.2 Training a Siamese network

Training a SNN is commonly performed by selecting pairs or triplets of images, which are fed as input to their respective CNN sub-networks [26]. Figure 2.12 shows an example of how this is done using triplets. Generated pairs or triplets can for instance receive a *target* on whether they are similar or not [31]. If both images from a pair originate from the same class, their target could for instance be 1 (similar). Otherwise, they could be labeled as 0 (dissimilar).

Input will be parsed through the sub-networks, outputting one embedding for each. These embeddings will further be processed for distance or similarity calculation [25]. Embeddings could for instance be concatenated before being passed to the loss function, which calculates the loss value. Another common step in the

[4]Anchor image by Moni Sertel, CC BY NC SA 2.0
[4]Positive image by S. Taheri, edited by Fir0002, CC BY-SA 2.5, via Wikipedia Commons
[4]Negative image by Martha de Jong-Lantink, CC BY ND 2.0

process is to pass the output of the distance embedding calculation to a sigmoid function. As touched upon in Section 2.3.1, passing the output through the sigmoid function allows for retrieving a similarity score between 0 and 1 [22], which could be more convenient to work with.

Lastly, back-propagation as discussed in Section 2.3.3, is applied to update the weights in the model's sub-networks. As shown in Figure 2.11 previously, weights and parameters are shared between the sub-networks [25][26]. This training process continues until the desired amount of epochs is reached.

2.6 Few-shot and one-shot learning

In scenarios where available training data is limited, it could be difficult to train a representative model of the problem at hand. Techniques such as *few-shot learning* can be utilized to cope with the lacking amount of training data. Humans are able to learn only from one or a few samples of data [32]. The concept of few-shot/one-shot learning applies the same principle to machine learning.

2.6.1 Few-shot learning

Few-shot learning is sometimes referred to as **N-way-K-shot-classification** [20]. *N* represent the amount of classes, while *K* represent the amount of samples per class. A few-shot learning approach could for instance consist of a dataset with $N = 10$ (10 classes) and $K = 5$ (5 samples within each class). For each class, the algorithm has "5 shots" at conducting the classification. Few-shot learning may also be referred to as *N-shot learning* [20], depending on how many samples exist within each class.

2.6.2 One-shot learning

One-shot learning is similar to few-shot learning, except that each class only has 1 sample ($K = 1$). Since the algorithm only has "one-shot" at each class, learning and prediction are more complex compared to few-shot. When several samples are presented, the algorithm has more baseline for training/prediction.

As the number of choices (classes *N*) to make increases, the algorithm grows more complex [33]. Having fewer classes to choose from hence makes decisions easier. Similarly, the number of samples (*K*) within each class affects decision-making. The higher the value of *K*, the easier it is to identify new samples belonging to that class. To maintain good model accuracy, *N-ways* should be as low as possible, while *K-shots* should be as high as possible [33].

2.6.3 Support set and query set

Few-shot and one-shot learning problems are commonly approached using support sets and query sets [20]. A support set is a reference dataset with N classes and K samples per class. The query set contains the training and testing images that will be measured toward samples in the support set. A query image is queried, and similarity is calculated towards each of the support set classes. The highest similarity determines which class the image belongs to. Through this approach, the algorithm is learning how to learn to classify, rather than being told how to classify [20].

A practical example of where one-shot learning with support sets could be utilized is biometric authentication. When a user registers in the authentication system for the first time, the fingerprint is stored as a support set sample. Every time the user authenticates, the current fingerprint scan (query image) is measured towards the support set classes in the database. Figure 2.13 shows an illustrative example of few-shot learning using a support set in animal classification.

Few-shot and one-shot learning with support sets may also be used with SNNs. The implemented SNN in this thesis has not utilized this approach, however.

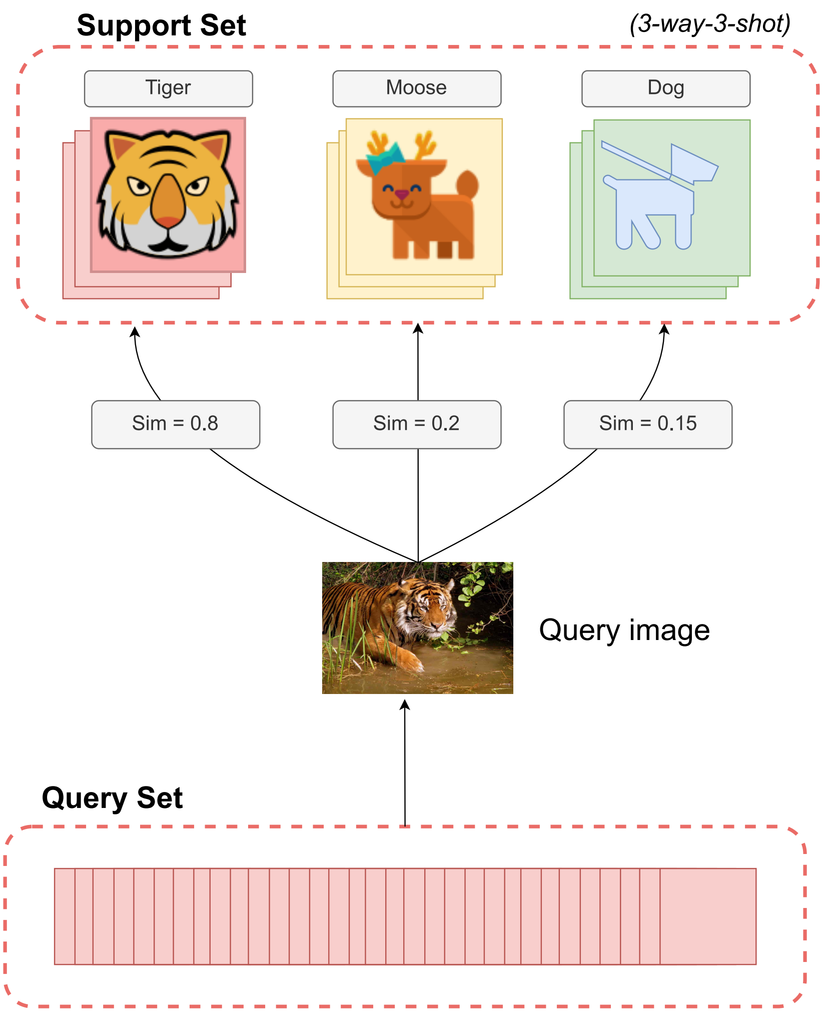


Figure 2.13: A few-shot learning example with 3 classes (N) and 3 samples per class (K), utilizing a support set[5].

2.7 PyTorch and neural network architectures

PyTorch is a machine learning framework for the Python programming language [34]. Although originally developed by Meta AI, the framework became part of the Linux Foundation in 2022[6]. The framework is well suited for the development of deep NNs, including CNNs. PyTorch may in many ways be similar to other deep learning (DL) frameworks, as for instance TensorFlow[7]. What mainly distinguishes PyTorch from other frameworks is simpler debugging and more intuitive

[5]Tiger photography by Moni Sertel. License: CC BY NC SA 2.0
[6]`https://pytorch.org/blog/PyTorchfoundation/`
[7]`https://www.tensorflow.org/`

correspondence to mathematical expressions [34].

In addition to ease of development of NNs, PyTorch can be utilized with several pre-trained models and weights [35]. This allows users to make use of already trained models when less data is available. Popular NN architectures are also available in PyTorch, as for instance InceptionV3, VGG and ResNet.

ResNet is a commonly used deep CNN for image recognition. Several versions of the architecture exist, for instance, Resnet-18 or ResNet-50. The number describes how many layers the NN consists of, making ResNet-50 an architecture with 50 layers total [36]. The more layers an architecture has, the higher the computational performance is needed. However, the more layers the network has, the more patterns, details, and features it may learn. ResNet architectures are designed to receive input images with a resolution of 224x224 pixels [36]. PyTorch offers the possibility to load pre-trained ResNet models. These models are trained on the ImageNet1k dataset [36], which is trained on 1000 different categories of images [37]. The pre-trained model's knowledge can be utilized in order to classify new types of images. This approach is also referred to as transfer learning.

Chapter 3

Related work

The following chapter will assess and discuss related work within the field of criminal website detection. Existing work, as discussed in this chapter, will provide supportive information regarding already existing experiments and research.

Detection and disruption process of replicated illegal websites is one of the core motivations for this thesis. Alice Hutchings et al. [2] highlights some of the challenges involved in these takedown processes seen from different entities' perspectives. Upon receiving a takedown inquiry, website registrars could find it challenging to establish the legitimacy of the request. Similarly, law enforcement may need to seize proper evidence on suspected illegal websites in order to proceed with a takedown [2]. There are also challenges tied to cross-jurisdictional issues with website takedowns, for instance, limitations of resources and competence at foreign law enforcement [2].

Nevertheless, time is of the essence in order to proceed with takedown actions while also disrupting and deterring further criminal activity. This fact is further emphasized by Moore and Clayton [8], in which their findings highlight that takedowns are indeed helpful for disruption. However, it will never be able to fully cope with the amount of criminal activity due to a takedown's time consumption. Utilization of machine learning models could increase earlier detection of criminal websites with less manual work.

Further research conducted by Jake M. Drew and Tyler Moore [4] reveals that similarities are often persistent within criminal websites. Their paper explores optimized clustering methods to rank features of relevance in criminal website detection. More specifically, Drew and Moore focused on *high-yield investment programs (HYIP)* and fake *escrow services*. HYIPs are *ponzi* schemes created to give the investor a high yield off an investment, often by exploiting other investor´s investments [38]. Scammers utilize HYIPs to steal money from unaware investors. An escrow service involves having a third-party handling the transaction and transportation between buyer and seller. Escrows are mostly utilized in high-value transactions that involve transportation of physical products [4]. Scammers create fake escrow services to trick sellers and buyers.

It was discovered by Drew & Moore that websites within these two categories respectively often had similar structure or content. Their experiments showed that website elements such as HTML DOM-tags, sentences, and file names have relevance in the detection of these websites. A combination of features sometimes had higher relevance. Screenshots of the websites in the experiment were also one of the features considered. These showed to have low relevance alone but worked well in combination with other textual features, such as sentences. This shows that the identification of criminal websites based on website screenshots may be challenging, without support from other textual features.

Text-based features have also been focused on in other experiments tied to criminal websites. Westlake & Bouchard et. al. [9] explores automated and manual classification of websites with *child exploitation (CE)* content. It is discovered that this type of criminal community utilizes specific keywords within their way of communicating. Keywords could be anonymous code words or community-specific slang. This shows that different types of criminal communities may use different types of language. Websites may hence have different types of similarities, depending on the website's theme.

Research conducted by Stanislas Morbieu et al. [39] explores methods for web-content extraction (WCE). Using WCE, they showcase extraction of a website's DOM, text and CSS-properties followed up by clustering of a webpage's *main content*. The article refers to a webpage's main content as the content giving relevant information to the end user. This could for instance be the text, image, or video of interest shown on the website. Semantic website elements are referred to as *noisy* content and include advertisements, navbars, sidebars, etc. In terms of detecting similarities in fraudulent websites, looking at different pieces of content could provide useful. As seen in Drew and Moore's paper [4] previously, similarities seem to persist mostly in a website's *noisy* content. Dynamic website content that is unique per website could thus be worth excluding when identifying similarities.

Based on reviewed literature, the detection of criminal websites has perhaps mostly considered textual elements. This includes features related to elements such as HTML semantics, sentences, words, and source code structure. Although website screenshots have been experimented with previously [4], this has not been seen in a DL context. To our knowledge, we have neither seen the use of SNNs within the field of criminal website detection. The fact that similarities are persistent within criminal websites could be advantageous. With this knowledge in mind, this thesis will explore how SNNs can be utilized to identify similarities in fraudulent websites, having website screenshots as the input medium.

Chapter 4

Methodology

The following chapter will describe the methods used to prepare and develop the environment for the experiments. Methods for data collection and dataset pre-processing will be discussed, followed up by an explanation of how the SNN was implemented. Validation techniques will also be described. The chapter also aims to maintain the reproducibility of methods for further work.

4.1 Practical Approach

Based on initial research and existing work, it was identified that less research exists focusing on features in images within criminal website detection. As previously described in Chapter 1, similarities tend to persist in some types of fraudulent websites for criminals to save time. It was decided to implement a SNN using two CNNs as sub-networks. Using DL was interesting in order to allow the network to learn by itself, rather than defining features beforehand. Figure 4.1 shows how the project was approached on a high level.

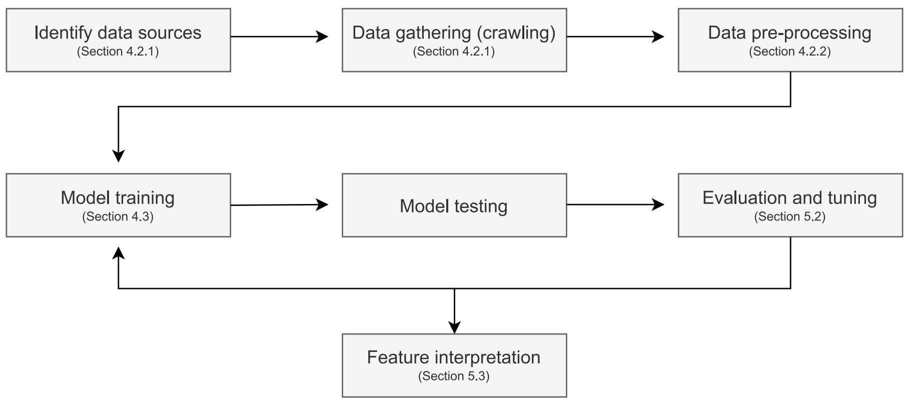


Figure 4.1: High-level process model and project approach

In order to further investigate and answer the research questions defined in Section 1.5, some requirement guidelines were defined. These were based on the research questions, relevant background material, and identified existing research.

- Images are used as input medium since previous research vastly explores text-based features in criminal website detection. To the author's knowledge, less research exists focusing on *visual* features captured from images of websites.
- The machine learning algorithm focuses on learning similarities. This is due to previous research indicating that certain categories of fraudulent websites tend to be similar due to website replication.
- The dataset should be populated by screenshots of websites within a category where similarities often persist.
- Instead of identifying potential relevant features manually, DL will be utilized in an attempt to let the model learn relevant features itself. After training a model with acceptable accuracy, learned features will be interpreted and analyzed.

As training NNs require more computational resources compared to more traditional machine learning methods [19], sufficient equipment was required. A cloud-based Ubuntu 20.04 instance was utilized, using the technical specifications stated in Table 4.1. The specifications were sufficient enough to ensure fast training performance. For the SNN implementation, Python was chosen as the programming language, further utilizing PyTorch as the machine learning framework. PyTorch was chosen due to the author having some previous experience with the framework. In addition, PyTorch's documentation contains an example of a SNN implementation. Python was also utilized for other administrative scripting and data collection tasks.

Hardware type	**Specification**
Graphics Processing Unit (GPU)	1/4th of a Tesla v100 10GB
Central Processing Unit (CPU)	Intel Xeon Gold 6240 CPU, 2.60GHz
HDD storage	100GB

Table 4.1: Cloud instance specifications.

During data collection, it became clear that similar websites had varying degrees of similarities. While some websites were duplicated with only minor changes, others had less obvious similarities. Table 4.2 defines terms to which the report will hereby refer regarding degrees of website similarity.

Term	Definition
Dissimilar	Samples that do not have any indication of being replicated.
Similar	Samples are identified to have similar features upon manual inspection, such as website structure, element positioning, or theme. It may not be directly obvious that the websites share features without closer inspection.
Near-duplicates	Sample websites that are clearly replicated but with minor changes to some features, such as logo, colors, text, etc.

Table 4.2: Definition of similarity terms.

4.2 Data collection and dataset population

This section will describe how relevant data was collected in order to populate the appropriate dataset for the task. The process is important to document in order to achieve reproducibility of experiments.

4.2.1 Website screenshot collection

Since fraudulent websites often tend to have similar features or duplicated content [4], the screenshots in which the SNN is trained on should preferably consist of websites that share a similar structure or visuals. This could for instance be website forums sharing the same framework or near-duplicate fraud web pages.

No publicly available dataset containing screenshots of similar websites was. Several datasets containing website URLs rather than website screenshots exist. These can be utilized by a crawler to visit and gather screenshots manually as described in Section 2.1. Considered dataset for this approach included *Malicious and Benign Webpages Dataset*[1] and *Spam URLs Classification Dataset*[2]. These datasets contain a bigger volume of website URLs that could be utilized to gather screenshots from. However, they do not contain just similar websites explicitly.

As stated in Chapter 3, similarities often persist in HYIP and fake escrow websites. Attempts were hence made to find lists of URLs within these two categories of fraudulent websites. The following three data sources were identified:

- A HYIP-dataset on Kaggle from 2018 [40], containing 1313 URLs.
- `Escrow-fraud.com` [41]: A tracker for (suspected) fake escrow sites, being updated from time to time.

