CS4043D Image Processing

Course Project

Date: 06/04/2024

Detection of c

GROUP: F

Athota Rama Govinda	B200822CS
Sai Krishna T	B200816CS
Kushal P	B200796CS
Jahnav Y	B200847CS
Padala Sawtvik	B200812CS

Department of Computer Science and Engineering
National Institute of Technology, Calicut

Student’s first and last name, Year

Declaration

This report has been prepared on the basis of our group’s work. All the other published and unpublished source materials have been cited in this report.

Student Name: Signature:

Kushal P	B200796CS
Sai Krishna T	B200816CS
Athota Rama Govinda	B200822CS
Jahnav Y	B200847CS
Padala Sawtvik	B200812CS

Table of Contents

Abstract

The project aims to develop a machine learning model for detecting image authenticity using Error Level Analysis (ELA) and Convolutional Neural Networks (CNNs). The proposed approach involves preprocessing images with ELA to highlight discrepancies in compression levels, which are indicative of potential editing or tampering. Subsequently, a CNN model is trained on the ELA-enhanced images to classify them as authentic or edited. The project utilizes Python libraries such as Pandas, NumPy, Matplotlib, Seaborn, and PIL, along with machine learning frameworks like Keras with TensorFlow backend. The model's performance is evaluated using metrics such as accuracy and confusion matrix, providing insights into its effectiveness in detecting image authenticity.

Project Specification

The project addresses the challenge of detecting image authenticity, particularly distinguishing between authentic and edited images. The main problem statement involves developing a robust machine learning model capable of accurately classifying images as either authentic (non-edited) or edited based on subtle differences in compression levels. This requires preprocessing techniques such as Error Level Analysis (ELA) to highlight discrepancies introduced during image editing processes. Additionally, the project aims to leverage Convolutional Neural Networks (CNNs) for image classification tasks, given their effectiveness in capturing spatial features. The scope of the project includes data preparation, model building, training, and evaluation, with an emphasis on optimizing the model's performance and generalization capabilities. Constraints such as dataset size, computational resources, and model interpretability are taken into consideration during the development process. The project's significance lies in its potential applications in image forensics, authentication, and tampering detection, with implications for various domains including digital media, law enforcement, and cybersecurity.

Chapter 1: **Introduction**

1.1 Background

Existing approaches

Splicing images using multi-size block discrete cosine transform (MBDCT)

Methodology: The implementation involves splicing images using Block Discrete Cosine Transform (BDCT) and Markov processes. BDCT partitions images into blocks, each transformed with DCT. Markov process models spatial dependencies within blocks. By splicing segments from different images and applying BDCT and Markov processes, seamless forgeries are created.

Overview: This technique allows forgeries to maintain statistical coherence across blocks, enhancing realism. The combination of BDCT and Markov processes provides an effective method for generating convincing image splices, enabling the creation of seamless forgeries with realistic spatial dependencies.

Identification of Bitmap Compression History Of a JPEG

Methodology: If one wants to remove JPEG artefacts or for JPEG recompression. A fast and efficient method is provided to determine whether an image has been previously JPEG compressed. After detecting a compression signature, compression parameters were estimated. Specifically, maximum likelihood estimation of JPEG quantization steps were developed.

Overview**:** Sometimes image processing units inherit images in raster bitmap format only, so that processing is to be carried without knowledge of past operations that may compromise image quality (e.g., compression). To carry further processing, it is useful to not only know whether the image has been previously JPEG compressed, but to learn what quantization table was used.The quantizer estimation method is very robust so that only sporadically an estimated quantizer step size is off, and when so, it is by one value.

1.2 Challenges

Detecting Spliced Images using MBDCT:

The challenge lies in maintaining statistical coherence across blocks when detecting spliced images using Multi-size Block Discrete Cosine Transform (MBDCT). Seamless forgeries created through MBDCT may exhibit realistic spatial dependencies, making it arduous to distinguish between authentic and spliced segments. Furthermore, the complexity of implementing MBDCT and Markov processes for image splicing entails intricate algorithms and computations, demanding sophisticated analysis techniques that can be computationally expensive. Moreover, achieving generalization across diverse datasets and scenarios poses a challenge due to variations in image content, resolution, and manipulation techniques, necessitating the development of robust detection methods that can generalize effectively.

