Spring 2020

Image Forgery Detection with Machine Learning

Lubna Alzamil
Central Washington University, lubna.alzamil@cwu.edu

IMAGE FORGERY DETECTION WITH MACHINE LEARNING

A Thesis

Presented to

The Graduate Faculty

Central Washington University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

Computational Science

by

Lubna Alzamil

June 2020

CENTRAL WASHINGTON UNIVERSITY

Graduate Studies

We hereby approve the thesis of

Lubna Alzamil

Candidate for the degree of Master of Science

APPROVED FOR THE GRADUATE FACULTY

______________ ______________________________

Dr. Razvan Andonie

______________ ______________________________

Dr. Boris Kovalerchuk

______________ ______________________________

Dr. Szilard Vajda

______________ ______________________________

Dean of Graduate Studies

ABSTRACT

IMAGE FORGERY DETECTION WITH MACHINE LEARNING

by

Lubna Alzamil

June 2020

The issue of forged images is currently a global issue that spreads mainly via social networks. Image forgery has weakened Internet users confidence in digital images. In recent years, extensive research has been devoted to the development of new techniques to combat various image forgery attacks. Detecting fake images prevents counterfeit photos from being used to deceive or cause harm to others. In this thesis, we propose methods using the error level analysis algorithm to detect manipulated images. We show that our combination of image pre-processing and machine learning techniques is an efficient approach to detecting image forgery attacks.

ACKNOWLEDGMENTS

I want to express my gratitude to my advisor, Dr. Razvan Andonie, for his guidance, support, and participation in the success of this thesis through all the steps involved. I also want to thank my committee members: Dr. Szilard Vajda and Dr. Boris Kovalerchuk for generously giving their time, support, advice, and goodwill during this document's review.

TABLE OF CONTENTS

Chapter Page

TABLE OF CONTENTS (CONTINUED)

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

With the advent of digital cameras and other smart devices, it has become easy for anyone to manipulate an image. Some manipulations are not harmful, such as changing the brightness of an image or converting it to black and white. On the other hand, some manipulations can cause harm to others and defame them, especially politicians and celebrities.

Image forgery is the process of manipulating a digital image to hide valuable or essential content or to force the viewer to believe an idea [1]. It has been defined as the process of manipulating an original digital image to either conceal its original identity or create an entirely different image than what was originally intended by the user of the digital platform [2]. Forged images can cause disappointment and emotional distress and affect public sentiment and behavior [3]. Images can transmit much more information than text. People tend to believe what they can see, and this affects their judgment, which leads to a series of unwanted responses. Because fabrications have become widespread, the urgency to detect forgeries has significantly increased. The copymove approach is one of the most widely used forgery techniques. It copies a part of the image and pastes it onto another part of the image. The technique itself is not harmful, but it can lead to critical situations if someone uses it with malicious intent.

Image forgery is done mainly for malicious reasons. It serves to distort information, spread immorality and fake news, obtain money fraudulently from an unsuspecting audience, ruin the reputation of a popular celebrity or any other public figure, and spread adverse political influence among the users of a digital platform. Therefore, clear authentication of images and videos uploaded to digital media platforms, before they are used in any way, makes it more difficult for digital information users to share information [4]. Image forgery is often used by malicious people to ruin the reputation of public figures. Image forgery, especially through Photoshop, can be used to display unethical behavior in public figures. It is also sometimes an attempt to influence consumers of the goods produced or the services offered by these public figures to shift to other markets [5]. This forgery could also be used for political reasons against opponent politicians or their agents, spreading images that portray their immoral side. This aims to convey a message to the public regarding the lack of integrity of the subject. Image forgery often leads to emotional problems for those whose images are released to public websites in disregard for their privacy. There have been reports of suicide due to the leaking of private images to public digital platforms, after which the victims undergo significant rebuke. These deaths negatively affect society.

Image forgery is also sometimes used to cheat victims of their money in increasingly common fraud schemes. The forged images are uploaded with embedded text, purportedly from the owner of the original image, with instructions that end in innocent people losing money. This is also done with images portraying people who are in dire need of help, with intentions of fraudulently acquiring money from unsuspecting members of the public. Society then ceases helping even those who are in genuine need of help because of the fear of being swindled.

For all of these reasons, it is vital to develop methods of detecting whether an image is forged and to locate the region of manipulation.

Related Work

Scientific studies on image forgery have provided various approaches to detecting whether an image is manipulated. In [6], Bunk et al. discuss how resampling is a vital signature of manipulated images. They proposed a method of detecting and locating the area of manipulation on an image using resampling detectors with deep learning classifiers. In [7], Abdalla et al. offered an approach to classifying whether an image is forged that involved transfer learning. Bappy et al. [8] applied a long-short term memory network with encoderdecoder architecture to detect and localize the area of manipulation. Wang et al. [9] employed a feature pyramid network based on ResNet with Mask R-CNN to identify and locate manipulated regions.

In [10], Rahmouni et al. classified an image by dividing it into 100 x 100 patches and passing these patches into a patch classifier (image pre-processing). They subsequently trained the resulting complete images with a convolutional neural network (CNN). In [11], Kaur and Manro pre-processed the images first. They then altered the images to grayscale space and performed Gaussian pyramid decomposition from time to time. Afterward, they detected image forgery using the block-based approach. Amit Doegar [12] proposed a method involving AlexNets with support vector machines (SVMs) to classify whether an image was forged without specifying the exact area of manipulation. Zhou et al. [13] employed a pre-trained model, VGG16, with a steganalysis rich model and CNNs.

In [14], Gupta et al. examined several block-matching algorithms, such as exact match and robust match, and compared their performances. In [15], Shivakumar and Baboo proposed an approach using the speeded-up robust features algorithm in parallel with the K-dimensional tree algorithm to identify the manipulated region. Salloum et al. [16] proposed using fully convolutional networks. In the beginning, they tried using single-task fully convolutional networks. However, they noticed that multi-task fully convolutional networks obtained better results than single-task fully convolutional networks. The Pham et al. [17] segmented the manipulated images into spliced areas and background areas in the manipulation-detection stage before the image-redemption stage to improve the accuracy of the redemption. They suggested a hybrid approach that could easily retrieve images using Zernike moment features and features found by a scale-invariant feature transform.

Contributions

At the beginning, I began working on the error level analysis (ELA) algorithm with various machine learning classifiers. A detailed discussion of the algorithm and the classifiers follow in Chapter II, Chapter III, and Chapter IV. I continued researching this area until I realized that the ELA algorithm is not the best way to detect image forgery. I discovered that image pre-processing techniques, discussed in Chapter II and Chapter V, obtain more accurate and promising results. I used the same pre-processing technique presented in [10]. The authors of [10] divided each image into 100 x 100 patches and passed them to a patch classifier that classifies whether the patch belongs to a raw image or a computer graphics image. They passed the resulting images to a CNN to predict the result. I used the same patch classifier technique, but instead of passing the resulting images into a standard CNN, I passed them to a VGG16 pre-trained model. The authors of [10] achieved high accuracy, 93.4%, with their model, but I obtained a higher accuracy, 94.5%, using VGG16.

CHAPTER II

THEORETICAL BACKGROUND

Convolutional Neural Networks

Convolutional Neural Networks (CNNs or ConvNets) are a type of neural network that are used effectively in image recognition classification and applications. Specifically, these neural networks are effective in facial, traffic-sign, and object identification. They also help power vision in robots and remote-controlled cars (self-driving) [18]. They evolved from the LeNet architecture, which was the initial CNN that was useful in the development of deep learning [19]. There are four operations that form the foundation of every CNN:

1. **An image composed of a matrix of pixel values**

   From computer graphics concepts, every image can be represented as pixel value matrices. Certain components of an image are referred to as channels. In digital camera images, three channels are present: red, green, and blue [20]. These three channels are stacked in layers in the form of 2-dimensional matrices, and each channel has a pixel value that is in the range of 0 to 255. Grayscale images are distinguished by the presence of only one channel.

2. **The convolution step**

   The convolution step entails extracting features from the input image. This operation maintains the patterns between the pixels with the help of small squares of input data that learn the features of the image. The operation involves sliding one channel matrix, such as orange, by one pixel over the original image, which could be green. This sliding step is referred to as a stride. Element-wise multiplication is performed for every position. Finally, these products are added to generate the final integer that represents a single element of the desired output matrix, such as pink. The operations in this block involve a filter and a convolved feature, which interact on the input image to detect features from it.

3. **Non-linearity (rectified linear unity [ReLU])**

   After every convolution operation, the ReLu involves a non-linear operation. The ReLu operates on every pixel and replaces all negative pixel values in the feature map with zero. The ReLu aims to introduce non-linearity to the CNN, as almost every datum learned in the CNN has a linear property.

4. **The pooling step**

   The pooling step retains the most significant information of the feature map while reducing the dimensionality of every feature through spatial pooling [21]. The pooling step involves pooling of different types such as average, sum, and max. For example, if the operation involves max pooling, then the spatial neighborhood must be defined, and subsequently, the largest element from the rectified feature map within the window is selected [21]. The operation uses the average instead of the largest element for average pooling and the sum for sum pooling. Therefore, pooling, convolution, and ReLU are the foundation blocks for the effective implementation of CNN.

