IMAGE FORGERY DETECTION

A Graduation Project Thesis Presented to

School of Information Technology and Computer Science

Nile University

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science

By

Maged Ahmed

Andrew Tharwat

Anas Hany

Ziad Tamer

Under Supervision of

Prof. Hala Zayed

Eng. Mostafa Fathi

7/2023

ACKNOWLEDGEMENTS

We extend our sincere gratitude and appreciation to the esteemed faculty members, teaching assistants, and staff of the Computer Science Faculty at Nile University. Without their unwavering support and invaluable guidance, our graduation project would not have been possible.

We are especially grateful to Dr. Hala Zayed and Eng. Mostafa Fathi for their exceptional mentorship, expert knowledge, and continuous encouragement throughout our project. Their insightful feedback, constructive criticism, and dedication to our success have played a pivotal role in shaping our research and pushing us to achieve our best.

We would also like to express our heartfelt thanks to our professors, whose expertise and passion for their respective fields have been a constant source of inspiration. Their commitment to academic excellence, tireless efforts in providing quality education, and willingness to go above and beyond to address our queries have greatly enriched our learning experience.

To the teaching assistants who have generously shared their knowledge, patiently assisted us in practical matters, and provided timely guidance, we extend our deepest appreciation. Their support has been instrumental in overcoming challenges and ensuring the smooth progress of our project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

This graduation project focused on detecting image forgery, specifically splicing and copy move variants. We achieved 93% accuracy in splicing detection using ResNet50 and 90% accuracy in copy move detection using InceptionV3. Additionally, we successfully identified AI-generated faces with 98% accuracy using DenseNet. Our algorithms were integrated into a user-friendly website, facilitating easy image upload and forgery detection. We also employed traditional machine learning techniques, achieving 98% accuracy in detecting copy move and splicing forgery using DCT with an SVM classifier. Evaluation on multiple datasets confirmed the robustness of our models. This project contributes to image forensics and emphasizes the need for reliable tools in combating digital image tampering.

Chapter 1: Introduction to Image Forgery and its detection

In today's digital era, images have become an integral part of our lives. They are used for various purposes, including communication, documentation, and entertainment. However, the widespread availability of powerful image editing tools has raised concerns about the authenticity and integrity of digital images. Image forgery, the act of manipulating or altering images to deceive or mislead viewers, has emerged as a significant challenge in the field of computer vision and image processing.

1.1. Image forgery techniques

1.1.1 Copy-move

One common type of image forgery is copy-move forgery, where a portion of an image is copied and pasted onto another part to conceal or duplicate objects.

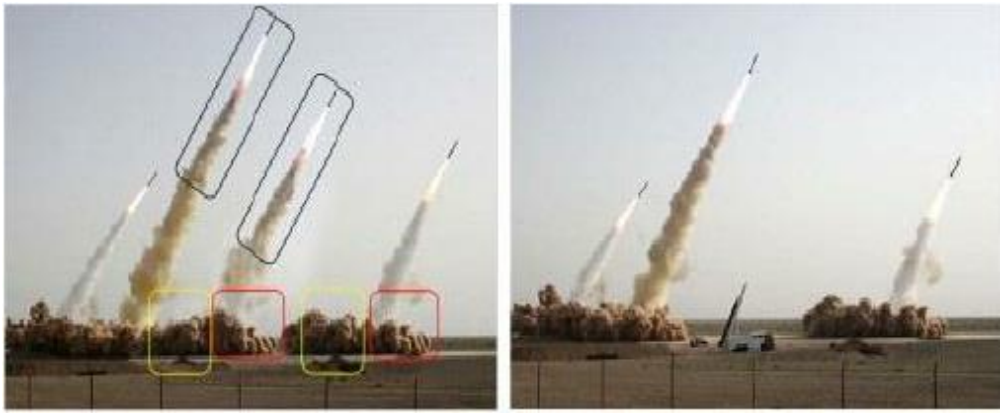


Figure 1 copy-move example

1.1.2 Splicing

Another prevalent form of image forgery is splicing, where distinct parts of multiple images are combined to create a composite image.

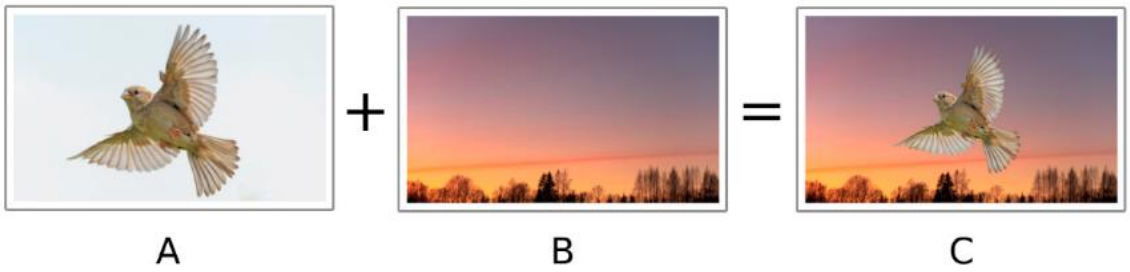


Figure 2 splicing.

1.1.3 Retouching

In Image Retouching, the images are less modified. It just enhances some features of the image. There are several subtypes of digital image retouching, technical retouching, and creative retouching.

Figure 3 retouching.

1.1.4 Resampling

Resampling is a mathematical technique to change the resolution (number of samples) of an image, to increase the size of the image (up sampling) for printing banners and hoardings.

Figure 4 Resampling.

1.2 Image forgery detection techniques

Image forgery detection is split into different techniques: pixel based, format based, camera based, physical based and geometry based. In this project for phase one and two we focus on pixel-based techniques.

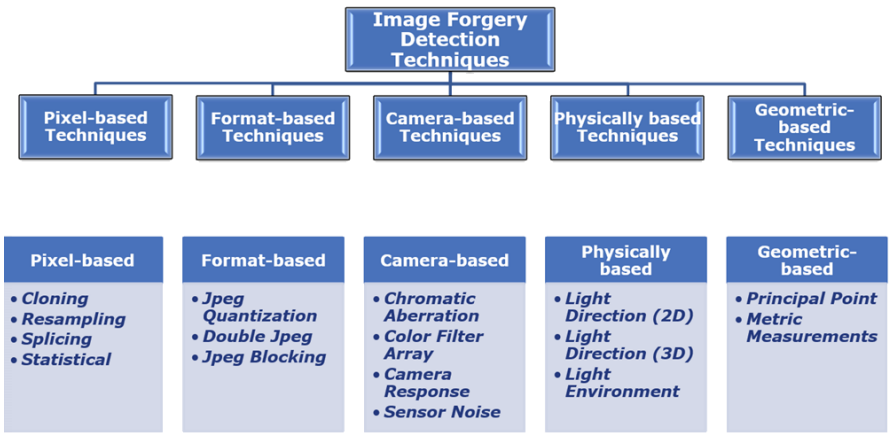


Figure 5 detection techniques.

1.2.1 Format-based

Format-based image forgery detection techniques are based on image formats, in which the JPEG format is preferable. These techniques can be further classified into JPEG quantization, Double JPEG compression, and JPEG Blocking.

1.2.2 Camera-based

Camera-based image forgery detection techniques focus on identifying any inconsistencies in digital footprints left by the camera during the complete process from image acquisition to image editing. These digital footprints are present in an image due to camera artifacts.

The four main methods that work on camera-based digital image forgery detection are sensor noise, color filter array, chromatic aberration, and camera response.

1.2.3 Physical-based

Physical-based image forgery detection techniques look for lighting anomalies in complex natural lighting. When splicing items from multiple photos, it is tough to achieve physically constant illumination, and research demonstrates that such errors are difficult to detect with human eyes.

1.2.4 Geometry-based

Geometry-based image forgery detection techniques involve the identification of inconsistencies in the geometric measurement of an object and its relative position with respect to the camera. These can be further divided into two types: Principal point and metric measurement.

1.2.5 pixel-based

Pixel-based image forgery detection is a blind approach which aims to verify the authenticity of digital images without any prior knowledge of the original image.

There are many ways for tampering an image such as cloning, resampling an image, addition, and removal of any object from the image.

1.3 Pixel-Based image forgery detection

1.3.1 Copy-Move image forgery detection:

Copy-move forgery involves duplicating a portion of an image and pasting it onto another region within the same image. To detect such forgeries, the following steps can be applied:

a. Feature Extraction: LBP is a widely used feature extraction technique for detecting copy-move forgeries. LBP encodes the texture information of an image by comparing the values of pixels in a neighborhood and constructing a binary pattern. LBP histograms or LBP-based descriptors are computed for each image block or pixel region to capture the distinctive patterns.

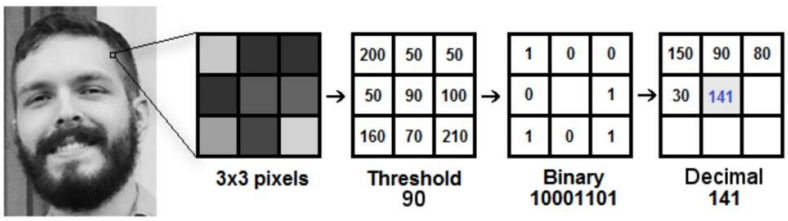


Figure 6 LBP.

b. Matching: Once the features are extracted, matching techniques such as block matching or KeyPoint-based matching can be employed to identify duplicated regions. Similarity measures like Euclidean distance or cross-correlation are commonly used to compare the extracted features.

And key-point based methods extract feature points only on regions from an image without any subdivisions of an image. It extracts feature points using different

methods like Scale-Invariant Feature Transform (SIFT) or Speeded Up Robust Features (SURF) algorithms without any image subdivision.

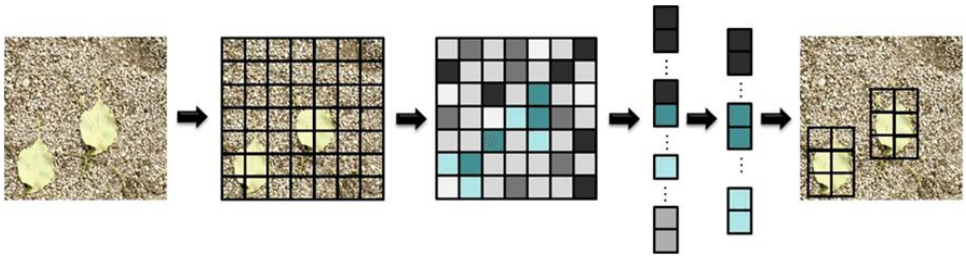


Figure 7

1.3.2 Splicing image Forgery detection:

Splicing detection has multiple methods too, such as DCT and DWT for feature extraction.

DCT: Discrete Cosine Transform decomposes the image into a set of frequency components. By applying DCT to overlapping blocks, features are extracted based on the coefficients of the transformed blocks. The differences or variations in these coefficients can indicate manipulated regions.

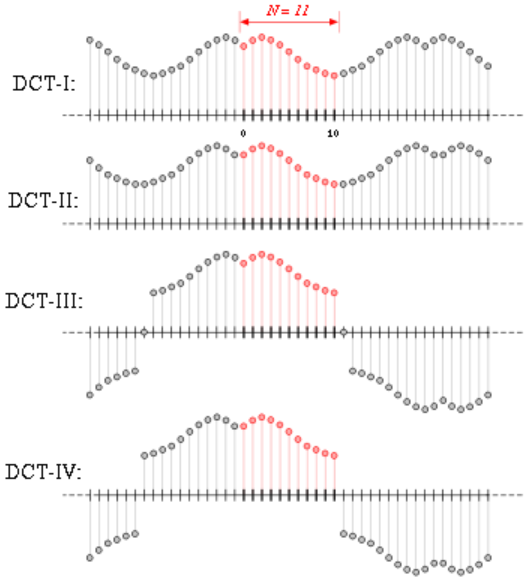


Figure 8 DCT

DWT: Discrete Wavelet Transform decomposes the image into different frequency bands or sub-bands, representing different scales. The statistical properties, such as mean and variance, of the wavelet coefficients are used as features for detecting splicing forgeries.

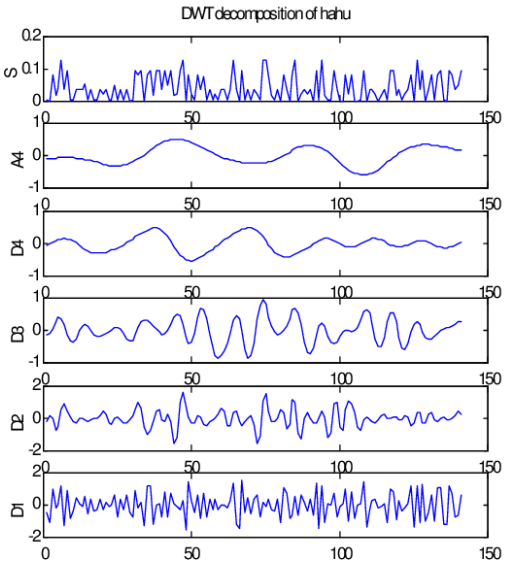


Figure 9 DWT

The extracted features from DCT or DWT are analyzed to identify inconsistencies in the statistical properties. Splicing forgeries often introduce irregularities in the frequency domain due to the blending of various sources. Deviations in the statistical properties can indicate tampered regions.

1.3.3 Using machine learning and deep learning for pixel-based forgery detection.

Machine learning, a subfield of artificial intelligence, involves the development of algorithms that can learn from data and make predictions or decisions without being explicitly programmed. Deep learning, a subset of machine learning, focuses on training deep neural networks with multiple layers to learn complex representations from raw data.

Machine learning algorithms, such as Support Vector Machines (SVM), Random Forests, can be trained using labeled datasets of authentic and manipulated images. These algorithms learn patterns and characteristics that distinguish genuine images from manipulated ones. Once trained, the model can classify new images as authentic or forged based on the extracted features.

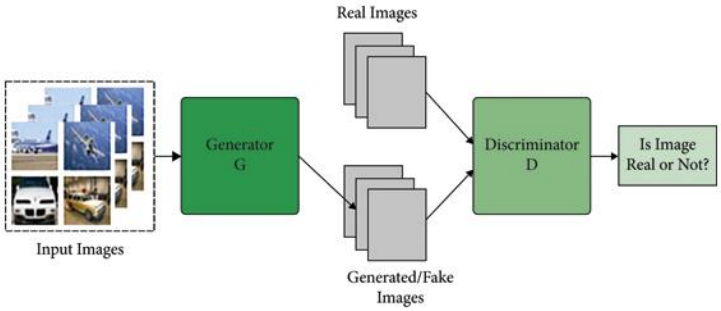


Figure 10

In summary, image forgery detection using machine learning involves feature extraction techniques like LBP, DCT, and DWT to capture distinctive patterns and

statistical properties. These features are then used for matching and analysis to identify copy-move and splicing forgeries in digital images using classifiers like support vector machines (SVM).

