Deepfake Classification Model

Ananya Singh, Anoush Pesuna, Aryan Panicker, Kulpunai Kurstanbek kyzy, Mrudav Mehta

SP Jain School of Global Management

Capstone Project

Professor JP Aggarwal, Professor Jafar Hamra

23 April 2024

Abstract

Deepfake technology poses a severe risk to the veracity of anything published online and can cause identity theft and the dissemination of misleading information, among other undesirable outcomes. Researchers and business experts have developed various deepfake detection algorithms intended to detect fabricated content to solve this problem precisely. This study provides a complete overview and analysis of deepfake detection algorithms, focusing on their methodologies, benefits, drawbacks, and performance metrics. This research aims to give stakeholders the knowledge and comprehension necessary to stop the spread of fraudulent media material by carefully examining recent advancements in deepfake detection techniques.

Method

Celeb-DF (v1)

The Celeb-DF (v1) dataset comprises DeepFake and real-world synthesised videos designed to emulate the visual quality typically encountered online. It encompasses 795 DeepFake videos, synthetically generated from 408 original videos sourced from YouTube. This diverse collection includes individuals of various ages, ethnicities, and genders, ensuring a broad representation.

Dataset Structure:

1. *Celeb-real:*
    - Celeb-real contains 158 short video clips featuring 15 actors, each with a designated set of clips.
    - File names follow the format "idX_YYYY," where "idX" refers to the actor's ID and "YYYY" denotes the clip's ID.
    - For instance, "id0_0000" corresponds to Actor [ID 0]'s clip [ID 0], such as a monologue by Aamir Khan on his talk show.
    - The combined size of these 158 videos is 257 megabytes.
2. *YouTube-real:*
    - YouTube-real serves as the test dataset, comprising 250 randomly selected video clips from YouTube.
    - These clips provide additional diversity and real-world representation for model evaluation.
    - The total size of these 250 videos is 573 megabytes.
3. *Celeb-synthesis:*

- Celeb-synthesis contains 795 DeepFake videos generated from the Celeb-real dataset.
- Each of the 15 actors' faces is swapped with those of other actors to produce DeepFake variations.
- File names follow the format "idX_idY_YYYY," representing the actor IDs involved in the face swap and the clip ID.
- For example, "id0_id1_0000" depicts a DeepFake of Actor [0]'s clip [0], swapping faces with Actor [ID 1], such as Aamir Khan's clip with Brad Pitt's face.
- The combined size of these 795 DeepFake videos is 1.2 gigabytes.

The structured organisation and diverse content of the Celeb-DF (v1) dataset provide an invaluable resource for training and evaluating deepfake detection models. They offer a comprehensive understanding of authentic and manipulated video content.

Platforms

After we have the data set, the next step is to find a suitable platform to process this massive amount of data. The following platforms were tested:

1. *Jupyter Notebook:*
    a. Within Jupyter, we ran into code optimisation issues; the local GPU and RAM weren't enough to compute a large dataset. Additionally, we couldn't utilise CUDA for parallel processing for most systems.
2. *AWS Sagemaker and AWS S3:*
    a. AWS required a more significant earning curve and additional financial support for complete deployment. Additionally, Torch requires a standalone AWS developer account.
3. *Google Colab Pro:*
    a. Google Colab was our final choice since it combined Jupyter's ease of use and AWS computation capabilities at a manageable cost and allowed each.

Architecture

A crucial component of our deepfake detection system design that enables us to utilise the power of the deep learning framework Torch entirely is Google Colab. Torch is renowned for its extensive library support and potent GPU acceleration, which make it ideal for complex neural network computations. Researchers and practitioners worldwide can now quickly and economically run Torch-based models without expensive hardware setups thanks to Google Colab.

By installing a Google Drive inside of Google Colab, users may quickly access datasets and model checkpoints stored in the cloud. This integration facilitates resource sharing among

users, which helps to foster teamwork and ease data administration. Our method ensures data persistence and efficient model training and evaluation are facilitated between sessions.

