UNIVERSITY INSTITUTE OF TECHNOLOGY
BARKATULLAH UNIVERSITY, BHOPAL

Department of Computer Science and Engineering

MAJOR PROJECT

On

"DEEPFAKE IMAGE DETECTION"

Submitted In Partial Fulfillment of the Requirement for the

Award of the Degree Of

Bachelor of Technology (B. Tech.)

Year: 2024

Barkatullah University, Bhopal

Submitted by:

VINAYAK MODI **ANKIT PATIL** **ARYAN SINGH**

Under the Guidance
Of

Ms. Kavita Chourasia
Co-Guide
CSE, UIT-BU, Bhopal

Dr. Kamini Maheshwar
Co-Guide
CSE, UIT-BU, Bhopal

Dr. Divakar Singh
Guide & HOD CSE
CSE, UIT-BU, Bhopal

Prof. N. K. Gaur
DIRECTOR
UIT-BU, Bhopal

UNIVERSITY INSTITUTE OF TECHNOLOGY
BARKATULLAH UNIVERSITY, BHOPAL

Department of computer Science and Engineering

CERTIFICATE
Year 2023-2024

This is to certify that minor project entitled **"DEEPFAKE IMAGE DETECTION"** submitted by **Vinayak Modi, Ankit Patil and Aryan Singh** in their partial fulfillment of the requirement for the award of degree of Bachelor of Engineering in Computer Science and Engineering in the year 2024.

Ms. Kavita Chourasia
Co-Guide
CSE, UIT-BU, Bhopal

Dr. Kamini Maheshwar
Co-Guide
CSE, UIT-BU, Bhopal

Dr. Divakar Singh
Guide & HOD CSE
CSE, UIT-BU, Bhopal

Prof. N. K. Gaur
DIRECTOR
UIT-BU, Bhopal

UNIVERSITY INSTITUTE OF TECHNOLOGY
BARKATULLAH UNIVERSITY, BHOPAL

Department of computer Science and Engineering

DECLARATION

Year 2020-2024

Hereby declare that the Minor Project work being presented in this report entitled **"DEEPFAKE IMAGE DETECTION"** using python submitted in the department of computer science, FACULTY OF TECHONOLGY, BARATULLAH UNIVERSITY INSTITUE OF TECHNOLOGY BHOPAL is the authentic work carried out by us under the guidance of Ms. Kavita Chourasia, Dr. Kamini Maheshwar Assistant Professor and Dr. Divakar Singh HOD Department of Computer Science Engineering, Barkatullah University, Bhopal

I declare that the work presented in this dissertation is original, it has not been submitted for any degree in any other universities.

VINAYAK MODI

ANKIT PATIL

ARYAN SINGH

Date: _ \ _ \ _

ACKNOWLEDGEMENT

We would like to express special thanks and gratitude to Ms. Kavita Chourasia and Dr. Kamini Maheshwar for help, guidance and throughout the work for our project without which we would not have been able to complete this project to such a success. We would also like to extend our special thanks and gratitude to HOD Dr. Divakar Singh for providing effective platform and support in the development of this project. And finally, we would like to render our thanks to Director Dr. N.K Gaur for his guidance in this major project titled "DEEPFAKE IMAGE DETECTION".

Last but the least we would like to thank our parents and friends for their support and cooperation. Regardless of the source, we wish to express our gratitude to those who may have contributed to this work, even though anonymously.

Submitted By:

VINAYAK MODI

ANKIT PATIL

ARYAN SINGH

ABSTRACT

The growing computation power has made the deep learning algorithms so powerful that creating an indistinguishable human synthesized video popularly called as deep fakes have become very simple. Scenarios where this realistic face swapped deep fakes are used to create political distress, fake terrorism events, revenge porn, blackmail peoples are easily envisioned. In this work, we describe a new deep learning-based method that can effectively distinguish AI-generated fake videos from real videos. Our method is capable of automatically detecting the replacement and reenactment deep fakes. We are trying to use Artificial Intelligence (AI) to fight Artificial Intelligence (AI).

Our system uses a Convolution neural network to extract the frame-level features and these features and further used to train Recurrent Neural Network (RNN) to classify whether the video is subject to any kind of manipulation or not, i.e., whether the video is deep fake or real video.

LIST OF FIGURES

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
PIP	Performance Improvement Plan
AI	Artificial Intelligence
GUI	Graphics User Interface
IDE	Integrated Development Environment
SSD	Solid States Drive

TABLE OF CONTENTS

CHAPTER 1

CHAPTER 1

INTRODUCTION

Artificial intelligence (AI) is used in deepfakes to produce synthetic material that mimics real world image, audio, visual, or video tape content. These edited media are constantly used to give the print that someone is talking or acting in a way that they no way would. Deepfakes may be used for good, like in jest, but they can also be used bad, like propagating false information, swaying public opinion, and ruining people's reports. The necessity for effective deepfake discovery ways is rising due to the possible detriment that deepfakes might do. These ways might be used to descry deepfakes before they come extensively circulated or to warn people to possible deepfakes. Deepfakes are a form of synthetic media that use artificial intelligence to produce realistic images of people performing conduct they no way actually did. They can be used for vicious purposes, similar as spreading false information or harming reports. Fake content might constitute only a small part of an otherwise long real video, as was initially suggested in. Such short modified segments have the power to alter the meaning and sentiment of the original content completely.
Deepfakes utilize artificial intelligence, particularly Generative Adversarial Networks (GANs), to learn and mimic the characteristics of real images and videos. Through clever manipulation of facial features, body movements, and even voices, deepfakes can create convincing illusions of people saying or doing things they never did.

1.1PURPOSE/ OBJECTIVE:

- Deepfake manipulated visuals can be spread through social media and news outlets, creating widespread confusion and eroding trust in legitimate information sources. Deepfake detection technology helps identify and debunk these false narratives, safeguarding the integrity of news and discourse.
- Deepfake can be used to damage reputations, spread slander, or even incite harassment and violence. Deepfake detection technology empowers individuals to protect their identities and fight back against such malicious manipulations.
- Deepfake can sway public opinion and undermine the integrity of democratic processes. Deepfake detection helps safeguard elections by exposing these fabricated materials and upholding fair democratic practices.
- Developing accessible and user-friendly deepfake detection tools empowers the public to become active participants in safeguarding the digital space.

Deepfake image detection serves as a critical line of defense against the negative impacts of manipulated visuals. By safeguarding against misinformation, protecting individual reputations, and upholding democratic processes, it paves the way for a more trustworthy and accountable digital future.

1.2EXISTING SYSTEM:

Deepfakes, the hyper-realistic synthetic images and videos generated through AI, have become increasingly sophisticated, blurring the lines between reality and fabrication. This raises critical concerns about misinformation and the potential for harm in areas like politics, finance, and personal reputation. To combat this, a diverse arsenal of deepfake image detection models has emerged, each wielding unique strengths to unmask these digital illusions.

1. **Anomaly Detection:** Imagine a seasoned detective scrutinizing a crime scene for subtle clues. Anomaly detection models operate similarly, analyzing vast datasets of real images to identify minute inconsistencies in manipulated ones. These anomalies can range from unnatural skin textures and lighting artifacts to subtle inconsistencies in facial expressions or body movements.
2. **Temporal Analysis:** Think of a meticulous film editor meticulously combing through footage for continuity errors. Temporal analysis models adopt a similar approach, scrutinizing video frames for inconsistencies across time. Flickering pixels, unnatural body movements, and inconsistencies in facial expressions across frames can all betray the handiwork of deepfakes.
3. **Biometric Analysis:** Just like fingerprints or iris scans, facial features and other physical characteristics can serve as unique identifiers. Biometric analysis models leverage this principle, comparing facial features and other biometric data in manipulated images to known reference data. Deviations from the reference data can expose inconsistencies suggestive of deepfakes.
4. **Meta-data Analysis:** The digital world leaves behind a trail of breadcrumbs, and meta-data analysis models are skilled at following them. They delve into the file metadata and creation timestamps of images and videos, searching for inconsistencies or anomalies that might point towards manipulation. For example, inconsistencies between the creation date of an image and the depicted event can raise red flags.

1.3 PROPOSED SYSTEM:

Detecting deepfakes involves analyzing the multimedia content to determine whether it has been tampered with or is original.

Traditionally, forgery detection techniques were used to detect deepfakes. However, in recent years, AI-based deepfake detection techniques have become more popular. Researchers have developed several deep learning models, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Generative Adversarial Networks (GANs), to detect deepfakes with increasing accuracy.

Benefits:

- **Combating Misinformation:** Deepfakes can be powerful tools for disseminating false information. Deepfake image detection helps mitigate this threat by identifying and flagging manipulated content. This fosters a more reliable information environment and protects users from being misled.

- **Enhancing Online Security:** Deepfakes pose a risk to online security by potentially bypassing facial recognition systems. Deepfake detection strengthens these security measures by exposing attempts to impersonate legitimate users. This safeguards sensitive information and protects against fraudulent activities.

- **Protecting Reputations:** Deepfakes can be used to damage someone's reputation by creating manipulated images. Deepfake image detection helps shield individuals from such attacks by verifying the authenticity of images. This fosters trust and protects personal brands.

- **Promoting Media Literacy:** Deepfake detection empowers users to critically evaluate online content. By identifying manipulated images, users become more discerning consumers of information, fostering a more media-literate society.

- **Bolstering Democratic Processes:** Deepfakes can be used to manipulate elections by discrediting candidates or swaying public opinion. Deepfake detection helps safeguard democratic processes by ensuring the authenticity of information used to influence voters.

1.4FEASIBILITY REPORT:

A feasibility study considers various constraints within which the system should be implemented and operated. In this stage the resource needed for implementation such as computing equipment, manpower and costs are estimated. The estimated are compared with available resources and a cost benefit analysis of the system is made. The main objectives of the feasibility study are to determine whether the project would be feasible in terms of the following categories:

- Technical feasibility: Since the android application uses software technologies and tools which are freely available and technical skills required can be easily manageable but requires a normal computing and the system server must be adequate and manageable in future. So, it is found that the hardware and software meet the need of the system. So, it’s clear that the proposed project is technically feasible.