Figure 1 shows our model architecture of the ELA algorithm as a pre-processing step and the passing of the resulting images into a CNN. The algorithm is discussed in Chapter III.

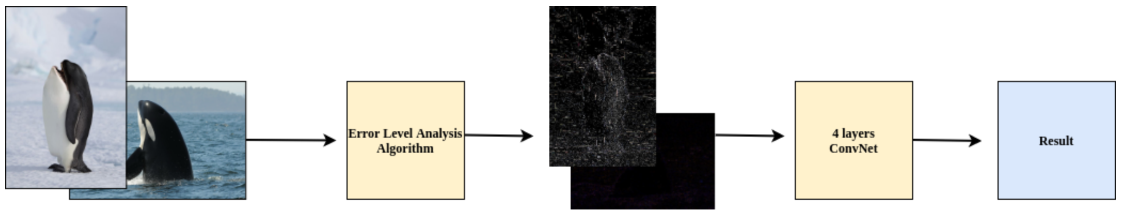


FIGURE 1: Error Level Analysis model architecture

Support Vector Machines

The Support Vector Machine (SVM) [22] is a model used in classification and regression. It can solve linear and non-linear problems and performs well on various practical challenges. The SVM algorithm generates a hyperplane that divides the data into categories [23]. It is best applied in regression and classification problems, and it produces the highest accuracy while using less computational power [24]. This algorithm can be applied in classification, where the hyperplane in an N-dimensional space classifies distinct data points. The SVM is classified as a supervised machine learning model. It categorizes sets of training data into one or two other categories, and then a training algorithm model is built to assign the categories to their respective groups. The SVM is a non-probabilistic binary linear classifier that employs methods such as Platts scaling.

When working with textual analysis classification tasks, the SVM process involves refining training data while employing other forms such as Naive Bayes algorithms. A confidence score is generated for each recognized text or digit. When confidence is achieved in the dataset, the SVM continues the classification by applying a classification algorithm that is suitable when in situations with limited data. The algorithm involves separating two classes of data points with various choices of hyperplane. The SVM focuses on finding the plane with the maximum margin that represents the distance between two data points in both classes. Classifying future data points becomes effective with reinforcement from the maximization of the margin distance.

In SVMs, hyperplanes represent decision boundaries that are essential for data point classification. The allocation of a data point to a certain class is based on the side of the hyperplane that the point falls on. There are various features that form the basis of the dimension of the hyperplane. Data points at the hyperplane determine the orientation of the hyperplane to define support vectors. The margin of the classifier is maximized through the support vectors. Support vector machines use the kernel trick to perform linear classification while implicitly mapping inputs into feature spaces of high dimension. The main goal of the SVM is to help classify data in most of the statistical problems presented to machine learning experts. Understanding the correct position of data points on the hyperplane makes it possible to apply the SVM effectively.

Support vector machines provide solutions to real problems in a wide range of applications. A main application is text and hypertext categorization, which reduces the requirement for labeled training in transductive and inductive settings. The classification of images is another major area employing SVMs. The SVM is believed to achieve the highest search accuracy compared to traditional query refinement techniques [25].

Random Forests

A random forest (RF) [26] is an ensemble algorithm, meaning that a decision is made using the results from various models. In most cases, the outcome from an ensemble model is better than that of any individual model [27]. Several decision trees are generated by RFs, and the decision is determined based on the outcome of all the decision trees [28]. An RF is a learning algorithm that randomly collects decision trees. Each decision tree consists of several decisions, and a combination of them forms the

RF [29]. An RF integrates a collection of decision models from individual decision trees in the forest to improve the accuracy of the results. This prevents relying on a single learning model from a single decision tree in the RF. The merging of individual decision trees, each with its own set of algorithms constituting the RF, therefore creates a classification algorithm. This classification algorithm is independent from the algorithms of the individual decision trees. This forms a basis of prediction that is more accurate than that prediction that could have been made by a single decision tree or by a combination of independent decision trees.

Decision trees are the building blocks of an RF, and the individual trees are used to differentiate various events based on their most unique aspects [30]. An RF consists of many trees that make the work more straightforward when complex sets of data are involved. They work on the principle that many uncorrelated decision trees, if made to operate as a group, will yield clearer and more accurate results than any individual decision tree. This is possible because the decision trees protect each other from the independent errors they make [31]. For an RF to work effectively, the decisions made by the individual decision trees should have little or no correlation with each other. There should also be unique signs in the differentiating features so that the models perform better than a random guess.

Random forests utilize the idea of bagging, a process that allows decision trees to sample from the dataset while making replacements, which result in different trees [32]. This is possible because individual decision trees are highly sensitive to the data that they are trained on. Bagging therefore produces results where the individual trees not only train on different sets of data but can also use different characteristics to make informed decisions.

Image Pre-Processing

Image pre-processing is the process of improving image data by performing various operations on images and suppressing unwanted distortions in them. It can also be used to enhance certain unique features in an image that are crucial to further processing. Image processing may be a basic task, such as resizing [33]. For example, to feed an image dataset into a deep learning model, all images must be of the same size. Other pre-processing tasks include geometric and color conversion, or the transition of color to grayscale; standardization; and data augmentation [34].

In this thesis, I used the pre-processing technique presented in [10]. The pre-processing step takes every image in the dataset and divides it into 100 x 100 patches. Subsequently, the patches are passed into a CNN classifier that classifies whether the given patch belongs to a raw image (green) or a computer graphics image (red). After the classification of patches, the complete images are generated again from the classified patches. The authors of [10] passed the resulting images into another CNN to classify whether an image is spliced. I used the VGG16 pre-trained model instead to accomplish better accuracy than a standard CNN. Figure 2 shows our model architecture. The pre-processing subfigure is from [10], and the VGG16 subfigure is from [35]. The datasets and experiments are discussed in Chapter V.

