For app.py code in project:

This code sets up a simple Flask web application for uploading videos and detecting deepfake content using a module called `deepfake\_detector`. Let's break down the code step by step:

1. \*\*Importing Modules\*\*:

- The code imports necessary modules like Flask for creating the web application, `render\_template` for rendering HTML templates, `request` for handling HTTP requests, `redirect` for redirecting to other routes, `url\_for` for URL generation, `os` for interacting with the operating system, `datetime` for working with dates and times, `json` for JSON manipulation, `current\_time` for getting the current time, and `importlib` for importing modules dynamically.

2. \*\*Setting Up Flask\*\*:

- An instance of the Flask application is created.

3. \*\*Configuring Upload Folder\*\*:

- An upload folder named `'static/videos'` is configured to store uploaded videos.

4. \*\*Defining Routes\*\*:

- \*\*Index Route (`'/'`)\*\*: Renders the `index.html` template when users visit the root URL.

- \*\*Upload Route (`'/upload'`)\*\*: Handles file uploads. When a video file is uploaded, it saves the file to the upload folder, generates a timestamped filename, saves the file with the new filename, imports a module named `deepfake\_detector`, calls a function named `run` from this module with the paths of the original and modified video files, collects information about the uploaded video, converts it to JSON format, and redirects to the result route (`'/result'`) with the video information.

- \*\*Result Route (`'/result'`)\*\*: Renders the `result.html` template, passing the video URL and video information obtained from the upload route.

5. \*\*Running the Application\*\*:

- The Flask application is run with debugging enabled if the script is executed directly.

The `deepfake\_detector` module seems to contain a function named `run` that presumably detects deepfake content in videos. The result of this detection is included in the information about the uploaded video (`video\_info`) and is passed to the result page to be displayed along with other video information.

Overall, this code sets up a basic web application for uploading videos, detecting deepfake content, and displaying the results.

Detector.py : information:

This Python code defines a function `run` that detects deepfake content in a video using a combination of computer vision techniques and deep learning models. Let's break down the code step by step:

1. \*\*Importing Libraries\*\*:

- `cv2`: OpenCV library for computer vision tasks.

- `numpy`: Library for numerical computations.

- `MTCNN`: Face detection model from the `facenet\_pytorch` library.

- `InceptionResnetV1`: Pretrained face recognition model from the `facenet\_pytorch` library.

- `functional` from `torchvision.transforms`: Utilities for image transformation.

- `time`: Module for working with time.

2. \*\*Function Definition\*\*:

- The `run` function takes two arguments: `video\_path` (path to the input video) and `video\_path2` (path to the output video with detection annotations).

3. \*\*Initialization\*\*:

- Initialization of variables and models including `mtcnn` (MTCNN face detector), `facenet\_model` (InceptionResnetV1 face recognition model), video capture from `cv2.VideoCapture`, and setup for writing output video.

4. \*\*Detection Loop\*\*:

- The main loop iterates through each frame of the input video.

- Every few frames (controlled by `frames\_between\_processing`), it detects faces using the MTCNN model.

- For each detected face, it extracts the facial region, resizes it, converts it to a tensor, and computes its embedding using the InceptionResnetV1 model.

- It then calculates the similarity between the current face embedding and the previous one.

- If the similarity is below a certain threshold (`threshold\_face\_similarity`), it increases a counter for potential deepfake frames (`deepfake\_count`).

- If the number of consecutive potential deepfake frames exceeds another threshold (`threshold\_frames\_for\_deepfake`), it marks the frame as a deepfake and increments the deepfake frame counter (`deep\_fake\_frame\_count`).

- Annotations are added to the video frames indicating whether they are detected as real or potential deepfake frames.

5. \*\*Post-processing and Cleanup\*\*:

- After processing all frames, the function calculates execution time, releases video resources, and calculates the accuracy of deepfake detection.

- The accuracy is calculated as the ratio of detected deepfake frames to the total number of frames, multiplied by 1000.

- If the calculated accuracy exceeds 1000 (which shouldn't happen but could due to rounding), it caps the accuracy at 95%.

6. \*\*Return\*\*:

- The function returns the calculated accuracy.

This code provides a basic framework for detecting deepfake content in videos using face detection and recognition techniques, combined with threshold-based checks for frame-to-frame consistency.

**Models used::**

**The model used in this code for face detection and recognition is a combination of two pretrained models:**

**1. \*\*MTCNN (Multi-Task Cascaded Convolutional Neural Network)\*\*:**

**- MTCNN is a popular deep learning model for face detection. It consists of three stages:**

**- \*\*Stage 1 (Proposal Network)\*\*: Generates candidate bounding boxes for faces using a convolutional neural network (CNN).**

**- \*\*Stage 2 (Refinement Network)\*\*: Refines the bounding boxes generated in Stage 1 and rejects those that are unlikely to contain a face.**

**- \*\*Stage 3 (Output Network)\*\*: Refines the bounding boxes further and performs facial landmark detection.**

**- MTCNN is known for its high accuracy in detecting faces under various conditions such as scale, pose, and occlusion.**

**2. \*\*InceptionResnetV1\*\*:**

**- InceptionResnetV1 is a deep convolutional neural network architecture designed for face recognition tasks. It combines elements from the Inception architecture and residual connections from ResNet.**

**- This model is pretrained on large-scale face recognition datasets such as VGGFace2.**

**- Given an input facial image, InceptionResnetV1 extracts a fixed-length feature vector (embedding) representing the face. This embedding is typically used to compare faces for similarity or to classify identities.**

**In the code:**

**- MTCNN is used for detecting faces in each frame of the input video.**

**- Detected faces are then passed through InceptionResnetV1 to extract embeddings representing each face.**

**- These embeddings are compared between consecutive frames to detect any significant changes, which may indicate the presence of a deepfake.**

**By using a combination of these models, the code aims to accurately identify potential deepfake frames by analyzing changes in facial features across frames.**

Overview:

he video file is opened, and various video properties such as fps, width, and height are obtained.

2- Face detection is performed using **MTCNN (Multi-Task Cascaded Convolutional Networks)**.

3- The detected face is transformed into a feature vector using a pre-trained **Inception Resnet V1 model (InceptionResnetV1)**.

4- A comparison is made with the face in the previous frame, and a similarity score is calculated.

5- Similarity scores below a certain threshold are considered as indicative of a deepfake.

6- If deepfakes are detected in a consecutive number of frames, it is marked as a deepfake, and a frame is added to the video.

7- Processed frames are written to an output video file.