Local Pixel Manipulation Detection with Deep Neural Networks

Detektion lokaler Pixelmanipulationen mit tiefen Neuronalen Netzen

Master-Thesis von Alexander Wölker

August 2019

TECHNISCHE UNIVERSITÄT DARMSTADT

Local Pixel Manipulation Detection with Deep Neural Networks
Detektion lokaler Pixelmanipulationen mit tiefen Neuronalen Netzen

Vorgelegte Master-Thesis von Alexander Wölker

1. Gutachten: Prof. Dr. Jan Peters
2. Gutachten: M. Sc. Daniel Tanneberg

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
In der abgegebenen Thesis stimmen die schriftliche und elektronische Fassung überein.

Darmstadt, den 19. August 2019

(Alexander Wölker)

Thesis Statement

I herewith formally declare that I have written the submitted thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.
In the submitted thesis the written copies and the electronic version are identical in content.

Darmstadt, August 19, 2019

(Alexander Wölker)

Abstract

In times of digital images and easy to achieve manipulations, tampered photos are ubiquitous. Well carried out manipulations are almost impossible to identify, even for humans. To earn back trust it is necessary to develop techniques that can detect such manipulations. Two of the most commonly used manipulations are splicing and copy-move, e.g., copy a part from one image and paste in into another or the same image. In this thesis, we present an approach based on deep learning that significantly improves the possibilities to detect such manipulations. The approach uses a combination of a VGG-Net for feature extraction and Global Average Pooling for classification as its architecture. Our approach not only classifies the images, for being manipulated or not, but also localizes where the manipulations occur. To achieve this localization, it uses patches combined with image classification. We evaluate our approach and conduct several experiments, in which we compare the rate of classification against other well-known techniques, such as SIFT (copy-move), Expectation Maximization with segmentation (splicing) and a retrained Faster-RCNN (both). Our approach achieves the highest F_1 score across all evaluated techniques, reaching about 77% macro average for our generated COCO segmentation data set for splicing and 79% for the Columbia data set, respectively.

Zusammenfassung

In Zeiten von digitalen Bildern und einfach zu erreichenden Bildmanipulationen sind gefälschte Fotos allgegenwärtig. Selbst für Menschen sind gut gemachte Manipulationen schwer zu identifizieren. Um das Vertrauen zurück zu gewinnen ist es notwendig Techniken zu entwickeln die diese Manipulationen detektieren können. Zwei der meist benutzten Manipulationen sind splicing und copy-move, z.B. kopieren eines Bildteils von einem Bild in ein anderes oder das selbige. In dieser Thesis präsentieren wir einen Ansatz basierend auf tiefem Lernen das die Wahrscheinlichkeit solche Manipulationen zu detektieren signifikant verbessert. Dieser Ansatz benutzt eine Kombination aus einem VGG-Netz um Features zu extrahieren und Global Average Pooling zur Klassifizierung als Architektur. Unser Ansatz klassifiziert nicht nur Bilder, ob diese manipuliert sind oder nicht, sondern lokalisiert auch wo diese auftreten. Um diese Lokalisierung zu erreichen, werden Bildteile in Kombination mit Klassifizierung benutzt. Wir evaluieren unseren Ansatz und führen mehrere Experimente durch, in denen wir die Klassifizierungsrate mit anderen bekannten Techniken vergleichen, wie z.B. SIFT (copy-move), Expectation Maximization mit Segmentierung (splicing) und einem um trainierten Faster-RCNN (beide). Unser Ansatz erreicht den höchsten F_1 Makrodurchschnitt über alle evaluierten Techniken, etwa 77% auf unserem generierten COCO Segmentierungs-Datensatz für splicing und ungefähr 79% für den Columbia Datensatz.

Acknowledgments

Firstly, i wanted to thank my thesis adviser Daniel Tanneberg of the Intelligent Autonomous Systems (IAS) group at TU Darmstadt. He supported me in this work by giving great feedback and ideas. I also would like to thank Prof. Dr. Jan Peters, head of IAS group, for making this thesis possible.
I am also glad about the support from the Company Klynveld Peat Marwick Goerdeler (KPMG) and my supervisors there, Dr. Sebastian Thieme and An Toung Le.

Contents

Figures and Tables

List of Figures

List of Tables

Abbreviations

List of Abbreviations

Notation	Description
ANMS	Adaptive Non-maximal Suppression
CFA	Color Filter Array
CGI	Computer Generated Images
CNN	Convolutional Neural Network
DCT	Discrete Cosine Transform
DER	Effective Detection Range
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
ELA	Error Level Analysis
EM	Expectation Maximization
Faster-RCNN	Faster-Regionbased Convoltional Neural Network
FMT	Fourier-Mellin Transform
GAP	Global Average Pooling
GCRF	Gaussian Conditional Random Field
GMM	Gaussian Mixture Model
GT	ground truth
k-d tree	k-dimensional tree
K-NN	K-Nearest Neighbors
LP	Linear Pattern
LRFfIMD	Learning Rich Features for Image Manipulation Detection
LSTM	Long Short-Term Memory Network
MFR	Median Filtering Residual
MLE	Maximum Likelihood Estimate

MRF	Markov Random Fields
MSE	Mean Squared Error
NN	Neural Network
PCT	Polar Cosine Transform
PFA	Purple Fringing Aberration
PRCG	Photo-Realistic Computer Generated Images
PRNU	Photo-Response Non-Uniformity Noise
PSD	Power Spectral Density
QPCET	Quaternion Polar Complex Exponential Transform
RANSAC	Random Sample Consensus
ReLU	Rectified Linear Unit
ResNet	Residual Network
ROI	Region of Interest
RPN	Region Proposal Network
SIFT	Scale Invariant Feature Transform
SMOTE	Synthetic Minority Over-sampling Technique
SNN	Siamese Neural Network
SRM	Spatial Rich Model
std	standard deviation
SURF	Speeded Up Robust Features
SVM	Support Vector Machine

1 Introduction

1.1 Motivation

In times of fake news where we can't trust images anymore, there is a need for detecting such image manipulations. There are multiple categories where and reasons why image manipulations are used. For example in politics, e.g., as propaganda to cover-up a miss launch of rockets, political subordination, influence elections through election posters or religion, e.g., no pictures of woman in public areas. Also in media respectively news sector, e.g., opinion making, covering the truth with fake image or war exaggeration to win more readers to name just some. An example is given in Figure 1.1. They are also used in satire and art in some kind of positive way.

Figure 1.1.: Left and middle show original taken photos. On the right is a spliced image published in the Los Angeles Times in 2003. Pictures are taken from [1].

Image manipulation is easy these days, there are no special skills needed anymore to manipulate a picture. To earn back trust it is necessary to detect if pictures are manipulated or not. To find these manipulations, we need blind image manipulation detection, which means we do not know if the image has been manipulated or in which way. In addition, the mechanism should not only identify if images are manipulated, it should also point to the location.

Also humans find it difficult to recognize the tampered regions, even with careful inspection of the image. A human mostly relies on what seems to not look authentic or appears to be logically wrong. Sharp edges are an easy indicator for humans, but could be challenging for machines to identify. The detection gets more difficult the better the manipulation has been integrated into the scene or background. The more elaborate the manipulation gets, the harder its detection becomes even for humans. In Figure 1.2 the left part of each image is easy to be recognized as manipulated for humans. On the right part, however, the manipulation is integrated well into the background, hence, it is hard to identify the tampered region even as a human.

Image manipulation can be described as any kind of manipulating pixels of an image which harms the authenticity of an image taken by a camera. Some of them are for example filters, distortions, Computer Generated Images (CGI) and local manipulations, e.g., copy-move, splicing, inpainting or removal. Two of those most commonly used manipulations are in the local pixel area. The first type, copy-move is also called cloning, where a contiguous portion of pixels is copied from one image and pasted at a different location within the same image. The second type splicing also called cut-paste is a technique which copies a region from an authentic image and pastes it into another image. We will further examine these two manipulation types and try to detect them. Further details and explanations of these two methods are presented in Subsection 1.2.1 and Subsection 1.2.2. The third one which we mention for completeness is called retouching or removal or in-painting, where pixels are copied altering to match the surrounding area to, e.g., remove distracting elements or prevent inelegant shapes.

Several methods on how to detect such manipulations have been investigated. One way would be active detection like digital signatures or watermarking. They get violated by manipulating these images. Another way is passive detection where manipulations can be found without preparing the images. We will use this passive method, without any prior knowledge or preconditions to get a working prediction method for image manipulation detection.

Figure 1.2.: For each image: On the left part a splicing attack which can be easily identified by a human. On the right part the artist[1] integrated these manipulations nearly perfectly into the background. These manipulations on the right part are really hard to identify as manipulated, even as a human.

We also localize where the image has been manipulated. The simplest approach to do so is slicing the image into identical sized sub parts (patches) and do image classification on each of it. This patch-based approach makes it also possible to have a comparable small model, because we are only interested in the local spatial differences of each patch to discriminate between authentic and tampered regions.

To evaluate how our model's performance, we compare it with two other models for each of the two problems. One of the other approaches is based on Neural Networks, while the other approaches use a different technique. Developing a methodology to compare the different approaches and their performance is explained in Section 4.4.

1.2 Related Work

There exists numerous manipulations which are global (over the whole image) or local (only on a part of the image). For example filters are mostly used globally, e.g., Brightness Adjustments, Colorization, Contrast Enhancement, etc. In CGI both are used, e.g., the complete image is rendered or only a part of it as often done in the movie industry. Otherwise, distortions used more widely locally like translation, rotation, scaling, flipping and cropping of objects to fit them to the scene. Several methods have been investigated to detect these manipulations. In this section we give an overview of detection approaches and techniques which have been used for widely-used local manipulations. It is the most commonly used type, where we have a closer look at already used detection methods. We have a look at the two main manipulations copy-move and splicing, which are further explained in the subsections.

[1] Image from artist maxasabin `https://www.deviantart.com/maxasabin/art/Sleepless-659653311` `https://www.deviantart.com/maxasabin/art/Night-Classic-Before-and-After-605953188` (visited on 05/15/19)

1.2.1 Copy-Move Attack

The most common manipulation is called copy-move attack or cloning. This attack is described as a contiguous portion of pixels(region or area) is copied from the image and pasted at a different location within the same image. An example for copy-move is given in Figure 4.3. We first have a look at some varieties of methods to detect these manipulations. Followed by a paragraph of the SIFT[2] descriptor which have been used from several papers as basis. We also have a look at Speeded Up Robust Features (SURF)[3] which is partly inspired from SIFT[2]. It is claimed by its authors to be more robust and several times faster. Each paragraph is in chronicle order in which they have been published.

In [4] they try to detect those manipulations by dividing an image into small overlapping blocks. First they compare the similarity of these blocks and finally identify possible duplicated regions. As features, they use the average of the red, green and blue components. And additional the block is divided into two equal parts in four directions(left|right, up|down, diagonal(left upper to right down, left down to right upper corner)) where the ratio of the sum of the first part to the sum of both parts is used as features too. These features are then used for matching the blocks and find similar ones to identify copy-move forgery. Another approach is presented in [5] where a Fourier-Mellin-Transform (which performs radial projection on the log-polar coordinate Fourier transformation of image blocks) is proposed to extract features along the radius direction of image blocks to find similar blocks. In [6] they extract Zernike moments from overlapping blocks of the image and use their magnitudes as feature representation. The detector employs locality sensitive hashing (LSH) for block matching and removes falsely matched block pairs by inspecting phase differences of corresponding Zernike moments to finally detect copy-move attacks.

Scale Invariant Feature Transform (SIFT)[2]

The **SIFT**[2] descriptor is state of the art for detecting feature points. These detected key points are very useful for detecting copy-move attacks combined with a matching algorithm. In [7] they use SIFT[2] with a generalization of the 2-nearest-neighbors (2NN) matching algorithm, which makes it possible to match multiple key-points. We use this paper as baseline for further comparisons in Chapter 4. In [8] they find the suspicious pairs of segmentation patches that may contain copy-move forgery regions by using SIFT[2] and then roughly estimate an affine transform matrix between the patches. Then this estimated matrix is refined with an EM-based algorithm by eliminating false patch pairs and to confirm the existence of copy-move forgery. In [9] they lower the contrast threshold of SIFT[2] and enlarge the input image to better find keypoints. They use this technique to generate a sufficient number of key-points that exist even in the small or smooth regions. Then they use hierarchical feature point matching via scale and overlapping gray level clustering for solving the key-point matching problems, that appear by changing the SIFT[2] parameters. Finally, a iterative homography estimation technique is suggested through exploiting the dominant orientation information of each key-point to identify copy-move keypoints.

Speeded Up Robust Features (SURF)[3]

The **SURF**[3] descriptor is based on the concept of SIFT[2] but uses an approximation based on an integral image and hessian matrix. In [10] they use a SURF[3] detector for copy-move detection, by extracted features along with k-dimensional tree (k-d tree) to identify the duplicated regions. SURF[3] is also used in [11] in combination with a new color space [12] for detection of a copy-move attack. This method also allows detecting points in the background, where SURF[3] normally can't detect anything. They use the points then to calculate delaunay triangles, where they use the inner circle of them to calculate features based on QPCET[13]. These features are then used for matching with Random Sample Consensus (RANSAC) to identify copy-move regions.

1.2.2 Splicing Attack

The second most used technique is called splicing or cut-paste. In this method a region is copied from an image and pasted it into another image. An example for splicing is given in Figure 4.4. We first have a look at some physics-based approaches, which need special conditions to work. Then some methods based on left image traces to identify spliced regions are shown. And finally we describe some Neural Network based approaches. Each paragraph is in chronicle order in which they have been published.

Physics based

In **physics based** approaches, it is assumed that different objects, of which, in a splicing attack one originates from another picture, show illumination inconsistencies. They assume a light source can be calculated for each object. If the light sources are different, they could lead to a digital tampering. For example in an outside area, there should just be one light source. In [14] the authors show that lighting inconsistencies can be a useful tool for revealing traces of digital tampering. To detect these inconsistencies, they borrow and extend tools from the field of computer vision. They estimate the direction of a point light source from only a single image. They extended their approach in [15] where they show how to approximate complex lighting environments with a low-dimensional model and further how to estimate the model's parameters from a single image. Inconsistencies in the lighting model are then used as evidence for tampering. The problem with these approaches is that they depend on the lighting, where the source is nor always calculable.

Image-Traces based

The **image-traces-based** approaches assume that artifacts introduced by various stages of the imaging process have left traces. Inconsistencies in these artifacts can then be used as evidence of tampering. In [16] the authors propose a method to detect the global addition of noise to a previously JPEG-compressed image by observing that the intrinsic fingerprint of a specific mapping will be altered. This method can be used to detect splicing attacks. Another approach is used in [17] where they divide the image into blocks to detect traces of rescaling and rotation in each block and estimating the parameters. They can effectively reveal the forged areas in an image that have been rescaled and/or rotated. They calculate an 1-D Discrete Fourier Transform (DFT) of each line of the edge map to obtain the frequency at which the rotation-induced peak occurs to identify the spliced area. In [18] they can detect splicing without any prior information. Local features are computed from the co-occurrence of image residuals and used to extract synthetic feature parameters. These features are learned from the image itself through the EM algorithm together with the segmentation in genuine and spliced parts. We use this paper as baseline for further comparison in Chapter 4.

Neural Network-based

The approaches are based on Neural Network (NN). Generally, Neural Network (NN)-based approaches are considered the current state of the art. In [19] a median filtering detection method based on Convolutional Neural Network (CNN) is used, which can automatically learn and obtain features directly from the image. The first layer of the CNN framework is a filter layer that accepts an image as its input, and outputs the images its Median Filtering Residual (MFR). Via alternating convolutional layers and pooling layers they learn hierarchical representations. They obtain multiple features for further classification and finally identify spliced areas. In [20] the authors try to improve extracted features from [18] by using an autoencoder, which also learns from only the image itself. Another approach is presented in [21] where they use a CNN and try to understand extracted features from each convolutional layer. They detect different types of image tampering through automatic feature learning. The proposed network involves five convolutional layers, two fully-connected layers and a softmax classifier. They also compare between softmax and Support Vector Machine (SVM) classification.

An also interesting idea is to use metadata for detecting splicing areas. In this approach they use typical photo EXIF metadata, that is automatically recorded by every digital camera. For every picture a lot of parameters are saved, for example focal length, camera type, white balance, etc. In [22] the authors use a Siamese Neural Network (SNN) of two random patches from different images. They try to learn if these patches are consistent within each other by estimating the probabilities that they share the same value of multiple metadata attributes to identify spliced areas. This approach is not general enough, because metadata is not always available.

1.2.3 Mixed Manipulation Detection's

In this subsection we will have a look at papers which are able to detect multiple manipulations in one approach. We first have a look at some methods based on left image traces to identify manipulated regions. After that detection's based on geometry inconsistencies are shown. Further dectections based on image formats are shown and finally state of the art Neural Networks based approaches. Each paragraph is in chronicle order in which they have been published.

Image-Traces based

The **image-traces-based** approaches assume that artifacts introduced by various stages of the imaging process have left traces. Inconsistencies in these artifacts can then be used as evidence of tampering. In [23] they assume that the lateral chromatic aberration(failure of a lens to focus all colors to the same point) is constant within each color channel. They use the green channel as reference, to estimate the aberration between green and red and green and blue channel to detect copy-moved parts or parts which have been moved around in the image.

In [24] they exploit image artifacts that are due to chromatic aberrations too, as indicators for evaluating image authenticity they use Purple Fringing Aberration (PFA) properties to locate the presumed image center. PFA direction map forms a "normal flow" map, which is usually sufficiently detailed to include data in both original and forged regions, if such exist. Analysis of the PFA direction map, as well as the calculated center, allows the detection of image regions that have been tampered since their acquisition by the camera. They can detect copy-move, splicing, cropping or objects moved in the image. In another approach it is assumed that each sensor has a typical pattern, for example of noise, which can be detected, if that noise differs in different areas.

