Image Forgery Detection

Naman Jain, Dhaval Pathak & Parinita Singh

[1*]Computer Science Department, BML Munjal University, Gurgaon, Haryana, India.

Abstract

Background and Objectives: In the digital age, image manipulation, especially copy-move forgery, presents a significant challenge. This study addresses the need for precise forgery detection by utilizing the diverse CASIA V2 dataset. Applying advanced deep learning, specifically convolutional neural networks (CNNs), and innovative image processing methods, our objective is to accurately identify manipulated sections within images. Diverging from conventional methods, our approach with CNNs yields markedly improved detection performance, contributing significantly to digital forensics and ensuring image integrity in an era where authenticity is often questioned. **Material & Methods**: Our methodology employs the comprehensive CASIA V2 dataset for copy-move forgery detection. Utilizing advanced deep learning, particularly CNNs, and innovative image processing, we aim to enhance the accuracy of detecting altered sections within images. Distinguishing from conventional approaches, the emphasis on CNNs significantly improves detection performance. The study utilizes the diverse CASIA V2 dataset, with methods incorporating advanced deep learning and image processing techniques. This combination underscores our pursuit of an effective solution for identifying and combatting image forgeries in the digital realm. **Results**: The suggested system's effectiveness was evaluated using F-score, accuracy, precision, and recall metrics. The Dual U-Net model, enhanced with ELA, demonstrated significant advancements, revealing intricate features with tailored pre-processing. It achieved notable success in forged region detection, showcasing potential for robust image forgery identification. The results position both models as effective tools, with the Vanilla CNN particularly standing out in overall performance among the evaluated models.

Keywords: Forgery Detection, Deep Learning Techniques, Image Processing Methods, Convolutional Neural Networks (CNNs), Digital Forensics, and CASIA V2 Dataset.

1 Introduction

In the contemporary digital landscape, the proliferation of powerful image manipulation tools has presented a critical challenge, jeopardizing the integrity of digital images. The surge in accessibility to various digital manipulation tools has fuelled an alarming rise in the potential for misinformation, necessitating urgent measures to establish the authenticity of digital images. This challenge is particularly pronounced in an era where sharing manipulated images across social media platforms has become commonplace, creating a pressing need for reliable methods to verify the credibility of visual content.

The recognition of the critical importance of preserving the integrity of digital media underscores the significance of this project. As image editing software becomes increasingly sophisticated and accessible, the risk of compromising the authenticity of digital images grows. Manipulations like object removal, splicing, and copy-move forgery have become prevalent across various platforms, necessitating innovative solutions. Through the development of the Image Tampering Detection, this project seeks to not only address the immediate challenge of detecting manipulated images but also contribute to the broader goal of safeguarding trust and credibility in the digital realm. This endeavour is particularly crucial in domains such as forensics, journalism, and security, where the trustworthiness and credibility of digital images play a pivotal role.

To this background, in this paper, we propose the CNN and Dual unet model using ELA architecture, to address the escalating challenges posed by image tampering. The Image Forgery Detection system acts as a safeguard against misinformation, ensuring the authenticity and integrity of digital images in an era where distinguishing between real and modified photos is increasingly challenging. Thus, the contributions of the study are:

- Providing a trustworthy method to verify digital image authenticity.
- Proposing the classification methodology based on two deep learning approaches: CNN and Dual U-Net model using ELA, to identify forged images.

The paper is organized as follows. Section 2 provides an overview of the literature review. Section 3 describes the methodology and the dataset used for this study. Section 4 presents the results and discussion of the obtained results. Finally, the conclusions are presented in Section 5.

2 Related Work

The proliferation of powerful image manipulation tools in the digital landscape poses a critical challenge to the integrity of digital images *(Nikolaidis & Pitas).* The increasing accessibility to various digital manipulation tools has fuelled a rise in potential misinformation, prompting the need for urgent measures to authenticate digital images *(Koptyra & Ogiela)*. Particularly concerning is the commonplace sharing of manipulated images on social media platforms, highlighting the necessity for robust methods to ensure the credibility of visual content.

This research centers around the detection of manipulated images, advocating for the development of a robust image forgery detection system *(Wue et al.)*. The primary objective is to categorize manipulated images effectively, addressing the damage caused by the widespread sharing of such content on social media.

In response to the escalating challenges posed by digital manipulation tools, this research introduces an Image Tampering Detection system. Leveraging advanced digital image processing techniques, the system aims to discern alterations, manipulations, or forgeries within digital images *(Abdalla et al.)*. This effort is particularly crucial in domains such as forensics, journalism, and security, where the trustworthiness and credibility of digital images play a pivotal role.

The preservation of the integrity of digital media is emphasized as image editing software becomes more sophisticated and accessible *(Amerini et al.)*. Manipulations such as object removal, splicing, and copy-move forgery have become widespread, necessitating innovative solutions. Through the development of the Image Tampering Detection system, this research not only addresses the immediate challenge of detecting manipulated images but also contributes significantly to the broader goal of safeguarding trust and credibility in the digital realm.

Furthermore, the exploration of novel methodologies, such as Image chain, offers a user-friendly and versatile solution for different applications *(Dong et al.)*. The introduction of databases like CASIA and CoMoFoD provides essential resources for training and benchmarking detection algorithms *(Tralic et al.)*, enabling researchers to test the effectiveness of their methods under diverse scenarios. The amalgamation of traditional algorithms with advanced deep learning techniques, particularly Convolutional Neural Networks (CNNs) and Generative Adversarial Networks (GANs), reflects the dynamic evolution of this field *(Gupta et al.)*. Each research paper discussed in this literature review contributes uniquely to the field of digital image forgery detection.

One noteworthy approach is presented by Amerini et al. employing the Scale Invariant Feature Transform (SIFT) technique. Their method involves SIFT feature extraction, matching, hierarchical clustering, and geometric transformation estimation. By focusing on key point clustering, the algorithm efficiently identifies potential tampering areas, enhancing overall forgery detection accuracy *(Amerini et al.)*.

Wue et al. introduce the BusterNet architecture, a groundbreaking two-branch system that excels in detecting and differentiating source and target regions of manipulation. Leveraging Convolutional Neural Networks (CNNs), their model exhibits superior accuracy and speed, surpassing existing copy-move forgery detection algorithms *(Wue et al.)*.

Abdalla et al. pioneer a sophisticated combination of Generative Adversarial Networks (GANs) and Convolutional Neural Networks (CNNs) for copy-move forgery detection. Through GANs, they generate realistic forgeries, and CNNs analyse these images for similarity detection. The integration of GANs and CNNs proves effective, demonstrating the synergy between generative and analytical models in forgery detection *(Abdalla et al.)*.

Ali et al. focus on splicing and copy-move forgeries, presenting a novel technique combining image recompression and Convolutional Neural Networks (CNNs). Their approach exploits the differences in compression artifacts to highlight forged regions, and the CNN efficiently learns to identify these artifacts, resulting in a robust and efficient forgery detection system *(Ali et al.).*

Chakraborty and Dey introduce a dual-branch CNN architecture incorporating Error Level Analysis (ELA) and Spatial Rich Model (SRM) noise residuals. By processing ELA and noise residuals separately, their model achieves improved tampering detection. The dual-branch configuration enhances the network's ability to discern alterations, contributing to reliable forgery detection *(Chakraborty and Dey)*.