[1]`https://www.data-in-brief.com/article/S2352-3409(20)31198-7/fulltext`
[2]`https://www.kaggle.com/datasets/shivamb/spam-url-prediction`

- `aa419.org` [42]: The *Artist Against 419* organization. Aims to spread awareness around Advance Fee Fraud websites.

Initial inspection of a handful of URLs from the lists revealed that similarities and near-duplicated content were consistent facts. As shown in Figure 4.2, some web pages were identical apart from having different colors or logos. Other examples shared the same structure but with minor changes to the element's positioning. The data sources above were hence considered relevant for the task at hand.

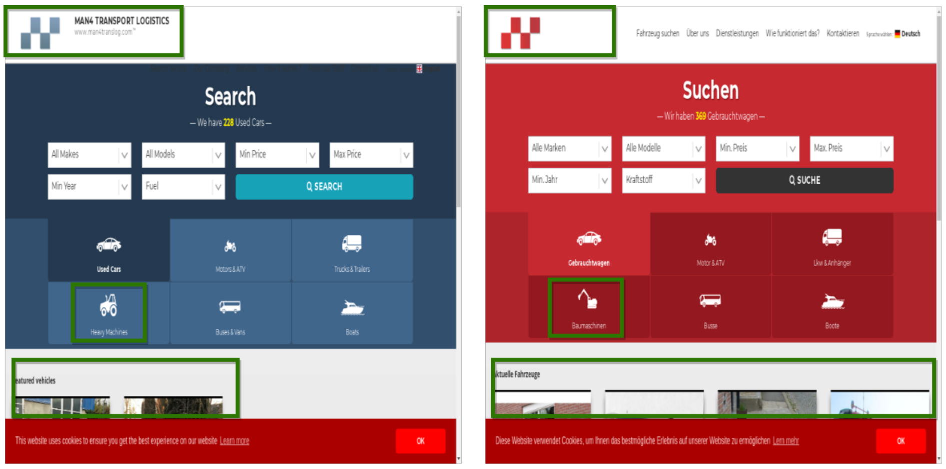


Figure 4.2: An example of two individual fake escrow campaigns which share duplicated content. Only minor elements, including language, colors, logos, and images, have been modified.

Crawling the HYIP-dataset

In order to collect website screenshots from the HYIP-dataset[40], Selenium browser automation for Python was utilized. The workings of Selenium were previously described in Chapter 2.1. Appendix A shows the source code for how the URLs in the HYIP-dataset was crawled. Selenium visits each URL in the dataset to access its index/front page. A screenshot of the Selenium browser's viewport is thus captured.

By default, Selenium captures screenshots at 800x600 pixels, which represents a 4:3 resolution. Initial inspection revealed that many of the websites in the dataset are not optimized for responsive resolutions. Since most of today's modern monitors support a 16:9 aspect ratio, the 1280x720 dimension was chosen to avoid webpage elements being lost due to a lack of responsiveness. 1280x720 were chosen above the more common resolution of 1920x1080 to keep the images at a smaller size, as this will affect the training performance of the model.

The crawler gathers two screenshots from each URL in the dataset. Firstly, it will visit the website directly and capture a screenshot if it is up and running. Since the HYIP-dataset was published in August 2018 [40], many of the HYIPs previously recorded were offline/taken down at the time of writing. The crawler will hence also lookup the URL at Wayback Machine[3], which is an internet archiver capturing snapshots of active websites [43]. Wayback Machine's API can be used to view the website as it was around a certain timestamp. The crawler conducts an API-request to `archive.org/wayback/available?url={url}&timestamp=20180820`, where "{url}" is replaced with the current URL being parsed. The request will return whether a snapshot is available around the requested timestamp. If so, it will also return the URL of the archived snapshot from Wayback Machine. This URL will further be visited by the Selenium crawler to capture the screenshot.

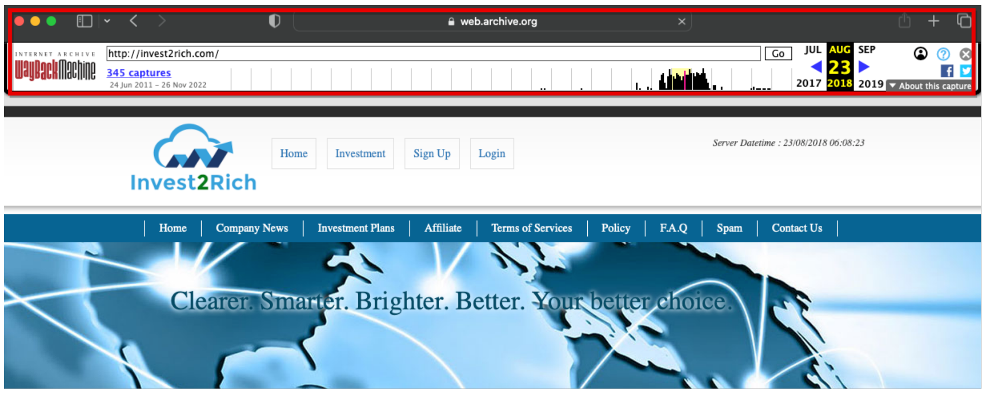

web.archive.org
http://invest2rich.com/
Go
345 captures
24 Jun 2011 - 26 Nov 2022
JUL AUG SEP
23
2017 2018 2019
About this capture
Invest2Rich
Home
Investment
Sign Up
Login
Server Datetime : 23/08/2018 06:08:23
Home | Company News | Investment Plans | Affiliate | Terms of Services | Policy | F.A.Q | Spam | Contact Us
Clearer. Smarter. Brighter. Better. Your better choice.

Figure 4.3: A HYIP-scam website visited through Wayback Machine. The grey bar at the top (marked in red) will be present in Selenium's view-port, and thus in the outputted screenshot.

Figure 4.3 shows how it looks when a website snapshot is opened through Wayback Machine. At the top of the browser's viewport is a white header containing snapshot information. When capturing a Selenium screenshot from Wayback Machine, this header will be a part of all screenshots, making it a continuous element. This may inaccurately influence similarity computation. Having the header atop as part of all screenshots may decrease the level of indifference calculated by the SNN, increasing the risk of false positives. Moreover, this Wayback Machine element is neither a part that appears on the original websites.

Selenium has the ability to locally modify a page's CSS using JavaScript. With this technique, the crawler sets the *display* CSS-property of Wayback Machine related elements to *none*. We thus remove elements not originally present in the websites to avoid it affecting similarity calculation.

[3]`https://archive.org/web/`

Crawling the escrow-fraud tracker

Collection of website screenshots from the previously identified data source `Escrow-Fraud.com` was completed similarly to how HYIP screenshots were collected. However, since the URLs, in this case, were found on a website rather than in a dataset, some additional pre-processing was required. The website's search page [41] consists of tables that keep track of suspected escrow frauds. To make the crawling process easier, the tables were *curled* to obtain their raw HTML-code locally. Using the Pandas module for Python[4] data frames were created out of the HTML tables. These data frames were further iterated using the same method as for the HYIP website collection. Both by visiting each website directly and through Wayback Machine.

Gathering fraud screenshots from aa419.org

`aa419.org` belongs to an organization named Artist Against 419, which aims to spread awareness of Nigerian Advance Fee Fraud campaigns [42]. Thematically, these frauds trick their targets into believing that they will receive a large financial reward [42]. In return, they ask their victims for fees for the reward to be transferred. Artist Against 419's website offers a public JSON-API which can be used to access their database of fraudulent websites. Using a personal API-key generated on the website, the 5000 latest URL entries were gathered using *curl* towards their API. These URLs were thus crawled for screenshot collection in the same manner as the HYIP-dataset and `escrow-fraud.com` tracker.

4.2.2 Dataset population

Crawling the URLs listed in the HYIP-dataset[40], `escrow-fraud.com` [41] and `aa419.org` [42] gathered approximately 3700 screenshots. However, a bigger part of the screenshots were considered not applicable to the task at hand. This included screenshots of websites that were offline, parked, empty, or loaded with advertisements. Including these types of screenshots in the dataset would introduce less relevant data in the dataset as many of them were identical but without relevant content.

After sorting out irrelevant website screenshots, the collected data ended up with approximately 700 images. Table 4.3 shows the final distribution of screenshots from the three data sources discussed in Section 4.2.1.

These 700 screenshots contained several categories of websites that shared similar content, structure, or visual appearance. These were identified by a manual walkthrough and human eyesight. Whereas some websites were clearly duplicates with minor changes from their original (as seen in Figure 4.2), some needed

[4]`https://pypi.org/project/pandas/`

Data source	**Amount of samples**
HYIP-dataset from Kaggle	158
Escrow-fraud.com	56
aa419.org	520
Sum	734

Table 4.3: Amount of samples from each data source used in the final populated dataset.

a closer look in order to be determined as similar. The less obvious similarities could, for instance, consist of websites sharing the same front-end framework, but with changed elemental positioning, etc.

Websites were categorized together based on their similarities, and each category represent a class. For instance, would all samples within class A be similar to each other, but dissimilar from the samples in class B or C.

Two datasets were created out of the 700 screenshots gathered: The *small dataset* and the *diverse dataset*. The small dataset has fewer samples but contains more near-duplicate samples. The diverse dataset contains more samples but fewer near-duplicate samples. This is due to the collected data from Section 4.2.1 ending up with more *similar* samples rather than near-duplicates in total. Classes with near-duplicate samples were hence smaller (10-20 samples each) compared to classes with *similar* samples.

Two sets were created to avoid dataset imbalance, keeping the number of samples within each class to approximately the same amount. This could also allow for evaluating how the SNN performs on near-duplicates versus less obvious similarities. Table 4.4 shows how samples are distributed within each dataset respectively.

Small dataset				
Total samples	**Total classes**	**Per class**	**Train samples**	**Test samples**
198	9	14 - 28	158	40
Diverse dataset				
540	9	20 - 108	430	110
Simset				
Dissimilar samples		**Similar samples**	**Near-duplicate samples**	
64		64	64	

Table 4.4: Distribution of samples in the populated datasets respectively.

Both datasets were trained on their respective models since their relevant features within each set could vary. The classes in the two datasets also vary slightly. Comparison of experiments made on the two datasets will be further showcased in Chapter 5.

A third dataset, the *simset*, was also created. The simset was not used during model training or regular prediction. It was created to answer research question 3, which explores which similarity scores define dissimilar, *similar*, and near-duplicate pairs. *Simset* contains three classes: The dissimilar class (0) contains samples that were collected but not included in the previous datasets. The similar class (1) contains samples with *similar* properties, that are not near-duplicates. The near-duplicate class (2) contains samples of replicated websites with minor changes. As opposed to the small and diverse datasets, the purpose of the simset is to evaluate similarity score thresholds that distinguish these three categories of websites. The distribution of samples within the simset is presented in Table 4.4. Experiments with the simset are further described in Section 5.4.

Class visualization

Figure 4.4 shows examples of samples from each class within the small dataset. As described previously, this set contains more near-duplicate samples. Similarly, Figure 4.5 showcases examples for each class in the diverse dataset. Comparing Figure 4.4 and Figure 4.5, it can be seen that the classes included in both datasets are similar. Apart from the number of samples per class, the major exception between the dataset is that class 1 in the small set is excluded from the diverse set. Class 7 in the diverse dataset contains cryptocurrency scam websites, which were not present in the small dataset. Table 4.5 further describes the differences thematically between the classes in each dataset. Although many classes overlap, the diverse dataset contains fewer samples that are near-duplicates.

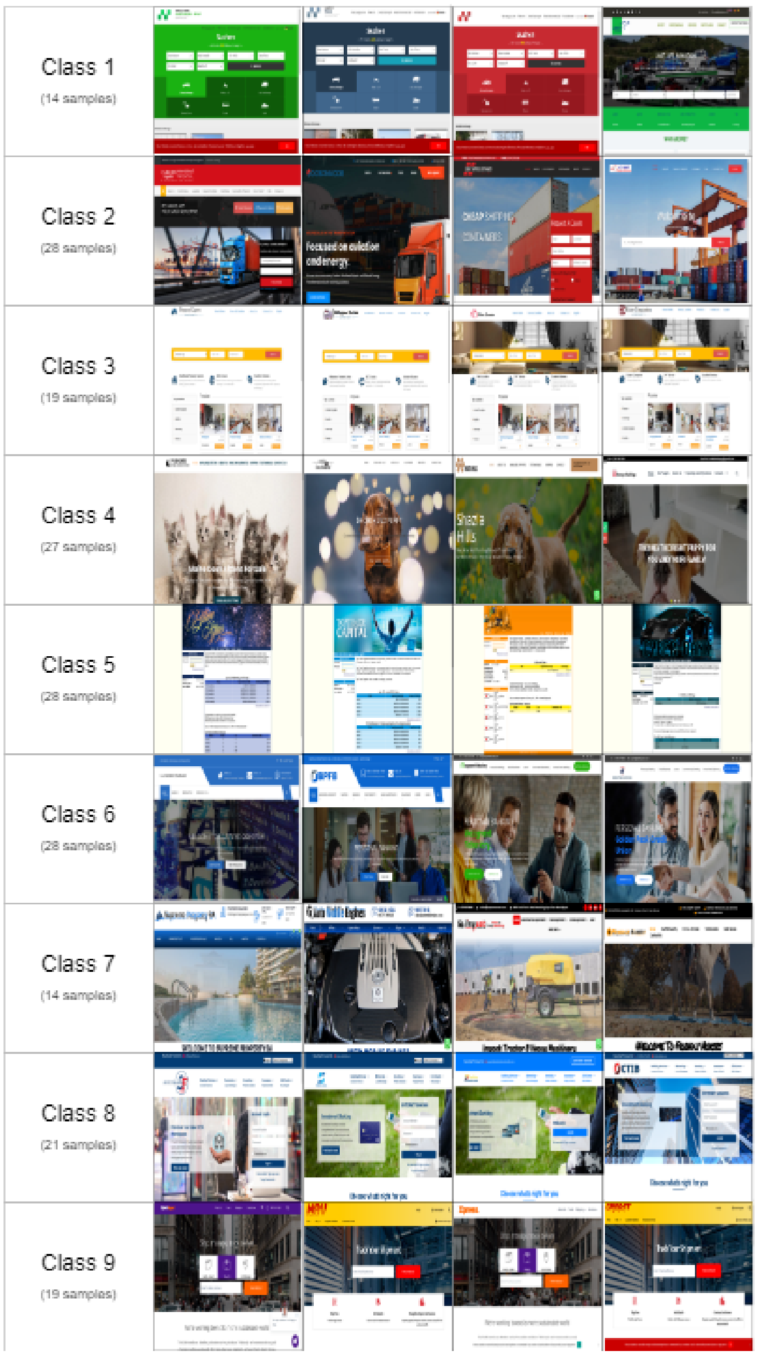


Figure 4.4: Sample examples from each class in the small dataset.

