IMAGE FORGERY DETECTION AND AUTHENTICATION

Karim Tarek (195322)
Dr. Motasem Mahmoud Elshourbagy

Faculty of Engineering, Electrical and Communication Engineering Department, The British University in Egypt, El Sherouk City, Misr-Ismalia Road, Cairo, Egypt
Emails: karim195322@bue.edu.eg & motasem.elshourbagy@bue.edu.eg

ABSTRACT

In the modern digital age, the rise of image manipulation tools raises significant concerns about the authenticity of digital images. The focus of this research is developing and evaluating a Convolutional Neural Network (CNN) based architecture that utilises Error Level Analysis (ELA) and Spatial Rich Model (SRM) filtering for image forgery detection.

The proposed dual-branch CNN model utilises images processed through ELA and SRM filters, enhancing the neural network's ability to identify subtle discrepancies indicative of manipulation in image data. Extensive experimentation was conducted on the most prominent datasets in image forgery detection field, including CASIA V2, CoMoFoD (small) and MICC-F2000.

Evaluation metrics such as accuracy, precision, recall and F1 score were used to benchmark the performance of the proposed model on the datasets mentioned. Notably, the model achieved near state-of-art-results with an accuracy of 98.00% on the MICC-F2000 dataset and the approach exhibits superior generalization capability across different types of image forgery and outperforming traditional methods. The findings highlight the potential of integrating ELA and SRM filtering in deep learning frameworks to enhance image forgery detection and establish a solid methodology for future advancements in the field.

1. Problem Definition

Image editing tools have become widespread and easily available for use, the authenticity of digital images is increasingly under threat. The decrease of confidence in the authenticity of digital images leads to misinformation and diminishes trust in digital media. Therefore, there is a critical need for advanced algorithms that reliably detects various types of image forgery to ensure the integrity of digital images. These algorithms are essential for purposes such as verification of the authenticity of images in a court setting.

2. Objectives

1. To develop an innovative method for image forgery detection using deep learning frameworks
2. Find the most appropriate preprocessing of images to improve the image forgery detection capabilities of deep learning methods.
3. Evaluate the performance of the developed algorithm across multiple prominent datasets, ensuring generalization capabilities to novel images
4. The developed algorithm should be able to achieve near state-of-the-art accuracy in detecting forgeries on the CASIA V2, CoMoFoD (small), and MICC-F2000 datasets.
5. Compare the proposed algorithm's performance on the on CASIA V2, CoMoFoD (small), and MICC-F2000 datasets with the existing state-of-the-art methods.
6. To provide insights and recommendations on future work that can done based on the developed algorithms.

3. Brief Introduction

3.1. Background

The ease and spread of image manipulation software has raised concerns about the authenticity of images that are exchanged. The concerns around the authenticity of digital images have resulted in the emergence of the field of digital image forgery. The creation of reliable digital image forgery detection and authentication algorithms improves the quality of life and dependability for workers of jobs that rely on information embedded in digital images such as lawyers, judges, law enforcement and health professionals.

Image Forgery detection approaches are classified into two primary approaches: active and passive approaches. Active image forgery detection relies on the pre-embedded information in the image, such as digital signatures or watermarks, which can be checked for the authenticity of the image. Digital signatures are embedded during the capture or processing stages of the image and are verified for any corruption during detection. Digital watermarking, often invisible, is altered if the image is manipulated, and any discrepancy during watermark extraction indicates forgery (Gill et al., 2017)

3.2. Traditional Image forgery detection

traditional image forgery techniques are the techniques that analyse inconsistencies in the physical and geometrics properties of an image these methods include:

- Pixel-Based Techniques: Techniques such as Error Level Analysis (ELA) emphasize the compression levels that suggest image manipulation (Krawetz, 2007).

- Camera-Based Techniques: These techniques identify image forgery by analysing the meta of the image or the camera noise patterns of the image

- Geometric-Based Methods: Techniques that rely on the geometric qualities of an image to detect inconsistencies such as detecting inconsistencies in the lighting, shadows, or the perspective of an image.

3.3. Deep Learning-Based Image forgery detection

Deep learning methods, especially Convolutional Neural Networks (CNNs), has proven to be superior at image forgery detection than traditional methods, CNNs can process visual data and learn to recognize patterns indicative of image manipulation.

Support Vector Machines (SVMs) have been also utilised in this field but CNNs have been proven superior in their image forgery detection capabilities.