Identification of Bitmap Compression History of a JPEG:

A key challenge in the identification of the bitmap compression history of a JPEG image is accurately detecting the compression signature. While the method offers a swift and efficient means of determining whether an image has been previously JPEG compressed, variations in compression artifacts and noise levels across different images and compression settings may affect the reliability of the detection process. Additionally, the estimation of compression parameters, particularly the quantization table used during JPEG compression, relies on robust maximum likelihood estimation techniques. However, variations in compression algorithms and optimization methods can introduce uncertainties in quantizer estimation, leading to inaccuracies in determining the compression history. Ensuring the robustness and accuracy of the quantizer estimation method is crucial for reliable identification of JPEG compression history, requiring the method to handle diverse image datasets and accurately infer the quantization table used, even in the presence of noise or artifacts.

Chapter 2: **Literature Review**

In contrast to the active image forensic approaches, e.g., semi-fragile watermarking and image hashing, passive techniques for image forensics are more useful, but more challenging. These techniques work on the assumption that although digital forges may leave no visual clues of what has been tampered with, they may alter the underlying statistics of an image. In recognition of this fact, a variety of image tampering detection techniques have been proposed in recent years. Shi et al. [1] proposed a natural image model for image splicing detection. They applied block discrete cosine transform (DCT) to images and combined the features extracted from statistical moments of characteristic functions with the ones from the Markov transition probability matrices in both spatial and DCT domain, so as to obtain the discriminative feature vectors for support vector machine (SVM) classification.

Fan and Queiroz[2]constructed a method determining whether an image has been previously JPEG compressed and to estimate compression parameters. A method for the maximum likelihood estimation was devised to estimate what quantization table was used.

Ying et al. [3] showed that the conventional deep learning framework may not be directly applied to image tampering detection, this because, with elaborated designed tools, the forgery images tend to closely resemble the authentic ones not only visually but also statistically. Therefore, they adopted the wavelet features of images as input of their deep autoencoder.

Motivated by the similar observation, Bayar et al. [4] proposed a new convolutional layer in their CNN model to learn prediction error filters with constraint to discover the traces left by image manipulations.

Passive image forensic techniques, while valuable, often face limitations such as reliance on handcrafted features, difficulty in distinguishing intricate forgeries, and challenges in handling diverse tampering scenarios. Methods like Shi et al.'s natural image model, Fan and Queiroz's JPEG compression detection, Ying et al.'s deep autoencoder with wavelet features, and Bayar et al.'s CNN model with prediction error filters may struggle with varying degrees of accuracy and robustness in detecting tampered images. Deep learning approaches offer a compelling solution by leveraging feature learning, adaptability to diverse data, and end-to-end learning capabilities. These advantages enable deep learning models to effectively capture complex patterns and nuances in manipulated images, making them a promising direction for improving the reliability and performance of image tampering detection.