In [25] they use Photo-Response Non-Uniformity Noise (PRNU) which can detect the absence of a camera in certain areas using sensor pattern noise. Markov Random Fields (MRF) is then used to model the spatial dependencies and then use Bayesian Estimation to take a decision on the whole image. They can detect copy-move, splicing and removal with that technique, but the camera needs to be known.

Geometry-based

In the **geometry-based** approach they make measurements of objects in the world and their positions relative to the camera. In [26], their analysis employs basic rules of reflective geometry and linear perspective projection. These rules make minimal assumptions about the scene geometry and only requires the user to identify corresponding points on an object and its reflection to detect splicing and removal.

Format-based

In the **format-based** approaches, they assume that the compression of the original image and manipulated part for example spliced regions will have different compression levels, when the image is saved. For example JPEG causes different compression artifacts every time when the image is saved, which can be detected by [27]. They use artifacts created by Color Filter Array (CFA) processing as in most digital cameras. They provide two different features. The first feature is extracted by identifying four Bayer CFA patterns of an image by re-interpolating them with these patterns. For each of the four Bayer pattern candidates the Maximum Likelihood Estimate (MLE) between input and re-interpolated image is calculated. It is expected that one out of the four MLEs should be significantly smaller than the others. They can detect copy-move and splicing with that method.

NN-based

The **NN-based** approaches are considered the current state of the art. In [28] the authors use a CNN to automatically learn hierarchical representations from the RGB images. The first layer weights are initialized with the basic high-pass filter set as used in calculation of residual maps in Spatial Rich Model (SRM)[29]. The two major steps are feature learning and extraction. First they pretrain a CNN model based on the labeled patch samples from the training images. The positives are elaborately drawn along the boundaries of the tampered regions in forged images, i.e., the boundaries of splicing and cloned patches. While the negative ones are randomly sampled from the authentic images. In this way, the CNN can concentrate on the local artifacts due to tampering operations and learn a hierarchical representation for the forged image. Then the pre-trained CNN is used to extract the patch based features for an image by applying a patch-sized sliding window to scan the whole image. The patch-based features are aggregated through feature fusion to obtain the discriminate feature for an image, which is then used to train the SVM for image forgery detection. They can detect splicing and copy-move with this method.

In [30] they use two methods, the first one is Radon transform of resampling features are computed on overlapping image patches. Then Deep learning classifiers and a Gaussian Conditional Random Field (GCRF) model are used to create a heatmap. Tampered regions are then located by using a Random Walk segmentation method. Second resampling features, which are computed on overlapping image patches, are passed through a Long Short-Term Memory Network (LSTM) based network for classification and localization. They extract a feature vector from each patch, where they apply a machine learning classifier to characterize any resampling. They are able to detect splicing, cloning and removal.

In [31] the authors use Linear Pattern (LP) of digital images as a global template whose integrity can be assessed in a localized manner. The consistency of the linear pattern estimated from the image noise residual is evaluated in overlapping blocks of pixels. The manipulated region is identified by the lack of similarity in terms of the correlation coefficient computed between the Power Spectral Density (PSD) of the LP in that region and the PSD averaged over the entire image. The method is potentially applicable to all images of sufficient resolution as long as the LP in the unmodified parts of the image has different spectral properties from that in the tampered area. They can detect spliced or inpainted areas.

In [32] they use a two-stream Faster-RCNN[33]. First is an RGB-stream to find tampering artifacts like strong contrast difference, unnatural boundaries with a Residual Network (ResNet) 101[34] which also proposes the probably tampered regions over a Region Proposal Network (RPN). Second is a noise stream(SRM[29] filter layer on RGB Image) that leverages the noise features extracted from a SRM filter layer to discover the noise inconsistency between authentic and tampered regions. They use the features from both streams through a bilinear pooling layer [35] to further incorporate spatial co-occurrence of these two modalities. They are able to detect splicing, copy-move and removal areas. We chose this paper for further comparison in Chapter 4.

1.3 Outlook

In Chapter 2 an overview of NN where they come from and how they lead to our approach is given. We go then into more details of methods used for our approach and why they are used in background information.

In Chapter 3 we show where parts of our approach come from and how they lead to our architecture. Further some details of our architecture and training of our model are presented.

In Chapter 4 the used data sets and how we generate them are described. Further the baseline approaches are explained and how we made them comparable to our model. Some experiments will show the potential of our approach in evaluation then.

In Chapter 5 we discuss our approach, variations, techniques and results.

In Chapter 6 our results are summarized and suggestions for future work are given.

2 Materials & Methods

In this section we firstly give a brief overview of some needed background information. We start with Neural Networks what they are, where they came from and how they lead to the used architecture. We have then a look a further details of the architecture and on the training methods used. Further we will talk about over-sampling and the measurement methodology.

2.1 Background Information

Neural Networks (NNs)[36] are based on a mathematical model inspired by neurons of a biological learning system (the brain). Where the dendrites are represented as weighted inputs and the axon by its output as shown in Figure 2.1. The neuron itself sums up the weighted inputs and "filters" the output over an activation function. It learns a mapping from the input to the output values. The simplest form is called perceptron[37] [38]. Many of the neurons in parallel are named a layer. If there are more than two layers connected to each other it is called **Deep Neural Network (DNN)**[39] [40], in its simplest form named multilayer perceptron[41]. In the Deep Learning[39] architecture there is at least an input, a hidden (usually there are multiple) and an output layer, in our case the last one represents classes. A small representation can be seen in Figure 2.2.

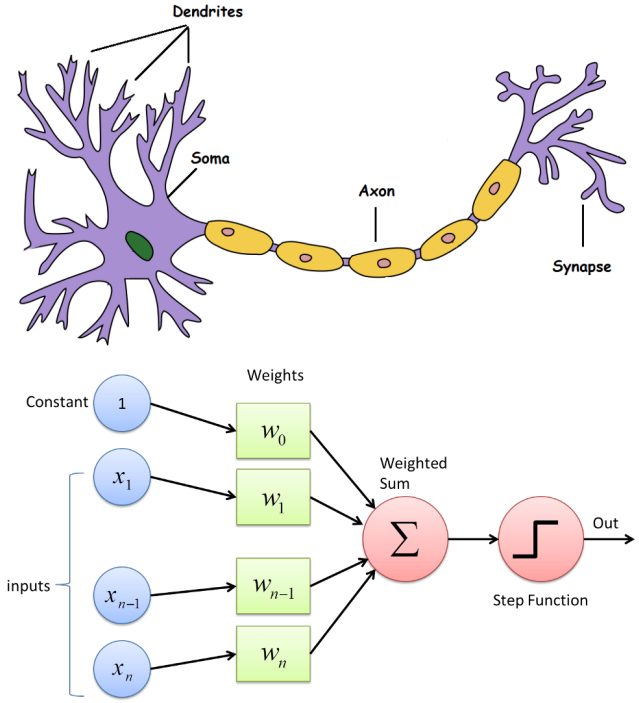

Figure 2.1.: A schematic representation of a biological neuron[1] in the brain and the mathematical model. From left to right: The Dentrites of the neuron represent the weighted inputs, the cell body(Soma) represents the sum of that weighted inputs, the Axon represents the step (activation) function and the Synapse represents the output.

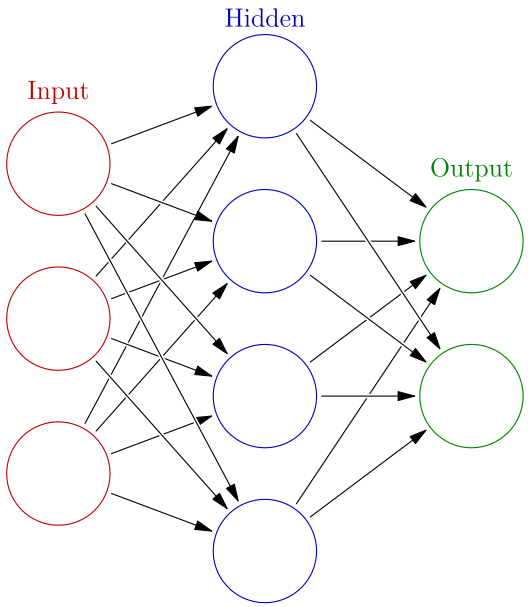

Figure 2.2.: A Multi Layer Perceptron[2] visualization. From left to right: Input layer, hidden layer and the output layer. If there is at least one hidden layer, we call it DNN. Every neuron is connected to all other neurons from the layer before.

[1] https://cdn-images-1.medium.com/max/1000/1*1Oh53dNdPITVnoOVGCCUFA.png (visited on 06/07/19), https://cdn-images-1.medium.com/max/1000/1*n6sJ4yZQzwKL9wnF5wnVNg.png (visited on 06/07/19)

[2] By Glosser.ca - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=24913461 (visited on 05/01/19)

The NNs[36] learns for example by forwarding the input until it reaches the output, this method is called feed forward network [42]. The neurons are learning on the changing input. If we do not simple forward the input, but also ask how far away we are from the wanted output(loss function) and update our weights in a way to minimize the distance, this method is called back propagation[43]. A learning rate and an optimizer are then used to do so, e.g., gradient decent also known as modified version called Adam[44].

If we also consider spatial information it is called **Convolutional Neural Network (CNN)**[45], which uses filters to extract the most useful information from the images. They are inspired by the visual cortex of animals. A typical architecture is shown in Figure 2.3. These filters are learned over back propagation[43] as described above, over the convolved input with a given kernel size, e.g., 3×3. A small convolved example is shown in Figure 2.5. Depending on the kernel size, we use padding, which is needed to preserve the size of the image, for example zero padding. This zero padding is done by adding zeros to the border at the size of the kernels surrounding area, e.g., 3×3 kernel leads to size of one pixel zero padding.

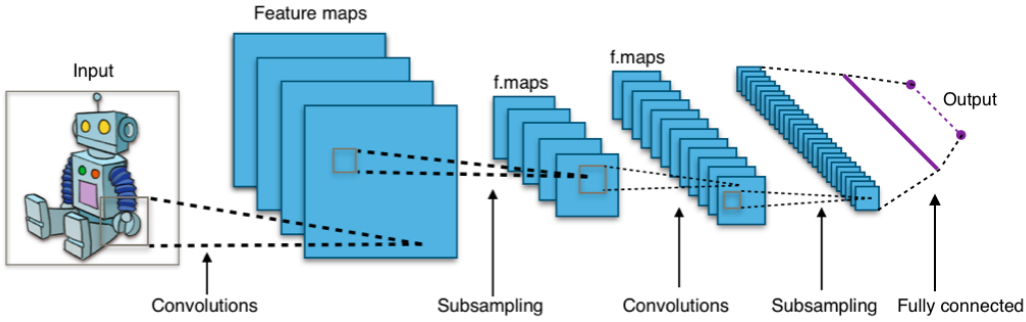

Figure 2.3.: This figure shows a typical CNN[45][3] architecture. From left to right: Input image, feature maps extracted from the input image over a kernel(filter) convoluted (see Figure 2.5) over all three channels which represents a layer. Then these feature maps are pooled(Subsampling), e.g., Max Pooling as shown in Figure 2.4. This is done multiple times, but the inputs are then convoluted or pooled from the previous layers feature maps instead of the channels. At the end a fully connected layer (every neuron in one layer is connected to every neuron in another layer) is used for classification, which finally maps to the number of classes.

Architecture and Learning-details

To further improve the learning process we use the following methods. The problem is, if weights change in early layers the inputs of later layers vary wildly. One technique to stabilize the learning process is called **Batch Normalization**[46] which is used to make the layer inputs more similar in distribution. In particular the neuron output is restricted to the area around zero. This method will lead to faster training and more accurate results. A batch in our case is called a bunch of image-patches which are fed to the network at once before it gets updated. The formulation is like the following as shown in [46]:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i, \qquad \sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2, \qquad \hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}, \qquad y_i = \gamma \hat{x}_i + \beta \tag{2.1}$$

Where x_i is the activation input of one image and m the number of images per mini batch (if we don't take the whole training data at once it is called mini batch of a batch size). μ_B is the mini-batch mean, σ_B the mini batch variance and ϵ a small number to prevent division by zero for the normalized activation input $\hat{x}_i$. This normalized input is then linear transformed by two trainable parameters γ to scale and β to shift to its new output y_i.

We further use **Max Pooling**[47] (which calculates the max value of a kernel of a certain size and only preserves the values which have the most impact) after each convolutional block to extract more useful information. A small example is shown in Figure 2.4. We additionally use a technique called **Dropout**[48] after each **Max Pooling**[47] to avoid overfitting of the training data and generalize better. To achieve this generalization effect, input units are randomly deactivated (set to zero) at a certain fraction rate.

[3] By Aphex34 - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=45679374` (visited on 05/01/19)

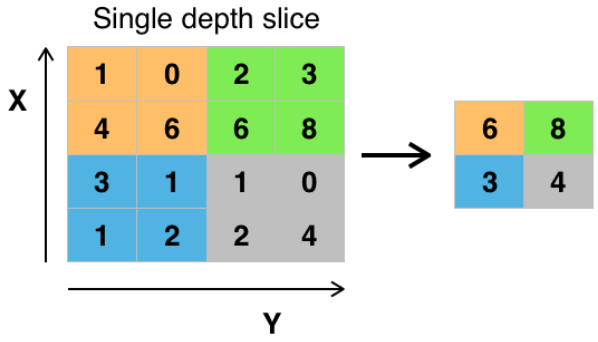

Figure 2.4.: This example[4] shows Max Pooling for a 4×4 matrix of one channel with a 2×2 kernel and a stride of two(kernels next position is two pixels forward). Each color shows one kernel in the matrix where the max value is taken from.

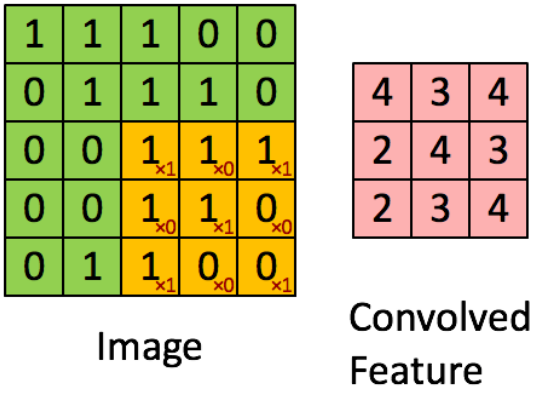

Figure 2.5.: This example[5] shows a 5×5 matrix of one channel (green) with a 3×3 kernel (orange) (the small numbers in the right bottom corner show the value which it is multiplied with and then summed up over the whole kernel) and a stride of one(kernels next position is one pixel forward).

For classification at the end of the network **Global Average Pooling (GAP)**[49] is used, which is like a normal Average Pooling layer (same as Max Pooling but takes the average instead of maximum) with the kernel size equals to the size of the input. We get the average of each filter, left over in 1 pixel, e.g., $h(height) \times w(width) \times d(dimensions$, in our case number of classes) leads to $1 \times 1 \times d$. A small example is shown in Figure 2.6. In [49] they have shown that this approach is an effective way, which also minimizes overfitting, instead of using fully connected layers for classification tasks.

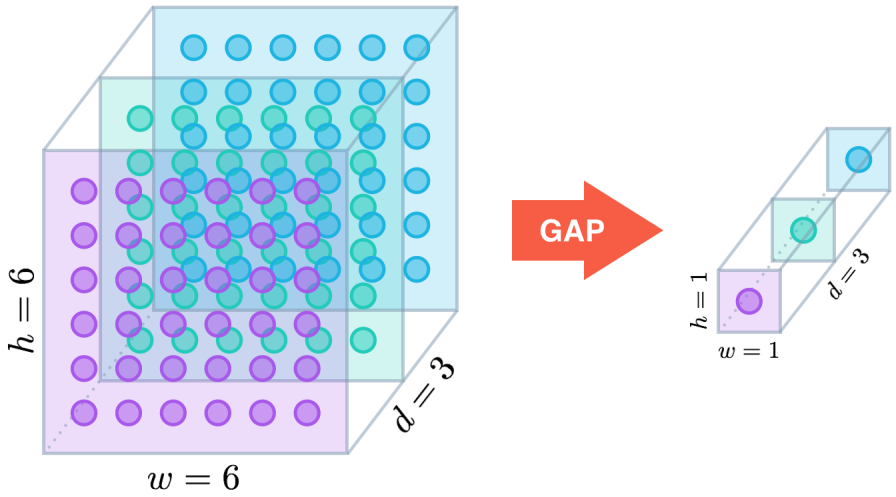

Figure 2.6.: This visualization[6] shows a 6×6 matrix for each dimension (in our case a class). Where w is the width, h the height and d the dimension(classes). On each dimension the average over all pixels is taken, which leaves one pixel(value) per class.

[4] By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581 (visited on 05/15/19)
[5] https://ujwlkarn.files.wordpress.com/2016/07/convolution_schematic.gif?w=268&h=196&zoom=2 (visited on 05/15/19)
[6] https://alexisbcook.github.io/assets/global_average_pooling.png (visited on 05/15/19)

As activation function of the CNN layers we use **Rectified Linear Unit (ReLU)**[50]. The purpose of this function is to establish non-linearity to a system that has just been computing linear operations. Defined as:

$$y = max(0, x) \tag{2.2}$$

Where x is the summed input over the weights and y the new output of the neuron where all negative numbers are set to zero. To get probabilities for the classification output layer we use **Softmax**[51] activation function which calculates probabilities between $0-1$ that sum up to 1. It is applied to the output scores s from the GAP[49] layer. Each element represents a class, so they can be seen as class probabilities. Defined as in [51]:

$$S(\eta)_c = \frac{e^{\eta_c}}{\sum_{c'=1}^{C} e^{\eta_{c'}}} \tag{2.3}$$

Where η and $\eta_{c'}$ are the scores for each class in C (number of classes) and e is the exponential function. In other words each class is normalized over all classes to calculate probabilities. As loss function **Categorical Cross Entropy** is used, which is a Softmax[51] where Cross Entropy loss[51] is applied after:

$$CE = -\sum_{i}^{C} g_i log(S(\eta)_c) \tag{2.4}$$

Where g_i is the ground truth (true label) and $S(\eta)_c$ the score for each class i in C which we get from the Softmax[51] of our network. In case of one hot encoding, which we are using, it is defined as:

$$CE = -log\left(\frac{e^{\eta_p}}{\sum_{c'=1}^{C} e^{\eta_{c'}}}\right) \tag{2.5}$$

so only the positive class C_p keeps its term in the loss (means in our case ([01] or [10]) positive is the 1). There is only one element of the ground truth vector g which is not zero $g_i = g_p(1)$.

