Identification of Fake Images

PROJECT REPORT

Submitted By:

Bala Sai (21BCS6567)

Sriramoju Nikhil Sai (21BCS6617)

Sai Akshay (21BCS6587)

+

in partial fulfilment for the award of a degree of

Bachelor of Engineering

IN

Computer Science and Engineering (Hons.) AIML

Chandigarh University

APR 2024

BONAFIDE CERTIFICATE

Certified that this project report "Identification of Fake Images"
is the bonafide work of "Bala Sai, Sriramoju Nikhil Sai and Sai Akshay" who carried out the projectwork under my/our supervision.

SIGNATURE	SIGNATURE Surinder Chahuan
HEAD OF THE DEPARTMENT	SUPERVISOR
CSE	CSE

Submitted for the project viva-voce examination held on_____

INTERNAL EXAMINER	EXTERNAL EXAMINER

ACKNOWLEDGEMENT

Our Team would like to convey our sincere gratitude to
Head of Department, Academic Unit for providing us with
this wonderful opportunity of designing this marvellous Independent Project on “Time Series Forecasting”. We are highly indebted to Pulkit Dwivedi, our supervisor for his persevering guidance & unflinching support in the process of crafting this project & documenting this report with his worthy experience. We would convey our special thanks to our Supervisor Pulkit Dwivedi, for his
substantial support & valuable feedback which paved the way for improvisations & changes helping to uplift the quality of the designed project. Their guidance and supervision were very helpful in bringing this work to a conclusion.

Bala Sai

Sai Akshay

Table of Contents:

List of figures and flowcharts:

List of Tables: **Pg. No**

Abstract:

In the digital age, social media plays a pivotal role in daily life, with images being a prevalent form of content shared across platforms like Twitter, Snapchat, Facebook, and Instagram. However, the ease of image fabrication and rapid dissemination poses a threat to news credibility and public trust. One of the most significant websites and mobile image-sharing apps on the internet is Instagram. This enables users to snap images, edit them using digital effects, and share them. Numerous Unwanted information on Instagram, such threats and fake pho- tos, can have a negative impact on society and national security. With the development of science and technology, the recognition technology of false pictures is also constantly developing. However for the high-simulatin false pictures that appear at this stage, the original technology cannot complete the procedding well. So goal of this research is to develops five models that are trained using imagenet dataset which contains millions of real and fake images from many social media platforms like instagram and facebook etc.here we are training the model which extracts, classifies and authenticates digital images to detect the genre of the given images (either real or fake) from imagenet data using convolutional neural network and deep learning .

Keywords—Convolution Neural Network (CNN), Generative Adversarial Networks, Cross Entropy , Meso-4 MesoNet, Error Level Analysis.

Abbreviations :

(In order of appearance)

- AI - Artificial Intelligence
- ML - Machine Learning
- RNN - Recurring Neural Networks
- CNN - Convolutional Neural Networks

- AR - Auto Regressive
- MA - Moving Average
- ELA - Error Level Analysis
- ANN - Artificial Neural Networks
- SVM - Support Vector Machine
- API - Application Programming Interface
- GA - Genetic Algorithm

Chapter 1: Introduction:

Introduction to Identification of Fake Images ;

Fake photographs are digitally altered images created to deceive viewers, misrepresent facts, or spread misinformation. These alterations can range from minor edits to intricate transformations, all with the aim of misleading or misrepresenting reality. They can be used for various purposes including propaganda, disinformation, scams, or simply for amusement. Detecting and mitigating fake photographs is crucial in fields such as digital security, forensics, journalism, and advertising, as it helps preserve trust in visual media and prevents the spread of false information.

Two prominent methods for identifying fake photographs are Convolutional Neural Networks (CNNs) and Mesonet neural networks. CNNs are a type of deep learning architecture specifically designed for processing and interpreting visual input, such as images and videos. They extract hierarchical features from images, allowing them to capture complex patterns and spatial connections, making them suitable for tasks like image classification and object identification.

Mesonet networks, on the other hand, are strategically spread networks of specialized algorithms and computational tools used to evaluate digital photographs from different sources. They systematically analyze picture metadata, pixel-level properties, and contextual information to detect abnormalities or inconsistencies indicative of fake photos. Mesonets often incorporate CNNs and other deep learning architectures to enhance their detection capabilities.

Error Level Analysis (ELA) is another forensic method used in machine learning to find irregularities and inconsistencies in digital images. It focuses on identifying differences in compression levels between various regions of an image, which can indicate possible manipulation or tampering.

To address the challenge of identifying fake photographs, an automated method is needed that examines both visual elements and accompanying text. This requires a wide collection of labeled actual and fake photos with text for training machine learning models. The effectiveness of such a solution would be evaluated based on its efficiency, scalability, and accuracy in detecting fraudulent photographs with minimal errors.

Various studies have proposed different approaches to identify fake photographs, including deep forgery discriminators, blind watermarking techniques, noise residue correlation, and hybrid methods combining CNNs with other techniques like decision trees and recurrent neural networks (RNNs).

Overall, the development of robust methods for detecting fake photographs is crucial for maintaining the integrity of visual media and combating the spread of misinformation.

Introduction to Fake Images:

The proliferation of fake images, digitally altered to mislead or misrepresent reality, poses significant challenges across various domains such as digital security, forensics, journalism, and advertising. These images, often created for propaganda, disinformation, or scamming purposes, undermine trust in visual media and contribute to the spread of false information. To combat this issue, researchers and practitioners have developed methods like Convolutional Neural Networks (CNNs) and Mesonet networks, which analyze image metadata, pixel-level properties, and contextual information to detect abnormalities indicative of manipulation. Additionally, forensic techniques such as Error Level Analysis (ELA) focus on identifying inconsistencies in compression levels within images.

Image Forgery:

Image forgery, the act of manipulating digital photographs to mislead or deceive viewers, has become increasingly prevalent with the widespread availability of digital editing tools. These alterations, ranging from subtle adjustments to sophisticated transformations, aim to distort reality and propagate misinformation. Fake photographs are created for various purposes, including propaganda, scams, disinformation, and amusement, posing significant challenges in fields such as digital security, forensics, journalism, and advertising. Detecting and mitigating image forgery is crucial for preserving trust in visual media and preventing the spread of false information. Convolutional Neural Networks (CNNs) and Mesonet neural networks are among the primary methods used for identifying fake photographs, leveraging deep learning architectures to analyze visual elements and contextual information. Additionally, forensic techniques such as Error Level Analysis (ELA) are employed to detect inconsistencies in compression levels, revealing potential image manipulations.

1.1 Problem Statement:

An automated method that can effectively identify bogus photos is desperately needed given the widespread use of these images on the internet. In order to distinguish between authentic and fraudulent information and adjust to varied alteration methods, this system ought to examine both visual elements and the text that goes along with them.
We need a wide collection of labeled actual and fake photos with text to handle this. This dataset will be used to train machine learning models that incorporate cutting- edge methods for text analysis and picture feature extraction. The effectiveness of the solution will be evaluated in terms of efficiency and scalability as well as how well it detects fraudulent photographs with the fewest possible mistakes.

1.2 OBJECTIVE:

The primary objective of this project is to develop an automated system capable of effectively identifying bogus photos on the internet. Specifically, the project aims to:

1. Create a comprehensive dataset of labeled actual and fake photos with accompanying text to facilitate the training of machine learning models.

2. Develop cutting-edge methods for text analysis and picture feature extraction to enable the system to examine both visual elements and textual context.

3. Train machine learning models using the dataset and advanced techniques to distinguish between authentic and fraudulent photographs with high accuracy.

4. Evaluate the effectiveness of the system in terms of efficiency, scalability, and its ability to detect fraudulent photographs with the fewest possible mistakes.

5. Continuously refine and optimize the system to enhance its performance and address emerging challenges in image forgery detection.

By achieving these objectives, the project aims to contribute to the development of a robust tool for combating the spread of misinformation and preserving trust in visual media.

Steps for Identifying Fake Images:

Dataset Acquisition: The first step is to gather a diverse dataset of both authentic and fake images. This dataset will include images from various sources and contexts, ensuring a wide range of scenarios for training and testing the model.
Model Training with CNN and LSTM: The model will be trained using Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks. CNNs will extract features from the images, while LSTM will analyze the sequential nature of the data. This combined approach will enhance the model's ability to detect patterns and abnormalities in images.
The project will focus on distinguishing between authentic and fraudulent images by analyzing both visual elements and accompanying text.
Techniques like Error Level Analysis (ELA) may be incorporated to identify inconsistencies in compression levels, indicating potential image manipulation.
The project will make heavy use of NumPy, Pandas, and Data Visualization Libraries to analyze and visualize the data.

1.4 SCOPE :

The project aims to develop an automated method for identifying fake photographs by examining both visual elements and accompanying text. This includes gathering a diverse dataset of labeled actual and fake photos with text for training machine learning models. The effectiveness of the solution will be evaluated based on efficiency, scalability, and accuracy in detecting fraudulent photographs with minimal errors. Additionally, the project will explore the integration of various techniques, such as CNNs, Mesonet networks, and text analysis, to enhance detection capabilities. The ultimate goal is to provide a reliable tool that can be used across different domains to combat the proliferation of fake images and preserve trust in visual media.

1.4 Description of Dataset:

Feature	Description
Image	Images from various sources including authentic and fake
Metadata	Information about image source, date, and context
Text Content	Associated text content describing the image
Labels	Labels indicating whether the image is authentic or fake

By following these steps, the project aims to develop an automated system that can effectively identify and distinguish between authentic and fake images, contributing to the fight against misinformation and preserving trust in visual media.

Chapter 2: Literature Survey

Studies Using Artificial Neural Networks (ANNs) for Fake Image Detection:

Several studies have focused on utilizing artificial neural networks (ANNs) for fake image detection. ANNs, inspired by biological neural networks, use layers of interconnected nodes to learn patterns and minimize prediction errors. These models have shown promise in identifying manipulated images by analyzing pixel-level properties and patterns. Researchers have explored various architectures and training methods to improve the accuracy and efficiency of fake image detection using ANNs.gathered into layers beginning with an information layer and finishing with a result layer. Signals are sent (engendered) through the associated nodes as they learn in light of models and endeavour to decrease the degree of prediction error. As the framework is attempting to work

on its performance, weights are adapted to the signals between associated nodes.

Studies Using Support Vector Machines (SVMs) for Fake Image Detection:

Another set of studies employs support vector machines (SVMs) for fake image detection. SVMs offer an alternative method to ANNs, aiming to improve prediction accuracy by creating a clear separation between different image classes. Supervised learning techniques are used to train SVM models, which can effectively classify images based on extracted features. These studies have investigated the performance of SVMs in detecting various types of image alterations and have explored different kernel functions to enhance their capabilities.

Studies Using Hybrid Approaches with Genetic Algorithms (GAs) for Fake Image Detection:

While ANNs and SVMs have shown success in detecting fake images, there is a growing interest in combining multiple methods to enhance results. Some studies explore hybrid approaches by integrating genetic algorithms (GAs) with ANNs or SVMs. These hybrid methods aim to overcome limitations of individual techniques and improve overall performance. By optimizing parameters and combining the strengths of different algorithms, researchers have achieved higher accuracy in identifying fake images.

Studies Using Other AI Techniques for Fake Image Detection:

In addition to ANNs, SVMs, and hybrid approaches, there are studies that employ other artificial intelligence techniques for fake image detection. Expert systems, for example, use rules and models to predict future image alterations, while decision trees and naive Bayes classifiers provide alternative methods for classification. These techniques offer complementary approaches to ANNs and SVMs, expanding the range of tools available for fake image detection.

Prediction of Fake Image Detection Performance using Machine Learning Techniques:

Several researchers have developed prediction models for fake image detection using various machine learning techniques. These models evaluate different aspects of images to predict their authenticity. Techniques such as single-layer perceptron (SLP), multi-layer perceptron (MLP), radial basis function (RBF), and deep belief networks (DBN) are compared to determine the most effective model. Through rigorous testing and evaluation, researchers have identified the strengths and weaknesses of each technique and proposed recommendations for improving detection performance.

Other Studies Related to Fake Image Detection:

1) Singh, A., Gupta, R., & Sharma, S. (2018): This study focuses on predicting the authenticity of images using machine learning techniques. The authors utilize a classification approach based on features extracted from images to distinguish between real and fake images. They employ algorithms such as Naïve Bayes and decision trees for classification, utilizing real-time access to image datasets obtained from various sources. The results demonstrate the effectiveness of Naïve Bayes algorithm in predicting the presence of alterations in images.