Chapter 3: **Proposed Method**

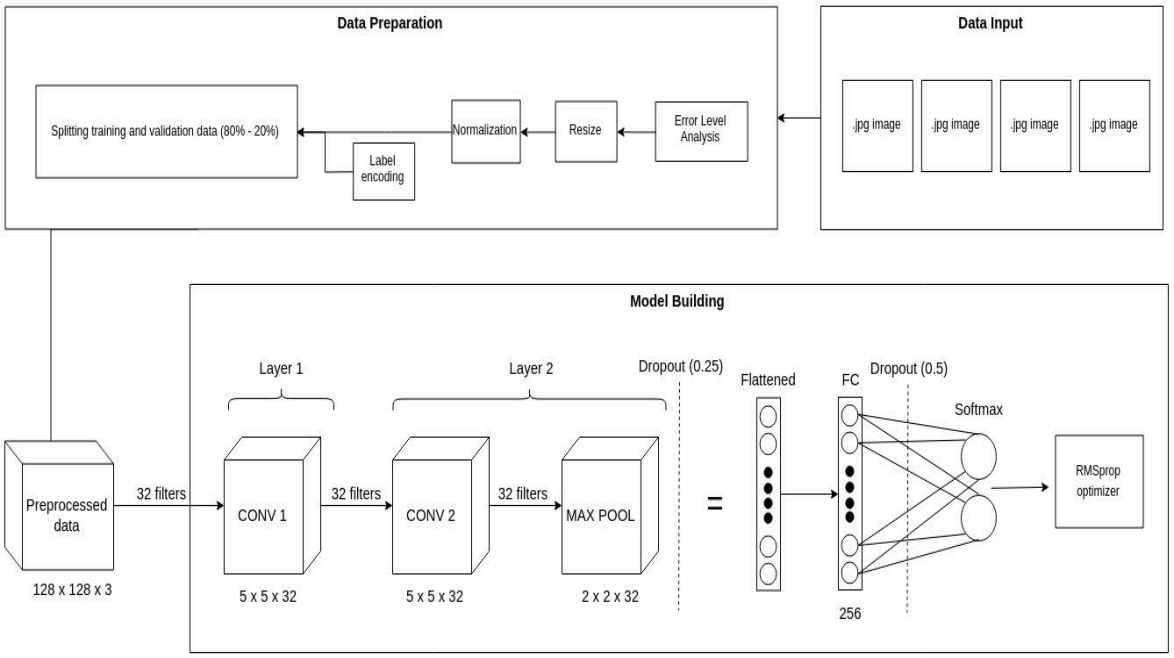


Figure 1. Graphical Design

Two primary methodologies are proposed for incorporation into this image processing: Error Level Analysis (ELA) and Convolutional Neural Network (CNN) with deep learning techniques.

Error Level Analysis (ELA):

Error Level Analysis is a technique primarily used in digital forensics to detect potential manipulations in digital images. It operates on the principle that when an image is compressed and then saved again, errors or artifacts are introduced into the image due to the compression process. ELA exploits these errors to identify regions of the image that may have been altered.

ELA compares the error levels in different parts of an image. When an image is manipulated, the error levels in the manipulated regions often differ from the rest of the image. By analysing these discrepancies, ELA can pinpoint areas that are likely to have been tampered with.

Convolutional Neural Network (CNN):

Convolutional Neural Networks are a class of deep learning models specifically designed for processing structured grid-like data, such as images. They have revolutionised various tasks in computer vision, including image classification, object detection, and segmentation.

Architecture: CNNs typically consist of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

Convolutional Layers: These layers act as feature extractors, where they apply filters to the input image to detect features like edges, textures, or patterns.

Pooling Layers: Pooling layers downsample the feature maps produced by convolutional layers, reducing computational complexity while preserving important features.

Fully Connected Layers: These layers interpret the features extracted by previous layers and perform high-level reasoning. In image classification tasks, they typically produce a probability distribution over the possible classes using techniques like the softmax function.

Training: CNNs are trained using labelled data through a process called backpropagation, where the network learns to adjust its parameters (weights and biases) to minimise the difference between predicted and actual outputs.

Incorporating both ELA and CNN techniques into image processing allows for a comprehensive approach to analysing and detecting digital image manipulation. ELA provides a forensic tool for detecting potential alterations, while CNNs leverage deep learning techniques to automatically learn and extract features from images, enabling tasks such as image classification with high accuracy and efficiency. Combining these methodologies can enhance the overall effectiveness of image analysis and forensic investigations.

3.1 Tools and Techniques

Python Libraries

Pandas, NumPy, Matplotlib, Seaborn: These libraries were instrumental in handling dataset manipulation, numerical computations, and data visualization tasks. Pandas facilitated the handling of datasets, while NumPy allowed for efficient numerical operations. Matplotlib and Seaborn were used for creating visualizations, including plots and images.

Image Processing Tools

Python Imaging Library (PIL)

PIL (Python Imaging Library): Used for opening, manipulating, and saving various image file formats. PIL played a crucial role in image processing tasks such as ELA, enabling the detection of areas within images that were potentially edited or tampered with.

Error Level Analysis (ELA)

ELA: An image forensic method employed as a preprocessing step to highlight discrepancies in compression levels within images. By analyzing the error levels, ELA helps distinguish between authentic and edited regions within images.

Machine Learning Frameworks

Keras with TensorFlow Backend

Keras with TensorFlow backend: A high-level neural networks API written in Python, capable of running on top of TensorFlow. Keras provided an intuitive interface for building and training deep

learning models, particularly Convolutional Neural Networks (CNNs), which were pivotal in image classification tasks.

ImageDataGenerator: A TensorFlow utility utilized for generating batches of tensor image data with real-time data augmentation. This tool was instrumental in augmenting the dataset to improve model generalization and robustness.

Model Training and Evaluation Techniques

Optimizers and Callbacks

RMSprop optimizer: An optimization algorithm employed for training neural networks. RMSprop adapts the learning rate during training to accelerate convergence.

EarlyStopping callback: A Keras callback utilized to halt training when monitored metrics have ceased to improve. This prevented overfitting and facilitated efficient model training.