4. Image forgery types

1. Copy-move: A region of an image is copied and pasted onto another region in the same image.
2. Splicing: a region of the image is copied and pasted onto a region in a different image
3. Image retouching: Retouching is subtly enhancing certain features in an image mainly for aesthetic appeal. This type of forgery is usually harmless

5. Summary of Previous Work

(Gupta et al., 2022) and (Sudiatmika et al., 2019)) highlighted the effectiveness of using ELA to preprocesses image to highlight inconsistencies indicative of image forgery detection. Thus, enhancing CNN's ability to detect forgery by emphasizing compression discrepancies in images.

(Joshi et al., 2022) demonstrated that preprocessing images using ELA and using pre-trained models as a feature extractor was an effective method of forgery detection. VGG-19 was the pre-trained model that produced the best performance metrics.

(Martin-Rodriguez et al., 2023) utilised pixel-wise feature extraction using ELA and Photo-Response Non-Uniformity (PRNU) for image preprocessing to differentiate artificial intelligence generated images from real ones achieving up to 99% accuracy.

(Mehrjardi et al., 2023) authored a very thorough survey of image forgery detection methods. Their work highlighted the advancements in the field and the various methods of image forgery detection

(Chakraborty et al., 2024) proposed an innovative method utilising a dual-branch CNN using ELA and SRM filtering to preprocess the input images. Achieving an impressive accuracy of 98.55%. The author’s method is effective and efficient in detecting various types of image forgeries.

6. Research Methodology

6.1. Dataset choice

For training and evaluating the deep learning model, three datasets were selected: CASIA V2, CoMoFoD (Small), and MICC-F2000.

CASIA V2: This dataset includes 12,614 images, of which there are 7491 authentic images, 5123 manipulated images. To utilise ELA to detect compression artifacts the JPEG format is crucial. There are over 9000 JPEG images in this dataset making it a good fit for the requirements of the model. Contains Copy-Move and Splicing forgeries.

CoMoFoD (Small): exclusively focused on Copy-Move forgeries. 10000 JPEG images, 5000 authentic images and 5000 forged images. Images are forged through various forms of transformations such as blurring, rotation and scaling, excellent choice for generalization.

MICC-F2000: includes high-resolution images (up to 2048x1536), MICC-F2000 has various transformations like scaling and rotation, it is designed to simulate real-world forgeries. Post-processing blurs boundaries to challenge the model in detecting subtle manipulations. High resolution images are effective for feature extraction

6.2. Error Level Analysis (ELA)

ELA detects subtle signs of image forgery by analysing compression artifacts across an image. ELA highlights regions with unusual compression levels. ELA involves saving an image at a lower quality, 90% was used my method. Then finding the difference between the saved image and the original image. ELA images have details that indicate may manipulation.

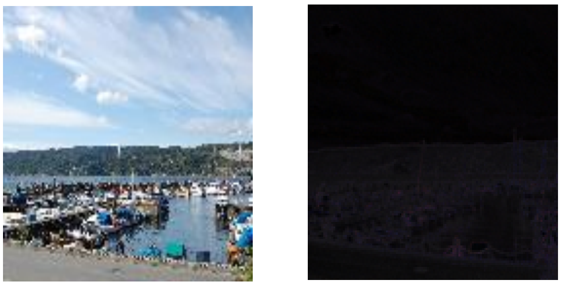


Figure 1: Example of an image processed with ELA (original left ELA right)

6.3. SRM (Spatial Rich Model) Filters

An image is filtered through 30 high SRM high pass proposed by (Fridrich & Kodovský, 2012). This process extracts subtle inconsistencies in the data that are not visible in the original image. The result is the noise residuals that contain isolated noise patterns of the image that may indicate manipulation.

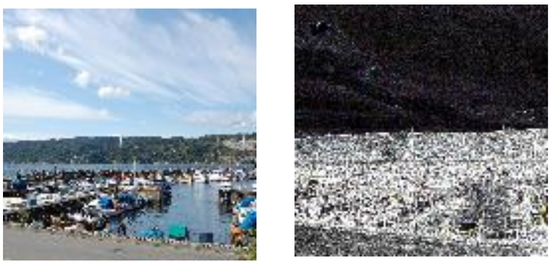


Figure 2: An example of an image processed with SRM (original left SRM right)

6.4. Convolutional Neural Networks (CNNs)

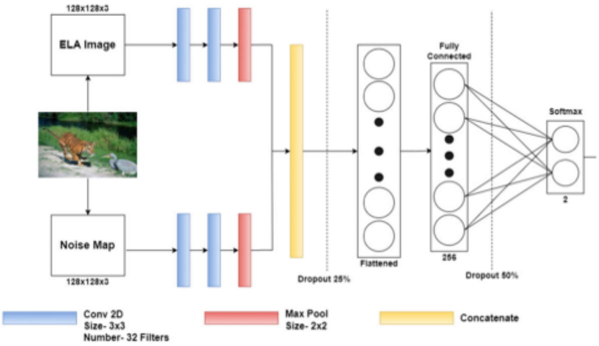
Convolutional Neural Networks (CNNs) are neural networks that contain at least one convolution layer. They are the most used type of neural networks for extracting features from visual data such as images. They were first introduced by LeCunn et. al. (1989) and CNNs are the state of the art when it comes to visual data extraction.