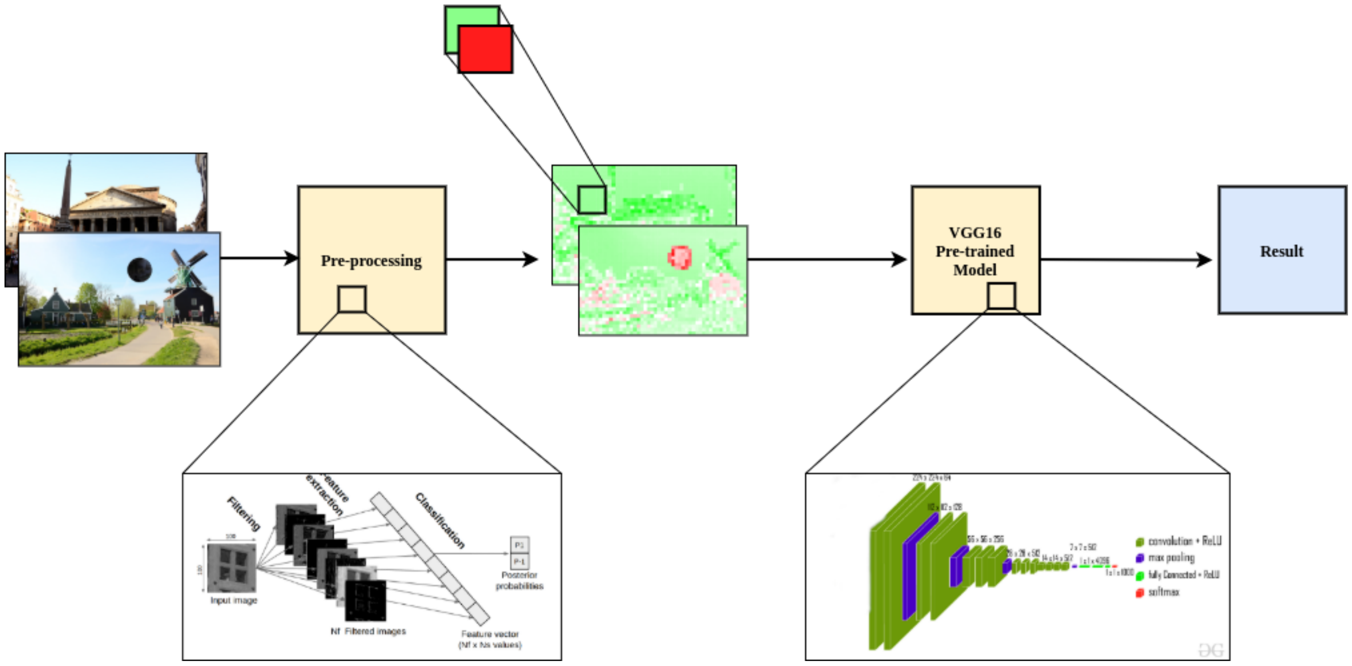


FIGURE 2: Model architecture

Pre-Trained ImageNet Models and Transfer Learning

A pre-trained ImageNet model is a model that has been trained on a significantly large dataset to solve a problem that is similar to the problem I want the model to solve [36]. I used a model pre-trained for a certain task on the ImageNet dataset. The initial training of the model could have been done on a similar or very different domain, but the ability to solve problems remains useful [37]. Studies on modern computer vision have revealed that models that perform better on ImageNet usually perform better on other vision tasks as well. It is common practice to use imported models, such as MobileNets or VGG, due to the relatively high costs involved in training these models from scratch. The task of importing, usually referred to as transfer learning, is not only effective but also cost friendly to any profit-making institution. Transfer learning is also common because pre-training a model requires a relatively large dataset for the model to extract the characteristics required for the given task [19]. For instance, ImageNet contains over one million images in 1000 categories [38]. A lack of data makes it difficult to train a model from scratch and makes it necessary to import a pre-trained model. Another reason for importing a pre-trained model instead of training one from scratch is the time required to train a model from scratch, depending on the experience of the trainer. This is because one must do many calculations and experiments before discovering a CNN architecture. Pre-trained models are also commonly used because training a model from scratch requires specific computational resources that might not be available. This also makes it necessary to import a pre-trained model. A pre-trained model that is imported is usually more efficient than a model that can be trained from scratch [39]. Pre-trained models are more accurate in most cases because they have been trained on a large number of classes, such as the 1000 classes in ImageNet. Employing a pre-trained model enhances their suitability to work on a wider range of issues compared to a model that is trained from scratch. Importing a pre-trained model is also advantageous because the most complex work of optimizing the parameters has usually been completed; what remains is only fine-tuning the model, a process that involves adjusting the hyperparameters to improve the pre-trained model. Another advantage of the pre-trained model is that it uses fewer steps before the convergence of the output [39]. This is because for a classification task, the features to be extracted will be similar, and it thus requires less time. Prior to choosing to import a pre-trained model, thorough research must be conducted on the problem in question, and the keywords should be determined based on the type of dataset to be used. This is because, depending on the complexity of the dataset, some models usually work better than others. VGG16 is a CNN model that is used in large-scale image recognition. It provides high accuracy testing using ImageNet, which consists of 100 classes of 14 million images. The model comprises 16 layers with weights, indicated by the value 16 in VGG16. Figure 3 shows sample VGG16 architecture from [40].

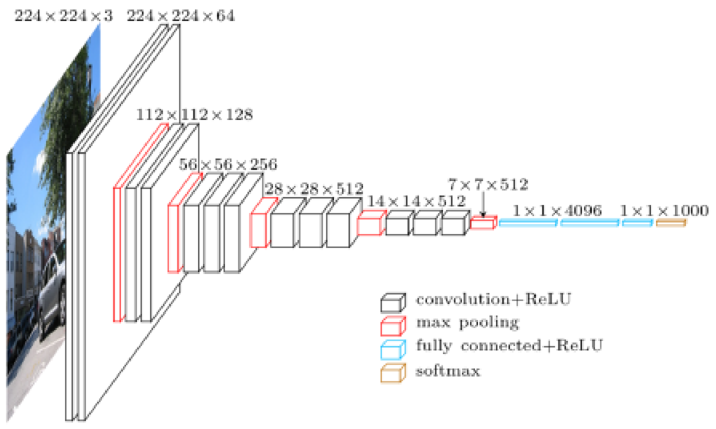


FIGURE 3: VGG16 pre-trained model architecture

The input to the first convolutional layer of the VGG16 model is an RGB image of fixed-size, 224 x 224. The image is passed through a stack of convolutional layers that represent the use of a receptive field of size 3x3, the smallest size that can represent bottom, up, left, right, and center. The VGG16 also makes use of 1 x 1 convolution filters in several of its configurations. This configuration is the transformation of linear input channels, which is followed by non-linearity. In this model, the convolutional stride has a fixed value of 1 pixel. Therefore, any spatial padding of the convolutional layer input must be preserved during convolution. A stack of convolutional layers forms the foundation for fully connected layers. However, the stack is different from the fully connected layers in architecture and depth. Each of the top two stacks comprises 4096 channels, and the third layer can perform a 1000-way ImageNet large-scale visual recognition challenge classification, giving it 1000 channels. The soft max layer forms the final layer that contributes to the configuration of the fully connected layers in the networks [41]. In the presence of hidden layers in pre-trained models, the ReLU non-linearity is the basic option that is applied. One of the challenges facing this model is the need for increased memory due to a high consumption of space. The VGG16 is a responsible model that assists machine learning experts in applying pre-trained networks to improve the level of learning [42].

CHAPTER III

ERROR LEVEL ANALYSIS ALGORITHM

Error Level Analysis (ELA) is a tool for exposing fabricated regions in JPEG images [41]. It can help recognize manipulations of compressed (JPEG) images by detecting the noise distribution present after resaving the image at a particular compression rate.

Overview

Neal Krawetz invented the concept of Error Level Analysis for images when he noticed how errors spread when a JPEG image is saved [43][44]. When cutting out a section of an image and pasting it into another image, the ELA for the pasted section often detects a more significant error, which means it is brighter than the rest of the image. There are several implementations of this algorithm, but they all follow the same steps.

The Algorithm

1. Compress the input image with a given compression rate and save it as a new image.
2. Calculate the pixel-by-pixel difference between the original image and the new image.
3. Store the difference in variable *elaImg*.
4. Compute the minimum and maximum pixel values for each band in the image and store them in variable *extrema*.

5. Compute the maximum pixel in *extrema* and store it in variable *max*.
6. Calculate the new scale by dividing 255 by the *max*.
7. Enhance the brightness of *elaImg* based on the resulting scale, then save and return the resulting ELA image.

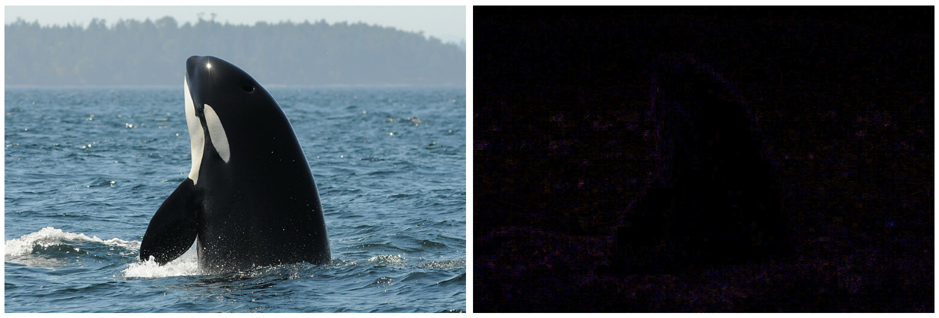
Figure 4 illustrates our Python implementation of this algorithm.

```
import os
from PIL import Image, ImageChops, ImageEnhance

def ToEla(path, quality):
    filename = path
    resavedImageName =filename.split(’.’)[0]+’.resaved.jpg’
    EalImageName = filename.split(’.’)[0] + ’.ela.png’

    img = Image.open(filename).convert(’RGB’)
    img.save(resavedImageName, ’JPEG’, quality=quality)
    imgResaved = Image.open(resavedImageName)

    elaImg = ImageChops.difference(img, imgResaved)

    extrema = elaImg.getextrema()
    diff = max([ex[1] for ex in extrema])
    if diff == 0:
    diff = 1
    scale = 255.0 / diff

    elaImg = ImageEnhance.Brightness(elaImg).enhance(scale)
    os.remove(resavedImageName)
    return elaImg
```

FIGURE 4: Error level analysis algorithm: Python implementation

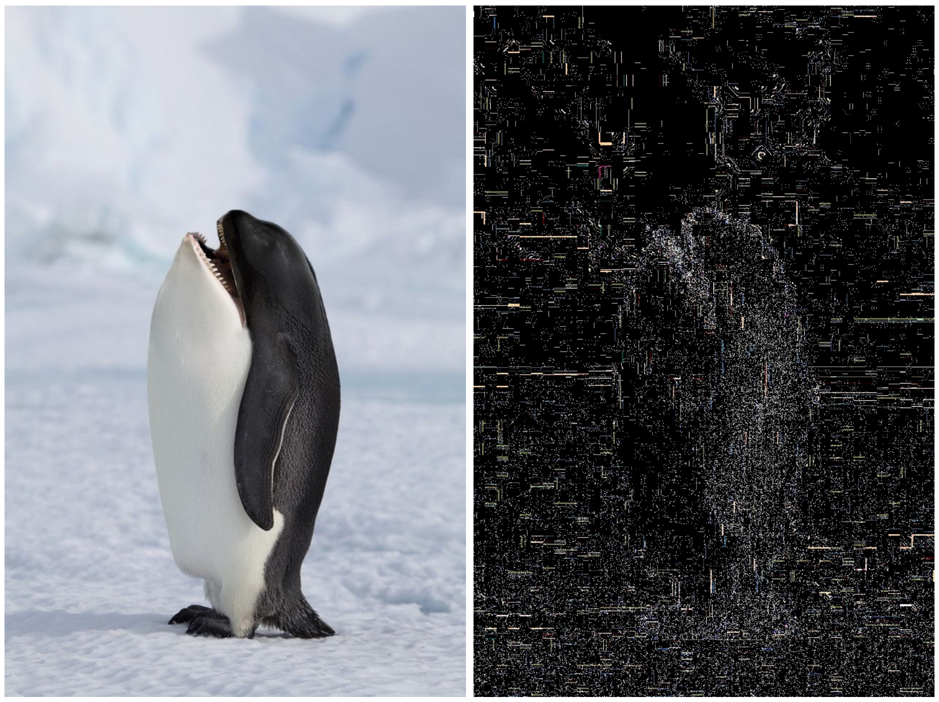
Algorithm Output

Figure 5 and Figure 6 show pristine and manipulated images with their corresponding error level analysis.