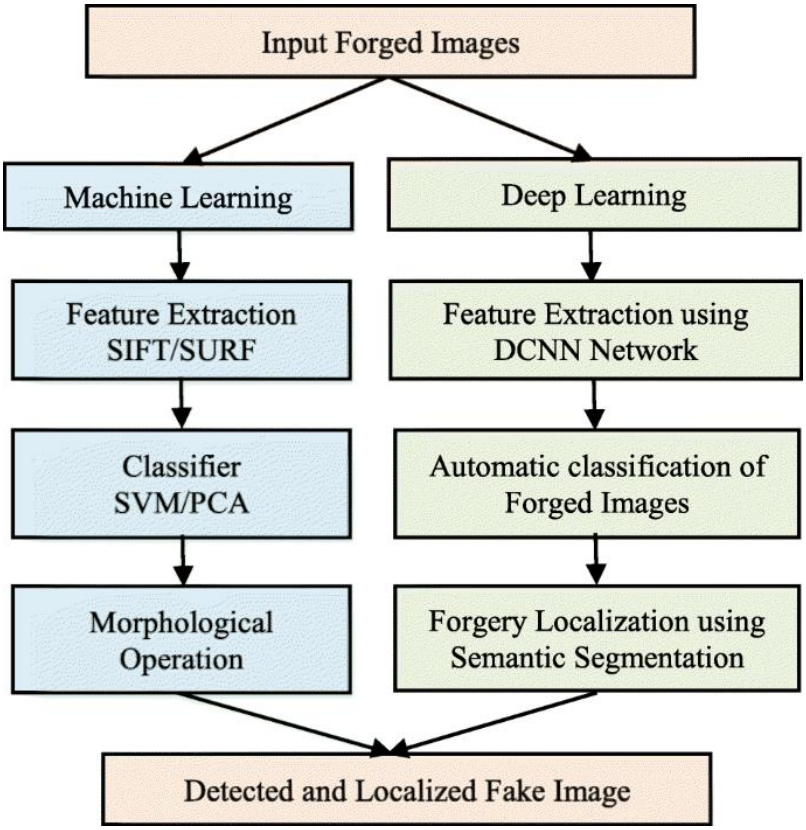


Figure 11 ML VS DL

The detection of image forgery using machine learning and deep learning techniques involves training models on features or datasets that they can use to learn to distinguish between authentic and forged images.

In the case of Machine Learning, we can extract features from the input image using

the methods mentioned previously. By training the model on these features, it can learn to detect these features in the input data and therefore predict if an image is authentic or tampered.

As for Deep Learning, the advantage with these models is they can learn the image features on their own. By training on large and diverse datasets, these models can learn to recognize common characteristics of manipulated images and generalize their knowledge to detect previously unseen forgeries.

Deep learning models, such as ResNet-50 and InceptionV3, have demonstrated significant success in image forgery detection, including the splicing and copy-move variants. By leveraging their deep architectures and ability to learn complex representations, these models can effectively identify and classify manipulated regions within images.

ResNet-50: with its residual connections, is a popular deep convolutional neural network (CNN) architecture that can be utilized for image forgery detection. It excels in capturing fine-grained details and subtle inconsistencies that are characteristic of splicing and copy-move forgeries. The deep structure of ResNet-50 allows it to learn hierarchical representations from images, enabling the detection of irregularities and duplications introduced by the manipulation techniques.

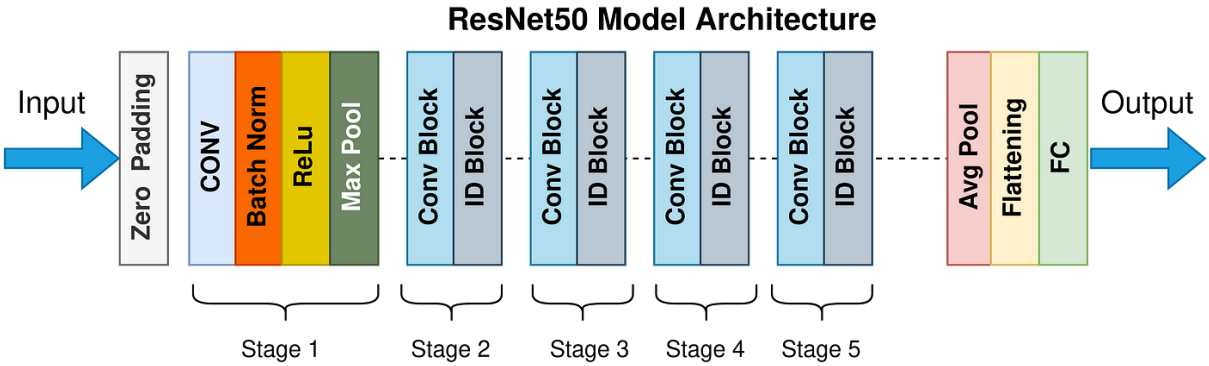


Figure 12 ResNet50.

InceptionV3: another deep CNN architecture, has demonstrated exceptional performance in various computer vision tasks and can be effectively employed for image forgery detection. InceptionV3 utilizes inception modules, which consist of parallel convolutional layers with different filter sizes, allowing the network to capture features at multiple scales. This multi-scale analysis is advantageous for detecting variations in size, rotation, and scaling introduced by splicing and copy-move forgeries.

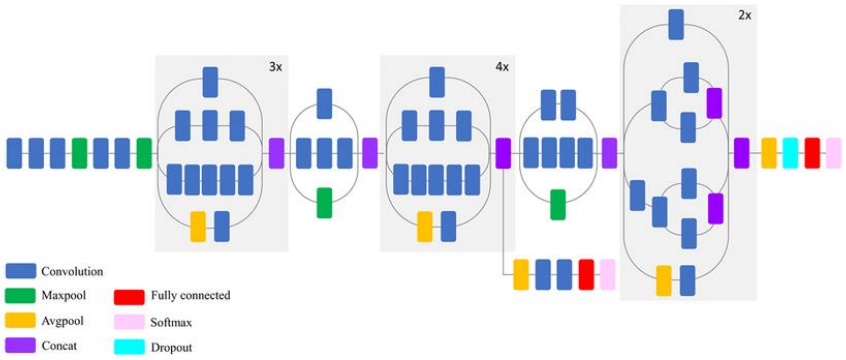


Figure 13 inceptionv3

To leverage these deep learning models for splicing and copy-move forgery detection, a two-step approach can be adopted:

Training Phase: In the training phase, a large dataset of labeled images comprising both authentic and manipulated examples is required. For splicing forgery detection, the dataset should include images with regions that have been copied and pasted. For copy-move forgery detection, the dataset should include images with duplicated regions. The deep learning models, such as ResNet-50 and InceptionV3, are trained on these datasets, learning to extract relevant features and classify images as authentic or manipulated.

Inference Phase: In the inference phase, the trained models are deployed to classify new images. The input images are passed through the deep learning models, and the models output the probability distribution over the classes (authentic or manipulated). By comparing the probabilities, the models can effectively identify and localize spliced or copied regions within the images.

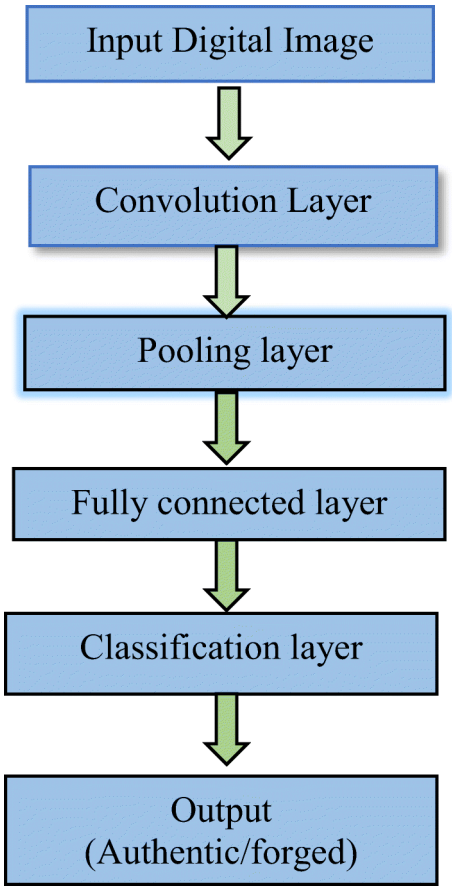


Figure 14 CNN general flow

The success of deep learning models for image forgery detection relies on the availability of diverse and well-labeled datasets. These datasets should encompass a wide range of splicing and copy-move forgeries, including variations in size, rotation, scaling, and content. Additionally, the models can benefit from data augmentation techniques, such as random cropping, flipping, and rotation, to increase their robustness and generalize better to unseen examples.

By leveraging the power of deep learning models, such as ResNet-50 and InceptionV3, image forgery detection of the splicing and copy-move variants can achieve high accuracy and reliability. The models' ability to learn complex representations and capture subtle inconsistencies makes them effective tools for maintaining image integrity and authenticity in various applications.

In conclusive image forgery detection using machine learning and deep learning techniques has emerged as a promising approach to address the challenges posed by the widespread availability of image editing tools. These techniques offer automated and efficient solutions to detect various forms of image manipulation, ensuring the integrity and authenticity of digital imagery. With ongoing advancements in algorithms and the availability of large-scale annotated datasets, the field of image forgery detection is poised to make further strides in combatting digital image forgery.

1.4 AI generated faces detection

AI-generated faces refer to synthetic images of human-like faces that are created using artificial intelligence algorithms. These algorithms can generate highly realistic and detailed facial images that can easily deceive human observers. Detecting AI-generated faces has become a crucial task in deep learning. Deep learning techniques, such as convolutional neural networks (CNNs), have been employed to develop.

In this project we focused on StyleGAN model and detecting the faces that it generates.

StyleGAN (Style Generative Adversarial Network) is a popular deep learning model architecture that is designed to generate realistic and high-quality images, specifically focusing on the generation of human faces. It was introduced by NVIDIA researchers in 2018. More information about it will be discussed later.

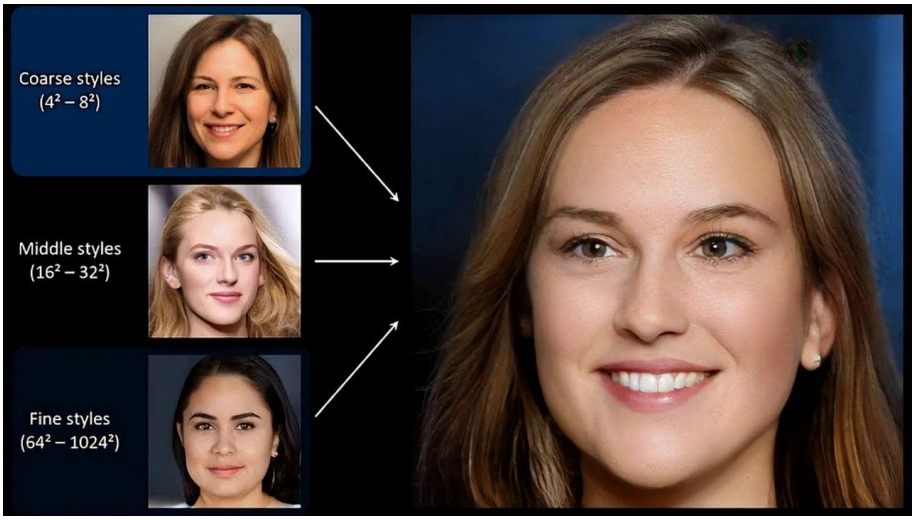


Figure 15Ai generated faces by StyleGAN

We also used for this task a dataset of 140k real and fake faces which contain 70k real faces from the Flickr dataset collected by Nvidia, as well as 70k fake faces sampled from 1 million fake faces generated by style GAN (generative adversarial networks). More details about it will be presented in the next section.

CHAPTER 2: PREVIOUS WORK

In this chapter we will introduce some of the previous techniques used for image forgery detection. We will categorize these techniques into techniques based on traditional machine learning, and techniques based on deep learning.

2.1Machine learning methods

2.1.1 Random Forest ML Algorithm

In this paper, the A. Doegar at. al [15] propose a method of detecting cloning and copy-move image forgery.

Architecture: The proposed approach for detecting cloned or copy-move forgery in images. Random Forest, a widely used machine learning algorithm, is employed for classification. The approach incorporates k-fold cross-validation with a value of 5 to divide the dataset into training and testing sets, while feature extraction is performed using GoogleNet before training the Random Forest model.

Dataset: The authors use the MICC-F220 dataset, which includes 110 non-forged and 110 forged images.

Results: The method achieved an accuracy of 89.55%, precision of 85.95%, true positive rate of 94.54%, and false positive rate of 15.45%.

Figure 16 original image

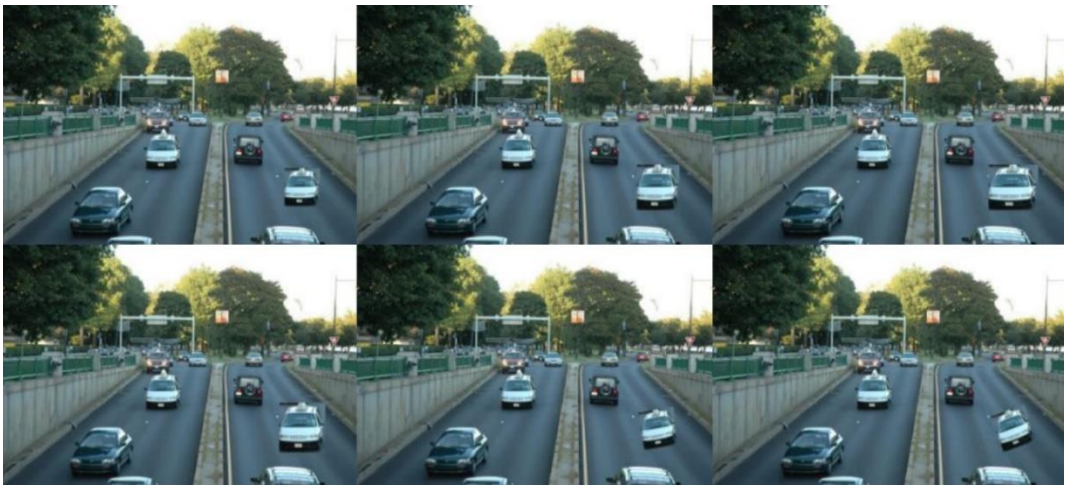


Figure 17 tampered images.