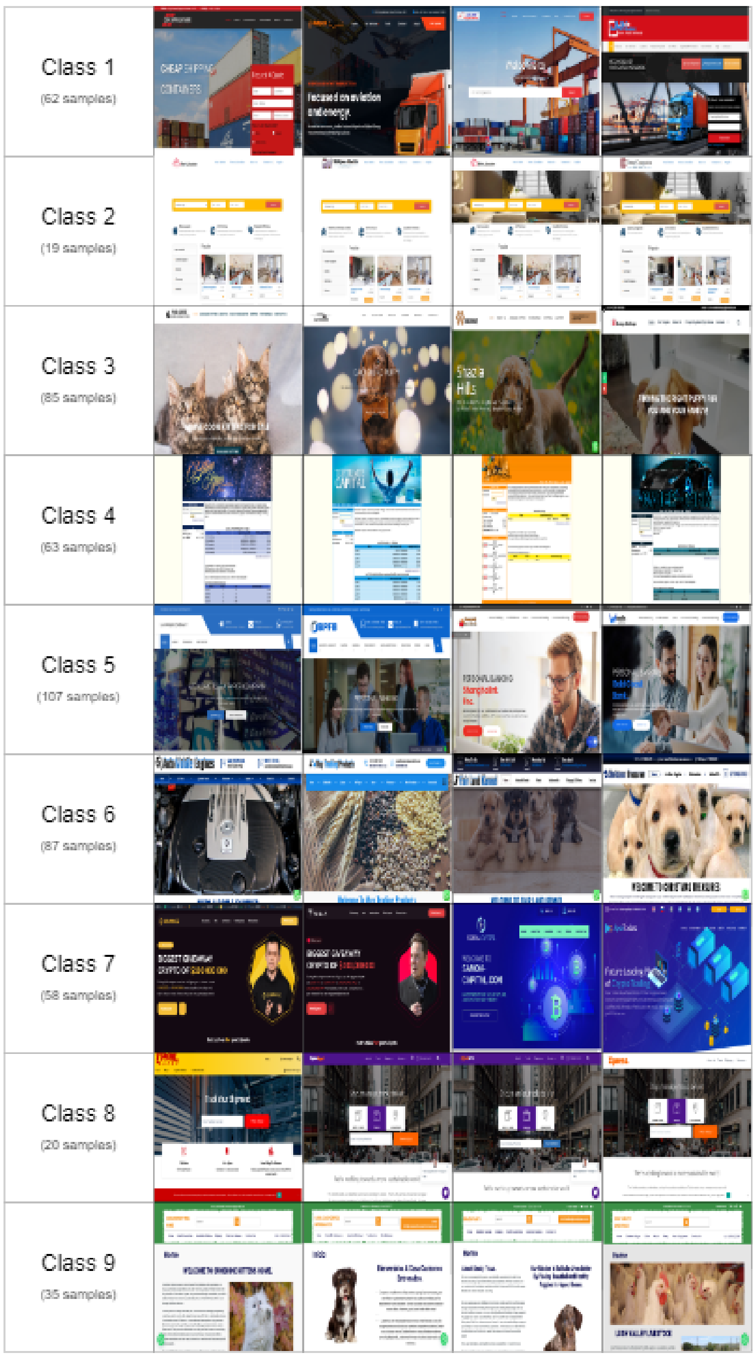


Figure 4.5: Sample examples from the diverse dataset

Class	Small dataset	Diverse dataset
1	Fake escrow transportation services	Fake cargo and transportation services
2	Fake cargo and transportation services	Fake real estate services
3	Fake real estate services	Fake animal breeders
4	Fake animal breeders	high-yield investment program (HYIP)-scams
5	HYIP-scams	Fake investment advise and HYIP-scams
6	Fake investment advise and HYIP-scams	Collection of scams within niches such as industry, food, motors and animal breeding
7	Collection of scams within niches such as industry, food, motors and animal breeding	Fake cryptocurrency investment programs.
8	Fake investment advise and HYIP-scams	Fake package tracking and shipping
9	Fake package tracking and shipping	Fake animal breeders and sellers

Table 4.5: Thematic descriptions of dataset classes.

Pre-processing

Pre-processing the populated datasets before training the model maintains training correctness and efficiency. It also prepares the input data so that it matches the model's expected input.

ResNet was chosen as the NN architecture due to its popularity in use with CNNs and for its support with the PyTorch framework. ResNet architectures require higher computational resources compared to other lightweight networks such as MobileNet. Since the resources available had great computational power, the use of ResNet was feasible. An overview of the workstation's technical specifications were presented in Table 4.1.

Since the amount of data at hand was limited, it could be challenging to train an accurate model. Especially considering the fact that a deep CNN architecture needs high volumes of data. Therefore, the implementation utilized a pre-trained version of ResNet-18 (also known as transfer learning) before training on the populated datasets. This could aid the model in the process of considering relevant features when less training data is available.

The screenshots in the dataset populated were properly resized to match the image input size supported by ResNet. The pre-trained ResNet models were trained on images with a size of 224x224 pixels. Input images from the dataset were resized to these dimensions in order to match samples that the pre-trained model was trained on. This is necessary to ensure that the collected training data is consistent to previous input.

During collection screenshots were captured at a resolution of 1280x720 pixels as described previously in Section 4.2.1. Input data thus have an aspect ratio of 16:9 rather than 1:1 which is the aspect ratio supported by the pre-trained ResNet. Gathered screenshots were resized to 720x720 pixels, changing the aspect ratio to 1:1. This implies that the width of the screenshots was shrunken from 1280px to 720px. Some image details will be lost in the process due to the screenshots becoming more narrow. However, the most important elements of the websites are still present since no cropping occurs. The image convolution will eliminate irrelevant image details in the process regardless. Losing minor detail should not have a significant impact on resulting training/prediction.

Using Torchvision's *transforms* module, tensor transformations were applied to each screenshot loaded into training and testing sets, further resizing them to a size of 224x224.

Training and testing datasets were also normalized before being processed by the SNN. Normalization techniques include calculating the datasets' mean and standard deviation, which were further applied to the input tensors as part of the preprocessing phase. This process assists in maintaining computational performance and consistency.

Training and testing data distribution

The distribution of training and testing samples within the datasets is done randomly. Using *train_test_split()* from *Scikit-learn*[5], the datasets are randomly split into 80% training samples and 20% test samples. Due to the amount of data at hand being limited, 20% test data was considered the maximum percentage, to allow for more training pairs. Table 4.4 previously showed the final distribution of samples between training and testing. Using PyTorch's *DataLoader* utility, the randomly split datasets were loaded into training and testing dataloaders respectively.

Using a randomly populated dataset for each test run would make it harder to debug which factors that does improve according to changes made in the program. In order to monitor model accuracy over time, a fixed seed was set using *train_test_split()*. When providing a specific seed, the datasets will be split equally

[5] `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`

every time the program is run, making it easier to monitor performance. In addition, a fixed seed can help to ensure better reproducibility of results for others who might use the datasets in the future.

4.3 Siamese Network Implementation

The SNN model was implemented using weights from a pre-trained ResNet-18 model. ResNet-18 was considered the most feasible alternative in relation to the volume of data available. Although the computational resources are available, deeper CNN architectures such as ResNet-50 or ResNet-101 may be too deep compared to the small amount of data. Having a too complex architecture could make it hard for the model to generalize well to the training data [14]. Similarly, having a too narrow architecture could impact the model's ability to learn the most relevant features.

When in action, input data (screenshots) is passed through two identical CNN sub-networks respectively. Cosine similarity is computed between the outputted embeddings before being passed to the loss function. Since two sub-networks are being used, pairs of images need to be loaded. The selection of screenshot pairs from the dataset works as follows:

1. A class is randomly picked.
2. A random screenshot from the above class is selected.
3. Calculate a random number.
4. If the above number is even, select a second screenshot from the same class as screenshot 1. Otherwise, select the second screenshot from another random class.
5. Choose a target, 1 or 0, depending on whether the two screenshots reside within the same class.

Using Pytorch's *DataLoader* module, batches of screenshot pairs and corresponding targets are loaded into training and testing dataloaders respectively. The full source code for the SNN implementation is available in the thesis' BitBucket repository [44].

It should be noted that the implemented SNN is not a model for classifying samples into classes. It is simply a similarity function computing similarity between two outputted embeddings. Samples in the dataset were indeed divided into classes initially. However, the purpose of this classification is to label pairs of images as either similar or dissimilar, depending on whether they belong to the same category. The SNN does not take classes into account beyond this. Model output is simply a score between 0 and 1 to represent the similarity between randomly selected image pairs. During prediction, websites with similarity scores higher than 0.5 were considered similar (1). Websites with similarity scores below this value were labeled as dissimilar (0). Experiments with other thresholds than 0.5 are described in Section 5.4.

4.3.1 Loss function

As described in Section 2.5.1, *ContrastiveLoss* and *TripletLoss* are perhaps some of the more common loss functions to use in Siamese Networks. TripletLoss may in many cases be the most preferable one, as it uses an anchor image to measure between positives and negatives. This could further increase training precision. However, TripletLoss could also introduce less unique training triplets, due to the utilization of an anchor. For smaller datasets such as in this project, TripletLoss may thus be too complex. The decision was made to proceed with the use of ContrastiveLoss to calculate loss between pairs rather than triplets, due to the small amount of data available.

4.3.2 Miner function

In order to strengthen the selection of training pairs within each batch of images, a *miner* from PyTorch Metric Learning has been utilized. A miner attempts to find positive and negative pairs within the batches that are particularly difficult [45]. This may benefit the training process by creating more diversity in the pairs that are chosen, opening for better model generalization. *MultiSimilarityMiner* and *PairMariginMiner* were the two miners attempted, as they both operate on pairs rather than triplets by default. These miners also allow for adjustment of margin and epsilon for fine-tuning how hard pairs should be selected.

4.3.3 Overfitting mitigations

Since the datasets are small, the amount of training data and unique training pairs may introduce overfitting of the model. In an attempt to reduce potential overfitting, several measures have been taken:

- **Data augmentation:** Different augmentation techniques as seen in Listing 4.1 are randomly applied to the input images to create more diversity in the data. This includes image rotation, color jitter, cropping (not affecting input resolution), contrast, and more. Manually assigned probabilities make sure that drastic augmentations have less chance to happen, in order to avoid total manipulation of the input data.
- **Unseen test data:** The model computes achieved accuracy during each epoch for both training and testing data. To get the most representative performance possible, the unknown test performance is compared to the training performance for each epoch. This allows monitoring for overfitting and underfitting.
- **Architecture choice:** In an attempt to prevent overfitting, ResNet-18 was chosen as the architecture as opposed to i.e. ResNet-50. Having a too complex architecture with too little data could cause overfitting.

Code listing 4.1: Data augmentation measures.

```python
from torchvision import transforms

pad = transforms.Pad(padding=(10))
gaussian = transforms.GaussianBlur(kernel_size=(3), sigma=(0.1, 0.7))
cj = transforms.ColorJitter()
elastic = transforms.ElasticTransform(alpha=25.0)
train_transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.RandomHorizontalFlip(p=0.2),
    transforms.RandomRotation(degrees=(-5,5)),
    transforms.RandomApply([pad], p=0.3),
    transforms.RandomPerspective(distortion_scale=0.5, p=0.3),
    transforms.RandomCrop(size=195),
    transforms.RandomAutocontrast(),
    transforms.RandomApply([gaussian, cj], p=0.5),
    transforms.RandomEqualize(p=0.4),
    transforms.RandomPosterize(bits=2, p=0.3),
    transforms.RandomApply([elastic], p=0.2),
    transforms.ToTensor(),
    transforms.Normalize(mean=mean, std=std)
])
```

4.4 Model validation

As discussed in Section 4.2.2, two datasets were populated based on the data collected and were further trained on their respective models. Terminology related to the validation methods is described in Table 4.6 and Table 4.8. The SNN models were evaluated using the validation methods listed below.

- Overall training & testing accuracy and loss for each dataset/model.
- Total TPs/TNs/FPs/FNs for each model.
- Individual class performance for each model.

Table 4.6 contains a description of validation terminology in the context of SNNs. To support these descriptions, Table 4.7 showcases practical examples for each scenario.

Term	**Description**
True Positive (TP)	Samples that are both labeled and predicted as positives (similar).
True Negatives (TN)	Samples that are both labeled and predicted as negatives (dissimilar).
False Positives (FP)	Samples that are labeled as negatives (dissimilar) but predicted as positives (similar).
False Negatives (FN)	Samples that are labeled as positives (similar) but predicted as negatives (dissimilar).

Table 4.6: Description of validation terms.

Term example	**Sample classes**	**Sample label**	**Sample prediction**	**Correctness**
True Positive (TP)	A & A	True (1)	True (1)	True (1)
True Negative (TN)	A & B	False (0)	False (0)	True (1)
False Positive (FP)	A & B	False (0)	True (1)	False (0)
False Negative (FN)	A & A	True (1)	False (0)	False (0)

Table 4.7: Validation terms usage example in the context of a siamese network

The trained models were initially evaluated using *accuracy* as the evaluation metric. However, since the datasets contain some imbalance, other evaluation metrics were considered appropriate:

- **Precision:** The ratio measured of *true* correctly predicted instances (TPs), compared to the total of positive predictions (TPs & FPs) [46]. It is used when the cost of FPs is high. The trained models compute similarity scores based on identified similarities. If the number of FPs is too high, it means that the model struggles with learning the similarities, which is the main goal of the algorithm. Precision is therefore considered.
- **Recall:** The sensitivity of the true positive rate [46]. Measures the ratio of TPs compared to FNs. Useful when the cost of false negatives is high. This thesis has been focused towards fraudulent website detection. In fraud detection, minimizing FNs is important to avoid that illegal activity does not go through undetected. Recall can be a more accurate metric to consider when FNs are important to detect.
- **F1-score:** The weighted average between precision and recall [46]. In some systems, having both of these metrics may be important for the scenario. However, they may also show very different results. F1 gives a mean value

between these metrics to give a more accurate metric when both are considered.

Table 4.8 defines validation formulas and descriptions for the used performance metrics.

Term	Formula	Description
Accuracy	(TP + TN) / (TP + TN + FP + FN)	Overall performance, correct predictions compared to the number of samples.
Precision	TP / (TP + FP)	Measures correctness of true predictions
Recall	TP / (TP + FN)	The sensitivity of true positives.
F1-score	2 * (precision * recall) / (precision + recall)	Weighted average of precision and recall.

Table 4.8: Description of performance terms.

As stated previously, each dataset was evaluated on two scenarios: Validation of each individual class and validation of overall model performance. It is important to highlight the fact that individual class validation may have some degree of bias when used to validate this SNN. The model(s) are created to output a binary value, either 1 or 0 depending on predicted similarity. Consider the following scenario: If a pair of images belonging to different classes are indeed predicted as TNs, the TN counter increments two times. One for each of the classes. On the other hand, if a pair of images belonging to the same class are indeed predicted as TP, the TP counter would only be incremented once. This is due to both images in the pair having the same label. In practice, this is considered an issue since the number of negatives (TN/FN) for a particular class will be significantly higher than the number of positives. The individual class validation is only performed to give an indication of which types of websites have the most persisting similarities. Only the overall model validation should be used as a measure to validate the SNN's performance.

4.5 Feature interpretation

In order to answer the second research question, the features considered by the SNN were visualized for interpretation. Since NN are able to learn relevant features from input data on their own, one of the main goals for the second research question was to examine the model's ability to capture futures. This analysis could provide input regarding relevant features in criminal website detection for further work.

The ResNet-18 CNN architecture consists of several layers producing feature maps for processed input data. This process was further described in Chapter 2. These layers have different focuses in terms of which features they represent [47]. Lower layers mainly capture basic features such as edges, corners, and lines. High-level feature maps capture more complex features such as colors, objects, and patterns [47].

In order to analyze how the CNN sub-networks in the SNN were interpreting features in website screenshots, a feature extractor model was developed. Briefly described, the Feature Extractor takes the diverse model, trained on the diverse dataset, as a parameter. A modified model will be produced, considering only the layer passed to *self.features*. This allows for feature visualization of specific layers in the architecture. Listing 4.2 shows how the Feature Extractor class was implemented to extract lower layer features ("layer1").

Code listing 4.2: Feature Extractor class.

```
import torch.nn as nn
import torch

class FeatureExtractor(nn.Module):
    def __init__(self, model):
        super(FeatureExtractor, self).__init__()
        self.model = model
        self.model = getattr(self.model, "cnn")
        self.features = getattr(self.model, "layer1")
        self.features = nn.Sequential(*self.features)
        self.pooling = getattr(self.model, "avgpool")
        self.flatten = nn.Flatten()
        self.fc = getattr(self.model, "fc")
        #self.fc = self.fc[0]

    def forward_once(self, x):
        output = self.model(x)
        output = output.view(output.size(), -1)
        return output

    def forward(self, x, y):
        output1 = self.forward_once(x)
        output2 = self.forward_once(y)
        output = torch.cat((output1, output2), 1)
        return output
```

Once a Feature Extractor model was created, saliency maps were utilized to visualize considered features. Saliency maps are created through *pixel attribution*, which is a method to highlight pixels in the image based on their contribution to the classification [48]. The resulting saliency maps can further be interpreted to analyze how well the network identifies objects, patterns edges, etc. Saliency maps were created by passing image tensor pairs to the modified model and calculating their gradients. Appendix B shows the complete workings of this process. Resulting visualization and saliency maps are presented in Chapter 5.

Chapter 5

Experiments

The following chapter will describe the experiments conducted in addition to the achieved results. Firstly, the chapter will discuss experiments related to **RQ1**, regarding testing and validation of the SNN model. Next, **RQ2** will be addressed. This includes visualizing relevant features learned by the SNN through pixel attribution. Lastly, experiments related to **RQ3** will be showcased. This involves experimenting with similarity score thresholds for grades of similarity, along with how this affects the outcome. Results and achievements showcased in this chapter will be further discussed in Chapter 6.

5.1 Hyperparameters

In order to achieve the highest performance possible, experiments related to various combinations of hyperparameters were conducted. We considered the following parameters as hyperparameters:

- **Epochs:** The number of iterations for training/testing functions. 1 epoch iterates all train/test batches of data once.
- **Batch size:** The size of each individual batch that is passed to the model. A set of 100 samples with a 16 batch size would for instance produce 6 batches.
- **Learning rate (LR):** Determines iteration step size and how fast the network should update its weights.
- **Gamma:** Gamma is multiplied by the LR and influences the learning rate accordingly.
- **Optimizer:** Optimizes the NN performance by updating weights, LR, attributes etc.
- **Scheduler:** A helper to adjust LR as the number of epochs increases.
- **Loss function:** Measures the predictions against ground truth to calculate how well the model performs. The goal is to minimize the loss as best as possible.
- **Miner:** A helper function that can be utilized to select *hard pairs* in training.