These methodologies collectively underscore the dynamic landscape of image forgery detection, showcasing advancements in key techniques, architectures, and integrative models. The evolving nature of digital manipulation necessitates continuous innovation in this field to ensure the integrity of visual content.

3 Methodology

3.1 Data-Set

For our project, we utilized CASIA V2, a dataset dedicated to forgery classification. It comprises 4,795 images, with a meticulously balanced distribution of 1,701 authentic and 3,274 forged instances.

This dataset is instrumental in advancing research on image forensics, specifically tailored for evaluating and enhancing forgery detection algorithms. Encompassing a diverse set of manipulation techniques, including copy-move, splicing, and retouching, CASIA V2 mirrors real-world challenges in digital forensics.

With a resolution that spans various content genres such as natural scenes, objects, and human subjects, CASIA V2 serves as a critical benchmark for researchers. Its utility extends to fostering the development of robust forgery detection models and promoting the continual evolution of techniques addressing the escalating concerns surrounding image authenticity and integrity.

3.2 Image Tampering Detection (Vanilla CNN)

3.2.1 Pre-processing

The image pre-processing pipeline employs Error Level Analysis (ELA) with a quality parameter of 85 to detect compression artifacts and potential manipulations. The process involves creating a temporary JPEG version of the image, calculating the ELA by differencing the original and compressed images, enhancing brightness based on the maximum difference, and scaling the brightness-enhanced image. The resulting 128x128 pixel array is flattened and normalized to values between 0 and 1, suitable for machine learning applications, particularly in image manipulation detection.

3.2.2 Error Level Analysis (ELA)

ELA was applied in conjunction with a vanilla CNN (Convolutional Neural Network) in our project. In our project, we employed Error Level Analysis (ELA), a forensic technique that identifies potential manipulation within digital images. ELA highlights variations in error levels resulting from compression and recompression. By saving an image at a specific compression level and analysing subsequent alterations, our project utilized ELA to detect inconsistencies in compression artifacts. This approach proved valuable for identifying edited or manipulated regions within the images, serving as a crucial tool for verifying image authenticity and detecting tampering in our research.

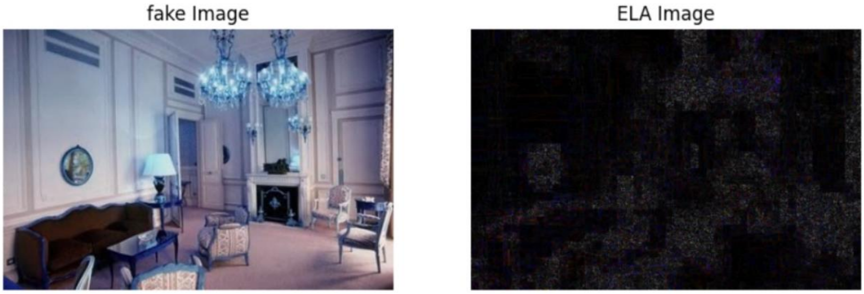
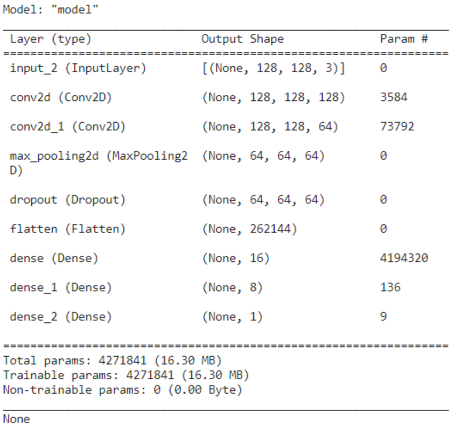


Fig. 3.1: Fake Image vs ELA Image

3.2.3 Model Architecture

The model comprises two convolutional layers (128 and 64 filters), a max pooling layer, dropout layer, and three dense layers (16, 8, and 1 neuron). The architecture is designed to identify patterns of varying sizes, reduce dimensionality, prevent overfitting, and facilitate generalization. The progressive dense layers suggest a decision-making process. With 4,271,841 total parameters, all trainable, the model is relatively simple yet effective for image recognition, leveraging convolutional and dense layers to capture complex relationships in the data.



```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 128, 128, 3)]     0

 conv2d (Conv2D)             (None, 128, 128, 128)     3584

 conv2d_1 (Conv2D)           (None, 128, 128, 64)      73792

 max_pooling2d (MaxPooling2  (None, 64, 64, 64)        0
 D)

 dropout (Dropout)           (None, 64, 64, 64)        0

 flatten (Flatten)           (None, 262144)            0

 dense (Dense)               (None, 16)                4194320

 dense_1 (Dense)             (None, 8)                 136

 dense_2 (Dense)             (None, 1)                 9

=================================================================
Total params: 4271841 (16.30 MB)
Trainable params: 4271841 (16.30 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

Fig. 3.2: The detailed architecture of proposed Vanilla CNN.

3.3 Forged Region Detection (Dual U-Net model)

3.3.1 Pre-processing

The image pre-processing for Forged Region Detection using a dual U-Net model begins by resizing both the input image and its corresponding mask to a standardized size of 512x512 pixels. The subsequent binary segmentation map creation in the mask involves assigning 0 to pixels with a value of 255, and 1 to non-255 pixels. These tailored pre-processing steps are fundamental for training the dual U-Net model, specifically designed for detecting forged regions in images, addressing tasks related to image manipulation or forgery detection.

3.3.2 SRM Filters

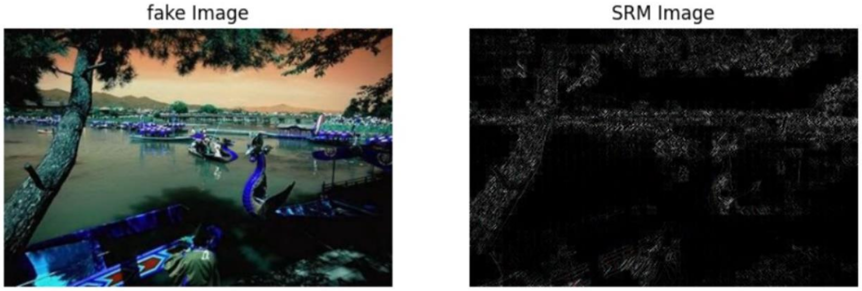
Spatial Rich Models (SRM) filters are weighted matrices applied to images, enhancing spatial features crucial for forged region detection in the dual U-Net model. These filters, incorporate specific weight distributions designed to highlight distinct image features.

- Filter1 is designed to capture vertical edges in an image by emphasizing changes in intensity along the vertical direction. Its application in forged region detection involves enhancing features related to vertical structures, specifically highlighting inconsistencies introduced by forged regions in vertical patterns.
  Filter1 Matrix = [[0, 0, 0, 0, 0],
  [0, -1, 2, -1, 0],
  [0, 2, -4, 2, 0],
  [0, -1, 2, -1, 0],
  [0, 0, 0, 0, 0]]

- Filter 2 is designed to highlight diagonal edges and gradients, beneficial for forged region detection by capturing inclined features. Its sensitivity to changes in diagonal intensity enhances the identification of potential tampering, contributing to the effectiveness of the dual U-Net model.
  Filter2 Matrix = [[-1, 2, -2, 2, -1],
  [2, -6, 8, -6, 2],
  [-2, 8, -12, 8, -2],
  [2, -6, 8, -6, 2],
  [-1, 2, -2, 2, -1]]

- Filter 3 focuses on horizontal edges, emphasizing changes in intensity. It captures features related to horizontal structures, highlighting inconsistencies introduced by forged regions. The weighted distribution enhances spatial information, contributing significantly to the model's ability to identify manipulated regions in the overall forged region detection process.