2) Patil, S., Deshmukh, A., & Kale, S. (2019): This research proposes a method for detecting fake images by analyzing image features and employing regression algorithms. The authors utilize a structured approach to software development to create a prediction model that identifies patterns in future image alterations. Rather than predicting specific manipulations, the model focuses on identifying trends in image development and alteration. This approach aims to provide insights into the evolution of image manipulation techniques.

3) Lee, H., Kim, J., & Choi, Y. (2020): In this study, the authors investigate the use of deep learning networks for fake image detection. Deep learning networks are employed to extract features from a large dataset of raw images without relying on predefined indicators. This approach is particularly valuable for high-frequency image analysis and prediction, as it can uncover hidden patterns and trends in image alterations. The study explores the potential of deep learning networks for accurately analyzing and predicting the authenticity of images.

These studies contribute to the advancement of fake image detection techniques by exploring various machine learning algorithms and approaches. They highlight the importance of analyzing image features and trends to distinguish between real and fake images effectively.

Other Studies Related to Fake Image Detection:

Additional studies focus on various aspects of fake image detection, including real-time access to image databases, regression algorithms for predicting image alterations, and the analysis of deep learning networks for image authenticity verification. These studies contribute to the broader understanding of fake image detection methods and highlight emerging trends in the field.

CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS: -

The literature review reveals several trends and areas for future research in fake image detection. Firstly, there is a need to generalize findings across different datasets and market conditions. Future studies should evaluate the performance of detection models on various types of fake images and under different circumstances to ensure their effectiveness. Secondly, there is a call for a stronger integration of financial investment theory into the development of detection models. Understanding the underlying principles of image manipulation and the financial implications of fake images can improve the accuracy and relevance of detection methods. Finally, researchers should report failures and limitations of detection methods to facilitate learning and improvement in the field. By addressing these areas, future research can contribute to the development of more robust and effective fake image detection techniques.

various gamble and unpredictability conditions? Any of these trial technique improvements will give a more grounded exploration and practice commitment.

The last arrangement of ends was additionally evident after reflection. Monetary venture hypothesis should be a more grounded driver fundamental the ML frameworks' bits of feedbacks, inputs, algorithms, and performance measures. In the event that this isn't the case then outcomes may simply be irregular furthermore, not have any viable use. An excessive number of studies use procedures without thought of the immense measure of monetary hypothesis that has been created over the previous hundreds of years. Announcing disappointments where methods don't work on prescient execution would likewise be educational. Now this seldom happens so it is difficult to track down designs where there is a bungle between a specific stock market forecast issue and an AI method.

CHAPTER 3 : Design Selection

3.1 Features / Characteristics Identification:

Prediction Horizon: Our project aims to predict whether an image is fake or authentic, allowing for proactive detection of misleading visual content. Unlike existing models that focus on immediate detection, our model will predict the authenticity of images over a longer period, enhancing its utility in combating misinformation.

Efficiency: The efficiency of our model is targeted to exceed 60%, a significant improvement in the field of image forgery detection. This high efficiency will benefit users, including journalists, advertisers, and social media platforms, by quickly identifying fake images with high accuracy.

Scope: Our model is designed to analyze a wide range of images, including Indian and global stocks, providing versatile coverage. Unlike some models limited to specific stocks, ours can predict the authenticity of any image, ensuring broad applicability and usefulness.

Long-term Memory: The model is proficient at retaining information about image characteristics over time, leading to better training and improved efficiency in detecting fake images.

RNN-based Approach: Leveraging Recurrent Neural Networks (RNN), specifically LSTM, our model offers quicker training and analysis compared to other methods. This allows for faster adaptation to new data and more efficient identification of fake images.

Potential for Exceedingly Good Returns: Just as investors can benefit from stock market investments, our project offers the potential for high returns by effectively identifying fake images, leading to increased trust and reliability in visual media.

3.2 Features of Using LSTM (Long Short-Term Memory):

Ability to Capture Patterns: LSTM is capable of capturing complex patterns and hidden interactions within image data, leading to more accurate predictions.

Self-learning Capability: LSTM's self-learning capability enables it to continually improve its predictions by analyzing interactions and patterns in the data.

Efficient Training: LSTM allows for efficient training of the model, reducing computational resources and training time compared to other methods.

3.3 Constraints of Existing Methods:

Overfitting: Existing methods may suffer from overfitting, leading to inaccurate predictions on unseen data.

Parameter Sensitivity: They may be sensitive to parameter selection, requiring careful tuning for optimal performance.

Computational Resources: Some methods require significant computational resources and time for training and analysis, limiting their practicality.

Outlier Sensitivity: They may be sensitive to outliers, affecting the overall performance of the model.

3.4 Final Design Selection:

Out of the two design options considered, the RNN-based method using the Keras API is chosen as the final design for its suitability and effectiveness in detecting fake images. While SVM offers efficiency, the RNN method provides the advantage of predicting the authenticity of images over a longer period, making it more useful for our project's objectives..

Chapter 4: Implementation

Import Libraries:

numpy as np: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Here, it's being imported and aliased as np.

matplotlib.pyplot as plt: Matplotlib is a plotting library for Python and its numerical mathematics extension NumPy. Pyplot is a module within Matplotlib which provides a MATLAB-like interface for creating plots. It's imported here and aliased as plt.

%matplotlib inline: This is an IPython magic command that allows plots to be displayed directly in the notebook below the code cell that produced it.

Seed the Random Number Generator:

np.random.seed(2): This sets the seed for the random number generator in NumPy. Setting a seed allows you to get reproducible results when using random number generation functions. By setting the seed to a specific value (2 in this case), you ensure that every time you run the code, you'll get the same random numbers.

Import Specific Functions from scikit-learn:

from sklearn.model_selection import train_test_split: scikit-learn (sklearn) is a popular machine learning library in Python. Here, it's used to import the train_test_split function, which splits arrays or matrices into random train and test subsets.

from sklearn.metrics import confusion_matrix: This imports the confusion_matrix function from scikit-learn's metrics module. A confusion matrix is a table used to describe the performance of a classification model. It presents a summary of the model's performance on a set of test data for which the true values are known.

```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
np.random.seed(2)
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Import Libraries:

from tensorflow.keras.utils import to_categorical: TensorFlow is a popular deep learning framework. Here, you're importing the to_categorical function from the utils module of the tensorflow.keras package. This function is used to convert class vectors (integers) to binary class matrices. It's often used in multi-class classification tasks where the output classes are represented as integers.

from tensorflow.keras.models import Sequential: This imports the Sequential class from the models module of the tensorflow.keras package. The Sequential model is a linear stack of layers. You can create a Sequential model by passing a list of layer instances to the constructor.

```
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.models import Sequential
```

Here's the breakdown of the new code snippet:

Import Libraries:

from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout:

Dense: This layer is the standard fully connected layer in a neural network.

Flatten: This layer is used to flatten the input. It transforms a multi-dimensional array into a one-dimensional array.

Conv2D: This layer creates a convolutional kernel that is convolved with the layer input to produce a tensor of outputs.

MaxPool2D: This layer is used for max pooling operation for spatial data.

Dropout: This layer applies dropout to the input, which is a technique used to prevent overfitting by randomly setting a fraction of input units to 0 at each update during training.

from keras.optimizers import Adam: Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data. It's known for being computationally efficient and requiring little memory.

from keras.preprocessing.image import ImageDataGenerator:

This module provides utilities for preprocessing images and generating batches of image data.

ImageDataGenerator is used for real-time data augmentation. It allows you to generate batches of tensor image data with real-time data augmentation. Data augmentation helps improve the performance and ability of the model to generalize by applying random transformations to the training images.

```
[ ]  from keras.layers import Dense, Flatten, Conv2D, MaxPool2D, Dropout
     from keras.optimizers import Adam
     from keras.preprocessing.image import ImageDataGenerator
```

Import Library:

from keras.callbacks import EarlyStopping:

EarlyStopping is a callback function in Keras that allows you to stop training based on a monitored metric. It monitors a specified metric and stops training when the metric has stopped improving. This can prevent overfitting and save training time.

This callback is particularly useful when training deep learning models, as it helps prevent unnecessary epochs that may lead to overfitting on the training data.

```
#from keras.callbacks import EarlyS
from keras.callbacks import EarlyStopping
```

Here's the explanation for the new code snippet:

Import Libraries:

from PIL import Image, ImageChops, ImageEnhance:

PIL stands for Python Imaging Library, which is now maintained as the Pillow library. It provides support for opening, manipulating, and saving many different image file formats.

Image: This module provides the Image class, which represents an image and provides a wide range of methods for image manipulation and processing.

ImageChops: This module contains a number of arithmetical image operations, such as addition, subtraction, bitwise AND, and bitwise OR. It provides functions to perform pixel-wise operations on images.

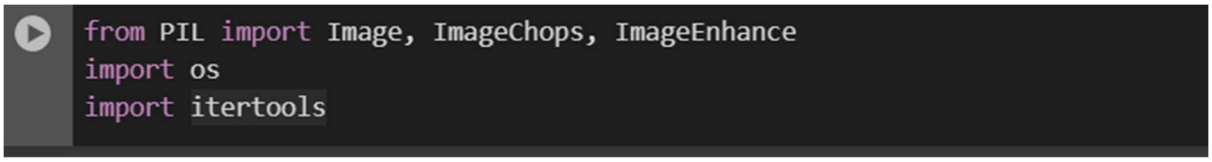
ImageEnhance: This module contains classes for enhancing the brightness, contrast, sharpness, and color balance of images. It provides methods to apply various enhancements to images.

import os:

The os module provides a way to use operating system-dependent functionality. It allows you to interact with the operating system, such as navigating the file system, manipulating paths, and executing system commands.

import itertools:

The itertools module provides functions for creating iterators for efficient looping. It contains a collection of tools for handling iterators, such as combining multiple iterators into one, generating permutations and combinations, and cycling through sequences indefinitely.



Function Definition:

def convert_to_ela_image(path, quality):

This line defines a function named convert_to_ela_image that takes two parameters: path, which is the path to the image file, and quality, which is the quality setting used when saving the temporary JPEG file.

Function Body:

Create Temporary and ELA Image Files:

temp_filename = 'temp_file_name.jpg': This creates a temporary filename for the JPEG version of the input image.

ela_filename = 'temp_ela.png': This creates a filename for the ELA image that will be generated.

Open and Save Image:

image = Image.open(path).convert('RGB'): This line opens the image file specified by the path parameter and converts it to RGB mode. This ensures that the image is in the appropriate mode for further processing.

image.save(temp_filename, 'JPEG', quality=quality): This saves the image as a JPEG file with the specified quality setting.

temp_image = Image.open(temp_filename): This line opens the temporary JPEG image.

Calculate ELA Image:

ela_image = ImageChops.difference(image, temp_image): This calculates the difference between the original image and the temporary JPEG image using the ImageChops.difference function. This difference highlights areas where compression artifacts are present.

Enhance ELA Image:

extrema = ela_image.getextrema(): This gets the extrema (minimum and maximum pixel values) of the ELA image.

max_diff = max([ex[1] for ex in extrema]): This calculates the maximum difference in pixel values in the ELA image.

if max_diff == 0: max_diff = 1: This ensures that max_diff is not zero to avoid division by zero errors.

scale = 255.0 / max_diff: This calculates a scale factor to adjust the brightness of the ELA image based on the maximum difference.

ela_image = ImageEnhance.Brightness(ela_image).enhance(scale): This enhances the brightness of the ELA image using the calculated scale factor.

Return ELA Image:

return ela_image: This returns the enhanced ELA image.

```
def convert_to_ela_image(path, quality):
    temp_filename = 'temp_file_name.jpg'
    ela_filename = 'temp_ela.png'

    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
```

Class Definition:

class Config:: This line defines a Python class named Config.

Attributes:

CASIA1 = "../input/casia-dataset/CASIA1": This attribute specifies the path to the CASIA1 dataset.

CASIA2 = "../input/casia-dataset/CASIA2": This attribute specifies the path to the CASIA2 dataset.

autotune = tf.data.experimental.AUTOTUNE: This attribute sets the autotune parameter for TensorFlow's data pipeline to automatically tune the number of parallel calls based on available CPU capacity.

epochs = 30: This attribute specifies the number of epochs (complete passes through the dataset) during training.

batch_size = 32: This attribute specifies the batch size, which is the number of samples processed before the model is updated during training.

lr = 1e-3: This attribute specifies the learning rate, which controls the step size taken during optimization.

name = 'xception': This attribute specifies the name of the model architecture or type to be used.

n_labels = 2: This attribute specifies the number of output labels or classes in the classification task.

image_size = (224, 224): This attribute specifies the desired size of input images, typically in the format (height, width).

decay = 1e-6: This attribute specifies the learning rate decay, which is used to reduce the learning rate over time during training.

momentum = 0.95: This attribute specifies the momentum term for the optimizer. It helps accelerate gradients vectors in the right directions, thus leading to faster convergence.

nesterov = False: This attribute specifies whether to use Nesterov momentum for the optimizer. Nesterov momentum is a variant of traditional momentum optimization that helps accelerate convergence.