Over-sampling
We use **patches** which are slices of an image extracted like a sliding kernel over the image. We use only non overlapping patches which size is elected in a way that they perfectly fit into the image. For example if the image is 224×224 the patch size could be 56×56, 28×28, etc. Because of these patches we have an imbalance in the labeling of them, we need a technique which is used to over-sample the minority class count to the same as the majority class count. This is necessary, because the loss function does not consider the data distribution, in the worst case the minority class could be treated as outliers of the majority class. We would classify only the majority class in this case. Sometimes the NN can learn on the imbalanced data, which is not the case here. The simplest way to do so is called RandomOverSampling[52], where randomly selected minority class patches are copied and pasted until the count is the same as the majority class. An already better approach is called Synthetic Minority Over-sampling Technique (SMOTE)[53] which generates new samples instead of just coping existing ones. These samples are generated in the following way, if we consider a sample x_i (patch $56 \times 56 \times 3$, width, height, color channels) then a new sample x_{new} is generated by its K-Nearest Neighbors (K-NN). One of these nearest-neighbors x_{si} is randomly selected to generate the new sample as follows:

$$x_{new} = x_i + \lambda(x_{si} - x_i) \tag{2.6}$$

where λ is a random probability. This interpolation will create a sample on the line between x_i and x_{si}. A small example can be seen in Figure 2.7.

This method does not use well-defined decision borders, because of that we use a special variation of SMOTE[53] called **Borderline-SMOTE**[54] which abstracts the border between the classes better. The difference of the three methods can be seen in Figure 2.8. In the borderline version 1, which we use, each sample will be classified for all nearest-neighbors: As noise (from different class) or as in danger (at least half are from the same class) or safe (all are from the same class). Samples in danger will be used to generate new samples which then belong to the same class[7]. We have a deeper look at the over-sampling problem for our case under Chapter 3.

[7] Basic explanation and equations from `https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html#mathematical-formulation` (visited on 05/01/19)

Measurement
We use $\boldsymbol{F_1}$ **score** as measurement which is defined as harmonic mean of precision(p) and recall(r):

$$p = \frac{t_p}{t_p + f_p}, \qquad r = \frac{t_p}{t_p + f_n}, \qquad F_1 = \frac{2pr}{p + r} \tag{2.7}$$

Where t_p are the true positives, f_p false positives and f_n the false negatives. We calculate the macro average the following:

$$p_{MA} = (p_{cl0} + p_{cl1})/2, \qquad r_{MA} = (r_{cl0} + r_{cl1})/2, \qquad F_{1_{MA}} = \frac{2p_{MA}r_{MA}}{p_{MA} + r_{MA}} \tag{2.8}$$

Where $cl0$ is the authentic class 0 and $cl1$ the tampered class 1 [8]. Why we need this measurement method is further explained in Chapter 3.

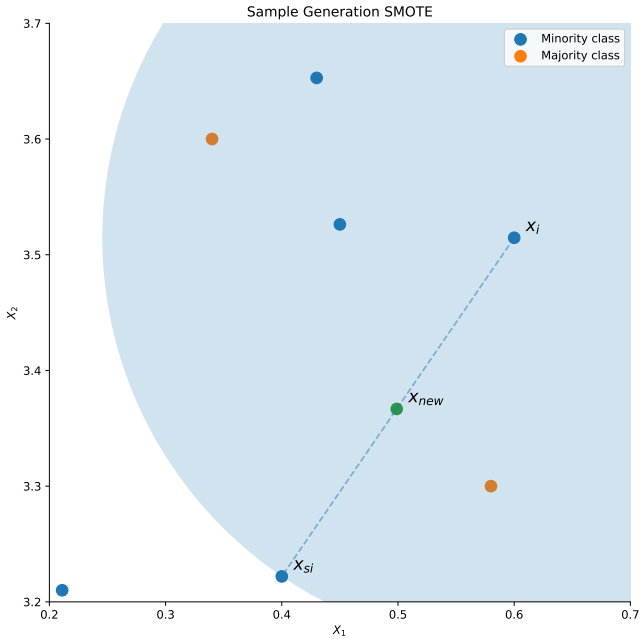

Figure 2.7.: This two-dimensional random number example shows how a new sample x_{new} is generated considering the three nearest-neighbors of sample x_i. One of these neighbors is then selected x_{si}. The line between these points represents the new possible samples depending on the probability λ [9].

[8] `https://en.wikipedia.org/wiki/F1_score` (visited on 06/07/19)
[9] Basic code and explanation from `https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html#mathematical-formulation` (visited on 06/01/19)

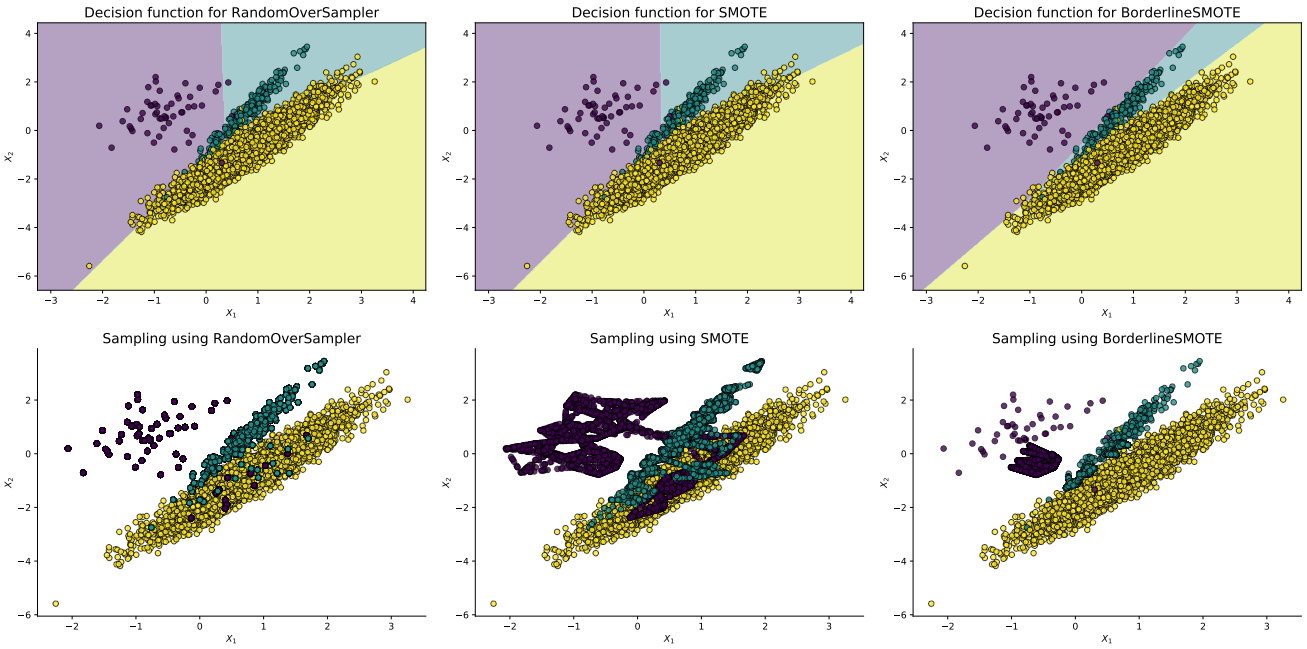

Figure 2.8.: This plot shows the difference between the three different over-sampling techniques from left to right: RandomOverSampler[52], SMOTE[53] and Borderline-SMOTE[54] version 1. Randomly two-dimensional data using different ratios is sampled for the plots. The first row shows the decision function for each and the second row how the data looks like after over-sampling. We use three different classes to show the effect of the decision functions and their over-sampling results. The RandomOverSampler[52] repeats some samples and balances the minority classes to the count of majority class. The Borderline-SMOTE[54] over-samples points which are in the border between two classes, while SMOTE[53] will not make any distinction.[10]

[10] Basic code and explanation from `https://imbalanced-learn.readthedocs.io/en/stable/over_sampling.html` (visited on 06/01/19)

3 Our Patch-based Model

In this chapter we describe the structure of our patch-based model. We use a CNN[45] based approach, which is state of the art for image classification. One of our main goals is to be efficient in training, which leads to a few training steps and a small model (fewer neurons to train, as many neurons as needed to represent the data well, but as few as possible to still generalize well).

We detect the most common used pixel manipulations which are locally detectable. So the model should be able to detect tampered areas without special conditions like image format, lighting or a special domain, e.g., faces. It should also be possible to detect on the image without additional information, for example metadata or user input, e.g., marking the area where a manipulation could be. These conditions mean we do blind image classification/detection without prior knowledge.

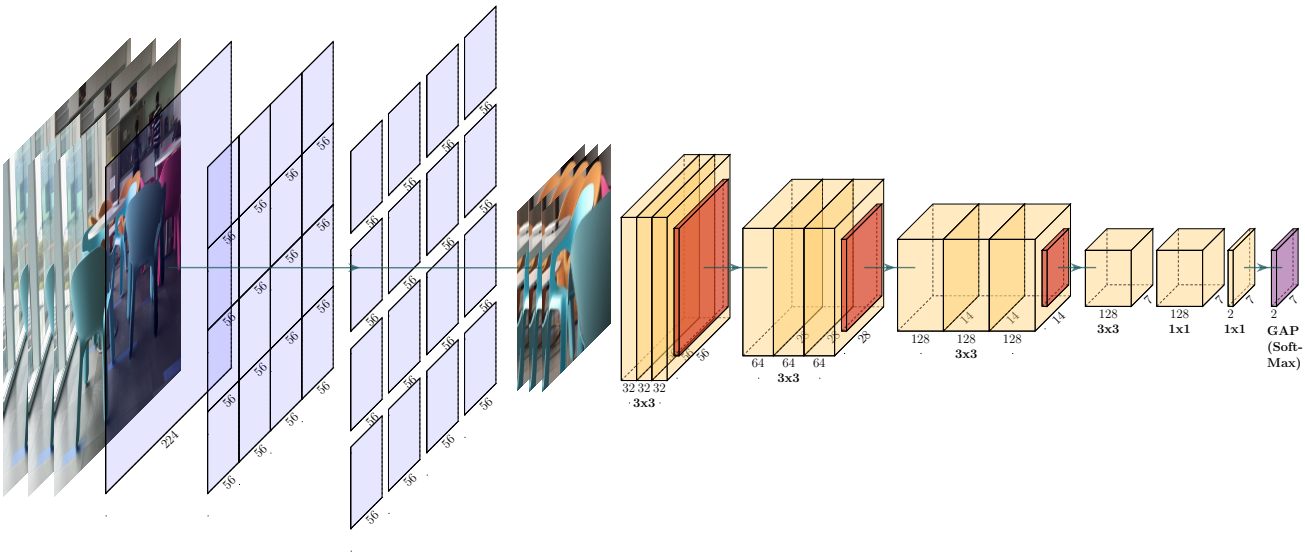

Figure 3.1.: This illustration shows our patch-based CNN model architecture[1]. The blue planes visualize the process from a 224×224 RGB image input, that is then sliced into patches of 56×56, which are used to train the network. The orange boxes represent convolutional layers with a 3×3 kernel. The red boxes after each convolutional block are Max-Pooling[47] layers with a 2×2 kernel combined with Dropout[48] after them. The final purple box is then a GAP[49] layer combined with SoftMax[55] activation for classification. The x-axis under each box represents the number of filters, while the z-axis represents the current quadratic image size. In bold under each block the used kernel-size is written.

[1] Image from COVERAGE[56] Dataset. Made with Tool: `https://github.com/HarisIqbal88/PlotNeuralNet/blob/master/README.md` (visited on 05/01/19)

3.1 Architecture of the Model

The model is oriented on the VGG-Net[57] in its general structure. We also use 224×224 resized images (bi-linear interpolation) as input before patch extraction. The last 3 blocks of VGG-Net[57] 16 configuration D before classification are used with a reduced filter number, to learn better on the small patches. With too many filters each filter does not learn enough or even doesn't get activated at all. If we use too less it is not possible to learn all the variations the data set provides. The number of filters used in each CNN[45] is also doubled after each Max-Pooling[47] layer as done in the VGG-Net[57]. The patch-size is also a stage of a certain layer after Max-Pooling[47] in the VGG-Net[57], e.g., 56×56 is derived from 224×224 Max-Pooling[47] 112×112 Max-Pooling[47] leads to 56×56. Before classification, we have an 7×7 as in the VGG-Net[57]. The final four layers for classification are taken from the paper [58], we also reduced the filter numbers here as done before. They use a GAP[49] layer which has been shown by [49] as an effective way, instead of using fully connected layers for the classification task. An illustration of our patch-based CNN[45] can be seen in Figure 3.1.

The model itself has three convolutional blocks with each three layers followed by a Max-Pooling[47] layer, after each Max-Pooling[47] layer we use Dropout[48] to achieve a better generalization. Each CNN[45] layer uses Batch-Normalization[46] before the non-linear activation function ReLU[50] as recommend in [34]. We also use the same kernel-size and stride as in [57]. In the last layer before GAP[49] we don't use any Batch-Normalization[46] or non-linear activation, since the output has to go through GAP[49] first. And finally we use a Softmax[55] activation for classification on the final GAP[49] layer as in [58].

3.2 Training of the Model

From the whole data which consists of randomly chosen alternated mixed (authentic, tampered) image pairs(to ensure that the original and the manipulated version are in one set), we take 10% for testing and another 10% from the 90% left, as validation. For example $80,000$ images -10% testing $8,000$ leads to $72,000$ images, where another 10% are taken as validation $7,200$. From the left $64,800$, after filtering, we take only the tampered for training $32,400$ (left if no pairs have been filtered). We then filter the data to first get rid of all image pairs where not even one patch has been marked as tampered in the labeling process over the threshold (the filtering is done, because the image pairs bring more imbalance to the data which has a bad impact on learning the minority class), which is shown in Table 3.1. Then only the tampered images are taken into count for the training from the leftovers, where the images are then sliced into patches 56×56 and normalized by 255 before feeding them batch wise to the network.

The ground truth labeling of the patches for training is done by building the difference between the tampered and the original image. This difference is sliced then into patches 56×56, where each color channel is checked against a lower 0.05% and an upper $0.08(1 - 0.08 = 0.92)\%$ threshold of changed pixels per patch to mark the patch as tampered. At least one channel has to fulfill the boundaries to be marked as tampered.

$$\%lower\_th <= \%(\#R-pixels \,||\, \#G-pixels \,||\, \#B-pixels) <= \%(1-upper\_th) \tag{3.1}$$

Where $\#R-pixels$, $\#G-pixels$ and $\#B-pixels$ are the percentage of the red, green and blue pixels of the difference to all possible pixels of one channel in that patch. The smaller the bound of $lower\_th$ the better it is possible to detect smaller objects. A too small bound leads to worse results, because the NN can not distinguish between the small area and the background anymore. This is mostly the same for the $upper\_th$ as bigger it gets ($1-upper\_th$, means smaller $upper\_th$) the better it can detect bigger objects. If the threshold is too big the border is too small to distinguish between the background and the tampered area anymore.

Method	Precision tamp.	Recall tamp.	F_1 score tamp.	F_1 score both	Ratio	#Epochs	#Training Images
All (A, T)	0.56±0.08	0.50±0.04	0.53±0.06	0.75±0.03	0.038±0	21.67±10.5	64,800±0
Tampered only	0.42±0.05	0.52±0.04	0.46±0.01	0.72±0.01	0.075±0	9.33±4.03	32,400±0
Filtered Out	0.50±0.06	0.57±0.08	0.53±0.03	0.75±0.01	0.117±0	15.66±4.11	20,888.33±18.52

Table 3.1.: Testing F_1 score macro average for copy-move segmentation, which leads to the decision made for filtering. The shortcuts represent A (Class Authentic) and T (Class Tampered). We use the mean±standard deviation (std) in macro average over three runs(same start training data $64,800$, same validation $7,200$ / test data $8,000$). As ground truth we used at least one channel fulfills lower bound 0.05% and upper bound $0.08(1-0.08=0.92)\%$ of pixels which are manipulated in one patch. Over-sampling method RandomOversampling with a bunch of 4992 (the biggest bunch which still fits into memory, but still fulfills bunch size / batch size $=0$) images is used. Number of epochs are computed with early stopping and restore best model weights. The ratio is calculated the following #patches (tampered/all). Filtering the data leads to the highest recall for class tampered and still have mostly the F_1 score as using all images for training, which also reduces needed training steps and epochs.

Because of the patches and the labeling, which also depends on the data set, are very imbalanced, we use over-sampling to compensate that. This imbalance leads to learning only the majority class and mostly forget about minority class, because the most examples the NN[36] have seen are the majority class. For example, one tampered image has just one marked patch, with an 224×224 image and patch-size 56×56 leads to 16 patches with one tampered. To further reduce imbalance we get rid of the authentic images too, as already mentioned.