```
Filter3 = [[0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0],
           [0, 1, -2, 1, 0],
           [0, 0, 0, 0, 0],
           [0, 0, 0, 0, 0]]
```


fake Image

SRM Image

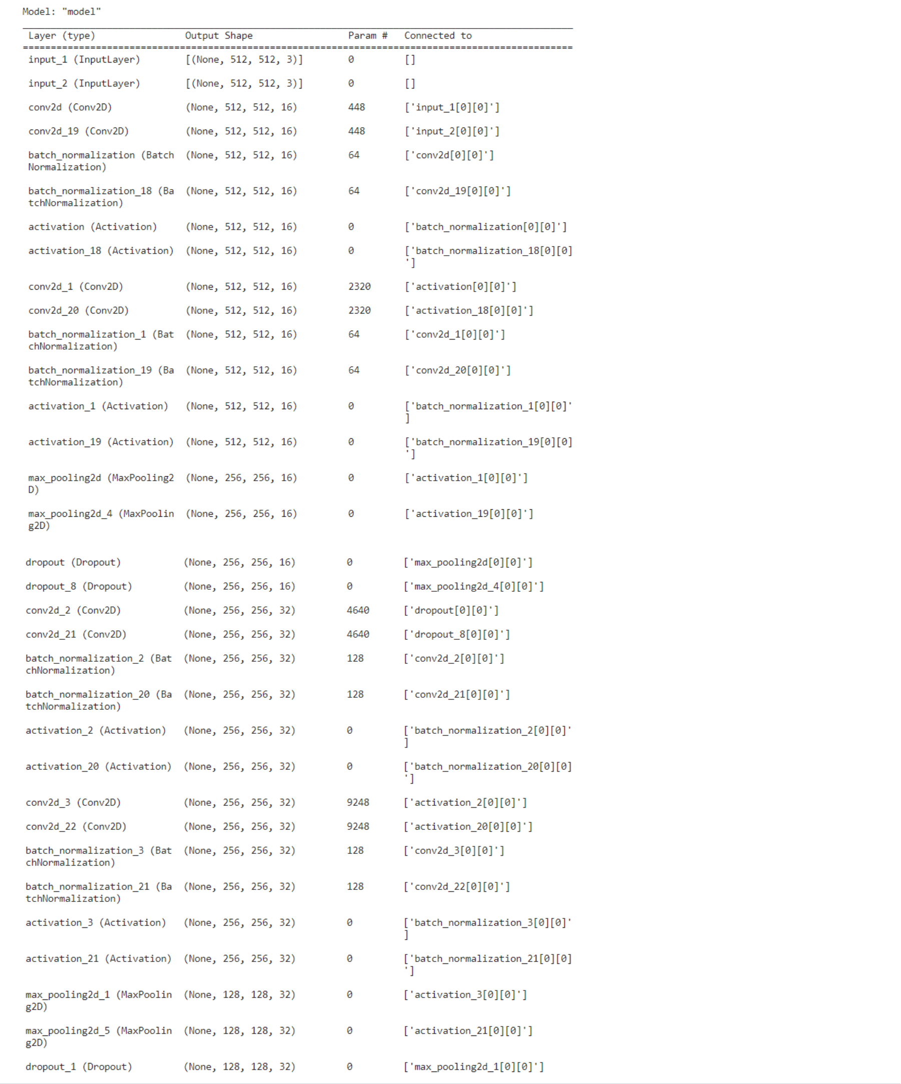
Fig. 3.3: Fake Image vs SRM Image

The SRM filters are applied using the OpenCV function on the input image, accentuating crucial features for subsequent stages of the dual U-Net model. Strategically employing these filters enables the dual U-Net model to leverage enhanced spatial features, enhancing its ability to discriminate between authentic and manipulated regions in forgery detection.

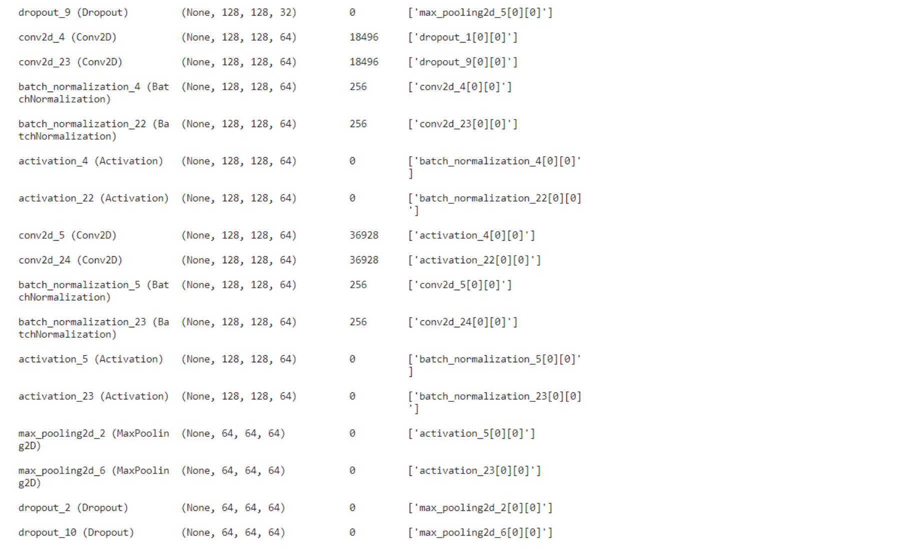
3.3.3 Model Architecture

The Dual U-Net model is a sophisticated architecture consisting of two parallel U-Net networks. The initial U-Net is responsible for extracting intricate features from input data, while the subsequent U-Net processes these features at a higher abstraction level for precise semantic segmentation. Skip connections between corresponding layers facilitate the seamless transfer of information, fostering robust feature localization and ensuring the model's capacity to discern and delineate complex structures within images.

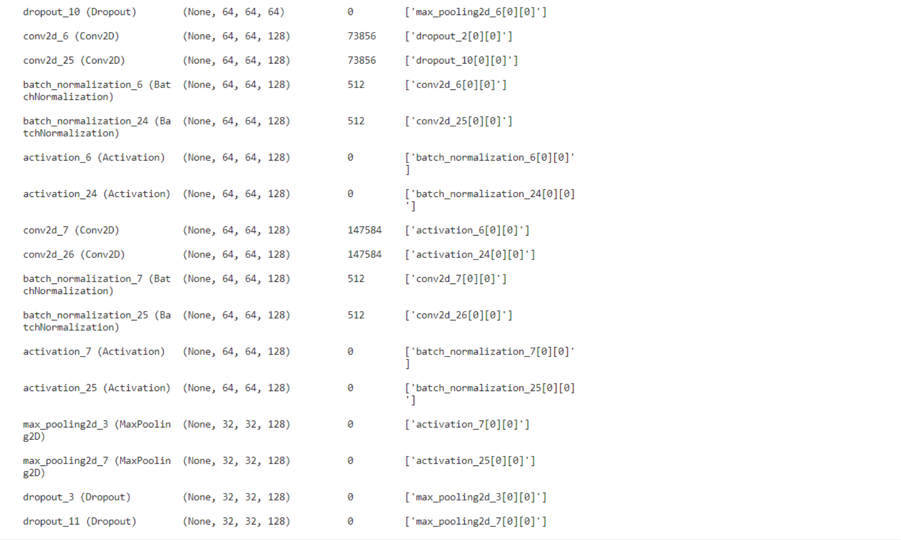
This dual-structure design enhances the model's overall performance in tasks such as image segmentation, especially in scenarios like forged region detection, where intricate details are vital.



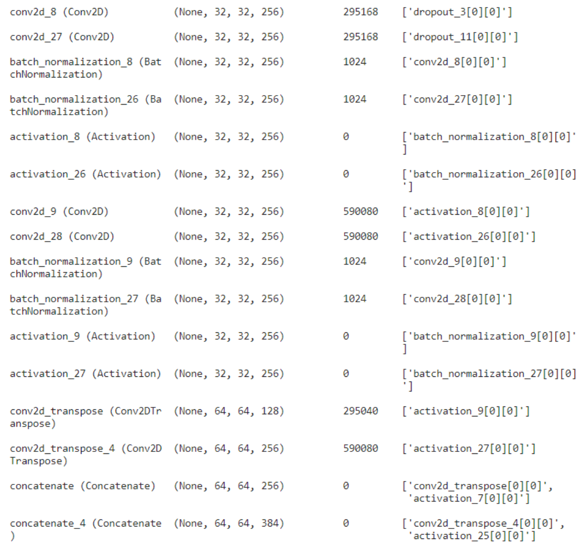
```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #     Connected to
==================================================================================================
 input_1 (InputLayer)        [(None, 512, 512, 3)]        0           []