Chapter 4: **Evaluation and Results**

< content.>

4.1 Evaluation and Comparison Of Results

<content>

The current model is being evaluated by calculating the confusion matrix and persorming the ELA (error level analysis) on the image to identify the area of forgery int the given image

4.1.1 ELA Conversion and Detection of Area of Forgery

It computes the ELA (Error Level Analysis) by taking the difference between the original and resaved images.Next, it calculates the maximum difference in pixel values between the original and resaved images.

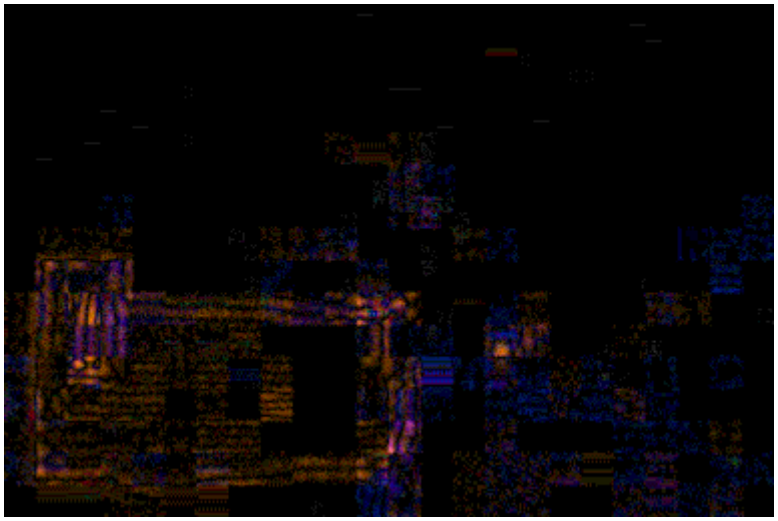
If the maximum difference is zero, it sets it to one to avoid division by zero.

It then scales the ELA image based on the maximum difference to enhance brightness.

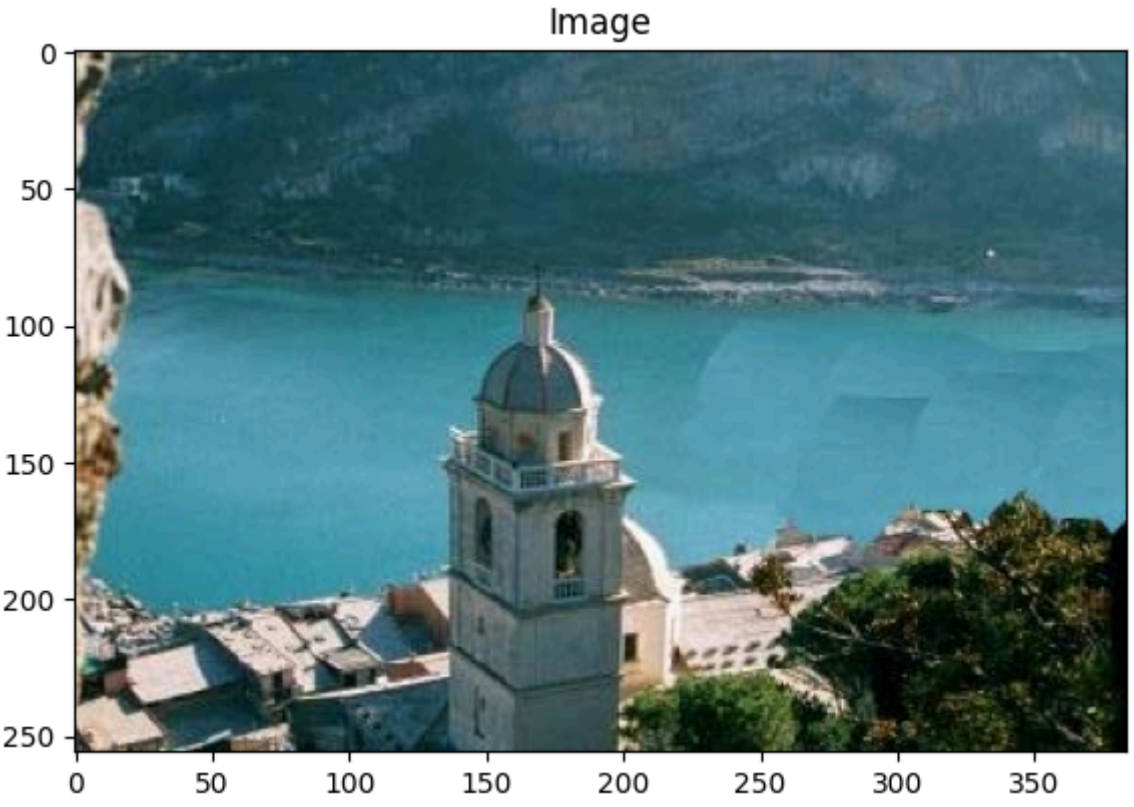
Finally, it returns the ELA image as a result, which highlights areas of potential manipulation or compression artifacts in the original image.