Additionally, using a Google Colab downloader expedites the data acquisition by granting instant access to datasets housed on Google Drive or other cloud storage platforms. This downloader optimises workflow productivity and streamlines the data preparation pipeline by eliminating the need for manual file uploading and downloading. By leveraging these three components—Google Drive for data storage, Google Colab for Torch execution, and a dedicated downloader for data access—our architecture maximises efficiency and speeds up the deepfake detection process.

Libraries

The Python platform requires appropriate libraries to use different algorithms. These libraries were used to create the model:

1. *OpenCV (cv2):* It facilitates operations such as frame extraction, video capture release, and various image manipulations essential for pre-processing.
2. *Dlib:* This library is critical for facial detection and landmark prediction.
3. *face_recognition:* This library is built on top of Dlib and is specifically used for detecting and re-recognising faces within the video frames.

Data Preprocessing

Video File Handling

We start by uploading the paths to all video files stored in the specified directory on Google Drive. For each video, we use the OpenCV cv2 method.VideoCapture for accessing video properties. The total number of frames (frame count) is extracted using cv2.CAP_PROP_FRAME_COUNT. This initial data collection is crucial because it allows us to calculate the following statistics, maximum, minimum and average number of frames in our dataset. This information is important for the next steps of preprocessing.

After analysing the videos, the number of which exceeds a certain number of frames (for example, 300 frames) is filtered out. This step ensures that the dataset contains only videos of acceptable duration, which helps to maintain the efficiency of computational resources used.

