university of groningen

faculty of science and engineering

# Siamese neural network for camera identification through sensor pattern noise

Bachelor's thesis

September 3, 2022

Student: C.T. van Herwijnen

Primary supervisor: Dr. George Azzopardi

Secondary supervisor: PhD. Student Guru Swaroop

# Abstract

For forensic investigations, gathering evidence is one of the most important jobs. Images can be traced back to a camera in the same way as bullets can be traced back to a gun. This can be done by looking at the unique fingerprint left by the sensor of the camera, which is called *sensor pattern noise*. We have designed a *siamese neural network* architecture that is able to compare a pair of images and that will determine the relationship between them, either being made with the same camera model or different cameras. We achieved an F-score of 74.59% on the natural subset of the publicly available *Dresden Image Database*. To the best of our knowledge, this is one of the first attempts at linking two images based on the pattern noise of a camera through the use of a siamese neural network.

# Contents

# List of Figures



# List of Tables



# Listings

# 1 Introduction

With the speed at which digital content is being created and spread, the chances that sensitive content (e.g. nude pictures, child sexual abuse material) is spread are higher than ever. This is problematic for Law Enforcement Agencies (LEAs) since they have to trace the content back to the perpetrator before they can build a case against the offender. Our aim for this project is to help the LEAs create more concrete evidence by finding relations between a pair of images that they could find on the darknet or on confiscated hardware. Finding relations between someone's darknet post and social media posts would help a great deal with fighting the crimes that go on on the darknet.

To find whether two images are related by being made using the same camera, we use the raw pixel data. This is the most robust way of detecting a relation between cameras because the metadata of images can easily be changed without a trace. When images are changed by either compression, resizing or filtering it is possible to identify the operations as shown by Kang and Wei [10], Kirchner and Fridrich [12], Wang and Zhang [20], this helps with the integrity of the relations made between a pair of images.

Lots of different methods have been researched in recent years to identify cameras both through images by Bayar and Stamm [2], Bennabhaktula et al. [3, 4] and frames taken from videos by Timmerman et al. [19]. These methods use the same underlying principle, that each camera has a unique fingerprint, also called sensor pattern noise [7, 14], made by the imperfections on the sensor during the manufacturing process of the camera. Our project has been built upon the previously done research, however, the big difference is that we compare a pair of images and find a relationship between the images, being it either from the same camera or two different camera models, instead of finding the camera identity of a single image.

With this project, we explore further the barely explored part of linking images together within the field of camera identification, which is a sub-field of the computer vision field within computer science. Due to the explorative nature of this project, we have kept the number of unknowns to a minimum. This means that we use unedited images and keep the neural network parameters as few as possible so that we can get quicker results and get a meaningful insight in the direction of camera identification with the help of siamese neural networks (SNNs). To guide the project we set a research question for ourselves to answer:

- What is the performance of a siamese neural network at detecting whether the same camera made a pair of images?

For the model to be feasible to deploy for LEAs, we would like it to be performing well enough. Our idea of performance is the right balance between time investment into training and an accurate enough tool so that the LEAs can use it to build evidence. It would not be feasible for LEAs to train a neural network over and over every time a new camera appears on the market. For LEAs it is of utmost importance to make an accurate tool, however, for this project, this would not be feasible due to the time investment needed to get such models. We have set ourselves the time constraint to train the neural network within 2 hours, this has the benefit that it can be easily retrained without much downtime.

Siamese neural networks are well suited for recognizing fingerprints since SNNs are capable of learning generic image features useful for making predictions about unknown class distributions even when very few examples from these new distributions are available

[13]. These two features are very useful for LEAs since most of the time there is not a lot of data available to train a network for a specific camera and the fact that an SNN can make predictions about unknown classes helps against the influx of new camera models all the time.

The paper is structured as follows: In Section 2 we will describe important concepts of the project, followed by the state-of-the-art in Section 3, subsequently, the main idea behind the project, and the pipeline used for the experiments is described in Section 4. The experiments and results are described in Section 5. Section 6 will go over the results and further research and the conclusion of the project can be found in Section 7.

# 2 Background

### Digital Forensics

Like normal forensics, digital forensics is the art of gathering, tracing and linking evidence, but in the digital world. Piva [17] provides a comprehensive insight into the challenges that come with image forensics, from explaining the physical workings of a camera and how the signal is being processed, to image editing and attempts at obfuscating the sensor pattern noise by performing antiforensics techniques. The amount of research on the topic and the diversity of techniques being explored show how important and urgent progress in the field is. Our project will explore a barely unexplored piece of the field and will contribute to the tools used by forensics.

### Model vs device level

The comparison between images can be made on two levels: model and device. For example, the model level comparison would be distinguishing between a Canon Ixus 70 and a Canon PowerShot A640. The device-based comparison would be to differentiate between a Canon Ixus 70 and another Canon Ixus 70. We choose to work on a model level since the devices within a certain model should have similar enough pattern noise to see them as one class and this would reduce the number of classes used for training the SNN

### Convolutional neural network (ConvNet)

A convolutional neural network has two different types of layers, convolutional layers and fully connected layers. Figure 2.1 shows the input and output of an example convolutional layer. The convolutional layer works as a sliding frame, called the kernel (orange in the figure), over the input array and multiplies the input with the kernel resulting in a smaller feature vector. The speed at which the kernel moves over the input is called the stride. The depth of a layer is equal to the amount of different kernel weight distributions it has. These weight distributions are set randomly at the start of training a neural network and changed by the backpropagation of the loss function, which happens after each iteration (epoch). The first fully connected layer will combine all the

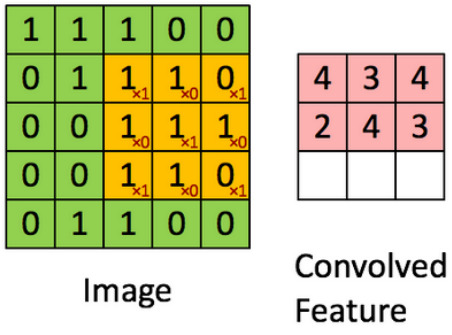



**Figure 2.1: Example input and output Convolution layer**

possible feature vectors into a single feature vector. And this single feature vector can be the input of another fully connected layer with the result of compressing the data to a more meaningful feature vector.

### Loss functions

Loss functions are methods, used in neural networks, that evaluate how well the predictions from the model come to the ground truth. There are many different loss functions[1], and each has its pros and cons when it comes to using it for a neural network. The loss function returns a value close to 0 when the ground truth and score from the model are close together, this means that the values in the kernel will not change to a great extent.