Input Image :

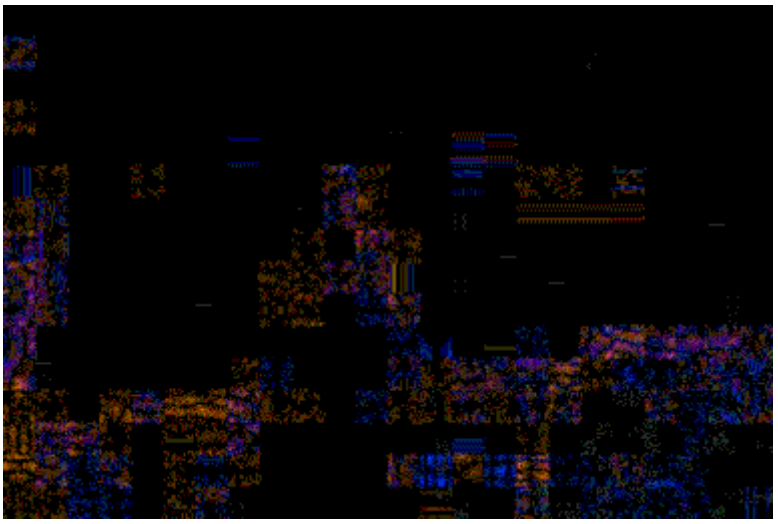
ELA converted image:



Input:



ELA Converted image:

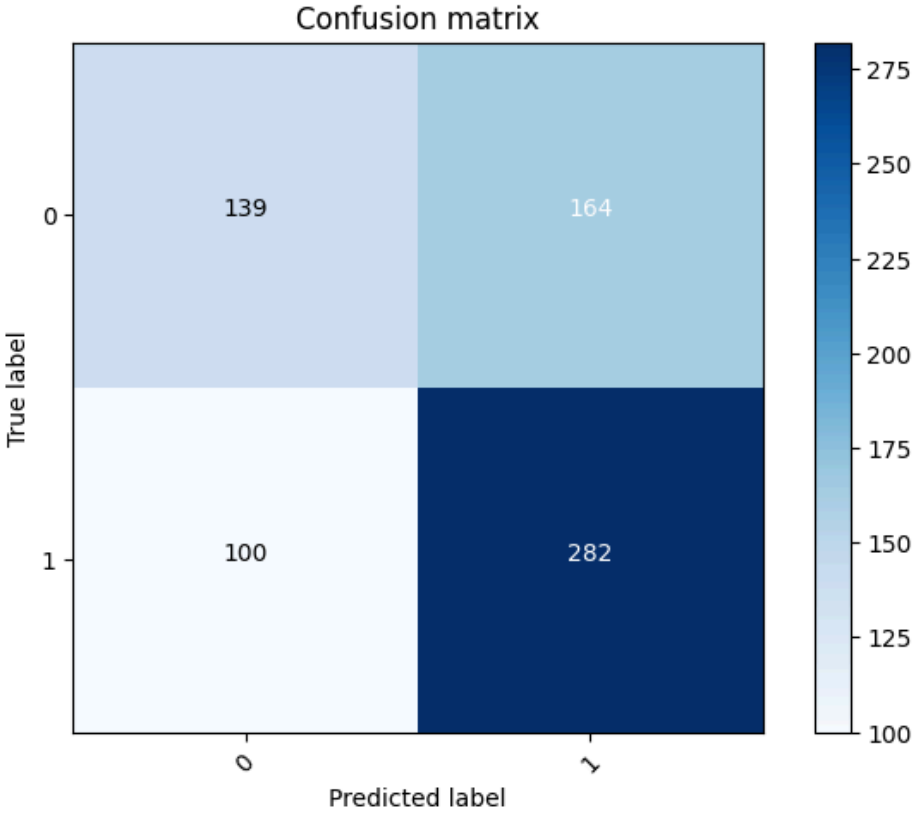


4.2 Comparison of results

Results of the model are normally compared and evaluated based on the confusion matrix and their respective accuracy scores .