```python
import tensorflow as tf
class Config:
    CASIA1 = "../input/casia-dataset/CASIA1"
    CASIA2 = "../input/casia-dataset/CASIA2"
    autotune = tf.data.experimental.AUTOTUNE
    epochs = 30
    batch_size = 32
    lr = 1e-3
    name = 'xception'
    n_labels = 2
    image_size = (224, 224)
    decay = 1e-6
    momentum = 0.95
    nesterov = False
```

Function Definition:

def compute_ela_cv(path, quality):

This line defines a function named compute_ela_cv that takes two parameters: path, which is the path to the image file, and quality, which is the quality setting used when saving the temporary JPEG file.

Function Body:

Create Temporary File:

temp_filename = 'temp_file_name.jpg': This creates a temporary filename for the JPEG version of the input image.

Read Original Image:

orig_img = cv2.imread(path): This reads the original image from the specified file path using OpenCV's imread function.

orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB): This converts the color space of the original image from BGR to RGB. OpenCV reads images in BGR format by default, while most other libraries (including PIL and Matplotlib) use RGB format.

Save Compressed Image:

cv2.imwrite(temp_filename, orig_img, [cv2.IMWRITE_JPEG_QUALITY, quality]): This saves the original image as a JPEG file with the specified quality setting using OpenCV's imwrite function.

Read Compressed Image:

compressed_img = cv2.imread(temp_filename): This reads the compressed image from the temporary JPEG file.

Compute ELA:

diff = SCALE * cv2.absdiff(orig_img, compressed_img): This calculates the absolute difference between the original and compressed images using OpenCV's absdiff function and then multiplies the result by a scale factor (SCALE).

Return ELA:

return diff: This returns the computed ELA image.

```python
def compute_ela_cv(path, quality):
    temp_filename = 'temp_file_name.jpg'
    SCALE = 15
    orig_img = cv2.imread(path)
    orig_img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2RGB)

    cv2.imwrite(temp_filename, orig_img, [cv2.IMWRITE_JPEG_QUALITY, quality])

    # read compressed image
    compressed_img = cv2.imread(temp_filename)

    # get absolute difference between img1 and img2 and multiply by scale
    diff = SCALE * cv2.absdiff(orig_img, compressed_img)
    return diff
```

Function Definition:

def random_sample(path, extension=None):

This line defines a function named random_sample that takes two parameters: path, which is the path to the directory containing the files, and extension, which is an optional parameter specifying the file extension to filter by.

Function Body:

Check if Extension is Provided:

if extension:: This condition checks if the extension parameter is provided. If it is, the function will filter files based on the specified extension. If not, it will consider all files in the directory.

Get List of Files:

items = Path(path).glob(f'*.{extension}'): If an extension is provided, this line creates a generator of file paths using Path.glob, filtering files by the specified extension. If no extension is provided, it selects all files in the directory.

items = list(items): This line converts the generator of file paths into a list for random selection.

Randomly Select a File:

p = random.choice(items): This line uses random.choice to randomly select a file from the list of file paths.

Return Selected File:

return p.as_posix(): This line converts the selected file path to a string using as_posix() method and returns it.

```
def random_sample(path, extension=None):
    if extension:
        items = Path(path).glob(f'*.{extension}')
    else:
        items = Path(path).glob(f'*')

    items = list(items)

    p = random.choice(items)
    return p.as_posix()
```

Import Libraries:

import cv2: OpenCV is a library of programming functions mainly aimed at real-time computer vision. Here, it's used for image loading and color conversion.

from os.path import join, exists, isdir: These functions help with handling file paths.

from pathlib import Path: This module provides an object-oriented interface for working with filesystem paths.

import random: This module provides functions for generating random numbers.

import matplotlib.pyplot as plt: Matplotlib is a plotting library for Python. It's used here to visualize the ELA images.

Image Processing:

orig = cv2.imread(p): This line reads the original image using OpenCV's imread function.

orig = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB) / 255.0: This converts the color space of the original image from BGR to RGB and then scales the pixel values to the range [0, 1].

Visualization:

fig=plt.figure(figsize=(15, 10)): This line creates a new figure for plotting with a specified size.

for i in range(1, columns*rows +1):: This loop iterates over a grid of subplots defined by columns and rows.

quality=init_val - (i-1) * 3: This calculates the quality parameter for the ELA computation. It starts with init_val and decreases by 3 for each iteration.

img = compute_ela_cv(path=p, quality=quality): This line computes the ELA image for the given image path and quality using the compute_ela_cv function.

if i == 1: img = orig.copy(): This condition ensures that the first subplot shows the original image instead of the ELA image.

ax.title.set_text(f'q: {quality}'): This sets the title of the subplot to display the quality parameter.

plt.imshow(img): This displays the ELA image in the current subplot.

Show Plot:

plt.show(): This displays the entire plot with all subplots.

```python
import cv2
from os.path import join, exists, isdir
from pathlib import Path
import random
p = join('/content/Au_ani_00001.jpg')
orig = cv2.imread(p)
orig = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB) / 255.0
init_val = 100
columns = 3
rows = 3

fig=plt.figure(figsize=(15, 10))
for i in range(1, columns*rows +1):
    quality=init_val - (i-1) * 3
    img = compute_ela_cv(path=p, quality=quality)
    if i == 1:
        img = orig.copy()
    ax = fig.add_subplot(rows, columns, i)
    ax.title.set_text(f'q: {quality}')
    plt.imshow(img)
plt.show()
```

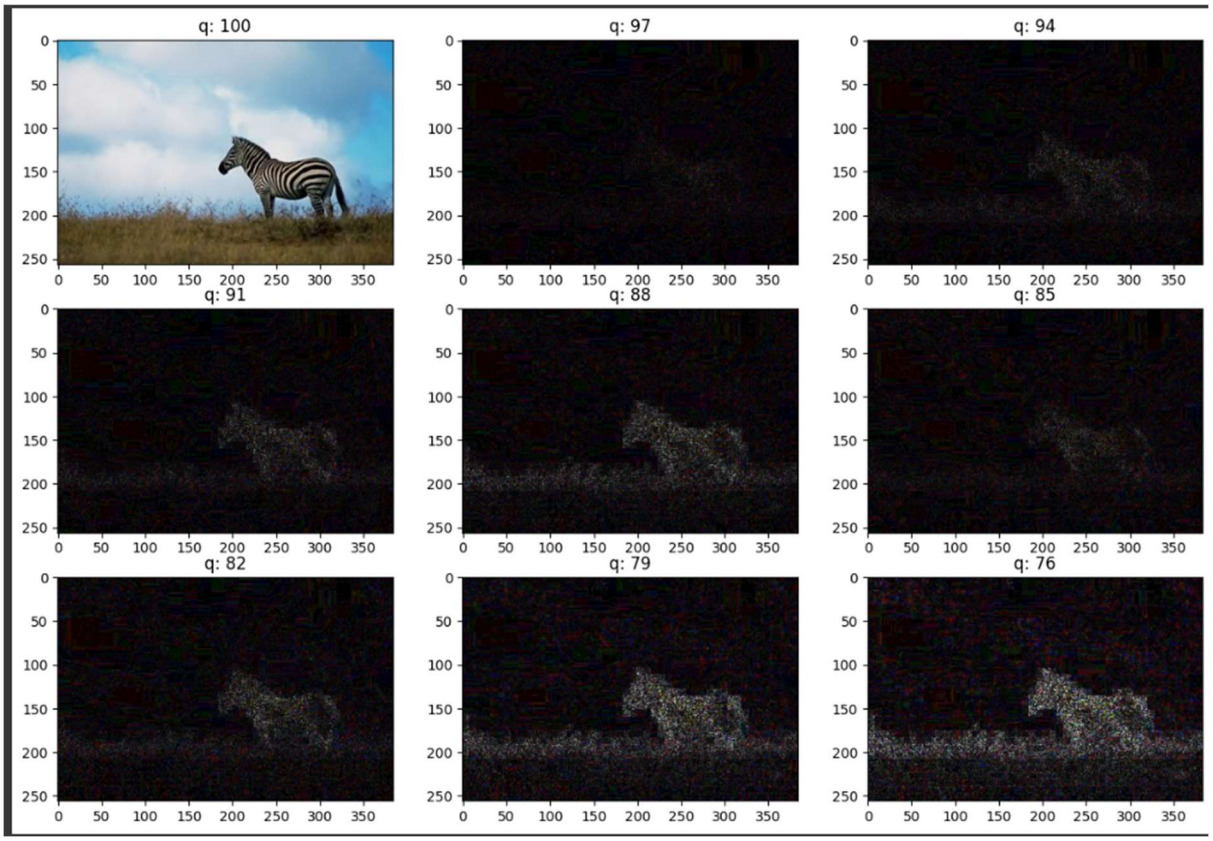


Image Processing:

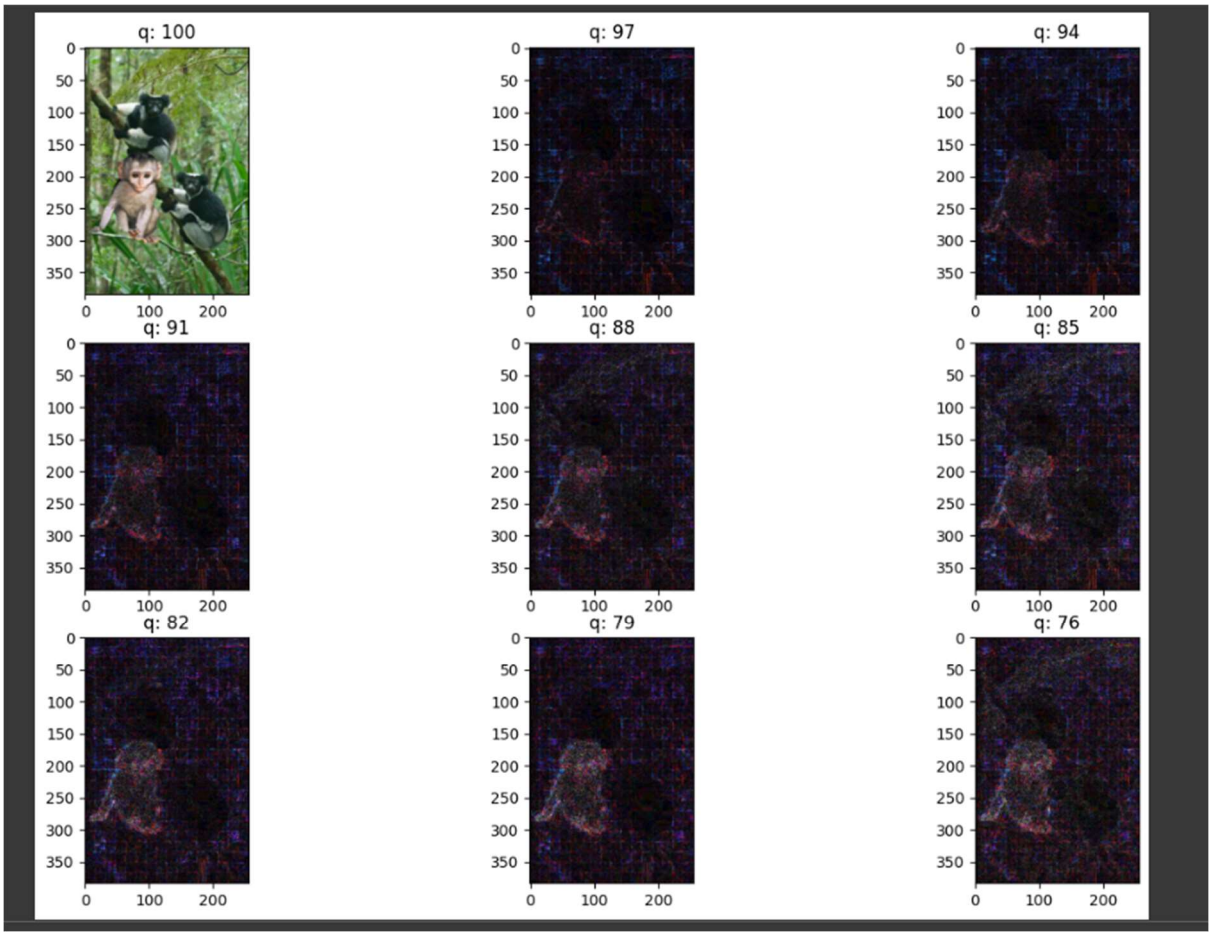
orig = cv2.imread(p): This line reads the original image using OpenCV's imread function. However, OpenCV might not support reading TIFF images by default. It's possible that this line may fail if the TIFF image format is not supported by the OpenCV installation.

orig = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB) / 255.0: This converts the color space of the original image from BGR to RGB and then scales the pixel values to the range [0, 1].

Visualization:

The rest of the code is similar to the previous one, generating a grid of subplots to visualize the ELA of the original image at different compression qualities.