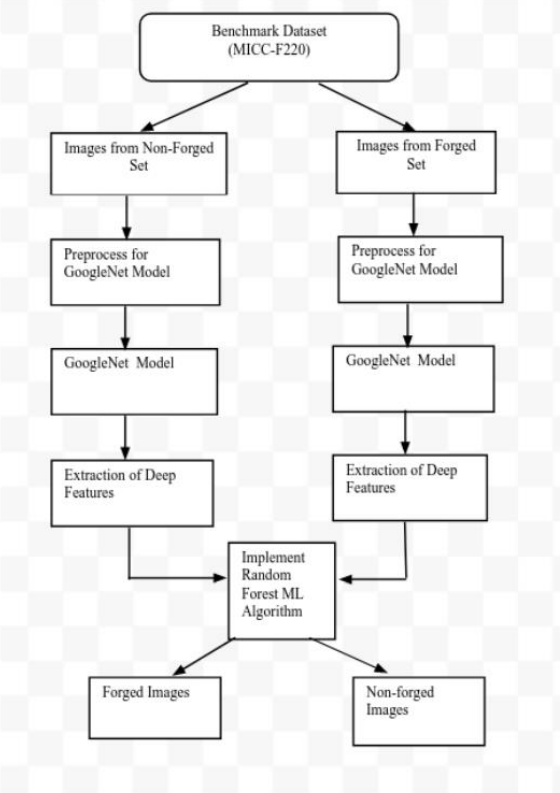


Figure 18 flow for google net and random forest algorithm.

2.1.2 A Passive Blind Approach for Image Splicing Detection Based on DWT and LBP Histograms

The paper by Mandeep et. Al [22], proposes a passive-blind approach for detecting image splicing, which is a usual form of forgery in digital images.

Architecture: The paper proposes a passive-blind approach for detecting image splicing using a combination of Discrete Wavelet Transform (DWT) and Local Binary Pattern (LBP) histograms. The image is decomposed using single-level DWT, and texture variation is studied in the detailed and approximation coefficients using LBP histograms. The LBP histograms from the four wavelet sub-bands are concatenated to form a feature vector. An SVM classifier is employed for classification.

Dataset: The method is tested on various standard spliced image databases, including CASIA TIDE v1.0, CASIA TIDE v2.0, Columbia spliced, and Columbia uncompressed.

Results: The proposed method achieves high accuracy, up to 97%, for JPEG images in the spliced image dataset. The accuracy is evaluated using 10-fold cross-validation. The performance of the algorithm is compared with other state-of-the-art methods, and the results indicate promising detection of splicing operations, particularly in JPEG images.

2.1.3 A Robust Forgery Detection Method for Copy–Move and Splicing Attacks in Images

The article Mohammad ET. Al. [21] wrote describes a proposed method for detecting forged images, with a special emphasis on efficient feature extraction methods.

Architecture: The proposed method for detecting forged images includes a feature extraction pre-processing stage and a classification step. The system utilizes the DCT-LBP method, which applies DCT (Discrete Cosine Transform) followed by LBP (Local Binary Pattern) to amplify modifications. The YCbCr color space is chosen for its superior performance, and the method can be applied to both grayscale and color photos. Square blocks are used for feature selection and dimensionality reduction, and the BDCT (Block-based Discrete Cosine Transform) technique is employed to identify local frequency distribution changes in tampered areas.

Dataset: Colombia grey, Colombia color, CASIA1, CASIA2, and FBDDF.

Results: The forgery detection approach achieved the following detection accuracies for gray scale images: 86%, 96%, 99%, 99%, and 96% for the Columbia Gray, Columbia Color, CASIA 1, CASIA 2, and FBDDF datasets, respectively. For color images, the detection accuracies were even higher: 98%, 100%, 100%, and 100% for the corresponding datasets. These results show the forgery detection approach's effectiveness in accurately identifying forged areas in gray scale and color images.

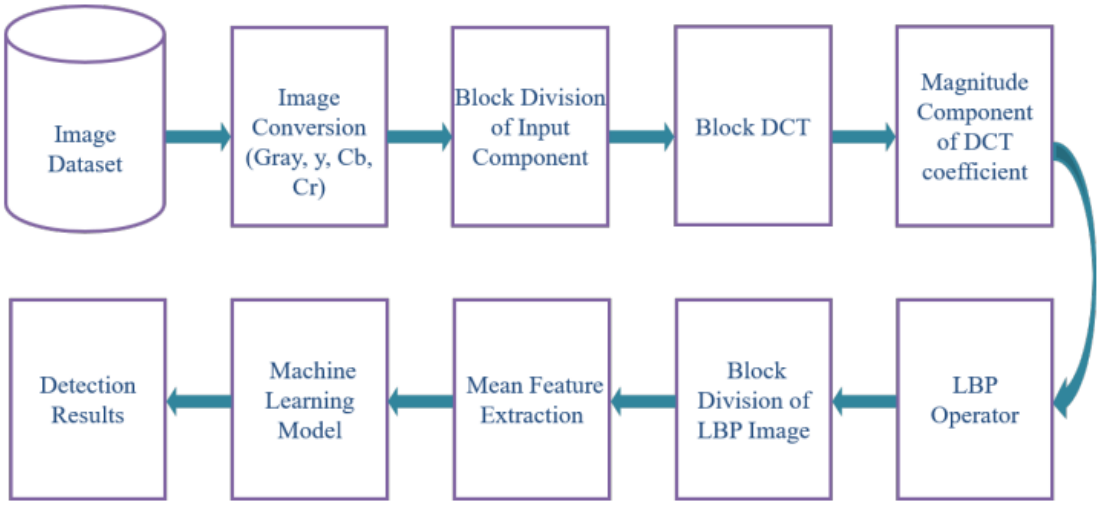


Figure 19 DCT LBP detection technique flow.

2.2Deep learning methods

2.2.1 Image Region Forgery Detection: A Deep Learning Approach.

In this paper, Zhang. Y et.al [16] inspected digital image forgery using a deep learning method.

Architecture: The paper proposes a deep learning method for detecting digital image forgery. It utilizes a two-stage approach, where the image is first converted into the YCrCb color space, known for its sensitivity to tampering. In the first stage, a Stacked Autoencoder (SAE) model is used to learn complex features for each individual patch

of the image. In the second stage, contextual information from each patch is integrated to improve the accuracy of detection.

Dataset Used: CASIA

Results: The new methodology presented in the paper achieves an overall accuracy of 91% in detecting forged images. It addresses the limitations of previous methods by detecting multiple tampering techniques and identifying the tampered regions. The accuracy of the proposed method is reported to be 91.09%. The paper is deemed useful for a project that aims to utilize machine learning and deep learning models for detecting tampered images.

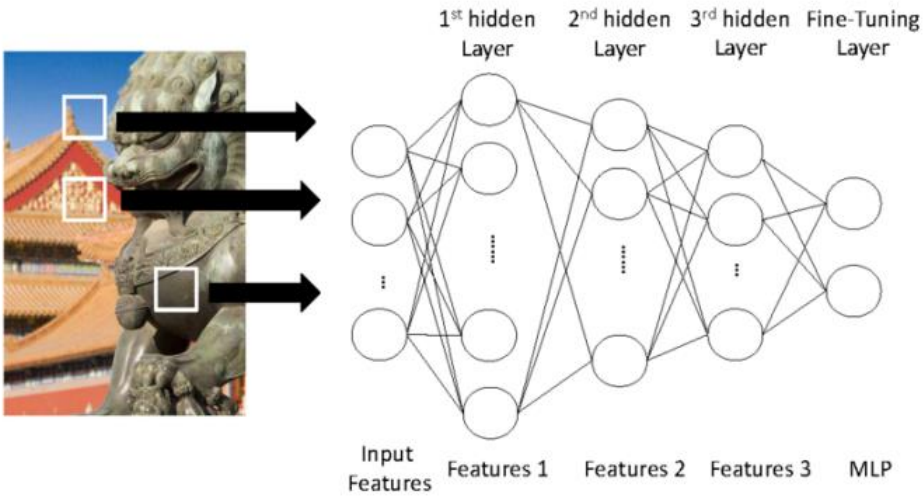


Figure 20 feature extraction and mapping.

2.2.2 Deep convolution neural network and semantic segmentation

In this paper the authors Abhishek & Jindal. N [1] created a deep convolution neural network and semantic segmentation system to detect image forgeries and the forged part of the image.

Architecture: The authors created a deep convolutional neural network (DCNN) and semantic segmentation system for detecting image forgeries and identifying the forged parts of the image. They employed color illumination for color mapping and trained the VGG-16 model with two classes to classify each pixel as forgery or non-forgery.

Dataset Used: The authors utilized publicly available datasets including GRIP, DVMM, CMFD, and BSDS300. The total dataset consisted of 47,566 images, with 28,540 images used for training and 19,026 images used for testing.

Results: The resulting algorithm achieved high accuracy in detecting forged pixels and non-forged areas, with an accuracy above 98%. The visualization of the results used white color for forged pixels and black color for non-forged areas. Figure 11 provides examples of forgery detection results, including the input forged image, image after color illumination, overlay image, and the detected forgery.

Figure 21 extracting mask for forged region.

2.2.3 Automated image splicing detection using deep CNN-learned features and ANN-based classifier.

S. Nath and R. Naskar [5] propose a deep learning approach for automated image splicing detection, which uses a convolutional neural network (CNN) to learn features from image patches and an artificial neural network (ANN) classifier to distinguish between spliced and authentic images.

Architecture: The proposed deep learning approach for automated image splicing detection utilizes a convolutional neural network (CNN) for feature learning from image patches and an artificial neural network (ANN) classifier to distinguish between spliced and authentic images. The authors describe the CNN architecture, feature extraction, and classification process in detail.

Dataset Used: The Casia 2.0 dataset is used, which consists of 12,614 color images. Out of these, 7,491 images are authentic, while the remaining 5,123 images are forged. The backbone of the network is based on the ResNet-50 architecture.

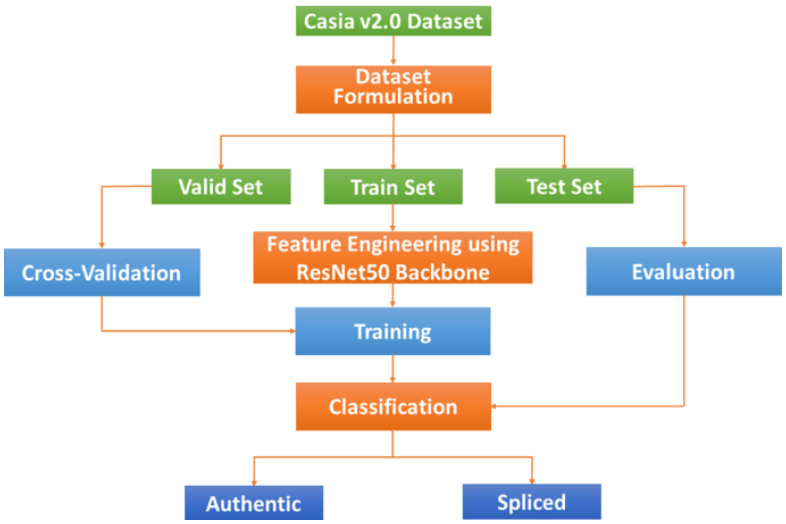


Figure 22 ResNet50 training phase.

Results:

Table 1

	Accuracy	Precision	Recall	F1 score	AUC score
Experiment 1	0.9645	0.9669	0.9415	0.9540	0.9755
Experiment 2	0.9208	0.9244	0.9167	0.9205	0.9525

2.2.4 An efficient method for image forgery detection based on trigonometric transforms and deep learning.

In this paper the authors F.M. Al_Azrak et. al [17] tackled the image forgery problem with innovative techniques.

Architecture: The authors present two methods for image forgery detection. The first method involves dividing the image into overlapping blocks and applying trigonometric transforms such as DFT (Discrete Fourier Transform), DCT (Discrete Cosine Transform), and DWT (Discrete Wavelet Transform). Features are then extracted from each block. The second method is based on deep learning, where multiple filters are applied to the image during the training process. The image goes through convolutional layers, max pooling, global average pooling, dropout layers for reducing overfitting, and a fully connected layer. A dense layer determines whether the image belongs to the original class (1) or tampered class (0).

Dataset Used: The authors used the MICC-F220 dataset for testing, which consists of 110 tampered images and 110 original images.

Results: The first method achieved an accuracy of 85%. However, by feeding the outputs of 2DDCT with DWT and 2DDFT with DWT to the CNN (Convolutional Neural Network), a 100% accuracy was achieved.

In summary, the authors propose two methods for image forgery detection. The first method utilizes trigonometric transforms on overlapping blocks, while the second method employs deep learning with multiple filters and a dense layer. The experiments conducted on the MICC-F220 dataset show promising results, with the second method

achieving a perfect accuracy of 100% when using specific combinations of transformations and feeding them to the CNN.

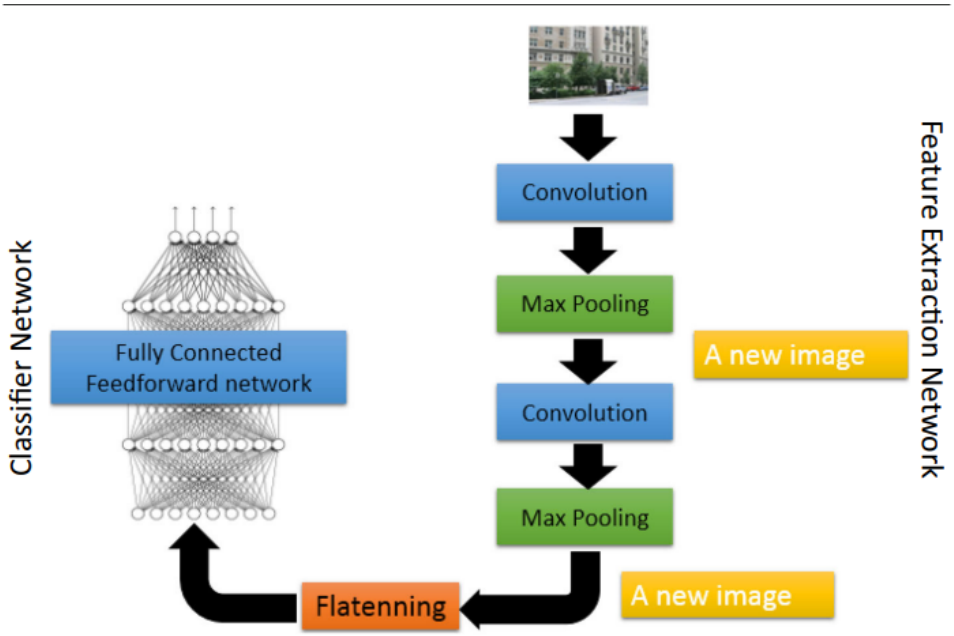


Figure 23 feature extraction network

2.2.5 An efficient copy moves forgery detection using deep learning feature extraction and matching algorithm.

In this paper the authors Agarwal. R et. al [14] tried to solve the image forgery problem with deep learning.