4.2.1 Confusion Matrix :

It is designed typically to evaluate the performance of a classification model.If `normalize` is set to `True`, it normalizes the confusion matrix by dividing each row by its sum to show proportions.



4.2.2 **Accuracy And Precision** :

Precision: Precision is a measure of the correctness of positive predictions made by a classifier. It is calculated as the ratio of true positive predictions to the total number of positive predictions (true positives + false positives).

Accuracy: Accuracy is a measure of the overall correctness of a classifier across all classes. It is calculated as the ratio of the total number of correct predictions (true positives + true negatives) to the total number of instances.

Table 1

Model\Performance metrics	Accuracy	Precison	F1 measure
CNN model	0.81459854014598	0.812286995515695	0.83115942028985

Chapter 5: **Implementation and Code**

5.1 Documentation

1) DATASET PREPARATION :

INPUT:

1. `path_original`: A string representing the directory path where original (pristine) images are stored.

2. `path_tampered`: A string representing the directory path where tampered (fake) images are stored.

3. `dataset_path`: A string representing the directory path where the dataset is located, possibly containing both original and tampered images.

OUTPUT:

1. `pristine_images`: A list containing the file paths of original (pristine) images. Each path is constructed by concatenating `dataset_path` and the file name in `total_original`.

2. `fake_images`: A list containing the file paths of tampered (fake) images. Each path is constructed by concatenating `dataset_path` and the file name in `total_tampered`.

It retrieves the list of file names in the original and tampered image directories. It constructs the full paths for each image file in both directories by concatenating the dataset path and the file names. This populates two lists, `pristine_images` and `fake_images`, with the constructed paths for original and tampered images, respectively.

2)**ELA CONVERSION AND DETECTION OF AREA OF FORGERY**

INPUT:

1.`path`: A string representing the file path of the input image.

2.`quality`: An integer representing the quality of the JPEG compression to be applied when resaving the image.

OUTPUT:

- `ela_im`: An ELA (Error Level Analysis) image, which highlights areas in the input image where compression artifacts are present.

It opens the input image, converts it to RGB mode, and resaves it with JPEG compression at the specified quality. It opens the resaved image and calculates the difference between the original and resaved images using the `ImageChops.difference()` function, resulting in an image highlighting the compression artifacts.It calculates the maximum difference value between pixel values in the ELA image. It scales the brightness of the ELA image based on the maximum difference to enhance the visualization of compression artifacts.

3)MODEL TRAINING :

INPUT:

- `X_train`: The input training data, typically a 4D array representing a collection of images. The

- `Y_train`: The target training labels, typically encoded as one-hot vectors.

- `X_val`: The input validation data, similar to `X_train`.

- `Y_val`: The target validation labels, similar to `Y_train`.

OUTPUT:

- `history`: An object containing information about the training history, including loss and accuracy metrics over each epoch.This is the trained model which is used for image forgery detection.

4)ACCURACY MATRIX CONSTRUCTION:

INPUT:

- `cm`: The confusion matrix to be plotted, typically a 2D array.

- `classes`: A list or array containing the class labels.

- `normalize`: A boolean indicating whether to normalize the confusion matrix or not. If `True`, each row of the confusion matrix is normalized so that it sums to 1.

OUTPUT:

- A plot displaying the confusion matrix.

5.2 Source Code

DATASET PREPARATION:

```
path_original = '/content/drive/MyDrive/CASIA1/Au/'

path_tampered = '/content/drive/MyDrive/CASIA1/Sp/'

dataset_path = '/content/drive/MyDrive/CASIA1/'

total_original = os.listdir(path_original)

total_tampered = os.listdir(path_tampered)

# total_tampered.remove('.DS_Store')
```

```
print('total number of pristine and tampered images are respectively:',len(total_original),',',len(total_tampered))

pristine_images = []

for i in total_original:

    pristine_images.append(dataset_path+i)

fake_images = []

for i in total_tampered:

    fake_images.append(dataset_path+i)
```