### Ground truth

The ground truth is a label given to the input data of a neural network so that the output of the network can be evaluated against this ground truth. In our case, the ground truth of an image pair would be 0 if the image pair is made by the same camera model and 1 if this is not the case.

### Performance metrics

Table 2.1 contains the performance metrics used in the project. These performance metrics are often used to compare the different models. Furthermore, we used histograms together with the mean and standard deviation to get better insights into our results. The scores outputted from the loss function, are used to plot a loss graph which is able to show the under- and overfitting of the model.

| Score | Comment | Equation |
|---|---|---|
| Accuracy | The fraction of correct instances in all instances | $\frac{TP+TN}{TP+FP+TN+FN}$ |
| Precision | The fraction of relevant instances among the retrieved instances | $\frac{TP}{TP+FP}$ |
| Recall | The fraction of relevant instances that were retrieved | $\frac{TP}{TP+FN}$ |
| F-score | The relation between precision and recall | $\frac{TP}{TP+\frac{1}{2}(FP+FN)}$ |
| | `TP = True Positive` `FN = False Negative`<br>`FP = False Positive` `TN = True Negative` | |

**Table 2.1: Performance metrics**
The different metrics used throughout the project

[1] `https://pytorch.org/docs/stable/nn.html#loss-functions`

# 3 Related work

## 3.1 Sensor pattern noise

Every camera has a sensor that captures the light to capture an image. The sensors are made from silicon and during the manufacturing process, slight imperfections occur on the sensor [7, 14]. As a result, some pixels will get corrupted. These corrupted pixels spread over the image combined are what we call sensor pattern noise and this pattern is unique for every camera, also called the fingerprint of a camera.

The pattern created by the imperfections is somewhat similar for each batch of sensors but still different enough to be able to identify the individual camera devices. This results in the fact that the same camera models will have some overlapping pattern noise, which we use to reduce the number of unknowns within our research. For a more in-depth explanation, we refer the reader to read the publication by Lukáš et al. [14].

## 3.2 Camera identification

In the field of forensics determining whether a camera was used to take a certain picture has been important already since before the 2000s. Geradts et al. [7] were asked by the court, whether it was possible to determine if an image had been made with a specific digital camera in 1999. Cameras have become more common in our day-to-day life with the development of smartphones and the number of pictures taken has skyrocketed. Additionally, it has become easier to maliciously alter pictures. With the methods proposed by Bayar and Stamm [2], Fanfani et al. [6], Lukáš et al. [14] it is possible to label altered images as such. All the more reason to develop the field of camera identification.

Many different methods that work with pattern noise have been developed, like being able to detect image forgeries Bayar and Stamm [2], Fanfani et al. [6], Lukáš et al. [14], camera identification through images Bennabhaktula et al. [3, 4] and capturing video frames [19]. We will use the papers about camera identification as our groundwork to build upon, however, we will predict a relation between a pair of images instead of linking a single image back to a camera identity.

Stamm et al. [18] have shown that is it possible to link patches of images together using a pre-trained ConvNet and training a separate similarity network. We build further upon this idea with the main difference that our feature extractor is trained on finding the difference between the images and not extracting the features separately.

## 3.3 Siamese neural network

The siamese neural network is often used for facial recognition, matching pieces of images [15], and even used for fingerprint identification [1]. The details of how we have adopted the concept of an SNN to be able to find a relation between a pair of images can be found in Section 4.2. One of the benefits of using an SNN is that it works well on small data sets, which might be the case during investigations done by LEAs when they try to find evidence from confiscated hardware. Furthermore, SNNs have to be trained only once, in comparison to the ConvNets [2, 3, 4, 19] used previously, which results in less time spent on setting up the system. This feature comes from the fact that SNNs are capable of learning generic image features useful for making predictions about unknown class distributions [13], while ConvNets are learning the fingerprint of every single camera

presented to them. This is why ConvNets are great at linking a single image back to a camera and thus good at checking the authenticity of the evidence.

# 4 Methodology

This section describes our approach to the project, where we use a siamese neural network to predict whether an image pair is made with the same camera. For this project, we have designed a pipeline, which can be seen in Figure 4.1, for training and testing the SNN. The pipeline is implemented in Python with the help of the library PyTorch [16]. The source code for the project can be found in our repository on GitHub[2].

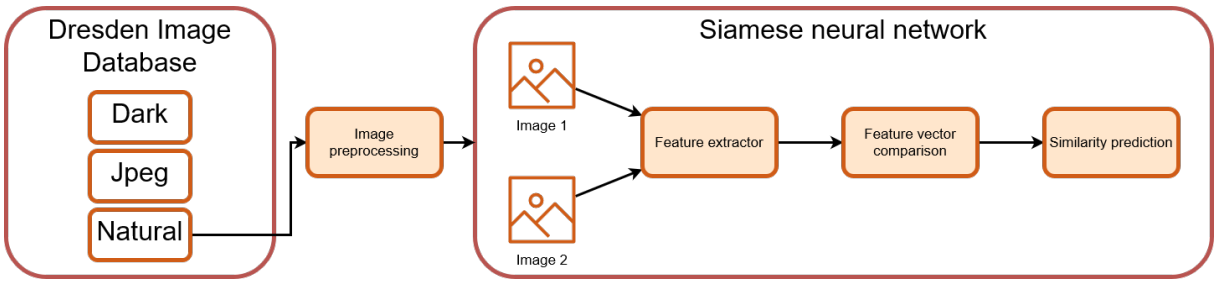



**Figure 4.1: General overview of the pipeline**

## 4.1 Image preprocessing

Before we can input the images from the data set into the network we have to preprocess them to uniform input size. Because, as Table 4.1 shows, the images of the data set are not uniform in width and height. The table shows the different amounts of pixel usage by Bennabhaktula et al. [4], Bayar and Stamm [2] and us respectively. We decided, due to GPU memory limitation, on an input size of 800x800 pixels in greyscale. This meant we had to drop the 640x480 pixels image out of the data set, since cropping only makes the image smaller.

Our reasoning behind using a larger input size than [2, 3, 4, 15], is that we believe that using more pixels results overall in more noise and thus a better fingerprint of the camera sensor. However, using only greyscale instead of the full RGB spectrum means that we lose some of the noise. To get all images uniform we use the transformation composite function from the PyTorch library. The composite consists of a centre crop followed by turning the image into a grey scale and finally storing it as a tensor, a multidimensional array, inside the main memory of the machine. Doing this before and not during the training/testing phase saves time during each epoch, so it becomes more efficient with more epochs ran.