(a) Pristine image of a whale (b) ELA of pristine image

FIGURE 5: Pristine image with corresponding error level analysis



(a) Image of hybrid creature (b) ELA of image of hybrid creature

FIGURE 6: Manipulated image with corresponding error level analysis

I loaded Figure 5a into the ELA algorithm, and the output is given in Figure 5b. Figure 5b shows a nearly entirely black box, indicating that there is no noise in the image, which means that the image is real and not manipulated.

I calculated the error level of Figure 6a , as shown in Figure6b. The noise in the image is clear and indicates that the image of the hybrid animal is manipulated.

Further Discussion on the Error Level Analysis Algorithm

A variable quality level is used control the amount of compression of a JPEG image. The amount of JPEG compression is typically measured as a percentage of the quality level. An image at 100% quality has (almost) no loss, and a 1%-quality image is a very low-quality image. In general, quality levels of 90% or higher are considered high quality, 8090% is medium quality, and 7080% is low quality. Anything below 70% is typically very low quality [45]. Low-quality images can reduce the ability of analysis algorithms to detect modifications. The ELA algorithm works by resaving an image at a known quality level, such as 75%, and during Step 3 in the algorithm, it then identifies the amount of error introduced [45].

Quality Dependence

Saving an image with different quality levels affects the ELA algorithm, and this leads to a distinct number of bright pixels. The lower the quality, the higher the number of bright pixels. An image can be modified and saved in a quality lower than the quality used in the ELA algorithm, and this makes it difficult to detect whether the image is manipulated. Figure 7 shows an example of a pristine image that is saved with different qualities. Subfigures (a) and (d) show the outcome of saving an image with 90% quality. Subfigures (b) and (e) show the outcome of saving an image with 70% quality. Subfigures

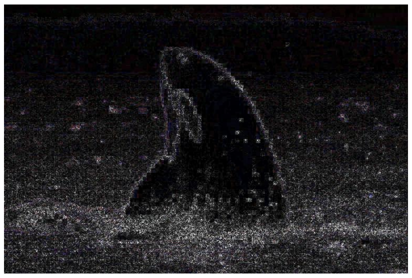
(c) and (f) show the outcome of saving an image with 48% quality. From Figure 7, I have concluded that the lower the quality, the greater the noise. This means that the ELA algorithm is not always accurate.

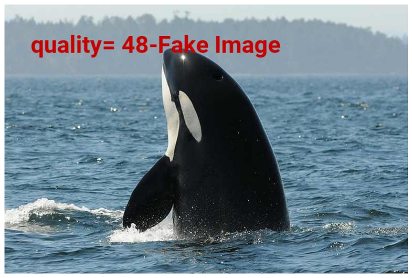
(a) ELA of 90% quality

(b) Result for 90% quality

(c) ELA of 70% quality

(d) Result for 70% quality


(e) ELA of 48% quality



(f) Result for 48% quality

FIGURE 7: The same image saved with different qualities

CHAPTER IV

EXPERIMENTS WITH ERROR LEVEL ANALYSIS AND RESULTS

The CASIA Dataset

The CASIA dataset [46] contains three categories that make it an appropriate dataset for this research:

1. Pristine images: Unspoiled images in their original form. Shown in Appendix A.
2. Copymove images: The manipulated region has been copied from the same image and pasted on another area of the same image. Shown in Appendix A.
3. Spliced images: The manipulated region has been copied from a different image and pasted on this image. Shown in Appendix A.

System Specifications

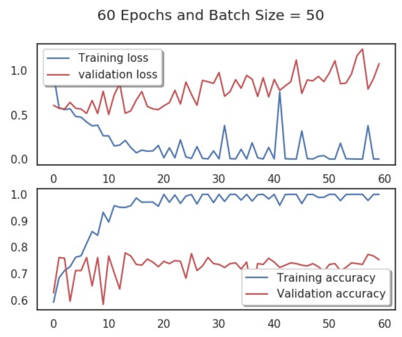
The specifications of the computer I used to conduct these experiments are the following:

- **Operating system**:Ubuntu 18.04.4 LTS.
- **CPU:** Intel® Core™i7.
- **RAM:** 16 GB.
- **GPU:** NVIDIA GeForce 2060.
- **Driver:** NVIDIA Driver 430.
- **External hard drive:** 6 TB external hard drive.

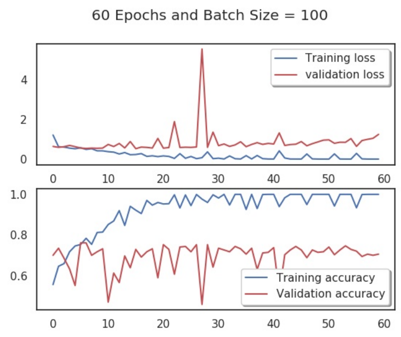
I tested the ELA algorithm on the CASIA dataset with three different classifiers: CNNs [47], SVMs, and RFs. The following sections contain explanations of and results for each classifier with the ELA algorithm.

Convolutional Neural Networks with ELA

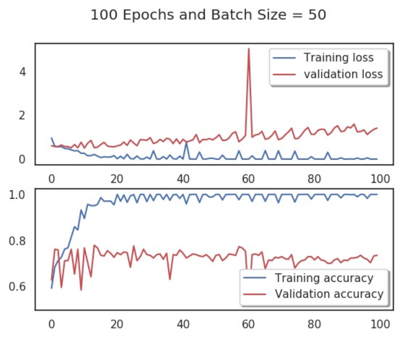
The image paths were passed to a function that converts the images into their ELA form, as discussed in Section 3.2. I then split the dataset into training and testing sets using the `train_test_split` method from `sklearn`. I subsequently created a CNN with three convolutional layers because it achieved better results than two layers. With four layers, the model became overfitted. The results are discussed in Section 4.3. The primary packages used include `pandas` to read images paths, `matplotlib.pyplot` to plot performance curves, `sklearn` and `keras` to create the neural network.
The `Image, ImageChops` and `ImageEnhance` modules were used in the ELA algorithm. Figure 8 shows validation and training accuracy for various numbers of epochs and batches.



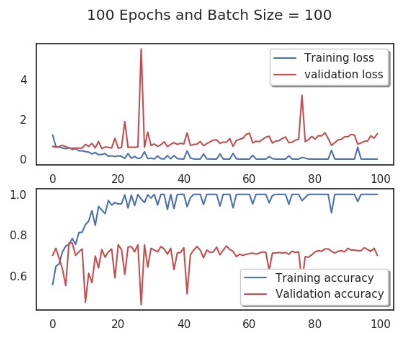
(a) 60 epochs batch size=50.



(b) 60 epochs batch size=100.



(c) 100 epochs batch size=50.



(d) 100 epochs batch size=100.

FIGURE 8: Testing with various numbers of epochs and batches

With the CNN, the model achieved a 79% accuracy. It had reached 80% when I initially ran it on PopOs 19.04. However, I were required to downgrade the system to Ubuntu 18.04 Bionic Beaver, since some Python packages and libraries are not yet supported by PopOs. Figure 8 shows various performance measure curves: (a) shows the accuracy of training with 60 epochs and a batch size of 50, (b) shows the accuracy of training with 60 epochs and a batch size of 100, (c) shows the accuracy of training with 100 epochs and a batch size of 50, and (d) shows the accuracy of training with 100 epochs and a batch size of 100. Of the training dataset, 80% was from the CASIA dataset. The remaining 20% was for validation. Figure 8 shows that training accuracy was high. With the validation data used on the model to evaluate its performance, the accuracy dropped to 72%.

Support Vector Machine with ELA

I converted all the images to their ELA format. I then passed the resulting two folders (pristine image folder and manipulated image folder) to the SVM [48] with `rbf` kernel. The results are shown in Section 4.4. The primary packages used include `skimage` to open images and `sklearn` and `keras` to use the SVM classifier.