My proposed method’s architecture contains each of the following:

- A Convolution layer: this layer Applies filters or kernels to the input image, to allow the network to extract features like edges, patterns, and textures to classify images correctly.
- A Pooling layer: Reduces the spatial dimensions of the input to decrease computational complexity and prevent overfitting while preserving important feature data.
- A Fully connected layer: this Processes the features that are passed from the convolution and pooling layers to be able to decide the final class of the data.
- Softmax layer: Typically found at the end of the network, this layer outputs a 1 for real images or a 0 for forged images.
- Dropout layer: this layer implement a regularization technique to reduce

overfitting and improve the generalization capabilities of the model.

6.5. Proposed Model Architecture



The proposed CNN architecture is a dual branch CNN architecture, a branch receives a ELA analysis image resized to 128x128 pixels resolution, the other branch recieves an image filtered with 30 high pass SRM filters and resized to 128x128 pixels resolution. The images are processed by 2 convolution layers with 32 filters each with a kernel size of 3x3, then the images go through a max pooling layer to reduce the spatial dimensions of the image. The output is concatenated and a dropout layer with a dropout rate of 0.25 follow the max pooling layer. The output then is flattened to then passes through a dense layer with 256 units using ReLU activation to learn complex patterns. The final layer is a 2-unit dense layer with a SoftMax function that outputs 1 if the image is real or 0 if the image is forged. The model was trained and evaluated using python with the TensorFlow library and optimized with class weights to address class balances,

7. Experimental Work and Main Results

The model was trained and evaluated on three different prominent datasets with an 80-20 train-test split, the results are show in Table 1. The state-of-the-art results for CASIA V2 accuracy according to (Zanardelli et al., 2022), is 99.07%. The proposed model achieved around 96.18% accuracy which is almost state of the art and within the margin of error of the state-of-the-art. the model could achieve higher accuracy, but time and computational power were limited. As for the MICC-F2000 dataset the model outperformed the state-of-the-art metrics mentioned by (Koul et al., 2022) making this approach the state of the art for the MICC-F2000 image forgery detection dataset.

Table 1 also showcases the improvement of the addition of SRM filtering to ELA for image preprocessing. There are improvements to metrics all around except for precision on the MICC-F2000 dataset which can be explained by the small size of the test data (200). Thus, there is a large margin of error for metrics MICC-F2000 as it is a small dataset

Metric	Dataset	ELA+SRM Accuracy (%)	Improvement (ELA+SRM - ELA) (%)
Accuracy	CASIA V2	96.18	1.59
Precision	CASIA V2	96.92	-0.09
Recall	CASIA V2	98.23	2.29
F1 Score	CASIA V2	97.57	1.1
Accuracy	CoMoFod (small)	89.25	0.5
Precision	CoMoFod (small)	88.52	0.37
Recall	CoMoFod (small)	90.2	0.23
F1 Score	CoMoFod (small)	89.35	0.3
Accuracy	MICC-F2000	98	0.75
Precision	MICC-F2000	94.59	-5.41
Recall	MICC-F2000	100	4.38
F1 Score	MICC-F2000	97.22	-0.54

Table 1: Evaluating the improvement from the dual branch implementation

8. Conclusion

integrating ELA and SRM filters within a deep learning framework significantly enhances the model's capability to detect image forgeries in comparison with the use of ELA exclusively. The processing of the image by both ELA and SRM filters the compression discrepancies are amplified and can then be easily identified by the dual branch CNN. Moreover, this approach does not only capture more detail but also it improves performance on novel images due to picking up more detail indicative of image manipulation in the hidden data of the noise residual image.

Extensive experimentation was conducted on prominent datasets, including CASIA V2, CoMoFoD (small), and MICC-F2000. These experiments concluded the efficacy of the proposed model as the performance of the dual branch CNN model achieves near state-of-the-art performance with accuracy rates reaching up to 98.00% on the MICC-F2000 dataset and 96.18% on the CASIA V2 dataset. This is particularly significant because it highlights how underutilised the combination of SRM and ELA in deep learning frameworks.

9. Future work

The methodologies and insights presented in this paper will be instrumental in maintaining the originality of images in the digital world in the future.