## 4.2 Implemented network

The input of the network is a pair of images and the ground truth of the pair. These pairs of images and ground truth are prepared in batches by the data loader, which is a function inside the PyTorch library. The data loader takes a batch size as input and will split the given data set into equal batches. We have set up the data loader in a way to create a balanced data set, meaning it will randomly pick two images either from the same camera

[2] `https://github.com/Liulangzhe98/CID-SNN`

| (Width x height Pixels) | # img | 128x128 RGB [4] | 256x256 Grey [2] | 800x800 Grey |
|---|---|---|---|---|
| (640, 480) | 1 | 5.33% | 7.11% | Too small |
| (2560, 1920) | 1003 | 0.33% | 0.44% | 4.34% |
| (2592, 1944) | 585 | 0.33% | 0.43% | 4.23% |
| (2748, 3664) | 272 | 0.16% | 0.22% | 2.12% |
| (3008, 2000) | 676 | 0.27% | 0.36% | 3.55% |
| (3072, 2304) | 2234 | 0.23% | 0.31% | 3.01% |
| (3264, 2448) | 2175 | 0.21% | 0.27% | 2.67% |
| (3456, 2592) | 541 | 0.18% | 0.24% | 2.38% |
| (3648, 2736) | 5040 | 0.16% | 0.22% | 2.14% |
| (3664, 2748) | 2042 | 0.16% | 0.22% | 2.12% |
| (3872, 2592) | 673 | 0.16% | 0.22% | 2.13% |
| (4000, 3000) | 638 | 0.14% | 0.18% | 1.78% |
| (4032, 3024) | 462 | 0.13% | 0.18% | 1.75% |
| (4352, 3264) | 842 | 0.12% | 0.15% | 1.50% |
| Average pixel usage | | 0.19% | 0.26% | 2.52% |

**Table 4.1: Image distribution**
The distribution of image sizes within the data set.

model or from different ones. Furthermore, the data loader will take the preprocessed images from the main memory, instead of reloading and processing the images every single time it has to make a new batch.

Figure 4.2 shows the input and the output of the SNN. The input shown in the figure is a batch size of 1, resulting in only one image pair being shown to the SNN. Our SNN has been trained with a batch size of 15, in comparison to the 128 used by Timmerman et al. [19] and 512 used by Bennabhaktula et al. [4], which is rather small. The fact that we use pairs of images instead of single images already halves our batch size potential. Furthermore, using the 800x800 greyscale image crops instead of the 128x128 RGB crops uses up 13x the amount of memory. This is why we were forced to use a smaller batch size.

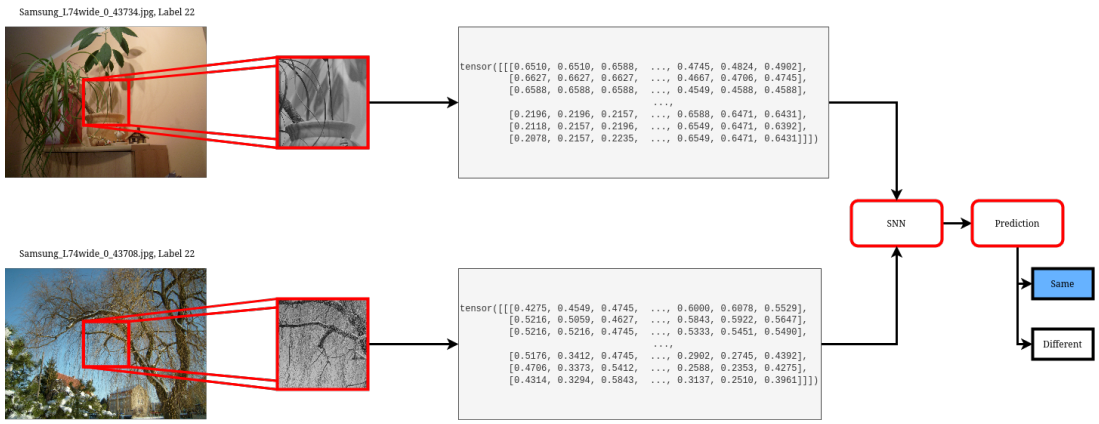



**Figure 4.2: Input and output of our SNN**

Following the pipeline shown in Figure 4.1, we have 3 steps within the SNN. These will be described in the following sections.

### 4.2.1 Feature extractor

The feature extractor is a simple ConvNet, however, due to the nature of a siamese neural network, we have a feature extractor that is used twice consecutively. This results in two feature vectors that can be compared in the next step of the pipeline. The shared weight property of an SNN makes sure that it does not matter in which order an image pair is presented, the two output vectors will not change due to the order.

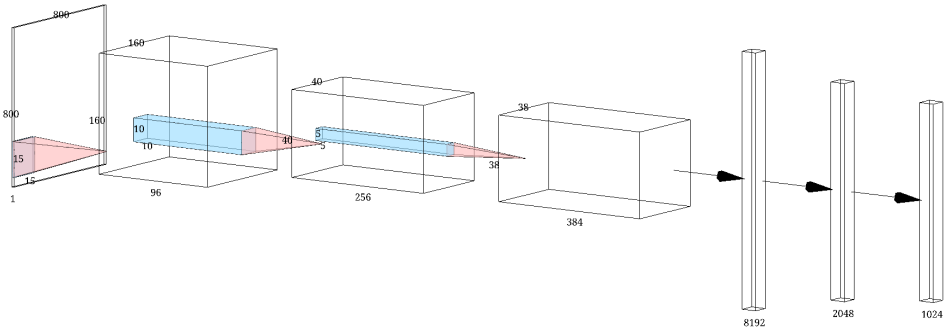



**Figure 4.3: Layering of feature extractor**
Between each layer we use the $ReLU$ activation function and a maxpool

The feature extractor consists of 3 convolutional layers and 3 fully connected layers, as shown in Figure 4.3. This is different from the previous research done by [2, 3, 4, 19] who used 4 convolutional layers. One of the reasons for using fewer layers is to reduce the number of parameters of our network, as this project was mainly focused on seeing if using an SNN was feasible at all. Furthermore, the previously mentioned studies were focusing on image-to-camera identification, while our study focuses on identifying a relationship between a pair of images. Due to these reasons, we had to change most of the architecture and use empirical experiments to create a baseline neural network.

All our kernels were made bigger than the previously done research because we were using larger images and using a small kernel size would lead to memory overflow. Using a bigger stride also has helped with memory constraints. The *tanh* activation function was substituted with a $ReLU$ activation function, this was done because our input is always positive and this means that $ReLU$ is more efficient. The output of the feature extractor is a tensor with 1024 values, we kept it on the larger side so that we did not lose too much information.