 input_2 (InputLayer)        [(None, 512, 512, 3)]        0           []

 conv2d (Conv2D)             (None, 512, 512, 16)         448         ['input_1[0][0]']

 conv2d_19 (Conv2D)          (None, 512, 512, 16)         448         ['input_2[0][0]']

 batch_normalization (Batch  (None, 512, 512, 16)         64          ['conv2d[0][0]']
 Normalization)

 batch_normalization_18 (Ba  (None, 512, 512, 16)         64          ['conv2d_19[0][0]']
 tchNormalization)

 activation (Activation)     (None, 512, 512, 16)         0           ['batch_normalization[0][0]']

 activation_18 (Activation)  (None, 512, 512, 16)         0           ['batch_normalization_18[0][0]
                                                                      ']

 conv2d_1 (Conv2D)           (None, 512, 512, 16)         2320        ['activation[0][0]']

 conv2d_20 (Conv2D)          (None, 512, 512, 16)         2320        ['activation_18[0][0]']

 batch_normalization_1 (Bat  (None, 512, 512, 16)         64          ['conv2d_1[0][0]']
 chNormalization)

 batch_normalization_19 (Ba  (None, 512, 512, 16)         64          ['conv2d_20[0][0]']
 tchNormalization)

 activation_1 (Activation)   (None, 512, 512, 16)         0           ['batch_normalization_1[0][0]'
                                                                      ]

 activation_19 (Activation)  (None, 512, 512, 16)         0           ['batch_normalization_19[0][0]
                                                                      ']

 max_pooling2d (MaxPooling2  (None, 256, 256, 16)         0           ['activation_1[0][0]']
 D)

 max_pooling2d_4 (MaxPoolin  (None, 256, 256, 16)         0           ['activation_19[0][0]']
 g2D)


 dropout (Dropout)           (None, 256, 256, 16)         0           ['max_pooling2d[0][0]']

 dropout_8 (Dropout)         (None, 256, 256, 16)         0           ['max_pooling2d_4[0][0]']

 conv2d_2 (Conv2D)           (None, 256, 256, 32)         4640        ['dropout[0][0]']

 conv2d_21 (Conv2D)          (None, 256, 256, 32)         4640        ['dropout_8[0][0]']

 batch_normalization_2 (Bat  (None, 256, 256, 32)         128         ['conv2d_2[0][0]']
 chNormalization)

 batch_normalization_20 (Ba  (None, 256, 256, 32)         128         ['conv2d_21[0][0]']
 tchNormalization)

 activation_2 (Activation)   (None, 256, 256, 32)         0           ['batch_normalization_2[0][0]'
                                                                      ]

 activation_20 (Activation)  (None, 256, 256, 32)         0           ['batch_normalization_20[0][0]
                                                                      ']

 conv2d_3 (Conv2D)           (None, 256, 256, 32)         9248        ['activation_2[0][0]']

 conv2d_22 (Conv2D)          (None, 256, 256, 32)         9248        ['activation_20[0][0]']

 batch_normalization_3 (Bat  (None, 256, 256, 32)         128         ['conv2d_3[0][0]']
 chNormalization)

 batch_normalization_21 (Ba  (None, 256, 256, 32)         128         ['conv2d_22[0][0]']
 tchNormalization)

 activation_3 (Activation)   (None, 256, 256, 32)         0           ['batch_normalization_3[0][0]'
                                                                      ]

 activation_21 (Activation)  (None, 256, 256, 32)         0           ['batch_normalization_21[0][0]
                                                                      ']

 max_pooling2d_1 (MaxPoolin  (None, 128, 128, 32)         0           ['activation_3[0][0]']
 g2D)

 max_pooling2d_5 (MaxPoolin  (None, 128, 128, 32)         0           ['activation_21[0][0]']
 g2D)

 dropout_1 (Dropout)         (None, 128, 128, 32)         0           ['max_pooling2d_1[0][0]']
```



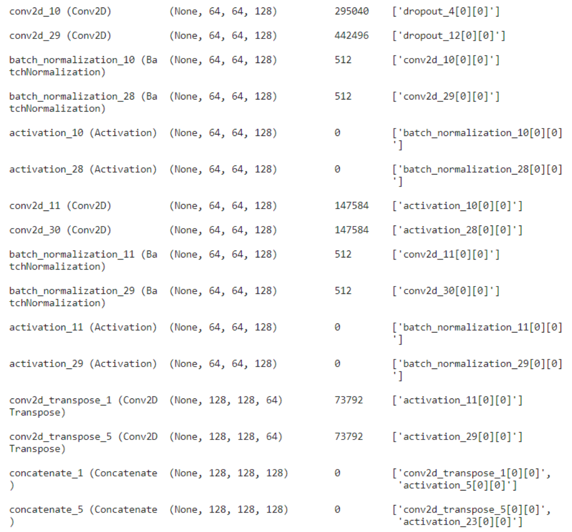
```
 dropout_9 (Dropout)         (None, 128, 128, 32)         0         ['max_pooling2d_5[0][0]']