- Economic feasibility: Economic feasibility attempts to weigh the costs of developing and implementing a new system, against the benefits that would gather from having the new system in place. This feasibility study gives the top management the economic justification for the new system. A simple economic analysis which gives the actual comparison of costs and benefits are much more

meaningful in this case. In addition, this proves to be a useful point of reference to compare actual costs as the project progresses.

- Operational feasibility: Since the android application is interactive and data driven, the user need to be only a bit familiar with the software system backed with graphical explanations and that can easily be understood faster in time with usage.

- Schedule feasibility: The dateline of a software system can be easily estimated if the proper team and achievable goals are formed.

CHAPTER 2

CHAPTER 2

REQUIREMENTS

2.1 SOFTWARE REQUIREMENTS:

- Operating system: Windows 10 or higher
- IDE: Kaggle Notebook
- GUI: Gradio or Hugging Faces
- Programming Language: Python
- Python Libraries: Tensor Flow, Keras, Delib, OS
- Platform: Hugging Face..
- Web Browser: Google Chrome, Microsoft Edge

2.2 HARDWARE REQUIREMENTS:

- Processor: Intel Core i5 or AMD Processor
- Storage: 500 GB SSD or higher
- RAM: 8 GB or higher
- GPU: 12GB or higher

CHAPTER 3

CHAPTER 3

TECHNICAL DESCRIPTION

3.1 PROJECT EXPLANATION

A generative deep learning system called DeepFake produces or modifies facial characteristics in a super-realistic manner that makes it difficult to tell the difference between actual and artificial features. This technology has developed significantly and encourages a variety of uses.

Thus, we will examine the problem of how DeepFake detection attempts to produce a generic DeepFake detection model. Lastly, the difficulties in creating and identifying DeepFakes will be covered.

- Deepfake manipulated visuals can be spread through social media and news outlets, creating widespread confusion and eroding trust in legitimate information sources. Deepfake detection technology helps identify and debunk these false narratives, safeguarding the integrity of news and discourse.
- Deepfake can be used to damage reputations, spread slander, or even incite harassment and violence. Deepfake detection technology empowers individuals to protect their identities and fight back against such malicious manipulations.
- Deepfake can sway public opinion and undermine the integrity of democratic processes. Deepfake detection helps safeguard elections by exposing these fabricated materials and upholding fair democratic practices.
- Developing accessible and user-friendly deepfake detection tools empowers the public to become active participants in safeguarding the digital space.

Deepfake image detection serves as a critical line of defense against the negative impacts of manipulated visuals. By safeguarding against misinformation, protecting individual reputations, and upholding democratic processes, it paves the way for a more trustworthy and accountable digital future.

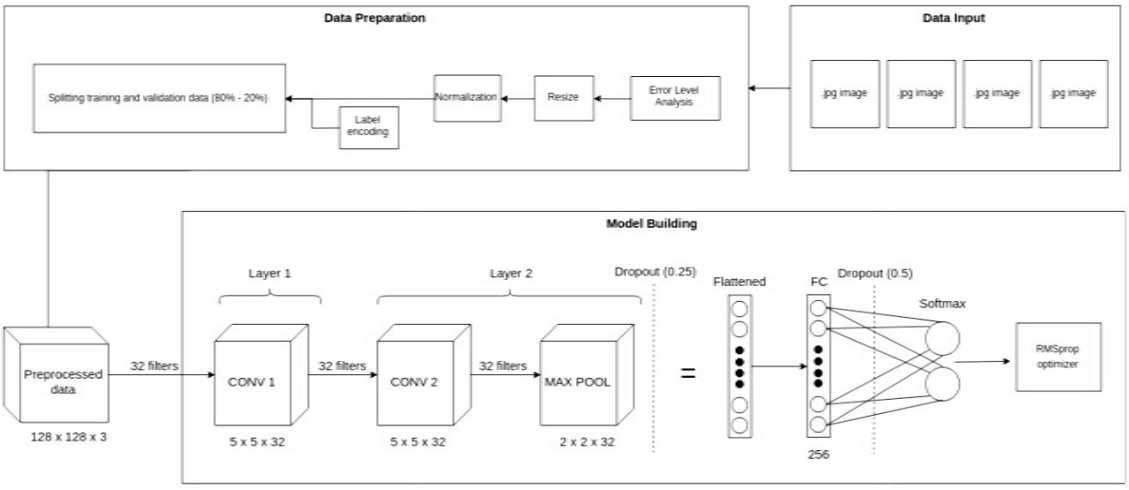


Fig. 3.1: Block Diagram of a Data Preparation and Model Building Process.

Above block diagram of a data preparation and model building process.

Data Preparation:

1. **Data Input:** The process starts with inputting image data, likely a collection of JPG images.
2. **Preprocessing:**
   - **Splitting:** The data is then split into training and validation sets, typically in an 80/20 ratio. The training set is used to train the model, while the validation set is used to assess its performance and prevent overfitting.
   - **Normalization:** Pixel values in the images are likely normalized to a specific range, often between 0 and 1, to prepare them for feeding into the neural network.
   - **Resize:** Images may be resized to a standard dimension to ensure consistency in the input data.
   - **Error Level:** Images may be analyzed for errors or corruptions before being fed into the model.
3. **Analysis:**
   - **Label encoding:** Image labels, likely indicating the category each image belongs to, are encoded into a numerical format that the model can understand.

Model Building:

The model building section appears to show a convolutional neural network architecture with two convolutional layers, followed by a max pooling layer, two dropout layers, a flattening layer, a fully connected layer, and finally a softmax output layer with an RMSprop optimizer.

- **Convolutional Layers:** These layers extract features from the input images using learnable filters. The first layer has 32 filters, meaning it can detect 32 different features in the images. The second layer also has 32 filters, but it operates on the outputs of the first layer, potentially detecting more complex features.
- **Max Pooling Layer:** This layer reduces the dimensionality of the data by taking the maximum value from a small region of the input. This helps to control overfitting and makes the model more robust to variations in the input data.
- **Dropout Layers:** These layers randomly deactivate a certain percentage of neurons in the network during training. This helps to prevent overfitting by forcing the model to learn more generalizable features.
- **Flattening Layer:** This layer converts the multi-dimensional output of the convolutional layers into a one-dimensional vector that can be fed into the fully connected layer.
- **Fully Connected Layer:** This layer connects all the neurons in the previous layer to all the neurons in the output layer. It learns to combine the features extracted by the convolutional layers to make predictions about the image class.
- **Softmax Output Layer:** This layer outputs the probability distribution of the image belonging to each class. The class with the highest probability is predicted as the image class.
- **RMSprop Optimizer:** This is an algorithm used to update the weights of the neurons in the network during training. It helps the model to learn quickly and efficiently.

3.2 TECHNOLOGY USED

1. **Convolutional Neural Networks (CNNs):** Unveiling the Secrets of Image Recognition. CNNs are a powerful type of deep learning architecture specifically designed to excel at image recognition and processing tasks. Unlike traditional neural networks, CNNs leverage a unique structure inspired by the human visual cortex. This structure allows them to efficiently extract features and patterns from images.

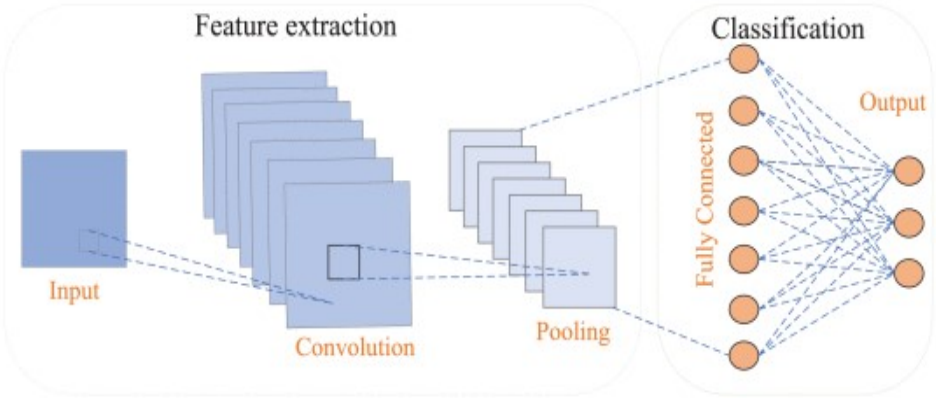


Fig. 3.2**:** The Basic architecture Of CNN

Key Components of a CNN:

- **Convolutional Layers:** The heart of a CNN, these layers apply filters (also called kernels) to the input image. As the filter slides across the image, it performs element-wise multiplication, capturing spatial relationships between pixels. This process generates feature maps that highlight specific aspects of the image, like edges, shapes, or colors.
- **Pooling Layers:** These layers downsample the feature maps, reducing their dimensionality while preserving essential information. Pooling techniques like max pooling select the maximum value within a specific window, summarizing the most prominent features. This helps control overfitting and reduces computational complexity.
- **Activation Layers:** These layers introduce non-linearity into the network, allowing it to learn complex patterns. Common activation functions include ReLU (Rectified Linear Unit) and tanh (hyperbolic tangent).
- **Fully-Connected Layers:** In the final stages, the network transitions to fully-connected layers, similar to those found in traditional neural networks. These layers receive the flattened output from the convolutional layers and perform classifications or predictions based on the learned features.

2. **Recurrent Neural Networks (RNNs)** are a type of deep learning architecture specifically designed to handle sequential data, where the order of elements matters. Unlike Convolutional Neural Networks (CNNs) that excel at analyzing grids like images, RNNs are adept at understanding the relationships between elements in a sequence.

   The defining characteristic of RNNs is their internal memory, allowing them to process information from previous steps in a sequence and influence their current

predictions. This capability makes them ideal for tasks like language translation, speech recognition, and time series forecasting.