Architecture: The authors propose a deep learning-based approach for image forgery detection. The architecture includes a preprocessing phase where the image is segmented into similar patches using Simple Linear Iterative Clustering (SLIC), a clustering algorithm based on K-means. The VGGNet model is then employed to

extract multi-scale features from the segmented patches. The model consists of Convolution and Max-pooling layers. During training, the patches are inputted to the VGGNet model, which learns the features using stochastic gradient descent. The model is designed to detect copy move forgeries, even if the copied region is scaled or rotated. The depth of every pixel in the patches is reconstructed after feature extraction using the adaptive patch matching method, and the key points extracted are merged with segmented patches using adaptive patch matching (ADM) to detect the forged region.

Dataset Used: The authors utilized the MICC-F220 dataset to test their model. The dataset includes 110 forged images from the copy move type and 110 original images.

Results: The proposed method achieved an accuracy of 95% in detecting copy move forgeries on the MICC-F220 dataset.

In summary, the authors' deep learning-based approach utilizes SLIC for image segmentation, VGGNet for feature extraction, and adaptive patch matching for detecting forged regions. The method achieved a high accuracy of 95% in detecting copy move forgeries on the MICC-F220 dataset.

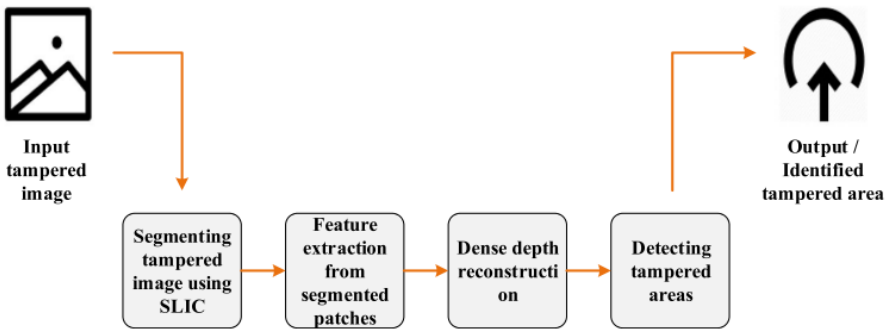


Figure 24 flow for tampering localization.

2.2.6 TransForensics: Image Forgery Localization with Dense Self-Attention

In this paper, the author, Hao. J et. al [13] use a fully convolutional network (FCN) for feature extraction, then self-attention encoders are used to model interactions between points in feature maps.

Architecture: The paper proposes the TransForensics method for image forgery detection. The architecture involves using a fully convolutional network (FCN) for feature extraction. Self-attention encoders are then utilized to model interactions between points in the feature maps. The authors address the limitations of other forgery detection techniques, such as hand-crafted methods and deep learning methods, by developing TransForensics. The method does not require prior knowledge of the forgery type.

Dataset Used: The authors utilized three datasets for their experiments: Casia, Coverage, and IMD2020. However, further details about these datasets are not provided in the given text.

Results: The specific results of the experiments conducted using the TransForensics method on the Casia, Coverage, and IMD2020 datasets are not mentioned in the given text.

Method	CASIA	COVER	IMD2020
ELA [20]	0.613	0.583	-
NOI1 [27]	0.612	0.587	-
CFA1 [14]	0.522	0.485	0.586
J-LSTM [3]	-	0.614	0.487
RGB-N [42]	0.795	0.817	-
BLK [23]	-	-	0.596
ADQ1 [25]	-	-	0.579
ManTra-Net [38]	0.817	0.819	0.748
LSTM-EnDec [4]	-	0.712	-
Ours (downsample)	**0.850**	**0.884***	0.847
Ours (upsample)	0.837	0.883*	**0.848**

Figure 25 results for TransForensics: Image Forgery Localization with Dense Self-Attention.

2.3 AI genrtaed faces detection methods

2.3.1 A Style-Based Generator Architecture for Generative Adversarial Networks

This paper by Tero Karras, Samuli Laine, and Timo Aila [18] proposes an alternative generator architecture for generative adversarial networks, borrowing from style transfer literature. The new architecture leads to an automatically learned, unsupervised separation of high-level attributes (e.g., pose and identity when trained on human faces) and stochastic variation in the generated images (e.g., freckles, hair), and it enables intuitive, scale-specific control of the synthesis.

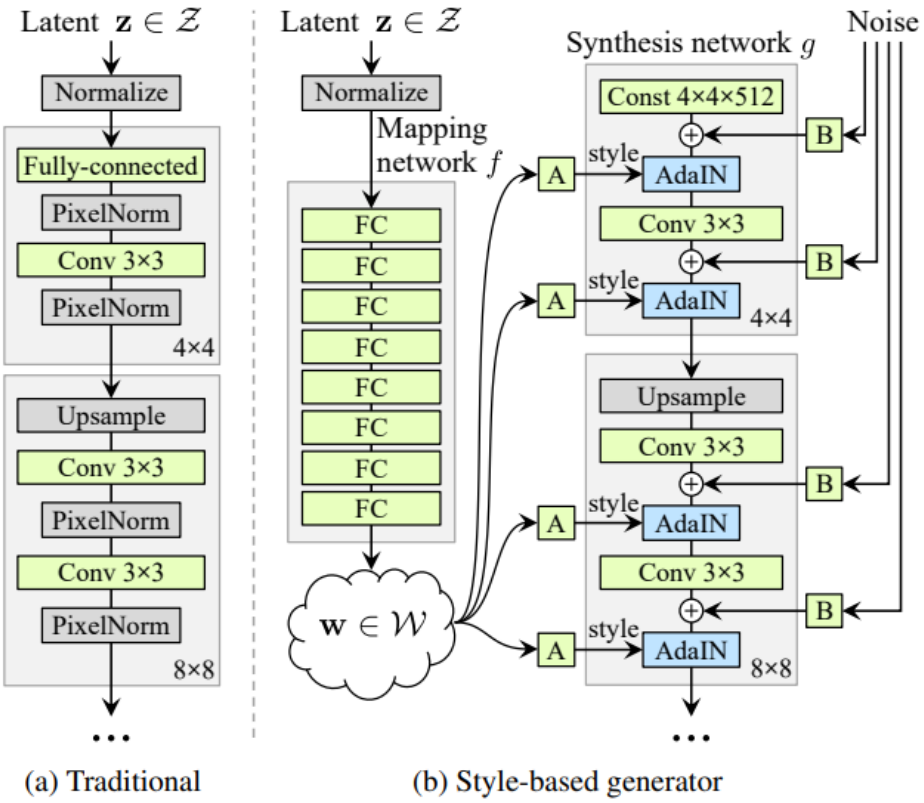


Figure 26 traditional vs style-based generator architectures

The new generator improves the state-of-the-art in terms of traditional distribution quality metrics, leads to demonstrably better interpolation properties, and better disentangles the latent factors of variation. To quantify interpolation quality and disentanglement, the authors propose two new, automated methods that are applicable to any generator architecture. Finally, they introduce a new, highly varied, and high-quality dataset of human faces.

2.3.2 Densely Connected Convolutional Networks

The paper by Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. It was published in 2016 [19] and introduces the Dense Convolutional Network (DenseNet) architecture. The paper shows that convolutional networks can be

deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. The DenseNet architecture connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections - one between each layer and its subsequent layer - DenseNet has L (L+1)/2 direct connections.

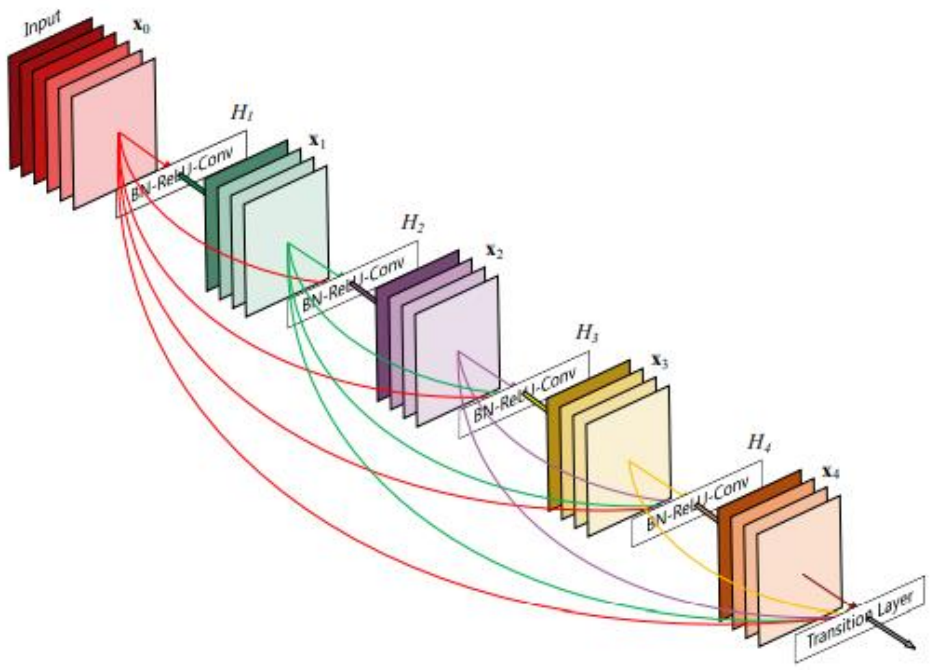


Figure 27 DenseNet architecture

DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and reduce the number of parameters. The authors evaluate their proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them while requiring less memory and computation to achieve high performance.

2.3.3 Challenges and Solutions in Deepfakes

The 140k dataset introduced in the paper [20] "Challenges and Solutions in Deepfakes" by Jatin Sharma and Sahil Sharma is a collection of real and fake faces. The dataset contains 70k real faces from the Flickr dataset collected by Nvidia, as well as 70k fake faces sampled from 1 million fake faces generated by style GAN. The dataset was created to help researchers develop deepfake detection algorithms.

The dataset is divided into two parts: the training set and the test set. The training set contains 120k images, and the test set contains 20k images. The training set is used to train deepfake detection algorithms, and the test set is used to evaluate the performance of the algorithms.

The authors of the paper evaluated the performance of several deepfake detection algorithms on the 140k dataset. The algorithms were able to detect deepfakes with an accuracy of up to 90%.

However, the authors also found that the algorithms were not able to detect all deepfakes. This is because the quality of deepfakes is constantly improving, and new techniques for creating deepfakes are being developed all the time.

The 140k dataset is a valuable resource for researchers who are working on deepfake detection. The dataset is large and diverse, and it can be used to train and evaluate deepfake detection algorithms. The dataset is also publicly available, which means it can be used by researchers worldwide.

Here are some of the benefits of using the 140k dataset:

The dataset is large and diverse, which means that it can be used to train and evaluate deepfake detection algorithms on a variety of diverse types of content.

The dataset is publicly available, which means that it can be used by researchers all over the world.

The dataset is well-curated, which means that it is free of noise and other artifacts that could interfere with the training and evaluation of deepfake detection algorithms.

In conclusion previous work focused on one or more image forgery types, using comparatively small datasets.

Table 2

Paper Title	Type of Forgery Detected	Dataset Used	Accuracy Achieved
A. Doegar et al. [15]	Cloning and Copy-Move Image Forgery	MICC-F220	89.55%
Mandeep et al. [22]	Splicing	Casia1, Casia2, Colombia	97%
Zhang et al. [16]	Digital Image Forgery	CASIA	91.09%
Abhishek & Jindal [1]	Image Forgeries and Forged Regions	GRIP, DVMM, CMFD, BSDS300	Above 98%
S. Nath and R. Naskar [5]	Image Splicing	Casia 2.0	Experiment 1: 96.45%, Experiment 2: 92.08%
F.M. Al_Azrak et al. [17]	Copy-Move and Splicing Forgeries	MICC-F220	Method 1: 85%, Method 2: 100%
Agarwal. R et al. [14]	Copy-Move Forgeries	MICC-F220	95%
Hao. J et al. [13]	Image Forgery Detection	Casia, Coverage, IMD2020	88%
M. M. Islam et al. [21]	Splicing and Copy move detection	Colombia grey, Colombia color, CASIA1, CASIA2, and FBDDF	For gray scale images: Columbia Gray: 85.56% Columbia Color: 96.10% CASIA 1: 98.61% CASIA 2: 99.29% FBDDF: 95.84% For color images: Columbia Gray: 98.20% Columbia Color: 99.55% CASIA 1: 99.88% CASIA 2: 100.00%

CHAPTER 3: IMPLEMENTED DETECTION TECHNIQUES

In this chapter we write about the various unsuccessful methods we tried and the final methods that proved to detect image forgeries with satisfactory accuracy.

Many models were used and tried to get maximum accuracy in this task. Among the models used in trial for splicing and copy move detection are regular CNN deep learning methods based on scholarly papers, EffnetB0 – B7 with various additional layers and architectures, resent 152 with various additional layers and architectures, DenseNet with various additional layers and architectures. In the end we settled on ResNet50 with additional layers for classification for splicing detection and InceptionV3 for copy move detection. Below there is a comparison between the different methods used throughout the project.

Splicing:

Table 3

Method used	accuracy	dataset
DenseNet	50%	Defacto Splicing
ResNet50	93%	Defacto Splicing
ResNet152	62%	Defacto Splicing

Copy move:

Table 4

Method used	accuracy	dataset
DenseNet	53%	Defacto Copy Move
ResNet50	75%	Defacto Copy Move
ResNet152	86%	Defacto Copy Move
Inception V3	93%	Defacto Copy Move
EffNetB0	70%	Defacto Copy Move

Splicing and Copy move:

Table 5

Method used	accuracy	dataset
SVGGNet	84%	CasiaV2
CNN	59%	CasiaV2
ResNet50		
EffNetB4	48%	CasiaV2
EffNetB5	81%	CASIA 1&2 +Colombia + MICCF2000 + IMD 2020 + DeFacto splicing
EffNetB7	70%	CasiaV2

3.1 Using machine learning (DCT-SVM)

3.1.1 Image forgery detection based on DCT and SVM for copy-move and splicing.

3.1.1.1 Architecture

We tried the methodology of a few papers, most notably Mohmed et. Al with an accuracy of 96%, Arman et. al with an accuracy of 95% and Mandeep et. Al with an accuracy of 97%

We settled on using the feature extraction method from Shipla et. Al (2020) due to its higher accuracy of 98%.