 conv2d_4 (Conv2D)           (None, 128, 128, 64)         18496     ['dropout_1[0][0]']

 conv2d_23 (Conv2D)          (None, 128, 128, 64)         18496     ['dropout_9[0][0]']

 batch_normalization_4 (Bat  (None, 128, 128, 64)         256       ['conv2d_4[0][0]']
 chNormalization)

 batch_normalization_22 (Ba  (None, 128, 128, 64)         256       ['conv2d_23[0][0]']
 tchNormalization)

 activation_4 (Activation)   (None, 128, 128, 64)         0         ['batch_normalization_4[0][0]'
                                                                    ]

 activation_22 (Activation)  (None, 128, 128, 64)         0         ['batch_normalization_22[0][0]
                                                                    ']

 conv2d_5 (Conv2D)           (None, 128, 128, 64)         36928     ['activation_4[0][0]']

 conv2d_24 (Conv2D)          (None, 128, 128, 64)         36928     ['activation_22[0][0]']

 batch_normalization_5 (Bat  (None, 128, 128, 64)         256       ['conv2d_5[0][0]']
 chNormalization)

 batch_normalization_23 (Ba  (None, 128, 128, 64)         256       ['conv2d_24[0][0]']
 tchNormalization)

 activation_5 (Activation)   (None, 128, 128, 64)         0         ['batch_normalization_5[0][0]'
                                                                    ]

 activation_23 (Activation)  (None, 128, 128, 64)         0         ['batch_normalization_23[0][0]
                                                                    ']

 max_pooling2d_2 (MaxPoolin  (None, 64, 64, 64)           0         ['activation_5[0][0]']
 g2D)

 max_pooling2d_6 (MaxPoolin  (None, 64, 64, 64)           0         ['activation_23[0][0]']
 g2D)

 dropout_2 (Dropout)         (None, 64, 64, 64)           0         ['max_pooling2d_2[0][0]']

 dropout_10 (Dropout)        (None, 64, 64, 64)           0         ['max_pooling2d_6[0][0]']
```



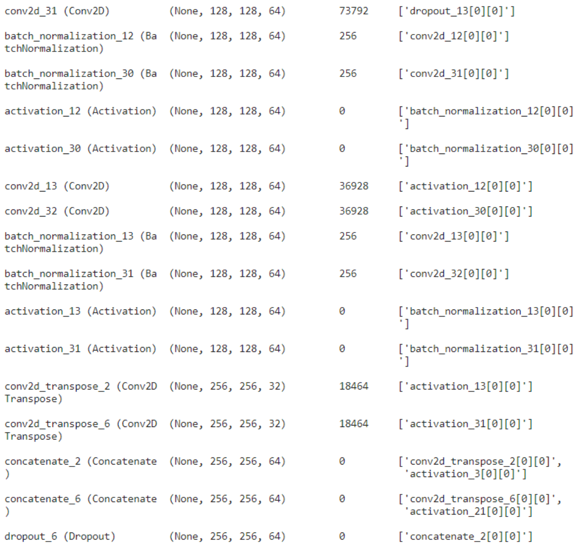
```
 dropout_10 (Dropout)        (None, 64, 64, 64)           0         ['max_pooling2d_6[0][0]']

 conv2d_6 (Conv2D)           (None, 64, 64, 128)          73856     ['dropout_2[0][0]']

 conv2d_25 (Conv2D)          (None, 64, 64, 128)          73856     ['dropout_10[0][0]']

 batch_normalization_6 (Bat  (None, 64, 64, 128)          512       ['conv2d_6[0][0]']
 chNormalization)

 batch_normalization_24 (Ba  (None, 64, 64, 128)          512       ['conv2d_25[0][0]']
 tchNormalization)

 activation_6 (Activation)   (None, 64, 64, 128)          0         ['batch_normalization_6[0][0]'
                                                                    ]

 activation_24 (Activation)  (None, 64, 64, 128)          0         ['batch_normalization_24[0][0]
                                                                    ']

 conv2d_7 (Conv2D)           (None, 64, 64, 128)          147584    ['activation_6[0][0]']

 conv2d_26 (Conv2D)          (None, 64, 64, 128)          147584    ['activation_24[0][0]']

 batch_normalization_7 (Bat  (None, 64, 64, 128)          512       ['conv2d_7[0][0]']
 chNormalization)

 batch_normalization_25 (Ba  (None, 64, 64, 128)          512       ['conv2d_26[0][0]']
 tchNormalization)

 activation_7 (Activation)   (None, 64, 64, 128)          0         ['batch_normalization_7[0][0]'
                                                                    ]

 activation_25 (Activation)  (None, 64, 64, 128)          0         ['batch_normalization_25[0][0]
                                                                    ']

 max_pooling2d_3 (MaxPoolin  (None, 32, 32, 128)          0         ['activation_7[0][0]']
 g2D)

 max_pooling2d_7 (MaxPoolin  (None, 32, 32, 128)          0         ['activation_25[0][0]']
 g2D)

 dropout_3 (Dropout)         (None, 32, 32, 128)          0         ['max_pooling2d_3[0][0]']

 dropout_11 (Dropout)        (None, 32, 32, 128)          0         ['max_pooling2d_7[0][0]']
```



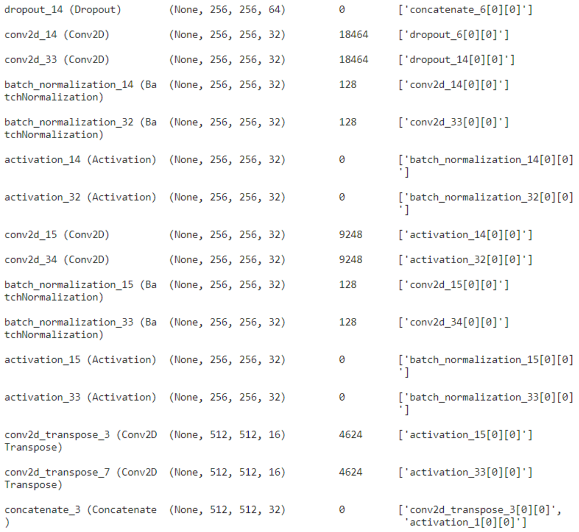
```
 conv2d_8 (Conv2D)           (None, 32, 32, 256)          295168    ['dropout_3[0][0]']

 conv2d_27 (Conv2D)          (None, 32, 32, 256)          295168    ['dropout_11[0][0]']

 batch_normalization_8 (Bat  (None, 32, 32, 256)          1024      ['conv2d_8[0][0]']
 chNormalization)