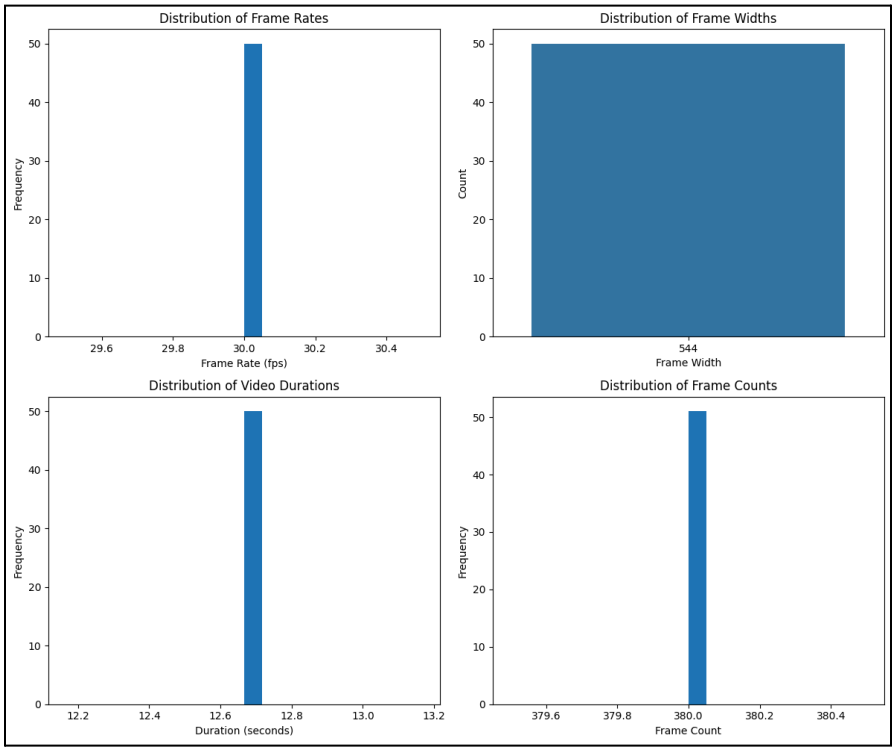

Figure 1 - Distribution of frame rates, widths, counts, and video durations

Face detection with facial landmarks

At this stage, we are using a full set of features, such as the dlib and OpenCV libraries, to improve our deep fake detection process. This includes the use of dlib.get_frontal_face_detector() for reliable face recognition optimised for frontal views, which is important under various conditions, and dlib.shape_predictor with a pre-trained model (shape_predictor_68_face_landmarks.dat) to annotate 68 landmarks on detected faces to provide detailed displays important for identifying manipulations. Each video frame is converted to shades of grey to reduce the computational load, after which the face landmarks are marked with green circles to give the frame a detailed face geometry. Frames are extracted sequentially from video files until they are exhausted, using a function that outputs one frame at a time, optimising memory and resource management. The create_face_videos function processes video track lists, generating new face-oriented videos, checking for previously processed files to avoid redundancy, and storing frames until a limit is reached (for example, 380 frames). The faces in each frame with landmark notes are detected, cropped, and resized to a consistent size (112x112 pixels) before being saved, which ensures uniformity of input data for the detection model. The size of the frames without detected faces is also changed and maintained, which ensures that there is no data loss, which makes our pre-processing steps critical for preparing the dataset for subsequent machine learning tasks.

Our pipeline is carefully designed to enhance the detection model's ability to accurately identify and analyse facial features in video, which play a key role in detecting deep fakes. In addition, resizing and trimming the frames to a consistent size guarantees consistency and data quality, which significantly improves the effectiveness of model training and prediction accuracy.

Figure 2 - Random set of videos taken to be trained

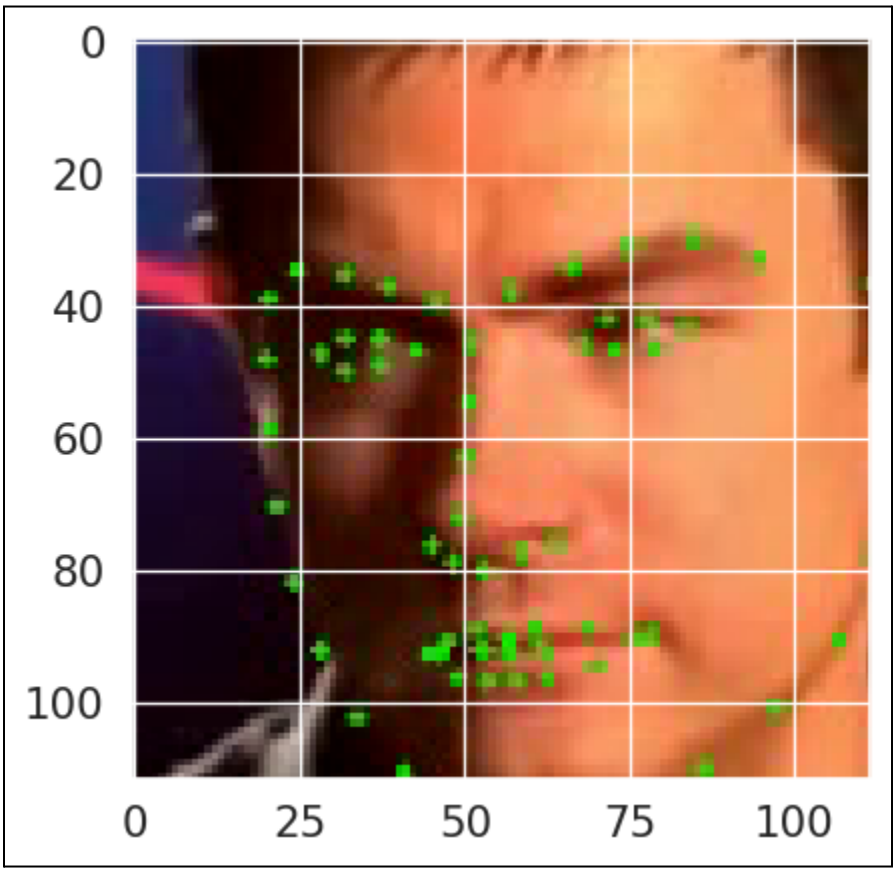

Figure 3 - Recognising facial landmarks

Model Development and Training

The video files stored on Google Drive are accessed randomly using the Python glob and random. Shuffle functions to prevent sequence bias, which is crucial for objective study and generalisation of the model. We use OpenCV cv2.VideoCapture to verify the integrity of each video. This optimises the use of computing resources by only training validated videos and underlies the development of a reliable model for detecting deep fakes.