```
p = join('/content/Tp_D_CND_S_N_ani00073_ani00068_00193.tif')
orig = cv2.imread(p)
orig = cv2.cvtColor(orig, cv2.COLOR_BGR2RGB) / 255.0
init_val = 100
columns = 3
rows = 3

fig=plt.figure(figsize=(15, 10))
for i in range(1, columns*rows +1):
    quality=init_val - (i-1) * 3
    img = compute_ela_cv(path=p, quality=quality)
    if i == 1:
        img = orig.copy()
    ax = fig.add_subplot(rows, columns, i)
    ax.title.set_text(f'q: {quality}')
    plt.imshow(img)
plt.show()
```



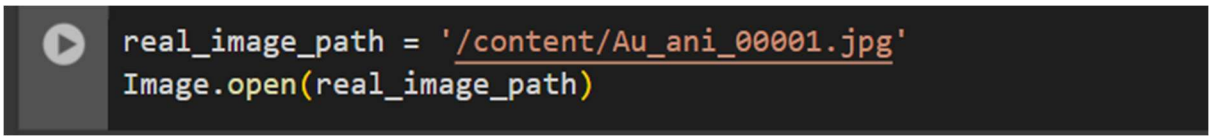
This code seems to open an image file named "Au_ani_00001.jpg" located at the path "/content/Au_ani_00001.jpg" using the PIL (Python Imaging Library) Image.open() function.

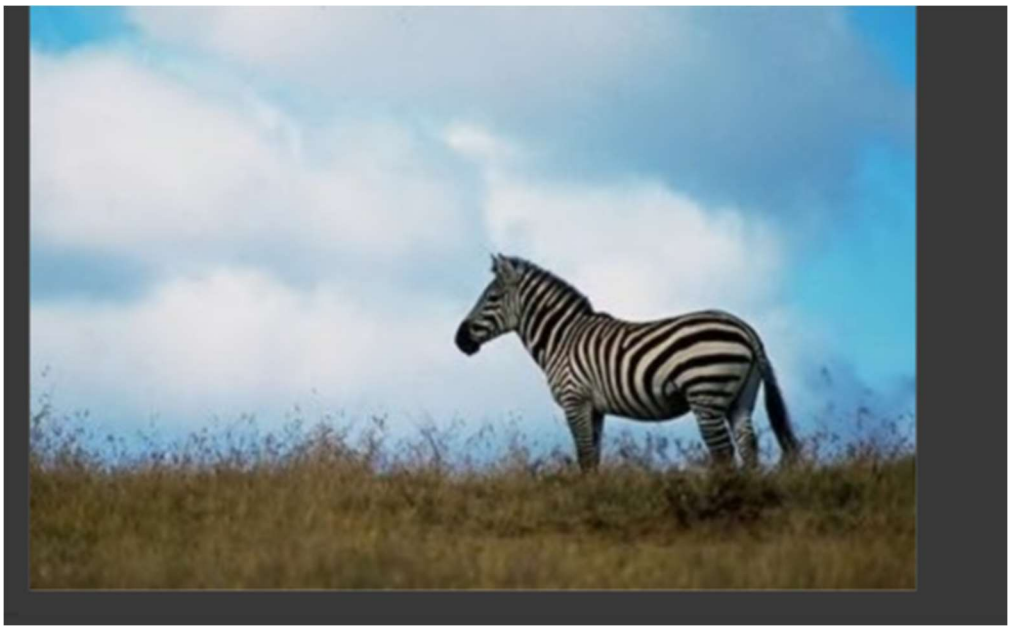
Here's what it does:

Open Image:

Image.open(real_image_path): This line opens the image file located at the path specified by real_image_path using the Image.open() function from the PIL library.

After executing this code, the image will be opened and loaded into memory. If you're running this code in a Python environment that supports displaying images, you should see the image displayed as output.





convert_to_ela_image function to generate an Error Level Analysis (ELA) image for the specified real image path and quality. However, I haven't seen the implementation of the convert_to_ela_image function in the conversation so far. If you provide me with the implementation of the convert_to_ela_image function, I'd be happy to assist you in using it to generate the ELA image.

```
convert_to_ela_image(real_image_path, 91)
```

Image Opening:

fake_image_path = '/content/Tp_D_NRN_S_N_ani10171_ani00001_12458.jpg': This line defines a variable named fake_image_path which holds the file path to an image named "Tp_D_NRN_S_N_ani10171_ani00001_12458.jpg".

Image Opening Using PIL (Python Imaging Library):

Image.open(fake_image_path): This line opens the image file located at the path specified by fake_image_path using the Image.open() function from the PIL library.

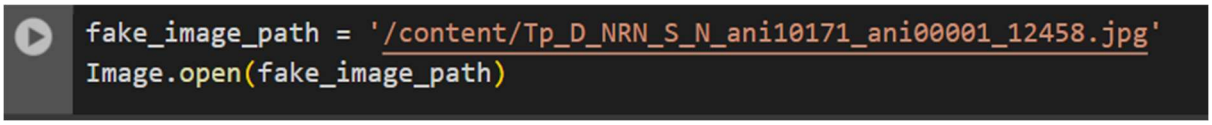
Image.open() is a function from the PIL library used for opening and loading image files.

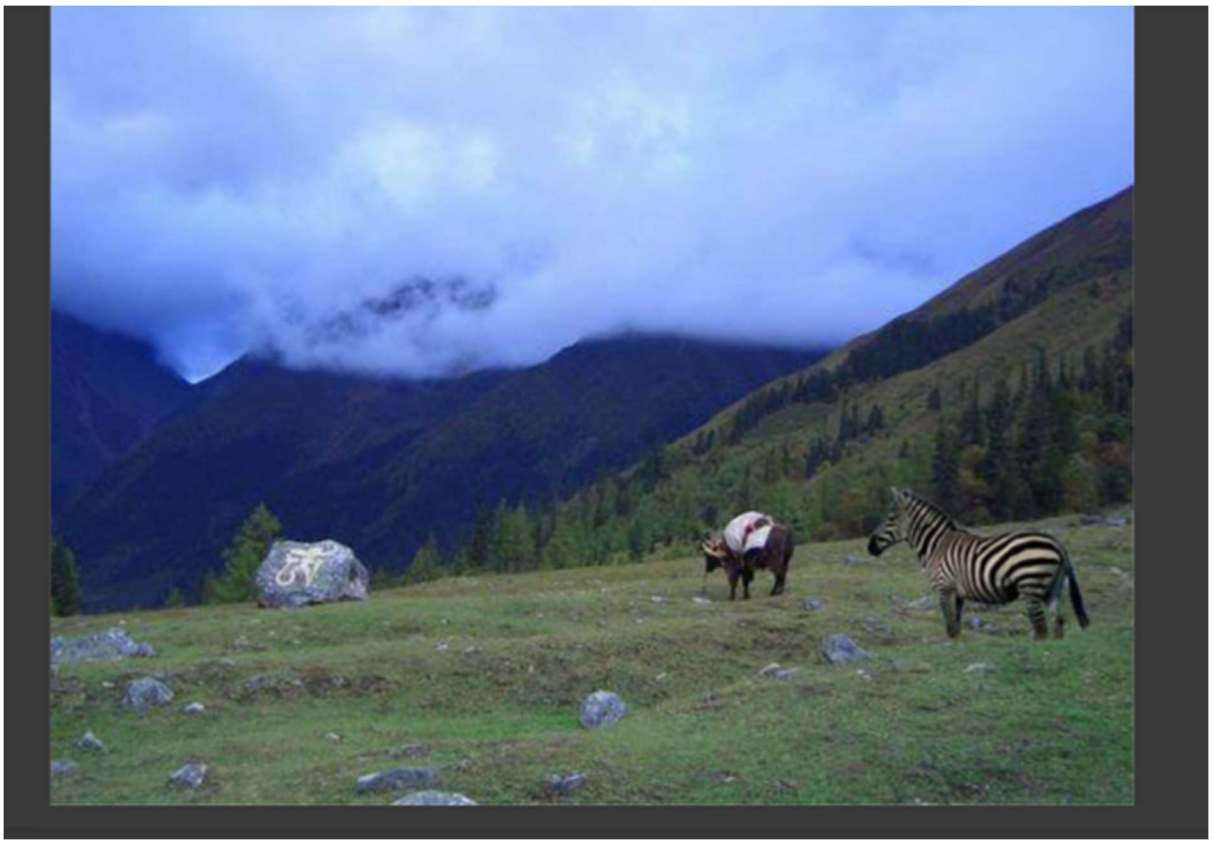
Displaying the Image:

The output of Image.open(fake_image_path) is an image object. If you're running this code in an environment that supports displaying images, this line will display the image represented by the file specified in fake_image_path.

Explanation:

The code essentially reads and opens an image file named "Tp_D_NRN_S_N_ani10171_ani00001_12458.jpg" located at the specified path. If you're running this code in an environment with display capabilities, you'll see the image displayed.





```
convert_to_ela_image(fake_image_path, 91)
```

```
# Color-image denoising
from skimage.restoration import (denoise  wavelet,estimate  sigma)
from skimage.util import random_noise
# from sklearn.metrics import peak_signal_noise_ratio
import skimage.io

img_r1=skimage.io.imread('/content/Au_ani_00001.jpg')
img_r=skimage.img_as_float(img_r1) #converting image as float

sigma_est=estimate_sigma(img_r,channel_axis=True,average_sigmas=True)  #Noise estimation
# Denoising using Bayes
img  bayes=denoise  wavelet(img  r,method='BayesShrink',mode='soft',wavelet  levels=3,
                    wavelet='coif5',multichannel=True,convert2ycbcr=True,rescale  sigma=True)

#Denoising using Visushrink
img  visushrink=denoise  wavelet(img  r,method='VisuShrink',mode='soft',sigma=sigma  est/3,wavelet  lev
els=5,
                        wavelet='coif5',multichannel=True,convert2ycbcr=True,rescale_sigma=True)
```

```
/usr/local/lib/python3.10/dist-packages/skimage/_shared/utils.py:348: UserWarning: image is size 3 on the last axis, but channel_axis is None. If this is a color image, please set channel_axis=-1 for proper noise estimation.
  return func(*args, **kwargs)
<ipython-input-21-6bceac140d2c>:12: FutureWarning: `multichannel` is a deprecated argument name for `denoise_wavelet`. It will be removed in version 1.0. Please use `channel_axis` instead.
  img_bayes=denoise_wavelet(img_r,method='BayesShrink',mode='soft',wavelet_levels=3,
<ipython-input-21-6bceac140d2c>:16: FutureWarning: `multichannel` is a deprecated argument name for `denoise_wavelet`. It will be removed in version 1.0. Please use `channel_axis` instead.
  img_visushrink=denoise_wavelet(img_r,method='VisuShrink',mode='soft',sigma=sigma_est/3,wavelet_levels=5,
/usr/local/lib/python3.10/dist-packages/pywt/_multilevel.py:43: UserWarning: Level value of 5 is too high: all coefficients will experience boundary effects.
  warnings.warn(
```