Results

The SVM achieved 72% accuracy, as shown in Figure 9 :

```
              precision    recall  f1-score   support
0               0.70       0.81      0.75       184
1               0.73       0.61      0.66       160
accuracy                             0.72       344
macro avg       0.72       0.71      0.71       344
weighted avg    0.72       0.72      0.71       344
```

FIGURE 9: Support vector machine results

Random Forests with ELA

For feature extraction, I chose it based on the brightness of the pixels. I began counting bright pixels on the ELA form. I estimated 150 pixels after consulting the RGB table shown in Appendix C. According to the table, three zeros represent the color black. The combination *(255,255,255)* represents white, the brightest shade. The second row of the RGB table shows the combination *(127,127,127)* that represents gray. In my opinion, the shade of gray provided by this combination was too dark. I therefore chose the combination *(150,150,150)* as a boundary. Pixels brighter than *(150,150,150)* were considered to be noise in the ELA images. The brighter the pixels in an ELA image, the greater the noise. Figure 6b shows the ELA for a fake image. It contains 634 bright pixels. I created the condition that if there are more than 300 bright pixels on an ELA image, then the image is fake. I estimated 300 because some pristine images may have some noise, as discussed in Section 3.4. The results are shown in Section 4.5. The primary packages used include `pandas` to read images paths and `sklearn` to create the RF.

Results

The RFs in this study achieved 91% accuracy, as shown in Figure 10 :

```
              precision    recall  f1-score   support
0               0.86      0.96      0.90        94
1               0.93      0.79      0.85        71
accuracy                            0.91       165
macro avg       0.90      0.87      0.91       165
weighted avg    0.89      0.91      0.91       165
```

FIGURE 10: Random forest results

CHAPTER V

VGG16 EXPERIMENTS AND RESULTS

RAISE Dataset and Reference Database

RAISE is a demanding real-world image data collection, developed primarily to test digital forgery detection algorithms. It consists of approximately 8000 high-resolution RAW images, which are uncompressed and never processed [49]. An image from this dataset is shown in Appendix B. Reference Database is a free set of tagged screenshots taken from games. It is a computer graphics database [50]. An image from this dataset is shown in Appendix B. I conducted our experiments on 1800 randomly chosen images to compare our results to those from [10]. The authors of [10] used CNNs to detect forgery in an image. I used the same approach for image pre-processing but with a pre-trained ImageNet model, VGG16, instead of a standard CNN.

Image Pre-Processing Approach

The authors of [10] made it possible to use their pre-processing approach by providing the `CGvsPhoto` Python package.

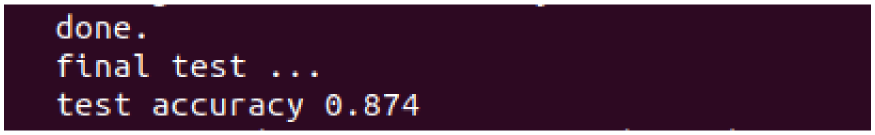
```
pip3 install CGvsPhoto
```

FIGURE 11: Command to install CGvsPhoto package

As mentioned in Chapter II, the pre-processing function takes every image in the dataset and divides it into 100 x 100 patches. The patches are then passed into a CNN classifier that classifies whether the given patch belongs to a raw image (green) or a computer graphics image (red). After classifying 100 patches, the pre-processing function displays the accuracy of the classification. The classifier saves weights in a `.ckpt` file after classifying 500 patches. After the classification of patches, the complete images are regenerated from the classified patches. The authors of [10] passed the resulting images into another CNN to obtain a final result, and they achieved an accuracy of 93.4%. I used the vGG16 pre-trained model to accomplish higher accuracy than using a standard CNN. For the pre-processing step, the authors of [10] achieved an accuracy of 84.4%.

VGG16 Pre-Trained Model

The authors of [10] achieved 84.4% accuracy on the pre-processing step. I obtained 87% on patch level. This higher accuracy was achieved because our weights were optimized. Figure 12 shows the obtained patch accuracy.



```
done.
final test ...
test accuracy 0.874
```

FIGURE 12: Patch accuracy

Figure 13 shows the curve of validation accuracy.

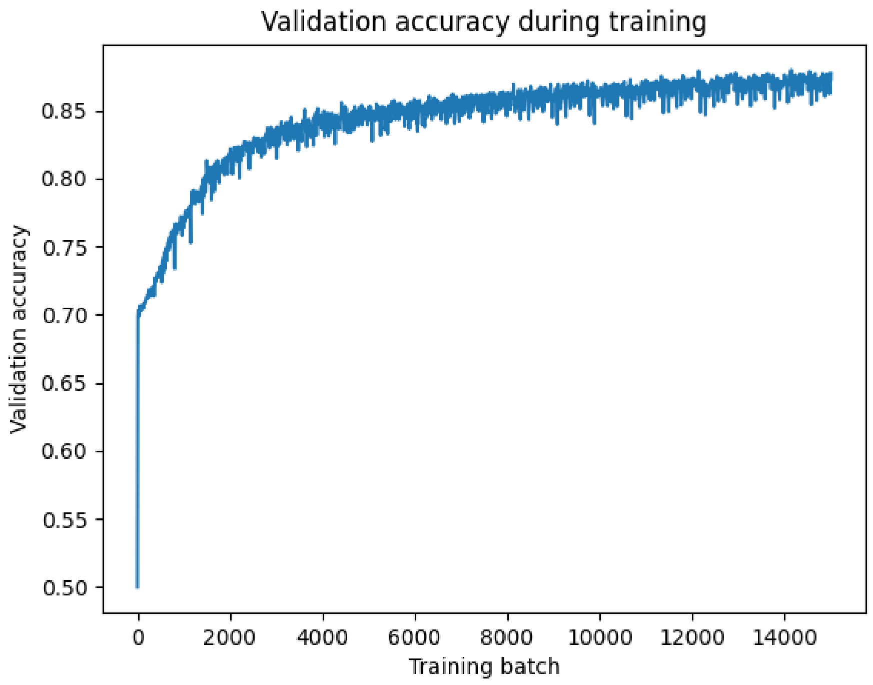
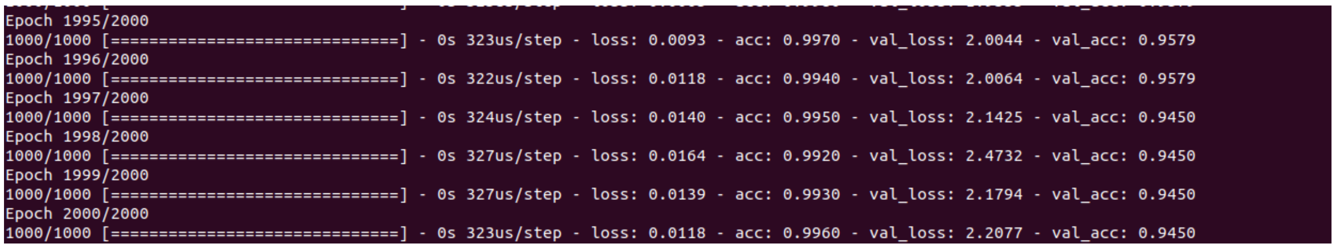


FIGURE 13: Patch validation accuracy

The authors of [10] passed the resulting complete images to another CNN and achieved 93.4% accuracy by training 1800 images. I obtained 94.5% accuracy by using a VGG16 pre-trained model. Figure 14 depicts our results.



```
Epoch 1995/2000
1000/1000 [==============================] - 0s 323us/step - loss: 0.0093 - acc: 0.9970 - val_loss: 2.0044 - val_acc: 0.9579
Epoch 1996/2000
1000/1000 [==============================] - 0s 322us/step - loss: 0.0118 - acc: 0.9940 - val_loss: 2.0064 - val_acc: 0.9579
Epoch 1997/2000
1000/1000 [==============================] - 0s 324us/step - loss: 0.0140 - acc: 0.9950 - val_loss: 2.1425 - val_acc: 0.9450
Epoch 1998/2000
1000/1000 [==============================] - 0s 327us/step - loss: 0.0164 - acc: 0.9920 - val_loss: 2.4732 - val_acc: 0.9450
Epoch 1999/2000
1000/1000 [==============================] - 0s 327us/step - loss: 0.0139 - acc: 0.9930 - val_loss: 2.1794 - val_acc: 0.9450
Epoch 2000/2000
1000/1000 [==============================] - 0s 323us/step - loss: 0.0118 - acc: 0.9960 - val_loss: 2.2077 - val_acc: 0.9450
```

FIGURE 14: Validation accuracy is 94.5%

Figure 15 and Figure 16 show the curves of training and validation.