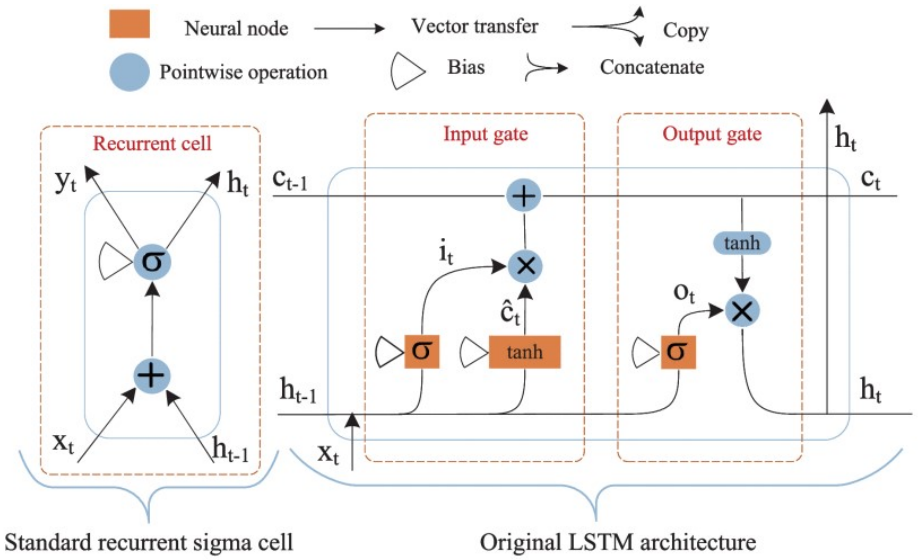


Fig. 3.3: The basic architecture of a RNN

Structure of an RNN:

- **Input Layer:** Receives the first element of the sequence data.
- **Hidden Layer:** The heart of the RNN, where the magic happens. It contains a loop that processes the current input and combines it with the information retained from the previous step. This loop allows the network to maintain a state that captures the context of the sequence.
- **Output Layer:** Generates the output for the current step, based on the processed information in the hidden layer. This output can be a prediction, classification, or another element in the sequence.

How RNNs Work:

1. **Initialization:** The hidden state of the RNN is set to zero or a random value.
2. **Processing the Sequence:** For each element in the sequence:
   - The element is fed into the input layer.
   - The input layer and the current hidden state are combined within the hidden layer.
   - An activation function is applied to the output of the hidden layer, introducing non-linearity.
   - The output layer generates an output based on the processed information in the hidden layer.
   - The hidden state is updated by incorporating the current input and the previous hidden state. This creates a chain of information that allows the network to consider past elements when processing the current one.
3. **Generating the Final Output:** Once the entire sequence is processed, the final output from the hidden layer or the output layer can be used for prediction or further processing.

3. **TensorFlow** is an open-source software library that empowers developers to build and train sophisticated machine learning models for various tasks. Its flexibility, wide range of applications, and active community make it a popular choice.
4. **Keras** is designed to be easy to use, while still being powerful and flexible. It provides a high-level interface for building and training deep learning models, making it a good choice for beginners and experienced developers alike.
5. **PyTorch** is an open-source deep learning framework built on the Python programming language and the Torch library. It's become a popular choice for researchers and developers due to its flexibility, ease of use, and focus on dynamic computation graphs.
6. While "**delib** deepfake" isn't a widely recognized term, it's likely a misspelling of "deliberate deepfake" or referring to a specific project.
7. **OpenCV**, or Open Source Computer Vision Library, is a powerful and widely used open-source library for computer vision and machine learning applications. It provides a comprehensive set of algorithms and functions for real-time image and video processing, object detection, facial recognition, motion tracking, and much more.
8. **Hugging face:** Hugging Face is both a company that creates machine learning tools and a platform for the machine learning community. The platform offers pre-trained models, datasets, and tools specifically useful for natural language processing tasks. It's a valuable resource for developers looking to build AI-powered applications.

9. **Installation:**

   The first line !pip install -U -q evaluate transformers datasets>=2.14.5 accelerate>=0.27 2>/dev/null installs several Python libraries required for the script to run. Here's what each library does:

   - evaluate: This library is used for model evaluation metrics.
   - transformers: This library provides pre-trained models and functionalities for working with transformers, a type of neural network architecture.
   - datasets: This library helps load and process datasets in various formats.
   - accelerate: This library is used for mixed-precision training and distributed training on multiple GPUs or TPUs.

   The flags used with pip are:

   - -U: Upgrades already installed packages to the latest version.
   - -q: Runs the installation in quiet mode without outputting messages.
   - 2>/dev/null: Redirects any error messages to /dev/null, effectively hiding them.

10. **Imports:**

    The rest of the code imports various Python libraries needed for the script's functionality. Here's a summary of some important ones:

    - warnings: Used to manage warning messages.
    - gc: Used for garbage collection.
    - numpy (np): Provides numerical computing functionalities.
    - pandas (pd): Used for data manipulation and analysis (often with DataFrames).
    - itertools: Provides functions for working with iterators.
    - collections.Counter: Used for creating and working with counters (e.g., counting occurrences of elements).
    - matplotlib.pyplot (plt): Used for creating visualizations like charts and plots.

11. **Machine Learning Libraries:**

    - sklearn.metrics: Provides various metrics for machine learning model evaluation, including accuracy, ROC-AUC score, confusion matrix, classification report, and F1-score.
    - imblearn.over_sampling.RandomOverSampler: This is used for addressing class imbalance in datasets by oversampling the minority class.

12. **Libraries for Working with Transformers:**

    - accelerate: Already installed as mentioned earlier, used for mixed-precision and distributed training.
    - evaluate: Already installed, used for model evaluation metrics.
    - datasets: Already installed, used for loading and processing datasets.
    - transformers: Already installed, provides pre-trained models and functionalities for working with transformers. Here, specific functionalities include:
        - TrainingArguments: Defines arguments and hyperparameters for training the model.
        - Trainer: A class that handles the training loop and processes the dataset.
        - ViTImageProcessor: A pre-processing class specifically designed for processing images for the ViT (Vision Transformer) model.
        - ViTForImageClassification: A pre-trained ViT model for image classification tasks.
        - DefaultDataCollator: A function that prepares batches of data for training the model.

13. **Other Libraries:**

    - torch: The core deep learning library used for building and running the model.
    - torch.utils.data.DataLoader: This class helps create iterators (loaders) over datasets.

- torchvision.transforms: Provides various image transformation functions for data augmentation. Here, specific transformations used include:
  - CenterCrop: Crops an image from the center.
  - Compose: Composes several transforms into a single pipeline.
  - Normalize: Normalizes pixel values in an image.
  - RandomRotation: Rotates an image randomly.
  - RandomResizedCrop: Randomly resizes and crops an image.
  - RandomHorizontalFlip: Flips an image horizontally with a random chance.
  - RandomAdjustSharpness: Randomly adjusts the sharpness of an image.
  - Resize: Resizes an image to a specific size.
  - ToTensor: Converts a PIL image to a PyTorch tensor.
- PIL: Python Imaging Library, used for working with images.
- pathlib.Path: Provides path manipulation functionalities.
- tqdm: A library that provides progress bars for iterators.
- os: Provides functionalities for interacting with the operating system.
- huggingface_hub: This library allows you to interact with the Hugging Face Hub, a platform for sharing and discovering machine learning models and datasets.
  - notebook_login: Logs into the Hugging Face Hub from a Jupyter Notebook environment (potentially).
  - HfApi: Provides an API for interacting with the Hugging Face Hub.

Overall, this code snippet sets up the environment for training a ViT model for image classification. It installs necessary libraries, imports functionalities for data manipulation, model training, evaluation, and potentially interacts with the Hugging Face Hub.

CHAPTER 4

CHAPTER 4

PROJECT DESCRIPTION

4.1 WORKING FLOW DIAGRAM

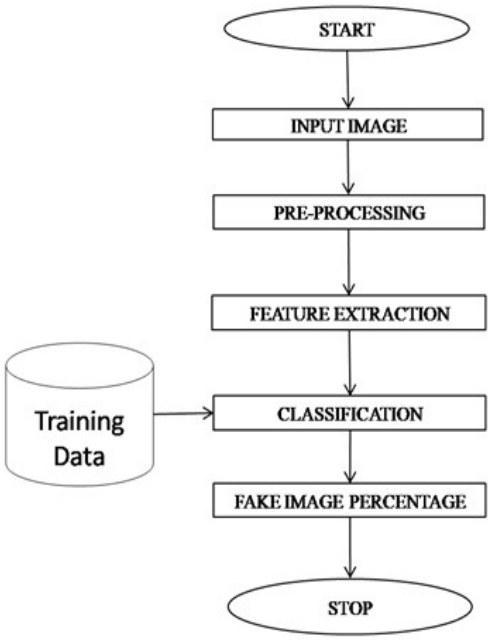


Fig 4.1 Flow diagram

1. Initialization: The process commences with a "START" symbol, signifying the beginning of the workflow.

2. Data Acquisition: The subsequent step involves acquiring input data, represented by the block labeled "INPUT IMAGE." This image presumably serves as the foundation for the subsequent stages.

3. Preprocessing: Prior to feature extraction, the image undergoes preprocessing within the designated block. This stage likely involves tasks such as:

- **Image Resizing:** Rescaling the image to a standardized size for consistent processing.
- **Normalization:** Adjusting the image's intensity values to fall within a specific range, facilitating effective feature extraction.
- **Noise Reduction:** Eliminating unwanted artifacts or distortions that might impede accurate feature extraction.

4. **Feature Extraction:** Once preprocessed, the image is subjected to feature extraction within the designated block. This crucial step involves identifying and extracting salient characteristics from the image that will be used for subsequent analysis. Common feature extraction techniques include:

- **Edge Detection:** Identifying the boundaries of objects within the image, providing valuable information for shape recognition.
- **Color Histograms:** Analyzing the distribution of colors within the image, potentially revealing dominant hues or color patterns.
- **Texture Analysis:** Examining the surface characteristics of the image, such as roughness or smoothness, aiding in material classification or medical image interpretation.

5. Training and Classification: The extracted features then serve as the basis for training a classification model. This model, likely housed within the "Training" and "Classification" blocks, is exposed to numerous labeled examples, enabling it to learn associations between specific features and corresponding classifications.