To finally fit to the count of the majority class, we use over-sampling on the minority class. The first naive approach would be RandomOverSampling[52], which just copies the minority class patches and their labels randomly. This helps to keep more attention on the minority class, while training. We use Borderline-SMOTE[54] which is a variation of SMOTE[53] to further improve the over-sampling quality and generalization. In a bunch of 32 filtered-out tampered images, they get sliced into patches, find their labels and then get over-sampled. This is the smallest possible image bunch size to still use standard parameter of Borderline-SMOTE[54]:

$$32(bunchsize) \times 16(patches) = 512,\ 512 \bmod 256(batchsize) = 0 \tag{3.2}$$

To calculate the number of steps we need for each epoch while training, where an epoch is defined as all data has been fed to the neural network and a step is one fed batch, we used the number of counted authentic patches multiplied by two. This number is the same after over-sampling, because the tampered patches have then the same count as the authentic patches. The final number of steps per epoch is then computed by dividing this number through the batch size. The number of validation steps is calculated by the number of validation images multiplied by the number of patches divided through the batch size.

We use Early Stop[59] if the validation F_1 score stops improving for five epochs and then restore the best weights for our model. For measurement, we take F_1 score to be able to measure both classes fairly, if we take accuracy which is standard, we would only measure the authentic class. We take categorical crossentropy[51] as loss-function with one-hot encoded labels to measure the distance from the predicted output to the wanted output. We use Adam[44] which is an on gradient decent based approach as optimizer with standard learning rate.

4 Experiments & Results

The following chapter describes the used data sets and how they are generated. We will also have a look at some baseline models and the NN paper used for comparison. Further we have done some experiments to show the potential of our approach. After that we evaluate on this data sets and compare their results. For the NN-based approaches we used for example $80,000$ start images -10% for testing $8,000$ which leads to $72,000$, where another 10% are used for validation $7,200$. From the left $64,800$ we take only the tampered for training $32,400$ (they get filtered in our approach). The baseline models have only been tested once on the test sets, because there is no need for training.

4.1 Data sets used for Generating, Training and Testing

The following data sets are used for training and testing. We use **COCO Image Database**[60] 2014 with segmentation annotations, which is used to generate data from it as done in [32]. The following three data sets have been used for testing only: The first **Columbia Splicing Data set**[61] focuses on uncompressed spliced images from different cameras. Sizes range from 757×568 to 1152×768 TIFF or BMP formats created from authentic ones using copying and pasting visually salient objects in Adobe PhotoShop. The second **COVERAGE Copy-Move Data set**[56] consists of forged images and their originals with similar but genuine objects. A forged image was synthesized via graphical manipulation of the original using Adobe Photoshop CS4. And finally the **Realistic Tampering Dataset**[62] which is a mixed data set of realistic forgeries like copy-move, splicing or removal, created by hand in modern photo-editing software (GIMP and Affinity Photo). An overview of their numbers can be seen in Table 4.1.

Figure 4.1.: Non contiguous pixel differences can happen in the data sets[1]. These are only visible, because we resized the image to 224×224 and take the difference of an unsigned integer. This difference means a range$(0, 255)$, so there are no negative numbers, for example $160 - 161 = -1$ leads to 255. These mostly happens in the green channel, where we get at one pixel a difference of 1. This has no effect on our approach as long they are under the threshold of 0.05% of one channel. The difference is only needed to mark the ground truth labels for each patch or bounding box. This problem appears in other data sets too, for example Columbia[61], where noise in all channels can appear. We need multiple cleaning stages to mark a good ground truth as described in Section 4.4.

[1] From COCO Image Database[60]

4.1.1 Data set Generators for Training

We used modified versions of the data set generator, which has been introduced in [32] using the COCO Image database[60] 2014 with segmentation annotations, because here are no big enough data sets which focuses on these problems. In the data sets differences (tampered - authentic) image, one and two pixel noise differences can appear, which are not caused by the manipulation itself. An example can be seen in Figure 4.1 for furthers details. It is necessary to know this problem, to understand why we chose the ground truth as we did for evaluation.

Data sets based on Segmentation

We generate images for the splicing attack detection using the generator from [32] which is nearly original, but we don't use a gaussian filter for some randomly chosen tampered images as they do. The gaussian filter is not used, because we build the image difference between tampered and authentic images for labeling the ground truth patches. This difference would not be clean, with a used gaussian filter the whole tampered image would be different from the authentic, not only the manipulated part. We also added a condition that the randomly chosen source and target image are not allowed to be the same.

For the copy-move attack we use a modified version of the splicing generator, we stay in the same image for the manipulation and use a random translation of the copied area. At least 100% have to be visible of the randomly chosen segmentation, for bigger selections only 50%. An object counts as big if it takes at least 20% of the image. There is also a need for a minimal distance between the source and target area, which has been calculated using the center of the bounding box distance to a corner twice for small objects, so they will not overlap. For bigger objects a smaller distance makes sense, so half overlapping of the objects is allowed, so we take the distance just once. Some examples are given in the first row of Figure 4.2.

Data sets based on Rectangles

To simulate an easier splicing attack, we select an area of a rectangle range(5, 25)% of the image width/height which is then randomly scaled between(-25, +25)%, rotated (-90, 90)°, translated and pasted into another image. The hard edges of the rectangle are easier to learn for the neural network than the complex edges from the segmentation. The whole rectangle needs to be visible at the new position.

Same for the copy-move attack, but we stay in the same image and the translation has to be far away enough from the copy area, so they don't overlap or for bigger ones partly. This is done in the same way as for segmentation but the corners of the rectangle are used for calculations instead of the bounding box. The distance minimum is then the sum of the old center to corner distance and the new one (center to corner distance), because of rotation and scaling this distance changes. Some examples are given in the second row of Figure 4.2.

Dataset	#Tampered	#Authentic
COCO Original[60]	0	82,783
Generated Segmentation Splicing	40,000	40,000
Generated Segmentation Copy-Move	40,000	40,000
Generated Rectangle Splicing	40,000	40,000
Generated Rectangle Copy-Move	40,000	40,000
Columbia[61]	180	180
COVERAGE[56]	100	100
Realistic Tampering Dataset[62]	220	220

Table 4.1.: Data sets Overview. The numbers show how many images are available for each data set for the authentic and tampered case.

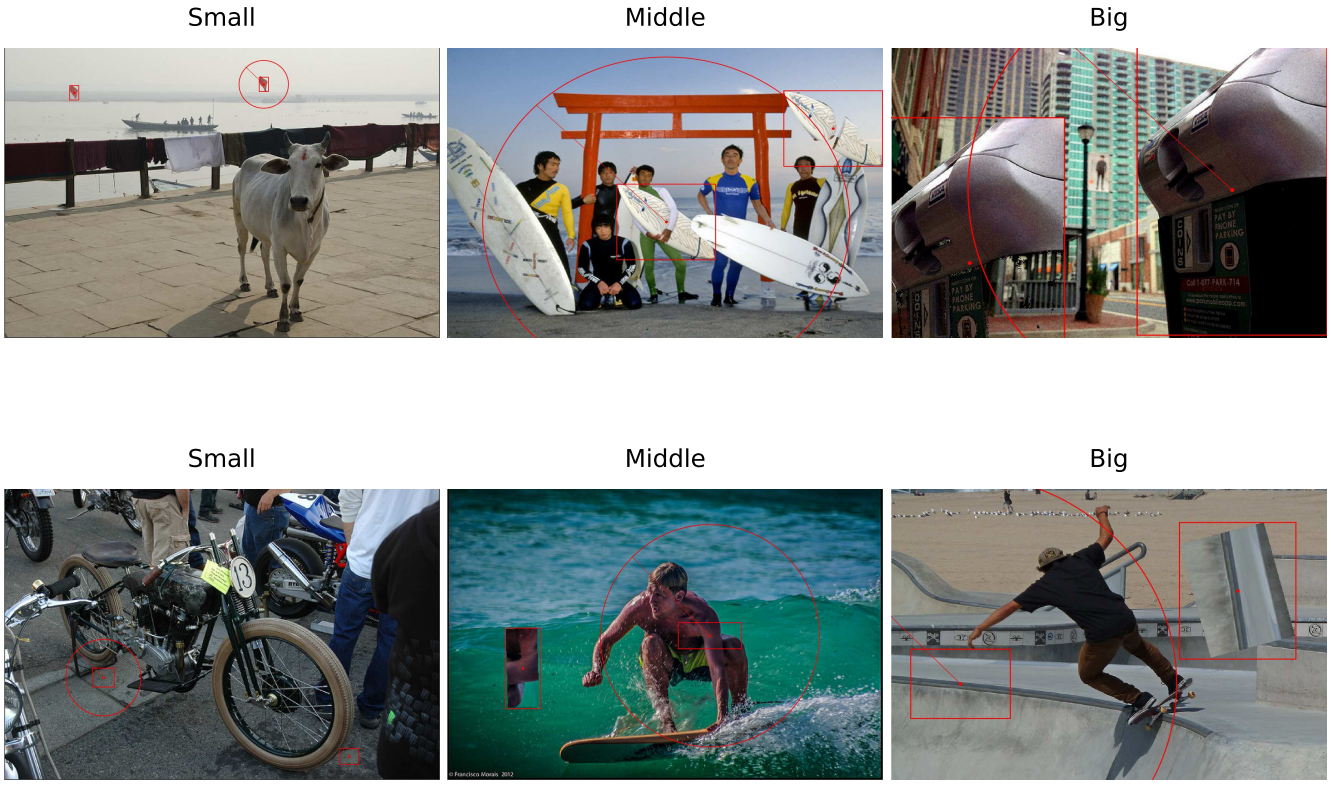

Figure 4.2.: Some examples how copy-move attacks are generated[2]. First row shows segmentation and second row rectangle examples. From left to right the area increases. The selected and pasted areas are marked with a red rectangle around them. The center of these rectangles are marked with a red point. A circle around the source indicates the minimum distance from source center point to destination center point. The radius is calculated by subtracting distance from X and Y of the source center point, because of this calculation we also get a bit of a buffer.

4.2 Baseline Models for Comparison

To compare the model with some state of the art methods we have chosen the first based on [63], which is pretty much standard for image point matching. The second choice seems to be one of the best which is not using a NN[64] based approach.

4.2.1 SIFT-Matching (Copy-Move)

In this approach [7] they use SIFT[2] with their own matching algorithm to identify copy-move attacks. They normalize the descriptors with norm L2 and take the dot product of it with its transposed. Then the points are matched using a threshold of 0.5 between the distance of the closest neighbor to that of the second-closest one to filter-out these points below. This threshold means if the first distance is smaller than the second multiplied by 0.5 they get filtered-out. The points also have to fulfill a greater euclidean distance than 10 to be matched points. So it is also possible to do multiple key-point matching with euclidean distance.

[2] From COCO Image Database[60]

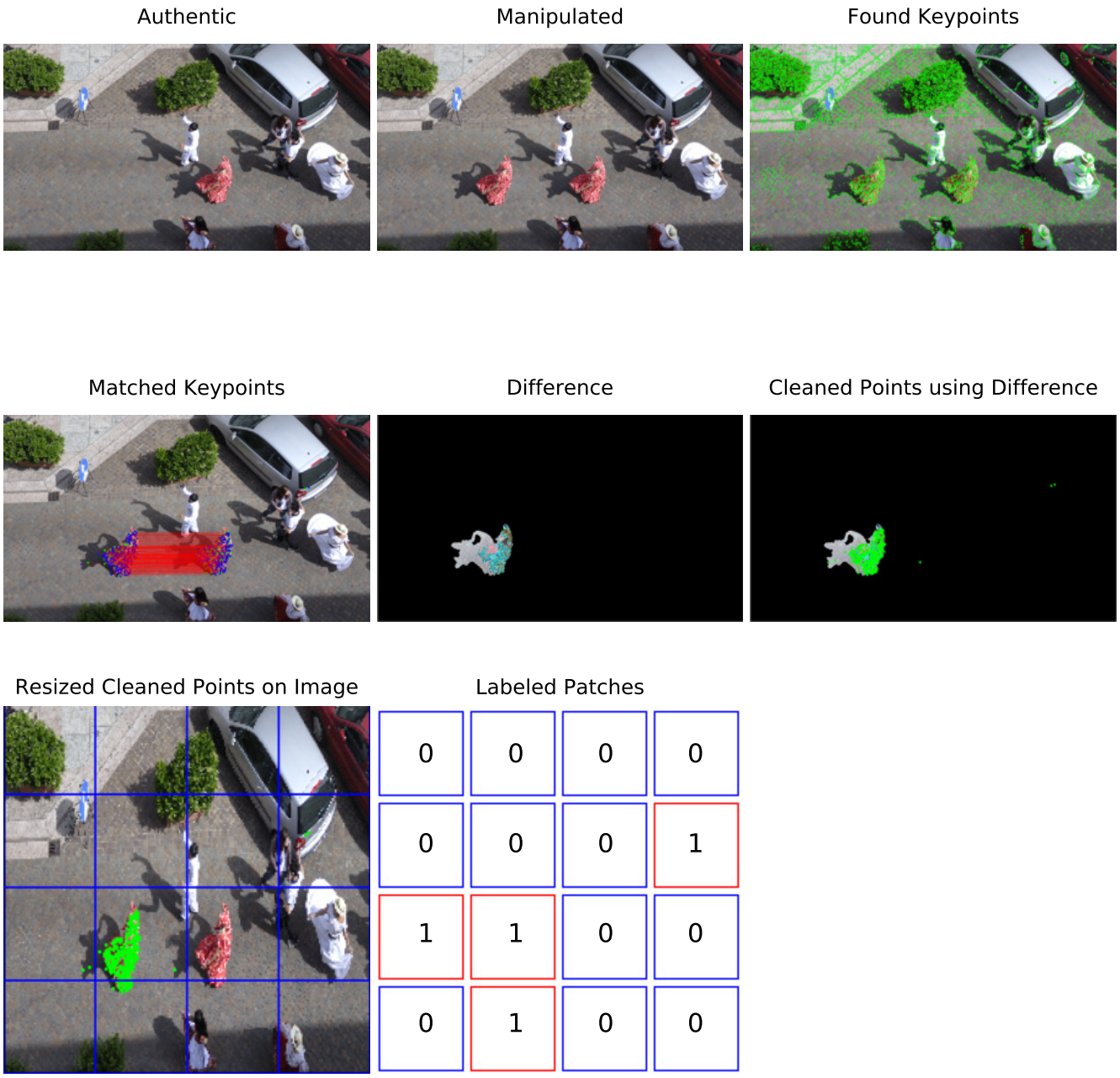

Figure 4.3.: These steps have been made to make SIFT[2] comparable starting with original image size[3]. The columns show from left to right: First row the authentic image, followed by the tampered and the found keypoints on tampered image. Second row shows detection result with matched keypoints on the tampered image, their difference(tampered - authentic) and the cleaned points using the difference. Then the third row shows the cleaned points on the tampered image resized to 224×224 with marked patches and finally the labels for each patch (label as 1(tampered), if at least one point is in the patch).

For detection the original image size is used, because that lead to better results. To make the detections comparable with the patch-based model, the point pairs will be checked against the difference of their images. If one of the points of the pair is inside the difference, the corresponding one is deleted if he is outside. This deletes the points of the source where we copied the area from and still leaves the outliers. To compare on our Patch-based approach level the points are scaled to 224×224, to check then if a point is in between the borders of a patch to mark it as tampered. How these steps look like can be seen in Figure 4.3.

[3] From Realistic Tampering Dataset[62]

The original code is written in matlab[4], we re-implemented the matching algorithm (match_features function) only, using SIFT[2] from OpenCV in python.

4.2.2 EM with Segmentation (Splicing)

As in this paper, which is called Splicebuster[18], proposed, we use the unsupervised scenario, which learns on the image itself, no prior knowledge is needed. For feature extraction they use computation of residuals through linear high-pass filtering of the third order. Then they quantize the residuals by using a very small number of bins(two) to obtain a limit feature length and then truncate the value at one. After that they compute a histogram of column-wise co-occurrences for four pixels in a row based on symmetry considerations. Then they pass the normalized histograms through a square-root non-linearity to obtain a final feature with unitary L2 norm, to capture traces left by locally in-camera processing. Euclidean distance is then used to compare the histograms. For classification with these features they use EM clustering. The authentic class is defined as multivariate Gaussian, while the tampered class is defined as uniform over the feature domain. They run it 30 times with different random initial parameters to select then the outcome, for which the data exhibit the highest likelihood, because the results of the EM algorithm strongly depend on the initialization. All images are converted to grayscale before processing. They also use different kernel-sizes depending on the image-size, if the image-size is greater than $20,000$ pixels the kernel is 128×128, otherwise 64×64.

For detection, the original image size is used to ensure better results. To make it comparable with the patch-based model, we take the 8 bit bitmap range $(0, 255)$ of the detection and scale it to 224×224. Then we use a threshold to mark a patch as tampered, if there is more than one pixel above the threshold. Their original code[5] is used for detection, but if there couldn't be detected any result the labels are filled with a not allowed value, which means no result, which is excluded in the statistics. How these steps look like can be seen in Figure 4.4.

4.2.3 Two Stream Faster-RCNN with Bilinear Pooling

This paper Learning Rich Features for Image Manipulation Detection (LRFfIMD)[32] uses a CNN[45] based approach and it can detect both local pixel manipulation problems and also removal-attacks. They used a two stream(RGB, SRM) Faster-RCNN[33] with a RPN. Regions are only proposed by the RGB-stream, the SRM[29]-stream (filters to extract image noise) is only to improve detection results. The RPN output is a bunch of box proposals that will be predicted by the possibility of an anchor proposal being background or foreground which are only extracted from the output of the RGB-Stream. Anchor proposals are generated with a sliding window over defined anchor-sizes. The output box proposals are then compared with the ground truth bounding box and give back the predicted bounding boxes and the probability the box contains a manipulated region. The region proposals are used then as input for Region of Interest (ROI) in both streams, where each bounding box is divided into equal-sized sections, where the max value will be calculated for each section. These ROIs are then used in Bilinear Pooling[35] to combine the two-streams of each CNN network while preserving spatial information to improve detection confidence. Each stream uses an on Image Net pre-trained ResNet 101[34] which then leads to bilinear pooling[35]. A ResNet[34] uses deep residual learning with residual blocks which is a convolutional block where at the end the input is added to the output of the block. This architecture leads to $89,372,434$ roughly $\sim 90m$ trainable parameters.