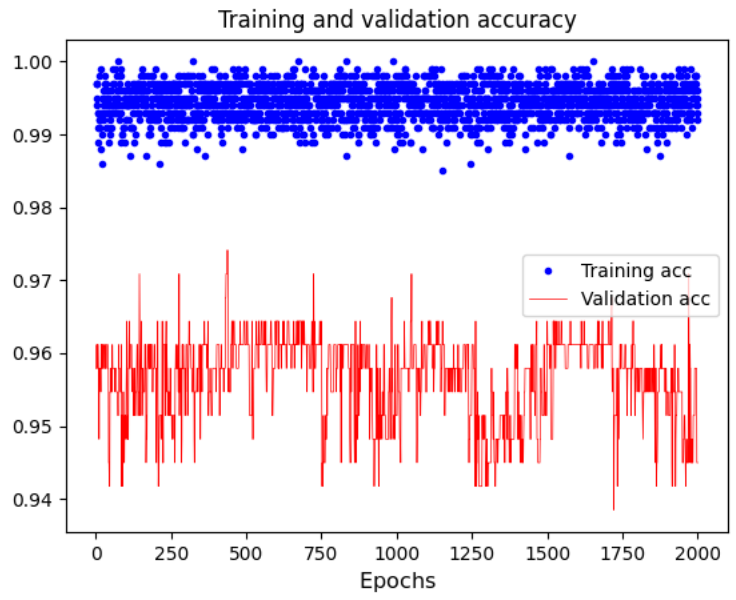


FIGURE 15: Whole image training and validation accuracy

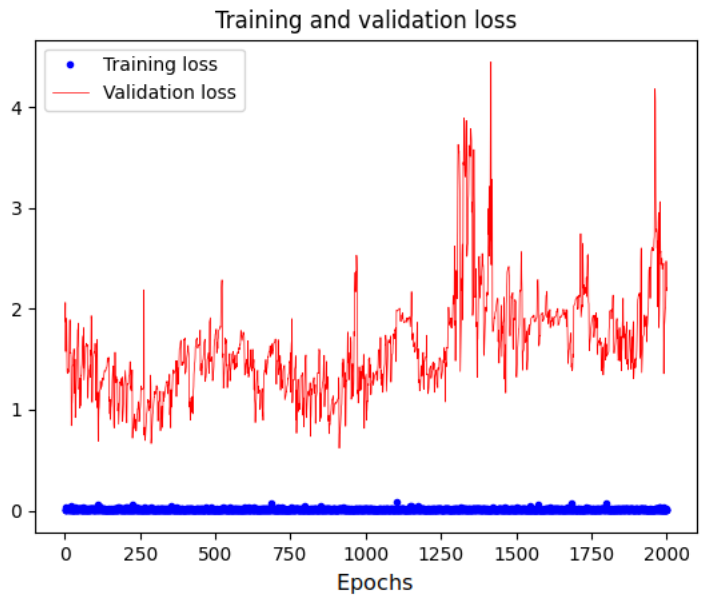


FIGURE 16: Whole image training and validation loss

I wrote a program that generates randomly spliced images, illustrated in Figure17.

```
from PIL import Image, ImageDraw, ImageFilter
import os, random

export="/lubna/CWU/Thesis/generatingFakeImages/SpHuge/"
realPath="/lubna/CWU/Thesis/generateFakeImages/realimSP/"
realfiles=os.listdir(realPath)
fakePath="/lubna/CWU/Thesis/generatingFake/fakeimgSP/"
fakefiles=os.listdir(fakePath)

for i in range(1439):
   left = 155
   top = 65
   right = 600
   bottom = 600
   x=random.randint(0,2300)
   y=random.randint(0,2300)
   a=random.choice(realfiles)
       b=random.choice(fakefiles)
   im1 = Image.open(realPath+a)
   im2 = Image.open(fakePath+b)
   im3=im2.crop((left, top, right, bottom))
   backim = im1.copy()
   backim.paste(im3, (x,y))
   backim.save(export+str(i)+'.jpg', quality=95)
```

FIGURE 17: Generating random spliced images

By using the image pre-processing function found in the library, I could locate the spliced area in an image. I tested our model with a randomly generated spliced image, and it was successful, as shown in Figure 18.

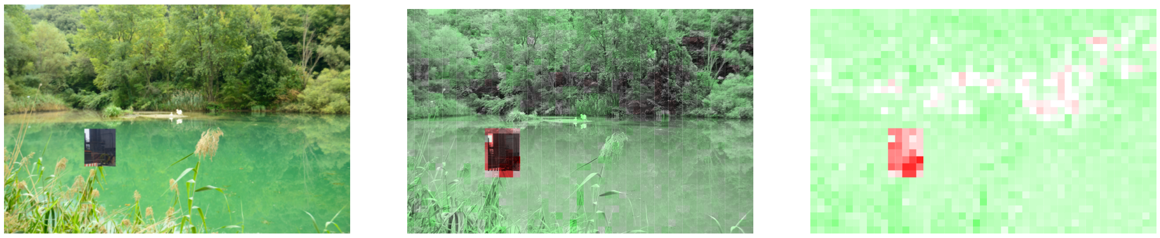


FIGURE 18: Detecting spliced area

I also took a photograph from a personal camera and pasted an object onto it. Figure 19 shows the result of detecting a spliced area in this photograph.

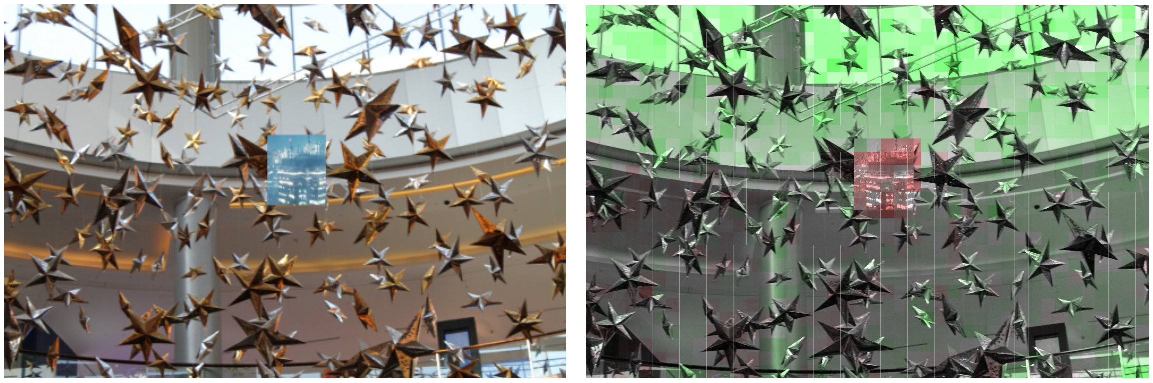


FIGURE 19: Benchmark 1: personal image with spliced region

I also attempted to input an image with lower brightness and a different object location, as shown in Figure 20.

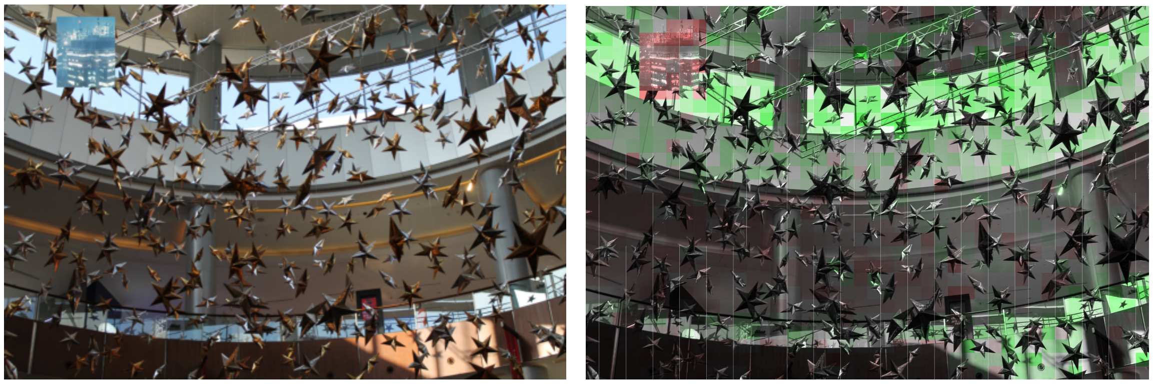


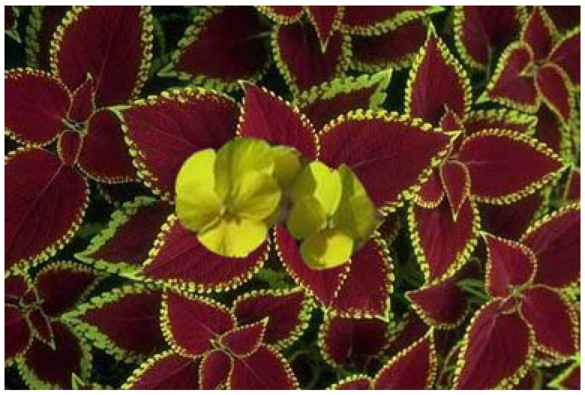
FIGURE 20: Benchmark 2: spliced personal image with low brightness

CHAPTER VI

FUTURE WORK AND CONCLUSIONS

Future Work : Ground Truth Masks

Generating the ground truth mask of a manipulated image is more reliable than the ELA algorithm, and it is not affected by the quality of the image. It also shows the exact area of manipulation. Obtaining the ground truth mask requires knowledge of image pre-processing and image segmentation techniques. Figure 21 shows a manipulated image with its ground truth mask.


(a) Manipulated image.