This could increase a model's robustness.
- **Similarity threshold:** The condition that must be met to decide whether the outcome is either dissimilar or similar.

Different experiments of the above parameters were tested in order to identify the best-performing combination. There were several considerations in mind during this process. By using a pre-trained model it was quickly observed that low learning rates and gamma values were appropriate. This is because model's weights are already set, and we want to utilize these weights in our own experiment. A too high learning rate caused the models to converge too early at low performance. This is perhaps due to the pre-trained model being good at representing universal input, considering it was trained on huge amounts of data. When presented with a smaller dataset, high learning rates could perhaps destroy the weights from the pre-trained model.

The optimal number of batch size were determined as 16 and 32. These two parameters were similar performance-wise, with slightly better performance for a 32-sized batch. A batch size of 8 provided less accurate results, and 64 were considered too high compared to the number of samples available. A high number of epochs was considered appropriate due to the learning rate being low. This is to ensure that the model has enough time to update its weights to relevant values since low learning rate means that the model will approach the minimum loss at a slower rate.

Two types of optimizers were tested: Adam and Adadelta. The latter is the default optimizer in Pytorch´s SNN example [31], and was tested initially. Adam was tested due to its popularity in the deep learning community, and since it also utilizes the adaptive learning rate from Adadelta [49]. Results gave higher performance using the Adam optimizer. For the scheduler, StepLR and MultiStepLR were tested. The two methods had little to no difference in performance. StepLR was thus chosen since it was the default scheduler in PyTorchs´s SNN example [31].

ContrastiveLoss and *PairMariginMiner* from PyTorch metric learning [45] were utilized as the loss and miner function respectively. *MultiSimilarityLoss* and *MultiSimilarityMiner* from [45] were also tested. The latter functions also operate with image pairs rather than triplets by default. Performance on the latter options was slightly decreased compared to the first options, however.

A similarity threshold of 50% (0.5) was chosen as the condition to match. In practice, this means that two compared pairs must have a computed similarity score higher than or equal to 0.5 to be considered similar. Comparisons with a lower similarity score are labeled as dissimilar. **RQ3** aims to explore whether other thresholds are more optimal for labeling pairs. Training and prediction using other thresholds are presented in Section 5.4.

Table 5.1 show an overview of the best performing hyperparameters identified through experimentation. To reproduce the experiments in the future, these hyperparameters may provide a starting point when used with a dataset similar to in this thesis. Ranges of used hyperparameter values are also specified in the table.

Parameter type		**Parameter value**
Loss function		ContrastiveLoss
Miner function		PairMariginMiner
Optimizer		Adam
Scheduler		StepLR
Other hyperparameters		
Parameter type	**Lowest value**	**Highest value**
Epochs	1	350
Learning rate	0.0001	0.01
Gamma	0.1	0.3
Batch size	8	32
Similarity threshold	50%	70%

Table 5.1: Best performing hyperparameter combinations identified through experiments

5.2 Siamese Network validation

As described in Section 4.4 the model was evaluated using two methods for each dataset respectively: Overall validation and individual validation per class. The latter was conducted to get an insight into which types of websites perform best on the trained models. The results presented here are relevant for answering the first research question (**RQ1**), related to how suitable a SNN is in terms of learning similarities in website screenshots.

5.2.1 Individual class validation

This section will show the validation of each class in the datasets individually. Individual class validation attempts to gain insight into which type of websites have the most persisting similarities. Studying these insights could help us understand how the model learns similarities across different types of websites.

Small dataset

The following Section will showcase experiments conducted with the small dataset. As stated previously in Table 4.4, the small dataset only contains 198 samples in total. These are divided into 158 train samples and 40 test samples. In this section to follow, the model trained on the small dataset will be evaluated. Evaluations will be visualized using tables, accuracy graphs/diagrams, and confusion matrices.

Table 5.2 shows the evaluated performance for each class individually for the small model. Figure 5.1 further visualized the distribution of accuracy for each class. The graph showcases for instance how classes 4 and 7 stand out.

Class	TPs	TNs	FPs	FNs	Accuracy	Precision	Recall	F1-score
1	130	46	155	0	53%	46%	100%	63%
2	61	84	122	47	46%	33%	56%	42%
3	118	80	167	0	54%	41%	100%	59%
4	66	149	89	44	62%	43%	60%	50%
5	118	66	149	0	55%	44%	100%	61%
6	95	71	169	0	50%	36%	100%	53%
7	105	189	45	1	86%	70%	99%	82%
8	70	74	136	33	46%	34%	68%	45%
9	112	51	158	0	51%	41%	100%	59%
Sum	875	810	1190	125	51,9%	43,1%	87%	57,1%

Table 5.2: Validation of each individual class for the small model. The best results are marked in green.

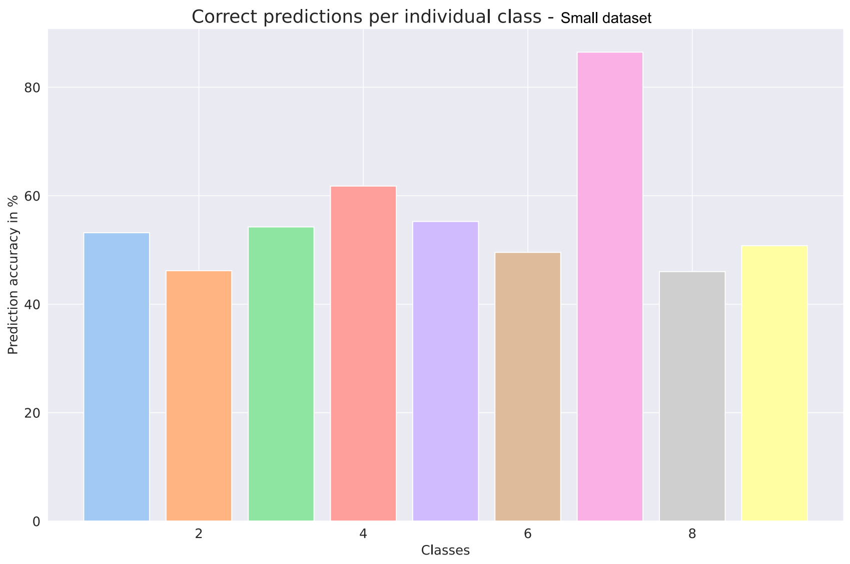


Figure 5.1: Accuracy measured for each class individually - small dataset.

Analyzing these results shows that there is a higher number of true predictions (TP/TN) compared to false predictions (FP/FN). True predictions account for

1685 out of 3000 predictions, which is slightly above 50%. The false predictions account for the remaining 1315 predictions. However, as discussed in Section 4.4, the individual class validation does not represent the overall model performance accurately. This is further showcased in Section 5.2.2. On the other hand, these insights can make it easier to interpret which classes perform better than others.

Class 7 has one of the better TP and TN rates, indicating that the model is able to distinguish (dis)similar samples with great confidence for these types of websites. Only 45 FPs and 1 FN were produced, resulting in high accuracy for this class. This also results in a high F1-score for the class. As showcased previously in Table 4.5, class 7 is a collection of scam campaigns within different niches.

Class 1 in the small dataset consited of fake escrow transportation websites. The amount of FPs are high. This indicate that identifying similarities within this class was difficult, even though the samples were near-duplicates. On the other hand, it was able to distinguish these websites from other types, considering the low number of FNs. Among the worst performing classes was class 2, which was fake cargo and transportation services.

Diverse dataset

The following Section will showcase the experiments conducted on the model trained *diverse* dataset. As previously described in Section 4.2.2, the diverse dataset contains more samples within each class. Thus, this dataset has much more variance. Samples in this dataset also contain some of the near-duplicate classes in the small dataset. However, it is mostly weighted by samples that have less obvious similarities. It is also worth noting that this dataset contains a higher amount of samples compared to the small dataset. This further influences the sums shown in the validation.

As seen in Figure 5.2, class 2, 4, and 8 stands out in terms of prediction accuracy. Examining Figure 4.5 from Section 4.2.2, class 2, 4, and 8 are actually all near-duplicate classes. These three classes are also the only near-duplicate classes in the diverse set. Class 2 contains fake real-estate services, while class 4 are the HYIP-scams websites. Lastly, class 8 is a collection of fake package tracking and shipping websites. These results are interesting. Even though the smaller model has more near-duplicate samples, the diverse model performed better at identifying these.

Table 5.3 shows validation per individual class for the diverse dataset. Class 2 has the best performance. With a 67% precision and 100% recall, it ends up with a weighted F1-score of 81%. The amount of TPs is the double of total FPs, and the TN rate is also very high. No FN predictions were made. This shows that the diverse model is able to identify the fake real estate websites with great confidence.

Class 4 performed very similarly to class 2. However, the FP rate is slightly higher for this class. Class 4 consisted of HYIP websites. Lastly, the fake package tracking samples in class 8 also showed a high performance in the number of true predictions.

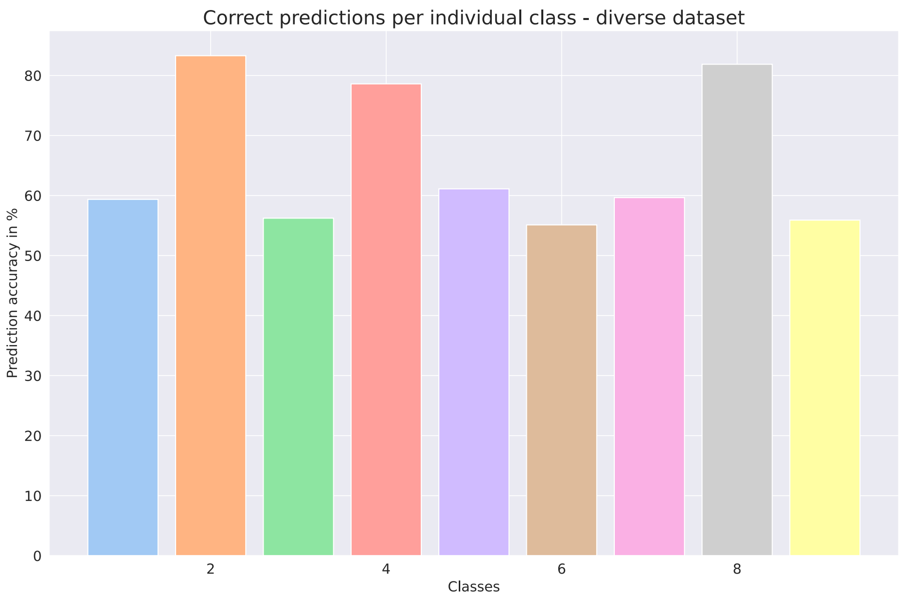


Figure 5.2: Accuracy measured for each class individually - diverse model.

Class	TPs	TNs	FPs	FNs	Accuracy	Precision	Recall	F1-score
1	321	218	362	7	59%	47%	98%	64%
2	309	434	149	0	83%	67%	100%	81%
3	270	240	375	22	56%	42%	92%	58%
4	308	412	196	0	79%	61%	100%	76%
5	310	270	364	5	61%	46%	98%	63%
6	267	240	350	63	55%	43%	81%	56%
7	279	243	351	2	60%	44%	99%	61%
8	264	427	153	0	82%	63%	100%	78%
9	271	226	390	2	56%	41%	99%	58%
Sum	2599	2710	2690	101	65,6%	50,4%	96,3%	66,1%

Table 5.3: Validation of each individual class for the diverse model. The best results are marked in green.

Summary

The above results show that the *diverse* dataset has the best performance based on individual class validation. It should be kept in mind that this evaluation method is not suitable for validating the overall model performance, but rather understand the performance between the different types of websites included in the experiment. A more accurate validation for the overall model is presented in Section 5.2.2.

For the small dataset, class 7 had the best results. As shown in Figure 5.3, website screenshots in class 7 contain a wide white header, navigation, hero image, and a header title. A similar type of logo is always displayed in the top left corner as well. Colors vary, and the theme for these websites is anything from motors to animals. The similarities are more dynamic compared to many of the other classes, and the samples are not obvious near-duplicates. Even though the website header design persists in samples within this class, the hero image covers about 70% of the page. The hero image also varies depending on the website's niche. Class 1 had 50% accuracy, but taking TPs, FPs, and FNs into account, it yielded the second-highest F1-score. The class contained near-duplicate samples, with only minor changes between the websites. These observations indicate that the small model struggled to generalize. Especially towards near-duplicate classes.

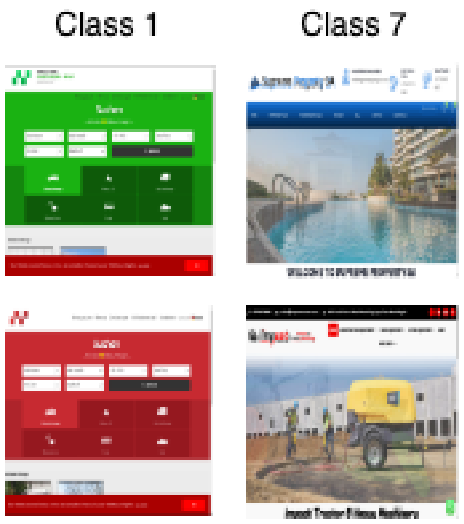


Figure 5.3: Classes with highest performance - small dataset.

The diverse model had the highest performance on class 2, 4, and 8, which all contains near-duplicate websites, as shown in Figure 5.4. This shows that the additional amount of data in the diverse dataset has made the model able to generalize better. As a result, better generalization may result in increased ability to identify near-duplicate websites. Further evaluation of the diverse model validation shows higher TP/TN rates and lower FP rates compared to the small model. These results support the hypothesis that the diverse model has generalized better to catch the website similarities. The next Section 5.2.2 will give a better indication of the two models' overall performance respectively.

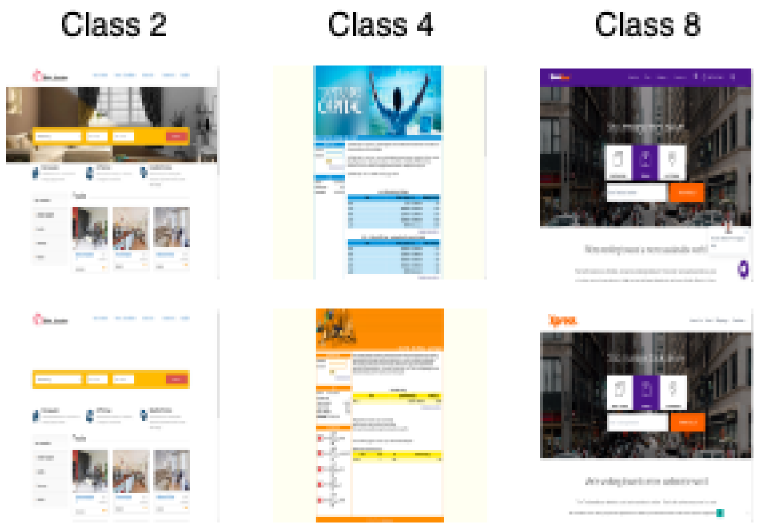


Figure 5.4: Classes with highest performance - diverse dataset.

5.2.2 Overall model validation

The following Section will discuss the overall model performance rather than considering each class individually. The following validation should hence be interpreted with more confidence for evaluating the trained models.

Small dataset

Table 5.4 describes the overall performance of the model trained on the small dataset. True positives out-weights the number of false positives, resulting in a precision of 60%. This shows that the model is somewhat precise at identifying similarities, but still has room for improvement. The model also performs well in predicting negative samples when website pairs are dissimilar. Due to the model having a low FN rate compared to TPs, the recall value is high. This leaves an F1-score of 71%. These results show that the model is capable of not letting fraudulent websites go undetected. At the same time, similarity identification can be improved.

TPs	TNs	FPs	FNs	Accuracy	Precision	Recall	F1-score
875	405	595	125	64%	60%	88%	71%

Table 5.4: Overall model validation - small model

Figure 5.5 visualizes the model validation in a confusion matrix. For true predictions, more saturated colors are desired. Less saturated colors are desired for false predictions. TPs are very saturated, while TNs and FPs have moderate saturation. FNs have no saturation due to their low count.