To build the ground truth (GT), we take the difference between the tampered and the original image for marking the ground truth (GT) bounding boxes. We use a 3×3 opening filter to get rid of the one and two pixel differences which we already mentioned. Especially for Columbia[61], with sometimes appearing noise differences, they need to be cleaned, to not mark the whole image as ground truth. If the difference was too small so the opening filter eliminated it completely, we use a logical check for the one pixel and a neighbor cleaning for the two pixel differences to remove them. Then we mark the difference with a bounding box (+20 pixels bigger area to better distinguish between tampered area and background as suggested in the paper).

[4] `http://lci.micc.unifi.it/labd/cmfd/sift-forensic.zip` (visited on 06/06/19)

[5] `http://www.grip.unina.it/download/prog/Splicebuster/Splicebuster.zip` (visited on 06/06/19)

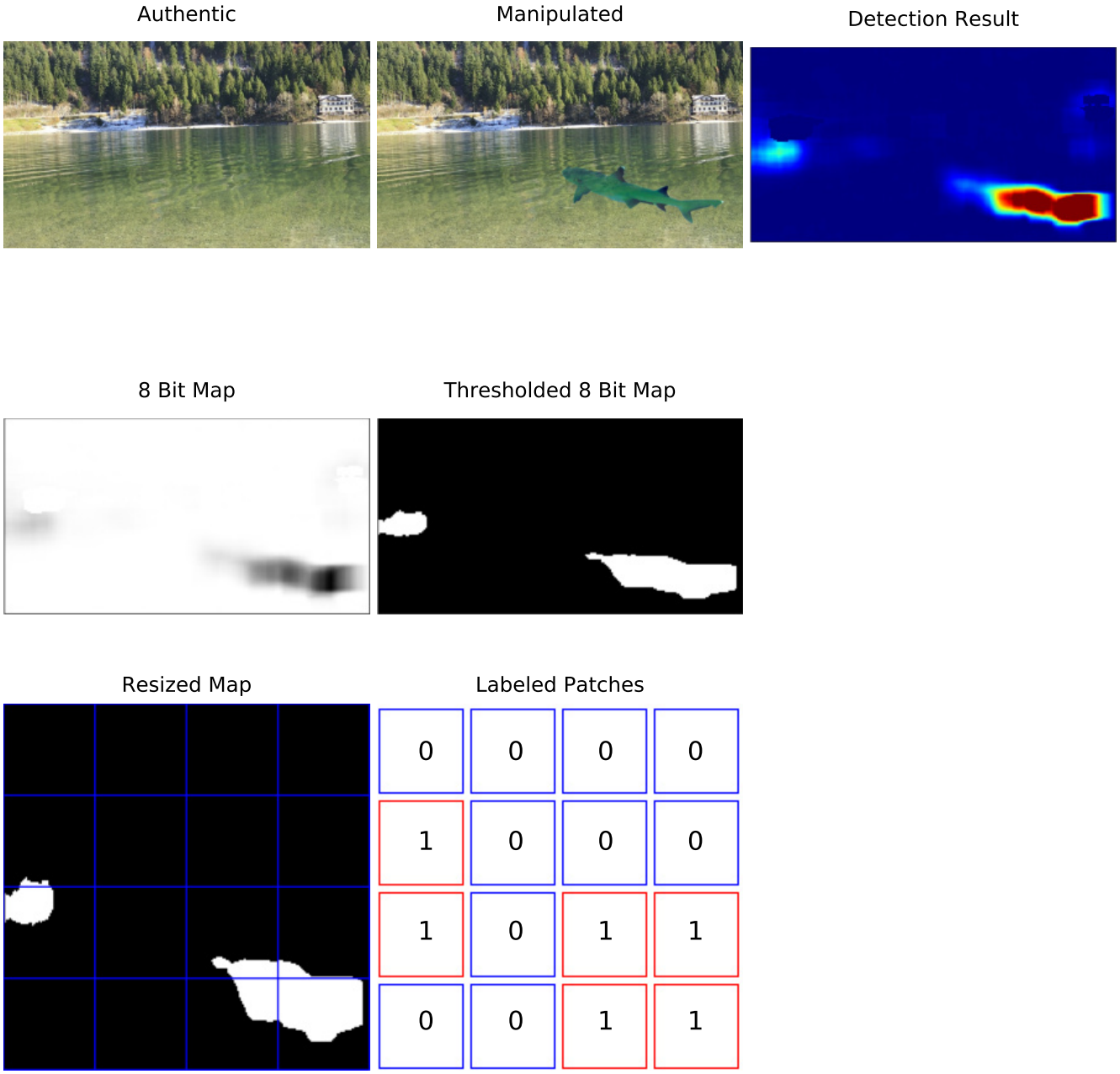

Figure 4.4.: These steps have been made to make Splicebuster[18] comparable, starting with original image size[6]. The columns show from left to right: First row the authentic image, followed by the tampered, detection result heatmap on the tampered image. Second row 8 Bit Map of the heatmap, thresholded 8 Bit Map (10% of 255 = 25, 5). Third row thresholded 8 Bit Map resized to 224×224 and finally the labels for each patch (label as 1(tampered), if at least one pixel is in the patch).

For training 32,400 and validation 3,600 tampered images are used, while for testing 8,000 tampered and authentic images are used, half-and-half. The authentic images are not marked with a GT which means no bounding box, because we use the target image background, not the source image as in the paper. The number of training steps are calculated the following: We use about 25 epochs for training, that's in average about the same number we use for our model (average epochs of three runs for copy-move $\sim 26,67$ and for splicing 24 for the segmentation case, which leads in total average to ~ 25 epochs). As written at the beginning we use 32,400 images for training divided by 64(batch size), which leads to 506.25(steps per epoch). These steps per epoch $\times 25$ epochs leads to $\sim 12,657$ steps for training.

[6] From Realistic Tampering Dataset[62]

Figure 4.5.: These steps have been made to make the two stream Faster-RCNN comparable. We start with original image size[7]. The columns show from left to right: First row the authentic image, followed by the tampered, difference between the images(tampered - authentic), difference cleaned with filters, marked calculated ground truth bounding box on the difference. Second row shows: Marked calculated ground truth on tampered image, until here this steps are preparation before training and detection. After detection, we start with: Marked detection result bounding boxes, cleaned detection result with threshold $>= 70\%$ bounding box score, result resized to 224×224 with marked patches and finally the labels for each patch (label as 1(tampered), if the box overlaps with a patch or the patch is inside the box).

After prediction, we get bounding boxes with scores for every image. To filter-out all lower score's we use a threshold as for example 70%, where everything above is used as confident detection result. Means this bounding boxes are marked as tampered. All images with non left bounding box are completely labeled as authentic.

To make it comparable with our patch-based model, we take the proposed detected bounding boxes above the threshold for each image and check if a part of it overlaps with a patch, to mark the patch as tampered. How these steps look like can be seen in Figure 4.5.

We use their original code[8] provided by the first author.

[7] From generated splicing COCO Dataset[60].

[8] `https://github.com/pengzhou1108/RGB-N` (visited on 06/06/19)

4.3 Patch-based Model (Our Approach)

In our setup it is better to have a higher recall than precision, because it is better to mark an authentic patch as tampered than the other way around. The best case would be both are high. In general we like to have a high F_1 score for classifying the tampered patches, which indicates a better overall performance. Of course the authentic images play a role too, but they are always classified with a high score. Our architecture leads to $646,978$ roughly $\sim 650k$ trainable parameters, which is just a small percentage compared to Learning Rich Features for Image Manipulation Detection (LRFfIMD)[32].

Some detection results on test data for the segmentation data sets can be seen at Figure 4.7 for splicing and at Figure 4.8 for copy-move, marked with the best detectable GT (would be the same as training GT). They could be nearly detected as expected for the splicing case, the copy-move case does not work so well as also can be seen under evaluation. Further detection results trained on the segmentation data set but tested on other data sets are shown in Figure 4.9 for copy-move and in Figure 4.10 for splicing, which are also used in the evaluation part. For some demonstration detection results on test data for the rectangle data set, some examples are given in Figure 4.11, also marked with the best detectable GT. They can be detected with a high score, which gets clear by having a look at the results in the evaluation part.

4.4 Evaluation

In this section we will evaluate our approach on different data sets compared with the above listed models. Firstly we have a look if filtering of the training data was a good decision. Then we compare the learning effect of the seen data over the trained epochs on the test data for F_1 score. For measurement, we take F_1 score to be able to measure both classes fairly, if we take accuracy which is standard, we would only measure the authentic class. After that we have a look at the results of the splicing and copy-move attack.

We define the training GT as at least one channel fulfills the lower bound of 0.05% and the upper bound $0.08(1-0.08 = 0.92)$% number of pixels in the difference per patch are manipulated. The real GT is defined as at least three and maximum all pixels are manipulated over all three channels per patch. Means we allow one and two pixel differences which are not connected to other pixels. This small differences can appear between some authentic and tampered image pairs, which we use for GT definition as already shown in **??**. This threshold is a logical consequence to make all models comparable and has also the effect that three changed pixels can be seen as manipulation and not as error as just one pixel. The GT for the Columbia[61] data set is cleaned, because of the noise differences, which can appear between authentic and tampered images, which leads to a wrong GT. For the training GT of the Columbia[61] data set we use only an opening filter for cleaning. For the real GT we used a small object cleaning additional, to make sure that there are no pixel differences left (this cleaning removes too much for the training GT thresholds, that is why we can not use it for both).

We then take the mean and the standard deviation of three random shuffled runs for each model. Each model is tested and trained on the exact same data, means the training/validation/testing data of the three different shuffle runs we use for all models are the same, to have a fair comparison. For the basic methods like SIFT[7] or Splicebuster[18] we only use one run on the COVERAGE[56], Columbia[61] and Realistic Tampering Dataset[62]. We take one run, because we take the complete data set for testing and there is no model change as for the models which are trained. Especially for Splicebuster[18] we measure the number of images which couldn't be calculated by the algorithm in an additional line.

4.4.1 Verification of the Out-filtering

In Table 4.2 we tested if the decision to filter-out all image pairs where not even one patch has been marked as tampered was the right one. The table shows that there is an improvement for the copy-move case, if we change from RandomOverSampling[52] Table 3.1 to Borderline-SMOTE[54] method with a changed image bunch-size. The biggest improvement compared to RandomOversampling[52] is achieved in precision, which has a positive effect on the F_1 score too. This effect is the case, because the algorithm tries to sample new patches in danger (at least half are from the same class, means close to class borders), which leads to better differentiation between authentic and tampered patches. We can also see there is still an improvement, if the data gets filtered before training, instead of using all tampered images for the copy-move case. On the other hand in the splicing case it seems to be slightly better to use the tampered images without filtering out. In summary the decision should be made by considering the data set and the over-sampling technique.

Dataset	Method	Precision tamp.	Recall tamp.	F_1 score tamp.	F_1 score both	Ratio	#Training Images
COCO[60] Segmentation Copy-Move	Tamp. only	**0.797** ±0.045	0.517 ±0.045	0.623 ±0.021	0.807 ±0.012	0.075±0	32,400±0
	Filtered Out	0.757 ±0.017	**0.580** ±0.014	**0.660** ±0.014	**0.823** ±0.005	0.117±0	20,888.33±18.52
COCO[60] Segmentation Splicing	Tamp. only	**0.883** ±0.017	0.703 ±0.021	**0.783** ±0.017	**0.880** ±0.008	0.169±0	32,400±0
	Filtered Out	0.847 ±0.005	**0.707** ±0.034	0.773 ±0.017	0.877 ±0.012	0.216±0	25,477.33±7.59

Table 4.2.: Testing of F_1 score macro average for segmentation to show if decision of filtering the data was right. We use the mean±std in macro average over three runs(same start training data $64,800$, same validation $7,200$ / test data $8,000$). As ground truth we used at least one channel fulfills lower bound 0.05% and upper bound $0.08(1-0.08=0.92)\%$ of pixels which are manipulated in one patch. Over-sampling method Borderline-SMOTE[54] with a bunch of 32 images is used. The ratio is calculated the following #patches (tampered/all).

4.4.2 Impact of the seen Data on the Results

The learning effect of the seen data for the LRFfIMD[32] model and ours can be seen in Figure 4.6. We show here the effect over 25 epochs, where one epoch means the whole training data set has been seen. We distinguish here between authentic and tampered class and their macro average F_1 score. In the first row we can see copy-move followed by splicing macro average F_1 score over 25 epochs tested on the segmentation test set with mean and std over three runs. The second row shows the same but only F_1 score for class tampered. Further explanations can be read at Figure 4.6.

In the copy-move case the LRFfIMD[32] model learns better to distinguish between authentic and tampered patches than our model. For the splicing case there is no improvement until epoch 15 for LRFfIMD[32], because the algorithm could not learn to distinguish between tampered and authentic yet for the chosen threshold. Followed by a linear learning effect, with a high variation. As we can see our approach has less variation over the three runs than LRFfIMD[32].

4.4.3 Comparison of the Splicing-detection Results

The Table 4.3 shows test results on the segmentation test data sets, the Columbia[61] data set and the Realistic Tampering Dataset[62]. For Splicebuster[18] we used three different thresholds, where the table shows clearly that 50% works best for all data sets, for both cases segmentation and rectangle. For LRFfIMD[32] we took thresholds which seems fair for filtering-out boxes and scores under it. The patch is marked as tampered if even a bit of the bounding box overlaps with the patch. Which threshold performs best changes for every data set, but overall data sets 50% with using only the highest score bounding box, seems to work best for segmentation and rectangle using the mean of F_1 score. Our own approach works best for the COCO[60] segmentation splicing and for the Columbia[61] data set it works even better. For the Realistic Tampering Dataset[62] Splicebuster[18] at 50% works best, but our approach is very close.

In Table 4.4 for the splicing rectangle case it can be seen that our model performs better than Splicebuster[18] and LRFfIMD[32] for all testing data sets but the Realistic Tampering Dataset[62]. Our model learns better to predict the rectangle than the segmentation case. The LRFfIMD[32] model can also better learn the rectangle than the segmentation case, but it generalizes badly on the tampered patches for Columbia[61] and the Realistic Tampering Dataset[62]. Quite interesting is that our model trained on the rectangle data sets is just slightly worse in predicting than the model trained on segmentation for the Columbia[61] data set.

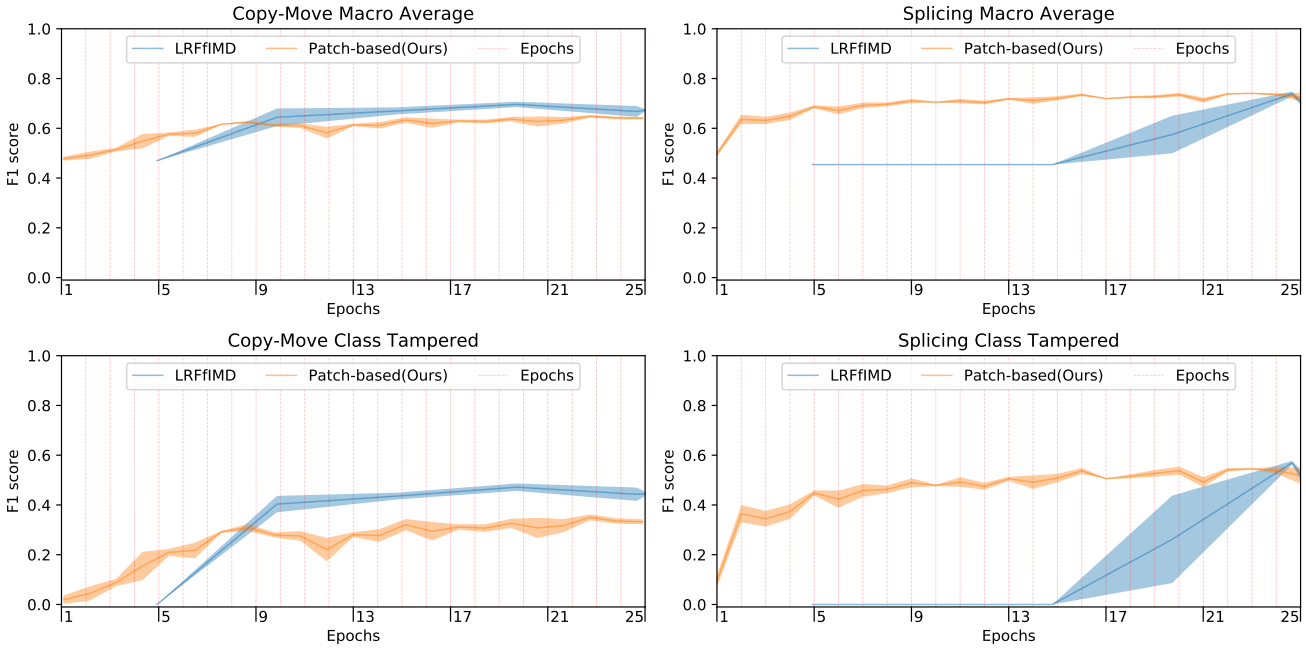

Figure 4.6.: We used 25 epochs for the copy-move and splicing segmentation case with a bunch-size of 32 images to show the impact of the seen data on F_1 score of the testing data. The paper LRFfIMD[32] is only calculated five times at about every 20% over all steps, because calculations are heavy. We use all 2496 steps for testing for both models, but in our approach the epochs have a different number of steps for each run, because of the filtering out. Further our model uses patches and LRFfIMD[32] uses whole images as learning basis. The step size comes from 312×16 (patches) $= 4992 =\sim 5000$ patches. Because of that the calculations are heavy we chose $312 \times 8 = 2946$ steps what nearly fits into the 20% of all steps for LRFfIMD[32] where every step represents 64 images. We choose then the same step-size for our own model but every step represents 256 patches. All steps of our own model are scaled to fit the LRFfIMD[32] steps to show them over the same number of epochs. The scale factor is calculated the following: $12657(steps)/25(epochs) = 506.28$ (steps per epoch), these steps per epoch are then divided by the steps per epoch for each run of our model. To make them comparable at the same ground truth basis we chose the threshold for LRFfIMD[32] at $>= 70\%$ for the bounding boxes scores with minimal overlapping for the patches to mark a patch a tampered. As ground truth three pixels as minimum manipulation have been chosen. The mean is represented by the middle line, surrounded by the std.