In that paper, the Image is transformed to YCBCR color space and for each sub image the following is done:

- Divided into 8x8 nonoverlapping blocks.
- 2d DCT operation is performed on each block and DC component is ignored and AC components are arranged into a vector.
- Getting the transpose of the vector and then concatenating all the vectors for all blocks into a matrix M.
- Then each row corresponds to the specific ac frequency across all blocks then standard deviation for each row is calculated and added to vector S.
- Sk =1.4826MAD (ak, i, ak, i+1⋯ak,N) is the standard deviation of kth AC coefficient of all blocks. In a comparable manner, the number of ones ok for each row are calculated and added to vector O.
- Signum function is applied to each row of the matrix.

- Obtain the cropped version of the given sub-image by removing the first 4 rows and 4 columns from top left corner. Then all the steps described above for uncropped version are repeated.
- To obtain the final feature vector (FV) of the candidate image, each of the feature vectors obtained through three sub-images (Y, Cb and Cr) is concatenated.

After the feature extraction an SVM model is built with the following parameters

C = 100, gamma = 0.001 and the kernel is RBF.

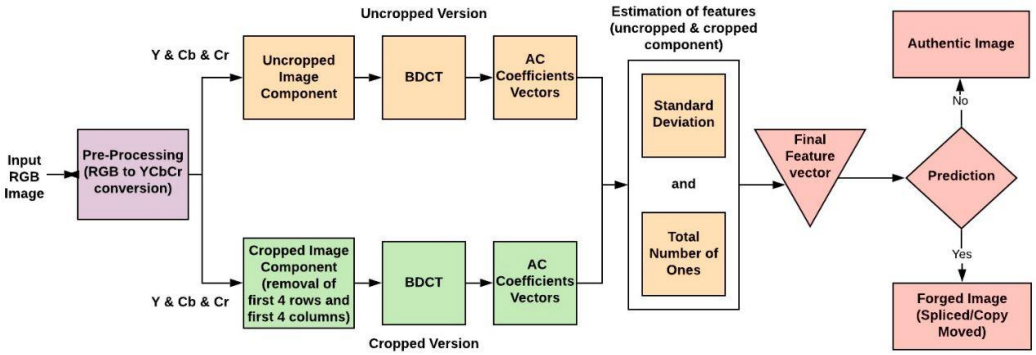


Figure 28 statistical coefficients of dct and svm flow chart

3.1.1.2 Datasets used:

CASIA v1.0, v2.0 and Colombia.

CASIA v1.0, v2.0 constructed by the Institute of Automation Chinese Academy of Sciences which is more challenging and realistic dataset for manipulation detection. In CASIA v1.0, there is a total of 1721 images in which 800 images are authentic and 921 images are tampered color images of size 384×256 and all are in JPEG format without any post-processing. On the other hand, CASIA v2.0 consists of multiple sized images with various post-processing applied across edges. CASIA v2.0 consists of 7491 authentic and 5123 forged color images with size varying from 240×160 to 900×600 pixels.

Meanwhile the Colombia dataset has 185 authentic images, and 182 spliced images.

Also, the images are available as uncompressed as well as in JPEG format with different quality factors.

3.1.1.3 Results:

In this project we used stranded metrics to measure the performance of our models. Mainly F1 score and accuracy.

We tried the methodology of a few papers, most notably Mohmed et. Al with an accuracy of 96%, Arman et. al with an accuracy of 95% and Mandeep et. Al with an accuracy of 97%. All models were run on the previously mentioned CASIA 1 & 2 and Colombia datasets.

We decided to work with Shilpa et. al because it has the highest relative accuracy and the largest image resolution and feature vector. This resulted in an accuracy of **98%**.

3.2. Deep learning techniques

3.2.1 InceptionV3 for copy move forgery detection.

3.2.1.1 Architecture

InceptionV3 is a convolutional neural network (CNN) architecture built as part of the Inception series by Google researchers. It is mostly utilized in computer vision applications for image categorization and recognition.

The word "Inception" refers to the use of "inception modules" within the network architecture. These modules are made up of numerous parallel convolutional layers of varying sizes, which allows the network to capture features at various scales. This aids in the learning and representation of complicated patterns in imagery.

InceptionV3 improves on previous versions of the Inception architecture by introducing various enhancements to boost performance and accuracy. To process and classify images, it employs a combination of convolutional layers, pooling layers, and fully connected layers.

One distinguishing characteristic of InceptionV3 is the use of "factorization" or "bottleneck" layers, which try to lower the network's processing complexity. These bottleneck layers minimize the number of input channels before applying larger convolutional filters, resulting in fewer calculations while retaining the network's representational capacity.

The use of auxiliary classifiers in intermediate layers is another important feature of InceptionV3. These auxiliary classifiers aid in overcoming the vanishing gradient problem and provide additional regularization during training, boosting the model's capacity to effectively generalize and categorize images.

InceptionV3 was trained using the ImageNet dataset, which comprises millions of labelled images from thousands of distinct categories. This pre-training enables the network to acquire a diverse range of features, which can then be fine-tuned on specific picture classification tasks or transferred to other related tasks.

Overall, InceptionV3 has outperformed numerous image classification benchmarks, making it a preferred choice for many computer vision applications such as object recognition, scene understanding, and picture retrieval.

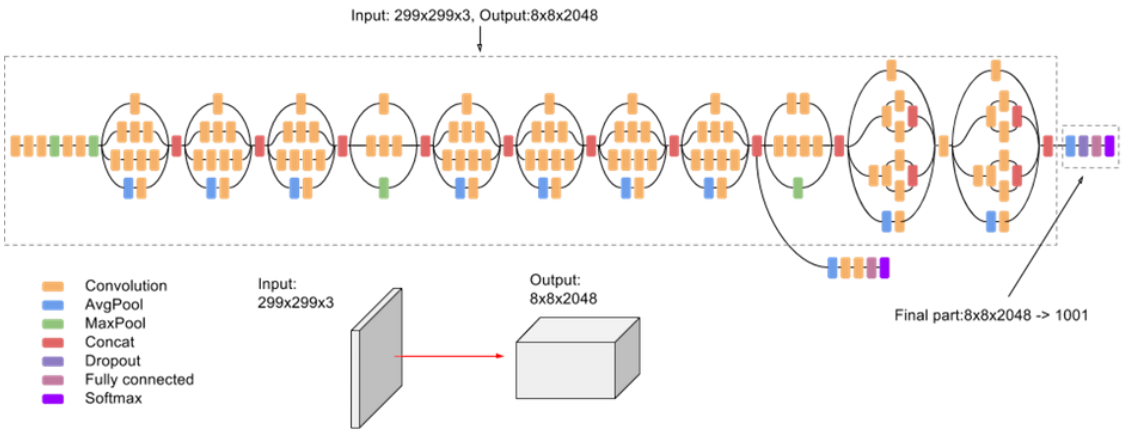


Figure 29 inceptionV3 architecture

3.2.1.2 Dataset:

For copy move detection we used the Defacto dataset which contains 18,000 copy move forged images augmented with 18,000 images from COCO2014 for authentic images.

3.2.1.3 Results:

With a train test validation split of 80% 10% and 10% respectively, the test batch was used with the evaluation method of Keras and the result was an accuracy of 90%. The training was done on the Linux Ubuntu operating system with GPU acceleration.

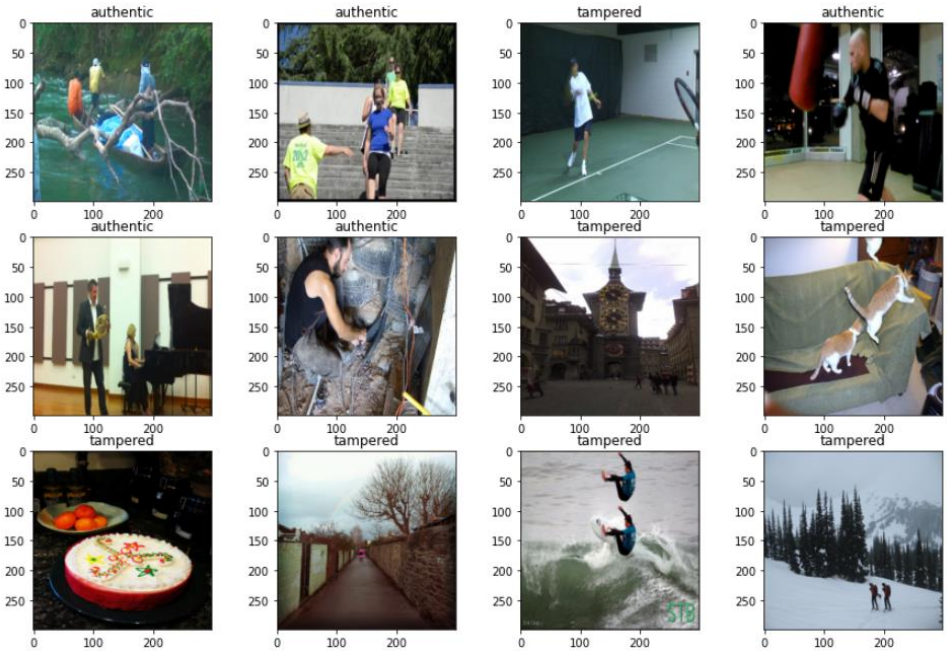


Figure 30 copy-move results

3.2.2 ResNet50 for splicing

3.2.2.1 architecture

The ResNet-50 architecture, short for Residual Network with 50 layers, is a popular deep convolutional neural network (CNN) architecture widely used for various computer vision tasks, including image forgery detection. Its deep structure enables it to learn intricate patterns and features from images, making it suitable for detecting complex forms of forgery, such as splicing.

The ResNet-50 architecture consists of multiple layers, including convolutional layers, pooling layers, and fully connected layers. It introduces the concept of residual connections, which mitigate the degradation problem encountered in training very deep networks. These connections allow the network to learn residual mappings, making it easier to optimize the network's performance as it becomes deeper.

In the context of image forgery detection, particularly the splicing variant, the ResNet-50 architecture can be adapted and fine-tuned to effectively identify the irregularities and inconsistencies in spliced images. Here is a high-level overview of the architecture's components:

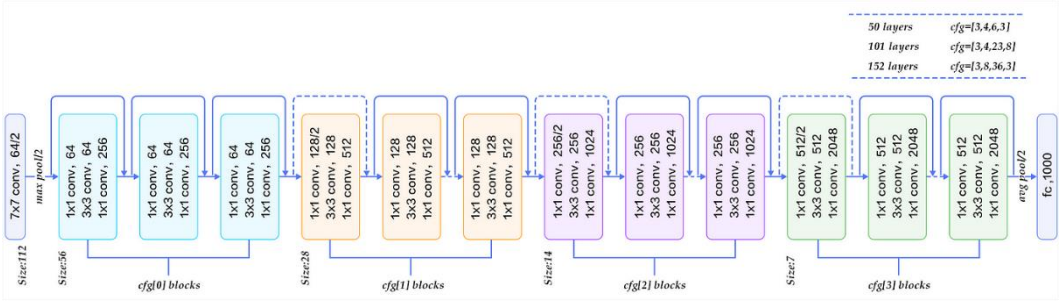


Figure 31ResNet50 architecture

- **Input Layer:** The network's input layer accepts the image data, typically in the form of a fixed-sized tensor.
- **Convolutional Layers:** A series of convolutional layers perform feature extraction by applying filters to the input image. These layers detect various low-level and high-level features, such as edges, textures, and shapes, which are crucial for detecting splicing in forgery images. The ResNet-50 architecture includes multiple convolutional layers stacked together, allowing for the extraction of increasingly complex features.
- **Residual Blocks:** The distinctive aspect of the ResNet-50 architecture is the presence of residual blocks. A residual block consists of multiple convolutional layers with shortcut connections that bypass one or more convolutional layers. These shortcut connections help propagate gradient information more effectively during training, enabling the network to learn deeper and more accurate representations. Residual blocks aid in capturing the fine-grained irregularities and inconsistencies that are characteristic of splicing forgery.
- **Pooling Layers**: Pooling layers sample the feature maps down, reducing their spatial dimensions while retaining the most salient information. This helps in reducing computational complexity and controlling overfitting.
- **Fully Connected Layers:** Towards the end of the network, fully connected layers are employed to learn high-level representations of the extracted features. These layers capture global context and relationships between distinctive features. They transform the feature maps into a vector representation that can be used for classification.

- **Output Layer:** The output layer consists of one or more neurons, depending on the number of classes or forgery types to be detected. In splicing forgery detection, the output layer comprises two neurons, representing the classes of authentic and spliced images, respectively. The output is passed through an appropriate activation function (e.g., SoftMax) to produce the probability distribution over the classes.
- **Flatten layer:** The ResNet-50 model typically ends with a convolutional layer, producing a 3-dimensional output tensor. However, for the subsequent dense layers, we need a 1-dimensional vector input. The flatten layer reshapes the output from the previous layer into a 1-dimensional vector, enabling compatibility with the dense layers.
- **Dense layers with ReLU activation:** The dense layers play a crucial role in learning complex patterns and representations from the flattened input. By using a high number of units (e.g., 1000), these layers can capture intricate features related to image forgery. The ReLU activation function introduces non-linearity, enabling the model to learn and represent more complex relationships between the input and the target.
- **BatchNormalization layers:** Batch normalization is applied after the dense layers to normalize the activations. This technique helps in stabilizing the training process and allows the network to converge faster. It reduces the internal covariate shift and makes the optimization more efficient.

- **Dropout layers:** Dropout is a regularization technique used to prevent overfitting. By randomly setting a fraction of the units to 0 during training, dropout forces the network to learn more robust and generalized features. In the context of splicing image forgery detection, dropout helps the model to avoid relying too heavily on specific units or features that might be indicative of the training set but not generalize well to new images.
- **Final dense layer with sigmoid activation:** The last dense layer with a single unit and sigmoid activation function is used to produce a binary classification output. It predicts the probability of the input image being a forgery. The sigmoid activation function squashes the output to the range [0, 1], where values closer to 1 indicate a higher likelihood of forgery, while values closer to 0 indicate a lower likelihood.

Additional layers were introduced for better classification.

```
x = layers.Flatten()(last_output)
x = layers.Dense(1000, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = layers.Dense(1000, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(500, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(500, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = layers.Dense(1, activation='sigmoid')(x)
```

Figure 32 added layers for ResNet50 backbone.