Recommendations for Future Work:

1. Research the use of a pre-trained model with ImageNet weights or YOLO weights as a feature extractor from the ELA and SRM images, this can greatly enhance the model's accuracy and generalization ability. This could not be researched in this work due to computational limitations and lack of funding for cloud computing.
2. Localization and description: I aimed to utilize ELA and SRM images to develop models that are capable of localization and description of image forgeries in natural language utilizing transformers. The features extracted from ELA and SRM enhance the learnability of localization greatly for deep learning models. This was not possible in this research due to
3. Exploration of other preprocessing techniques: ELA and SRM are a powerful combination to extract patterns that are indicative of image manipulation, the efficacy of other preprocessing techniques combined with these two techniques must be studied to establish if there are any improvements with another technique added to this combination.
4. Generalization: Testing the image forgery detection algorithm on large real-world images was not possible due to a lack of computational power. This should be done in the future with large datasets for training and testing Such as the ImageNet dataset which contains over a million images.

References

Chakraborty, S., Chatterjee, K., & Dey, P. (2024). Detection of Image Tampering Using Deep Learning, Error Levels and Noise Residuals. *Neural Processing Letters*, *56*(2). https://doi.org/10.1007/s11063-024-11448-9

Fridrich, J., & Kodovský, J. (2012). *Rich Models for Steganalysis of Digital Images*.

Gill, N. K., Garg, R., & Doegar, E. A. (2017, December 13). A review paper on digital image forgery detection techniques. *8th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2017*. https://doi.org/10.1109/ICCCNT.2017.8203904

Gupta, A., Joshi, R., & Laban, R. (2022). *Detection of Tool based Edited Images from Error Level Analysis and Convolutional Neural Network*. http://arxiv.org/abs/2204.09075

Koul, S., Kumar, M., Khurana, S. S., Mushtaq, F., & Kumar, K. (2022). An efficient approach for copy-move image forgery detection using convolution neural network. *Multimedia Tools and Applications, 81*(8), 11259–11277. https://doi.org/10.1007/s11042-022-11974-5

Krawetz, N. (2007). *A Picture's Worth... Digital Image Analysis and Forensics*. www.hackerfactor.com

Martin-Rodriguez, F., Garcia-Mojon, R., & Fernandez-Barciela, M. (2023). Detection of AI-Created Images Using Pixel-Wise Feature Extraction and Convolutional Neural Networks. *Sensors*, *23*(22). https://doi.org/10.3390/s23229037

Mehrjardi, F. Z., Latif, A. M., Zarchi, M. S., & Sheikhpour, R. (2023). A survey on deep learning-based image forgery detection. *Pattern Recognition*, *144*. https://doi.org/10.1016/j.patcog.2023.109778

Sudiatmika, I. B. K., Rahman, F., Trisno, & Suyoto. (2019). Image forgery detection using error level analysis and deep learning. *Telkomnika (Telecommunication Computing Electronics and Control), 17*(2), 653–659. https://doi.org/10.12928/TELKOMNIKA.V17I2.8976