Data Preparation and Transformation

We first check all videos for distortion or corruption. Once the video is validated, it transforms uniform input into the model. The first transformation involves resizing the frames to ensure consistency of all input data. Then, we apply normalisation to adjust the frame colour values by aligning the pixel intensity distribution to predefined averages and standard deviations. This normalisation is necessary to minimise the model's sensitivity to changes in lighting and colour in different videos.

We stack multiple frames into singular tensors to implement batch processing. This increases processing efficiency and allows you to use the temporal information contained in the sequence of video frames.

To correctly label each video, we created a metadata.csv with the IDs of each video and the labels - "Real" or "Fake". These labels are key to our supervised learning approach, providing the necessary foundation for model learning.

The videos are then split into training and verification datasets in an 80/20 ratio. We have another dataset - ‘youtube.real’ for the final checks.

Custom Dataset Class Implementation

To effectively manage video data within our machine learning platform, we implement a custom video_dataset class in PyTorch - very common to create video/image datasets. This class includes sequentially downloading video files, applying the necessary transformations to each frame, and preparing these frames for batch processing.

Model Architecture and Setup

The chosen model architecture is a mixture of residual networks (ResNets) and long-term, short-term memory (LSTM) modules designed to accurately capture and analyse the spatial and temporal characteristics of video sequences. This hybrid approach effectively combines the advantages of CNNs (convolutional neural networks) and RNNs (recurrent neural networks), making it exceptionally suitable for tasks that require understanding both the details of individual frames and the dynamics between successive frames, which is typical for video processing tasks such as deep fake detection.

Residual Networks (ResNet): ResNet helps solve the vanishing gradient problem that can occur in traditional deep convolution networks by introducing pass-through connections that allow gradients to pass directly through the network. In our model, the subnets are primarily responsible for extracting detailed spatial characteristics from each video frame. These functions form the fundamental building blocks to detect minor inconsistencies or manipulations with image data that may indicate a deepfake.

Long-term Short-term Memory Modules (LSTM): LSTM is a type of RNN that is capable of studying long-term dependencies in data sequences. By integrating LSTM modules into our architecture, the model is able to understand the temporal relationships between

successive frames. This is crucial for detecting deep fakes where anomalies often manifest as irregularities in movement or facial expression over time.

Combining Resets and LSTM: Integrating resets and LSTM allows our model to process and interpret both deep spatial hierarchies and temporal sequences in videos, providing the comprehensive understanding needed to accurately identify a deep fake. This combination provides effective capture and analysis of both sharp and barely noticeable deepfake signatures, whether in a single frame or in a sequence of frames.

Overfitting prevention: To ensure a good generalisation of the model to new, invisible data, rather than memorising the training dataset, drop-down levels are included in the architecture. Dropping out randomly disables some of the neurons in the learning process, which helps prevent the model from becoming overly dependent on any particular set of functions, thereby increasing its reliability and generalisation ability.

Output Layer: A fully connected layer completes the architecture, combining features learned by both ResNet and LSTM networks into final predictions. This level classifies each input video sequence into "real" or "Fake" based on data collected through spatial and temporal analysis.

Training Process and Validation Mechanics

Training our deep fake detection model involves uploading preprocessed video data in batches and calculating losses using a cross-entropy criterion that effectively measures the discrepancy between predicted probabilities and actual labels ("Real" or "Fake"). We use the Adam optimizer because of its effectiveness in setting model parameters through back propagation, improving learning over subsequent epochs. We also ensure that the model does not overfit by ensuring it does not train after the testing accuracy remains the same after 3 epochs. Additionally we also use regularisation parameters.