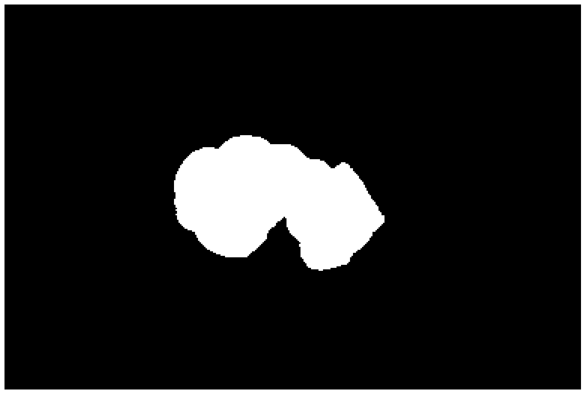

(b) Ground truth mask.

FIGURE 21: Manipulated image with its ground truth mask

Conclusions

Image forgery involves distorting images, sometimes images of people, for malicious reasons. This involves a genuine image that had been displayed on a public website or a digital communication platform and is edited into an entirely different image. The new image will likely be immoral in nature or targeted to spread negative publicity.

The ELA algorithm shows whether an image is manipulated when the input images quality is close to the quality used in the algorithm. If there is a large difference between the quality of the image and the quality of the algorithm, then the result will always be incorrect. Furthermore, the algorithm does not show the exact area of manipulation.

A pre-trained model is a model that has been trained on a certain task on the ImageNet dataset. It is a model that has been trained to solve issues that might be similar to the problem at hand. A pre-trained model is preferred in most cases to training a model from scratch. The process of importing a pre-trained model is referred to as transfer learning.

Other approaches do not depend on the quality of the images and show the exact area of manipulation. The patch classification approach is not affected by the quality of the image and achieves more accurate results. Commonly imported models such as VGG and MobileNets have been trained on large sets of data and are therefore very efficient on any given dataset. The time required to train a model from scratch, depending on experience, is relatively high, which makes it necessary to consider using pre-trained models.

Bibliography

[1] M. Sridevi, C. Mala, and S. Sanyam, “Comparative study of image forgery and copy-move techniques,” in *Advances in Computer Science, Engineering & Applications*, D. C. Wyld, J. Zizka, and D. Nagamalai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 715–723.

[2] S. Walia and K. Saluja, “Digital image forgery detection: a systematic scrutiny,” *Australian Journal of Forensic Sciences*, pp. 1–39, 03 2018.

[3] C. Shen, M. Kasra, W. Pan, G. A. Bassett, Y. Malloch, and J. F. OBrien, “Fake images: The effects of source, intermediary, and digital media literacy on contextual assessment of image credibility online,” *New Media & Society*, vol. 21, no. 2, pp. 438–463, 2019. [Online]. Available: https://doi.org/10.1177/1461444818799526

[4] C. N. Bharti and P. Tandel, “A survey of image forgery detection techniques,” *2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pp. 877–881, 2016.

[5] C. Salge, “Is that social bot behaving unethically?” *Communications of the ACM*, vol. 60, pp. 29–31, 08 2017.

[6] J. Bunk, J. H. Bappy, T. M. Mohammed, L. Nataraj, A. Flenner, B. S. Manjunath, S. Chandrasekaran, A. K. Roy-Chowdhury, and L. Peterson, “Detection and localization of image forgeries using resampling features and deep learning.” in *CVPR Workshops*. IEEE Computer Society, 2017, pp. 1881–1889. [Online]. Available: http://dblp.uni-trier.de/db/conf/cvpr/cvprw2017.html#BunkBMNFMCRP17

[7] Y. Abdalla, M. Iqbal, and M. Shehata, "Image forgery detection based on deep transfer learning," *European Journal of Electrical Engineering and Computer Science*, vol. 3, no. 5, Sep. 2019. [Online]. Available: https://ejece.org/index.php/ejece/article/view/125

[8] J. H. Bappy, C. Simons, L. Nataraj, B. S. Manjunath, and A. K. Roy-Chowdhury, "Hybrid lstm and encoderdecoder architecture for detection of image forgeries," *IEEE Transactions on Image Processing*, vol. 28, no. 7, pp. 3286–3300, July 2019.

[9] X. Wang, H. Wang, S. Niu, and J. Zhang, "Detection and localization of image forgeries using improved mask regional convolutional neural network."

[10] N. Rahmouni, V. Nozick, J. Yamagishi, and I. Echizen, "Distinguishing computer graphics from natural images using convolution neural networks," in *2017 IEEE Workshop on Information Forensics and Security (WIFS)*, 2017, pp. 1–6.

[11] G. Kaur and D. R. Manro, "A brief review : Copy-move forgery detection," 2018.

[12] K. G. Amit Doegar, Maitreyee Dutta, "Cnn based image forgery detection using pre-trained alexnet model," Ph.D. dissertation, CHANDIGARH, India, 2020.

[13] P. Zhou, X. Han, V. I. Morariu, and L. S. Davis, "Learning rich features for image manipulation detection," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[14] A. Gupta, N. Saxena, and S. Kumar, "Detecting copy move forgery in digital images," *International Journal of Engineering Research and Applications*, vol. 3, no. 1, pp. 94–97, 03 2013.

[15] B. Shivakumar and S. Baboo, "Detection of region duplication forgery in digital images using surf," *International Journal of Computer Science Issues*, vol. 8, no. 3, pp. 199–205, 07 2011.

[16] R. Salloum, Y. Ren, and C.-C. J. Kuo, "Image splicing localization using a multi-task fully convolutional network (mfcn)," *J. Vis. Commun. Image Represent.*, vol. 51, pp. 201–209, 2018.

[17] N. T. Pham, J.-W. Lee, G.-R. Kwon, and C.-S. Park, "Hybrid image-retrieval method for image-splicing validation," *Symmetry*, vol. 11, no. 1, p. 83, 2019.

[18] A. L. Caterini and D. E. Chang, *Deep Neural Networks in a Mathematical Framework*, 1st ed. Springer Publishing Company, Incorporated, 2018.

[19] K. Kang, "Comparison of face recognition and detection models: Using different convolution neural networks," *Opt. Mem. Neural Netw.*, vol. 28, no. 2, p. 101108, Apr. 2019. [Online]. Available: https://doi.org/10.3103/S1060992X19020036

[20] Y. Mao, Z. He, Z. Ma, X. Tang, and Z. Wang, "Efficient convolution neural networks for object tracking using separable convolution and filter pruning," *IEEE Access*, vol. 7, pp. 106 466–106 474, 2019.

[21] K. Uchida, M. Tanaka, and M. Okutomi, "Coupled convolution layer for convolutional neural network," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016, pp. 3548–3553.

[22] Wikipedia, "Support-vector networks," pp. 273–297, 1995.

[23] R. Pupale, "Support Vector Machines(SVM) An Overview," https://towardsdatascience.com/, 2018.

[24] W. Land and J. Schaffer, *The Support Vector Machine*, 01 2020, pp. 45–76.

[25] X. Zhang, *Support Vector Machines*, 01 2017, pp. 1214–1220.

[26] Tin Kam Ho, “Random decision forests,” pp. 278–282 vol.1, 1995.

[27] Sklearn.ensemble.RandomForestClassifier, “sklearn Random forest Classifier,” https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, 2020.

[28] N. Horning, “Random forests: An algorithm for image classification and generation of continuous fields data sets,” in *Proceedings of the International Conference on Geoinformatics for Spatial Infrastructure Development in Earth and Allied Sciences, Osaka, Japan*, vol. 911, 2010.

[29] R. Pandya and J. Pandya, “C5. 0 algorithm to improved decision tree with feature selection and reduced error pruning,” *International Journal of Computer Applications*, vol. 117, pp. 18–21, 05 2015.

[30] B. Gregorutti, B. Michel, and P. Saint-Pierre, “Correlation and variable importance in random forests,” *Statistics and Computing*, vol. 27, 10 2013.

[31] Q. Zhang, Y. Yang, H. Ma, and Y. Wu, “Interpreting cnns via decision trees,” 06 2019, pp. 6254–6263.

[32] L. Ma, B. Sun, and Z. Li, “Bagging likelihood-based belief decision trees,” 07 2017.

[33] K. Pal and K. Sudeep, “Preprocessing for image classification by convolutional neural networks,” 05 2016, pp. 1778–1781.

[34] M. Elgendy, *Deep Learning for Vision Systems*.

[35] GeeksforGeeks, “VGG-16 — CNN model,” shorturl.at/tvAKM, 2019.

[36] S. Hinterstoisser, V. Lepetit, P. Wohlhart, and K. Konolige, *On Pre-trained Image Features and Synthetic Images for Deep Learning: Munich, Germany, September 8-14, 2018, Proceedings, Part I*, 01 2019, pp. 682–697.

[37] M. Simon, E. Rodner, and J. Denzler, “Imagenet pre-trained models with batch normalization,” vol. 21, no. 1, pp. 2–4, 12 2016.

[38] M. Long, H. Zhu, J. Wang, and M. I. Jordan, “Deep transfer learning with joint adaptation networks,” p. 22082217, 2017.

[39] K. He, R. Girshick, and P. Dollar, “Rethinking imagenet pre-training,” in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, vol. 8, no. 2, 2019, pp. 4917–4926.