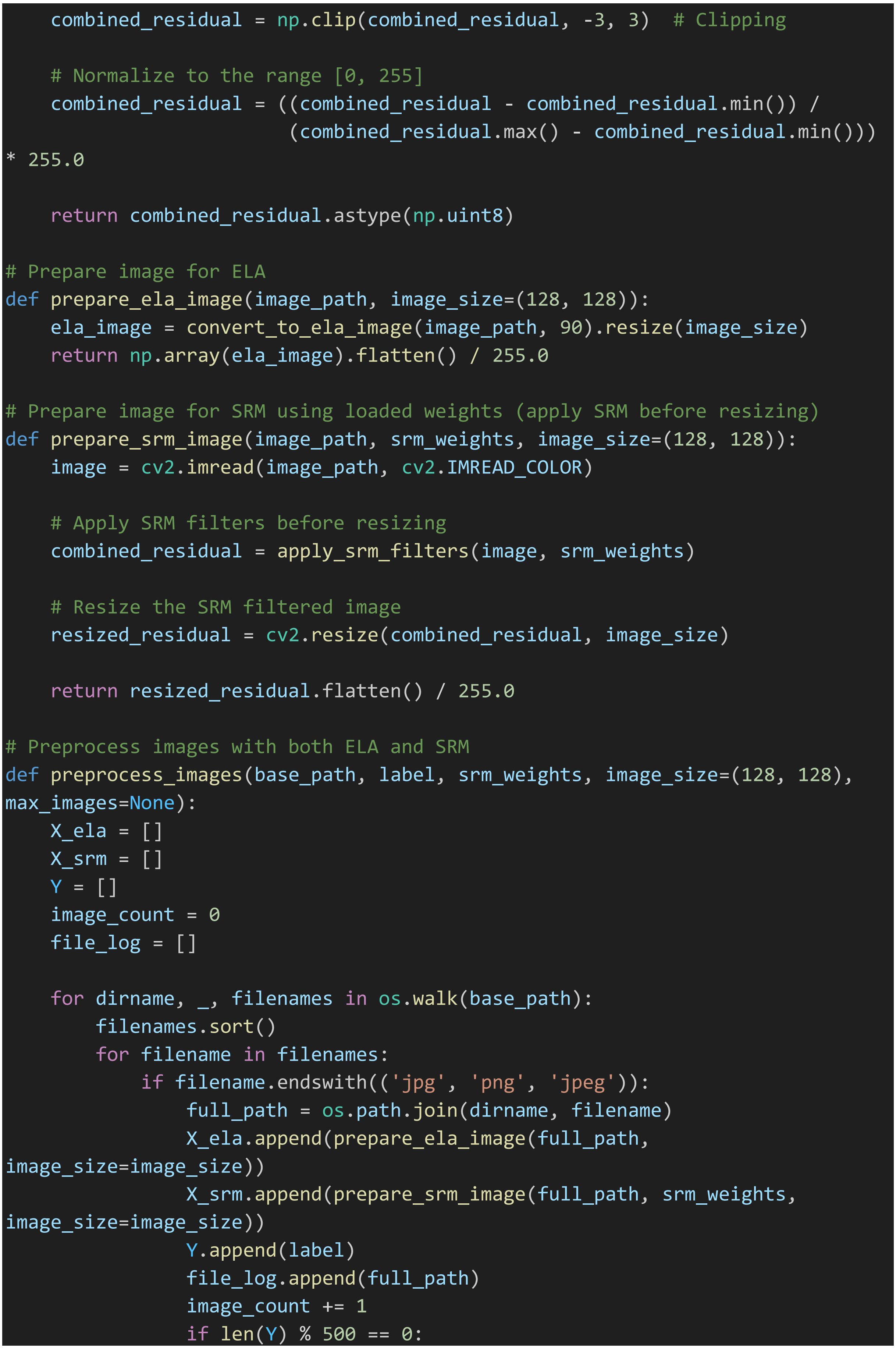
Zanardelli, M., Guerrini, F., Leonardi, R., & Adami, N. (2022). Image forgery detection: a survey of recent deep-learning approaches. *Multimedia Tools and Applications*, *82*, 17521–17566. https://doi.org/10.1007/s11042-022-13797-w