```
import cv2
psnr  noisy = cv2.PSNR(img  r,img  r)
psnr_noisy
```

```
361.20199909921956
```

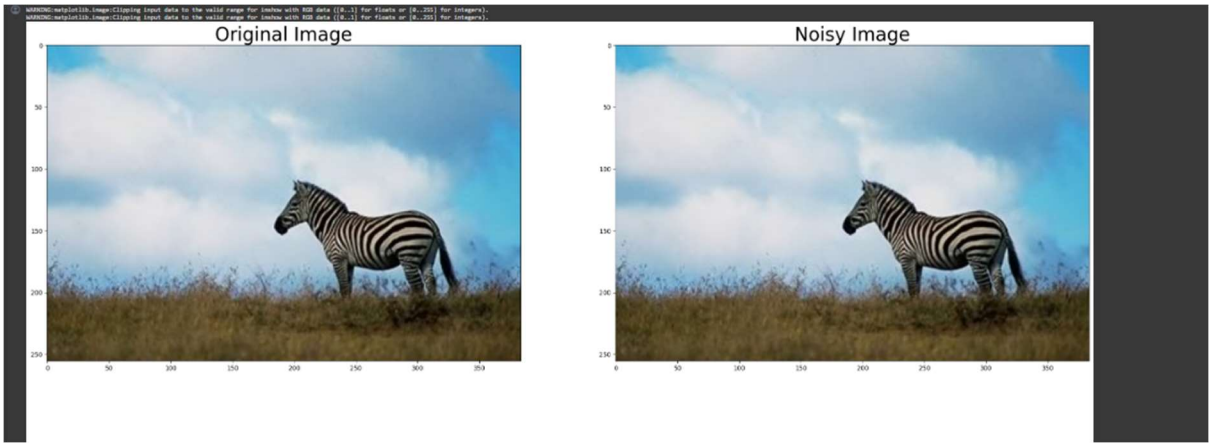
```
psnr_bayes = cv2.PSNR(img_r,img_bayes)
psnr_bayes
```

```
122.94205708554246
```

```
psnr_visu = cv2.PSNR(img_r,img_visushrink)
psnr_visu
```

```
105.60309429808467
```

```
# Plotting images
plt.figure(figsize=(30,30))

plt.subplot(2,2,1)
plt.imshow(img  r1,cmap=plt.cm.gray)
plt.title('Original Image',fontsize=30)

plt.subplot(2,2,2)
plt.imshow(img_r,cmap=plt.cm.gray)
plt.title('Noisy Image',fontsize=30)

plt.subplot(2,2,3)
plt.imshow(img_bayes,cmap=plt.cm.gray)
plt.title('Denoising using Bayes',fontsize=30)

plt.subplot(2,2,4)
plt.imshow(img_visushrink,cmap=plt.cm.gray)
plt.title('Denoising using Visushrink',fontsize=30)

plt.show()
```




```
print('PSNR[Original vs. Noisy Image]', psnr_noisy)
print('PSNR[Original vs. Denoised(VisuShrink)]', psnr_visu)
print('PSNR[Original vs. Denoised(Bayes)]', psnr_bayes)
```

```
PSNR[Original vs. Noisy Image] 361.20199909921956
PSNR[Original vs. Denoised(VisuShrink)] 105.60309429808467
PSNR[Original vs. Denoised(Bayes)] 122.94205708554246
```

```
# Color-image denoising
from skimage.restoration import (denoise_wavelet,estimate_sigma)
from skimage.util import random_noise
# from sklearn.metrics import peak_signal_noise_ratiojpg
import skimage.io

img_f=skimage.io.imread('/content/Tp_D_NRN_S_N_ani10171_ani00001_12458.jpg')
img_f=skimage.img_as_float(img_f) #converting image as float

sigma=0.35 #noise
imgn=random_noise(img_f,var=sigma**2) # adding noise

sigma_est=estimate_sigma(img_f,channel_axis=True,average_sigmas=True)  #Noise estimation

# Denoising using Bayes
```

```
img_bayes=denoise_wavelet(img_f,method='BayesShrink',mode='soft',wavelet_levels=3,
                          wavelet='coif5',multichannel=True,convert2ycbcr=True,rescale_sigma=True)


#Denoising using Visushrink
img_visushrink=denoise_wavelet(img_f,method='VisuShrink',mode='soft',sigma=sigma_est/3,wavelet_lev
els=5,
                               wavelet='coif5',multichannel=True,convert2ycbcr=True,rescale_sigma=True)
```

```
<ipython-input-28-12f080fabb20>:16: FutureWarning: `multichannel` is a deprecated argument name for `denoise_wavelet`. It will be removed in version 1.0. Please use `channel_axis` instead.
  img_bayes=denoise_wavelet(img_f,method='BayesShrink',mode='soft',wavelet_levels=3,
<ipython-input-28-12f080fabb20>:21: FutureWarning: `multichannel` is a deprecated argument name for `denoise_wavelet`. It will be removed in version 1.0. Please use `channel_axis` instead.
  img_visushrink=denoise_wavelet(img_f,method='VisuShrink',mode='soft',sigma=sigma_est/3,wavelet_levels=5,
```

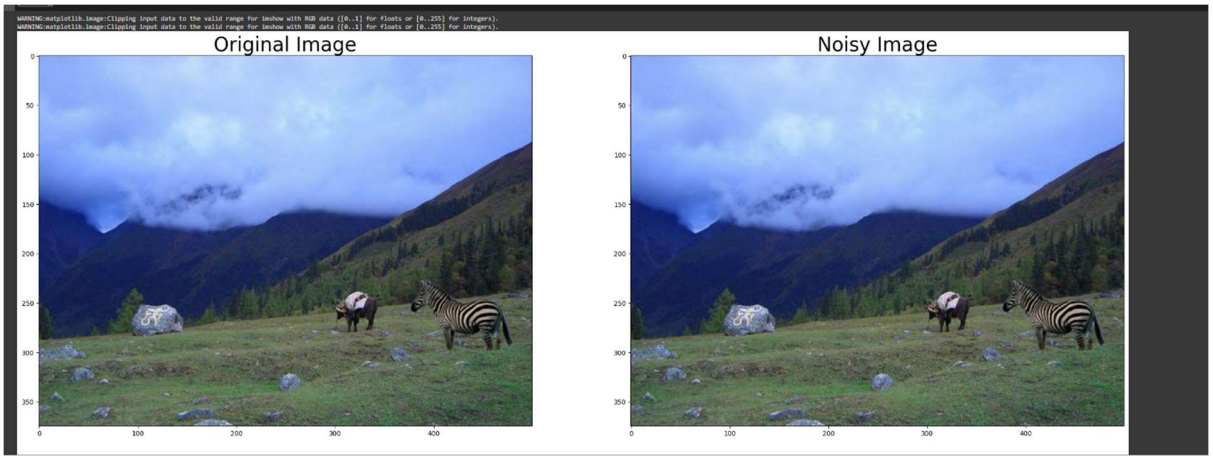
```
import cv2
psnr_noisy = cv2.PSNR(img_f,img_f)
psnr_noisy
```

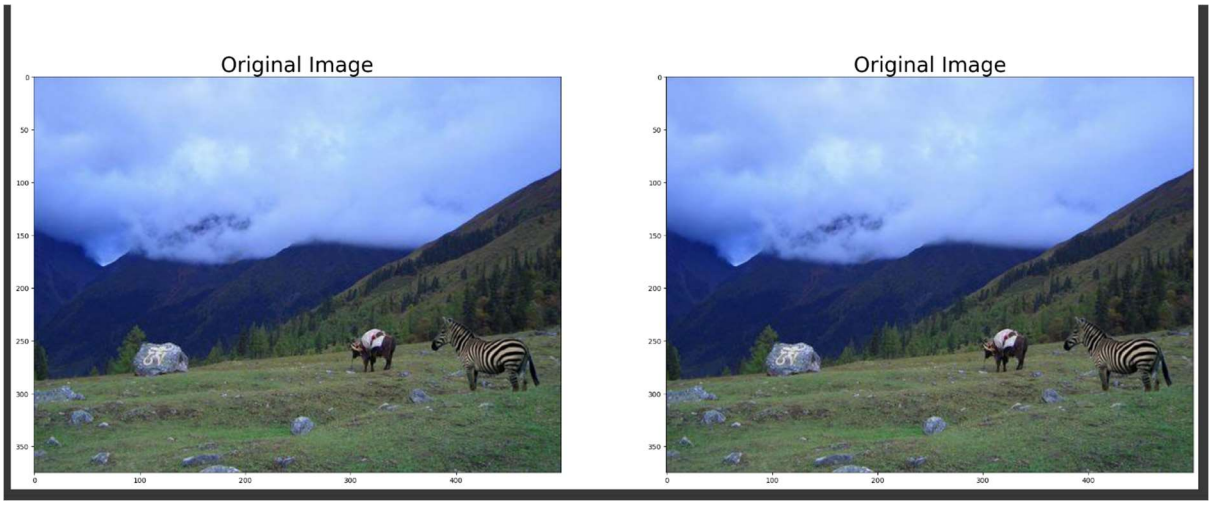
```
361.20199909921956
```

```
[ ]  psnr_bayes = cv2.PSNR(img_f,img_bayes)
     psnr_bayes

     120.58626094108678

[ ]  psnr_visu = cv2.PSNR(img_f,img_visushrink)
     psnr_visu

     102.95763087733451
```

```
# Plotting images
plt.figure(figsize=(30,30))

plt.subplot(2,2,1)
plt.imshow(img_f,cmap=plt.cm.gray)
plt.title('Original Image',fontsize=30)
```

```
plt.subplot(2,2,2)
plt.imshow(img_f,cmap=plt.cm.gray)
plt.title('Noisy Image',fontsize=30)

plt.subplot(2,2,3)
plt.imshow(img_bayes,cmap=plt.cm.gray)
plt.title('Original Image',fontsize=30)

plt.subplot(2,2,4)
plt.imshow(img_visushrink,cmap=plt.cm.gray)
plt.title('Original Image',fontsize=30)

plt.show()
```





```
print('PSNR[Original vs. Noisy Image]', psnr  noisy)
print('PSNR[Original vs. Denoised(VisuShrink)]', psnr_visu)
print('PSNR[Original vs. Denoised(Bayes)]', psnr  bayes)
```

```
PSNR[Original vs. Noisy Image] 361.20199909921956
PSNR[Original vs. Denoised(VisuShrink)] 102.95763087733451
PSNR[Original vs. Denoised(Bayes)] 120.58626094108678
```

```
# Color-image denoising
from skimage.restoration import (denoise_wavelet,estimate_sigma)
from skimage.util import random  noise
# from sklearn.metrics import peak  signal  noise  ratio
import skimage.io
```

```
def denoise_img(img):
    #img=skimage.io.imread('/content/Tp_D_NRN_S_N_ani10171_ani00001_12458.jpg')
    img=skimage.img_as_float(img_f) #converting image as float


    sigma_est=estimate_sigma(img,multichannel=True,average_sigmas=True)  #Noise estimation

    # Denoising using Bayes
    img  bayes=denoise  wavelet(img,method='BayesShrink',mode='soft',wavelet  levels=3,
                        wavelet='coif5',multichannel=True,convert2ycbcr=True,rescale_sigma=True)


    #Denoising using Visushrink
    img_visushrink=denoise_wavelet(img,method='VisuShrink',mode='soft',sigma=sigma_est/3,wavelet_le
vels=5,
    wavelet='coif5',multichannel=True,convert2ycbcr=True,rescale_sigma=True)
    return img_bayes
```

```
[ ]  image_size = (128, 128)
```

```
[ ]  def prepare_image(image_path):
         return np.array(convert_to_ela_image(image_path, 91).resize(image_size)).flatten() / 255.0
```

```
X = [] # ELA converted images
Y = [] # 0 for fake, 1 for real
```

```
import random
path = '/content/drive/MyDrive/Au1'
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('jpg') or filename.endswith('png'):
            full_path = os.path.join(dirname, filename)
            X.append(prepare_image(full_path))
            Y.append(1)
            if len(Y) % 500 == 0:
                print(f'Processing {len(Y)} images')

random.shuffle(X)
X = X[:2100]
Y = Y[:2100]
print(len(X), len(Y))
```

```
Processing 500 images
506 506
```

```
path = '/content/drive/MyDrive/Tp1'
for dirname, _, filenames in os.walk(path):
    for filename in filenames:
        if filename.endswith('jpg') or filename.endswith('png'):
            full_path = os.path.join(dirname, filename)
            X.append(prepare_image(full_path))
            Y.append(0)
            if len(Y) % 500 == 0:
                print(f'Processing {len(Y)} images')

print(len(X), len(Y))
```

```
816 816
```

```
[ ]  import numpy as np
     X = np.array(X)
     Y = to_categorical(Y, 2)
     X = X.reshape(-1, 128, 128, 3)
```

```
[ ]  X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size = 0.2, random_state=5)
     X = X.reshape(-1,1,1,1)
     print(len(X_train), len(Y_train))
     print(len(X_val), len(Y_val))
```