### 4.2.2 Feature comparator and prediction

The two feature vectors are pairwise compared using euclidean distance, this is one of the simplest ways to compare the vectors. We capped the euclidean distance to 1 so that the result would be between 0 (similar) and 1 (different). This similarity score is compared against the ground truth and the loss is calculated through a loss function, in our case the Mean Squared Error (MSE). MSE is chosen because it is one of the least complex loss functions. The MSE function will penalize the similarity scores that are far away from the actual values more heavily than the similarity scores close to the ground truths.

The part of forming the similarity score into a prediction about the connection of the image pair using Equation 4.1 is only used during the testing of our model. Because now we can use the similarity scores to calculate the mean and standard deviation of our model. And use statistics to tell the accuracy of a prediction.

We used $\epsilon$ as our threshold value so that the similarity score which was a range is changed into a binary label. The output is a bit unintuitive since 1 is normally seen as *True* and in our case it is *False*, meaning that the pair is made by two different camera models.

$$Pred = \begin{cases} 0 & \text{similarity score} < \epsilon \\ 1 & \text{otherwise} \end{cases} \tag{4.1}$$

## 4.3 Training and validation

The training of a neural network has two parts, the training itself and the validation part. We use the validation step to see if our neural network is not overfitting to the train data. Overfitting means that the model is getting more accurate on the training data set and getting less accurate on a data set not used for training. We would like our SNN to be as generic as possible so that it is compatible with as many cameras as possible. So we use the validation step to make sure the model is not overfitting. A simplified version of the training and validation algorithm can be found in Listing 1.

The SNN is trained over multiple epochs. We choose to train our SNN for a maximum of 50 epochs and applied early stopping. Early stopping means that the training can be terminated before the maximum amount of epochs is hit. This is done via a condition based on the output of the Mean Squared Error (MSE) loss function from the validation step.

Furthermore, for the optimization of the model's parameters, we have used the Adam optimizer [11], with a learning rate of 0.0002 and an exponential decay of 0.9 after every epoch. We had to use a very low learning rate, otherwise, the model would not converge at all. The exponential decay is used to converge the neural network along the epochs.

In each epoch, we train the neural network by presenting the neural network with all the image pairs in the training data set in a shuffled order. These pairs are presented in batches and after each batch is shown to the network, the weights of the kernels is being updated through the value of the loss function. Once the training for the epoch has finished, the neural network will start the validation step in a similar way to the training step, but it does not affect the weights of the kernels. The average loss scores of the training and validation step are gathered and once the training terminates, either through early stopping or hitting the maximum epochs, the scores are plotted in Figure 4.4.

## 4.4 Testing

The testing of our model is done separately from the training of our model so that we can test the neural network multiple times afterwards. The testing phase starts with dividing the test data set into separate batches and balancing the batches into a 50:50 ratio of image pairs made by similar and different camera models. The batches are then fed through the neural network and the similarity score is turned into a prediction about the relation of the image pairs with the help of Equation 4.1. The binary prediction is then compared to the ground truth of said image pairs and the results are stored so that we are able to calculate the precision, recall, and F-score of the model. Furthermore, the similarity scores are stored grouped by the camera model and the ground truth label so that we can plot a histogram of each of the camera models and a combined overview of


**Figure 4.4: Loss graph during training phase**
The average loss of each epoch is plotted and the mean of the averages of 5 epochs is used for early stopping

the whole neural network. A simplified version of the testing algorithm can be found in Listing 2.

# 5 Experiments and Results

We started off our experiments with lots of trial and error, due to the novelty of machine learning as a researched and explored domain. Section 5.2 describes two of the more significant improvements made during this trial and error period. During this phase, the code got changed a lot and the figures in this section only show the results of the experiments done after. Since we got decent and stable results only after this phase.

The experiments have been run on one of the nodes of the HPC[3] at Rijksuniversiteit Groningen. The node has the following hardware specifications:

- 6 cores @ 2.7 GHz (12 cores with hyperthreading)
- 128 GB memory
- 1 Nvidia V100 GPU accelerator card (32GB VRAM, 5120 cuda cores)

[3] `https://www.rug.nl/society-business/centre-for-information-technology/research/services/hpc/facilities/peregrine-hpc-cluster`

## 5.1 Data

### 5.1.1 Data set

The publicly available Dresden Image Database [8], will be mentioned as data set from now on, and will be used since it is specifically created for bench-marking digital image forensic tools. The data set is split up into three categories: *dark, JPEG* and *natural*. For our study, we have decided to use the natural subset, since it consists of 84 different scenes (e.g. trees, indoor, buildings). Resulting in the most complex data set and most realistic real-world scenario. This subset contains more than 17,000 images from 74 devices and 26 different models.

To perform the learning each image needs to have a label, which depends on the camera used to make the picture. The labelling is done on a model-level basis since each device from the same model (e.g Canon Ixus70_0, Canon Ixus70_1, and the Canon Ixus70_2) shares some of the pattern noise. This is done to have fewer different labels and hopefully increase the accuracy of the model. The labels and the number of images per device can be seen in Table A.1

### 5.1.2 Data augmentation

"Data augmentation in data analysis are techniques used to increase the amount of data by adding slightly modified copies of already existing data or newly created synthetic data from existing data. It acts as a regularizer and helps reduce overfitting when training a machine learning model."[4] To make up for the loss of information due to greyscale and cropping, we implemented data augmentation by randomly flipping the image pairs either horizontally or vertically[5]. This should help the model get more and different data and thus be able to generalize more with the existing data.

### 5.1.3 Data splitting

The qualified images are split with an 80:20 ratio between the training and testing set. The training set is further split into a 90:10 ratio, 10% of the training set is used for validating the model during the training process. This set of images does not influence the training of the parameters of the model, it is used for checking that the model is not under- or overfitting to the presented data.