Results

Based on train data, and our epochs we were able to plot the following results -

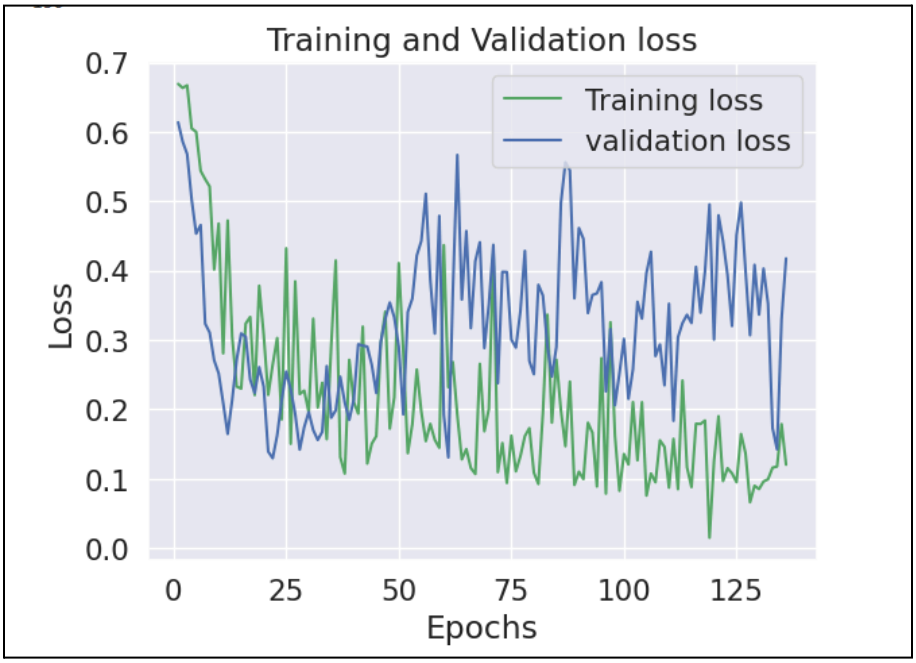

Figure 4 - Training and validation loss

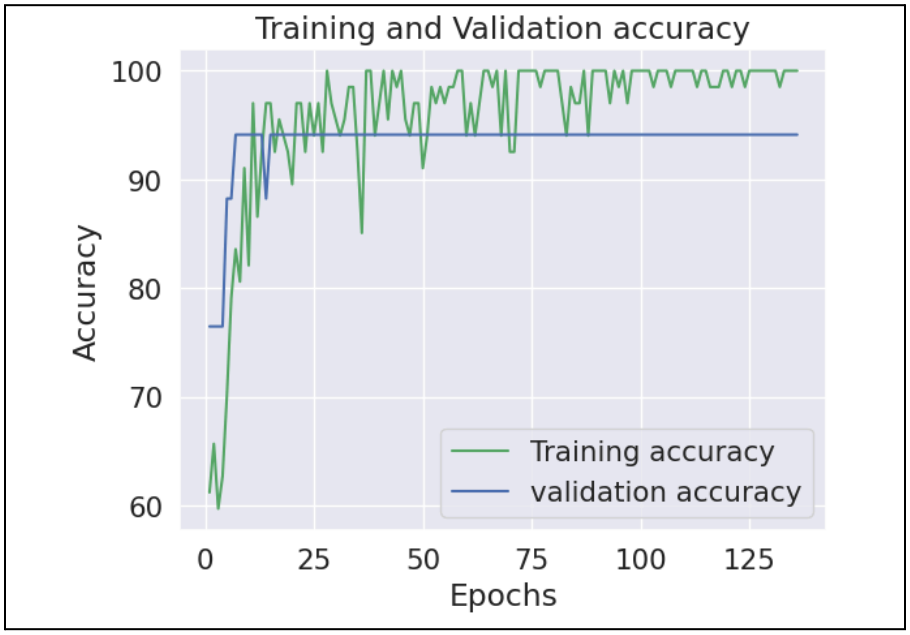

Figure 5 - Training and validation accuracy

We were able to procure a testing accuracy of - **94.11%**

The confusion matrix is as follows -

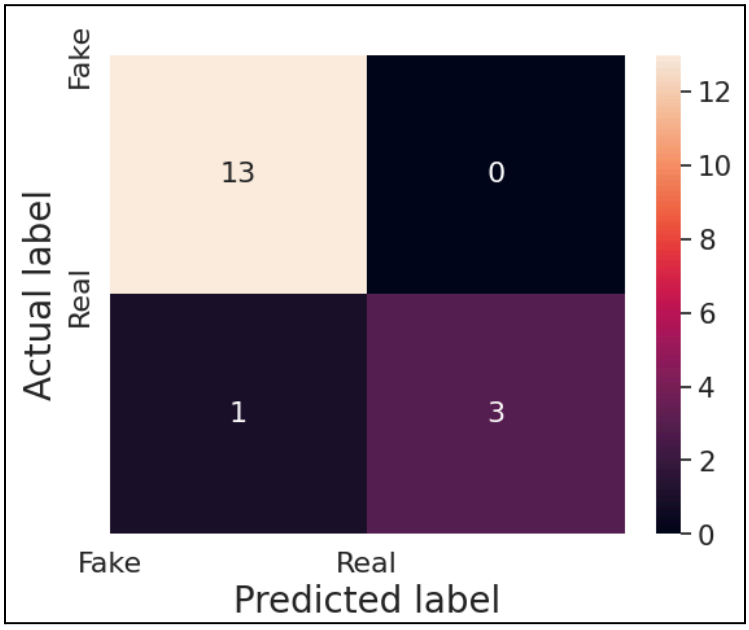

Figure 6 - Confusion matrix of the model

Prediction

The predict function is the core component of deepfake detection. It preprocesses input tensors, runs them through the model, and computes the softmax probability distribution to obtain projected class labels and confidence ratings. Additionally, it generates heatmaps that highlight significant image regions in the forecasts. This function highlights the key components of the deepfake detection system and provides insights into the model's decision-making process.

The system makes predictions using the trained deepfake detection model and the provided dataset. It describes the first image changes for input images, including scaling and normalisation. The dataset is initialised using the validation_dataset class, providing video paths and transformation options. Every video in the dataset is exposed to repeated predictions after the trained model has been loaded. The following expected labels—REAL or FAKE—are then printed for review.

The code segment adds to the validation_dataset class and offers an additional way to process whole frames while generating predictions. This allows for the inclusion of entire frames for evaluation, thereby satisfying scenarios where face cropping would not be permissible.

The final trained and tested model is saved as a .pt file, which allows it to be used on a different machine as we have in our frontend deployment.

Overview of FrontEnd

Overview of Flask

Flask is a lightweight WSGI (Web Server Gateway Interface) web application framework. It's designed to make getting started quick and easy, with the ability to scale up to complex applications. It provides tools, libraries, and technologies that allow you to build a web application. This web application can be some web pages, a blog, a wiki, or go as big as a web-based calendar application or a commercial website.

Structure

`app.py`serves as the backbone of the project, managing the interactions between the front end and the machine learning model. Here's how each part contributes to the application:

1. *Flask App Initialization and Configuration:*

- `UPLOAD_FOLDER`: Directory where uploaded files are stored.
- `ALLOWED_EXTENSIONS`: Restricts uploads to files with an 'mp4' extension to ensure only video files are processed.
- `MAX_CONTENT_LENGTH`: Limits the size of uploaded files to prevent overload.
- `SECRET_KEY`: Used by Flask for securely signing the session cookie.