In summary, the chosen layers and their configurations aim to create a deep learning model that can effectively learn and detect splicing image forgery. The dense layers with ReLU activations learn complex representations, batch normalization normalizes activations, dropout regularization prevents overfitting, and the final sigmoid layer produces a probability-based forgery prediction. Together, these layers form a deep learning pipeline optimized for splicing image forgery detection.

3.2.2.2 dataset:

During the training phase, the ResNet-50 architecture is fine-tuned using a large dataset of labeled images called Defacto splicing, available on Kaggle, it consists of 212,000 images split evenly between authentic and tampered, comprising both authentic and spliced examples. The network learns to identify the discriminative features that differentiate between authentic and spliced images, enabling it to make accurate predictions on unseen images.

Some dense layers were added to increase the depth of the network. Layers such as Batch normalization and dropout were also added to avoid over fitting.

By leveraging the deep structure and residual connections, the ResNet-50 architecture demonstrates the capability to capture and analyze complex visual patterns, making it a powerful tool for image forgery detection, specifically in the context of splicing forgery.

3.2.2.2 Results:

For splicing detection, we used, as previously mentioned, Resnet50 with additional layers. With a train test validation split of 80% 10% and 10% respectively, the test batch was used with the evaluation method of Keras and the result was an accuracy of 93.37%. The training was done on the Linux Ubuntu operating system with GPU acceleration.

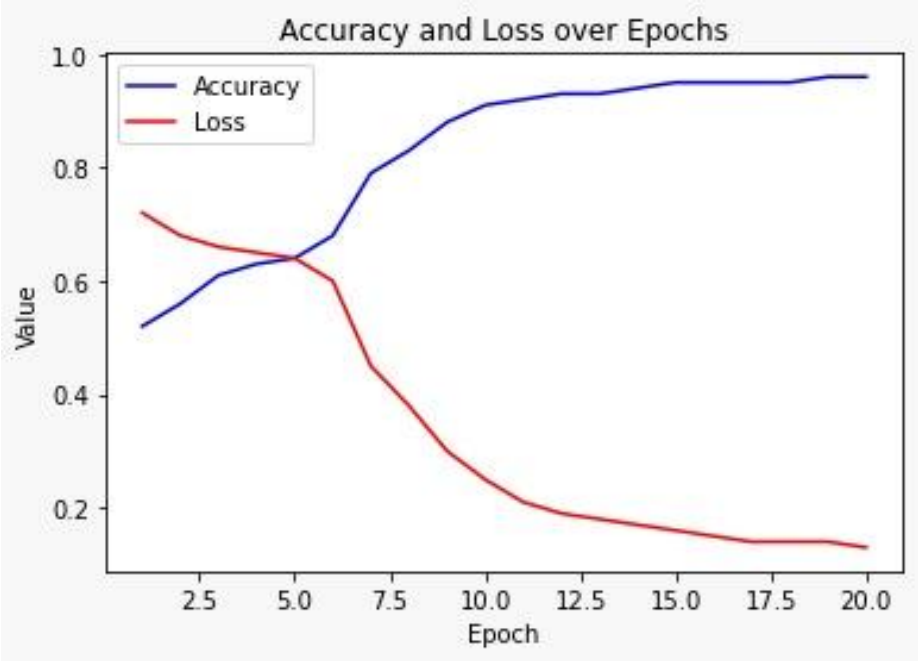


Figure 33 ResNet50 results for splicing

3.2.3 DenseNet for AI gen faces

3.2.3.1 architecture

In the context of AI-generated face detection, the DenseNet architecture can be utilized as a powerful deep learning model for effectively identifying AI-generated faces. DenseNet's unique connectivity pattern and feature reuse mechanism makes it particularly suitable for capturing the subtle differences and intricate patterns that distinguish AI-generated faces from real ones.

The DenseNet architecture consists of densely connected blocks of convolutional layers. Each block receives feature maps not only from its preceding layer but also from all preceding layers. This dense connectivity promotes information flow and encourages the network to learn highly discriminative features from different scales and levels of abstraction.

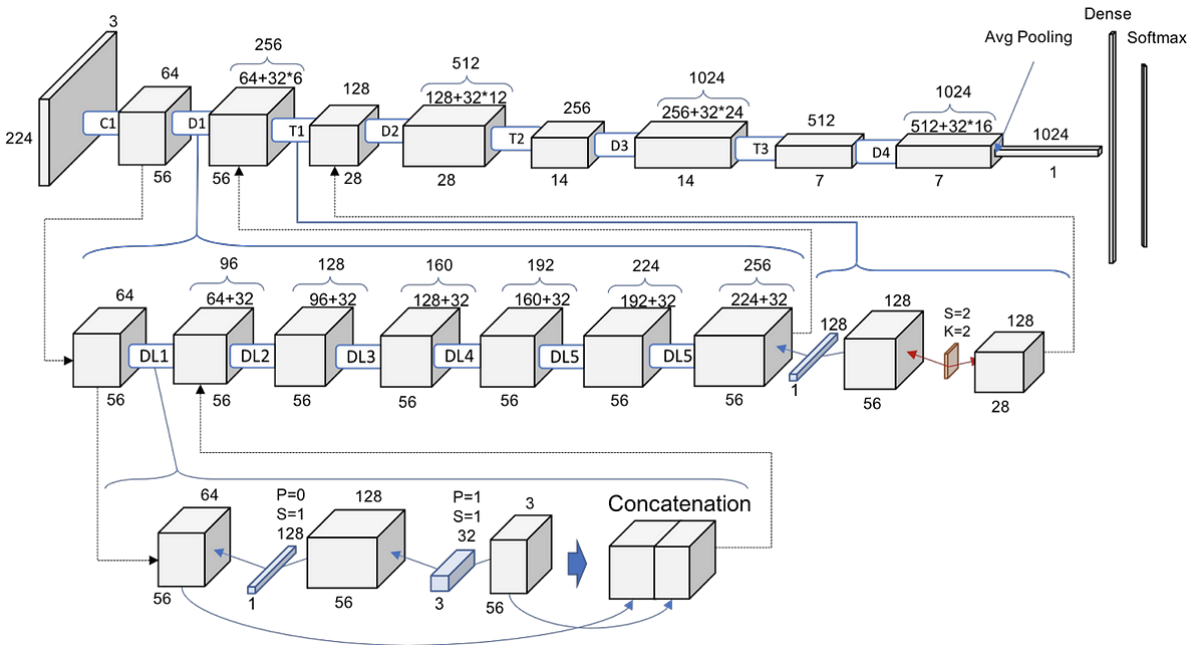


Figure 34 DenseNet architecture

To leverage DenseNet for AI-generated face detection, the architecture can be adapted and trained specifically for this task. The initial layers of the DenseNet can serve as a feature extractor, capturing generic facial features that are shared between real and AI-generated faces. These features are learned during pre-training on a large dataset of real faces.

The last few layers of the DenseNet, responsible for classification, are modified to accommodate the detection task. In this case, the network is trained to classify whether a given face image is real or AI-generated. This fine-tuning stage utilizes a new dataset that contains labeled samples of both real and AI-generated faces.

During training, the DenseNet model learns to recognize the unique features and patterns associated with AI-generated faces. By leveraging the dense connectivity and feature reuse, the model can effectively capture and exploit the intricate variations specific to AI-generated images.

Once the DenseNet is trained, it can be used to detect AI-generated faces in unseen images. By passing an image through the network, the model outputs a prediction indicating whether the face is likely to be real or AI-generated. This detection capability can aid in identifying instances of AI-generated faces, helping to address potential misuse and ethical concerns associated with such content.

3.2.3.2 Dataset used:

We used for this task the dataset introduced in [20] “Challenges and Solutions in Deepfakes” by Jatin Sharma and Sahil Sharma. In this paper, they introduce a dataset of 140k real and fake faces which contain 70k real faces from the Flickr dataset collected by Nvidia, as well as 70k fake faces sampled from 1 million fake faces generated by style GAN. They will train their model on the dataset so that their model can identify real or fake faces.

The 140k Real and Fake Faces dataset is a collection of facial images consisting of 140,000 samples. This dataset is specifically curated to include both real and fake faces, making it valuable for tasks related to facial image analysis, computer vision, and deep learning.

The dataset is designed to provide a diverse range of facial images, including various ethnicities, genders, ages, and facial expressions. It covers a wide spectrum of human appearances, enabling researchers and developers to explore and address different challenges in face recognition, facial expression analysis, age estimation, and other related applications.

The real faces subset of the dataset comprises authentic facial images captured from real individuals, representing genuine expressions, poses, and variations in appearance. These images are obtained from various sources, such as public image repositories, licensed databases, or social media platforms. The real faces subset serves as a benchmark for evaluating the performance of face-related algorithms in real-world scenarios.

On the other hand, the fake faces subset contains synthetic or artificially generated facial images. These images are created using advanced techniques like generative adversarial networks (GANs) or deep learning models. The fake faces subset is specifically included to address the growing concern of synthetic media, including deepfakes, and to facilitate research on detecting and combating such manipulations.

The dataset provides a balanced distribution between real and fake face images, ensuring that both categories are adequately represented. This balance enables researchers to develop and evaluate algorithms that can effectively differentiate between real and fake faces, contributing to the advancement of facial image forensics and authenticity verification techniques.

3.2.3.2 results:

Detecting AI generated faces using DenseNet required us to only focus on one model, StyleGAN, and in that task the model excels, with an accuracy of 98%.

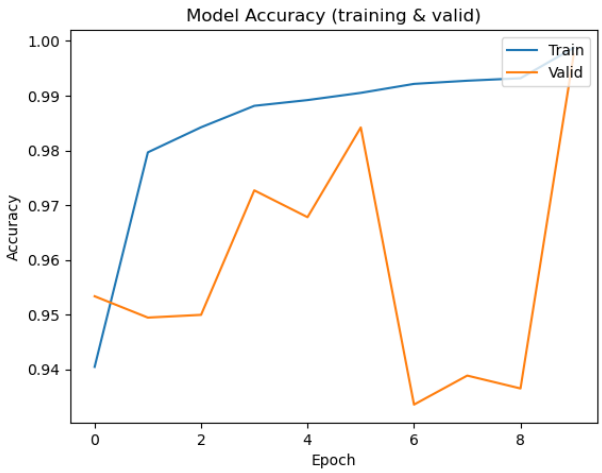


Figure 35 AI faces detection accuracy

Figure 36 detection of AI gen faces

3.3 Web Application

The user interface for this project is a web application written using HTML, CSS, and Flask.

The website presents the user with two options: Machine Learning (Done in part 1 of the graduation project), and Deep learning. There is a button to choose the image which allows the user to browse for an image on their device. Once an image is selected the user can click “Submit” and the server will begin running the code. The website then displays the result, that the image is either forged or authentic. In the case of the deep learning model, if the image is forged, the localized forgery area will be displayed.

The architecture of the application is:

- Index.html file, which houses the basic text elements.
- Style.css file, which has all the styling for the application.
- Server.py file, which contains the code for handling user requests and responses, and for machine learning and deep learning models.

The decision to use Flask was made to have ease of communication with the models, as Flask is written using Python, which makes it possible to run the models directly on the server. This simplifies the architecture of the application as we can get the request from the user on the front-end, and directly pass it to the models on the server. Then the response can be fetched directly back to the front-end.

This is done using a simple GET request from the front-end which contains the image uploaded by the user. The server catches the request, then runs the code, and returns the response to the front-end.

The CSS of the website uses new features such as Flexbox to have the website be easily responsive on smaller screen sizes, like mobile phones and tablets.

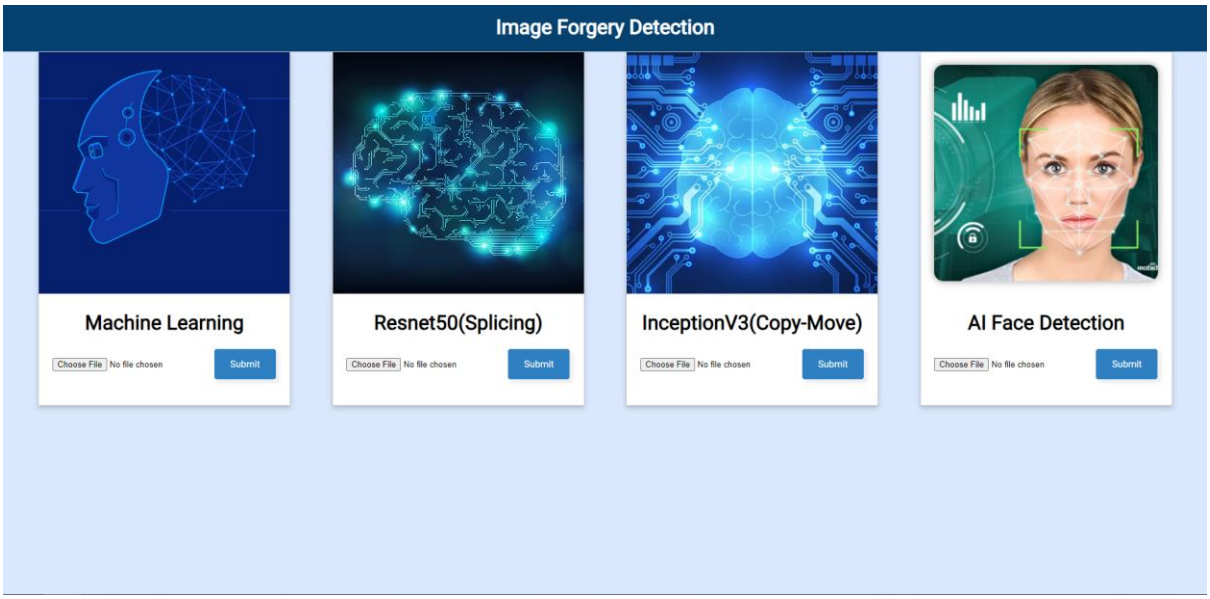


Figure 37 website interface

CHAPTER 4: CONCLUSION AND FUTURE WORK

4.1 Conclusion

In conclusion, our graduation project focused on the detection of image forgery, specifically targeting the splicing and copy move variants. We utilized state-of-the-art deep learning models, ResNet50 for splicing detection and InceptionV3 for copy move detection. With the ResNet50 model, we achieved an impressive accuracy of 93% in identifying splicing forgeries, while the InceptionV3 model yielded a commendable accuracy of 90% in detecting copy move manipulations.

Moreover, we expanded our project by addressing an additional challenge: detecting AI-generated faces from the StyleGAN Model. Leveraging the power of DenseNet, we successfully achieved an outstanding accuracy of 98% in identifying these synthesized images.

To make our project more accessible and user-friendly, we integrated these detection algorithms into a website. This allows users to easily upload images and obtain instant feedback on potential forgeries. The website serves as a valuable tool for experts and non-experts in image forensics, aiding in the identification and prevention of digital image tampering.