6. Fake Image Detection: Interestingly, the flowchart incorporates a "FAKE IMAGE PERCENTAGE" block. This suggests the potential presence of a mechanism to detect and quantify the proportion of inauthentic images within the training data. Identifying synthetic or manipulated images is crucial for ensuring the model's robustness and preventing biased learning outcomes

7. Termination: Finally, the process concludes with a "STOP" symbol, indicating the completion of the workflow. The specific criteria for termination remain unclear from the image but could be based on achieving a desired level of classification accuracy, processing a predetermined number of images, or encountering another pre-defined stopping condition.

4.2 PYTHON LIBRARIES USE IN PROJECT:

The libraries used in the provided code are:

- **Tensor Flow:** It is a powerful open-source software library developed by Google for numerical computation and large-scale machine learning. It's particularly well-suited for building and training deep neural networks, but it can also be used for a variety of other machine learning tasks**.**
- **Keras:** It is a powerful and versatile tool that can be used to build a wide range of deep learning models. It is a good choice for developers who want to get started with deep learning or who want to build complex models quickly and easily.
- **Delib:** De-lib, in the context of deepfake detection, refers to a technique that aims to mitigate the impact of deliberate manipulations or adversarial attacks on deepfake detection systems. It seeks to make these systems more robust and resilient to attempts to fool them.
- **gradio:** Used for creating the user interface to interact with the comment classifier.

Please make sure you have these libraries installed before running the code. You can use `pip install` to install any missing libraries.

4.3 PYTHON:

Python is an interpreter, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse.

The Python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Python is the primary programming language used in the provided code for the deepfake detection.

Here are the key uses of Python in the project:

- **Gradio Interface:** Python, along with the Gradio library, is employed to create a user interface for the comment classifier. The code defines the input and output interfaces using Gradio's Image input and Label components, respectively. Python is also used to define the classify comment function, which performs the classification based on user input.
- **General Programming:** Python serves as the general-purpose programming language for writing the overall logic of the code. It handles tasks such as function definitions, variable assignments, flow control, and interactions with external resources.

Python's simplicity, extensive libraries, and readability make it well-suited for tasks such as data manipulation, preprocessing, machine learning, and creating user interfaces. Its ecosystem of libraries and frameworks provides efficient solutions for various aspects of the comment classifier project.

4.4 TENSOR FLOW:

TensorFlow, a popular open-source library, plays a crucial role in deepfake image detection. Here's a brief overview:

Model Building:

- **Convolutional Neural Networks (CNNs):** The go-to architecture for image analysis, CNNs extract features from images through layered processing. Tensorflow provides efficient tools for constructing and training these networks.
- **Pre-trained models:** Leveraging pre-trained models like VGG16 or ResNet50 can reduce training time and improve accuracy. Tensorflow seamlessly integrates with these models.
- **Custom architectures:** For specific needs, you can build custom CNNs or explore cutting-edge architectures like Siamese networks for deepfake detection. Tensorflow offers flexibility for experimentation.

Data Processing:

- **Data Augmentation:** To improve model generalizability, techniques like flipping, cropping, and color jittering are employed. Tensorflow offers built-in data augmentation tools for efficient processing.
- **Training datasets:** Large, diverse datasets containing real and deepfake images are crucial. Tensorflow can handle large datasets and manage the training process efficiently.

4.5 KERAS:

Keras is a popular deep learning library built on top of TensorFlow, making it a powerful tool for building and training deepfake image detection models. Here's a brief overview of its role in this domain:

1. Model Building:

- **Convolutional Neural Networks (CNNs):** Keras offers various building blocks for constructing CNNs, the backbone of deepfake detection models. These include convolutional layers, pooling layers, activation functions, and more.
- **Pre-trained Models:** Keras provides access to pre-trained models like VGG16, ResNet, and EfficientNet, which can be fine-tuned for deepfake detection tasks. This saves time and computational resources compared to building a model from scratch

2. Data Preprocessing and Augmentation:

- **Image Augmentation:** Keras offers image augmentation techniques like flipping, cropping, and rotation to artificially increase the size and diversity of your training data, improving modelgeneralizability.
- **Data Standardization:** Keras helps normalize and standardize your image data, ensuring all images have similar scales and distributions, leading to better model training.

4.6 DELIB:

Delib, also known as Deep Learning-based Video Forensics, is a powerful approach for deepfake image detection that utilizes deep learning algorithms to analyze features within images and videos. Here's a breakdown of its role in this domain:

1. Identifying Tampering Techniques:

- Delib focuses on identifying specific manipulation techniques commonly used in deepfakes, such as facial swapping, expression synthesis, and body puppeteering.
- It analyzes subtle inconsistencies in lighting, shadows, textures, and motion patterns that may arise from these manipulations.

2. Feature Extraction and Analysis:

- Delib utilizes convolutional neural networks (CNNs) trained on large datasets of real and manipulated images/videos.
- These CNNs extract relevant features from the input data, such as edges, textures, and optical flow, which are then analyzed for anomalies indicative of deepfakes.

3. Anomaly Detection and Classification:

- Delib algorithms employ anomaly detection techniques to identify deviations from the learned patterns of real images/videos

- Based on these deviations, the model classifies the input as either real or manipulated (deepfake).

4. Advantages of Delib:

- **High Accuracy:** Delib models can achieve high accuracy in deepfake detection, often outperforming traditional forensic methods.
- **Generality:** These models can be trained to detect a variety of deepfake manipulation techniques, making them adaptable to future advancements.
- **Automation:** Delib offers automated detection, significantly reducing the time and expertise required compared to manual forensic analysis.

4.7 HUGGING FACE

Streamlined Deployment Solutions with Hugging Face

Hugging Face empowers you to seamlessly transition your machine learning models from development to production. They offer two primary deployment methods to fulfill your specific needs:

- **Hugging Face Inference Endpoints:** This managed service enables rapid deployment with minimal configuration. Simply select your model, choose a cloud provider and region, and define the desired security level. Inference Endpoints handle infrastructure provisioning, autoscaling for efficient resource utilization, and robust security features, including secure offline deployments with SOC 2 Type 2 compliance. This option is ideal for production-ready deployments requiring minimal setup time.
- **Bring Your Own Cloud (BYOC):** For organizations with established cloud infrastructure or specific compliance requirements, Hugging Face offers BYOC deployment. This approach grants you granular control over the deployment environment, allowing integration with your existing cloud platform (e.g., Amazon SageMaker). While offering greater flexibility, BYOC deployments necessitate independent infrastructure management. Hugging Face provides comprehensive documentation and tutorials to facilitate deployments on various cloud platforms.

Beyond the Core: Flexible API Integrations

Hugging Face's capabilities extend beyond core deployment functionalities. They integrate seamlessly with FastAPI, a popular Python framework for building web APIs. This enables the creation of custom APIs tailored to your specific model. Once developed, the API can be deployed using containerization tools like Docker, fostering a highly customizable and scalable deployment approach.

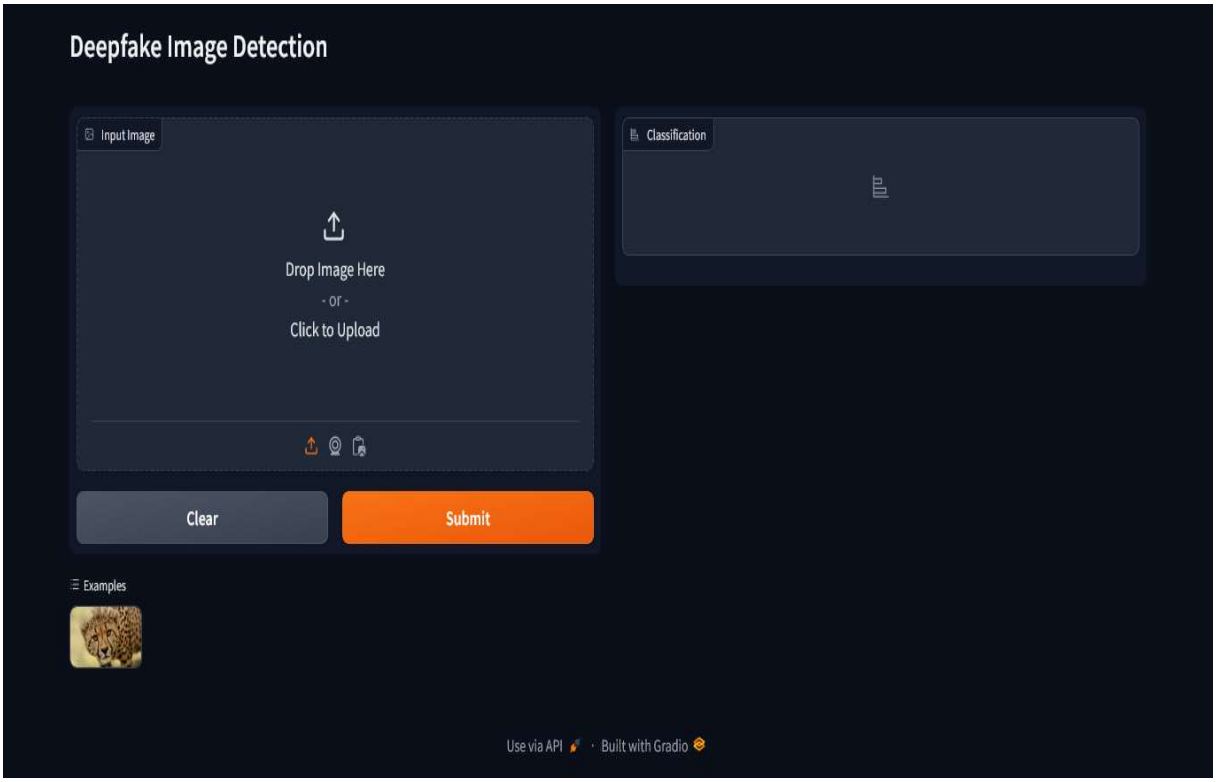

Deepfake Image Detection
Input Image
Drop Image Here
- or -
Click to Upload
Clear
Submit
Classification
Examples
Use via API
Built with Gradio