 batch_normalization_26 (Ba  (None, 32, 32, 256)          1024      ['conv2d_27[0][0]']
 tchNormalization)

 activation_8 (Activation)   (None, 32, 32, 256)          0         ['batch_normalization_8[0][0]'
                                                                     ]

 activation_26 (Activation)  (None, 32, 32, 256)          0         ['batch_normalization_26[0][0]
                                                                     ']

 conv2d_9 (Conv2D)           (None, 32, 32, 256)          590080    ['activation_8[0][0]']

 conv2d_28 (Conv2D)          (None, 32, 32, 256)          590080    ['activation_26[0][0]']

 batch_normalization_9 (Bat  (None, 32, 32, 256)          1024      ['conv2d_9[0][0]']
 chNormalization)

 batch_normalization_27 (Ba  (None, 32, 32, 256)          1024      ['conv2d_28[0][0]']
 tchNormalization)

 activation_9 (Activation)   (None, 32, 32, 256)          0         ['batch_normalization_9[0][0]'
                                                                     ]

 activation_27 (Activation)  (None, 32, 32, 256)          0         ['batch_normalization_27[0][0]
                                                                     ']

 conv2d_transpose (Conv2DTr  (None, 64, 64, 128)          295040    ['activation_9[0][0]']
 anspose)

 conv2d_transpose_4 (Conv2D  (None, 64, 64, 256)          590080    ['activation_27[0][0]']
 Transpose)

 concatenate (Concatenate)   (None, 64, 64, 256)          0         ['conv2d_transpose[0][0]',
                                                                      'activation_7[0][0]']

 concatenate_4 (Concatenate  (None, 64, 64, 384)          0         ['conv2d_transpose_4[0][0]',
 )                                                                    'activation_25[0][0]']
```



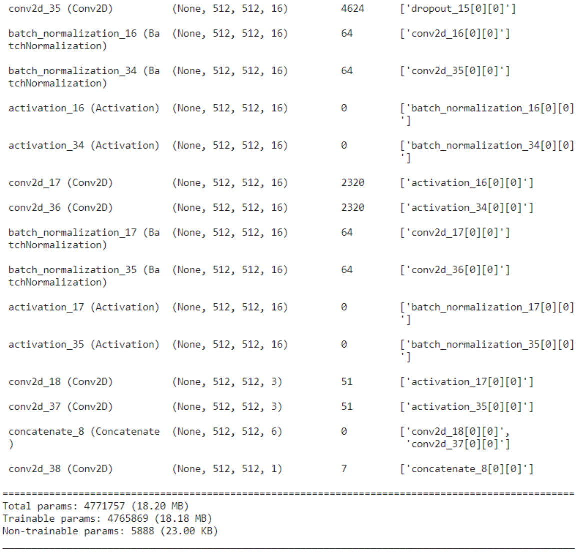
```
 conv2d_10 (Conv2D)          (None, 64, 64, 128)          295040    ['dropout_4[0][0]']

 conv2d_29 (Conv2D)          (None, 64, 64, 128)          442496    ['dropout_12[0][0]']

 batch_normalization_10 (Ba  (None, 64, 64, 128)          512       ['conv2d_10[0][0]']
 tchNormalization)

 batch_normalization_28 (Ba  (None, 64, 64, 128)          512       ['conv2d_29[0][0]']
 tchNormalization)

 activation_10 (Activation)  (None, 64, 64, 128)          0         ['batch_normalization_10[0][0]
                                                                     ']

 activation_28 (Activation)  (None, 64, 64, 128)          0         ['batch_normalization_28[0][0]
                                                                     ']

 conv2d_11 (Conv2D)          (None, 64, 64, 128)          147584    ['activation_10[0][0]']

 conv2d_30 (Conv2D)          (None, 64, 64, 128)          147584    ['activation_28[0][0]']

 batch_normalization_11 (Ba  (None, 64, 64, 128)          512       ['conv2d_11[0][0]']
 tchNormalization)

 batch_normalization_29 (Ba  (None, 64, 64, 128)          512       ['conv2d_30[0][0]']
 tchNormalization)

 activation_11 (Activation)  (None, 64, 64, 128)          0         ['batch_normalization_11[0][0]
                                                                     ']

 activation_29 (Activation)  (None, 64, 64, 128)          0         ['batch_normalization_29[0][0]
                                                                     ']

 conv2d_transpose_1 (Conv2D  (None, 128, 128, 64)         73792     ['activation_11[0][0]']
 Transpose)

 conv2d_transpose_5 (Conv2D  (None, 128, 128, 64)         73792     ['activation_29[0][0]']
 Transpose)

 concatenate_1 (Concatenate  (None, 128, 128, 128)        0         ['conv2d_transpose_1[0][0]',
 )                                                                    'activation_5[0][0]']

 concatenate_5 (Concatenate  (None, 128, 128, 128)        0         ['conv2d_transpose_5[0][0]',
 )                                                                    'activation_23[0][0]']
```



```
conv2d_31 (Conv2D)          (None, 128, 128, 64)      73792      ['dropout_13[0][0]']

batch_normalization_12 (Ba  (None, 128, 128, 64)      256        ['conv2d_12[0][0]']
tchNormalization)

batch_normalization_30 (Ba  (None, 128, 128, 64)      256        ['conv2d_31[0][0]']
tchNormalization)

activation_12 (Activation)  (None, 128, 128, 64)      0          ['batch_normalization_12[0][0]
                                                                  ']

activation_30 (Activation)  (None, 128, 128, 64)      0          ['batch_normalization_30[0][0]
                                                                  ']

conv2d_13 (Conv2D)          (None, 128, 128, 64)      36928      ['activation_12[0][0]']

conv2d_32 (Conv2D)          (None, 128, 128, 64)      36928      ['activation_30[0][0]']

batch_normalization_13 (Ba  (None, 128, 128, 64)      256        ['conv2d_13[0][0]']
tchNormalization)

batch_normalization_31 (Ba  (None, 128, 128, 64)      256        ['conv2d_32[0][0]']
tchNormalization)

activation_13 (Activation)  (None, 128, 128, 64)      0          ['batch_normalization_13[0][0]
                                                                  ']

activation_31 (Activation)  (None, 128, 128, 64)      0          ['batch_normalization_31[0][0]
                                                                  ']

conv2d_transpose_2 (Conv2D  (None, 256, 256, 32)      18464      ['activation_13[0][0]']
Transpose)

conv2d_transpose_6 (Conv2D  (None, 256, 256, 32)      18464      ['activation_31[0][0]']
Transpose)

concatenate_2 (Concatenate  (None, 256, 256, 64)      0          ['conv2d_transpose_2[0][0]',
)                                                                 'activation_3[0][0]']

concatenate_6 (Concatenate  (None, 256, 256, 64)      0          ['conv2d_transpose_6[0][0]',
)                                                                 'activation_21[0][0]']

dropout_6 (Dropout)         (None, 256, 256, 64)      0          ['concatenate_2[0][0]']
```