Appendices

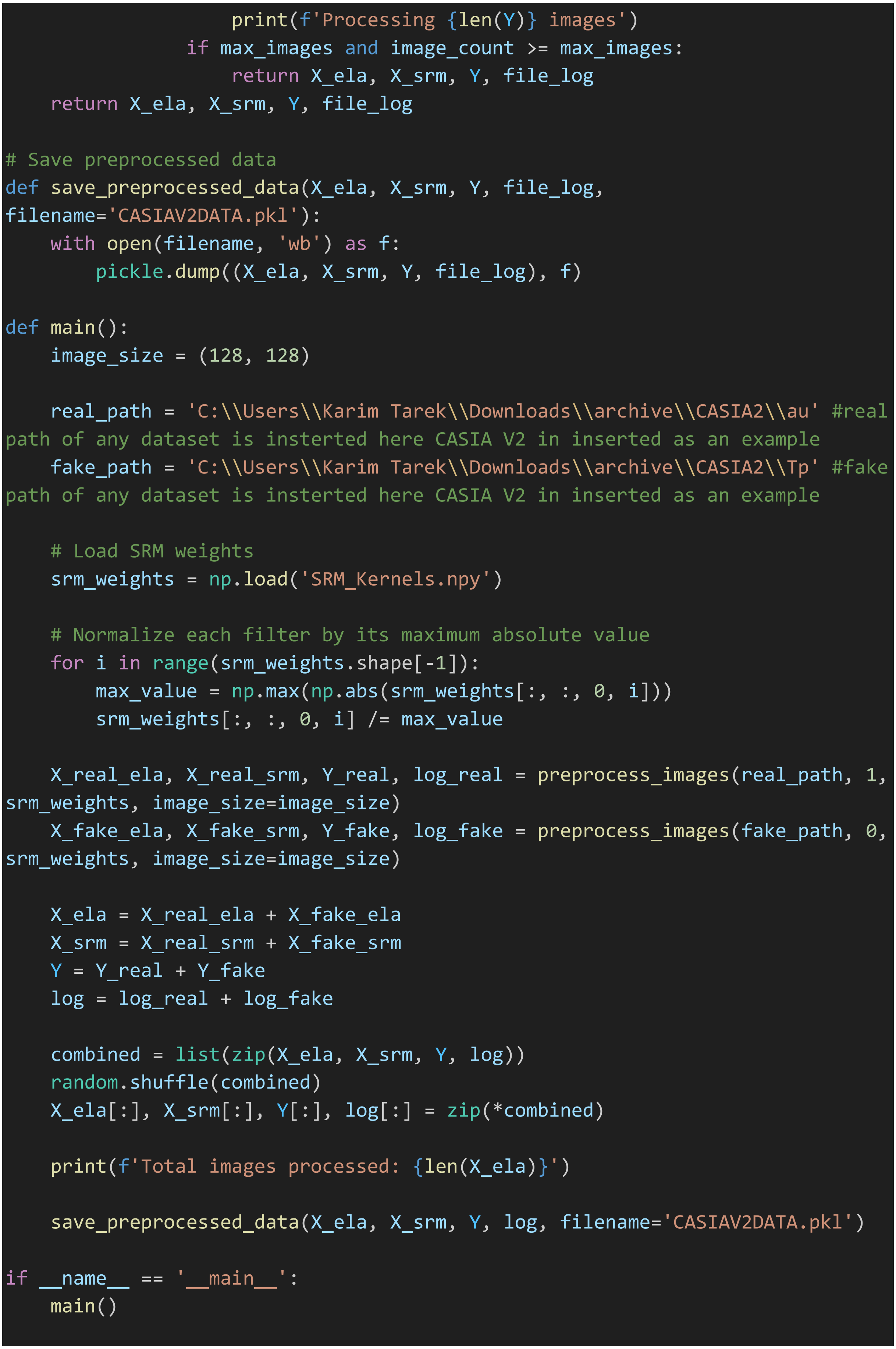
Appendix A: Code

Preprocessing (CASIA V2 example)

```
import cv2
import numpy as np
from PIL import Image, ImageChops, ImageEnhance
import random
import pickle
import os

# Function to convert an image to an ELA image
def convert_to_ela_image(path, quality=90):
    temp_filename = 'temp_file_name.jpg'

    image = Image.open(path).convert('RGB')
    image.save(temp_filename, 'JPEG', quality=quality)
    temp_image = Image.open(temp_filename)

    ela_image = ImageChops.difference(image, temp_image)

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

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

    return ela_image

# Function to apply SRM filters using loaded weights
def apply_srm_filters(image, srm_weights):
    steps = srm_weights.shape[-1]  # Number of filters
    filtered_channels = []

    for step in range(steps):
        filtered_channels_step = []
        for channel in range(3):  # Apply each filter to each color channel
            filter_kernel = srm_weights[:, :, 0, step]
            filtered_image = cv2.filter2D(image[:, :, channel], -1,
filter_kernel)
            filtered_channels_step.append(filtered_image)

        combined_residual = np.stack(filtered_channels_step, axis=-1)
        filtered_channels.append(combined_residual)

    combined_residual = np.mean(filtered_channels, axis=0)  # Average over all
filters
```



```
    combined_residual = np.clip(combined_residual, -3, 3)  # Clipping

    # Normalize to the range [0, 255]
    combined_residual = ((combined_residual - combined_residual.min()) /
                         (combined_residual.max() - combined_residual.min()))
* 255.0

    return combined_residual.astype(np.uint8)

# Prepare image for ELA
def prepare_ela_image(image_path, image_size=(128, 128)):
    ela_image = convert_to_ela_image(image_path, 90).resize(image_size)
    return np.array(ela_image).flatten() / 255.0

# Prepare image for SRM using loaded weights (apply SRM before resizing)
def prepare_srm_image(image_path, srm_weights, image_size=(128, 128)):
    image = cv2.imread(image_path, cv2.IMREAD_COLOR)

    # Apply SRM filters before resizing
    combined_residual = apply_srm_filters(image, srm_weights)

    # Resize the SRM filtered image
    resized_residual = cv2.resize(combined_residual, image_size)

    return resized_residual.flatten() / 255.0