2. Model Loading (`load_model()`):

- The model is loaded from `checkpoint.pt` using PyTorch. This includes setting the model to evaluation mode (`model.eval()`), which is crucial for deployment as it tells PyTorch that the model is being used for inference rather than training.

3. Video Processing (`validation_dataset` class):

- This custom dataset class is responsible for processing video files into a tensor format that your model can use. It uses OpenCV to extract frames from videos, applies transformations, and stacks these into a tensor.

4. Flask Routes:

- `@app.route('/')`: Handles the homepage and file uploads. If a file is posted, it is saved and the user is redirected to the result page.
- `@app.route('/result/<filename>')`: Processes the uploaded video using the model, determines if the video is 'REAL' or 'FAKE', and shows the result.
- `@app.route('/uploads/<filename>')`: Serves uploaded files from the `UPLOAD_FOLDER`.

Supporting Files

1. model.py:

- Contains the definition of your PyTorch model (`Model`). This file defines how your neural network is structured, which is crucial for correctly loading the model and running predictions.

2. upload.html (Frontend F+orm):

- This HTML file contains a form that users use to upload video files. It includes frontend validation to ensure that only video files are uploaded.

3. result.html(Results Display):

- Displays the results of the video analysis. It shows whether the uploaded video was predicted as 'REAL' or 'FAKE' and provides a link to upload another video.

4. styles.css:

- Contains CSS styles that define the look and feel of your website. This might include styles for the form, buttons, text, and responsive design elements to ensure the site looks good on different devices.

5. checkpoint.pt:

- A PyTorch checkpoint file that contains the trained model weights. This file is crucial for making predictions as it allows the `Model` class to be instantiated with learned parameters.

How Flask Supports the Program

In our case, Flask acts as the server that routes HTTP requests to the appropriate Python functions. It handles the web server's responsibilities such as managing requests and responses, serving files, and redirecting users. It connects the HTML front end with the backend Python logic that processes the videos through your deep learning model.

The Flask application provides a full-stack solution by combining HTML/CSS for the front end, Flask as the web framework to handle routing and server-side logic, and PyTorch for executing machine learning predictions.

Ethical Considerations

The deployment of deepfake detection technologies, particularly those involving facial recognition, raises significant ethical considerations that must be thoroughly addressed. Firstly, the use of video data involves privacy concerns; it is imperative to ensure that all data is acquired and used with appropriate consent and in compliance with data protection regulations.

Secondly, the potential for bias in facial detection algorithms is a critical issue. Biases can occur due to disparities in the training datasets, where certain demographic groups are underrepresented, leading to less accurate detection for those groups. This could result in unfair treatment or discrimination, undermining the integrity and fairness of the technology.

To mitigate these risks, it is essential to employ diverse and representative datasets and to continuously evaluate and refine the algorithms to ensure they operate equitably across different populations. Additionally, transparency in how these technologies are developed and used is crucial for building trust and accountability, particularly in applications as impactful as combating deepfake videos.

Addressing these ethical challenges is not just about compliance but is fundamental to the responsible development and deployment of technologies that respect individual rights and promote fairness.

Conclusion

In conclusion, we have used ResNet to build a deepfake detection model and deployed it through a Flask application. Albeit having strong ethical considerations, a detection model like this is vital in the current world climate. We have recently seen many videos circulating online and if deployed correctly, this could prevent fake videos being spread.

References

- Grinberg, M. (2018). Flask Web Development: Developing Web Applications with Python (2nd ed.). O'Reilly Media.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32.
- Duckett, J. (2011). HTML and CSS: Design and Build Websites. John Wiley & Sons.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
- Szeliski, R. (2010). Computer Vision: Algorithms and Applications. Springer Science & Business Media.
- Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). Large-scale Video Classification with Convolutional Neural Networks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.
- OpenAI. (2024). ChatGPT [Large language model]. https://chat.openai.com/chat
- Papadopoulos, A. (n.d.). How to train an ensemble of convolutional neural networks for image classification. Medium.