```
dropout_14 (Dropout)        (None, 256, 256, 64)      0          ['concatenate_6[0][0]']

conv2d_14 (Conv2D)          (None, 256, 256, 32)      18464      ['dropout_6[0][0]']

conv2d_33 (Conv2D)          (None, 256, 256, 32)      18464      ['dropout_14[0][0]']

batch_normalization_14 (Ba  (None, 256, 256, 32)      128        ['conv2d_14[0][0]']
tchNormalization)

batch_normalization_32 (Ba  (None, 256, 256, 32)      128        ['conv2d_33[0][0]']
tchNormalization)

activation_14 (Activation)  (None, 256, 256, 32)      0          ['batch_normalization_14[0][0]
                                                                  ']

activation_32 (Activation)  (None, 256, 256, 32)      0          ['batch_normalization_32[0][0]
                                                                  ']

conv2d_15 (Conv2D)          (None, 256, 256, 32)      9248       ['activation_14[0][0]']

conv2d_34 (Conv2D)          (None, 256, 256, 32)      9248       ['activation_32[0][0]']

batch_normalization_15 (Ba  (None, 256, 256, 32)      128        ['conv2d_15[0][0]']
tchNormalization)

batch_normalization_33 (Ba  (None, 256, 256, 32)      128        ['conv2d_34[0][0]']
tchNormalization)

activation_15 (Activation)  (None, 256, 256, 32)      0          ['batch_normalization_15[0][0]
                                                                  ']

activation_33 (Activation)  (None, 256, 256, 32)      0          ['batch_normalization_33[0][0]
                                                                  ']

conv2d_transpose_3 (Conv2D  (None, 512, 512, 16)      4624       ['activation_15[0][0]']
Transpose)

conv2d_transpose_7 (Conv2D  (None, 512, 512, 16)      4624       ['activation_33[0][0]']
Transpose)

concatenate_3 (Concatenate  (None, 512, 512, 32)      0          ['conv2d_transpose_3[0][0]',
)                                                                 'activation_1[0][0]']
```



```
 conv2d_35 (Conv2D)          (None, 512, 512, 16)        4624      ['dropout_15[0][0]']

 batch_normalization_16 (Ba  (None, 512, 512, 16)        64        ['conv2d_16[0][0]']
 tchNormalization)

 batch_normalization_34 (Ba  (None, 512, 512, 16)        64        ['conv2d_35[0][0]']
 tchNormalization)

 activation_16 (Activation)  (None, 512, 512, 16)        0         ['batch_normalization_16[0][0]
                                                                   ']

 activation_34 (Activation)  (None, 512, 512, 16)        0         ['batch_normalization_34[0][0]
                                                                   ']

 conv2d_17 (Conv2D)          (None, 512, 512, 16)        2320      ['activation_16[0][0]']

 conv2d_36 (Conv2D)          (None, 512, 512, 16)        2320      ['activation_34[0][0]']

 batch_normalization_17 (Ba  (None, 512, 512, 16)        64        ['conv2d_17[0][0]']
 tchNormalization)

 batch_normalization_35 (Ba  (None, 512, 512, 16)        64        ['conv2d_36[0][0]']
 tchNormalization)

 activation_17 (Activation)  (None, 512, 512, 16)        0         ['batch_normalization_17[0][0]
                                                                   ']

 activation_35 (Activation)  (None, 512, 512, 16)        0         ['batch_normalization_35[0][0]
                                                                   ']

 conv2d_18 (Conv2D)          (None, 512, 512, 3)         51        ['activation_17[0][0]']

 conv2d_37 (Conv2D)          (None, 512, 512, 3)         51        ['activation_35[0][0]']

 concatenate_8 (Concatenate  (None, 512, 512, 6)         0         ['conv2d_18[0][0]',
 )                                                                  'conv2d_37[0][0]']

 conv2d_38 (Conv2D)          (None, 512, 512, 1)         7         ['concatenate_8[0][0]']