In addition to deep learning approaches, we also employed traditional machine learning techniques to tackle copy move and splicing forgery detection. By utilizing the Discrete Cosine Transform (DCT) and a Support Vector Machine (SVM) classifier, we achieved a respectable accuracy of 98% in distinguishing manipulated regions within images.

To ensure the reliability and generalizability of our models, we conducted extensive evaluations on various datasets. These included the widely recognized DeFacto dataset for splicing and copy move, as well as CASIA 1 and 2, Colombia, MICF220, and IMD2020 datasets. Our models demonstrated consistent performance across these diverse datasets, proving their effectiveness in real-world scenarios.

In summary, our graduation project successfully addressed the challenging task of detecting image forgeries, encompassing splicing, copy move, and AI-generated faces. By leveraging deep learning models and traditional machine learning techniques, we achieved high accuracies across diverse types of manipulations. The integration of our algorithms into a user-friendly website enhances the accessibility of our work and provides a valuable resource for image forensics. Overall, our project contributes to the field of digital image forensics and reinforces the importance of developing robust tools to combat image tampering in today's digital age.

4.2 Future Work:

Although our graduation project made significant advancements in the detection of image forgeries, there are several potential areas for future research and improvement. Future work can focus on localization and detecting more general AI-generated images.

Localization of Manipulated Regions: Our current models provide accurate detection of splicing and copy move manipulations, but they do not provide precise localization of the forged regions within an image. Future work can explore techniques to improve the localization capabilities, allowing users to pinpoint the exact regions where the

manipulation has occurred. This would provide more detailed information and assist forensic experts in analyzing and investigating the tampered images.

Detection of Advanced AI-Generated Images: While we achieved a remarkable accuracy of 98% in identifying AI-generated faces from the StyleGAN Model, it is crucial to expand the scope of our research to detect other types of AI-generated images as well. As AI techniques continue to evolve rapidly, future work can focus on developing models that can effectively detect deepfake videos, text generation, and other advanced AI-generated content. This would help combat the increasing sophistication of digital forgeries and provide a more comprehensive solution for image forensics.

Robustness and Generalization: Our models demonstrated consistent performance across diverse datasets, indicating their effectiveness in real-world scenarios. However, it is essential to further investigate the robustness and generalization capabilities of the models. Future research can explore the impact of different lighting conditions, image resolutions, compression artifacts, and other environmental factors on the detection accuracy. Additionally, expanding the evaluation to include more comprehensive and challenging datasets can help validate the models' performance in various practical settings.

By addressing these areas of future work, we can advance the field of digital image forensics and develop more robust and effective tools to combat the ever-evolving challenges of image tampering in the digital age.

REFERENCES

1. Abhishek and N. Jindal, "Copy move and splicing forgery detection using deep convolution neural network, and semantic segmentation" Multimedia Tools and Applications, vol. 80, no. 3, pp. 3571-3599, 2020. doi: 10.1007/s11042-020-09816-3.
2. Rani, A. Jain, and M. Kumar, "Identification of copy-move and splicing based forgeries using advanced SURF and revised template matching," Multimedia Tools and Applications, vol. 80, no. 16, pp. 23877-23898, 2021.
3. J. Xu, D. Feng, J. Wu, and Z. Cui, "An image inpainting technique based on 8-neighborhood fast sweeping method," in 2009 WRI International Conference on Communications and Mobile Computing, 2009, pp. 626-630. doi: 10.1109/CMC.2009.369.
4. Y. Fan, P. Carré, and C. Fernandez-Maloigne, "Image splicing detection with local illumination estimation," in 2015 IEEE (Institute of Electrical and Electronics Engineers) International Conference on Image Processing (ICIP), 2015, pp. 2940-2944. doi: 10.1109/ICIP.2015.7351341.
5. S. Nath and R. Naskar, "Automated image splicing detection using deep CNN-learned features and ANN-based classifier," SIViP, vol. 15, pp. 1601-1608, 2021. doi: 10.1007/s11760-021-01895-5.
6. S. Dua, J. Singh, and H. Parthasarathy, "Image forgery detection based on statistical features of block DCT coefficients," Procedia Computer Science, vol. 171, pp. 369-378, 2020. doi: 10.1016/j.procs.2020.04.038.

7. M. D. Ansari, S. P. Ghrera, and V. Tyagi, "Pixel-Based Image Forgery Detection: A Review," IETE Journal of Education, vol. 55, no. 1, pp. 40-46, 2014. doi: 10.1080/09747338.2014.921415.
8. P. Sharma, M. Kumar, and H. Sharma, "Comprehensive analyses of image forgery detection methods from traditional to deep learning approaches: An evaluation," Multimedia Tools and Applications, 2022. doi: 10.1007/s11042-022-13808-w.
9. Z. J. Barad and M. M. Goswami, "Image Forgery Detection using Deep Learning: A Survey," in 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), 2020, pp. 571-576. doi: 10.1109/ICACCS48705.2020.9074408.
10. [Z. Xi, W. Huang, K. Wei, W. Luo, and P. Zheng, "AI-Generated Image Detection using a Cross-Attention Enhanced Dual-Stream Network, 2023.
11. M. Zhu, H. Chen, Q. Yan, X. Huang, G. Lin, W. Li, Z. Tu, H. Hu, J. Hu, and Y. Wang, "GenImage: A Million-Scale Benchmark for Detecting AI-Generated Image," Huawei Noah's Ark Lab, arXiv:2306.08571v2 [cs.CV], Jun. 24, 2023. Available: https://arxiv.org/abs/2306.08571v2
12. M. N. Marzuki, N. A. Abdul Wahab, and M. F. Mahmud, "Detection of copy-move forgery in digital images," IEEE Access, vol. 6, pp. 26702-26712, 2018.
13. J. Hao, Z. Zhang, S. Yang, D. Xie, and S. Pu, "TransForensics: Image Forgery Localization with Dense Self-Attention," in Proceedings of the 2021 IEEE/CVF International Conference on Computer Vision (ICCV), Montreal, QC, 2021, doi: 10.1109/ICCV48922.2021.01478.

14. R. Agarwal and O.P. Verma, "An efficient copy move forgery detection using deep learning feature extraction and matching algorithm" Multimed Tools Appl, vol. 79, pp. 7355-7376, 2020. doi: 10.1007/s11042-019-08495-
15. A. Doegar, M. Dutta, and G. Kumar, "Image Forgery Detection Using Google Net and Random Forest Machine Learning Algorithm," J. Univ. Shanghai Sci. Technol., doi: 10.51201/12508.
16. Y. Zhang, J. Goh, L. L. Win, and V. Thing, "Image Region Forgery Detection: A Deep Learning Approach," in Proceedings of the Singapore Cyber-Security Conference (SG-CRC) 2016, A. Mathur and A. Roychoudhury (Eds.), Singapore, 2016, doi: 10.3233/978-1-61499-617-0-1.
17. F.M. Al_Azrak, A. Sedik, M.I. Dessowky, et al., An efficient method for image forgery detection based on trigonometric transforms and deep learning," Multimed Tools Appl, vol. 79, pp. 18221-18243, 2020. doi: 10.1007/s11042-019-08162-3.
18. T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," arXiv:1812.04948 [cs.NE], Mar. 2019. [Online]. Available: https://doi.org/10.48550/arXiv.1812.04948
19. G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," arXiv:1608.06993 [cs.CV], Jan. 2018. [Online]. Available: https://doi.org/10.48550/arXiv.1608.06993
20. J. Sharma and S. Sharma, “Challenges and Solutions in Deepfakes,” arXiv:2109.05397 [cs.CV], Sep. 2021.
21. M. M. Islam, G. Karmakar, J. Kamruzzaman, and M. Murshed, "A Robust Forgery Detection Method for Copy–Move and Splicing Attacks in Images," in IEEE Transactions on Image Processing, vol. 29, pp. 9244-9256, 2020.

22. M. Kaur and S. Gupta, "A Passive Blind Approach for Image Splicing Detection Based on DWT and LBP Histograms," in Proceedings of the Security in Computing and Communications (SSCC) 2016, P. Mueller, S. Thampi, M. Alam Bhuiyan, R. Ko, R. Doss, and J. Alcaraz Calero, Eds., vol. 625, Communications in Computer and Information Science, Singapore: Springer, 2016, pp. [Page Numbers]. DOI: 10.1007/978-981-10-2738-3_27.

APPENDIX A

ResNet50 Splicing detection code

```
from tensorflow.keras import layers, Model
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.metrics import BinaryAccuracy, Precision, Recall, FalseNegatives, FalsePositives, TrueNegatives, \
    TruePositives, AUC
from tensorflow.keras.losses import BinaryCrossentropy
import os
import math
import numpy as np
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.layers import Dense, Flatten, Dropout,BatchNormalization
from tensorflow.keras.callbacks import ModelCheckpoint
]
```

```
[2]
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

Outputs are collapsed ···

```
[3]
print(tf.config.list_physical_devices('GPU'))
```

Outputs are collapsed ···

```
[4]
physical_devices = tf.config.experimental.list_physical_devices('GPU')
if physical_devices:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
    tf.config.experimental.set_visible_devices(physical_devices[0], 'GPU')
```

```
[5]
os.environ['CUDA_VISIBLE_DEVICES'] = '0'
```

```
[6]
np.random.seed(42)
tf.random.set_seed(42)
```

```
class CustomImageDataGenerator(tf.keras.preprocessing.image.ImageDataGenerator):
    def __init__(self, **kwargs):
        super().__init__(**kwargs)

    def flow_from_directory(self, *args, **kwargs):
        iterator = super().flow_from_directory(*args, **kwargs)
        while True:
            try:
                data = next(iterator)
                yield data
            except (OSError, tf.errors.InvalidArgumentError):
                continue
```

```
IMG_HEIGHT = 224
IMG_WIDTH = 224
BATCH_SIZE = 8

img_data_gn = CustomImageDataGenerator(rescale=1./255,rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
train_path = "/media/maged/New Volume/NU/fall 22/Grad project/datasets/Splicing huge/Train"
valid_path = "/media/maged/New Volume/NU/fall 22/Grad project/datasets/Splicing huge/Validation"
```

```
train_batches = img_data_gn.flow_from_directory(train_path,
                                                target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                class_mode='binary',
                                                classes=['Au', 'Tp'],
                                                batch_size=BATCH_SIZE,
                                                shuffle=True)

valid_batches = img_data_gn.flow_from_directory(valid_path,
                                                target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                class_mode='binary',
                                                classes=['Au', 'Tp'],
                                                batch_size=BATCH_SIZE, shuffle=True)
```

```
train_image_count = sum([len(files) for _, _, files in os.walk(train_path)])
valid_image_count = sum([len(files) for _, _, files in os.walk(valid_path)])


steps_per_epoch = math.ceil(train_image_count / BATCH_SIZE)
validation_steps = math.ceil(valid_image_count / BATCH_SIZE)
```

```
model = ResNet50(input_shape=(IMG_HEIGHT, IMG_WIDTH, 3), weights= None , include_top=False)
```

```
last_layer = model.get_layer('conv5_block3_out')
last_output = last_layer.output
```

6]

```
x = layers.Flatten()(last_output)
x = layers.Dense(1000, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = layers.Dense(1000, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(500, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(500, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = layers.Dense(1, activation='sigmoid')(x)
```

7]

```
newModel = Model(model.input, x)
```

8]

```
newModel.compile(Adam(learning_rate=0.001), loss=BinaryCrossentropy(),
                 metrics=[AUC(), BinaryAccuracy(), Precision(), Recall(), FalseNegatives(),
                          FalsePositives(), TrueNegatives(), TruePositives() ])
```

9]

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=0.0000001, verbose=1)
```

[20] Python

```
#class_weights_dict = {0: 1, 1: 3}
history1 = newModel.fit(train_batches, verbose=1, validation_data=valid_batches,
                        epochs=20, steps_per_epoch=steps_per_epoch, validation_steps=validation_steps,
                        callbacks=[early_stopping, reduce_lr])
```

[21] Python

```
Found 145383 images belonging to 2 classes.
Epoch 1/20
2023-06-16 22:30:35.908102: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a v
         [[{{node Placeholder/_0}}]]
2023-06-16 22:30:47.532496: I tensorflow/compiler/xla/stream_executor/cuda/cuda_dnn.cc:424] Loaded cuDNN version 8600
2023-06-16 22:30:53.155285: I tensorflow/compiler/xla/stream_executor/cuda/cuda_blas.cc:637] TensorFloat-32 will be used for the matrix multiplication. This will only be logged once.
2023-06-16 22:30:53.298693: I tensorflow/compiler/xla/service/service.cc:169] XLA service 0x32a69740 initialized for platform CUDA (this does not guarantee that XLA will be used). Devices:
2023-06-16 22:30:53.298711: I tensorflow/compiler/xla/service/service.cc:177]   StreamExecutor device (0): NVIDIA GeForce RTX 3060 Ti, Compute Capability 8.6
2023-06-16 22:30:53.447525: I tensorflow/compiler/mlir/tensorflow/utils/dump_mlir_util.cc:269] disabling MLIR crash reproducer, set env var `MLIR_CRASH_REPRODUCER_DIRECTORY` to enable.
2023-06-16 22:30:54.319744: I ./tensorflow/compiler/jit/device_compiler.h:180] Compiled cluster using XLA!  This line is logged at most once for the lifetime of the process.
18173/18173 [==============================] - ETA: 0s - loss: 0.7189 - auc: 0.5272 - binary_accuracy: 0.5196 - precision: 0.5199 - recall: 0.5161 - false_negatives: 35194.0000 - false_positives: 34653.0000 - true_negatives: 38003.000
2023-06-16 23:42:04.016898: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignore this message): INVALID_ARGUMENT: You must feed a v
         [[{{node Placeholder/_0}}]]
18173/18173 [==============================] - 5276s 288ms/step - loss: 0.7189 - auc: 0.5272 - binary_accuracy: 0.5196 - precision: 0.5199 - recall: 0.5161 - false_negatives: 35194.0000 - false_positives: 34653.0000 - true_negatives:
Epoch 2/20
18173/18173 [==============================] - 4535s 250ms/step - loss: 0.6844 - auc: 0.5877 - binary_accuracy: 0.5627 - precision: 0.5604 - recall: 0.5831 - false_negatives: 30316.0000 - false_positives: 33264.0000 - true_negatives:
Epoch 3/20
18173/18173 [==============================] - 4140s 228ms/step - loss: 0.6595 - auc: 0.6467 - binary_accuracy: 0.6112 - precision: 0.6000 - recall: 0.6680 - false_negatives: 24143.0000 - false_positives: 32386.0000 - true_negatives:
Epoch 4/20
18173/18173 [==============================] - 4157s 229ms/step - loss: 0.6498 - auc: 0.6648 - binary_accuracy: 0.6259 - precision: 0.6118 - recall: 0.6901 - false_negatives: 22539.0000 - false_positives: 31847.0000 - true_negatives:
Epoch 5/20
18173/18173 [==============================] - ETA: 0s - loss: 0.6428 - auc: 0.6780 - binary_accuracy: 0.6350 - precision: 0.6214 - recall: 0.6917 - false_negatives: 22419.0000 - false_positives: 30648.0000 - true_negatives: 42017.000
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
18173/18173 [==============================] - 4192s 231ms/step - loss: 0.6428 - auc: 0.6780 - binary_accuracy: 0.6350 - precision: 0.6214 - recall: 0.6917 - false_negatives: 22419.0000 - false_positives: 30648.0000 - true_negatives:
Epoch 6/20
18173/18173 [==============================] - 4103s 226ms/step - loss: 0.5974 - auc: 0.7431 - binary_accuracy: 0.6823 - precision: 0.6694 - recall: 0.7213 - false_negatives: 20272.0000 - false_positives: 25915.0000 - true_negatives:
Epoch 7/20
18173/18173 [==============================] - 4184s 230ms/step - loss: 0.4524 - auc: 0.8697 - binary_accuracy: 0.7914 - precision: 0.7868 - recall: 0.7997 - false_negatives: 14570.0000 - false_positives: 15755.0000 - true_negatives:
Epoch 8/20
18173/18173 [==============================] - 4208s 232ms/step - loss: 0.3783 - auc: 0.9111 - binary_accuracy: 0.8305 - precision: 0.8269 - recall: 0.8363 - false_negatives: 11907.0000 - false_positives: 12732.0000 - true_negatives:
Epoch 9/20
18173/18173 [==============================] - 4230s 233ms/step - loss: 0.3008 - auc: 0.9444 - binary_accuracy: 0.8790 - precision: 0.8755 - recall: 0.8838 - false_negatives: 8450.0000 - false_positives: 9141.0000 - true_negatives: 63
```