## 5.2 Experiments

### 5.2.1 Balancing data set

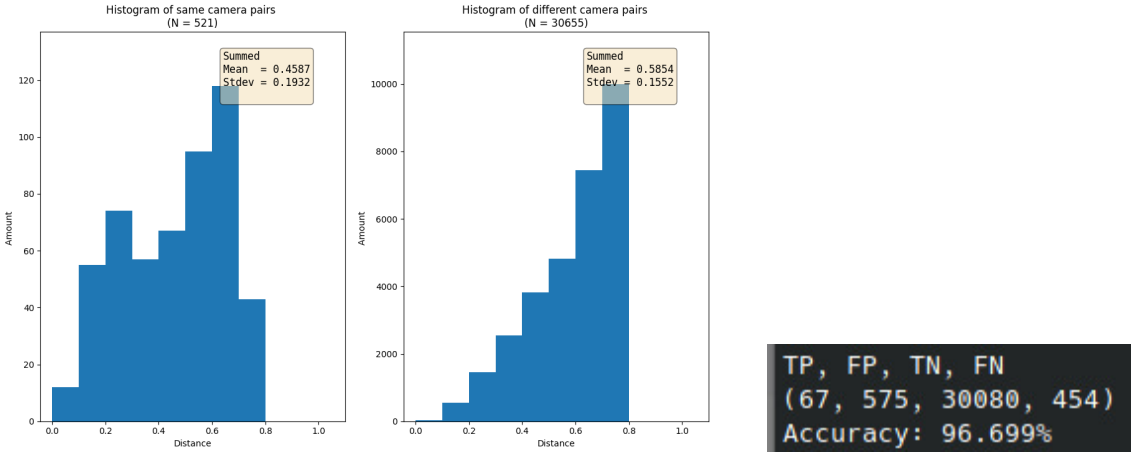
In this set of experiments, we started by comparing the results of different models to each other by looking at the mean and standard deviation of the similarity score grouped by label (similar and different) as well as through the histograms made from the similarity scores, like Figure 5.1a, where we would like to see a left-skewed distribution for the same camera pairs and for the different pairs a right-skewed distribution. We added the accuracy score as a performance metric as an extra guide to build upon. To get the accuracy scores we had to change the similarity score into a binary prediction this was done through the

[4] `https://en.wikipedia.org/wiki/Data_augmentation`

[5] `https://pytorch.org/vision/stable/transforms.html`, (RandomHorizontalFlip, RandomVerticalFlip)

use of a threshold value, seen in Equation 4.1, we had set this to an initial value of 0.2, based on our findings from the histograms. The histograms and accuracy scores looked very promising as shown in Figure 5.1. Sadly these scores came from the unbalance in our testing method.

To test our model, we compared 9 images each from a different camera model to the rest of the data set. This resulted in a 96.7 % accurate model, the histogram and scores can be seen in Figure 5.1. This high accuracy score came mainly from the fact that the true negatives took a big part of the results. After realizing this was not the correct method of testing our model we changed the testing phase completely to the algorithm described in Section 4.4.




**(a) Skewed histogram** **(b) Skewed scores**

**Figure 5.1: Skewness during early testing**
During the trial and error period we reached high accuracy due to bad testing.

### 5.2.2 Determining threshold value

During this set of experiments, we started using the F-score, precision and recall for better comparison between the different iterations. This was mainly done so that we would notice skewness easier than in the previous set of experiments. The threshold value, $\epsilon = \{x \in \mathbb{Q} : 0 \leq x \leq 1\}$, was determined after some early experiments resulted in bad accuracy when we used $\epsilon = 0.20$. A low threshold value would lead to a high precision score and a high threshold value would lead to a high recall score, so we decided to do the experiments with $\epsilon = 0.5$. However, after doing multiple experiments with this threshold value, we wondered if there is a better threshold value to use. So we decided to make a script that analyzes the similarity scores achieved by the model to see if a different threshold value would yield better results.

This script used the similarity scores and searched for the highest F-score, using the pseudo-code in Listing 3 going over all the possible values within the range of $\epsilon$. When the script shows that we can use a higher threshold value, it means that the model is trained better to make the distinction between the patterns of similar camera models and different ones.

The results of the script showed that using a threshold value between 0.59 and 0.67 would increase the F-score by an average of 1 percentage point. For our final experiments, we went with a threshold value of 0.61. The script is now integrated into the main part of

the code, so we can have a better indication of how our model performs with regards to the possible maximum F-score with the current architecture and (hyper)parameters.

## 5.3 Results

After the big impact experiments described in the previous sections, we started making smaller changes to the base model to further improve our results. After the creation of the model we test the model in the same script, and at least ten times more after this to ensure that the results are stable and not a lucky hit. We have created a boxplot, Figure 5.3, which shows the results of the different trained models. From the boxplot, we left out the experiments that failed completely and resulted in an F-score of below 5%. For each test run of each model, we got the F-score and the best achievable F-score via the script mentioned in Section 5.2.2.

At the start of these experiments, we changed the format of our histogram. We have added the threshold value (red divider line) and the green and red areas, so that is more clear which scores are TP, FN, FP, and TN respectively. This was done to further improve the ability to compare our models against each other.

Figure 5.2 shows the new format of the histogram. This figure was made with the results of the test that ran right after the creation of iteration 6. The top part clearly shows a left skew, which we would like to see and expected, while the histogram for different camera pairs does not show the right skew as hoped. But looking at the mean and standard deviation of the model, we can clearly see a difference between the same camera pair predictions and the pairs consisting of different camera models. And thus the difference the model makes between the two binary classes.