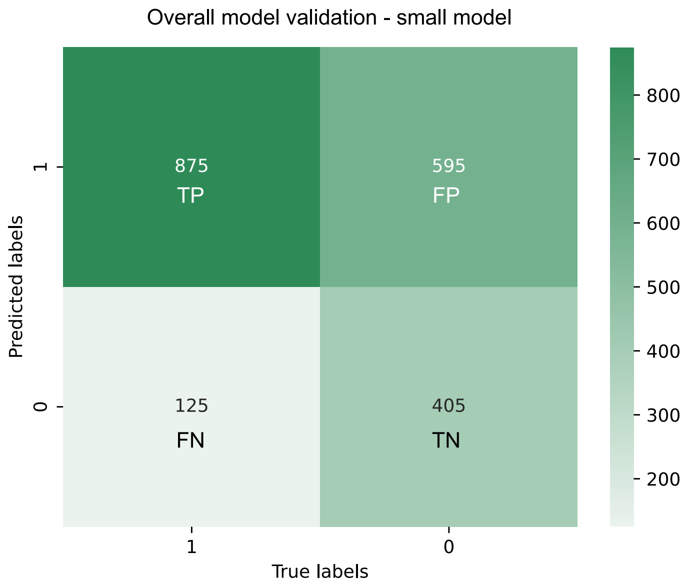
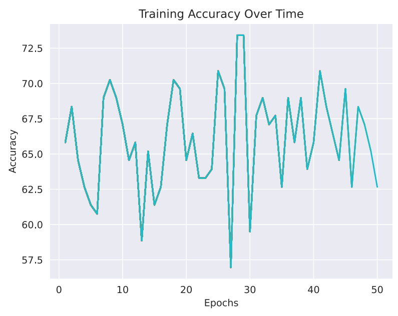


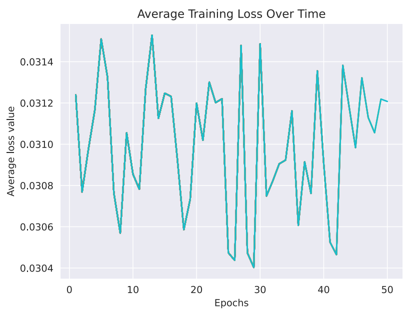
Figure 5.5: Confusion matrix for the small model.

Figure 5.6a and Figure 5.6b show the evolution of training accuracy and loss for each epoch. These measurements are conducted on the training portion of the data. Both accuracy and loss show that sporadic swings in performance take place between every epoch. This could mean that the model struggles with learning from samples in the dataset. The samples could be too difficult for the model to further adapt towards it. The way classes and samples were distributed could be the reason for this. It can also be seen that training struggles to improve over time.

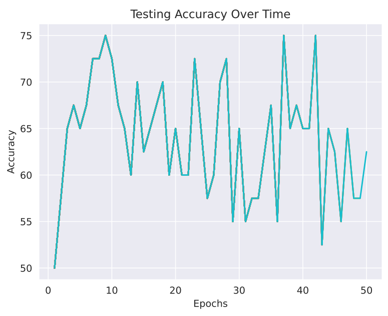
Figure 5.6c and Figure 5.6d show the model's prediction accuracy and average test loss. While training performance shows how training performs over time, prediction accuracy gives a better indication of how the model performs on unseen data. A high variance of accuracy between each epoch can be seen here as well, and the performance does not converge. Prediction performance rather decreases over time. Average test loss also reflects this, which is increasing over time. Showcased



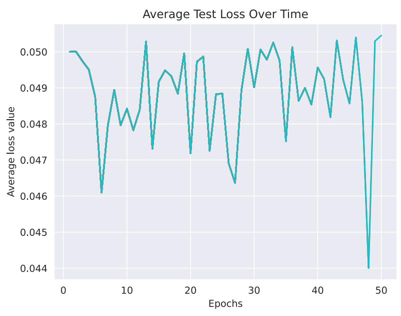
(a) Training accuracy.



(b) Training loss.



(c) Prediction accuracy.



(d) Prediction loss.

Figure 5.6: Training and prediction accuracy and loss for the small model. Shown over 50 epochs.

prediction performance supports the fact that the dataset may be too difficult for the model to learn from each epoch.

As described by Goodfellow & Bengio et al. [14], overfitting occurs when the gap between the training and testing error is high. Comparing training and prediction accuracy and loss can hence give an indication of the model´s fitness. Training accuracy is slightly higher compared to prediction accuracy, but has a and has a non-improving trend. The decreasing prediction trend however signalizes overfitting, as the training is not sufficient enough to generalize to the testing data. The amount of training used in the small dataset is also limited, affecting the model's ability to generalize.

Diverse dataset

Table 5.5 describes the overall performance of the model trained on the diverse dataset. TPs are close to double the amount of FPs. TN predictions also perform very well compared to FNs. Even though the model has more total correct predictions, the FPs are still significant. This means that the model still has potential for improvement in identifying true similar samples.

TPs	TNs	FPs	FNs	Accuracy	Precision	Recall	F1-score
2599	1355	1345	101	73%	66%	96%	78%

Table 5.5: Overall model validation - diverse model.

The results show that the diverse model is robust against not slipping fraudulent websites through, considering the high recall score. In a practical scenario, it would be reliable for fraud detection due to its low number of FNs. However, precision shows that many dissimilar samples may be falsely predicted as similar. This would create some noise and additional after-work in a practical scenario, with such low precision. Collectively, it yields an F1-score of 78%, which shows that model can be utilized in practice with correct training and fine-tuning.

Figure 5.7 shows the visualized confusion matrix for the values in Table 5.5. For true predictions, more saturated colors are desired. For false ones, less saturated colors are preferred. Similar to the small model, TPs have a high saturation, whereas FPs and TNs have moderate saturation. The FN saturation is weak here as well.

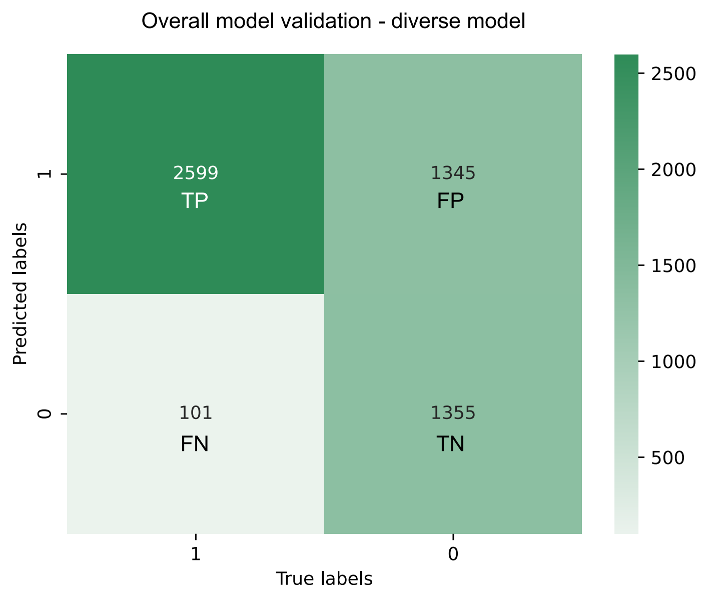
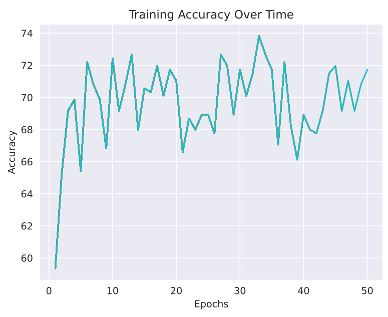


Figure 5.7: Confusion matrix calculated based on the diverse model's overall performance

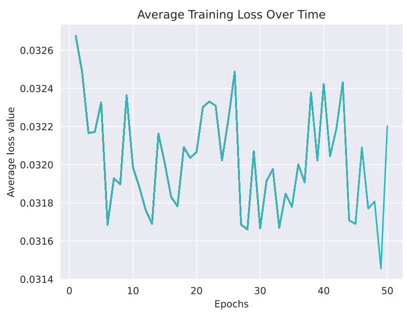
Figure 5.8a and Figure 5.8b shows the evolution of training accuracy and loss for each epoch. These values were calculated on the training portion of the dataset. It can be seen that training accuracy is slowly increasing. This indicates that the model is learning over time and that improvement may happen through additional training. Similarly, training loss is decreasing over time.

Figure 5.8c and Figure 5.8d show the evolution of prediction accuracy and loss for each epoch. These values were calculated on the unseen test data. Prediction accuracy is neither increasing nor decreasing. It stays stable thoroughly. This indicates that even though training improves, the model has not had enough input to improve further. Moreover, it is a sign of underfitting. Since the model has not been able to improve enough, the generalization is harder. Allowing for more training, addition of data, or utilization of a more complex architecture, could improve performance and eliminate underfitting.

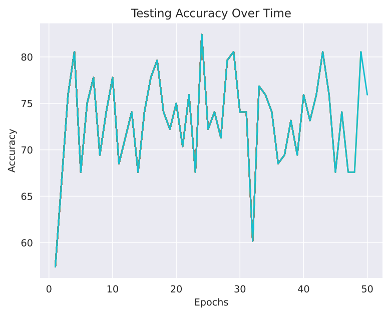
Similar to the small model, each epoch in the diverse dataset also contains major swings in performance. As stated previously, this is likely due to the model not being able to handle certain samples in the dataset. Changing the dataset's content or sample distribution may improve these observations.



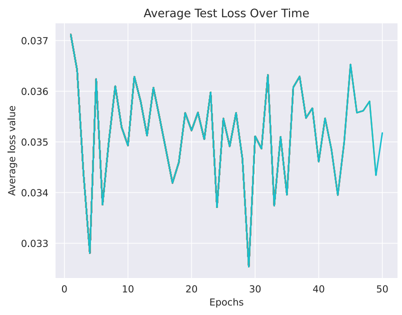
(a) Training accuracy.



(b) Training loss.



(c) Prediction accuracy.



(d) Prediction loss.

Figure 5.8: Training and prediction accuracy and loss for the diverse model. Shown over 50 epochs.

Summary

Comparing the experiments conducted on the small and diverse dataset shows variation in how they perform. The small model performed more poorly. It has a high recall score, but the amount of FPs makes it less precise. Analysis of training and prediction performance showed that the small model is likely overfitting since training does not improve and prediction accuracy is gradually decreasing.

The diverse model had higher overall performance than the small model. Low number of FNs makes it a trustworthy appliance for fraud detection scenarios where FNs have a high cost. Still, the precision has room for improvement. Without increasing model precision, the model will predict struggle with predicting dissimilar sites as similar. Training and prediction performance showed that the model may be underfitting. Training improved over time, but prediction stays stable without improvement. In addition, the average prediction accuracy was higher than the training accuracy. The addition of more data or utilization of a more complex architecture could thus improve performance.

5.3 Pixel attribution analysis

The following section will showcase experiments related to **RQ2**, which explores the interpretation of how the model captures relevant features in website screenshots. The technicalities of the process are described in Section 4.5. Since the SNN consists of two CNN sub-networks, considered features will be visualized through pixel attribution. This process is first applied to the CNN's low-level features, followed up by an inspection of high-level features. Pixel attribution was attempted for both the small and diverse models. However, only the diverse model created results of relevance. The small model was not able to visualize features in saliency maps. This could be due to the low amount of data on which the small model was trained. Saliency maps were successfully produced for the diverse model, but it is also reflected here that the amount of data was not sufficient.

5.3.1 Low-level feature extraction

Lower convolutional layers of ResNet-18 attempt to capture low-level features in images, such as edges, corners, and lines [47]. These features have been visualized using saliency maps as described in Section 4.5. Figure 5.9 shows the original screenshots and generated low-level saliency maps for three different websites.

Analyzing these saliency maps reveals that the model is able to capture low-level features such as edges and lines. The first website has a pattern of lines in the hero background, which are represented in the saliency map as well. For website 2, the edges of the driving trucks and lines of the road they are driving on are captured in the saliency map. It is also possible to identify the red navigation bar. The saliency map of website 3, the Bitcoin website, also reflects the ability to separate edges. Looking at the original website, a radial gradient can be seen surrounding the symbols at the bottom half of the page. The saliency map shows that the model is able to identify the edge of this gradient, in addition to considering the stapled lines between the objects. It can also be seen that the lower half of the website has more of the model's attention compared to the upper half, which contains the text and navigation bar etc.

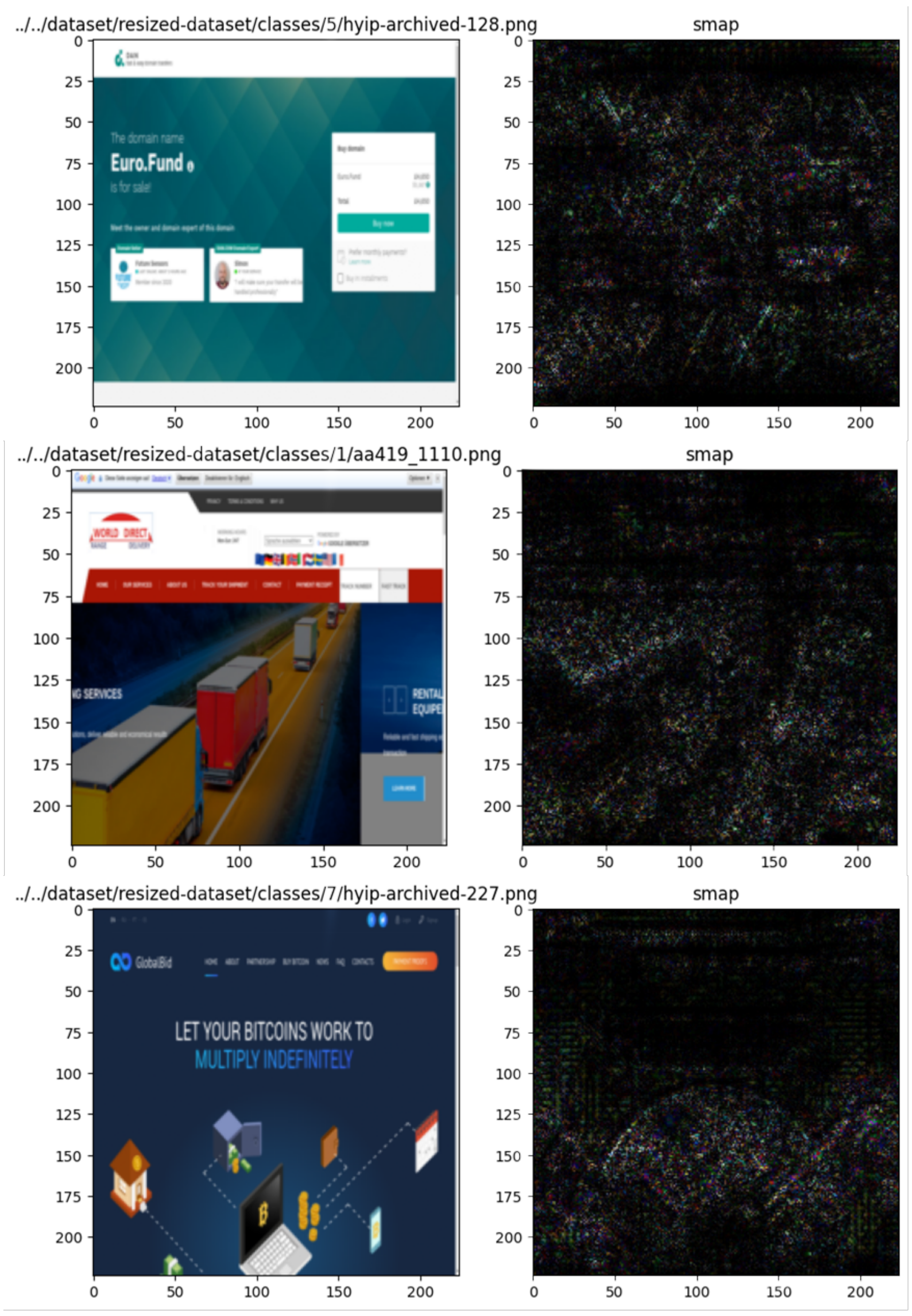

../../dataset/resized-dataset/classes/5/hyip-archived-128.png
smap
The domain name
Euro.Fund
is for sale!
../../dataset/resized-dataset/classes/1/aa419_1110.png
smap
WORLD DIRECT
../../dataset/resized-dataset/classes/7/hyip-archived-227.png
smap
GlobalBid
LET YOUR BITCOINS WORK TO
MULTIPLY INDEFINITELY

Figure 5.9: Low-level features capturing lines, edges, etc.

Figure 5.10 shows another example of saliency maps created from low-level features. Analyzing the first website, it is possible to distinguish the background fill on the left from the image on the right on the saliency map. The model seems to struggle with recognizing text, as this is represented with black, empty areas. In website 2's saliency map, it is possible to identify the edges of the people in the image. With a close look, the woman to the right and her handshake are well represented.