Fig. 4.2. GUI

CHAPTER 5

CHAPTER 5
CODING
Import Libraries, Load and Transform Data

```
# Install necessary Python packages using pip
!pip install -U -q evaluate transformers datasets>=2.14.5 accelerate>=0.27 2>/dev/null
# Importing necessary libraries and modules
# Import the 'warnings' module for handling warnings
import warnings
# Ignore warnings during execution
warnings.filterwarnings("ignore")
# Import the 'gc' module for garbage collection
import gc
# Import NumPy for numerical operations
import numpy as np
# Import Pandas for data manipulation
import pandas as pd
# Import 'itertools' for iterators and looping
import itertools
# Import 'Counter' for counting elements
from collections import Counter
# Import Matplotlib for data visualization
import matplotlib.pyplot as plt
# Import various metrics from scikit-learn
from sklearn.metrics import (
    accuracy_score,
```

```
    roc_auc_score,
    confusion_matrix,
    classification_report,
    f1_score
)
# Import custom modules and classes
# import RandomOverSampler
from imblearn.over_sampling import RandomOverSampler
import accelerate
import evaluate
from datasets import Dataset, Image, ClassLabel
from transformers import (
    TrainingArguments,
    Trainer,
    ViTImageProcessor,
    ViTForImageClassification,
    DefaultDataCollator
)
import torch
from torch.utils.data import DataLoader
from torchvision.transforms import (
    CenterCrop,
    Compose,
    Normalize,
    RandomRotation,
    RandomResizedCrop,
```

```
    RandomHorizontalFlip,
    RandomAdjustSharpness,
    Resize,
    ToTensor  )
# Import the necessary module from the Python Imaging Library (PIL).
from PIL import ImageFile
# Enable the option to load truncated images.
ImageFile.LOAD_TRUNCATED_IMAGES = True
# use https://huggingface.co/docs/datasets/image_load for reference
# Import necessary libraries
image_dict = {}
# Define the list of file names
from pathlib import Path
from tqdm import tqdm
import os
# Initialize empty lists to store file names and labels
file_names = [ ]
labels = [ ]
# Iterate through all image files in the specified directory
for file in sorted((Path('/kaggle/input/deepfake-and-real-
images/Dataset/').glob('*/*/*.*'))):
    label = str(file).split('/')[-2]
    labels.append(label)
    file_names.append(str(file))
print(len(file_names), len(labels))
df = pd.DataFrame.from_dict({"image": file_names, "label": labels})
```

print(df.shape)

190335 190335

(190335, 2)

df.head()

	image	label
0	/kaggle/input/deepfake-and-real-images/Dataset...	Fake
1	/kaggle/input/deepfake-and-real-images/Dataset...	Fake
2	/kaggle/input/deepfake-and-real-images/Dataset...	Fake
3	/kaggle/input/deepfake-and-real-images/Dataset...	Fake
4	/kaggle/input/deepfake-and-real-images/Dataset...	Fake

df['label'].unique()

array(['Fake', 'Real'], dtype=object)

'y' contains the target variable (label) we want to predict

y = df[['label']]

Drop the 'label' column from the DataFrame 'df' to separate features from the target variable

df = df.drop(['label'], axis=1)

Create a RandomOverSampler object with a specified random seed (random_state=83)

Use the RandomOverSampler to resample the dataset by oversampling the minority class

ros = RandomOverSampler(random_state=83)

'df' contains the feature data, and 'y_resampled' will contain the resampled target variable

df, y_resampled = ros.fit_resample(df, y)

Delete the original 'y' variable to save memory as it's no longer needed

del y

Add the resampled target variable 'y_resampled' as a new 'label' column in the DataFrame 'df'

df['label'] = y_resampled

Delete the 'y_resampled' variable to save memory as it's no longer needed

```
del y_resampled
# Perform garbage collection to free up memory used by discarded variables
gc.collect()
print(df.shape)
```

(190402, 2)

```
# Create a dataset from a Pandas DataFrame.
dataset = Dataset.from_pandas(df).cast_column("image", Image())
# Display the first image in the dataset
dataset[0]["image"]
# The slicing syntax [:5] selects elements from the beginning up to (but not including) the 5th element.
labels_subset = labels[:5]
# Printing the subset of labels to inspect the content.
print(labels_subset)
```

['Fake', 'Fake', 'Fake', 'Fake', 'Fake']

```
# Create a list of unique labels by converting 'labels' to a set and then back to a list
labels_list = ['Real', 'Fake'] # list(set(labels))
# Initialize empty dictionaries to map labels to IDs and vice versa
label2id, id2label = dict(), dict()
# Iterate over the unique labels and assign each label an ID, and vice versa
for i, label in enumerate(labels_list):
    label2id[label] = i  # Map the label to its corresponding ID
    id2label[i] = label  # Map the ID to its corresponding label
# Print the resulting dictionaries for reference
print("Mapping of IDs to Labels:", id2label, '\n')
```

```
print("Mapping of Labels to IDs:", label2id)
```

```
Mapping of IDs to Labels: {0: 'Real', 1: 'Fake'}
Mapping of Labels to IDs: {'Real': 0, 'Fake': 1}
```

```
# Creating classlabels to match labels to IDs
ClassLabels = ClassLabel(num_classes=len(labels_list), names=labels_list)
# Mapping labels to IDs
def map_label2id(example):
    example['label'] = ClassLabels.str2int(example['label'])
    return example
dataset = dataset.map(map_label2id, batched=True)
# Casting label column to ClassLabel Object
dataset = dataset.cast_column('label', ClassLabels)
# Splitting the dataset into training and testing sets using an 60-40 split ratio.
dataset = dataset.train_test_split(test_size=0.4, shuffle=True,
stratify_by_column="label")
# Extracting the training data from the split dataset.
train_data = dataset['train']
# Extracting the testing data from the split dataset.
test_data = dataset['test']
# Define the pre-trained ViT model string
model_str = "dima806/deepfake_vs_real_image_detection" # 'google/vit-base-patch16-
224-in21k'
# Create a processor for ViT model input from the pre-trained model
processor = ViTImageProcessor.from_pretrained(model_str)
# Retrieve the image mean and standard deviation used for normalization
image_mean, image_std = processor.image_mean, processor.image_std
```

```
# Get the size (height) of the ViT model's input images
size = processor.size["height"]
print("Size: ", size)
# Define a normalization transformation for the input images
normalize = Normalize(mean=image_mean, std=image_std)
# Define a set of transformations for training data
_train_transforms = Compose(
  [   Resize((size, size)),
      RandomRotation(90),
      RandomAdjustSharpness(2),
      ToTensor(),
      normalize    ]
)
# Define a set of transformations for validation data
_val_transforms = Compose(
   [
      Resize((size, size)),
      ToTensor(),
      Normalize    ]
)
# Define a function to apply training transformations to a batch of examples
def train_transforms(examples):
   examples['pixel_values'] = [_train_transforms(image.convert("RGB")) for image in examples['image']]
   return examples
# Define a function to apply validation transformations to a batch of examples
```

```
def val_transforms(examples):
    examples['pixel_values'] = [_val_transforms(image.convert("RGB")) for image in examples['image']]
    return examples
```

Size: 224

```
# Set the transforms for the training data
train_data.set_transform(train_transforms)
# Set the transforms for the test/validation data
test_data.set_transform(val_transforms)
# Define a collate function that prepares batched data for model training.
def collate_fn(examples):
    # Stack the pixel values from individual examples into a single tensor.
    pixel_values = torch.stack([example["pixel_values"] for example in examples])
    # Convert the label strings in examples to corresponding numeric IDs using label2id dictionary.
    labels = torch.tensor([example['label'] for example in examples])
    # Return a dictionary containing the batched pixel values and labels.
    return {"pixel_values": pixel_values, "labels": labels}
```

Load, Train, and Evaluate Model

```
# Create a ViTForImageClassification model from a pretrained checkpoint with a specified number of output labels.

model = ViTForImageClassification.from_pretrained(model_str, num_labels=len(labels_list))

# Configure the mapping of class labels to their corresponding indices for later reference.

model.config.id2label = id2label

model.config.label2id = label2id

# Calculate and print the number of trainable parameters in millions for the model.

print(model.num_parameters(only_trainable=True) / 1e6)
```

85.800194

```
# Load the accuracy metric from a module named 'evaluate'

accuracy = evaluate.load("accuracy")

# Define a function 'compute_metrics' to calculate evaluation metrics

def compute_metrics(eval_pred):

# Extract model predictions from the evaluation prediction object

    predictions = eval_pred.predictions

# Extract true labels from the evaluation prediction object

    label_ids = eval_pred.label_ids

# Calculate accuracy using the loaded accuracy metric

probability (argmax)

    predicted_labels = predictions.argmax(axis=1)

# Calculate accuracy score by comparing predicted labels to true labels

    acc_score = accuracy.compute(predictions=predicted_labels, references=label_ids)['accuracy']
```

```
# Return the computed accuracy as a dictionary with the key "accuracy"
    return {
        "accuracy": acc_score
    }
# Define the name of the evaluation metric to be used during training and evaluation.
metric_name = "accuracy"
# Define the name of the model, which will be used to create a directory for saving model checkpoints and outputs.
model_name = "deepfake_vs_real_image_detection"
# Define the number of training epochs for the model.
num_train_epochs = 2
# Create an instance of TrainingArguments to configure training settings.
args = TrainingArguments(
# Specify the directory where model checkpoints and outputs will be saved.
    output_dir=model_name,
# Specify the directory where training logs will be stored.
    logging_dir='./logs',
# Define the evaluation strategy, which is performed at the end of each epoch.
    evaluation_strategy="epoch",
# Set the learning rate for the optimizer.
    learning_rate=1e-6,
# Define the batch size for training on each device.
    per_device_train_batch_size=32,
# Define the batch size for evaluation on each device.
    per_device_eval_batch_size=8,
# Specify the total number of training epochs.
```

```
    num_train_epochs=num_train_epochs,
# Apply weight decay to prevent overfitting.
    weight_decay=0.02,
# Set the number of warm-up steps for the learning rate scheduler.
    warmup_steps=50,
# Disable the removal of unused columns from the dataset.
    remove_unused_columns=False,
# Define the strategy for saving model checkpoints (per epoch in this case).
    save_strategy='epoch',
# Load the best model at the end of training.
    load_best_model_at_end=True,
# Limit the total number of saved checkpoints to save space.
    save_total_limit=1,
# Specify that training progress should not be reported.
    report_to="none"
)
# Create a Trainer instance for fine-tuning a language model.
trainer = Trainer(
    model,
    args,
    train_dataset=train_data,
    eval_dataset=test_data,
    data_collator=collate_fn,
    compute_metrics=compute_metrics,
    tokenizer=processor,
)
```

Evaluate the pre-training model's performance on a test dataset.