4.4.4 Comparison of the Copy-Move-detection Results

The results in Table 4.5 show that SIFT[7] works best over all data sets. Using at least three points in one patch to mark it as tampered performs better than just using one point for segmentation and rectangle. Our model only performs better than SIFT[7] for the COCO segmentation data set. For LRFfIMD[32] we took the same thresholds as for the splicing case. The paper seems to perform best with 70% by using only the best score bounding box for segmentation and rectangle using the mean of F_1 score over all data sets. It performs best for the COCO[60] segmentation data set overall other methods, but our approach is really close to the best. For COVERAGE[56] and the Realistic Tampering Dataset[62] our model performs better than LRFfIMD[32].

As in Table 4.6 can be seen SIFT[7] is still better on the COVERAGE[56] and the Realistic Tampering Dataset[62], but our model performs better than SIFT[7] in the rectangle test data set. Our model learns better to predict the rectangle than the segmentation case. The LRFfIMD[32] model can also better learn the rectangle than the segmentation case, but it generalizes badly on the tampered patches for Columbia[61] and the Realistic Tampering Dataset[62].

Dataset	COCO[60] Segmentation Splicing			Columbia[61]			Realistic Tampering Dataset[62]		
#Tested Images (50:50)	8,000			360			440		
Method (#Train Images)	A	T	Both	A	T	Both	A	T	Both
Splicebuster[18] (#0)									
undetectable	(#9.67)			(#0)			(#0)		
(th 1, 1 pix)	0.142 ±0.0022	0.247 ±0.0028	0.194 ±0.0025	0.302 (1 run)	0.426 (1 run)	0.364 (1 run)	0.142 (1 run)	0.268 (1 run)	0.205 (1 run)
(th 10%, 1 pix)	0.645 ±0.0010	0.214 ±0.0005	0.429 ±0.0003	0.719 (1 run)	0.507 (1 run)	0.613 (1 run)	0.628 (1 run)	**0.304** (1 run)	0.466 (1 run)
(th 50%, 1 pix)	0.824 ±0.0010	0.162 ±0.0006	0.493 ±0.0003	0.810 (1 run)	0.421 (1 run)	0.616 (1 run)	0.830 (1 run)	0.296 (1 run)	**0.563** (1 run)
LRFfIMD[32] (#32,400)									
(th 70%) all	0.889 ±0.015	0.528 ±0.018	0.708 ±0.015	0.856 ±0.008	0.538 ±0.037	0.697 ±0.018	0.890 ±0.012	0.146 ±0.029	0.518 ±0.009
(th 70%) best	0.909 ±0.008	0.532 ±0.020	0.721 ±0.012	0.852 ±0.007	0.537 ±0.036	0.695 ±0.018	**0.899** ±0.008	0.124 ±0.017	0.511 ±0.005
(th 50%) all	0.847 ±0.021	0.487 ±0.022	0.667 ±0.021	0.838 ±0.013	0.627 ±0.003	0.733 ±0.005	0.867 ±0.016	0.185 ±0.029	0.526 ±0.007
(th 50%) best	0.890 ±0.008	0.514 ±0.020	0.702 ±0.014	0.843 ±0.013	0.598 ±0.011	0.720 ±0.010	0.887 ±0.007	0.158 ±0.023	0.523 ±0.009
Patch-based(Ours) (#25,477.25)									
Real GT	**0.952** ±0.001	**0.589** ±0.018	**0.771** ±0.010	**0.907** ±0.002	**0.680** ±0.008	**0.794** ±0.004	0.868 ±0.007	0.248 ±0.011	0.558 ±0.007
Training GT	0.981 ±0.001	0.773 ±0.020	0.877 ±0.011	0.941 ±0.003	0.736 ±0.007	0.838 ±0.004	0.890 ±0.006	0.240 ±0.010	0.565 ±0.007

Table 4.3.: Testing of F_1 score macro average for the splicing segmentation case on the real ground truth. The shortcuts represent A (Class Authentic) and T (Class Tampered). After the method name the (#mean number of train images) follows. We show the mean±std over three runs in each column. The tested images data set are half-and-half on authentic images and their manipulated version. For Splicebuster[18] the first line shows the (#mean of undetectable images) over three runs. in the columns (1 run) means we only tested it once, because the data set does not change. We used three different thresholds, 1 means we have a range of values from 0 to 255 at 1 the pixel values are interpreted as tampered. The 10% threshold represents 25.5 and 50% 127.5 as pixel threshold number. One pixel in a patch is then used as threshold to mark a patch a tampered. For LRFfIMD[32] minimal overlapping of the detected bounding boxes with the patches is used instead. The threshold used here means percentage under which the scores and their boxes are discarded. The word "all" means the threshold is used for all boxes, "best" means we only keep the best score and then use the threshold. The real ground truth is defined, that at least three and maximum all pixels are manipulated per patch. Training ground truth is defined as at least one channel fulfills lower bound 0.05% and upper bound $0.08(1-0.08=0.92)$% of pixels which are manipulated in one patch.

Dataset	COCO[60] Rectangle Splicing			Columbia[61]			Realistic Tampering Dataset[62]		
#Tested Images (50:50)	8,000			360			440		
Type (#Train Images)	A	T	Both	A	T	Both	A	T	Both
Splicebuster[18] (#0)									
undetectable	(#4.67)			(#0)			(#0)		
(th 1, 1 pix)	0.141 ±0.003	0.156 ±0.0004	0.148 ±0.002	0.302 (1 run)	0.426 (1 run)	0.364 (1 run)	0.142 (1 run)	0.268 (1 run)	0.205 (1 run)
(th 10%, 1 pix)	0.674 ±0.0014	0.172 ±0.0015	0.423 ±0.001	0.719 (1 run)	0.507 (1 run)	0.613 (1 run)	0.628 (1 run)	**0.304** (1 run)	0.466 (1 run)
(th 50%, 1 pix)	0.862 ±0.0008	0.184 ±0.001	0.523 ±0.0002	0.810 (1 run)	0.421 (1 run)	0.616 (1 run)	0.830 (1 run)	0.296 (1 run)	**0.563** (1 run)
LRFfIMD[32] (#32,400)									
(th 70%) all	0.961 ±0.004	0.698 ±0.019	0.830 ±0.011	0.850 ±0.002	0.128 ±0.037	0.489 ±0.019	0.915 ±0.003	0.046 ±0.031	0.481 ±0.014
(th 70%) best	0.964 ±0.002	0.711 ±0.014	0.837 ±0.008	0.850 ±0.002	0.118 ±0.025	0.484 ±0.013	**0.916** ±0.003	0.042 ±0.028	0.479 ±0.013
(th 50%) all	0.959 ±0.005	0.686 ±0.026	0.823 ±0.016	0.850 ±0.003	0.163 ±0.042	0.506 ±0.021	0.914 ±0.005	0.057 ±0.032	0.485 ±0.014
(th 50%) best	0.962 ±0.003	0.704 ±0.018	0.833 ±0.011	0.849 ±0.003	0.149 ±0.031	0.499 ±0.015	0.915 ±0.004	0.052 ±0.029	0.483 ±0.013
Patch-based(Ours) (#31,338)									
Real GT	**0.981** ±0.0002	**0.733** ±0.001	**0.857** ±0.0007	**0.906** ±0.003	**0.651** ±0.006	**0.779** ±0.003	0.906 ±0.004	0.179 ±0.025	0.543 ±0.012
Training GT	0.996 ±0.0001	0.922 ±0.002	0.959 ±0.001	0.969 ±0.005	0.832 ±0.019	0.900 ±0.012	0.930 ±0.004	0.200 ±0.025	0.565 ±0.012

Table 4.4.: Testing of F_1 score macro average for the splicing rectangle case on the real ground truth. The shortcuts represent A (Class Authentic) and T (Class Tampered). After the method name the (#mean number of train images) follows. We show the mean±std over three runs in each column. The tested images data set are half-and-half on authentic images and their manipulated version. For Splicebuster[18] the first line shows the (#mean of undetectable images) over three runs. in the columns (1 run) means we only tested it once, because the data set does not change. We used three different thresholds, 1 means we have a range of values from 0 to 255 at 1 the pixel values are interpreted as tampered. The 10% threshold represents 25.5 and 50% 127.5 as pixel threshold number. One pixel in a patch is then used as threshold to mark a patch a tampered. For LRFfIMD[32] minimal overlapping of the detected bounding boxes with the patches is used instead. The threshold used here means percentage under which the scores and their boxes are discarded. The word "all" means the threshold is used for all boxes, "best" means we only keep the best score and then use the threshold. The real ground truth is defined, that at least three and maximum all pixels are manipulated per patch. Training ground truth is defined as at least one channel fulfills lower bound 0.05% and upper bound $0.08(1-0.08=0.92)$% of pixels which are manipulated in one patch.

Dataset	COCO[60] Segmentation Copy-Move			COVERAGE[56]			Realistic Tampering Dataset[62]		
#Tested Images (50:50)	8,000			200			440		
Type (#Train Images)	A	T	Both	A	T	Both	A	T	Both
SIFT[7] (#0)									
(1 point)	0.953 ±0.0004	0.443 ±0.0003	0.698 ±0.002	0.839 (1 run)	0.504 (1 run)	0.672 (1 run)	0.904 (1 run)	**0.323** (1 run)	**0.614** (1 run)
(3 points)	0.966 ±0.0001	0.429 ±0.008	0.698 ±0.004	0.891 (1 run)	**0.541** (1 run)	**0.716** (1 run)	**0.916** (1 run)	0.295 (1 run)	0.605 (1 run)
LRFfIMD[32] (#32,400)									
(th 70%) all	0.918 ±0.0008	0.449 ±0.007	0.683 ±0.004	0.850 ±0.016	0.309 ±0.043	0.580 ±0.015	0.862 ±0.009	0.214 ±0.021	0.538 ±0.015
(th 70%) best	0.944 ±0.002	**0.517** ±0.006	**0.730** ±0.004	0.862 ±0.012	0.277 ±0.030	0.569 ±0.012	0.887 ±0.006	0.182 ±0.028	0.534 ±0.016
(th 50%) all	0.882 ±0.009	0.384 ±0.014	0.633 ±0.012	0.823 ±0.032	0.325 ±0.041	0.574 ±0.011	0.826 ±0.011	0.233 ±0.013	0.529 ±0.012
(th 50%) best	0.933 ±0.005	0.482 ±0.010	0.707 ±0.007	0.850 ±0.018	0.289 ±0.040	0.569 ±0.014	0.877 ±0.004	0.190 ±0.021	0.533 ±0.011
Patch-based(Ours) (#20,888.33)									
Real GT	**0.972** ±0.0003	0.481 ±0.010	0.727 ±0.005	**0.894** ±0.004	0.278 ±0.011	0.586 ±0.007	0.851 ±0.007	0.241 ±0.007	0.546 ±0.004
Training GT	0.988 ±0.0004	0.658 ±0.014	0.823 ±0.007	0.912 ±0.004	0.294 ±0.014	0.603 ±0.009	0.871 ±0.007	0.225 ±0.005	0.548 ±0.004

Table 4.5.: Testing of F_1 score macro average for the copy-move segmentation case on the real ground truth. The shortcuts represent A (Class Authentic) and T (Class Tampered). After the method name the (#mean number of train images) follows. We show the mean±std over three runs in each column. The tested images data set are half-and-half on authentic images and their manipulated version. For SIFT[7] (1 run) means we only tested it once, because the data set does not change. The number of points means how many we used as minimal in one patch to mark a patch a tampered. For LRFfIMD[32] minimal overlapping of the detected bounding boxes with the patches is used instead. The threshold used here means percentage under which the scores and their boxes are discarded. The word "all" means the threshold is used for all boxes, "best" means we only keep the best score and then use the threshold. The real ground truth is defined, that at least three and maximum all pixels are manipulated per patch. Training ground truth is defined as at least one channel fulfills lower bound 0.05% and upper bound $0.08(1-0.08=0.92)\%$ of pixels which are manipulated in one patch.

Dataset	COCO[60] Rectangle Copy-Move			COVERAGE[56]			Realistic Tampering Dataset[62]		
#Tested Images (50:50)	8,000			200			440		
Type (#Train Images)	A	T	Both	A	T	Both	A	T	Both
SIFT[7] (#0)									
(1 point)	0.949 ±0.0002	0.407 ±0.003	0.678 ±0.002	0.839 (1 run)	0.504 (1 run)	0.672 (1 run)	0.904 (1 run)	**0.323** (1 run)	**0.614** (1 run)
(3 points)	0.959 ±0.0003	0.348 ±0.005	0.654 ±0.003	0.891 (1 run)	**0.541** (1 run)	**0.716** (1 run)	**0.916** (1 run)	0.295 (1 run)	0.605 (1 run)
LRFfIMD[32] (#32,400)									
(th 70%) all	0.947 ±0.014	0.637 ±0.055	0.792 ±0.034	0.848 ±0.033	0.187 ±0.044	0.517 ±0.006	0.902 ±0.009	0.097 ±0.048	0.500 ±0.020
(th 70%) best	0.957 ±0.008	0.675 ±0.036	0.816 ±0.022	0.864 ±0.018	0.153 ±0.025	0.508 ±0.005	0.906 ±0.007	0.085 ±0.039	0.496 ±0.016
(th 50%) all	0.938 ±0.019	0.605 ±0.066	0.772 ±0.042	0.835 ±0.039	0.198 ±0.048	0.517 ±0.005	0.895 ±0.013	0.115 ±0.050	0.505 ±0.019
(th 50%) best	0.953 ±0.009	0.656 ±0.041	0.804 ±0.025	0.858 ±0.021	0.161 ±0.029	0.509 ±0.010	0.901 ±0.010	0.099 ±0.040	0.500 ±0.015
Patch-based(Ours) (#31,595)									
Real GT	**0.978** ±0.0008	**0.681** ±0.013	**0.830** ±0.007	**0.895** ±0.001	0.143 ±0.011	0.519 ±0.005	0.893 ±0.006	0.188 ±0.014	0.540 ±0.005
Training GT	0.993 ±0.0007	0.872 ±0.002	0.932 ±0.008	0.914 ±0.001	0.157 ±0.013	0.535 ±0.007	0.916 ±0.006	0.199 ±0.010	0.557 ±0.004

Table 4.6.: Testing of F_1 score macro average for the copy-move rectangle case on the real ground truth. The shortcuts represent A (Class Authentic) and T (Class Tampered). After the method name the (#mean number of train images) follows. We show the mean±std over three runs in each column. The tested images data set are half-and-half on authentic images and their manipulated version. For SIFT[7] (1 run) means we only tested it once, because the data set does not change. The number of points means how many we used as minimal in one patch to mark a patch a tampered. For LRFfIMD[32] minimal overlapping of the detected bounding boxes with the patches is used instead. The threshold used here means percentage under which the scores and their boxes are discarded. The word "all" means the threshold is used for all boxes, "best" means we only keep the best score and then use the threshold. The real ground truth is defined, that at least three and maximum all pixels are manipulated per patch. Training ground truth is defined as at least one channel fulfills lower bound 0.05% and upper bound $0.08(1-0.08=0.92)\%$ of pixels which are manipulated in one patch.

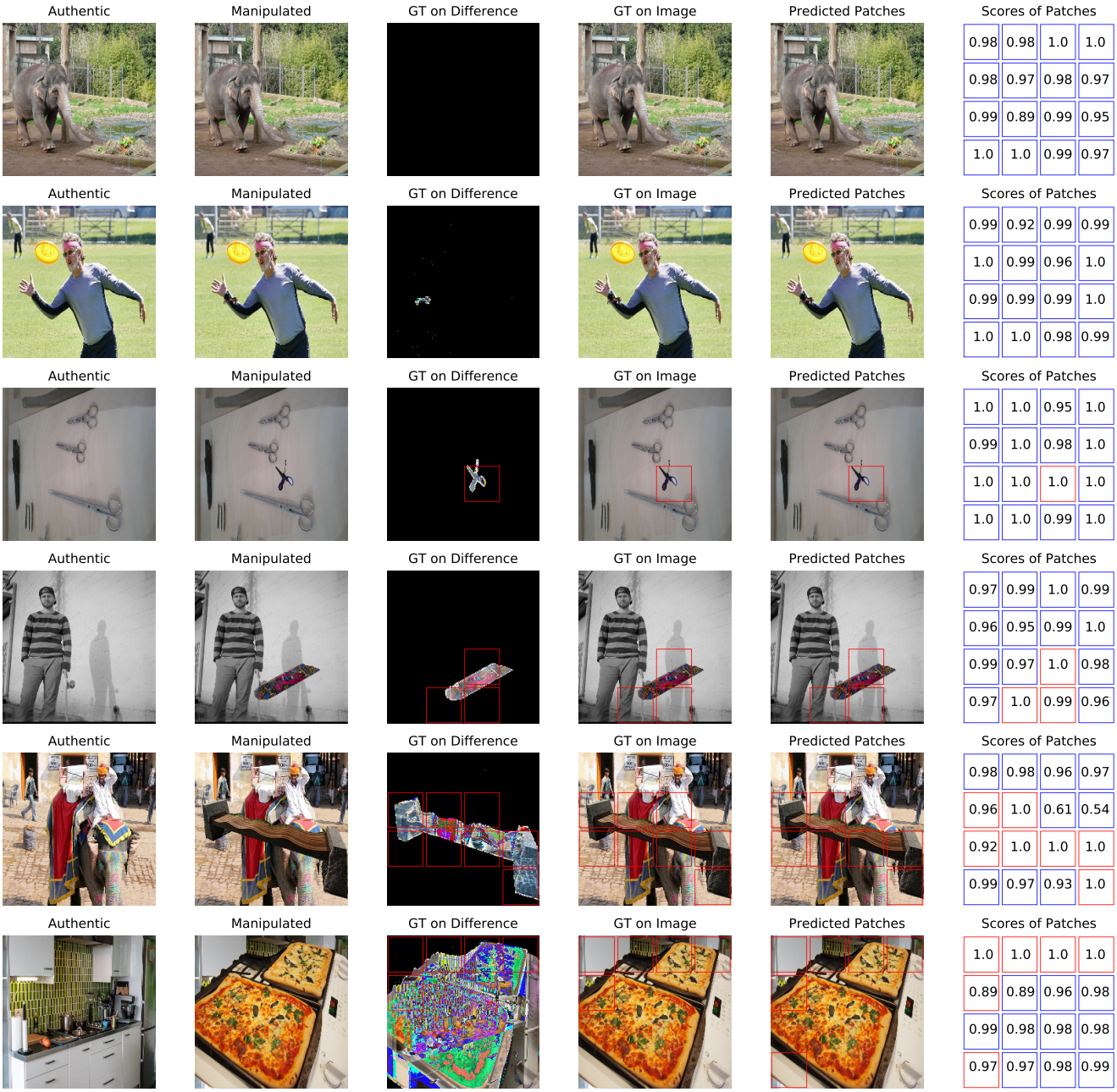

Figure 4.7.: Some detection results of the splicing segmentation case are shown here. Images[9] are resized to 224×224. The columns show from left to right: The authentic image, followed by the tampered, their difference(tampered - authentic), marked best detectable ground truth on the difference, marked best ground truth on the tampered image, detection result and finally the predicted scores for each patch which leads to the result. First row shows a normal image, second a too small difference(under the threshold of 0.05%), third a small sized, fourth a normal sized, fifth a big sized and the last shows one where the area is too big(over the threshold of $0.08(1-0.08=0.92)\%$), so the whole patch is filled with another image. The gaps between marked patches are only to make them better visible.