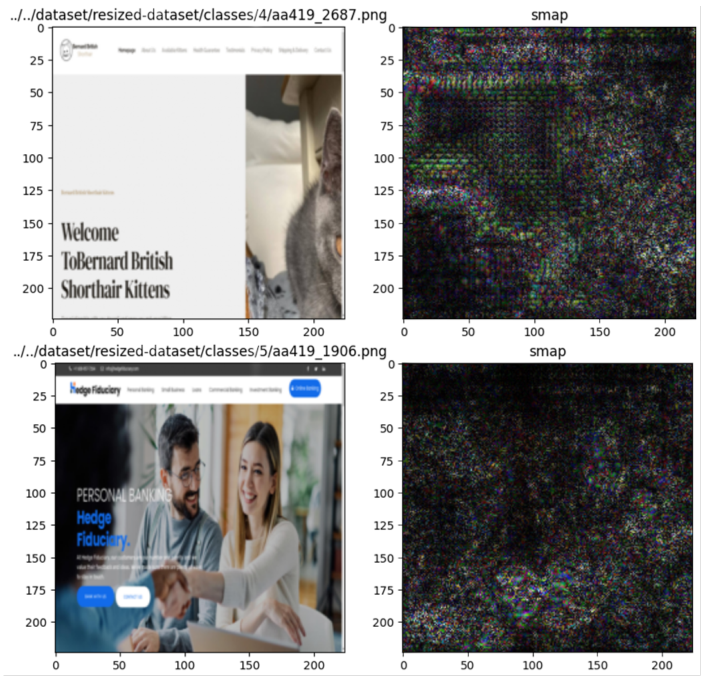

../../dataset/resized-dataset/classes/4/aa419_2687.png
smap
Welcome ToBernard British Shorthair Kittens
../../dataset/resized-dataset/classes/5/aa419_1906.png
smap
Hedge Fiduciary
PERSONAL BANKING
Hedge Fiduciary.
0
25
50
75
100
125
150
175
200

Figure 5.10: Low-level features capturing page structure and human edges within website images.

5.3.2 High-level feature extraction

The higher levels of ResNet-18 are more focused on capturing dynamic and unique features within images, such as colors, objects, and patterns [47]. Figure 5.11 shows generated saliency maps for the model's high-level features. Interpreting these saliency maps was harder compared to the low-level ones since the features are more specific and less general. By studying the maps closely, it is possible to recognize some patterns.

In the saliency map of website 1, it is possible to identify the noses of the cows. These can be seen with small pink circles. Website 2 captures the eyes of the monkey, while saliency map 3 is able to resemble the shape of the chickens. Similarly to low-level features, high-level features are not capturing website semantics, such as navbars, logos, or text. These elements are mostly black areas in the respective saliency maps.

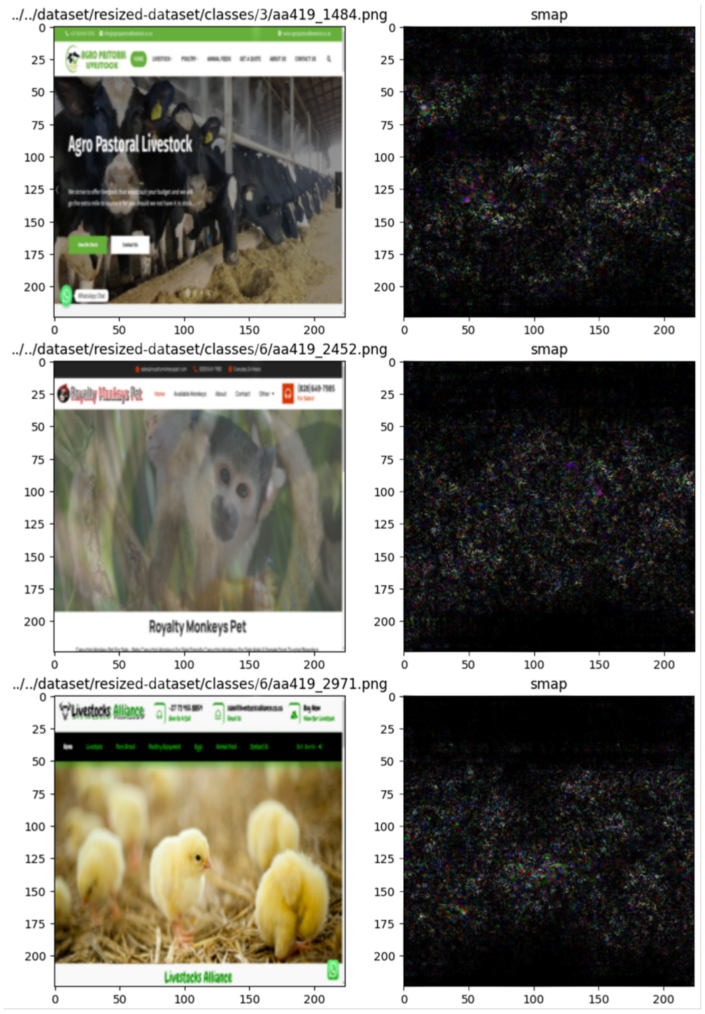

../../dataset/resized-dataset/classes/3/aa419_1484.png
smap
../../dataset/resized-dataset/classes/6/aa419_2452.png
smap
../../dataset/resized-dataset/classes/6/aa419_2971.png
smap

Figure 5.11: Example saliency maps for high-level features.

5.3.3 Summary and comparison

The above sections have considered the model trained using the diverse dataset on a pre-trained ResNet-18 architecture. Low-level features and high-level features were visualized respectively in order to interpret them individually. Analyzing the above saliency maps, it can be seen that the model is able to capture features on a low level. High-level features were harder to interpret. Regardless, it can be seen that the model is better at identifying photographs and illustrative objects within the websites, rather than website semantics and structure. The pre-trained version of ResNet-18 was trained on ImageNet-1k, which consists of 1000 classes of different photographs. Thus, the use of transfer learning, combined with the lack of website training data, may have influenced the model to generalize better to photographs rather than website-specific elements.

5.4 Similarity threshold experiments

As stated in Section 4.1, we defined three website-similarity terms: dissimilar, similar, and near-duplicates. However, the trained models only label pairs as either dissimilar (0) or similar (1). In the following Section, experiments regarding the model's ability to distinguish these categories will be showcased. This includes inspecting the outputted similarity scores for compared pairs during prediction. Using these similarity scores, the model's ability to consider dissimilar, similar, or near-duplicate samples is assessed individually. The section addresses **RQ3**, regarding the definition of similarity thresholds.

The *simset*, as described in Section 4.2.2, was used to inspect similarity values. As opposed to the small and diverse datasets, the simset only contains 3 classes of website screenshots. These represent the *dissimilar* class, *similar* class, and *near-duplicate* class respectively. Samples in the *dissimilar* class were unclassified during the data population process (Section 4.2.2), and not part of any class that the models were trained on. This was necessary in order to populate pairs with a sufficient amount of dissimilarity. Neither of the samples in the three classes was part of the models training data.

Section 4.3 described how the generation of training and testing pairs were conducted. Samples belonging to the same class received target 1 (similar), while samples paired from different classes received target 0 (dissimilar). For this approach, an additional target (2) for near-duplicates was needed. The population of the prediction dataloader was thus changed to take this into account. The process was as follows:

1. Select a random class in the *simset* (1-3).
2. Select a random image within the selected class as image 1.
3. If the near-duplicate class (3) was chosen, randomly select image 2 from the same class. The target is set to 2.
4. If a similar class (2) is chosen, randomly select image 2 from the same class. The target is set to 1.
5. If the dissimilar class (1) is chosen, select image 2 either from class 2 or 3. This produces a pair with dissimilar samples: 1 from the dissimilar class, and 1 from any other class. The target is set to 0.

It was only necessary to run predictions through the already trained models to retrieve information about the computed similarity scores. The modified *simset* dataloader was passed through both the small and diverse models with 3 epochs each.

5.4.1 Similarity computation - small model

Figure 5.12 represents a graph with all pairs compared during the 3 epochs on the small model. The x-axis is divided into percentiles of 10% each. The y-axis shows the total count of comparisons residing within the specific percentiles. Graph colors visualize each class respectively.

Most comparisons on the small model ended up with a similarity score between 70%-90%. This was regardless of whether the pairs were dissimilar, similar, or near-duplicates. It is visible that the near-duplicates have a higher concentration within this percentile. Dissimilar samples would optimally be concentrated within the lower percentiles since these are less similar to each other. Instead, *dissimilars* have a higher concentration than *similars* in the 70%-90% range. This indicates that the small model has not generalized well enough to distinguish (dis)similar samples. The high volume of *dissimilars* within this range also reflects the amount of FP predictions for the small model.

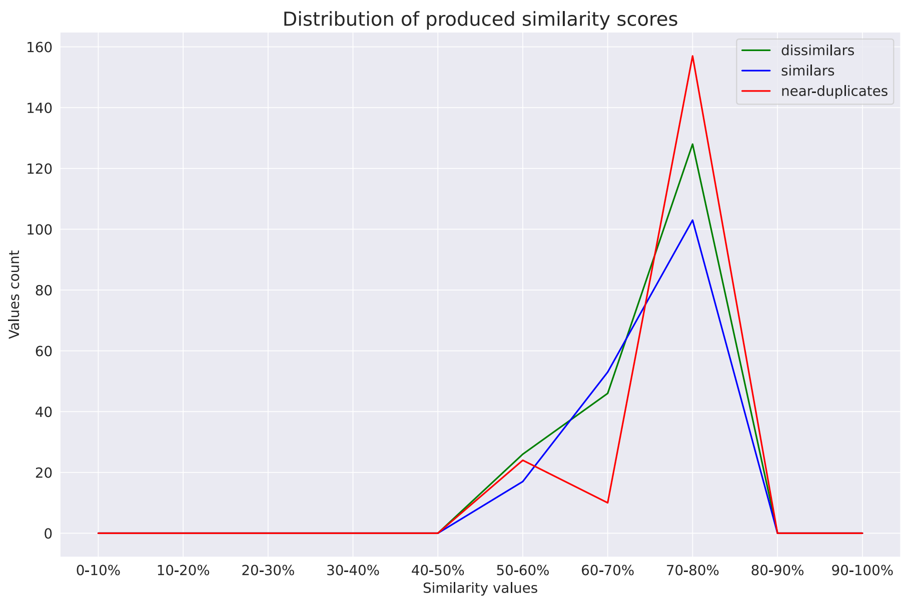


Figure 5.12: Distribution of similarity prediction scores for the small model.

5.4.2 Similarity computation - diverse model

Figure 5.13 shows the graph for all pairs compared in the similarity calculation for the diverse model. The diverse model has different similarity distribution compared to the small model. Most of the comparisons between near-duplicate samples are within the 70%-90% range. On the other hand, *similar* predictions are also very well represented within this percentile. Thus, labeling the difference between near-duplicates and "just similar" samples may be challenging for the model. Regardless, the high volume of near-duplicate and similar predictions within this percentile shows that the model is able to distinguish similarities from dissimilarities with these percentiles as criteria.

The dissimilar predictions show better results for the diverse model compared to the small model. Approximately 70 predictions are within the 20%-30% range. Having comparisons from this class within the lower percentiles shows that the model is able to distinguish dissimilar websites. A spike in dissimilar comparisons can also be seen in the 70%-90% percentiles, however. This indicates some challenges in dissimilar distinguishing and is also reflected in the model's false predictions evaluated in Section 5.2.2. Based on the graph, a similarity threshold of at least 70% could be suitable for creating a better distinction between similar and dissimilar pairs.

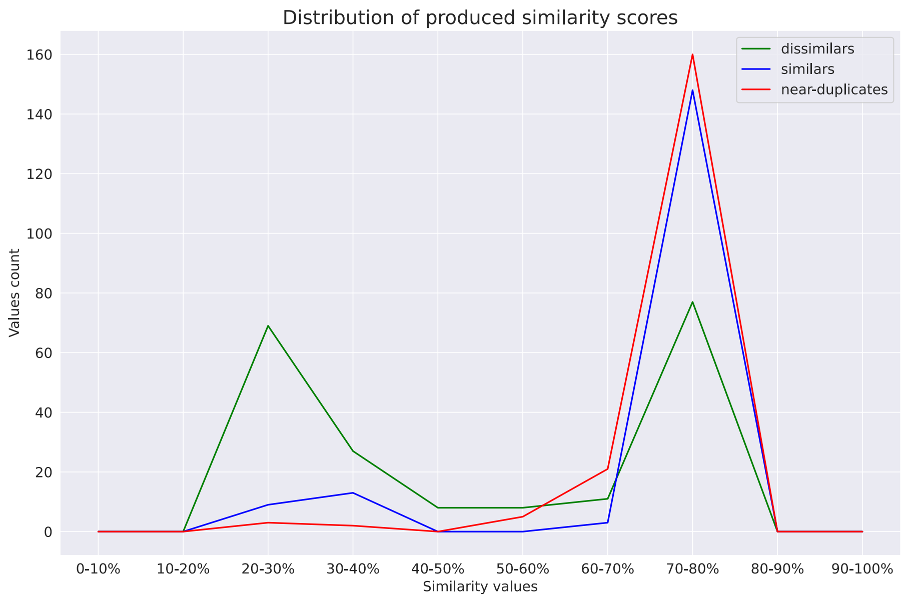


Figure 5.13: Distribution of similarity prediction scores for the diverse model

5.4.3 Threshold experimentation

Adjustments to the similarity score thresholds were tested to review how this affects model performance. As stated in Chapter 4, the model labels a pair as similar if the computed similarity score is above or equal to 0.5 (50%). Experiments showed that similar/near-duplicate pairs most often have similarity scores between 70% - 90%. Prediction thresholds were thus adjusted, such that only pairs having a score equal to or above 70% were labeled as similar.

Tests with this threshold were run on both the small and diverse datasets. This also involved re-trained the models with this threshold. Table 5.6 shows performance for the small model with a 0.7 threshold. Other hyperparameters used were identical to those described in Section 5.1. Accuracy values from previous results, with a threshold of 0.5, have also been included for comparison. Both training and prediction accuracy is slightly improved when the threshold is increased. This could be due to the number of dissimilar samples being within the 50% - 70% percentiles (refer to Figure 5.12). Increasing the similarity threshold excludes these dissimilar pairs from being labeled as similar, as compared to when the threshold was at 0.5. It should also be noted that the number of TNs and FNs has increased. This is due to the threshold making it harder to label positives.

Small dataset			
Similarity score >= 0.5		**Similarity score >= 0.7**	
Training accuracy	**Prediction accuracy**	**Training accuracy**	**Prediction accuracy**
67%	64%	70%	66%
TPs	**TNs**	**FPs**	**FNs**
780	639	361	220

Table 5.6: Achieved performance with 0.7 as the similarity threshold - small model.

Table 5.7 shows achieved performance for the diverse dataset when the threshold was set to 0.7. It can be seen that the performance is lower than seen previously using 0.5 as the threshold. As seen for the small dataset, the number of negatives has increased. This is due to the requirements for being labeled as similar have increased. Referring to Figure 5.13, the near-duplicates within the 50% - 70% percentiles are likely contributing to this since these do not meet the threshold requirement of 0.7. It is possible that experiments with 0.7 as the threshold would improve over time with more training data available. This may also be the case with other threshold values (i.e. 0.6). Although overall performance decreased, validation with 0.7 as the threshold may be more representative since most *similar* and near-duplicates pairs resided within the 70%-90% percentiles.

Diverse dataset			
Similarity score >= 0.5		**Similarity score >= 0.7**	
Training accuracy	**Prediction accuracy**	**Training accuracy**	**Prediction accuracy**
69%	73%	67%	69%
TPs	**TNs**	**FPs**	**FNs**
2235	1499	1201	465

Table 5.7: Achieved performance with 0.7 as the similarity threshold - diverse model.

Chapter 6

Discussion, Conclusion and Further Work

The previous chapters have described how criminals operate more advantageously than law enforcement with website replication. Existing work within the field of criminal website disruption was assessed, and research questions were defined with this in mind. Project methodology along with a description of the underlying theory was presented, followed up by experiments to answer the research questions. The following chapter will discuss the resulting outcome of previously shown experiments. This includes belonging implications, recommendations, concluding words, and further work.

6.1 Theoretical implications

Research within classification and detection of criminal websites are also previously explored topics. Existing work in the field was identified and reviewed. We discovered that most machine learning techniques for criminal website classification/detection are based upon textual features. These include features such as a website's HTML tags, HTML-element occurrences, positioning, and sentences. Less research focusing on the visual elements of a website existed. Moreover, most machine learning methods attempted for this purpose previously were, to our knowledge, traditional clustering and classification methods. The use of NNs for this purpose was not identified in reviewed literature.

6.1.1 Research question 1

How precise are siamese neural networks in identifying similarities between screenshots of fraudulent websites?

Existing research showed that similarities are often persistent in replicated criminal websites. Replication of existing websites with minor tweaks allows actors to replicate fraud campaigns with efficiency and persistence. We thus purposed a method to label compared websites as either similar or dissimilar. Using CNNs as sub-networks, pairs of websites screenshots and their corresponding label (similar/dissimilar) are processed. The SNN computes the similarity between the two outputted embeddings.