This function calculates various metrics such as accuracy, loss, etc., to assess how well the model is performing on unseen data.

trainer.evaluate()

{'eval_loss': 0.02272443287074566,
'eval_accuracy': 0.9926865456073318,
'eval_runtime': 1098.4332,
'eval_samples_per_second': 69.336,
'eval_steps_per_second': 8.668}

Start training the model using the trainer object.

trainer.train()

Epoch	Training Loss	Validation Loss	Accuracy
1	0.060300	0.023107	0.992726
2	0.049700	0.023382	0.992713

TrainOutput(global_step=7142, training_loss=0.055540765243162794, metrics={'train_runtime': 7114.6852, 'train_samples_per_second': 32.114, 'train_steps_per_second': 1.004, 'total_flos': 1.770552477112121e+19, 'train_loss': 0.055540765243162794, 'epoch': 2.0})

This function computes various evaluation metrics like accuracy, loss, etc. and provides insights into how well the model is performing.

trainer.evaluate()

{'eval_loss': 0.02310723066329956,
'eval_accuracy': 0.9927259358464305,
'eval_runtime': 772.9666,
'eval_samples_per_second': 98.531,
'eval_steps_per_second': 12.317,

'epoch': 2.0}

```
# Use the trained 'trainer' to make predictions on the 'test_data'.

outputs = trainer.predict(test_data)

# Print the metrics obtained from the prediction outputs.

print(outputs.metrics)
```

{'test_loss': 0.02310723066329956, 'test_accuracy': 0.9927259358464305,
'test_runtime': 781.6526, 'test_samples_per_second': 97.436,
'test_steps_per_second': 12.181}

```
# Extract the true labels from the model outputs

y_true = outputs.label_ids

# Predict the labels by selecting the class with the highest probability

y_pred = outputs.predictions.argmax(1)

# Define a function to plot a confusion matrix

def plot_confusion_matrix(cm, classes, title='Confusion Matrix', cmap=plt.cm.Blues,
figsize=(10, 8)):

# Create a figure with a specified size

    plt.figure(figsize=figsize)

# Display the confusion matrix as an image with a colormap

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

    plt.title(title)

    plt.colorbar()

# Define tick marks and labels for the classes on the axes

    tick_marks = np.arange(len(classes))

    plt.xticks(tick_marks, classes, rotation=90)

    plt.yticks(tick_marks, classes)

    fmt = '.0f'

# Add text annotations to the plot indicating the values in the cells
```

```
    thresh = cm.max() / 2.0
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt), horizontalalignment="center", color="white" if cm[i, j] > thresh else "black")
# Label the axes
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
# Ensure the plot layout is tight
    plt.tight_layout()
# Display the plot
    plt.show()
# Calculate accuracy and F1 score
accuracy = accuracy_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred, average='macro')
# Display accuracy and F1 score
print(f"Accuracy: {accuracy:.4f}")
print(f"F1 Score: {f1:.4f}")
# Get the confusion matrix if there are a small number of labels
if len(labels_list) <= 150:
# Compute the confusion matrix
    cm = confusion_matrix(y_true, y_pred)
# Plot the confusion matrix using the defined function
    plot_confusion_matrix(cm, labels_list, figsize=(8, 6))
```

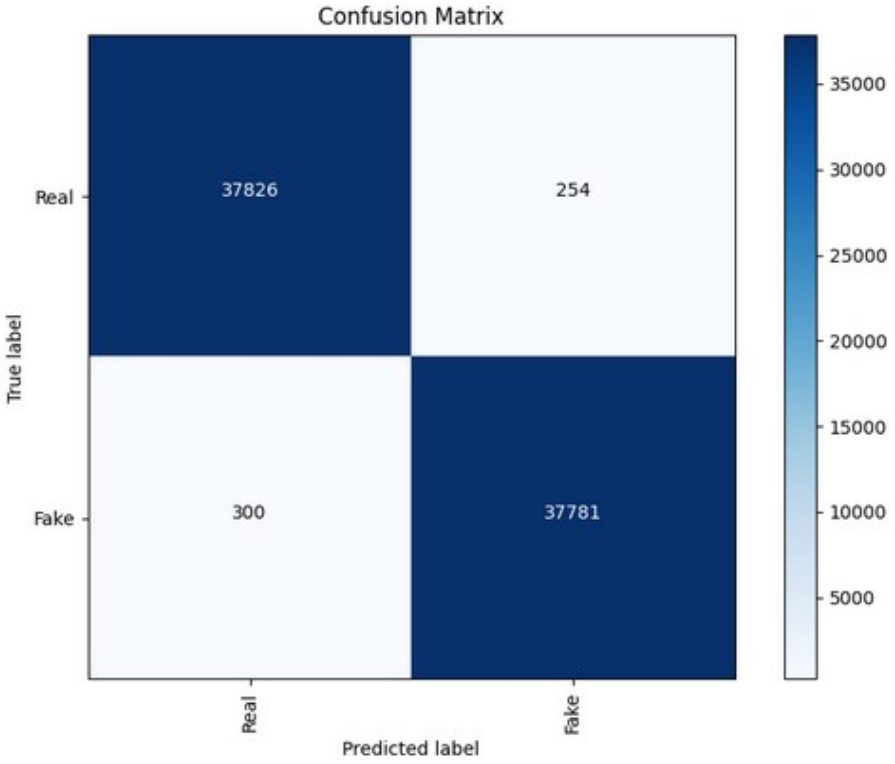
```
print()

print("Classification report:")

print()

print(classification_report(y_true, y_pred, target_names=labels_list, digits=4))
```

Accuracy: 0.9927
F1 Score: 0.9927



Classification report:

	precision	Recall	f1-score	support
Real	0.9921	0.9933	0.9927	38080
Fake	0.9933	0.9921	0.9927	38081
accuracy			0.9927	76161
macro avg	0.9927	0.9927	0.9927	76161
weighted avg	0.9927	0.9927	0.9927	76161

```
# Save the trained model

trainer.save_model()

# Import the 'pipeline' function from the 'transformers' library.

from transformers import pipeline

# Create a pipeline for image classification tasks.

# You need to specify the 'model_name' and the 'device' to use for inference.

# - 'model_name': The name of the pre-trained model to be used for image
classification.

pipe = pipeline('image-classification', model=model_name, device=0)

# Accessing an image from the 'test_data' dataset using index 1.

image = test_data[1]["image"]

# Displaying the 'image' variable.

image
```

```
# Apply the 'pipe' function to process the 'image' variable.

pipe(image)
```

```
[{'label': 'Fake', 'score': 0.9995982050895691},
{'label': 'Real', 'score': 0.0004017664468847215}]
```

```
# This line of code accesses the "label" attribute of a specific element in the test_data
list.

id2label[test_data[1]["label"]]
```

```
'Fake'
```

Send Model to Hugging face

```
# Import the necessary module to interact with the Hugging Face Hub.
from huggingface_hub import notebook_login
# Perform a login to the Hugging Face Hub.
notebook_login()
# Import the HfApi class from the huggingface_hub library.
from huggingface_hub import HfApi
# Create an instance of the HfApi class.
api = HfApi()
repo_id = f"dima806/{model_name}"
try:
# Attempt to create a new repository on the Hugging Face Model Hub using the specified repo_id.
    api.create_repo(repo_id)
# If the repository creation is successful, print a message indicating that the repository was created.
    print(f"Repo {repo_id} created")
except:
# If an exception is raised, print a message indicating that the repository already exists.
    print(f"Repo {repo_id} already exists")
```

Repo dima806/deepfake_vs_real_image_detection already exists

```
# Uploading a folder to the Hugging Face Model Hub
api.upload_folder(
# The path to the folder to be uploaded
    folder_path=model_name,
```

```
# The path where the folder will be stored in the repository
    path_in_repo=".",
# The ID of the repository where the folder will be uploaded
    repo_id=repo_id,
# The type of the repository (in this case, a model repository)
    repo_type="model",
    revision="main"
)
```

CHAPTER 6

CHAPTER 6
TESTING

Deepfakes, where faces are seamlessly swapped or expressions manipulated, pose a growing threat to online trust. To combat this, researchers are developing deepfake detection methods, but ensuring their reliability requires rigorous testing. Let's delve deeper into how deepfake image detection is tested.

Building the Foundation: Datasets

The cornerstone of testing lies in the data used to train and evaluate the detection models. Deepfake detection models are trained on datasets meticulously curated to include both real and deepfake images. Here are some commonly used datasets:

- **FaceForensics++**: This popular dataset boasts high-quality deepfakes generated using different techniques, allowing researchers to test a model's ability to generalize across manipulation methods.
- **CelebA**: This dataset contains a vast collection of celebrity faces, often used for training facial recognition models. While not specifically designed for deepfakes, it can be repurposed to train models to recognize the subtle inconsistencies deepfakes can introduce into facial features.
- **FFHQ (Flickr-Faces-High-Quality)**: This dataset complements CelebA by providing high-resolution celebrity portraits. The combination allows researchers to train models on a wider range of facial variations, improving their ability to detect deepfakes that might use less common facial structures.

Unveiling the Techniques: Binary Classification vs. Anomaly Detection

There are two main approaches to testing deepfake detection, each with its own strengths:

- **Binary Classification:** This is the most prevalent approach. The model is trained on labelled images (real or deepfake) and learns to categorize new, unseen images into these two classes. The success of this method hinges on the quality and diversity of the training data.
- **Anomaly Detection:** Here, the model takes a different approach. Instead of learning specific deepfake characteristics, it builds a comprehensive understanding of what a real image "looks like" based on the training data. Any image that deviates significantly from this learned norm is then flagged as a potential deepfake. This method can be advantageous when dealing with entirely new deepfake creation techniques, as it doesn't rely on identifying specific manipulation artifacts.