==================================================================================================
Total params: 4771757 (18.20 MB)
Trainable params: 4765869 (18.18 MB)
Non-trainable params: 5888 (23.00 KB)
__________________________________________________________________________________________________
```

Fig. 3.4: The detailed architecture of Dual U-Net Model.

The Intersection over Union (IoU) score, serves as a crucial metric for training models, particularly in the context of image segmentation tasks. By calculating the overlap between the projected and actual segmentation masks, this metric assesses how accurate the model's predictions are. A higher IoU score indicates better agreement between the predicted and actual regions of interest.

In the training process, the IoU score is commonly employed as a loss function or evaluation metric, guiding the model to improve its ability to accurately delineate objects or regions in images, crucial for tasks such as semantic segmentation or object detection.

4 Results

The table below illustrates the comparative outcomes of various trained models, specifically designated for analysing their effectiveness in the classification of tampered images.

Table 4.1: Comparing models for tampered image classification.

Models	Accuracy	Precision	Recall	F1 Score
Vanilla Model	92.09 %	97.16 %	92.78 %	94.92 %
MobileNet	70 %	78 %	64 %	70 %
VGG16	66 %	68 %	75 %	71 %
ResNet50	46 %	51%	57 %	54 %
DenseNet121	60 %	42 %	50 %	45 %

Among the various models evaluated, the Vanilla model demonstrated superior performance, emerging as the most effective with an accuracy rate of 92.09% out of the five models considered.

The original image undergoes a meticulous tampering detection process using Error Level Analysis (ELA) and Spatial Rich Model (SRM). The combined results of ELA and SRM form a Predicted Forged Region Mask, indicating areas prone to tampering. The overall probability of a Fake Image, at 91.53% in the example given in figure, strongly suggests potential manipulation.

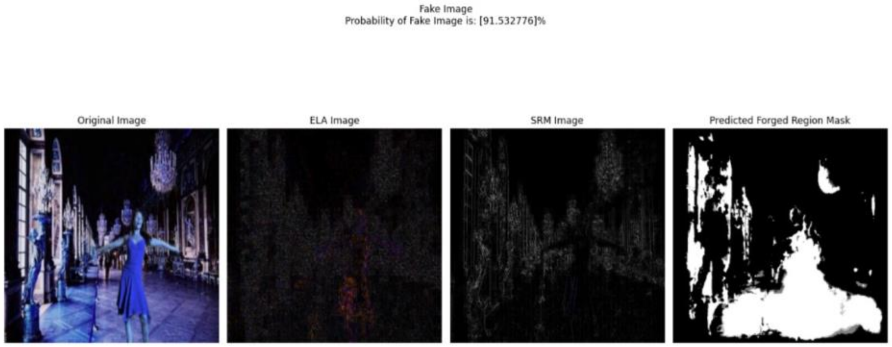

Fake Image
Probability of Fake Image is: [91.532776]%
Original Image
ELA Image
SRM Image
Predicted Forged Region Mask

Fig. 4.1: Predicting Possible Forged Region

The table below compares the outcomes of the approach we adopted with those of approaches taken in related works pertaining to our project.

Table 4.2: Comparing models from related works.

Related Works	Accuracy	Precision	Recall	F1 Score
Our approach	92.09%	97.16%	92.78%	94.92%
Detection of Image Tampering Using Deep Learning, Error Levels & Noise Residuals [2]	98.55%	98.71%	99.26%	98.98%
Image Forgery Detection Using Deep Learning by Recompressing Images [3]	92.23%	85%	97%	91.08%
An efficient copy move forgery detection using deep learning feature extraction and matching algorithm [4]	95%	98%	89%	93.28%
Image Manipulation Detection using Convolutional Neural Network [5]	90.5%	87.4%	92.3%	89.8%

4.1 Vanilla CNN Model

The integration of Error Level Analysis (ELA) with a Vanilla Convolutional Neural Network (CNN) in our image tampering detection approach produced notable outcomes. ELA, applied with a quality parameter of 85 during pre-processing, effectively identified compression artifacts and potential manipulations. The combination of ELA with the Vanilla CNN showcased its capability in revealing inconsistencies induced by compression and recompression, enhancing our ability to detect potential image tampering.

The Vanilla model exhibited remarkable performance, achieving an accuracy of 92.09%, precision of 97.16%, recall of 92.78%, and an F1 Score of 94.92%. Among the evaluated models, the Vanilla CNN emerged as the most effective, underscoring its proficiency in image manipulation detection. These results position the Vanilla CNN as a valuable tool for authenticating images and detecting tampering in research applications.

The Confusion Matrix Heatmap [Fig. 4.2] visually assesses model performance by displaying true positives, true negatives, false positives, and false negatives, aiding clarity in classification evaluation.

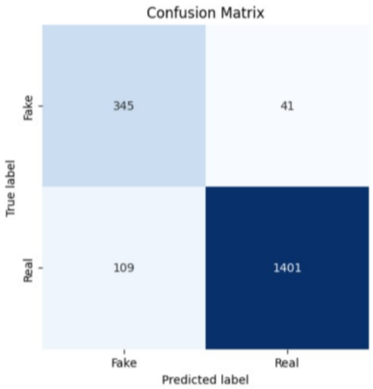


Fig. 4.2: Confusion Matrix Heatmap: Model Performance

AUC of 0.91 in the ROC curve [Fig. 4.3] indicates strong discrimination ability, effectively distinguishing true positives and negatives. The model demonstrates proficiency in accurate classification, making it a robust binary classifier.

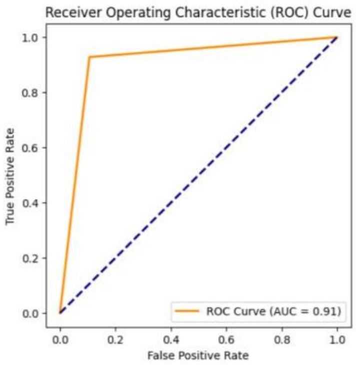


Fig. 4.3: Receiver Operating Characteristics (ROC)

The precision-recall curve [Fig. 4.4] shows that the model has a high precision, but a low recall. This means that the model is good at identifying true positives, but it misses several true positives.

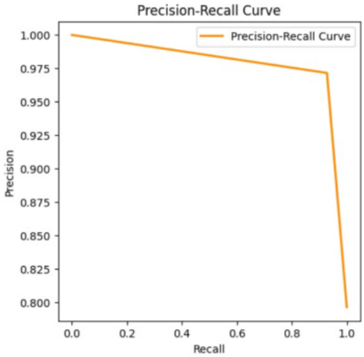


Fig. 4.4: Precision-Recall Curve

4.2 Dual U-Net Model using ELA

The implementation of the Dual U-Net model, coupled with Error Level Analysis (ELA), for Forged Region Detection, showcased significant advancements in image tampering identification. The tailored pre-processing, involving resizing and binary segmentation map creation, established a foundation for the dual U-Net's training specifically designed for forged region detection. The integration of Spatial Rich Models (SRM) filters further enriched the model's ability to discern crucial spatial features essential for detecting forged regions. The sophisticated architecture of the Dual U-Net, comprising two parallel networks with skip connections, demonstrated exceptional performance in intricate feature extraction and precise semantic segmentation.

These results underscore the efficacy of the Dual U-Net model in forged region detection, showcasing its potential for robust image manipulation detection and forgery identification in diverse scenarios.

5 Conclusion

In conclusion, this project addresses the escalating challenges posed by digital image manipulation in the contemporary digital landscape. The proliferation of powerful image manipulation tools has created a pressing need for reliable methods to verify the authenticity of digital images, especially in the context of widespread misinformation. Leveraging advanced deep learning techniques, including the Vanilla CNN model with Error Level Analysis (ELA) and the Dual U-Net model, this study makes a substantial contribution to the field of image tampering detection. The proposed models demonstrated superior performance, with the Vanilla CNN standing out as the most effective among the evaluated models. The robustness of these models in detecting forged regions and potential manipulations highlights their potential applications in diverse domains, such as forensics, journalism, and security. Overall, this project offers a trustworthy solution for verifying digital image authenticity and contributes to the broader goal of safeguarding trust and credibility in the digital realm.

References

[1] Wu, Y., Abd-Almageed, W., & Natarajan, P. (2018). Busternet: Detecting copy-move image forgery with source/target localization. In *Proceedings of the European conference on computer vision (ECCV)* (pp. 168-184).

[2] Chakraborty, S., Chatterjee, K., & Dey, P. (2022). Detection of Image Tampering Using Deep Learning, Error Levels & Noise Residuals.

[3] Ali, S. S., Ganapathi, I. I., Vu, N. S., Ali, S. D., Saxena, N., & Werghi, N. (2022). Image forgery detection using deep learning by recompressing images. *Electronics*, *11*(3), 403.

[4] Agarwal, R., & Verma, O. P. (2020). An efficient copy move forgery detection using deep learning feature extraction and matching algorithm. *Multimedia Tools and Applications*, *79*(11-12), 7355-7376.

[5] Kim, D. H., & Lee, H. Y. (2017). Image manipulation detection using convolutional neural network. *International Journal of Applied Engineering Research*, *12*(21), 11640-11646.

[6] Amerini, I., Ballan, L., Caldelli, R., Del Bimbo, A., & Serra, G. (2011). A sift-based forensic method for copy–move attack detection and transformation recovery. *IEEE transactions on information forensics and security*, *6*(3), 1099-1110.

[7] Tralic, D., Zupancic, I., Grgic, S., & Grgic, M. (2013, September). CoMoFoD—New database for copy-move forgery detection. In *Proceedings ELMAR-2013* (pp. 49-54). IEEE.

[8] Abdalla, Y., Iqbal, M. T., & Shehata, M. (2019). Copy-move forgery detection and localization using a generative adversarial network and convolutional neural-network. *Information*, *10*(9), 286.

[9] Gupta, P., Rajpoot, C. S., Shanthi, T. S., Prasad, D., Kumar, A., & Kumar, S. S. (2022, October). Image Forgery Detection using Deep Learning Model. In *2022 3rd International Conference on Smart Electronics and Communication (ICOSEC)* (pp. 1256-1262). IEEE.

[10] Dong, J., Wang, W., & Tan, T. (2013, July). Casia image tampering detection evaluation database. In *2013 IEEE China summit and international conference on signal and information processing* (pp. 422-426). IEEE.