6 From COCO Image Database[60], images produced with the generator.

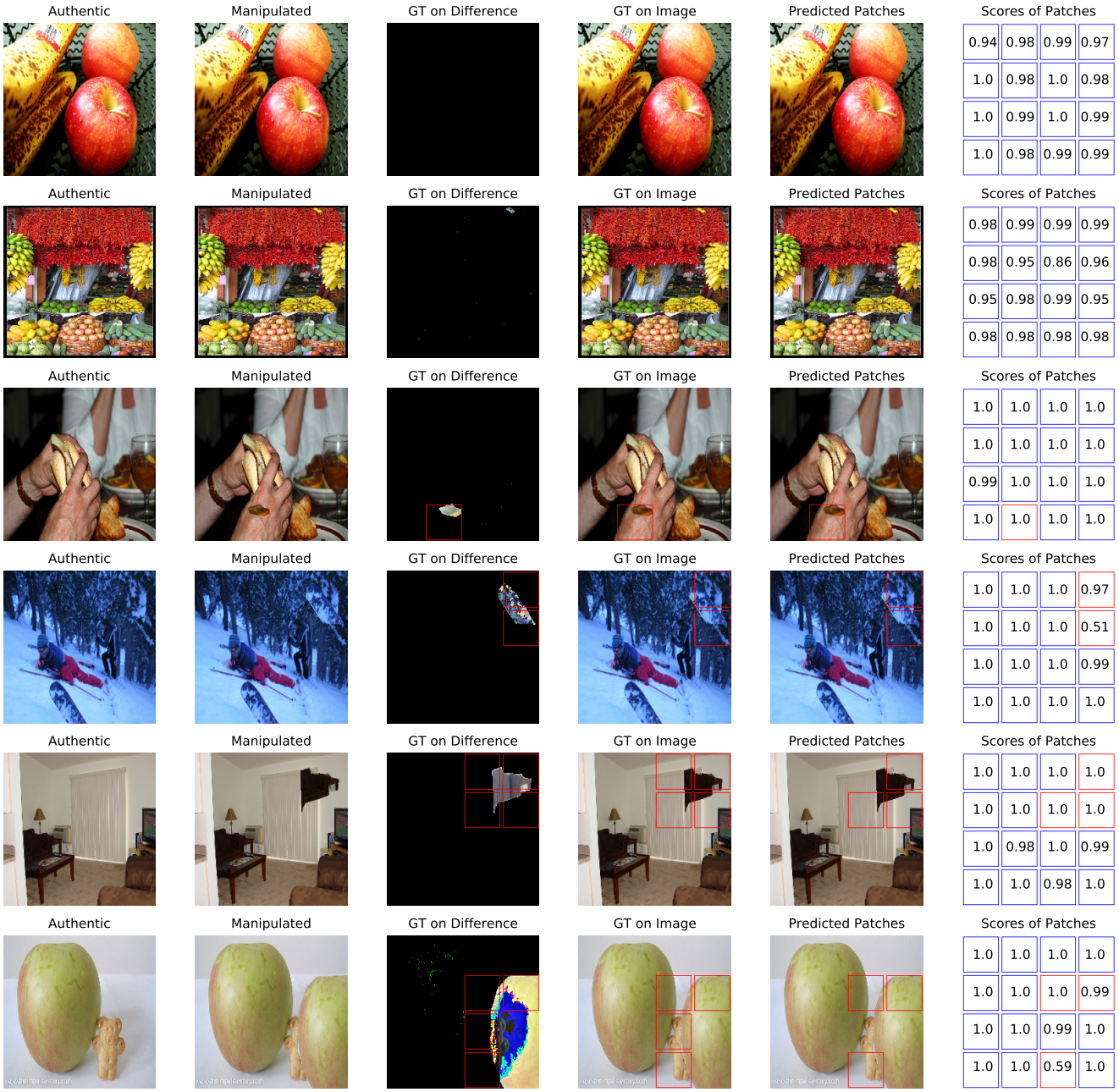

Figure 4.8.: Some detection results of the copy-move segmentation case are shown here. Images[10] are resized to 224×224. The columns show from left to right: The authentic image, followed by the tampered, their difference(tampered - authentic), marked best detectable ground truth on the difference, marked best ground truth on the tampered image, detection result and finally the predicted scores for each patch which leads to the result. First row shows a normal image, second a too small difference(under the threshold of 0.05%), third a small sized, fourth a normal sized, fifth a big sized and the last shows one where the area is too big(over the threshold of $0.08(1-0.08=0.92)\%$), so the whole patch is filled with another image. The gaps between marked patches are only to make them better visible.

[7] From COCO Image Database[60], images produced with the generator.

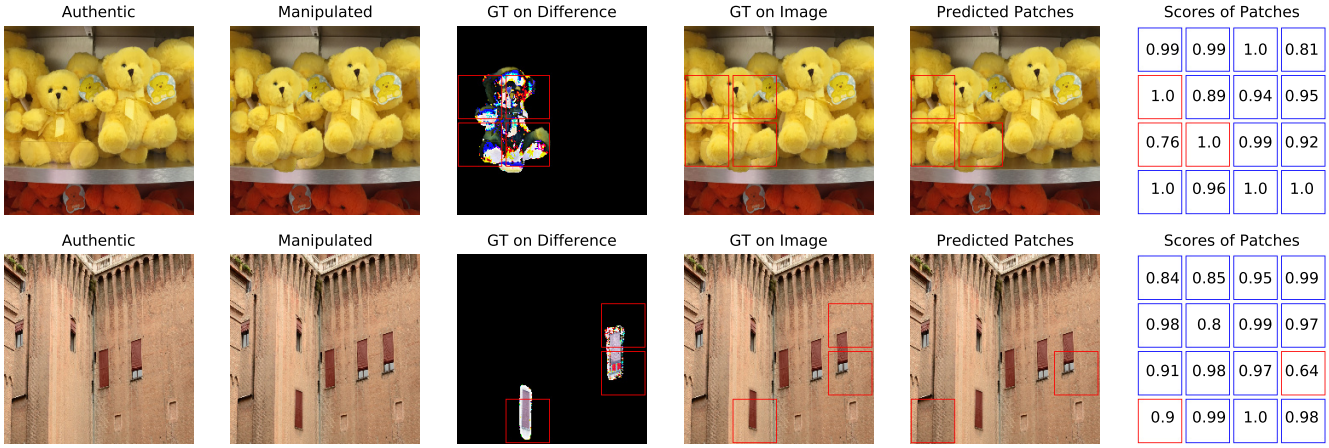

Figure 4.9.: Some detection results tested on both other data sets, trained on the copy-move segmentation case. Images[11] are resized to 224×224. The columns show from left to right: The authentic image, followed by the tampered, their difference(tampered - authentic), marked best detectable ground truth on the difference, marked best ground truth on the tampered image, detection result and finally the predicted scores for each patch which leads to the result. The first row shows COVERAGE[56] data set and the second row the Real Tampering Dataset[62]. The ground truth we used here is the same as for training which is defined as at least one channel fulfills lower bound 0.05% and upper bound $0.08(1 - 0.08 = 0.92)\%$ of pixels which are manipulated. The gaps between marked patches are only to make them better visible.

[8] First row COVERAGE[56], second row Real Tampering Dataset[62].

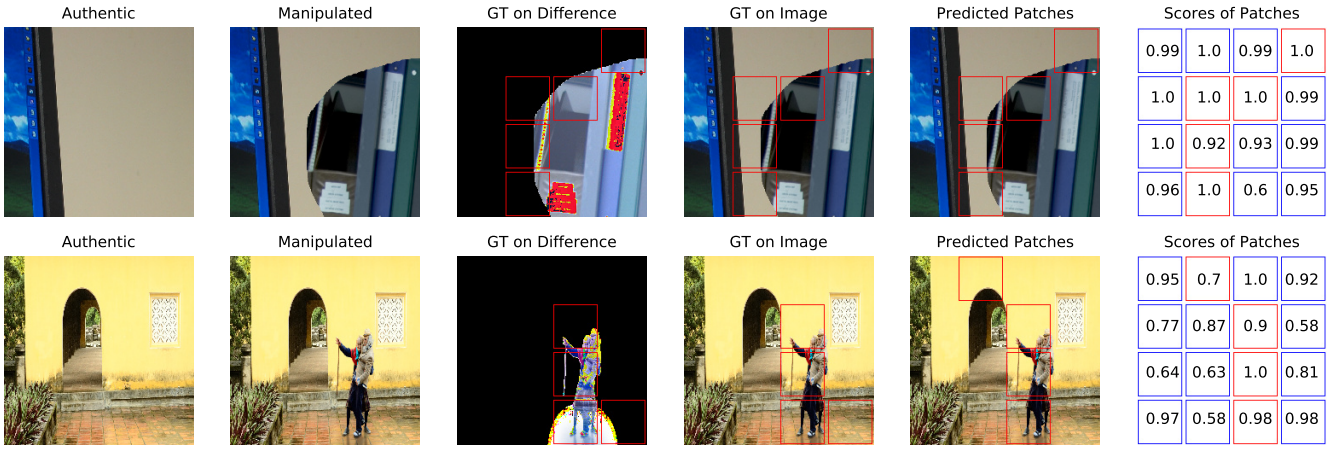

Figure 4.10.: Some detection results tested on both other data sets, trained on the splicing segmentation case. Images[12] are resized to 224×224. The columns show from left to right: The authentic image, followed by the tampered, their difference(tampered - authentic), marked best detectable ground truth on the difference, marked best ground truth on the tampered image, detection result and finally the predicted scores for each patch which leads to the result. The first row shows Columbia[61] data set and the second row the Real Tampering Dataset[62]. The ground truth we used here is the same as for training which is defined as at least one channel fulfills lower bound 0.05% and upper bound $0.08(1 - 0.08 = 0.92)\%$ of pixels which are manipulated. The gaps between marked patches are only to make them better visible.

[9] First row Columbia[61], second row Real Tampering Dataset[62].

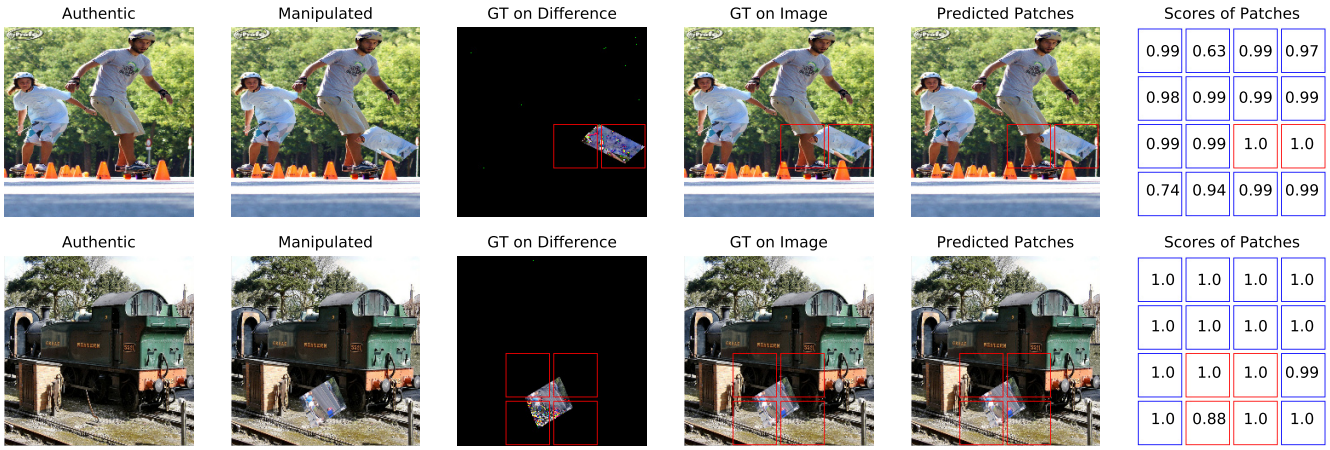

Figure 4.11.: Some detection results trained and tested on the rectangle case. Images[13] are resized to 224×224. The columns show from left to right: The authentic image, followed by the tampered, their difference(tampered - authentic), marked best detectable ground truth on the difference, marked best ground truth on the tampered image, detection result and finally the predicted scores for each patch which leads to the result. The first row shows the copy-move case and second row the splicing case. Both are trained and tested on their on data set. The ground truth we used here is the same as for training which is defined as at least one channel fulfills lower bound 0.05% and upper bound $0.08(1 - 0.08 = 0.92)\%$ of pixels which are manipulated. The gaps between marked patches are only to make them better visible.

[10] From COCO Image Database[60], images produced with the generator.

5 Discussion

In this chapter we summarize some insights which we observed during the work on this thesis. In the first section we have a look at what could be considered for manipulation and the ground truth definition. Then we show possible model architecture variations. Afterwards sampling techniques and parameters are investigated. Further we have a look at training parameters possibilities, consideration for a fair model comparison and possible interpretations of the results.

5.1 Manipulation and Ground Truth Definition

What is a good ground truth (GT) definition? Depending on the data sets and how clear the difference of the image pairs is, the GT has to be chosen wisely. We need at least three pixels to be manipulated (could be for all channels or just for one channel), because of the pixel differences. If there is no clear difference between the authentic and the tampered image, we need to make sure to mark the GT right. A better way could be to generate a GT for our generated data sets and use the GT from the other data sets if provided. If there is no GT provided, it could be still a good start to use the difference between tampered and the authentic images. Maybe this difference needs to be cleaned to get a neatly ground truth. How to clean the difference best, needs to be investigated further. The definition of the GT, at which point they should be detected, e.g., the smallest possible manipulation needs to be defined. If the manipulation is too small, it could be still seen as original with some failure. If it is too big, it can be seen as a new completely different image.

5.2 Model Architecture

The model architecture may not be best, there could be better architectures than ours. It should be tested, if for example a standard ResNet[34] or VGG-Net[57] architecture would work better, if we rescale each patch to the needed network input size. However, these architectures would lead to a bigger Neural Network too and may not improve the results.

The initialization of the Neural Network with currently Keras standard parameters might not be the best. It could be useful to try out others for example as suggested in [28], to use basic high-pass filter set.

The choice of the number of filters has also to be considered. If the network width(filters) is too big, each filter will not learn enough variation for itself. If we choose fewer filters, they can not learn all the possible variations needed to distinguish well. On the other side the depth of the network also learns more complexity the deeper it gets. If too deep we can not generalize well, if not deep enough, the needed details can not be learned to classify well.

There is also still a discussion if Batch Normalization[46] should be used before or after non-linearity. In the original paper it is used before, but some had better results with using it after[1].

For fighting overfitting we use Dropout[48] after each Max Pooling[47] to further improve generalization. Whether the position of Dropout[48] and the used values are optimal needs to be further investigated.

The choice of the patch-size seems the best for the current architecture. However, for example a smaller patch size provides better localization, but offers less information in each patch to learn from. A bigger patch size provides more information per patch, but the localization of the problem is worse than the current one. So the patch-size is a trade-off. If we change the size, the model architecture needs to be optimized to the new input size. Further patches could be extracted overlapping each other, which does not work well, because partly information could have two different labels tampered and original for the same pixels (information) we learn from.

The current approach only has a look at the spatial information at patch level, but it could be useful to consider the relation in one image between the patches too. For example in the copy-move problem, we only have a look at the differences in one patch and the relation of the same copied pixels don't play a role. To further improve the copy-move detection the relation between the patches(similarities) should be considered too.

[1] https://github.com/ducha-aiki/caffenet-benchmark/blob/master/batchnorm.md (visited on 06/03/19)

5.3 Over-sampling Methods & Parameters

The chosen sampling technique may not be the best. There are many others, for example ADASYN, and many variations of SMOTE[53] like SVM-SMOTE which have potential. Under-sampling, for example, could also work out well, which has to be tested like Random Undersampler, Cluster Centroids, Near Miss and many more. Another way of over-sampling which could lead to better results, is to over-sample tampered patches, to have more than authentic patches, to dominate the minority class. This technique should lead to a higher recall for the tampered class.