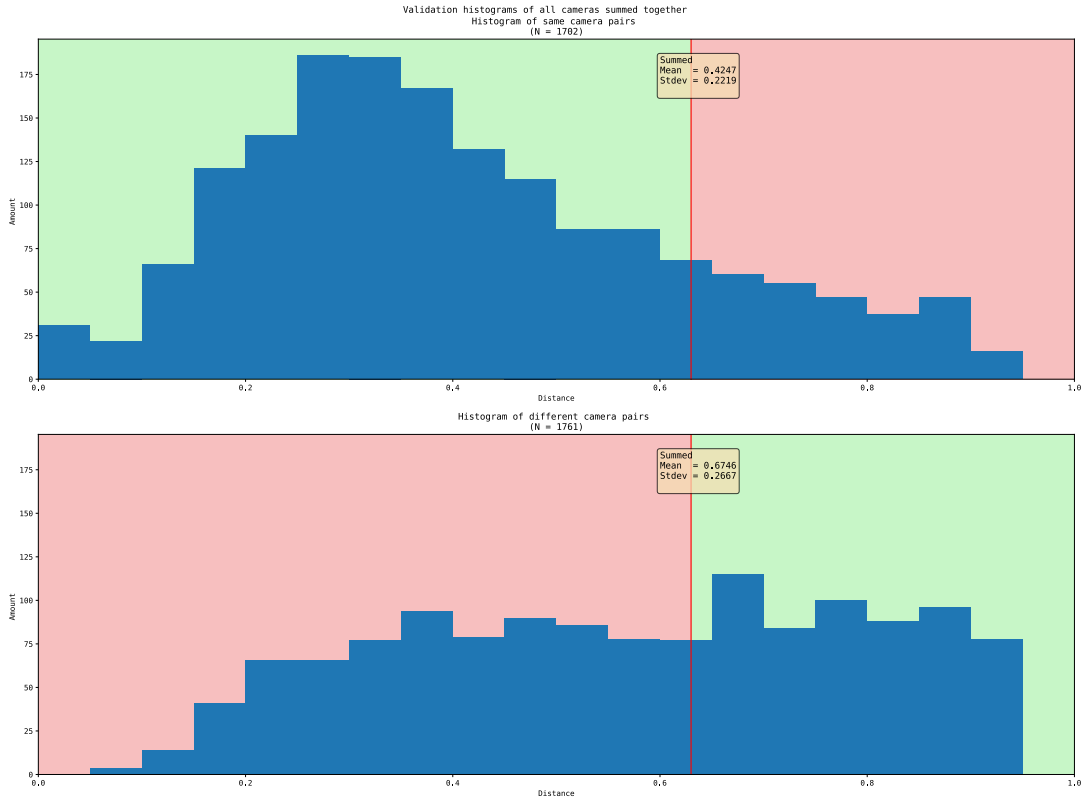



**Figure 5.2: Histogram of similarity scores**

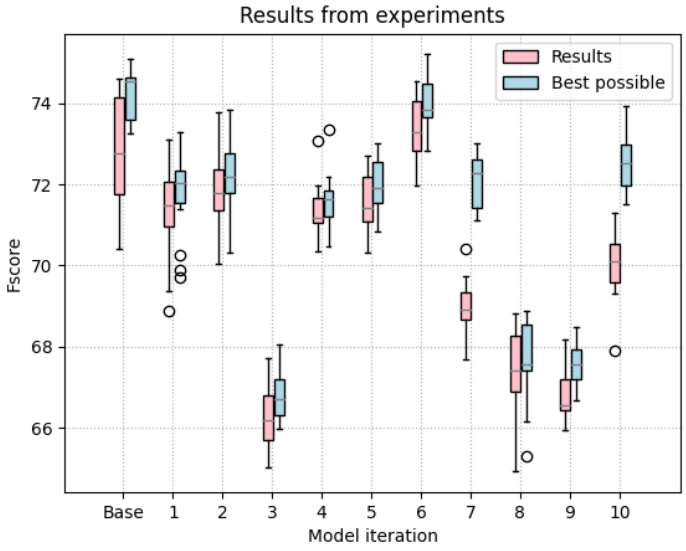
For iterations 1 and 2 after the base model, we decided to lower the number of epochs the neural network would be trained for because the loss graph of the base model showed that the training loss score was converging, but the validation loss score was not converging as fast.

After seeing that lowering the epoch did not help we implemented early stopping so that in the future the neural network would stop itself once it converged. Iteration 3 showed us that we needed to change the strictness of our early stopping condition. We used patience of 2, meaning that if the loss score of the validation step would increase in 2 consecutive epochs we would terminate early.

Iterations 4 and 5 had very similar results, however, iteration 4 converged a lot faster than iteration 5. Iteration 4 was stopped by the new condition after 29 epochs while iteration 5 was not stopped and ran for the full 50 epochs. These iterations showed us the randomness while working on neural networks and that changing one parameter can have different impacts which might not be seen right after one training run.

Up to iteration 6, we flipped our pairs of images randomly either horizontally or vertically[6] to add more data to the model. Literature suggested that data augmentation would increase the accuracy of our model [5]. So we expected the scores of iteration 6 to be lower because we turned off the flipping of images, however, our scores increased slightly almost equal to the scores of our base model. This model was trained with only 26 epochs instead of the 50 epochs used for the base model resulting in only half the training time used.

Lastly in iterations 7 to 10, we changed the early stopping condition to look at the mean loss over the last 5 epochs. This early stopping condition needs a lot more attention for it to be feasible to use with our model. Theoretically, this method would be the best at spotting when the loss graph is converging. Since it is calculating the steepness of the loss graph, however, we did not get the results that we wanted with this condition.




| Iteration | Comment |
|---|---|
| Base | Baseline model |
| 1, 2 | Changed the max epochs to 30 from 50 |
| 3 | Early stopping with patience of 2 |
| 4,5 | Early stopping patience set to 3 from 2 and max epoch to 50 from 30 |
| 6 | No longer randomly flipping horizontal or vertical |
| 7 | Changed early stopping to mean loss |
| 8 - 10 | Finetuning the early stopping threshold |

**Figure 5.3: Results of the different experiments**
The highest F-score is 74.5925%, achieved in the base model.

[6] `https://pytorch.org/vision/stable/transforms.html`, (RandomHorizontalFlip, RandomVerticalFlip)

# 6 Discussion

Since our project is exploring new territory within the field of camera identification, it is difficult to compare our results with the current state of the art. However, reaching an F-score close to 75% means that this method that we have implemented can be further explored. The biggest advantage over the state-of-the-art methods is that there is no need for retraining the network once it gets good results. As mentioned before we use a relatively small batch size compared to the state-of-the-art models, this could explain the difference in our loss graph (Figure 4.4) and the loss graph of Bennabhaktula et al. [4, see fig. 10]. The smaller batch sizes mean a less accurate estimate of the weight changes of the kernel, which means that the weights of the model will be moving past the ideal settings easier. This can be clearly seen in our loss graph by the vertical movement on the validation part of the graph even in the later epochs. Ideally, a loss graph should converge towards the end and not have a lot of vertical movement.

Due to the high degree of certainty needed for tools used by LEAs, we consider our model not yet good enough to be used right away. Nevertheless, the model can be used as a stepping stone towards making more accurate models, before doing the more computational heavy techniques, like comparing an image against a whole database.

As a consequence of the time constraints we set ourselves, we can say that the training of the model is more cost-effective than state-of-the-art models. On the flip side, our model does not outperform the state-of-the-art models, so finding the balance between time/hardware usage versus accuracy is an important task for the future. The biggest efficiency gain comes from the fact that no retraining is needed for unknown cameras since our model generalizes the pattern noise and finds the link between the images. Other models would have to be retrained to work with unknown cameras.

Our bottleneck with the project was the memory usage of the model. Because we used larger images and had a pair of images as input, we had to make decisions that would impact the accuracy of our model. This bottleneck affects the possibility to deploy the model to LEAs, since the model was trained on a High-Performance Cluster of the university. It would only be deployable to LEAs with similar HPCs.

A direction for future research is to see if using different image crop sizes or RGB images makes a difference in the accuracy of our implementation. Our expectations are that having a larger image crop size will have a positive factor on the accuracy of the SNN, however, this comes with the downside of using smaller batches and thus reducing the accuracy again. This sparks interest in finding the right balance between batch size and input size.

Another direction would be to see if the model is also working with images made by smartphones since the Dresden Image Database did not include any smartphone cameras. Furthermore, most sensitive content is made via smartphones and webcams and not with cameras used in the data set used. Some smartphones use postprocessing on the pictures taken and this can have a big influence on the ability to extract the pattern noise and be able to find a relationship between images.