The amount of data input data (website screenshots) collected was not significant. Typically, larger amounts of data are required when training deep NNs. We utilized a pre-trained ResNet-18 model with ImageNet-1K V2 weights as a baseline for further training. The initial thought was that this could strengthen the model since the amount of collected data was too small to train an accurate model alone.

Conducted experiments show that SNNs has the potential for being utilized within criminal website detection where similarities are present. This is supported by the fact that performance increased when more data was added, comparing the small model to the diverse one. The addition of data augmentation techniques to training data as seen in Listing 4.1 also increased prediction accuracy by approximately 5%.

Our experiments hence contributed to new research within the field of criminal website detection. Using screenshots of websites as an input medium for a NN has shown to work. This reduces the amount of effort needed to manually identify key features, by letting the network learn from inputted images. With more sophisticated data collection and bigger-scale training, our proposed method may become more trustworthy in a real-world application. The thesis has also strengthened knowledge from existing work, showing that persisting website similarities may indeed be utilized in detection through distance metric methods.

6.1.2 Research Question 2

How can the trained model be interpreted to analyze relevant features learned from fraudulent website screenshots?

The diverse model was interpreted in order to analyze whether it was able to capture features of website screenshots. The goal of this process was to assess what website elements the sub-CNNs were able to identify after training. Moreover, this insight could give an indication of which website semantics that would be interesting to focus on in further work.

Since we are working with images, interpretation of pixel attribution visualization can give information on the network's ability to capture features. Saliency maps were generated for arbitrary samples in the test dataset to visualize pixel attribution. As seen in Section 5.3, it was possible to recognize parts of the website through low-level saliency maps. This shows that the model is able to recognize website elements. A challenge was also uncovered through this analysis. The model is seemingly more robust at identifying photographs in websites, rather than semantic website elements (navbars, positional elements, etc.). This is most likely due to the use of pre-trained ImageNet-1k weights, as these are trained on various photographs to solve universal problems. Photographs are dynamic elements likely to change throughout websites. It was seen during data collection that logos and image assets also change in near-duplicate websites. This introduces challenges since similarities often persist in semantic website elements as mentioned above, rather than in photograph assets.

During our work, we have showcased how transfer learning can support training a model when insufficient amounts of data are available. The model was able to capture features such as lines, gradients, edges, and patterns using this approach. On the other hand, results showed how transfer learning may also greatly influence the types of features considered by the model. Further work can take this into account when training deep NNs for this purpose. Although pre-trained models can compensate for the lack of data, they cannot necessarily adapt to the problem at hand. For the sake of website screenshots as the input medium, training a model from scratch explicitly on website screenshots may make the model more clever at capturing website semantics. Then again, this requires more data. Experimentation with different pre-trained models or architectures could also impact the ability to capture features of relevance.

6.1.3 Research Question 3

What are possible similarity thresholds for determining whether pairs of websites are dissimilar, similar, or near-duplicates?

Through initial experiments, it was shown that SNN as an algorithm has the potential for being utilized to identify similarities within fraudulent websites. The proposed method labeled unseen testing pairs of websites into two categories: Similar (1) or dissimilar (0). As previously discussed, three different grades of similarity were defined: Dissimilar pairs, similar pairs, or near-duplicate pairs. By investigating this research question, we wanted to explore whether the trained models were able to distinguish between not only (dis)similar pairs but also similar and near-duplicate pairs. More specifically, the similarity score that the SNN model computes for pairs within these categories respectively was inspected.

Results showed that most of the near-duplicate samples have a similarity score of 70% - 80% for the small model. The model struggles however with the fact that most dissimilar and similar pairs also compute similarity scores within these percentiles. This is also reflected by the amount of FP predictions and signs of overfitting. The small model was thus not representative enough to provide answers to this research question.

Similarity threshold experiments done on the diverse model show more trustworthy results, considering that the model generalized better. Most similar and near-duplicate pairs receive a similarity score within the 70% - 80% percentile. Our experiments thus show that a threshold of 0.7 can work as the minimum condition for labeling similar or near-duplicate pairs. Distinguishing between *similar* and near-duplicate pairs however was challenging, since they mostly relied upon the same percentile. It should be taken into account that the models were only trained on outputting two targets: dissimilar (0) or similar (1). If the models were trained to output the near-duplicate target (2) as well during training, it would perhaps be able to distinguish these two categories better. Nevertheless, it is clear that a minimum threshold of 0.7 can be suitable to identify *similar* and near-duplicate pairs from dissimilar ones. This theory was also strengthened through retraining a model using a 0.7 threshold rather than 0.5, as shown in Section 5.4.3.

Our experiments present examples of similarity thresholds that can be utilized to identify similarities in further work. Although previous work showcased ***what*** types of similarities often persist within criminal websites, it has not been discussed ***how*** much replicated criminal websites differ in variance. At least to our knowledge. The achieved results show that a similarity score above or equal to 70% should be considered when labeling similar samples in the future.

Similarity scores presented for this research question are not conclusive. They do however provide a starting point for creating a threshold condition in other appliances. The grade in which *similar* websites differ from near-duplicates remains to be explored in further work.

6.2 Practical recommendations

Website screenshots were collected in order to populate the datasets. The collection used an automated web crawler to visit fraudulent websites and capture screenshots. For training a more accurate and robust model, it is recommended to upscale the data collection process. This would also allow the model to generalize better to new data. Data collection can be conducted over longer periods of time by monitoring online data sources regularly. This approach was for instance utilized by Drew & Moore [4] and Westlake et. al. [9], who collected data by monitoring data sources over several months.

Manual data gathering and dataset population made the experiments hard to reproduce. Samples gathered from other sources that vary thematically could produce other results, even though the implementation is identical. Manual sorting of dataset classes introduced subjectivity in the way samples were interpreted and arranged. To strengthen the integrity of the populated datasets, clustering could be utilized to group samples. This could reduce the grade of subjectivity in the process and make population more reproducible. However, when dealing with supervised learning, manual class distribution may indeed be more suitable to ensure correctness. Benefiting from more human resources to sort and quality assure the dataset population is perhaps more feasible if possible.

The technical implementation on the other hand is easy to re-implement. The SNN was based upon PyTorch's implementation example [31], and adapted further for this task. Some changes were made to how the program loads data, the training, the validation, and the model structure. The principles are still the same, however. To support development, other easy-to-use frameworks such as `numpy`, `SciKit-learn` and `pandas` were utilized. These are well documented packages easy to integrate with Python machine learning applications. `Selenium` is also well documented and straightforward to use for website crawling.

Re-production of the experiments in a bigger scale scenario (i.e. in digital forensics) could be resource intensive. Although the framework and third-party packages are free and easy to implement, training the SNN requires sufficient computational resources. Using CNNs and images as the input data is especially computationally expensive compared to using text as input data. The addition of data and deeper architectures will also increase training time. Investing in sufficient Graphical Processing Units could be economically expensive and introduce high energy costs. Utilizing transfer learning will reduce the computational cost, however, since the amount of training needed is reduced. Implementers should keep these expenses in mind when assessing a project budget.

Our proposed method used a supervised learning approach. Pre-processing is more demanding in supervised learning since corresponding labels must be determined on beforehand. Implementers may benefit from attempting semi-supervised learning, where only a portion of the training data is supervised. The rest is learned through unlabelled data. This can save a lot of resources and time consumption during pre-processing.

Digital forensics investigators could benefit from a SNN by combining it with other website detection techniques. A practical scenario could be the detection of newly spawned darknet marketplaces. A separate crawler entity could for instance gather information on new webpages spawned on darknet once every 24 hours. Assuming that darknet markets of interest have persisting website similarities (i.e. sharing front-end framework), the crawler could automatically pipe a screenshot for comparison to the trained SNN. Depending on similarity thresholds in place, alerts could be created in a front-end system. This is one example of how the model can be utilized in practice.

6.3 General discussion

Website replication techniques to host fraudulent campaigns will continue to challenge website takedown processes in the future. As described by Moore and Clayton [8], it will never be possible to fully cope with the amount of digital criminal activity. Fraudulent website detection through SNNs could hence aid in earlier detection and disruption of such websites.

Supervised classification algorithms and unsupervised clustering methods have provided relevant results in previous research. Especially methods considering text-based website features have produced significant results. Using images as the input medium for this purpose was, to our knowledge, not as common. Neither was the utilization of NNs. Considering that similarities often are persistent in fraudulent websites, SNNs may perhaps be the most optimal candidate for such tasks, being a metric learning algorithm

Using website screenshots (images) as input rather than text-based features (i.e. website source code) introduces differences in how the applied algorithm learns. Using screenshots, all visible elements on the website can be interpreted. This allows training of a machine learning algorithm that interprets classification based on what it sees directly, without additional information such as metadata. The machine sees the same elements as the human sees. Likewise, this poses a challenge in terms of analyzing hidden elements on fraudulent websites. Malicious source code, website traffic, and URL parameters are not considered in our approach. What cannot be seen visually and hence cannot be considered. Future work could benefit from adding functionality for interpreting hidden indicators as well.

Through our conducted experiments, we have shown that SNNs have potential to detect similarities in fraudulent websites. However, the limited availability of data did limit performance and ambition. Digital forensics investigators and security researchers with appropriate resources at hand can utilize accomplished experiments in further work. They may give an indication for future decision criteria, such as determining when websites are considered similar, and whether transfer learning is a suitable approach.

6.4 Conclusion

Criminal operations hosting fraudulent websites operate more advantageous than law-enforcements. Replicating a website, i.e. through TLDH, requires less time and resources compared to a full website takedown procedure. Disruption of such operations could hence increase the amount of resources and complexity that the criminals must put into their work. Therefore, one of the intended goals of this thesis was to investigate untested methods that could aid in criminal website detection and disruption.

Existing research showed how similarities are likely to persist within certain categories of criminal websites [4]. We have shown the usage of SNNs to identify similarities in screenshots of HYIP, escrow, and *fee fraud* websites.

The first research question as defined in Chapter 1 was related to how SNN as an algorithm can be utilized for this purpose. Our results showed that the *small model* was most inaccurate and overfit. Validation of the *diverse model* showed that the SNN were able to label near-duplicate websites to a great extent. Several near-duplicate classes had a prediction accuracy of approximately 80%. The model also was also able to identify less obvious similarities. Overall model validation yielded a collected F1-accuracy of 78%. This shows that SNNs are able to identify website similarities and patterns with screenshots as input data. This implies training in a bigger scale scenario with more data available to prevent underfitting.

The second research question was related to analyzing whether the trained model was able to capture relevant features in the website screenshots. Saliency maps were utilized to interpret pixel attribution on a set of test images. Our analysis showed that the model was capable of capturing low-level features such as edges, lines, and patterns in website screenshots. More specifically, the model did better at identifying the photographs used in these websites, rather than persisting website semantics. Using transfer learning with pre-trained weights has likely influenced this. Training without pre-trained weights could make the model capture features more relevant for website semantics.

Lastly, experiments related to the third research question discussed adjustments of similarity thresholds. Initially, the models were trained to distinguish dissimilar and similar samples, based on whether computed similarity was above or below 50%. Experiments attempted to investigate how the degree of similarity distinguishes dissimilar, similar, and near-duplicate websites. Our result showed that a minimum threshold of 70% can be used to distinguish *similar* and near-duplicate pairs from dissimilar ones. However, the model was not able to distinguish between *similar* and near-duplicate websites. The 70% similarity threshold could be utilized as a starting point in future work.

The findings in this thesis have provided a contribution towards untested methods to support criminal website disruption or detection. By exploiting persistent similarities in replicated websites, we have shown SNNs ability to identify these through limited training. SNN may hence be reliable on a bigger scale. In practice, automated solutions could utilize SNNs to detect re-appearing websites if generalized properly. Digital forensics could benefit from this by identifying replication at a faster scale, further causing disruption. We hope that our work has motivated further research to continue the exploration of SNNs in criminal website detection & disruption.

6.5 Further work

Based on conducted experiments and achieved results, we have highlighted some key takeaways that can be improved upon to strengthen future research.

Population of a more balanced and bigger dataset
The populated dataset of fraudulent websites used in this thesis was limited in size. Collection was conducted within a limited period of time, affecting the total amount of collected samples. Manual grouping of website screenshots into classes was also conducted. This introduces subjectivity into the way website (dis)similarities was interpreted.

It is encouraged to conduct website screenshot collection over a longer period of time to populate a bigger dataset. This could be performed by monitoring fraud website trackers, such as `aa419.org`, over time. If manual dataset population is conducted, more human resources should be utilized to quality assure classes and website samples. It is also advised to explore automated methods such as clustering or k-fold cross validation.

Training a model without using transfer learning
Transfer learning was utilized on a ResNet-18 architecture with pre-trained ImageNet-1k weights. This was done to accompany for lack of available data. Pixel attribution discovered that the use of pre-trained weights may have generalized the model more towards identifying photographs within website screenshots, rather than the website's structure.

It is advised to attempt training a model without using weights from a pre-trained model as a baseline. This could allow the model to generalize better towards website semantics, rather than photographs displayed on websites. To maintain performance with this approach, a high amount of training data must be available. Experimenting with other pre-trained models may also produce results of interest.

Experimenting with other CNN architectures
The ResNet-18 architecture was used for both the small and diverse models. Experiments showed that the small model was overfitting. Changing to a less complex architecture that is not as deep may hence improve performance on smaller datasets. Results from the diverse model were the opposite, suggesting that this model underfitting. Apart from adding more data, changing to a more complex architecture could improve performance.

Additional feature visualization techniques
Features were extracted from the sub-CNNs lower and higher layers respectively and visualized through saliency maps. It is encouraged to explore other CNN visualization techniques, in order to further interpret relevant features within fraud website identification. This could for instance be other visualization techniques, as

for instance, *principal component analysis (PCA)* or *t-Distributed Stochastic Neighbor Embedding (t-SNE)*.

Combining both website screenshots and website source code as input data
SNNs have been tested with website screenshots as the input data. As stated in Chapter 3, previous research in criminal website detection has been mostly focused on text-based features. It is encouraged to take both existing work and the experiments conducted here in combination. A SNN considering both text-based features and visual features could have richer knowledge regarding a website's relevant properties.

siamese neural network with one-shot learning
SNNs may be developed with one-shot/few-shot when less data is available. In this thesis however, the amount of data available was concluded as not being sufficient enough. Available data was thus utilized for training in order to get as many training pairs as possible. When sufficient data are available, it is encouraged to attempt a one-shot/few-shot approach. This could also be more scalable in a real-world scenario.

Using TripletLoss as the loss-function
Our developed models use *ContrastiveLoss* from Pytorch Metric Learning [45] as the loss function. This function takes image pairs as inputs. *TripletLoss* has grown popular with the use of SNNs. It may be able to learn more precisely since it uses a 'ground truth' anchor image in addition to the compared pairs. It is encouraged to explore the use of *TripletLoss* when not limited to the amount of data available.

Validating the implementation using another dataset.
Our experiments have only been evaluated on provided training data and unseen test samples. To maintain performance integrity, it is advised to adapt and run the source code against a dataset where we know the desired accuracy beforehand. Examples could be running the implementation with the popular *Omniglot* or *CIFAR10* datasets. By doing so, it is possible to determine whether experiment bias is caused by the utilized datasets or the implemented program.

Experimentation of near-duplicates and similarity scores.
Trained models used a similarity score above or equal to 0.5 in order to determine two samples as similar. 0.7 as the similarity score was also attempted in Section 5.4, considering that most *similar* and near-duplicate pairs have scores computed within this range. It is advised to inspect the similarity threshold further. Adding a target for near-duplicates to the model output, as described in Section 6.1.3, is also encouraged for further exploring where the threshold between *similar* and near-duplicate samples.

Adding additional overfitting mitigations

When overfit, the model will generalize too well towards the training data, making it inaccurate for predicting unknown data. As seen in Section 4.3.3, some measures were implemented to prevent overfitting. To mitigate the possibility of overfitting further, techniques such as *dropout* and *regularization* could be implemented.

Bibliography

[1] B. G. Westlake, M. Bouchard, 'Criminal careers in cyberspace: Examining website failure within child exploitation networks,' *Justice Quarterly*, vol. 33:7, pp. 1154–1181, 2015. DOI: 10.1080/07418825.2015.1046393.

[2] A. Hutchings, R. Clayton, R. Anderson, 'Taking down websites to prevent crime,' *2016 APWG Symposium on Electronic Crime Research (eCrime)*, pp. 1–10, 2016. DOI: 10.1109/ECRIME.2016.7487947.

[3] IWF - Internet Watch Foundation, *What is top-level domain hopping?* Last accessed May 2nd 2023, 2021. [Online]. Available: https://annualreport2020.iwf.org.uk/trends/international/other/toplevel.

[4] J. M. Drew, T. Moore, 'Optimized combined-clustering methods for finding replicated criminal websites,' *EURASIP J. on Info. Security*, vol. 14, 2014. DOI: 10.1186/s13635-014-0014-4.