Not all batches, which are filled with image patches are fed to the network, only those patches where the batch is completely filled. This is the case, because of the variations of the data we over-sample from. This leads to bit loss of information for each bunch, which gets less over every epoch. The network always learns from different batches at every epoch, which leads to further generalization improvement. The reason for this is we over-sample the tampered patches on every bunch of images and randomly shuffle them to the batches. A bigger bunch size should also further improve the results in theory, because it leads to a bigger pool where we sample data from, which makes the samples more accurate.

5.4 Training Parameter

It seems that filtering-out the pairs, in which not even one patch is marked as tampered is not always the best. It depends on the data set and over-sampling algorithm, which impact it has on the results. The effect can be seen in Table 4.2. This should be tested individually for data set and sampling technique to find the best solution, i.e, whether we should train on all, tampered only or on the filtered-out version. The filtering also has an impact on how much data we use for training. It should help to generate more data to learn on more patches to further improve the generalization.

If we made the right decision for the upper/lower threshold, needs to be tested more intensively. These parameters should be optimal for each learning problem and data set. To find to optimal lower bound which helps to improve to detect smaller objects, but still makes sure that a detection is possible needs to be tested. For the upper bound we need to make sure that the object can fill as much as possible of the patch, but still makes it feasible to distinguishing between object and background for detection.

The parameters for Early Stop[59] seem to be the right one made on observations, but it could still improve, if we give it for example more epochs to improve results.

The batch size should be chosen as big that all needed information is in there to learn from, but not too big that the learn effect is too small. The size has an impact on the gradient and how it converges.

There is a potential that training the Neural Network for both problems (copy-move, splicing) at once could lead to general better results for both.

5.5 Model Comparison Parameters

The threshold for Splicebuster[18] has been chosen logically, but we could find the optimal threshold for each data set, which leads to the best F_1 score. Even better would be to find the best threshold which generalizes best for all data sets by testing each threshold step, which leads to a lot of calculation time. The same could be done for LRFfIMD[32] too, for the bounding box score threshold. We also have to consider at how many pixels in one patch it should be marked as tampered for example for Splicebuster[18]. Or how many points should in the patch to mark it as tampered in the SIFT[7] case. For LRFfIMD[32] how many percentage should be used for overlapping in to mark a patch as tampered.

Our approach would probably work better on the COVERAGE[56] and the Realistic Tampering Dataset[62] with retraining on some data of them. So the network learns more variations which it has not seen already. This should be the same for the splicing case on the Columbia[61] data set and the Realistic Tampering Dataset[62].

5.6 Interpretation of the Results

In Figure 4.6, the results of the copy-move data set are not as good as the splicing results. One explanation for that could be in the copy move case the objects are in general smaller to fulfill the condition of copy-move. In consequence there are less tampered patches, which leads to a smaller ratio balance and the over-sampling will have less variation. Another lead could be that, because of the filtering-out of not usable image pairs we get less training data to learn from as in the splicing case. This effect maybe also be the case for the rectangle data set, because fewer pairs get filtered-out than in the segmentation case. The rectangle splicing detection is also only slightly better than the copy-move one.

Splicing
In Table 4.3 our own approach works best for the COCO[60] segmentation splicing and for the Columbia[61] data set it works even better. The reason for this is that the data set provides clearer rounded edges, then our generated segmentation data set, which has more pixel like sharp edges. The results get even better for the rectangle data set as can ce seen in Table 4.4, because we only have straight lines in there, which are easier to distinguish, hence simpler to detect. They can be better learned by the NN, because the differences between the background and the rectangle are clearer than the complicated borders in the segmentation case. We also used more images for training, because there are not so many tiny or huge parts in a patch, which then get filtered over the threshold. This filtering has an impact on the detection results of the testing data sets which are not as good as in the segmentation case. In the Realistic Tampering Dataset[62] the results are not the best, because it is a mixture of all three mentioned manipulations. We are not able to detect removal or copy-move at all here. Overall data sets our approach works best.

Copy-Move
In Table 4.5 and Table 4.6 using at least three points in one patch for SIFT[7] to mark it as tampered performs better than just using one point, because we get rid of some outliers. Our model only performs better than SIFT[7] for the COCO segmentation and rectangle data set, because the copied areas are too small or just plain which makes it impossible to detect feature points on the basis of edges and corners for SIFT[7]. SIFT[7] works best, because it finds similarities in the image, which helps in the copy-move case. Our approach and LRFfIMD[32] learn differences between the copied part and the background instead. On COVERAGE[56] and the Realistic Tampering Dataset[62] SIFT[7] works better, because if the image is a copy-move case there are mostly detectable features in the manipulated part of the image.

6 Conclusion & Future Work

6.1 Conclusion

In this thesis we proposed an approach that does passive blind image manipulation detection without prior knowledge for copy-move and splicing attacks. The approach also localizes the manipulations in the image. We compared our approach with other models to show the potential of detection.

We showed that the proposed patch-based approach is capable of learning differences between authentic and tampered regions. Its architecture is a partly adapted combination of the VGG-Net and Global Average Pooling. As a result of the approach's architecture, it uses only $\sim 650k$ trainable parameters in the Neural Network (NN) without any pretraining. This few parameters leads to less training steps and faster convergence, compared to other approaches used in this thesis. In particular, our results can keep up with LRFfIMD, which uses a pretrained NN with $\sim 90m$ trainable parameters using the same number of epochs.

One advantage is our NN is small to the compared others. Also, using patches for localization works well, even if it could be more precise. Further, choosing the right patch-size and the architecture which fits to it is challenging as well. On the other hand, what is below or over the threshold is nearly undetectable, because the patches don't get labeled as tampered for training. So in these cases there is no ground truth where the network could learn from. In general, too small manipulations or if nearly the whole patch has been changed are undetectable.

We also get a highly imbalanced data set, because of the patch-based labeling process of the ground truth, which leads to learning problems. A Neural Network mostly can't learn from a highly imbalanced data set, it will only learn the majority class, in our case the authentic patches. This imbalance needs to be compensated by a sampling technique to bring the data set in balance.

Furthermore, we only consider very local spatial information concerning the patches, so we look at less information at once, as we would use the whole image. This can be both an advantage and disadvantage as the same time. The model learns more detail for a smaller area, which can lead to better results, but it also only considers the patch itself, not the whole image.

Some experiments have been carried out on the copy-move and the splicing attack detection with two different data set approaches for training, segmentation and rectangle. Both training approaches show potential, but the more complex segmentation leads to a more sophisticated learned NN which indicates a better overall detection potential. The detection has been tested on several data sets, including our generated.
The results show, we can detect the manipulation attacks well to the compared models. While the splicing detection works well, there is potential that copy-move detection can achieve this results too.

Overall, we showed that our approach led to a significantly smaller model, while maintaining a good F_1 score for the splicing attack.

6.2 Future Work

The current approach learns the difference between authentic and tampered regions over hard edges. Too further improve detection results and robustness it makes sense to randomly use for example a gaussian filter on the tampered patches, to make the edges more smoothly. This will make it harder for the Neural Network to learn the differences. Furthermore, techniques like flipping, etc. could improve the generalization too.

The chosen sampling technique may not be the best, there are many other sampling techniques, which could lead to further improved detection results. Another shoot could be to train Generative Adversarial Networks (GAN) on tampered patches to be able to sample them for training.

We could also use other channels than RGB to learn from, which have been shown in some papers can work out well, like Black and White (B/W) or a new Color Space[12] which is used in [11] or SRM[29] from [32]. These channel streams could then also be combined as done in [32] to improve results.

Bibliography

[1] T. Qiao, *Statistical detection for digital image forensics*. PhD thesis, Troyes, 2016.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[3] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[4] W. Luo, J. Huang, and G. Qiu, "Robust detection of region-duplication forgery in digital image," in *Proceedings of the 18th International Conference on Pattern Recognition-Volume 04*, pp. 746–749, IEEE Computer Society, 2006.

[5] W. Li and N. Yu, "Rotation robust detection of copy-move forgery.," in *ICIP*, pp. 2113–2116, Citeseer, 2010.

[6] S.-J. Ryu, M. Kirchner, M.-J. Lee, and H.-K. Lee, "Rotation invariant localization of duplicated image regions based on zernike moments," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 8, pp. 1355–1370, 2013.

[7] I. Amerini, L. Ballan, R. Caldelli, A. Del Bimbo, and G. Serra, "A sift-based forensic method for copy–move attack detection and transformation recovery," *IEEE transactions on information forensics and security*, vol. 6, no. 3, pp. 1099–1110, 2011.

[8] J. Li, X. Li, B. Yang, and X. Sun, "Segmentation-based image copy-move forgery detection scheme," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 3, pp. 507–518, 2015.

[9] Y. Li and J. Zhou, "Image copy-move forgery detection using hierarchical feature point matching," in *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2016 Asia-Pacific*, pp. 1–4, IEEE, 2016.

[10] B. Shivakumar and S. S. Baboo, "Detection of region duplication forgery in digital images using surf," *International Journal of Computer Science Issues (IJCSI)*, vol. 8, no. 4, p. 199, 2011.

[11] X.-Y. Wang, L.-X. Jiao, X.-B. Wang, H.-Y. Yang, and P.-P. Niu, "A new keypoint-based copy-move forgery detection for color image," *Applied Intelligence*, vol. 48, no. 10, pp. 3630–3652, 2018.

[12] J.-M. Geusebroek, R. Van den Boomgaard, A. W. M. Smeulders, and H. Geerts, "Color invariance," *IEEE Transactions on Pattern analysis and machine intelligence*, vol. 23, no. 12, pp. 1338–1350, 2001.

[13] X.-y. Wang, W.-y. Li, H.-y. Yang, P. Wang, and Y.-w. Li, "Quaternion polar complex exponential transform for invariant color image description," *Applied Mathematics and Computation*, vol. 256, pp. 951–967, 2015.

[14] M. K. Johnson and H. Farid, "Exposing digital forgeries by detecting inconsistencies in lighting," in *Proceedings of the 7th workshop on Multimedia and security*, pp. 1–10, ACM, 2005.

[15] M. K. Johnson and H. Farid, "Exposing digital forgeries in complex lighting environments," *IEEE Transactions on Information Forensics and Security*, vol. 2, no. 3, pp. 450–461, 2007.

[16] M. C. Stamm and K. R. Liu, "Forensic detection of image manipulation using statistical intrinsic fingerprints," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 3, pp. 492–506, 2010.

[17] W. Wei, S. Wang, X. Zhang, and Z. Tang, "Estimation of image rotation angle using interpolation-related spectral signatures with application to blind detection of image forgery," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 3, pp. 507–517, 2010.

[18] D. Cozzolino, G. Poggi, and L. Verdoliva, "Splicebuster: A new blind image splicing detector," in *2015 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, IEEE, 2015.

[19] J. Chen, X. Kang, Y. Liu, and Z. J. Wang, "Median filtering forensics based on convolutional neural networks," *IEEE Signal Processing Letters*, vol. 22, no. 11, pp. 1849–1853, 2015.

[20] D. Cozzolino and L. Verdoliva, "Single-image splicing localization through autoencoder-based anomaly detection," in *2016 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, IEEE, 2016.

[21] N. Huang, J. He, and N. Zhu, "A novel method for detecting image forgery based on convolutional neural network," in *2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*, pp. 1702–1705, IEEE, 2018.

[22] M. Huh, A. Liu, A. Owens, and A. A. Efros, "Fighting fake news: Image splice detection via learned self-consistency," *arXiv preprint arXiv:1805.04096*, 2018.

[23] M. K. Johnson and H. Farid, "Exposing digital forgeries through chromatic aberration," in *Proceedings of the 8th workshop on Multimedia and security*, pp. 48–55, ACM, 2006.

[24] I. Yerushalmy and H. Hel-Or, "Digital image forgery detection based on lens and sensor aberration," *International journal of computer vision*, vol. 92, no. 1, pp. 71–91, 2011.

[25] G. Chierchia, G. Poggi, C. Sansone, and L. Verdoliva, "A bayesian-mrf approach for prnu-based image forgery detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 4, pp. 554–567, 2014.

[26] J. F. O'Brien and H. Farid, "Exposing photo manipulation with inconsistent reflections.," *ACM Trans. Graph.*, vol. 31, no. 1, pp. 4–1, 2012.

[27] A. E. Dirik and N. Memon, "Image tamper detection based on demosaicing artifacts," in *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pp. 1497–1500, IEEE, 2009.

[28] Y. Rao and J. Ni, "A deep learning approach to detection of splicing and copy-move forgeries in images," in *Information Forensics and Security (WIFS), 2016 IEEE International Workshop on*, pp. 1–6, IEEE, 2016.

[29] J. Fridrich and J. Kodovsky, "Rich models for steganalysis of digital images," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 3, pp. 868–882, 2012.

[30] J. Bunk, J. H. Bappy, T. M. Mohammed, L. Nataraj, A. Flenner, B. Manjunath, S. Chandrasekaran, A. K. Roy-Chowdhury, and L. Peterson, "Detection and localization of image forgeries using resampling features and deep learning," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pp. 1881–1889, IEEE, 2017.

[31] M. Goljan, J. Fridrich, and M. Kirchner, "Image manipulation detection using sensor linear pattern," *Electronic Imaging*, vol. 2018, no. 7, pp. 1–10, 2018.

[32] P. Zhou, X. Han, V. I. Morariu, and L. S. Davis, "Learning rich features for image manipulation detection," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1053–1061, IEEE, 2018.

[33] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[35] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear cnn models for fine-grained visual recognition," in *Proceedings of the IEEE international conference on computer vision*, pp. 1449–1457, 2015.

[36] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[37] M. L. Minsky and S. Papert, "Perceptrons: an introduction to computational geometry," 1969.

[38] M. Minsky and S. A. Papert, *Perceptrons: An introduction to computational geometry*. MIT press, 2017.

[39] Y. Bengio *et al.*, "Learning deep architectures for ai," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[40] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," tech. rep., California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[42] A. Zell, *Simulation neuronaler netze*, vol. 1. Addison-Wesley Bonn, 1994.

[43] D. E. Rumelhart, G. E. Hinton, R. J. Williams, *et al.*, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, p. 1, 1988.

[44] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[45] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[46] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning-Volume 37*, pp. 448–456, JMLR. org, 2015.

[47] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *International conference on artificial neural networks*, pp. 92–101, Springer, 2010.

[48] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[49] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[50] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[51] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[52] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[53] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[54] H. Han, W.-Y. Wang, and B.-H. Mao, "Borderline-smote: a new over-sampling method in imbalanced data sets learning," in *International conference on intelligent computing*, pp. 878–887, Springer, 2005.

[55] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[56] B. Wen, Y. Zhu, R. Subramanian, T.-T. Ng, X. Shen, and S. Winkler, "Coverage—a novel database for copy-move forgery detection," in *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 161–165, IEEE, 2016.

[57] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[58] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[59] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.

[60] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*, pp. 740–755, Springer, 2014.

[61] Y.-F. Hsu and S.-F. Chang, "Detecting image splicing using geometry invariants and camera characteristics consistency," in *2006 IEEE International Conference on Multimedia and Expo*, pp. 549–552, IEEE, 2006.

[62] P. Korus and J. Huang, "Multi-scale analysis strategies in prnu-based tampering localization," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 4, pp. 809–824, 2017.

[63] D. G. Lowe, "Object recognition from local scale-invariant features," in *iccv*, p. 1150, Ieee, 1999.

[64] S. Haykin, *Neural networks*, vol. 2. Prentice hall New York, 1994.

A Some Appendix

Used Hardware for calculations:

- PC: Intel(R) Core(TM) i3-7100 CPU @ 3.90GHz, 32 GB RAM, Geforce GTX 1080 8GB RAM
- PC: AMD Ryzen 7 1700X Eight-Core Processor 3.40GHz, 32 GB RAM, Geforce GTX 1070 8GB RAM

A.1 Implementation Details

A.1.1 Our Approach

Training:

- EarlyStop on val_f1_score 0.001 delta, patience 5, mode max, restore best weights, max number of epochs 200
- Adam learning rate 0.001
- categorical_crossentropy with one hot encoding
- F_1 score measurement
- batch size of 256
- bunch size for 4992 for RandomOverSampling and 32 for Borderline-SMOTE. Have to be a number which fulfills: All images have to fit into memory, bunch size * #patches mod batch size = 0, so there will be no remainder after devision, so all data fits into the batches by taken steps
- bunch size validation 96, number is chosen on the condition, number * #patches mod batch size = 0
- #training steps ((#patches labeled normal after filtered training images) * 2) / batch size # because this is the number we reach after over-sampling
- #validation steps (#images * #patches) / batch size

Important used packages:

- python 3.6.6
- keras 2.2.4 with tensorflow-gpu 1.12.0 backend
- opencv-python 3.4.4.19
- scipy 1.1.0
- scikit-image 0.14.1
- scikit-learn 0.20.1
- imbalanced-learn 0.4.3

A.1.2 Splicebuster

Important used packages:

- python 3.6.6
- scipy 0.18.1
- scikit-image 0.12.3
- scikit-learn 0.17
- opencv-python 4.0.0.21

A.1.3 SIFT

Important used packages:

- python 3.6.6
- scipy 1.2.1
- scikit-learn 0.20.2
- opencv-python 3.3.0.10, SIFT still works here
- opencv-contrib-python 3.3.0.10, SIFT still works here

A.1.4 Learning Rich Features for Image Manipulation Detection (LRFfIMD)

Training:

- To make the code work on a normal GPU, we added GPU memory fraction 0.8 to reduce the used memory.
- We use "," as separator instead of " ", because the category names can have a space.

Important used packages:

- python 3.5.2
- tensorflow-gpu 0.12.1
- scipy 1.2.0
- scikit-image 0.14.2
- opencv-python 4.0.0.21
- Cython 0.29.3
- PyYAML 3.13
- pycocotools 2.0
- image 1.5.27
- Pillow 5.4.1