And finally, we suggest looking into the similarity function, because we only used the simplest similarity function, euclidean distance, to keep the number of parameters we had to work with down. We expect that different similarity functions, especially the cosine similarity function, can have a positive influence on the accuracy of the siamese neural network. As said by Han et al. [9, see 2.4.7]: “Cosine similarity measures the similarity between two vectors of an inner product space. ... It is often used to measure document

similarity in text analysis.", the feature vector made by our SNN are the attributes that describe the images. This is very similar to the way documents are described for text analysis, so it would be interesting to see if this similarity would also work with our model.

# 7 Conclusion

The results that we got suggest that it is possible to train a siamese neural network to determine if two images come from the same camera. However, there is a lot of room for improvement and it is not suitable yet to deploy to LEAs. We managed to train our model sufficiently enough with the two-hour time constraint that we set ourselves, but this has to be done with the high-end graphics card due to the sheer amount of memory used for processing the images during training.

To answer our research question, 'What is the performance of a siamese neural network at detecting whether the same camera made a pair of images?', our best performing model got a 74.59% F-score, which is the best-reported score on the full 'natural' subset of the Dresden Image Database [8] with an image pair to identification model to our knowledge. This model was trained within 80 minutes resulting, in our opinion, in a well-performing model for the time invested to train it.

# 8 Acknowledgements

We thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Peregrine high performance computing cluster.

# References


[1] P. Baldi and Y. Chauvin. Neural networks for fingerprint recognition. *Neural Computation*, 5:402–418, 1993. doi:10.1162/neco.1993.5.3.402.

[2] B. Bayar and M. Stamm. Constrained convolutional neural networks: A new approach towards general purpose image manipulation detection. *IEEE Transactions on Information Forensics and Security*, 13:2691–2706, 2018. ISSN 1556-6013. doi:10.1109/TIFS.2018.2825953. URL https://doi.org/10.1109/TIFS.2018.2825953. 2691.

[3] G. S. Bennabhaktula, E. Alegre., D. Karastoyanova., and G. Azzopardi. Device-based image matching with similarity learning by convolutional neural networks that exploit the underlying camera sensor pattern noise. pages 578–584. SciTePress, 2020. ISBN 978-989-758-397-1. doi:10.5220/0009155505780584.

[4] G. S. Bennabhaktula, E. Alegre, D. Karastoyanova, and G. Azzopardi. Camera model identification based on forensic traces extracted from homogeneous patches. *Expert Systems with Applications*, 206:117769, 2022. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2022.117769. URL https://www.sciencedirect.com/science/article/pii/S0957417422010430.

[5] J. C., R. A., and 27th European Signal Processing Conference, EUSIPCO 2019 27 2019 09 02 - 2019 09 06. Data augmentation for drum transcription with convolutional neural networks. *European Signal Processing Conference*, 2019-September, 2019. ISSN 2219-5491.

[6] M. Fanfani, A. Piva, and C. Colombo. Prnu registration under scale and rotation transform based on convolutional neural networks. *Pattern Recognition*, 124:108413, 2022. ISSN 0031-3203. doi:https://doi.org/10.1016/j.patcog.2021.108413. URL https://www.sciencedirect.com/science/article/pii/S0031320321005896.

[7] Z. J. Geradts, J. Bijhold, M. Kieft, K. Kurosawa, K. Kuroki, and N. Saitoh. Methods for identification of images acquired with digital cameras. volume 4232, page 505 – 512. SPIE, 2001. doi:10.1117/12.417569. URL https://doi.org/10.1117/12.417569.

[8] T. Gloe and R. Böhme. The 'dresden image database' for benchmarking digital image forensics. pages 1584–1590. Association for Computing Machinery, 2010. ISBN 9781605586397. doi:10.1145/1774088.1774427. URL https://doi.org/10.1145/1774088.1774427.

[9] J. Han, M. Kamber, and J. Pei. 2 - getting to know your data. In J. Han, M. Kamber, and J. Pei, editors, *Data Mining (Third Edition)*, The Morgan Kaufmann Series in Data Management Systems, pages 39–82. Morgan Kaufmann, Boston, third edition edition, 2012. ISBN 978-0-12-381479-1. doi:https://doi.org/10.1016/B978-0-12-381479-1.00002-2. URL https://www.sciencedirect.com/science/article/pii/B9780123814791000022.

[10] X. Kang and S. Wei. Identifying tampered regions using singular value decomposition in digital image forensics. In *2008 International conference on computer science and software engineering*, volume 3, pages 926–930. IEEE, 2008.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization, 2014. URL https://arxiv.org/abs/1412.6980.

[12] M. Kirchner and J. Fridrich. On detection of median filtering in digital images. In *Media forensics and security II*, volume 7541, pages 371–382. SPIE, 2010.

[13] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. 2015.

[14] J. Lukáš, J. Fridrich, and M. Goljan. Detecting digital image forgeries using sensor pattern noise. volume 6072, page 362 – 372. SPIE, 2006. doi:10.1117/12.640109. URL https://doi.org/10.1117/12.640109.

[15] I. Melekhov, J. Kannala, E. Rahtu, and 23rd International Conference on Pattern Recognition ICPR 2016 23 2016 12 04 2016 12 08. Siamese network features for image matching. *Proceedings - International Conference on Pattern Recognition*, pages 378–383, 2016. ISSN 1051-4651. doi:10.1109/ICPR.2016.7899663. URL https://doi.org/10.1109/ICPR.2016.7899663. 378.

[16] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019. URL http://arxiv.org/abs/1912.01703.

[17] A. Piva. An overview on image forensics. *International Scholarly Research Notices*, 2013, 2013.

[18] M. Stamm, O. Mayer, and 2018 IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2018 2018 04 15 - 2018 04 20. Learned forensic source similarity for unknown camera models. *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, 2018-April:2012–2016, 2018. ISSN 1520-6149. doi:10.1109/ICASSP.2018.8462585. URL https://doi.org/10.1109/ICASSP.2018.8462585. 2012.

[19] D. Timmerman, G. S. Bennabhaktula, E. Alegre, and G. Azzopardi. Video camera identification from sensor pattern noise with a constrained convnet. *CoRR*, abs/2012.06277, 2020. URL https://arxiv.org/abs/2012.06277.

[20] Q. Wang and R. Zhang. Double jpeg compression forensics based on a convolutional neural network. *EURASIP Journal on Information Security*, 2016(1):1–12, 2016.