[40] R. Thakur, “Step by step VGG16 implementation in Keras for beginners,” shorturl.at/gjzL5, 2019.

[41] A. S. Pankaj Kumar Kandpal, Ashish Mehta, “Honey bee bearing pollen and non-pollen image classification, vgg16 transfer learning method using different optimizing functions,” *International Journal of Innovative Technology and Exploring Engineering (IJITEE)*, vol. 57, pp. 2–5, 12 2019.

[42] A. Omar, “Lung ct parenchyma segmentation using vgg-16 based segnet model,” *International Journal of Computer Applications*, vol. 178, pp. 10–13, 08 2019.

[43] I. Steadman, “ ’Fake’ World Press Photo isn’t fake, is lesson in need for forensic restraint,” https://www.wired.co.uk/article/photo-faking-controversy, 2012.

[44] S. Masters, “ Error Level Analysis,” shorturl.at/cfuBE, 2016.

[45] Fotoforensics, “Tutorial: Error Level Analysis,” https://fotoforensics.com/tutorial-ela.php, 2012-2019.

[46] Y. Z. IEEE Dataport, “CASIA Dataset,” http://dx.doi.org/10.21227/c1h8-kf39, 2019.

[47] I. Zafar, G. Tzanidou, R. Burton, N. Patel, and L. Araujo, *Hands-On Convolutional Neural Networks with TensorFlow: Solve Computer Vision Problems with Modeling in TensorFlow and Python*. Packt Publishing, 2018.

[48] Sklearn, “Support Vector Machines,” https://scikit-learn.org/stable/modules/svm.html, 2020.

[49] D.-T. Dang-Nguyen, C. Pasquini, V. Conotter, and G. Boato, “Raise: A raw images dataset for digital image forensics,” in *Proceedings of the 6th ACM Multimedia Systems Conference*, ser. MMSys 15. New York, NY, USA: Association for Computing Machinery, 2015, p. 219224. [Online]. Available: https://doi.org/10.1145/2713168.2713194

[50] L. Design, “Reference Database,” http://level-design.org/referencedb/, 2009-2020.

[51] T. Binary, “Convert Truecolor To Binary Online,” shorturl.at/quQX4, 2019.

APPENDIX A

CASIA DATASET

FIGURE 22: Pristine image

In Figure 23, the top right flower has been cut, resized, and pasted onto the lower left section of the image.

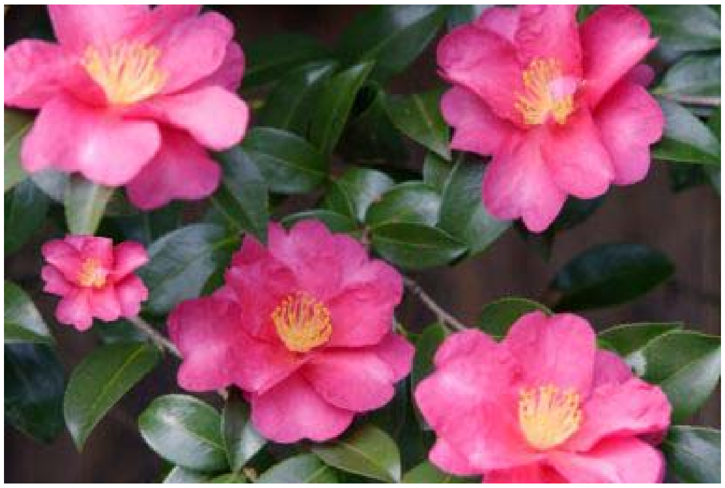


FIGURE 23: Copy-move image.

In Figure 24, the two yellow flowers in the top left have been cut from Figure 24a and pasted onto Figure 24b. The result is shown in Figure 24c.

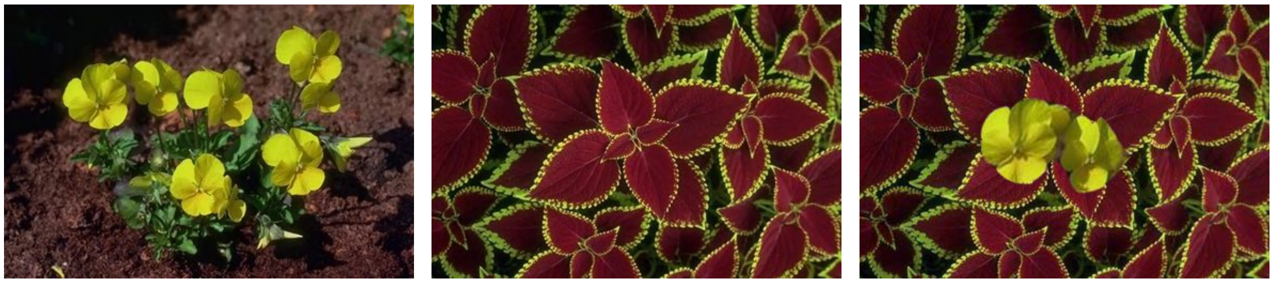

(a) Source (b) Destination (c) Spliced image

FIGURE 24: Image splicing

APPENDIX B

RAISE DATASET AND REFERENCE DATABASE

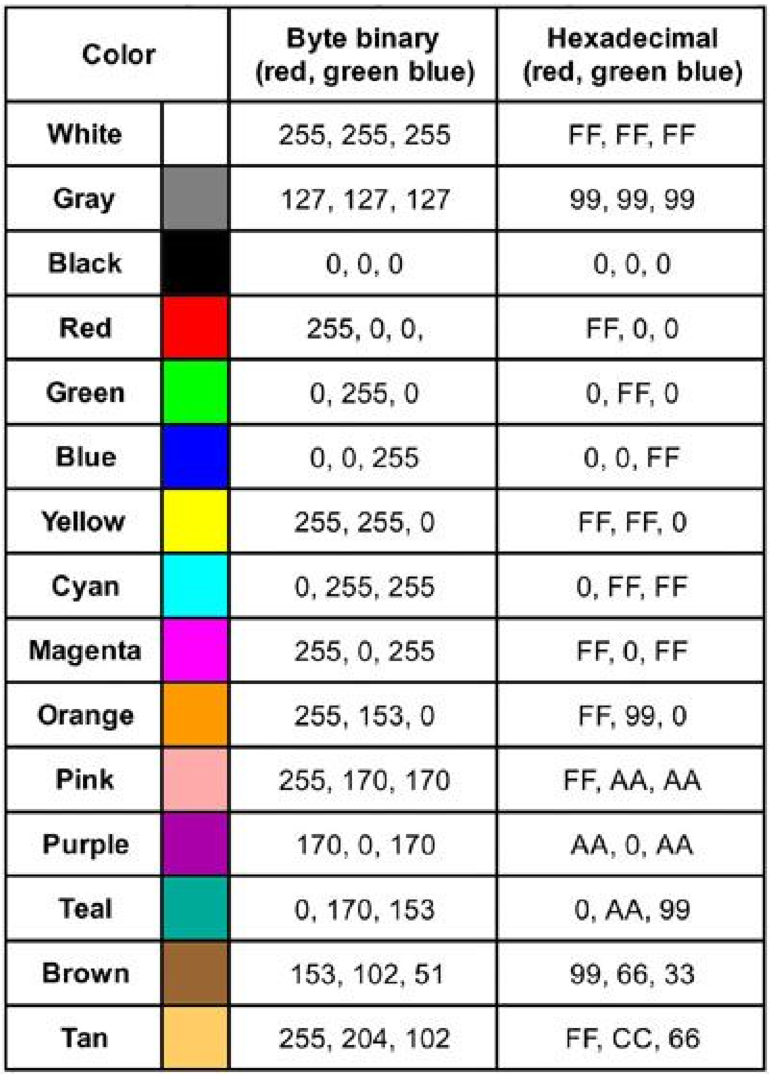
FIGURE 25: An image from the RAISE dataset

FIGURE 26: An image from the Reference Database

APPENDIX C

RGB TABLE

Figure 27 shows the RGB table from [51]



Color		Byte binary (red, green blue)	Hexadecimal (red, green blue)
White	⬜	255, 255, 255	FF, FF, FF
Gray	◼	127, 127, 127	99, 99, 99
Black	⬛	0, 0, 0	0, 0, 0
Red	🟥	255, 0, 0,	FF, 0, 0
Green	🟩	0, 255, 0	0, FF, 0
Blue	🟦	0, 0, 255	0, 0, FF
Yellow	🟨	255, 255, 0	FF, FF, 0
Cyan	🟦	0, 255, 255	0, FF, FF
Magenta	🟪	255, 0, 255	FF, 0, FF
Orange	🟧	255, 153, 0	FF, 99, 0
Pink	🟥	255, 170, 170	FF, AA, AA
Purple	🟪	170, 0, 170	AA, 0, AA
Teal	🟩	0, 170, 153	0, AA, 99
Brown	🟫	153, 102, 51	99, 66, 33
Tan	🟨	255, 204, 102	FF, CC, 66

FIGURE 27: RGB table