# Preprocess images with both ELA and SRM
def preprocess_images(base_path, label, srm_weights, image_size=(128, 128),
max_images=None):
    X_ela = []
    X_srm = []
    Y = []
    image_count = 0
    file_log = []

    for dirname, _, filenames in os.walk(base_path):
        filenames.sort()
        for filename in filenames:
            if filename.endswith(('jpg', 'png', 'jpeg')):
                full_path = os.path.join(dirname, filename)
                X_ela.append(prepare_ela_image(full_path,
image_size=image_size))
                X_srm.append(prepare_srm_image(full_path, srm_weights,
image_size=image_size))
                Y.append(label)
                file_log.append(full_path)
                image_count += 1
                if len(Y) % 500 == 0:
```



```
                    print(f'Processing {len(Y)} images')
                if max_images and image_count >= max_images:
                    return X_ela, X_srm, Y, file_log
    return X_ela, X_srm, Y, file_log

# Save preprocessed data
def save_preprocessed_data(X_ela, X_srm, Y, file_log,
filename='CASIAV2DATA.pkl'):
    with open(filename, 'wb') as f:
        pickle.dump((X_ela, X_srm, Y, file_log), f)

def main():
    image_size = (128, 128)

    real_path = 'C:\\Users\\Karim Tarek\\Downloads\\archive\\CASIA2\\au' #real
path of any dataset is insterted here CASIA V2 in inserted as an example
    fake_path = 'C:\\Users\\Karim Tarek\\Downloads\\archive\\CASIA2\\Tp' #fake
path of any dataset is insterted here CASIA V2 in inserted as an example

    # Load SRM weights
    srm_weights = np.load('SRM_Kernels.npy')

    # Normalize each filter by its maximum absolute value
    for i in range(srm_weights.shape[-1]):
        max_value = np.max(np.abs(srm_weights[:, :, 0, i]))
        srm_weights[:, :, 0, i] /= max_value

    X_real_ela, X_real_srm, Y_real, log_real = preprocess_images(real_path, 1,
srm_weights, image_size=image_size)
    X_fake_ela, X_fake_srm, Y_fake, log_fake = preprocess_images(fake_path, 0,
srm_weights, image_size=image_size)

    X_ela = X_real_ela + X_fake_ela
    X_srm = X_real_srm + X_fake_srm
    Y = Y_real + Y_fake
    log = log_real + log_fake

    combined = list(zip(X_ela, X_srm, Y, log))
    random.shuffle(combined)
    X_ela[:], X_srm[:], Y[:], log[:] = zip(*combined)

    print(f'Total images processed: {len(X_ela)}')

    save_preprocessed_data(X_ela, X_srm, Y, log, filename='CASIAV2DATA.pkl')

if __name__ == '__main__':
    main()
```