Evaluating Performance: Going Beyond Accuracy

Once a model is trained, we need robust metrics to assess its effectiveness. Here are some key metrics used to evaluate deepfake detection models:

- **Accuracy:** This is the most common metric, representing the percentage of images correctly classified as real or fake. While a high accuracy is desirable, it doesn't tell the whole story.
- **Precision:** This metric focuses on the quality of the model's positive detections. It tells us what proportion of images identified as deepfakes are actually deepfakes. A high precision is crucial to avoid flagging real images as manipulated.
- **Recall:** This metric looks at the model's ability to catch all deepfakes. It tells us what proportion of actual deepfakes were correctly identified. A high recall is essential to minimize the chances of deepfakes slipping through the cracks.

Confronting the Real World: Challenges and Countermeasures

Deepfake testing needs to extend beyond controlled lab settings and consider the complexities of the real world, where deepfakes are encountered:

- **Compression Artifacts:** Deepfakes are often compressed before being shared online, introducing artifacts like blurring or reduced quality. These artifacts can interfere with the model's ability to detect subtle inconsistencies indicative of manipulation.
- **Low Resolution Images:** Deepfakes may be deliberately created at low resolutions to make detection harder. Deepfake detection models need to be robust enough to handle these low-resolution scenarios where key details might be obscured.
- **Adversarial Attacks:** Malicious actors might try to evade detection by manipulating deepfakes specifically to exploit weaknesses in the detection model. This highlights the need for models to be constantly evolving and adapting to new deepfake creation techniques.

Staying Ahead of the Curve: Continuous Improvement Strategies

Researchers are actively developing methods to improve deepfake detection and address real-world challenges:

- **Transfer Learning:** By pre-training models on vast datasets of real images, researchers can leverage the model's ability to learn general features of faces. This knowledge can then be adapted to the specific task of deepfake detection.
- **Multi-modal Analysis:** Some methods go beyond analyzing just the image content. They incorporate physiological signals, such as blinking patterns or subtle inconsistencies in head movement, to strengthen the detection process.
- **Continuous Learning:** As deepfake techniques evolve, so too must detection models. Continuous learning algorithms allow models to be continually updated with new data, enabling them to stay ahead of the curve and adapt to the ever-changing landscape of deepfakes.

By employing robust datasets, evaluation metrics, and considering the challenges of the real world, researchers can ensure their deepfake detection models are effective safeguards against misinformation and online manipulation.