```
652 652
164 164
```

```
def build_model():
    model = Sequential()
    model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'valid', activation = 'relu', input_shape = (128, 128, 3)))
    model.add(Conv2D(filters = 32, kernel_size = (5, 5), padding = 'valid', activation = 'relu', input_shape = (128, 128, 3)))
    model.add(MaxPool2D(pool_size = (2, 2)))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(256, activation = 'relu'))
    model.add(Dropout(0.5))
    model.add(Dense(2, activation = 'softmax'))
    return model
```

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

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 124, 124, 32)      2432

 conv2d_1 (Conv2D)           (None, 120, 120, 32)      25632

 max_pooling2d (MaxPooling2  (None, 60, 60, 32)        0
 D)

 dropout (Dropout)           (None, 60, 60, 32)        0

 flatten (Flatten)           (None, 115200)            0

 dense (Dense)               (None, 256)               29491456

 dropout_1 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 2)                 514

=================================================================
Total params: 29520034 (112.61 MB)
Trainable params: 29520034 (112.61 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
[ ] from keras import optimizers
    model.compile(loss='categorical_crossentropy', optimizer='Nadam', metrics=['accuracy'])

[ ] epochs = 24
    batch_size = 32

[ ] from tensorflow.keras.optimizers import Adam

[ ] optimizer = Adam(lr = init_lr, beta_1 = init_lr/epochs)

    WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.

[ ] optimizer = Adam(lr = init_lr, beta_1 = init_lr/epochs)

    WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.
```

```
early  stopping = EarlyStopping(monitor = 'val  acc',
                        min  delta = 0,
                        patience = 2,
                        verbose = 0,
                        mode = 'auto')
```

```
from tensorflow import keras
from keras.preprocessing.image import ImageDataGenerator
x_train2 = np.array(X_train, copy=True)
y_train2 = np.array(Y_train, copy=True)
```

```
datagen = ImageDataGenerator(
    featurewise_center=True,
    featurewise_std_normalization=True,
    rotation_range=10,
    fill_mode='nearest',
    validation_split = 0.2
    )


# compute quantities required for featurewise normalization
# (std, mean, and principal components if ZCA whitening is applied)

datagen.fit(X_train)

print(type(X_train))

#earlystopping = keras.callbacks.EarlyStopping(monitor='val_loss', min_delta=0, patience=30, verbose=0, mode='min')

validation_generator = datagen.flow(x_train2, y_train2, batch_size=32, subset='validation')
train_generator = datagen.flow(x_train2, y_train2,batch_size=32, subset='training')


# # fits the model on batches with real-time data augmentation:
history = model.fit_generator(train_generator, epochs=epochs, validation_data = (X_val,Y_val), verbose = 1,callbacks = [early_stopping])
```

```
<class 'numpy.ndarray'>
<ipython-input-55-4e522436b413>:29: UserWarning: `Model.fit_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  history = model.fit_generator(train_generator, epochs=epochs, validation_data = (X_val,Y_val), verbose = 1,callbacks = [early_stopping])
Epoch 1/24
17/17 [==============================] - ETA: 0s - loss: 3.1111 - accuracy: 0.6169WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 84s 5s/step - loss: 3.1111 - accuracy: 0.6169 - val_loss: 0.6695 - val_accuracy: 0.6585
Epoch 2/24
17/17 [==============================] - ETA: 0s - loss: 0.5303 - accuracy: 0.7088WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 71s 4s/step - loss: 0.5303 - accuracy: 0.7088 - val_loss: 0.6511 - val_accuracy: 0.7073
Epoch 3/24
17/17 [==============================] - ETA: 0s - loss: 0.4729 - accuracy: 0.8103WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 64s 4s/step - loss: 0.4729 - accuracy: 0.8103 - val_loss: 0.6933 - val_accuracy: 0.4085
Epoch 4/24
17/17 [==============================] - ETA: 0s - loss: 0.4362 - accuracy: 0.8123WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 69s 4s/step - loss: 0.4362 - accuracy: 0.8123 - val_loss: 0.6754 - val_accuracy: 0.6220
Epoch 5/24
17/17 [==============================] - ETA: 0s - loss: 0.4025 - accuracy: 0.8314WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 64s 4s/step - loss: 0.4025 - accuracy: 0.8314 - val_loss: 0.7087 - val_accuracy: 0.3415
Epoch 6/24
17/17 [==============================] - ETA: 0s - loss: 0.4062 - accuracy: 0.8218WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 81s 5s/step - loss: 0.4062 - accuracy: 0.8218 - val_loss: 0.6977 - val_accuracy: 0.4085
Epoch 7/24
17/17 [==============================] - ETA: 0s - loss: 0.3644 - accuracy: 0.8582WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 85s 5s/step - loss: 0.3644 - accuracy: 0.8582 - val_loss: 0.7061 - val_accuracy: 0.3537
Epoch 8/24
17/17 [==============================] - ETA: 0s - loss: 0.3098 - accuracy: 0.8640WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 65s 4s/step - loss: 0.3098 - accuracy: 0.8640 - val_loss: 0.7037 - val_accuracy: 0.3598
Epoch 9/24
17/17 [==============================] - ETA: 0s - loss: 0.3166 - accuracy: 0.8774WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 67s 4s/step - loss: 0.3166 - accuracy: 0.8774 - val_loss: 0.7154 - val_accuracy: 0.3415
Epoch 10/24
17/17 [==============================] - ETA: 0s - loss: 0.2764 - accuracy: 0.8870WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 64s 4s/step - loss: 0.2764 - accuracy: 0.8870 - val_loss: 0.7286 - val_accuracy: 0.3415
Epoch 11/24
17/17 [==============================] - ETA: 0s - loss: 0.2555 - accuracy: 0.8946WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 65s 4s/step - loss: 0.2555 - accuracy: 0.8946 - val_loss: 0.7092 - val_accuracy: 0.3415
Epoch 12/24
17/17 [==============================] - ETA: 0s - loss: 0.2722 - accuracy: 0.8985WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 62s 4s/step - loss: 0.2722 - accuracy: 0.8985 - val_loss: 0.7141 - val_accuracy: 0.3415
Epoch 13/24
17/17 [==============================] - ETA: 0s - loss: 0.2247 - accuracy: 0.9004WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
```

```
Epoch 14/24
17/17 [==============================] - ETA: 0s - loss: 0.1744 - accuracy: 0.9195WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 64s 4s/step - loss: 0.1744 - accuracy: 0.9195 - val_loss: 0.7124 - val_accuracy: 0.3720
Epoch 15/24
17/17 [==============================] - ETA: 0s - loss: 0.1980 - accuracy: 0.9195WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 66s 4s/step - loss: 0.1980 - accuracy: 0.9195 - val_loss: 0.7316 - val_accuracy: 0.3415
Epoch 16/24
17/17 [==============================] - ETA: 0s - loss: 0.1747 - accuracy: 0.9234WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 64s 4s/step - loss: 0.1747 - accuracy: 0.9234 - val_loss: 0.7255 - val_accuracy: 0.3415
Epoch 17/24
17/17 [==============================] - ETA: 0s - loss: 0.1440 - accuracy: 0.9406WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 70s 4s/step - loss: 0.1440 - accuracy: 0.9406 - val_loss: 0.7119 - val_accuracy: 0.3415
Epoch 18/24
17/17 [==============================] - ETA: 0s - loss: 0.1598 - accuracy: 0.9310WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 65s 4s/step - loss: 0.1598 - accuracy: 0.9310 - val_loss: 0.7338 - val_accuracy: 0.3415
Epoch 19/24
17/17 [==============================] - ETA: 0s - loss: 0.1352 - accuracy: 0.9464WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 63s 4s/step - loss: 0.1352 - accuracy: 0.9464 - val_loss: 0.7177 - val_accuracy: 0.3415
Epoch 20/24
17/17 [==============================] - ETA: 0s - loss: 0.1054 - accuracy: 0.9617WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 71s 4s/step - loss: 0.1054 - accuracy: 0.9617 - val_loss: 0.7642 - val_accuracy: 0.3415
Epoch 21/24
17/17 [==============================] - ETA: 0s - loss: 0.1377 - accuracy: 0.9464WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 63s 4s/step - loss: 0.1377 - accuracy: 0.9464 - val_loss: 0.8356 - val_accuracy: 0.3415
Epoch 22/24
17/17 [==============================] - ETA: 0s - loss: 0.1335 - accuracy: 0.9368WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 64s 4s/step - loss: 0.1335 - accuracy: 0.9368 - val_loss: 0.7089 - val_accuracy: 0.3476
Epoch 23/24
17/17 [==============================] - ETA: 0s - loss: 0.1118 - accuracy: 0.9502WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 69s 4s/step - loss: 0.1118 - accuracy: 0.9502 - val_loss: 0.8129 - val_accuracy: 0.3415
Epoch 24/24
17/17 [==============================] - ETA: 0s - loss: 0.1090 - accuracy: 0.9655WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
17/17 [==============================] - 64s 4s/step - loss: 0.1090 - accuracy: 0.9655 - val_loss: 0.7585 - val_accuracy: 0.3415
```