[5] A. Årnes, *Digital Forensics*, A. Årnes, Ed. John Wiley & Sons Inc, 2017, ISBN: 9781119262381.

[6] M. I. Jordan, T. M. Mitchell, 'Machine learning: Trends, perspectives, and prospects,' *Science*, vol. 349, pp. 255–260, 2015. DOI: 10.1126/science.aaa8415.

[7] C. M. Bishop, 'Neural networks and their applications,' *Review of Scientific Instruments*, vol. 65:6, pp. 1803–1832, 1994. DOI: 10.1063/1.1144830.

[8] T. Moore, R. Clayton, 'Examining the impact of website take-down on phishing,' *eCrime Researchers Summit 07*, vol. 07, pp. 1–13, 2007. DOI: 10.1145/1299015.1299016.

[9] B.G. Westlake, M. Bouchard, A. Girodat, 'How obvious is it? the content of child sexual exploitation websites,' *Deviant Behaviour*, vol. 38:3, pp. 282–293, 2017. DOI: 10.1080/01639625.2016.1197001.

[10] C. Olston, M. Najork, 'Web crawling,' *Foundations and Trends® in Information Retrieval*, vol. 4:3, pp. 175–246, 2010. DOI: 10.1561/1500000017.

[11] J. Peng, Y. Ma, F. Zhou, S. Wang, Z. Zheng, J. Li, 'Web crawler of power grid based on selenium,' *16th International Computer Conference on Wavelet Active Media Technology and Information Processing*, pp. 114–118, 2019. DOI: 10.1109/ICCWAMTIP47768.2019.9067730.

[12] C. Janiesch, P. Zschech, K. Heinrich, 'Machine learning and deep learning,' *Electron Markets*, vol. 31, pp. 685–695, 2021. DOI: 10.1007/s12525-021-00475-2.

[13] S. Russel, P. Norvig, *Artificial Intelligence: A modern approach*, T. Dunkelberger, Ed. Pearson Education, 2010, vol. 3.

[14] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[15] M. Mohri, A. Rostamizadeh, A. Talwalkar, *Foundations of machine learning, Second Edition*, F. Bach, Ed. MIT Press, 2018, ISBN: 9780262039406.

[16] J. Nabi. 'Machine learning —fundamentals.' (2018), [Online]. Available: https://towardsdatascience.com/machine-learning-basics-part-1-a36d38c7916 (visited on 26/02/2023).

[17] I. Kononenko, M. Kukar, *Machine Learning and Data Mining: Introduction to Principles and Algorithms*. Horwood Publishing Limited, 2007.

[18] N. K. Chauhan, K. Singh, 'A review on conventional machine learning vs deep learning,' *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, pp. 347–352, 2017. DOI: 10.1109/GUCON.2018.8675097.

[19] S. Mahapatra. 'Why deep learning over traditional machine learning?' (2018), [Online]. Available: https://towardsdatascience.com/why-deep-learning-is-needed-over-traditional-machine-learning-1b6a99177063 (visited on 26/02/2023).

[20] V. Lyashenko. 'Understanding few-shot learning in computer vision: What you need to know.' (2023), [Online]. Available: https://neptune.ai/blog/understanding-few-shot-learning-in-computer-vision (visited on 22/05/2023).

[21] M. Boudiaf, J. Rony, I. M. Ziko, E. Granger, M. Pedersoli, P. Piantanida, I. B. Ayed, 'A unifying mutual information view of metric learning: Cross-entropy vs. pairwise losses,' *European Conference on Computer Vision (ECCV)*, 2021. DOI: 10.48550/arXiv.2003.08983.

[22] S. Sharma, S. Sharma, 'Activation functions in neural networks,' *International Journal of Engineering Applied Sciences and Technology*, vol. 4:12, pp. 310–316, 2020. DOI: 10.33564/IJEAST.2020.v04i12.054.

[23] K. O'Shea, R. Nash, 'An introduction to convolutional neural networks,' *Neural and Evolutionary Computing (cs.NE)*, 2015. DOI: 10.48550/arXiv.1511.08458.

[24] J. Wu. 'Introduction to convolutional neural networks.' (2017), [Online]. Available: https://cs.nju.edu.cn/wujx/paper/CNN.pdf (visited on 20/03/2023).

[25] J. Bromley, I. Guyon, Y. LeChun, E. Säckinger, R. Shah, *Signature Verification using a Siamese Time Delay Neural Network*, J. Cowan, G. Tesauro, J. Alspector, Ed. Morgan-Kaufmann, 1993, vol. 6. [Online]. Available: `https://proceedings.neurips.cc/paper/1993/file/288cc0ff022877bd3df94bc9360b9c5d-Paper.pdf`.

[26] Q. Guo, W. Feng, C. Zhou, R. Huang, L. Wan, S. Wang, 'Learning dynamic siamese network for visual object tracking,' *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 1781–1789, 2017. DOI: `10.1109/ICCV.2017.196`.

[27] G. Koch, R. Zemel, R. Salakhutdinov. 'Siamese neural networks for one-shot image recognition.' (2015), [Online]. Available: `https://www.cs.cmu.edu/~rsalakhu/papers/oneshot1.pdf` (visited on 20/03/2023).

[28] R. Hadsell, S. Chopra, Y. LeCun, 'Dimensionality reduction by learning an invariant mapping,' *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, pp. 1735–1742, 2006. DOI: `10.1109/CVPR.2006.100`.

[29] F. Schroff, D. Kalenichenko, J. Philbin, 'Facenet: A unified embedding for face recognition and clustering,' *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015. DOI: `10.1109/CVPR.2015.7298682`.

[30] S. Benhur, *A friendly introduction to siamese networks*, 2020. [Online]. Available: `https://towardsdatascience.com/a-friendly-introduction-to-siamese-networks-85ab17522942` (visited on 26/04/2023).

[31] PyTorch, *Siamese network example*, Last accessed May 1st 2023, 2023. [Online]. Available: `https://github.com/pytorch/examples/tree/main/siamese_network`.

[32] B. Lake, R. Salakhutdinov, J. Gross, J. B. Tenenbaum, 'One shot learning of simple visual concepts,' *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 33, 2011.

[33] S. Want, *Few-shot learning (1/3): Basic concepts*, Last accessed May 22nd 2023, 2020. [Online]. Available: `https://www.youtube.com/watch?v=hE7eGew4eeg&t=677s`.

[34] N. Ketkar, *Introduction to PyTorch*. Apress, 2017, pp. 195–208, ISBN: 978-1-4842-2766-4. DOI: `10.1007/978-1-4842-2766-4_12`.

[35] PyTorch, *Torchvision models*, 2017. [Online]. Available: `https://pytorch.org/vision/0.8/models.html` (visited on 21/03/2023).

[36] MathWorks, *Resnet50*, Last accessed March 19th 2023, 2023. [Online]. Available: `https://www.mathworks.com/help/deeplearning/ref/resnet50.html`.

[37] ImageNet, *About imagenet*, Last accessed March 19th 2023, 2023. [Online]. Available: `https://www.image-net.org/about.php`.

[38] T. Moore, J. Han, R. Clayton, 'The postmodern ponzi scheme: Empirical analysis of high-yield investment programs,' *Financial Cryptography and Data Security. FC 2012. Lecture Notes in Computer Science*, vol. 7397, 2012. DOI: 10.1007/978-3-642-32946-3_4.

[39] S. Morbieu, G. Bruneval, M. Lacarne, M. Kone, F.-X. Bois, 'Main content extraction from web pages,' *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, vol. 19, pp. 1002–1005, 2020. DOI: 10.1109/ICMLA51294.2020.00162.

[40] S. Yevhen, *Hyip dataset: Data collected from hyip monitors and seo services*, Last accessed March 2nd 2023, 2018. [Online]. Available: https://www.kaggle.com/datasets/nikshev/hyip-dataset.

[41] Escrow-Fraud.com, *Stop escrow fraud*, Last accessed March 18th 2023, 2023. [Online]. Available: https://escrow-fraud.com/search.php.

[42] Artists Against 419, *Advance fee fraud*, Last accessed March 7th 2023, 2018. [Online]. Available: http://wiki.aa419.org/index.php/Advance_Fee_Fraud.

[43] The Internet Archive, *About the internet archive*, Last accessed March 19th 2023, 2023. [Online]. Available: https://archive.org/about/.

[44] E. T. Vestad, *Exploring siamese neural networks for similarity detection in fraudulent websites*, 2023. [Online]. Available: https://bitbucket.org/espur/mis4900-snn-fraudulent-website-detection.

[45] Kevin Musgrave, *Pytorch metric learning*, Last accessed March 20th 2023, 2023. [Online]. Available: https://kevinmusgrave.github.io/pytorch-metric-learning/.

[46] D. M. W. Powers. 'Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation.' (2020), [Online]. Available: https://doi.org/10.48550/arXiv.2010.16061 (visited on 28/04/2023).

[47] N. Si, W. Zhang, D. Qu, X. Lou, H. Chuang, T. Niu, 'Spatial-channel attention-based class activation mapping for interpreting cnn-based image classification models,' *Security and Communication Networks*, vol. 2021, pp. 1–13, 2021. DOI: 10.1155/2021/6682293.

[48] C. Molnar, *Pixel Attribution (Saliency Maps)*. Lean Publishing, 2023, ISBN: 978-0-244-76852-2. [Online]. Available: https://christophm.github.io/interpretable-ml-book/.

[49] S. Ruder. 'An overview of gradient descent optimization algorithms.' (2017), [Online]. Available: 10.48550/arXiv.1609.04747 (visited on 26/05/2023).

Appendix A

Data collection scripts

A.1 Website screenshot crawler script

Listing A.1 showcases how data was collected for the thesis. It receives a CSV-file with URLs as input, which is further converted to a pandas dataframe. A Selenium Chromium driver is configured with appropriate settings. The script will then iterate through all URLs in the dataframe to request snapshot availability on Wayback Machine. Based on the response code, the crawler will capture a screenshot from the archived webpage. Lastly, the URL is visited directly through the Selenium browser to capture the actively running website on the URL. This process was utilized to capture HYIP-websites, escrow-fraud websites, and fee-fraud websites.

Code listing A.1: The HYIP-dataset URL-crawler.

```
import requests as r
from selenium import webdriver
from selenium.webdriver.chrome.options import Options
from selenium.webdriver.common.by import By
from time import sleep
import pandas as pd

# Initialize dataset with URLs to dataframe
df = pd.read_csv('output.csv', usecols=[0, 3])

# Initialize chrome driver and appropriate options
chrome_options = Options()
chrome_options.add_argument("--headless")
chrome_options.add_argument("--no-sandbox")
chrome_options.add_argument("--window-size=1280,720")
browser = webdriver.Chrome(options=chrome_options)

# Loop through and visit each URL in the dataset over Wayback Machine
# and through direct visit if possible
for i in range(len(df)):
    print(f"Visiting website {str(i)}")
    url = str(df.loc[i, "site"])
    # Fetch URL-availability through WBM API
    # Date is set to Aug 2018, which is around the time
    # the HYIP dataset was published on Kaggle
    full_url = f"https://archive.org/wayback/available?url={url}&timestamp=20180820
        "
    response = r.get(full_url)
    data = response.json()
    # Check Whether wayback machine (WBM) archive exists
    try:
        if data['archived_snapshots']['closest']['status'] == '200' and data['
            archived_snapshots']['closest']['available'] == True:
            print("available")
            archived_url = data['archived_snapshots']['closest']['url']
            print(f"found archive {archived_url}")
            # Try to visit WBM record with Selenium browser.
            try:
                browser.get(archived_url)
                browser.implicitly_wait(120)
                # Remove WBM related elements before capturing screenshot
                element = browser.find_element(By.ID, 'wm-ipp-base')
                js_code = "arguments[0].style.display = 'none'"
                browser.execute_script(js_code, element)
                # Capture screenshot
                browser.save_screenshot(f'wayback-images/screenshot{str(i)}.png')
            except Exception as e:
                print(e)
    except Exception as e:
        print(e)

    # If the site is online, attempt capture screenshot without WBM,
    # This is in case the site has changed over time, allowing for more data
    try:
        response = r.get(url)
        if response.status_code == 200:
            try:
                browser.get(url)
                sleep(20)
                browser.save_screenshot(f"online-images/screenshot{str(i)}_2.png")
            except Exception as e:
                print(e)
    except Exception as e:
        print(e)
browser.close()
```

Appendix B

Feature Visualization

B.1 Pixel attribution and saliency maps

Listing B.1 shows the process of creating saliency maps through pixel attribution. A modified model is initially created, as described in Section 4.5. Input images are pre-processed and converted to PyTorch tensors. Gradients are computed for the outputted embeddings, which further generate the saliency maps. Lastly, the maps are plotted together with the original website screenshot.

Code listing B.1: Code for creating saliency maps.

```
import matplotlib as plt
import numpy as np
import torch
from torchvision import transforms
from PIL import Image
import torch.nn as nn

new_model = FeatureExtractor(old_model)

def org_img(inp_img, process):
    image_path = inp_img
    image = Image.open(image_path).convert('RGB')
    preprocess = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor()
    ])
    image = preprocess(image)
    if process:
        mean = [1.2295, 1.2129, 1.1906]
        std = [0.5329, 0.5126, 0.5415]
        normalize = transforms.Normalize(mean=mean, std=std)
        image = normalize(image)
    return image

def saliency(img1, img2, label):
    path1, path2 = img1, img2
    org_img1, org_img2 = org_img(img1, False), org_img(img2, False)
    img1, img2 = org_img(img1, True), org_img(img2, True)

    img1, img2 = img1.unsqueeze(0), img2.unsqueeze(0)
    if label == 0:
        print("True")
        target = torch.tensor(0, dtype=torch.int)
    else:
        print("false")
        target = torch.tensor(1, dtype=torch.int)
    img1, img2, target = img1.to(device), img2.to(device), target.to(device)
    img1.requires_grad_()
    img2.requires_grad_()
    output = new_model(img1, img2)
    loss = output[0, target.tolist()].sum()
    new_model.zero_grad()
    loss.backward(retain_graph=True)

    smap1 = img1.grad.abs().squeeze().cpu().detach().numpy().transpose((1, 2, 0))
    smap2 = img2.grad.abs().squeeze().cpu().detach().numpy().transpose((1, 2, 0))

    smap1, smap2 = torch.from_numpy(smap1), torch.from_numpy(smap2)
    smap1, smap2= F.normalize(smap1, p=2, dim=0), F.normalize(smap2, p=2, dim=0)
    gamma = 1.2
    smap1, smap2 = torch.pow(smap1, gamma), torch.pow(smap2, gamma)
    org_img1, org_img2 = org_img1.permute(1, 2, 0), org_img2.permute(1, 2, 0)

    fig1, axes1 = plt.subplots(ncols=2, figsize=(10, 5))
    fig2, axes2 = plt.subplots(ncols=2, figsize=(10, 5))
    axes1[0].imshow(org_img1)
    axes1[0].set_title(path1)
    axes1[1].imshow(smap1, cmap='hot')
    axes1[1].set_title('smap')
    axes2[0].imshow(org_img2)
    axes2[0].set_title(path1)
    axes2[1].imshow(smap2, cmap='hot')
    axes2[1].set_title('smap')

    plt.show()

for (images_1, images_2, targets, file_info) in test_dataloader:
    for y in range(images_1.shape[0]):
        current_img_1 = file_info[0][y-1]
        current_img_2 = file_info[1][y-1]
        saliency(current_img_1, current_img_2, targets[y])
```

Appendix C

Siamese Network Code

C.1 Main Siamese Network Class

Listing C.1 shows how the siamese network model was initialized in the program. First, pre-trained Resnet18 weights are initialized. These are further set inside the *SiameseNetwork* class, before modifying the fully-connected layer. This layer takes 1000 features as input and outputs a 512-dimensional feature representation.

The *forward()* function is used to pass input to the model. *forward()* calls *forward_once()* twice, one time for each input. When *forward_once()* receives an input, it passes it to the model and returns the computed embedding. When the two embeddings are computed, they are returned to the training/testing loop, which further calculates the similarity.

This Listing only showcases the SNN class definition. The full program implementation is available in the thesis' BitBucket repository [44].

Code listing C.1: Definition of the SNN class.

```python
import torchvision
import torch
from torchvision.models import resnet18, ResNet18_Weights
import torch.nn as nn

resnet = resnet18(weights=ResNet18_Weights.DEFAULT)

class SiameseNetwork(nn.Module):
    def __init__(self):
        super(SiameseNetwork, self).__init__()
        self.cnn = resnet
        self.fc = getattr(self.cnn, "fc")
        self.fc = nn.Linear(1000, 512)

    def forward_once(self, x):
        output = self.cnn(x)
        output = output.view(output.size()[0], -1)
        return output

    def forward(self, input1, input2):
        output1 = self.forward_once(input1)
        output2 = self.forward_once(input2)
        output1 = self.fc(output1)
        output2 = self.fc(output2)
        return output1, output2
```