Image Forgery Detection using Error Level Analysis (ELA)

Introduction

Image forgery detection is a critical aspect of digital forensics, ensuring the integrity and authenticity of visual content. This document presents a Python script implementing an Error Level Analysis (ELA) algorithm for passive image forgery detection. ELA is particularly effective in identifying areas of an image that may have undergone digital manipulations, such as splicing.

Error Level Analysis (ELA) Logic

JPEG Compression and Error Potential

JPEG compression is a widely used method to reduce the file size of images while maintaining reasonable quality. The ELA technique focuses on detecting variations in compression levels within the same image. In a normal JPEG-compressed image, all regions should exhibit a consistent compression level. However, if an area has a significantly different error level, it indicates a potential digital manipulation. ELA highlights areas likely to degrade colors upon recompression, emphasizing regions with higher potential for degradation compared to the rest of the image.

Implementation

Python Code for ELA Implementation

The provided Python script utilizes the Python Imaging Library (PIL) and OpenCV to implement the ELA algorithm. The ELA image is generated by comparing the original image with a temporally compressed version. The brightness of the ELA image is then normalized to highlight potential manipulated areas.

```
import os  # Import the os module for interacting with the
    operating system
```

```
from PIL import Image, ImageChops, ImageEnhance  # Import specific modules from the PIL library

def convert_to_ela_image(path, quality):

    # Temporary file names
    temp_filename = 'temp_file_name.jpg'

    # Assign a temporary filename for the original image
    ela_filename = 'temp_ela.png'

    # Assign a filename for the ELA image

    # Load and Save Original Image
    image = Image.open(path).convert('RGB')

    # Open the original image and convert it to the RGB color mode
    image.save(temp_filename, 'JPEG', quality=quality)

    # Save the original image temporarily with the specified quality

    # Load Temporary Image
    temp_image = Image.open(temp_filename)

    # Open the temporarily saved image

    # Calculate ELA by finding the absolute difference
    ela_image = ImageChops.difference(image, temp_image)

    # Calculate the absolute difference between the original and temporary images

    # Normalize ELA image brightness
    extrema = ela_image.getextrema()

    # Get the extrema (minimum and maximum values) of the ELA image
    max_diff = max([ex[1] for ex in extrema])

    # Find the maximum difference value from the extrema
    if max_diff == 0:
        max_diff = 1

        # Set a default maximum difference value if it is zero
    scale = 255.0 / max_diff

    # Calculate the scale factor for normalizing brightness
```

```python
46     ela_image = ImageEnhance.Brightness(ela_image).enhance(scale)
47     # Normalize the brightness of the ELA image
48 
49     return ela_image
50 
51     # Return the generated ELA image
```

Listing 1: ELA Implementation

Explanation:

- **Line 1-2:** Import the necessary modules - ‘os‘ for operating system interactions and specific modules from PIL.
- **Line 4-6:** Definition of temporary file names for original and ELA images.
- **Line 8-12:** Loading the original image, saving it temporarily, and loading the temporary image.
- **Line 15-19:** Calculating ELA by finding the absolute difference between the original and temporary images.
- **Line 22-28:** Normalizing ELA image brightness to highlight potential manipulated areas.

Output Image Analysis

The script analyzes both real and potentially manipulated images using the ELA algorithm. The output image is saved, highlighting areas affected by potential splicing in white color against the original content. Researchers can visually inspect these areas to verify authenticity.

```python
# Real Image Analysis
real_image_path = '/content/input_dir/r1.jpg'

# Assign the path for the real image
Image.open(real_image_path)

# Open the real image for visualization
convert_to_ela_image(real_image_path, 90)

# Analyze the real image using the ELA algorithm with a specified quality level

# Fake Image Analysis
fake_image_path = '/content/input_dir/f1.jpeg'

# Assign the path for the potentially manipulated image
Image.open(fake_image_path)
```

```
17
18 # Open the potentially manipulated image for visualization
19 convert_to_ela_image(fake_image_path, 90)
20
21 # Analyze the potentially manipulated image using the ELA
     algorithm with a specified quality level
```

Listing 2: Output Image Analysis

Explanation:

- **Line 32-36:** Analyzing a real image using the ELA algorithm with a specified quality level.
- **Line 39-43:** Analyzing a potentially manipulated image using the ELA algorithm with a specified quality level.

Conclusion

The presented algorithm provides an effective means for detecting image forgery by leveraging ELA. By combining theoretical principles with practical implementation, it offers a valuable tool for forensic image analysis. Researchers and forensic experts can utilize this script to identify potential manipulations in digital images, ensuring the integrity of visual content.