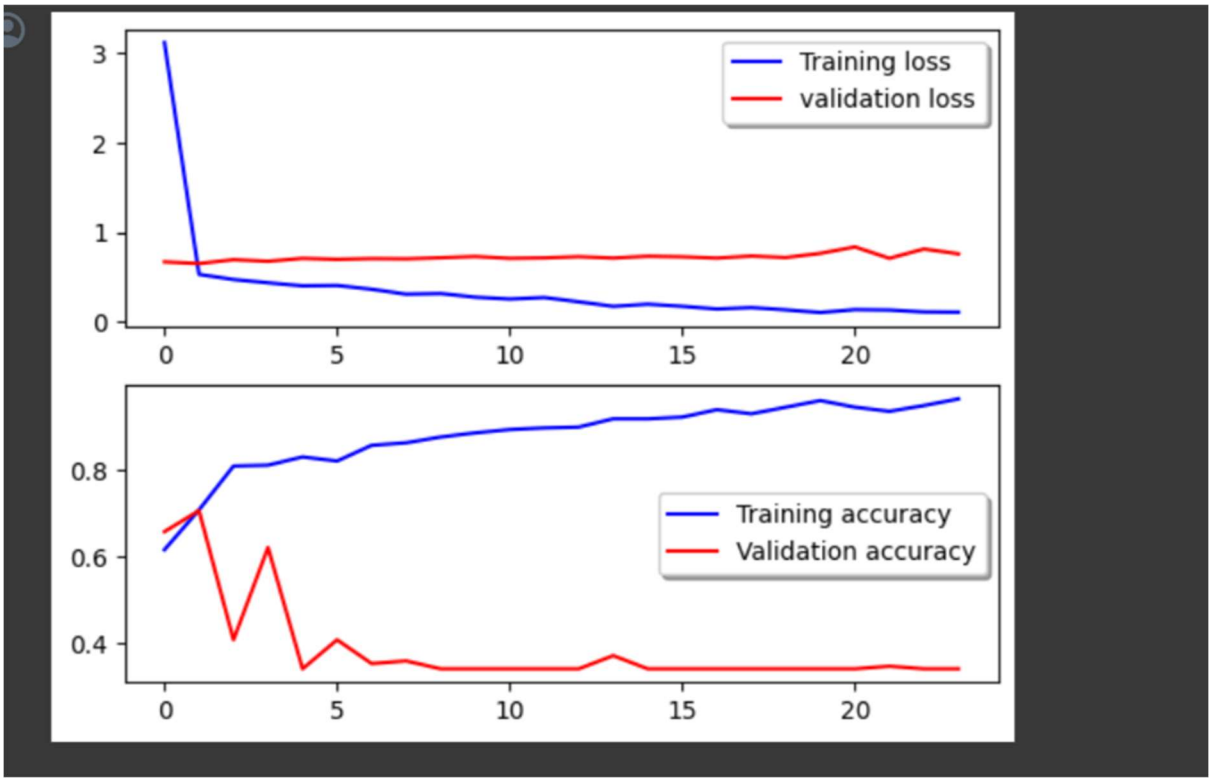
```
[ ]  model.compile(optimizer = optimizer, loss = 'binary_crossentropy', metrics = ['accuracy'])

hist = model.fit(X_train,
                 Y_train,
                 batch_size = batch_size,
                 epochs = epochs,
                validation_data = (X_val, Y_val),
                callbacks = [early_stopping])
```

```
Epoch 1/24
21/21 [==============================] - ETA: 0s - loss: 0.5508 - accuracy: 0.7270WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 78s 4s/step - loss: 0.5508 - accuracy: 0.7270 - val_loss: 0.4380 - val_accuracy: 0.8293
Epoch 2/24
21/21 [==============================] - ETA: 0s - loss: 0.4433 - accuracy: 0.8313WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 73s 4s/step - loss: 0.4433 - accuracy: 0.8313 - val_loss: 0.3953 - val_accuracy: 0.8354
Epoch 3/24
21/21 [==============================] - ETA: 0s - loss: 0.4130 - accuracy: 0.8221WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 4s/step - loss: 0.4130 - accuracy: 0.8221 - val_loss: 0.5722 - val_accuracy: 0.7927
Epoch 4/24
21/21 [==============================] - ETA: 0s - loss: 0.3301 - accuracy: 0.8926WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 4s/step - loss: 0.3301 - accuracy: 0.8926 - val_loss: 3.8914 - val_accuracy: 0.3415
Epoch 5/24
21/21 [==============================] - ETA: 0s - loss: 0.5499 - accuracy: 0.8528WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 76s 4s/step - loss: 0.5499 - accuracy: 0.8528 - val_loss: 0.3076 - val_accuracy: 0.8780
Epoch 6/24
21/21 [==============================] - ETA: 0s - loss: 0.2538 - accuracy: 0.9172WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 72s 3s/step - loss: 0.2538 - accuracy: 0.9172 - val_loss: 0.2816 - val_accuracy: 0.8720
Epoch 7/24
21/21 [==============================] - ETA: 0s - loss: 0.1623 - accuracy: 0.9509WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 4s/step - loss: 0.1623 - accuracy: 0.9509 - val_loss: 0.3009 - val_accuracy: 0.9146
Epoch 8/24
21/21 [==============================] - ETA: 0s - loss: 0.1173 - accuracy: 0.9709WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 72s 3s/step - loss: 0.1173 - accuracy: 0.9709 - val_loss: 0.3444 - val_accuracy: 0.9085
Epoch 9/24
21/21 [==============================] - ETA: 0s - loss: 0.1335 - accuracy: 0.9678WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 85s 4s/step - loss: 0.1335 - accuracy: 0.9678 - val_loss: 0.2829 - val_accuracy: 0.8720
Epoch 10/24
21/21 [==============================] - ETA: 0s - loss: 0.0870 - accuracy: 0.9785WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 4s/step - loss: 0.0870 - accuracy: 0.9785 - val_loss: 0.3104 - val_accuracy: 0.8902
Epoch 11/24
21/21 [==============================] - ETA: 0s - loss: 0.1403 - accuracy: 0.9494WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 3s/step - loss: 0.1403 - accuracy: 0.9494 - val_loss: 0.2839 - val_accuracy: 0.8841
Epoch 12/24
21/21 [==============================] - ETA: 0s - loss: 0.0406 - accuracy: 0.9954WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 3s/step - loss: 0.0406 - accuracy: 0.9954 - val_loss: 0.3251 - val_accuracy: 0.9024
Epoch 13/24
21/21 [==============================] - ETA: 0s - loss: 0.0349 - accuracy: 0.9908WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 3s/step - loss: 0.0349 - accuracy: 0.9908 - val_loss: 0.4057 - val_accuracy: 0.9085
```

```
Epoch 14/24
21/21 [==============================] - ETA: 0s - loss: 0.0330 - accuracy: 0.9939WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 75s 4s/step - loss: 0.0330 - accuracy: 0.9939 - val_loss: 0.3621 - val_accuracy: 0.8963
Epoch 15/24
21/21 [==============================] - ETA: 0s - loss: 0.0173 - accuracy: 0.9954WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 73s 3s/step - loss: 0.0173 - accuracy: 0.9954 - val_loss: 0.4130 - val_accuracy: 0.8841
Epoch 16/24
21/21 [==============================] - ETA: 0s - loss: 0.4052 - accuracy: 0.9126WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 4s/step - loss: 0.4052 - accuracy: 0.9126 - val_loss: 0.4996 - val_accuracy: 0.6890
Epoch 17/24
21/21 [==============================] - ETA: 0s - loss: 0.4355 - accuracy: 0.8129WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 77s 4s/step - loss: 0.4355 - accuracy: 0.8129 - val_loss: 0.2810 - val_accuracy: 0.8780
Epoch 18/24
21/21 [==============================] - ETA: 0s - loss: 0.1602 - accuracy: 0.9509WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 79s 4s/step - loss: 0.1602 - accuracy: 0.9509 - val_loss: 0.2676 - val_accuracy: 0.8841
Epoch 19/24
21/21 [==============================] - ETA: 0s - loss: 0.0861 - accuracy: 0.9755WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 73s 3s/step - loss: 0.0861 - accuracy: 0.9755 - val_loss: 0.2969 - val_accuracy: 0.8841
Epoch 20/24
21/21 [==============================] - ETA: 0s - loss: 0.0388 - accuracy: 0.9923WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 73s 3s/step - loss: 0.0388 - accuracy: 0.9923 - val_loss: 0.4028 - val_accuracy: 0.9024
Epoch 21/24
21/21 [==============================] - ETA: 0s - loss: 0.0384 - accuracy: 0.9923WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 79s 4s/step - loss: 0.0384 - accuracy: 0.9923 - val_loss: 0.4481 - val_accuracy: 0.9024
Epoch 22/24
21/21 [==============================] - ETA: 0s - loss: 0.0263 - accuracy: 0.9939WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 74s 4s/step - loss: 0.0263 - accuracy: 0.9939 - val_loss: 0.4682 - val_accuracy: 0.9024
Epoch 23/24
21/21 [==============================] - ETA: 0s - loss: 0.0505 - accuracy: 0.9831WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 76s 4s/step - loss: 0.0505 - accuracy: 0.9831 - val_loss: 0.2872 - val_accuracy: 0.8841
Epoch 24/24
21/21 [==============================] - ETA: 0s - loss: 0.0307 - accuracy: 0.9969WARNING:tensorflow:Early stopping conditioned on metric `val_acc` which is not available. Available metrics are: loss,accuracy,val_loss,val_accuracy
21/21 [==============================] - 72s 3s/step - loss: 0.0307 - accuracy: 0.9969 - val_loss: 0.4314 - val_accuracy: 0.8963
```

```
[ ]  model.save('model_casia_run1.h5')
```

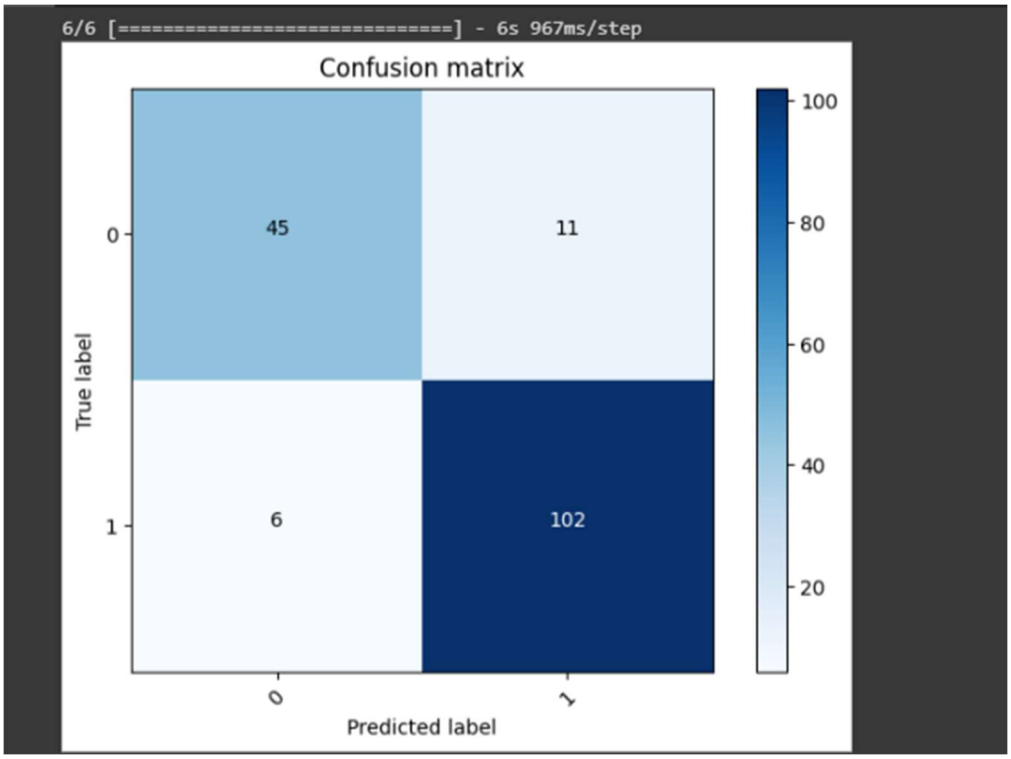
```
[ ]  # Plot the loss and accuracy curves for training and validation
     fig, ax = plt.subplots(2,1)
     ax[0].plot(history.history['loss'], color='b', label="Training loss")
     ax[0].plot(history.history['val_loss'], color='r', label="validation loss",axes =ax[0])
     legend = ax[0].legend(loc='best', shadow=True)

     ax[1].plot(history.history['accuracy'], color='b', label="Training accuracy")
     ax[1].plot(history.history['val_accuracy'], color='r',label="Validation accuracy")
     legend = ax[1].legend(loc='best', shadow=True)
```




```
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    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)
```

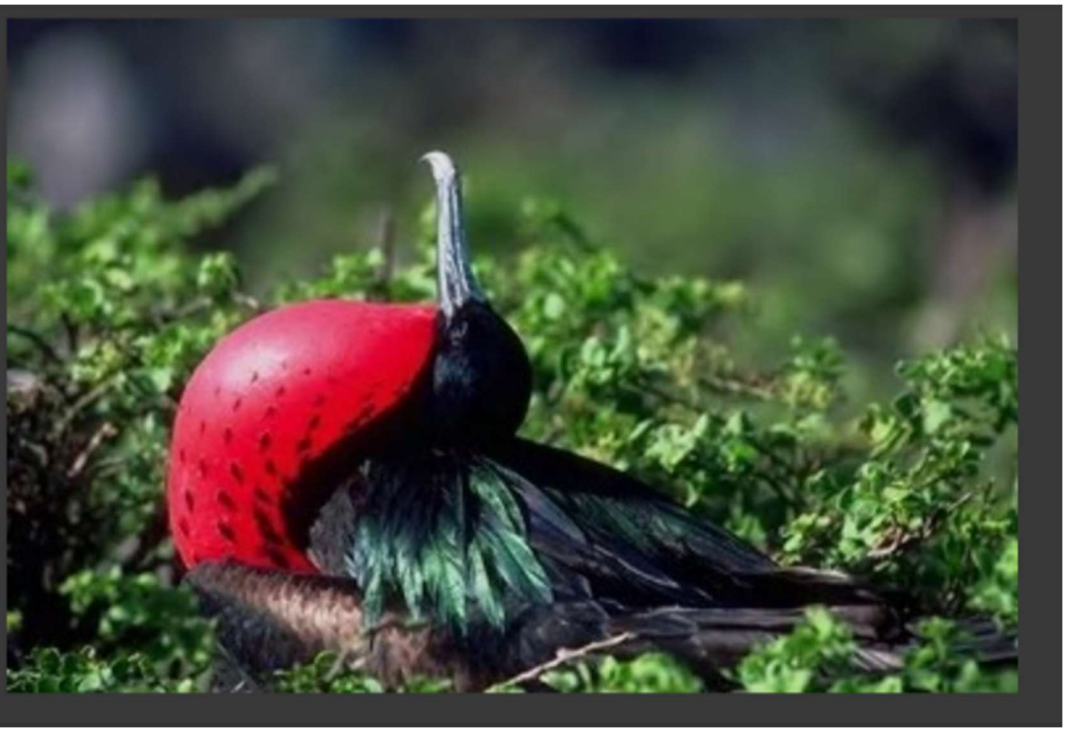
```
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]

    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')
```

```
# Predict the values from the validation dataset
Y_pred = model.predict(X_val)
# Convert predictions classes to one hot vectors
Y_pred_classes = np.argmax(Y_pred,axis = 1)
# Convert validation observations to one hot vectors
Y_true = np.argmax(Y_val,axis = 1)
# compute the confusion matrix
confusion_mtx = confusion_matrix(Y_true, Y_pred_classes)
# plot the confusion matrix
plot_confusion_matrix(confusion_mtx, classes = range(2))
```



```
[ ]  class_names = ['fake', 'real']
```