DATASET CREATION AND CONVERSION INTO .CSV FILE:

```
def get_imlist(path):

        return [os.path.join(path,f) for f in os.listdir(path) if f.endswith('.jpg') or f.endswith('.JPG') or f.endswith('.png') or f.endswith('.tif')]

images = []

for file in tqdm(os.listdir(path_original)):

try:

    if file.endswith('jpg') or file.endswith('JPG') or file.endswith('jpeg') or file.endswith('JPEG'):

            if int(os.stat(path_original + file).st_size) > 10000:

                line = path_original + file  + ',0\n'

                images.append(line)

    except:

        print(path_original+file)

for file in tqdm(os.listdir(path_tampered)):

    try:

        if file.endswith('jpg'):
```

```
            if int(os.stat(path_tampered + file).st_size) > 10000:
                    line = path_tampered + file + ',1\n'
                    images.append(line)
        if file.endswith('tif'):
            if int(os.stat(path_tampered + file).st_size) > 10000:
                    line = path_tampered + file + ',1\n'
                    images.append(line)
    except:
          print(path_tampered+file)
image_name = []
label = []
for i in tqdm(range(len(images))):
    image_name.append(images[i][0:-3])
    label.append(images[i][-2])

dataset = pd.DataFrame({'image':image_name,'class_label':label})
```

ELA CONVERSION :

```
def convert_to_ela_image(path, quality):
    filename = path
    resaved_filename = filename.split('.')[0] + '.resaved.jpg'
    ELA_filename = filename.split('.')[0] + '.ela.png'

    im = Image.open(filename).convert('RGB')
    im.save(resaved_filename, 'JPEG', quality=quality)
    resaved_im = Image.open(resaved_filename)
```

```
    ela_im = ImageChops.difference(im, resaved_im)


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


    ela_im = ImageEnhance.Brightness(ela_im).enhance(scale)


    return ela_im
```

MODEL DESCRIPTION AND TRAINING:

```
model = Sequential()
model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'valid',
                 activation ='relu', input_shape = (128,128,3)))
print("Input: ", model.input_shape)
print("Output: ", model.output_shape)


model.add(Conv2D(filters = 32, kernel_size = (5,5),padding = 'valid',
                 activation ='relu'))
print("Input: ", model.input_shape)
print("Output: ", model.output_shape)
model.add(MaxPool2D(pool_size=(2,2)))
```

```
model.add(Dropout(0.25))

print("Input: ", model.input_shape)

print("Output: ", model.output_shape)



model.add(Flatten())

model.add(Dense(256, activation = "relu"))

model.add(Dropout(0.5))

model.add(Dense(2, activation = "softmax"))
```

CREATION OF ACCURACY MATRIX :

```
def plot_confusion_matrix(cm, classes,

                          normalize=False,

                          title='Confusion matrix',

                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest', cmap=cmap)

    plt.title(title)

    plt.colorbar()

    tick_marks = np.arange(len(classes))

    plt.xticks(tick_marks, classes, rotation=45)

    plt.yticks(tick_marks, classes)



    if normalize:

        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]



    thresh = cm.max() / 2.
```

```
            for   i,   j   in   itertools.product(range(cm.shape[0]),
range(cm.shape[1])):

        plt.text(j, i, cm[i, j],

                 horizontalalignment="center",

                 color="white" if cm[i, j] > thresh else "black")



    plt.tight_layout()

    plt.ylabel('True label')

    plt.xlabel('Predicted label')



Y_pred = model.predict(X_val)

Y_pred_classes = np.argmax(Y_pred,axis = 1)

Y_true = np.argmax(Y_val,axis = 1)

confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)

plot_confusion_matrix(confusion_mtx, classes = range(2))
```

References

[1] Y. Q. Shi, C. Chen, and W. Chen, "A natural image model approach to splicing detection," in Proceedings of the 9th workshop on Multimedia & security. (MM & Sec), Dallas, TX, USA, 2007, pp. 51–62.
[2] Identification of bitmap compression history: JPEG detection and quantizer estimation
[3] Fan, Zhigang,De Queiroz, Ricardo L, 2003.
[4] Z. Ying, J. Goha, L. Wina and V. Thinga, "Image Region Forgery Detection: A Deep Learning Approach," in Proceedings of the Singapore Cyber-Security Conference (SG-CRC), 2016, vol. 14, p. 1–11.
[5] B. Bayar, and M. C. Stamm. "A Deep Learning Approach to Universal Image Manipulation Detection Using a New Convolutional Layer," in Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security, 2016, pp. 5–10