CHAPTER 7

CHAPTER 7
OUTPUT

➢ Detecting Real Face

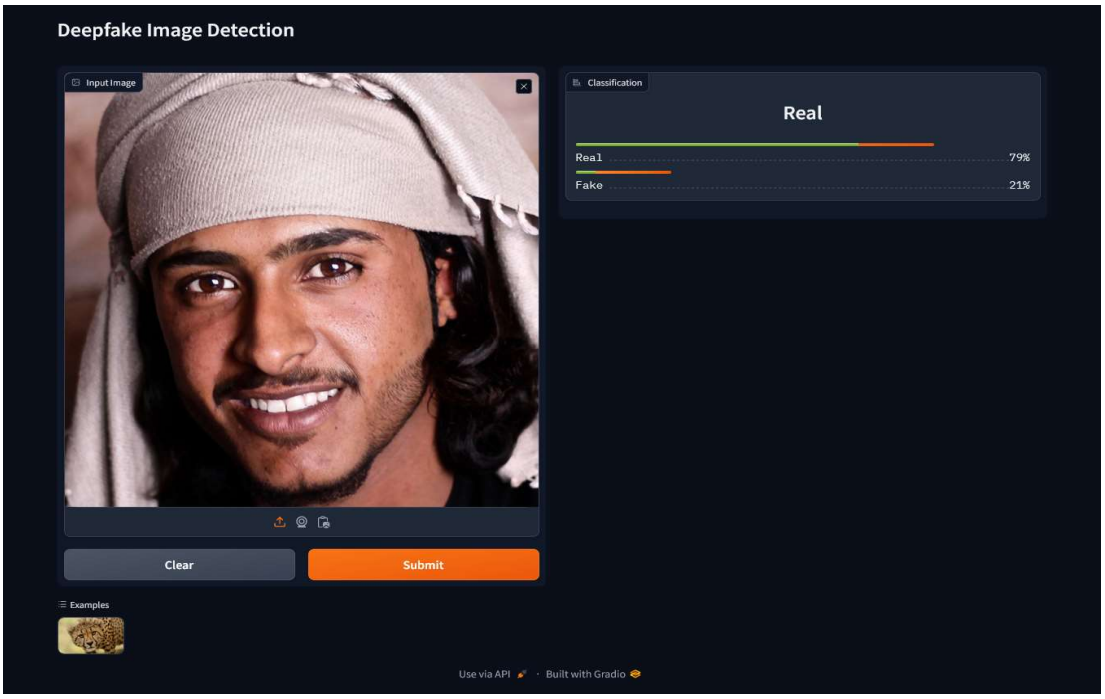


Fig.7.1 (a): Real Face Detection

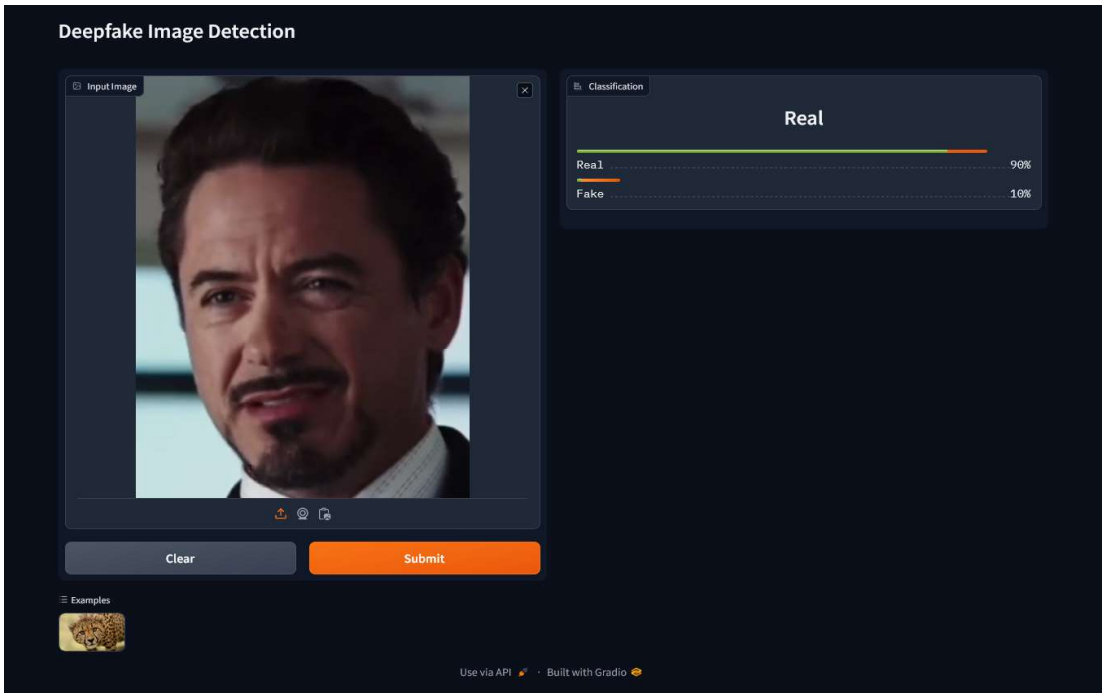


Fig. 7.1(b): Real. Face Detection

➢ Detecting Fake Face

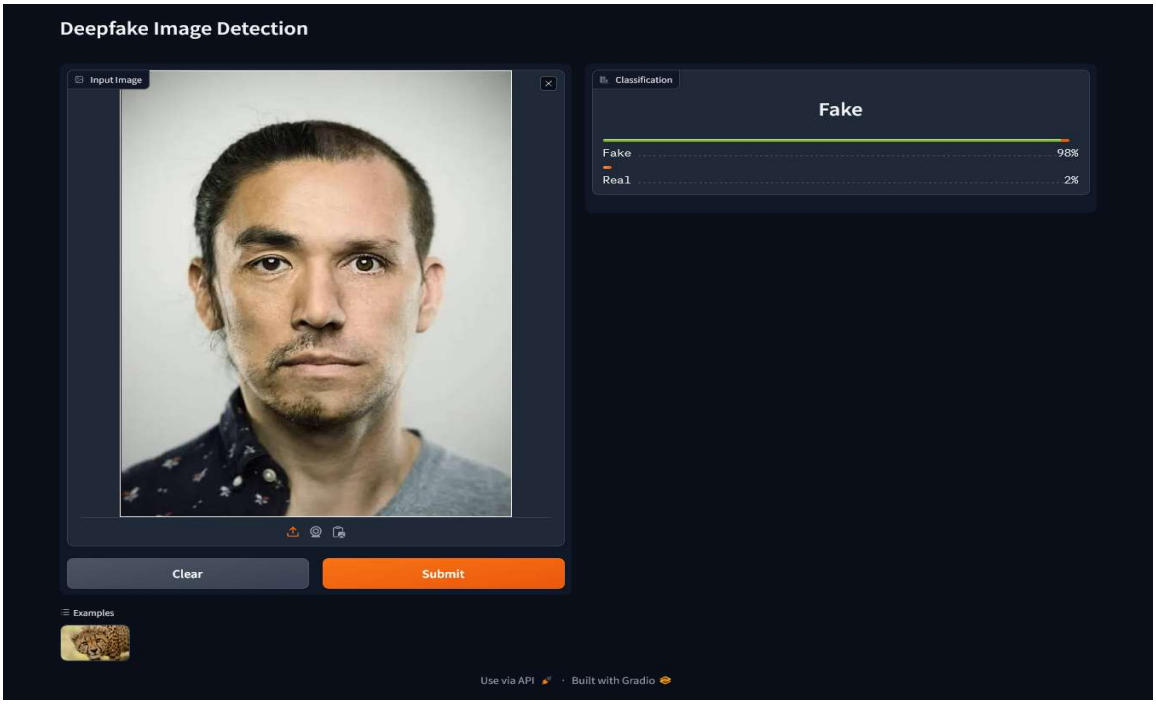


Fig 7.2(a) Fake Face Detection

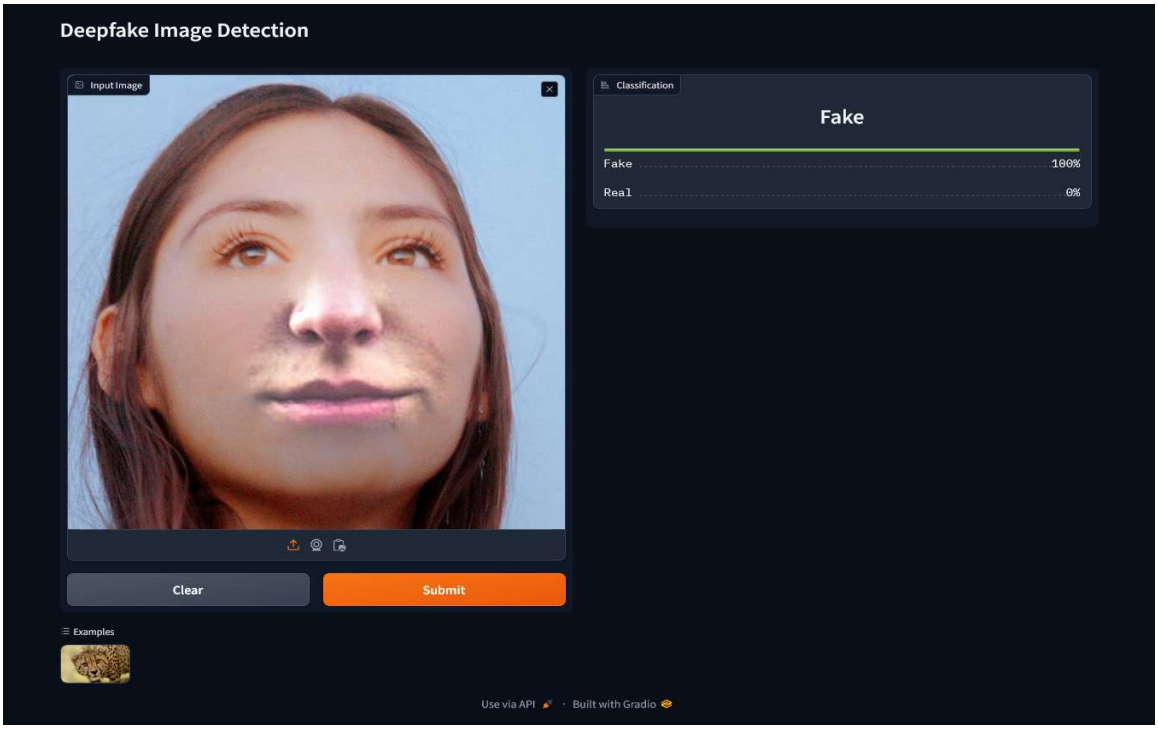


Fig 7.2(b) Fake Face Detection

CHAPTER 8

CHAPTER 8
ADVANTAGES AND DISADVANTAGES

8.1 ADVANTAGES:

Here are some key advantages of deepfake detection image models:

1. **Prevention of Misinformation and Manipulation:**
   Deepfake detection models play a crucial role in preventing the spread of misinformation and manipulation. By identifying and flagging manipulated content, these models help safeguard the integrity of visual information.

2. **Protection Against Fraud:**
   Deepfake detection is essential in protecting against various forms of fraud, including identity theft and financial scams. If deepfake detection is integrated into authentication systems, it can help verify the authenticity of visual information used for identification purposes.

3. **Security in Sensitive Domains:**
   In sensitive domains such as law enforcement, national security, and criminal investigations, deepfake detection models provide an additional layer of security. Ensuring the authenticity of visual evidence is critical for making informed decisions and maintaining the rule of law.

4. **Preservation of Privacy:**
   Deepfake detection contributes to the preservation of individual privacy by identifying and preventing the malicious use of deepfake technology for creating non-consensual, misleading, or harmful content.

5. **Enhanced Media Credibility:**
   Media outlets and platforms can benefit from deepfake detection models to enhance the credibility of the content they share. By ensuring that the visual information they distribute is authentic, these entities can maintain their reputation and credibility.

6. **Regulatory Compliance:**
   In certain industries and regions, there may be regulations and compliance requirements regarding the authenticity of visual content. Deepfake detection models help organizations meet these standards and avoid legal consequences.

7. **Research and Development:**
   The ongoing development of deepfake detection models contributes to advancements in artificial intelligence and computer vision research. The challenges posed by deepfake technology drive innovation in model architectures, training techniques, and detection methodologies.

8.2 DISADVANTAGES:

Here are some key drawbacks:

1. **Adversarial Techniques:**
   Adversarial attacks can be used to generate deepfakes that are specifically designed to deceive detection models. As deepfake generation techniques evolve, there is an ongoing challenge to develop detection models that are robust against increasingly sophisticated adversarial strategies.

2. **Computational Resources:**
   Deepfake detection models often require significant computational resources for training and inference. This can be a limiting factor, particularly for organizations or systems with constrained computing capabilities.

3. **Generalization Challenges:**
   Ensuring that deepfake detection models generalize well to diverse datasets and scenarios is a complex task. Models trained on specific types of deepfakes may struggle to detect novel or unforeseen manipulation techniques.

4. **Privacy Concerns:**
   Deepfake detection involves analyzing and processing visual content, which raises privacy concerns. The deployment of detection models may inadvertently compromise individual privacy, especially in cases where the content being analyzed contains sensitive information.

5. **Limited Training Data:**
   Access to diverse and representative datasets for training deepfake detection models can be limited. The scarcity of comprehensive datasets may hinder the ability of models to generalize well across different types of deepfakes.

6. **Ethical Considerations:**
   There are ethical considerations surrounding the use of deepfake detection technology. Determining when and how to deploy detection models, as well as addressing potential misuse, requires careful ethical considerations to avoid unintended consequences.

7. **Rapid Evolution of Deepfake Technology:**
   The rapid evolution of deepfake generation techniques makes it challenging for detection models to keep up. As deepfake methods become more sophisticated, there is a constant need for researchers to adapt and improve detection strategies.

CHAPTER 9

CHAPTER 9
FUTURE SCOPE

Deepfake detection technology has been a rapidly evolving field of research in recent years. Researchers have been exploring various techniques and tools to detect deepfakes and safeguard the digital space[1]. Some of the future trends in deepfake detection include:

1. **Efficient and robust detection methods**: Researchers are working on developing more efficient and robust deepfake detection methods that can detect even the most advanced deepfakes.
2. **Multimodal data modalities**: Researchers are exploring the use of multimodal data modalities such as audio and video to detect deepfakes.
3. **High-quality datasets**: The availability of high-quality datasets is crucial for the development and evaluation of deepfake detection methods.
4. **Blockchain-based techniques**: Blockchain-based techniques are being explored as a way to enhance the security and reliability of deepfake detection.
5. **Physical and biological signals**: Recent research has shown that utilizing underlying physical and biological signals can make deepfake detection more robust.

Developing accessible and user-friendly deepfake detection tools empowers the public to become active participants in safeguarding the digital space. By equipping individuals with the ability to critically evaluate online content, we can collectively combat the spread of misinformation and foster a more responsible and accountable online environment.

CHAPTER 10

CHPATER 10
CONCLUSION

DeepFake, a new and prominent technology. It covers the basics, benefits, and risks associated with DeepFake, including GAN-based DeepFake applications. In this we also discuss DeepFake detection models. Most existing deep learning-based detection methods are unable to transfer and generalize, indicating that multimedia forensics has not yet reached its peak. Many important organizations and experts are working to improve applied techniques. However, more effort is needed to ensure data integrity, which requires additional protection methods. Additionally, experts predict a new wave of DeepFake propaganda in AI against AI encounters where neither side has an advantage over the other.

The code demonstrates the use of Python libraries like tensor Flow, keras, and for image manipulation functionalities. It also employs the Gradio library to create an intuitive interface for inputting image and obtaining classification results.

REFERENCES

- https://www.kaggle.com/
- https://www.google.com/
- https://www.python.org/
- https://www.youtube.com/
- https://www.huggingface.co/
- Asad Malik, Minoru Kuribayashi, Sani M. Abdullahi And Ahmad Neyaz Khan (2022) DeepFake Detection for Human Face Images and Videos: A Survey.
- Hasin Shahed Shad , Md. Mashfiq Rizvee, Nishat Tasnim Roza, S. M. Ahsanul Hoq, Mohammad Monirujjaman Khan, Arjun Singh, Atef Zaguia and Sami Bourouis (2023) Comparative Analysis of Deepfake Image Detection Method Using Convolutional Neural Network, Computational Intelligence and Neuroscience
- Raza, A.; Munir, K., Almutairi, M. (2022) A Novel Deep Learning Approach for Deepfake Image Detection.Image Detection. Appl. Sci. 2022
- Li, Y., & Lyu, S. (2020). Exposing deepfake videos by leveraging visual artifacts. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition
- Zhixi Cai, Kalin Stefanov, Abhinav Dhall and Munawar Hayat, Do You Really Mean That? Content Driven Audio-Visual Deepfake Dataset and Multimodal Method for Temporal Forgery Localization.
- K. Chugh, P. Gupta, A. Dhall, and R. Subramanian, “Not made for each other- Audio-Visual Dissonance-based Deepfake Detection and Localization,” in ACM MM, 2020, pp. 439–447.
- V. Kazemi and J. Sullivan, "One millisecond face alignment with an ensemble of regression trees," 2014 IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 2014, pp. 1867-1874, doi: 10.1109/CVPR.2014.241.
- P. F. Felzenszwalb, R. B. Girshick, D. McAllester and D. Ramanan, "Object Detection with Discriminatively Trained Part-Based Models," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pp. 1627-1645, Sept. 2010, doi: 10.1109/TPAMI.2009.167.
- Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun,Deep Residual Learning for Image Recognition https://arxiv.org/abs/1512.03385.