APPENDIX B

inceptionV3 copy move code.

```
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.layers import Dense, Flatten, Dropout, BatchNormalization , MaxPooling2D,GlobalAveragePooling2D,UpSampling2D , concatenate
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.metrics import AUC, BinaryAccuracy, Precision, Recall, FalseNegatives, FalsePositives, TrueNegatives, TruePositives
from tensorflow.keras.losses import BinaryCrossentropy
import os
import math
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, Dense, Dropout, Flatten, BatchNormalization, Input, Lambda, Multiply
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import Lambda, Dense, Reshape
```

```
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

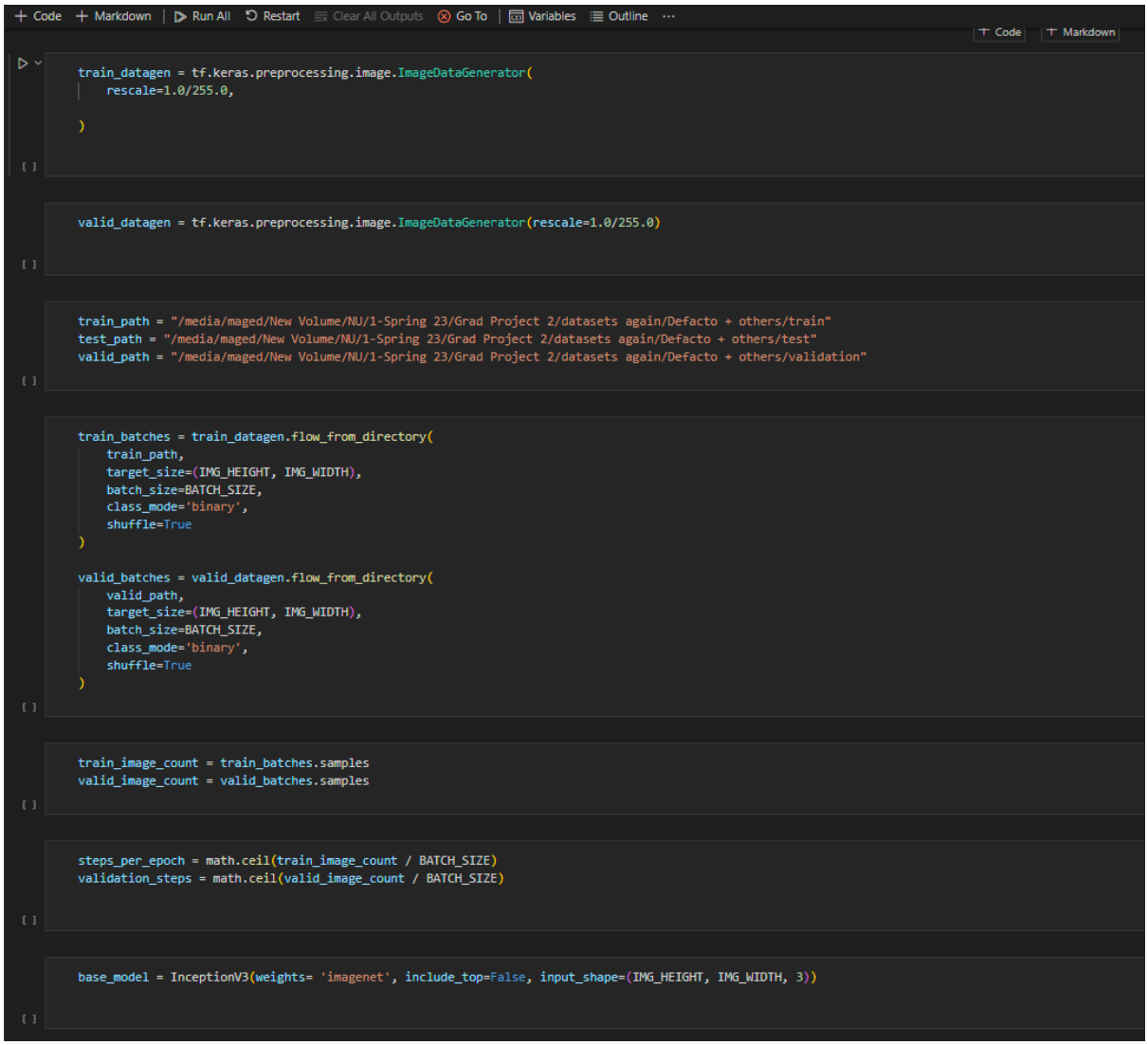
```
print(tf.config.list_physical_devices('GPU'))
```

```
# Set the GPU device
physical_devices = tf.config.experimental.list_physical_devices('GPU')
if physical_devices:
    tf.config.experimental.set_memory_growth(physical_devices[0], True)
    tf.config.experimental.set_visible_devices(physical_devices[0], 'GPU')
```

```
# Set GPU device
os.environ['CUDA_VISIBLE_DEVICES'] = '0'  # Replace '0' with the GPU device ID
```

```
np.random.seed(42)
tf.random.set_seed(42)
```

```
IMG_HEIGHT = 299
IMG_WIDTH = 299
BATCH_SIZE = 8
```



```
+ Code + Markdown | Run All Restart Clear All Outputs Go To | Variables Outline ···
                                                                    Code  Markdown

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1.0/255.0,

)
[ ]

valid_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale=1.0/255.0)
[ ]

train_path = "/media/maged/New Volume/NU/1-Spring 23/Grad Project 2/datasets again/Defacto + others/train"
test_path = "/media/maged/New Volume/NU/1-Spring 23/Grad Project 2/datasets again/Defacto + others/test"
valid_path = "/media/maged/New Volume/NU/1-Spring 23/Grad Project 2/datasets again/Defacto + others/validation"
[ ]

train_batches = train_datagen.flow_from_directory(
    train_path,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    shuffle=True
)

valid_batches = valid_datagen.flow_from_directory(
    valid_path,
    target_size=(IMG_HEIGHT, IMG_WIDTH),
    batch_size=BATCH_SIZE,
    class_mode='binary',
    shuffle=True
)
[ ]

train_image_count = train_batches.samples
valid_image_count = valid_batches.samples
[ ]

steps_per_epoch = math.ceil(train_image_count / BATCH_SIZE)
validation_steps = math.ceil(valid_image_count / BATCH_SIZE)
[ ]

base_model = InceptionV3(weights= 'imagenet', include_top=False, input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))
[ ]
```

```
for layer in base_model.layers:
    layer.trainable = False
```

```
x = base_model.output
attention = Conv2D(1, (1, 1), activation='sigmoid')(x)
attended_features = multiply([x, attention])

# Classification layers
x = Flatten()(attended_features)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(1024, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)

# Output layer
output = Dense(1, activation='sigmoid')(x)
```

```
model = Model(inputs=base_model.input, outputs=output)
```

```
model.compile(optimizer=Adam(learning_rate=0.001),
              loss=BinaryCrossentropy(),
              metrics=[AUC(), BinaryAccuracy(), Precision(), Recall(), FalseNegatives(), FalsePositives(), TrueNegatives(), TruePositives()])

model.summary()
```

```
early_stopping = EarlyStopping(monitor='val_loss', patience=5)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=0.0000001, verbose=1)
```

APPENDIX C

Machine learning code

Code implementation of - **S. Dua, J. Singh, and H. Parthasarathy, "Image forgery detection based on statistical f (CoCoNet'19). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050920310048**

```
import cv2
import pandas as pd
import numpy as np
import os
from scipy.fftpack import fft, dct
from sklearn import svm
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from skimage.util import view_as_windows
```

```
def get_patches(image_mat):
    """
    Extract patches rom an image
    :param image_mat: The image as a matrix
    :param stride: The stride of the patch extraction process
    :returns: The patches
    """
    stride=8 #stride is same as window's breadth so that it gives non-overlapping blocks.
    window_shape = (8, 8)
    image_mat=np.array(image_mat)

    windows = view_as_windows(image_mat, window_shape, step=stride)
#     print('windows shape:',windows.shape)

    patches = []
    for m in range(windows.shape[0]):
        for n in range(windows.shape[1]):
#             print("window shape: ",windows[m][n].shape)
            patches += [windows[m][n]]
    return patches
```

```
def std_and_ones(type_of_sub_image_blocks):
    ac_dct_stack=[]
    number_of_ones=[]

    for block in type_of_sub_image_blocks:
        dct_block = dct(block, type=2, n=None, axis=-1, norm=None, overwrite_x=False)
        dct_block_row = dct_block.flatten() # 2d dct array to 1d row array.
        ac_dct = dct_block_row[1:] # only AC component, removing the first DC comp.
        ac_dct_stack.append(ac_dct)

    ac_dct_stack=np.asarray(ac_dct_stack) #1536X63
    ac_dct_stack=ac_dct_stack.T # 63X1536

#     print("AC stacked shape: ", ac_dct_stack.shape)

    ac_dct_std = np.std(ac_dct_stack, axis=1) # row wise standard-deviation.

    for i in range(ac_dct_stack.shape[0]):
        count_one=0
        for j in range(ac_dct_stack.shape[1]):
            if(ac_dct_stack[i][j]>0):   # row wise counting number of ones.
                count_one+=1
        number_of_ones.append(count_one)

    number_of_ones=np.asarray(number_of_ones)

    return(ac_dct_std, number_of_ones)
```

```
    def feature_sub_image(sub_image):
        sub_image_blocks = get_patches(sub_image) #Gives the 8x8 patches/blocks of sub_image.

        sub_image_cropped = sub_image[4:,4:] #removing 4 rows and 4 cols.
        sub_image_cropped_blocks = get_patches(sub_image_cropped)

        STD_full_image, ONE_full_image = std_and_ones(sub_image_blocks)
        STD_cropped_image, ONE_cropped_image = std_and_ones(sub_image_cropped_blocks)

        #             print("STD_full image shape: ",STD_full_image.shape)
        #             print("one_full image shape: ",ONE_full_image.shape)
        #             print("STD_crop image shape: ",STD_cropped_image.shape)
        #             print("One_crop image shape: ",ONE_cropped_image.shape)

        #63x4 stacked F-sub-image
        F_sub_image=np.column_stack((STD_full_image, ONE_full_image, STD_cropped_image, ONE_cropped_image))

        F_sub_image_flat=F_sub_image.T.flatten() #column wise flattening, 63*4=252 features
        return(F_sub_image_flat)
[10]
```

```
#main function to extract the features.
def feature_extraction(path_to_folder, class_label):
    data_list=[]
    for file_name in os.listdir(path_to_folder):
        path_to_img = os.path.join(path_to_folder,file_name)
        img = cv2.imread(path_to_img)

        if np.shape(img) == ():
            continue

        img = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb) #changing to YCrCb color space.
        img_y = img[:,:,0] # the Y channel only.
        img_cr = img[:,:,1] # the Cr channel only.
        img_cb = img[:,:,2] # the Cb channel only.

        Fy = feature_sub_image(img_y)
        Fcr = feature_sub_image(img_cr)
        Fcb = feature_sub_image(img_cb)
#         print("fy shape: ",Fy.shape)
#         print("fcr shape: ",Fcr.shape)
#         print("fcb shape: ",Fcb.shape)

        final_feature = np.concatenate((Fy, Fcb, Fcr), axis=None) #63*4*3=756 flattened features.
#         print("final feature shape: ",final_feature.shape)

        final_feature=list(final_feature)
        final_feature.insert(0,file_name)
        final_feature.insert(1,class_label)
        data_list.append(final_feature)


    return(data_list)
```

```
au_path="D:/THE_SHRIMP/gam3a/grad/Image_Forgery_Detection-main/Image_Forgery_Detection-main/Alahmadi et al/CASIA 1.0 dataset/Au/Au"
tp_cm_path="D:/THE_SHRIMP/gam3a/grad/Image_Forgery_Detection-main/Image_Forgery_Detection-main/Alahmadi et al/CASIA 1.0 dataset/Modified Tp/Tp/CM"
tp_sp_path="D:/THE_SHRIMP/gam3a/grad/Image_Forgery_Detection-main/Image_Forgery_Detection-main/Alahmadi et al/CASIA 1.0 dataset/Modified Tp/Tp/Sp"
output_name='Casia1.0_features_NOSCALE.csv'

data_list1 = feature_extraction(au_path, 0)
data_list2 = feature_extraction(tp_cm_path, 1)
data_list3 = feature_extraction(tp_sp_path, 1)

df = pd.DataFrame(data_list1)
df = df.append(pd.DataFrame(data_list2), ignore_index=True)
df = df.append(pd.DataFrame(data_list3), ignore_index=True)

df.rename(columns = {0: "image_names", 1: "label"}, inplace = True)
```