# A Appendix

## A.1 Tables

| ID | Device name | # img | ID | Device name | # img |
|---|---|---|---|---|---|
| 0 | Agfa_DC-504_0 | 166 | 15 | Olympus_mju_1050SW_3 | 192 |
| 1 | Agfa_DC-733s_0 | 528 | 15 | Olympus_mju_1050SW_4 | 187 |
| 2 | Agfa_DC-830i_0 | 695 | 16 | Panasonic_DMC-FZ50_0 | 265 |
| 3 | Agfa_Sensor505-x_0 | 167 | 16 | Panasonic_DMC-FZ50_1 | 415 |
| 4 | Agfa_Sensor530s_0 | 357 | 16 | Panasonic_DMC-FZ50_2 | 251 |
| 5 | Canon_Ixus55_0 | 418 | 17 | Pentax_OptioA40_0 | 169 |
| 6 | Canon_Ixus70_0 | 172 | 17 | Pentax_OptioA40_1 | 158 |
| 6 | Canon_Ixus70_1 | 179 | 17 | Pentax_OptioA40_2 | 156 |
| 6 | Canon_Ixus70_2 | 171 | 17 | Pentax_OptioA40_3 | 155 |
| 7 | Canon_PowerShotA640_0 | 188 | 18 | Pentax_OptioW60_0 | 192 |
| 8 | Casio_EX-Z150_0 | 166 | 19 | Praktica_DCZ5.9_0 | 194 |
| 8 | Casio_EX-Z150_1 | 174 | 19 | Praktica_DCZ5.9_1 | 185 |
| 8 | Casio_EX-Z150_2 | 172 | 19 | Praktica_DCZ5.9_2 | 190 |
| 8 | Casio_EX-Z150_3 | 172 | 19 | Praktica_DCZ5.9_3 | 189 |
| 8 | Casio_EX-Z150_4 | 166 | 19 | Praktica_DCZ5.9_4 | 184 |
| 9 | FujiFilm_FinePixJ50_0 | 210 | 20 | Ricoh_GX100_0 | 192 |
| 9 | FujiFilm_FinePixJ50_1 | 205 | 20 | Ricoh_GX100_1 | 164 |
| 9 | FujiFilm_FinePixJ50_2 | 215 | 20 | Ricoh_GX100_2 | 175 |
| 10 | Kodak_M1063_0 | 449 | 20 | Ricoh_GX100_3 | 165 |
| 10 | Kodak_M1063_1 | 443 | 20 | Ricoh_GX100_4 | 158 |
| 10 | Kodak_M1063_2 | 423 | 21 | Rollei_RCP-7325XS_0 | 183 |
| 10 | Kodak_M1063_3 | 445 | 21 | Rollei_RCP-7325XS_1 | 179 |
| 10 | Kodak_M1063_4 | 554 | 21 | Rollei_RCP-7325XS_2 | 182 |
| 11 | Nikon_CoolPixS710_0 | 171 | 22 | Samsung_L74wide_0 | 215 |
| 11 | Nikon_CoolPixS710_1 | 181 | 22 | Samsung_L74wide_1 | 209 |
| 11 | Nikon_CoolPixS710_2 | 168 | 22 | Samsung_L74wide_2 | 216 |
| 11 | Nikon_CoolPixS710_3 | 156 | 23 | Samsung_NV15_0 | 202 |
| 11 | Nikon_CoolPixS710_4 | 166 | 23 | Samsung_NV15_1 | 198 |
| 12 | Nikon_D200_0 | 335 | 23 | Samsung_NV15_2 | 199 |
| 12 | Nikon_D200_1 | 338 | 24 | Sony_DSC-H50_0 | 284 |
| 13 | Nikon_D70_0 | 165 | 24 | Sony_DSC-H50_1 | 257 |
| 13 | Nikon_D70_1 | 174 | 25 | Sony_DSC-T77_0 | 362 |
| 14 | Nikon_D70s_0 | 163 | 25 | Sony_DSC-T77_1 | 171 |
| 14 | Nikon_D70s_1 | 174 | 25 | Sony_DSC-T77_2 | 189 |
| 15 | Olympus_mju_1050SW_0 | 189 | 25 | Sony_DSC-T77_3 | 184 |
| 15 | Olympus_mju_1050SW_1 | 194 | 26 | Sony_DSC-W170_0 | 205 |
| 15 | Olympus_mju_1050SW_2 | 203 | 26 | Sony_DSC-W170_1 | 200 |

**Table A.1: Camera devices**
All the camera devices from the natural subset of the Dresden Image Database with their corresponding label and amount of images.

## A.2 Listings

```
def train_model(model, train_data_set, validation_data_set):
    # Setup optimizer and learning rate scheduler
    loss_history = []
    for epoch in range(MAX_EPOCH):
        train_loss = train_one_epoch(model, train_data_set)
        valid_loss = validation(model, validation_data_set)
        loss_history.append((train_loss, valid_loss))
        #Early termination check
    create_loss_plot(loss_history)


def train_one_epoch(model, data_set):
    total_loss = 0
    loss_func = MSELOSS() # From the pytorch library
    for image_pairs, ground_truths in data_set:
    # image_pairs & ground_truths are batches (arrays)
        similarity_score = model(image_pairs)
        loss = loss_func(similarity_score, ground_truths)
        # Back propagate this loss score
        total_loss += loss
    return total_loss / (number of batches)


def vaidation(model, data_set):
    total_loss = 0
    loss_func = MSELOSS() # From the pytorch library
    for image_pairs, ground_truths in data_set:
    # image_pairs & ground_truths are batches (arrays)
        similarity_score = model(image_pairs)
        loss = loss_func(similarity_score, ground_truths)
        total_loss += loss
    return total_loss / (number of batches)
```

**Listing 1: Simplified code for training the model**

```
def test_model(model, balanced_data_set):
    truth_list = []
    prediction_list = []
    similarity_scores = []
    threshold = 0.5
    for image_pair in balanced_data_set:
        gt = ground truth of image_pair
        truth_list.append(gt)
        similarity_score = model(image_pair)
        similarity_scores.append(similarity_score)
        if similarity_score <= threshold :
            prediction_list.append(0)
        else:
            prediction_list.append(1)

    calculate_scores(truth_list, prediction_list)
    create_histograms(similarity_scores)
```

**Listing 2: Simplified code for testing the model**

```python
# same and diff are the scores achieved in the testing phase
def find_threshold(same, diff):
    best_f = (0, 0) # F-score and threshold value
        for e in range(10, 100):
            # Score range between 0.1 and 1.0 to limit calculation time
            e /= 100
            tp = len([x for x in same if x < e])
            fn = len(same) - tp
            fp = len([x for x in diff if x < e])
            tn = len(diff) - fp

            prec = tp/(tp+fp)
            recall = tp/(tp+fn)
            fscore = 2*(recall*prec)/(recall+prec)

            if fscore > best_f[0]:
                best_f = (fscore, e)
    return best_f
```

**Listing 3: Code for finding the best threshold value**