Model Training and evaluation (CASIA V2 example)

```
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(2)
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from keras.utils import to_categorical
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras import layers, models
import tensorflow as tf
import pickle
# from keras.callbacks import ModelCheckpoint
from CoMoFodpreprocessing import save_preprocessed_data
import os
import itertools
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score
from keras.models import load_model
import time
from tqdm import tqdm
import miccpreprocessing as pp
from sklearn.model_selection import StratifiedShuffleSplit
import json
gpus = tf.config.experimental.list_physical_devices('GPU')
if gpus:
    try:
        # memory growth needs to be the same across GPUs
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        logical_gpus = tf.config.experimental.list_logical_devices('GPU')
        print(f"{len(gpus)} Physical GPUs, {len(logical_gpus)} Logical GPUs")
    except RuntimeError as e:
        # Memory growth must be set before GPUs have been initialized
        print(e)
        os.environ['CUDA_VISIBLE_DEVICES'] = ''
def load_preprocessed_data(filename):
    with open(filename, 'rb') as f:
        X_ela, X_srm, Y, log = pickle.load(f)
    return np.array(X_ela), np.array(X_srm), np.array(Y), log
X_ela, X_srm, Y, log = load_preprocessed_data('CASIAV2DATA.pkl') #CASIA V2
DATASET LOCADED YOU CAN LOADE THE DATASET YOU WANT TO USE

X_ela = X_ela.reshape(-1, 128, 128, 3)
X_srm = X_srm.reshape(-1, 128, 128, 3)

# One-hot encode the labels
Y = to_categorical(Y, 2)
```

```
# Stratified split to ensure same class ratio in train and validation sets
sss = StratifiedShuffleSplit(n_splits=1, test_size=0.1, random_state=5)

for train_index, val_index in sss.split(X_ela, np.argmax(Y, axis=1)):
    X_ela_train, X_ela_val = X_ela[train_index], X_ela[val_index]
    X_srm_train, X_srm_val = X_srm[train_index], X_srm[val_index]
    Y_train, Y_val = Y[train_index], Y[val_index]

# Print the number of samples
print(f'Training samples: {len(X_ela_train)}, Validation samples:
{len(X_ela_val)}')

# Checking the distribution of classes in the training and validation sets
print('Class distribution in training set:', np.bincount(np.argmax(Y_train,
axis=1)))
print('Class distribution in validation set:', np.bincount(np.argmax(Y_val,
axis=1)))
def build_model():
    # ELA Branch
    input_ela = layers.Input(shape=(128,128,3), name='ELA_Input')
    x_ela = layers.Conv2D(32, (3, 3), activation='relu',
padding='same')(input_ela)
    x_ela = layers.Conv2D(32, (3, 3), activation='relu',
padding='same')(x_ela)
    x_ela = layers.MaxPooling2D((2, 2))(x_ela)

    # SRM Branch
    input_srm = layers.Input(shape=(128,128,3), name='SRM_Input')
    x_srm = layers.Conv2D(32, (3, 3), activation='relu',
padding='same')(input_srm)
    x_srm = layers.Conv2D(32, (3, 3), activation='relu',
padding='same')(x_srm)
    x_srm = layers.MaxPooling2D((2, 2))(x_srm)

    # Concatenate the outputs of both branches
    combined = layers.Concatenate()([x_ela, x_srm])
    combined = layers.Dropout(0.25)(combined)
    combined = layers.Flatten()(combined)

    # Fully connected layer
    fc = layers.Dense(256, activation='relu')(combined)
    fc = layers.Dropout(0.5)(fc)

    output = layers.Dense(2, activation='softmax')(fc)

    # Create the model
    model = models.Model(inputs=[input_ela, input_srm], outputs=output)
    return model
```

```
model = build_model()
model.summary()

epochs = 200
batch_size = 16

optimizer = optimizer = tf.keras.optimizers.Adam(
    learning_rate=0.001,
    beta_1=0.9,
    beta_2=0.999,
    epsilon=1e-08,
    decay=0.0001
)

model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])

early_stopping = EarlyStopping(monitor = 'val_accuracy',
                              min_delta = 0,
                              patience = 7,
                              verbose = 1,
                              mode = 'auto')

no_au = 7354
no_tp = 2064
total_images = no_au + no_tp

weight_for_au = (1 / no_au) * (total_images / 2.0)
weight_for_tp= (1 / no_tp) * (total_images / 2.0)
class_weights = {0: weight_for_tp, 1: weight_for_au}
checkpoint_filepath = 'finalmodelcasiatest5.h5'
checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    monitor='val_accuracy',
    save_best_only=True,            # Save only the best weights
    mode='max',
    verbose=1
)
hist = model.fit([X_ela_train, X_srm_train],
                    Y_train,
                    batch_size=batch_size,
                    class_weight=class_weights,
                    epochs=epochs,
                    validation_data=([X_ela_val, X_srm_val], Y_val),
                    callbacks=[early_stopping, checkpoint_callback])
    #early stopping implemented to prevent overfitting
model = load_model('finalmodelcasiatest5.h5')
```

```
history=hist.history
history_dict = hist.history
with open('finalmodelcasiatest5.pkl', 'wb') as file_pi:
    pickle.dump(history_dict, file_pi)
predictions = model.predict([X_ela_val, X_srm_val])
y_pred = np.argmax(predictions, axis=1)
y_true = np.argmax(Y_val, axis=1)
def generate_metrics(y_true, y_pred):
    print("Accuracy = " , accuracy_score(y_true, y_pred))
    print("Precision = " ,precision_score(y_true, y_pred))
    print("Recall = " ,recall_score(y_true, y_pred))
    print("F1 Score = " ,f1_score(y_true, y_pred))
    pass# Make predictions

generate_metrics(y_true, y_pred)

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    cm_flipped = cm[::-1, ::-1]

    plt.imshow(cm_flipped, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes[::-1], rotation=45)
    plt.yticks(tick_marks, classes[::-1])

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm_flipped.shape[0]),
range(cm_flipped.shape[1])):
        plt.text(j, i, cm_flipped[i, j],
                 horizontalalignment="center",
                 color="white" if cm_flipped[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('Predicted label')
    plt.xlabel('Actual label')

Y_pred = model.predict([X_ela_val, X_srm_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)
class_names = ['fake','real']
```

```
plot_confusion_matrix(confusion_mtx, classes=class_names, title='Confusion Matrix, Image Forgery Detection')

fig, ax = plt.subplots(2, 1)
#code for visulating training and validation loss and accuracy history
ax[0].plot(hist.history['loss'], color='b', label="Training loss")
ax[0].plot(hist.history['val_loss'], color='r', label="Validation loss")
ax[0].set_title('Training and Validation Loss')
ax[0].set_xlabel('Epoch')
ax[0].set_ylabel('Loss')
ax[0].legend(loc='best', shadow=True)

ax[1].plot(hist.history['accuracy'], color='b', label="Training accuracy")
ax[1].plot(hist.history['val_accuracy'], color='r', label="Validation accuracy")
ax[1].set_title('Training and Validation Accuracy')
ax[1].set_xlabel('Epoch')
ax[1].set_ylabel('Accuracy')
ax[1].legend(loc='best', shadow=True)
plt.tight_layout()
plt.show()
```