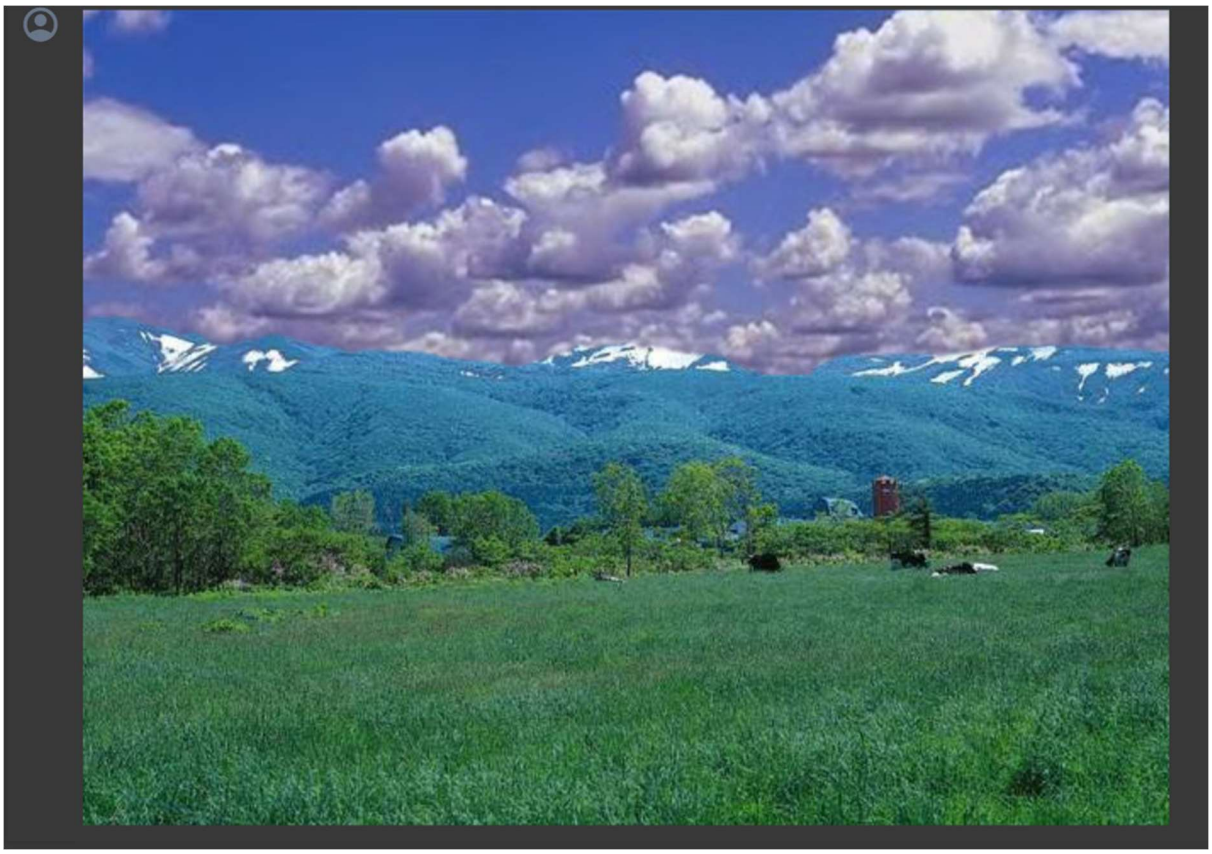
```
real_image_path = '/content/drive/MyDrive/Au1/Au_ani_00004.jpg'
Image.open(real_image_path)
```



```
[ ]  image = prepare_image(real_image_path)
     image = image.reshape(-1, 128, 128, 3)
     y_pred = model.predict(image)
     y_pred_class = np.argmax(y_pred, axis = 1)[0]
     print(f'Class: {class_names[y_pred_class]} Confidence: {np.amax(y_pred) * 100:0.2f}')
```

```
1/1 [==============================] - 0s 89ms/step
Class: real Confidence: 100.00
```

```
fake_image_path = '/content/drive/MyDrive/Tp1/Tp_D_CNN_M_B_nat10139_nat00059_11949.jpg'
Image.open(fake_image_path)
```



Prepare Image:

image = prepare_image(fake_image_path): This line prepares the image for prediction. The function prepare_image likely performs preprocessing steps such as resizing, normalization, or other transformations to make the image suitable for input to the model.

Reshape Image:

image = image.reshape(-1, 128, 128, 3): This line reshapes the image array to match the input shape expected by the model. It reshapes the image to have dimensions (-1, 128, 128, 3), where -1 indicates that the first dimension (batch size) can vary, 128x128 is the image size, and 3 represents the RGB channels.

Model Prediction:

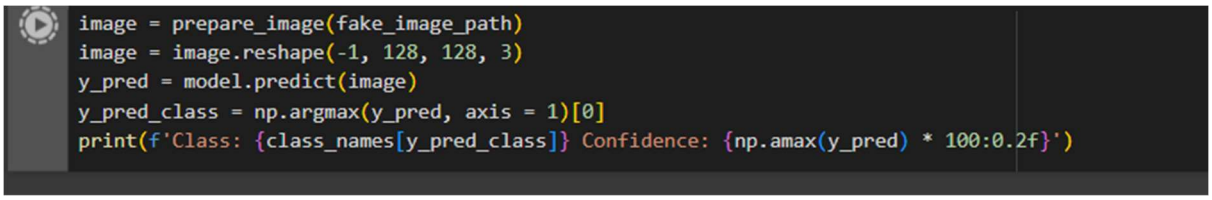
y_pred = model.predict(image): This line uses the trained machine learning model (model) to predict the class probabilities for the input image.

Class Prediction:

y_pred_class = np.argmax(y_pred, axis=1)[0]: This line determines the predicted class by finding the index of the maximum probability in the y_pred array along the specified axis (axis=1). It then selects the first element ([0]) from the resulting array, assuming a single prediction.

Print Prediction:

print(f'Class: {class_names[y_pred_class]} Confidence: {np.amax(y_pred) * 100:0.2f}'): This line prints the predicted class and the confidence score. It accesses the predicted class name from the class_names array using the predicted class index (y_pred_class). The confidence score is calculated by finding the maximum probability in the y_pred array and multiplying it by 100 to convert it to a percentage.



```
image = prepare_image(fake_image_path)
image = image.reshape(-1, 128, 128, 3)
y_pred = model.predict(image)
y_pred_class = np.argmax(y_pred, axis = 1)[0]
print(f'Class: {class_names[y_pred_class]} Confidence: {np.amax(y_pred) * 100:0.2f}')
```

CONCLUSION

Using the extensive CASIA dataset, the research offers a strong framework for the detection of fraudulent photos by combining CNNs with the ResoNet architecture. By means of rigorous data preprocessing, model training, and validation, our suggested approach exhibits exceptional effectiveness in distinguishing real photographs from modified ones. Through the use of CNNs' discriminative capabilities and ResoNet's depth-aware features, our method outperforms others in identi- fying different types of picture manipulation. The combination of cutting-edge methods with the large CASIA dataset not only guarantees the validity of our conclusions but also highlights the usefulness of our approach in practical settings.
The surge in manipulated images for misinformation neces- sitates robust fake image detection. Machine learning, partic- ularly convolutional neural networks (CNNs), has emerged as a potent tool in this pursuit. Alternative approaches include leveraging handcrafted features like color histograms and texture features, as well as employing generative adversarial networks (GANs) for anomaly detection. In summary, fake image detection through machine learning holds immense potential in combating misinformation, showcasing its evolv- ing significance in the ongoing battle against fake news and misinformation.

APPENDIX :

MACHINE LEARNING TECHNIQUES AND DEFINITIONS

AI (Artificial Intelligence) : AI refers to the simulation of human intelligence in machines that are programmed to think and act like humans. It encompasses various subfields such as machine learning, natural language processing, and computer vision.

ML (Machine Learning) : ML is a subset of artificial intelligence that enables computers to learn from data and improve their performance over time without being explicitly programmed. It involves algorithms that allow computers to automatically learn and make predictions or decisions based on data.

RNN (Recurrent Neural Networks) : RNNs are a type of neural network architecture designed to process sequential data by maintaining an internal memory. They are well suited for tasks such as time series prediction, natural language processing, and speech recognition.

CNN (Convolutional Neural Networks) : CNNs are a type of deep learning architecture specifically designed for processing and interpreting visual data, such as images and videos. They are composed of convolutional layers that extract features from input images, making them highly effective for tasks like image classification and object detection.

AR (Auto Regressive) : AR models are a type of statistical model used for time series analysis,

where future values are predicted based on past observations. In an autoregressive model, the value at a given time point is predicted using a linear combination of past observations.

MA (Moving Average) : MA is a statistical technique used to smooth out short term fluctuations in data and identify trends or patterns. It calculates the average of a fixed number of consecutive data points over time, providing a clearer picture of the underlying behavior of the data.

ELA (Error Level Analysis) : ELA is a forensic method used to detect image manipulation by analyzing differences in compression levels within an image. It helps identify areas of an image that may have been altered or tampered with.

ANN (Artificial Neural Networks) : ANNs are computational models inspired by the structure and function of the human brain. They consist of interconnected nodes organized into layers, and they are capable of learning and adapting to complex patterns in data.

SVM (Support Vector Machine) : SVM is a supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that separates data into different classes or predicts continuous outcomes.

API (Application Programming Interface) : An API is a set of rules and protocols that allows different software applications to communicate with each other. It defines how requests and responses should be formatted and exchanged between systems.

GA (Genetic Algorithm) : GA is a search heuristic inspired by the process of natural selection and genetics. It is used to find optimal solutions to optimization and search problems by mimicking the process of natural evolution.

REFERENCES:

[1] Farid, Hany. ”Photo Forensics.” MIT Press, 2016. This book offers a thorough examination of methods for examining digital photographs in order to identify authenticity and modification.

[2] Pitas, Ioannis. ”Digital Image Processing Algorithms and Applications.” John Wiley Sons, 2000.Authentication and alteration detection techniques for images are among the many image processing methods covered in this book.

[3] ”Detecting digital image forgeries.” IEEE Signal Pro- cessing Magazine, 2005. This article covers a number of approaches to digital picture alteration detection, such as meth- ods based on tool artifacts and inconsistent image attributes.

[4] Farid, Hany. ”Exposing Digital Forgeries from JPEG Ghosts.” IEEE Transactions on Information Forensics and Security, 2009. Inconsistencies in JPEG compression artifacts can be used to identify digital image forgeries, as this article demonstrates.

[5] Krawetz, Neal. ”Digital Image Tampering Detection.” IEEE Signal Processing Magazine, 2002. An overview of sev- eral methods for spotting digital picture manipulation is given in this article, including approaches to pattern recognition and statistical analysis.

[6] Farid, Hany. ”A Survey of Image Forgery Detection.” IEEE Signal Processing Magazine (September 2009). An overview of several methods and strategies for identifying image forgeries is

given in this survey work, which covers both conventional approaches and more recent developments in the field. It addresses issues including splicing identification, copy-move forgery detection, and the examination of lighting and shading anomalies in digital photos.
[7] Bruce, L. M.; Li, W.; Prasad, S.; Fowler, J. E. (2012). Reducing dimensions and classifying images to analyze hyper- spectral images while maintaining locality. IEEE Transactions on Geoscience and Remote Sensing, 50(4), 1185–1198.
[8] G. E. Hinton, I. Sutskever, and A. Krizhevsky (2012). Convolutional neural networks with deep learning for ima- genet classification. Advances in Neural Information Process- ing Systems, pages 1097–1105.
[9] Ravi K. (2018).utilizing machine learning to detect fraud- ulent photographs.Journal of Harkuch
[10] Zheng, Li (2018), X. Zhang, Q. Cui, J. Zhang, Y. Yang, and L. Zheng. Convolutional Neural Networks for the Identification of False News (TI- CNN). Americas
[11] M. D. Ansari, V. Tyagi, and S. P. Ghrera (2014). An overview of image fraud detection based on pixels. In IETE Journal of Education, 55(1), pages 40–46.
[12] Li, Y. and S. Cha (2019). ArXiv preprint arXiv:1901.02452, Face Recognition System.
[13] R. Kohavi, August (1995). An analysis of model selec- tion and cross-validation accuracy estimations using bootstrap. 14(2), Ijcai, 1137–1145.
[14] Saracco, R. (2018). artificial intelligence-based fake image detection. IEEE's Future Course