Truth Behind Deepfakes

Khuswant Kaswan (241110035)
Computer Science Engineering
IIT Kanpur
Kanpur, India
khushwantk24@cse.iitk.ac.in

Rishit Kumar (241110056)
Computer Science Engineering
IIT Kanpur
Kanpur, India
rishitk24@cse.iitk.ac.in

Krishanu Ray (241110037)
Computer Science Engineering
IIT Kanpur
Kanpur, India
krishanur24@cse.iitk.ac.in

Senthil Ganesh P (241110089)
Computer Science Engineering
IIT Kanpur
Kanpur, India
senthil24@cse.iitk.ac.in

Divyanshu (241110023)
Computer Science Engineering
IIT Kanpur
Kanpur, India
divyanshu24@cse.iitk.ac.in

Rajan Kumar (241110087)
Computer Science Engineering
IIT Kanpur
Kanpur, India
rajank24@cse.iitk.ac.in

***Abstract*—With the rapid evolution of generative technologies, deepfakes highly convincing synthetic images and videos have become increasingly prevalent. These manipulated media forms presents serious risks to the authenticity of digital content and have broader implication for security, misinformation, and public perception. This project introduces a two phase detection pipline designd to effectivly identify deepfakes in both images and vedios. The frst phase focuses on image classification, where mutiple CNN arcitectures, including VGG16, DenseNet121, and EfficentNet-B0, were comparred. EfficentNet-B0 was finaly selected for its strong balnce between accurracy and computational coast. The second phse adress video-based detection, using temporal modeling tecniques. Here, we combine CNN features extractors EfficentNet-B0 [15] and ResNext50 [13] with LSTM [9] layers to catch temporal pattrns. Also, we implemented the Swin Transfomer to leverage spatial and temmporal features via its attention based mechanisim. To inprove overall performnce, we ensemle prediction from all three model. The pipline is more enhanced by advansed preprocessing step like dual-stage face verifiction using MTCNN, uniform frame sapmling, and training optimizations with mixed-precision techniks. Evalutions were done on datasest like FaceForensics++ [10], UADFV [14], and CelebDFv1 [4], giving accuracy of upto 96.76% on benchmarks and good generalzation to new datas. At last, a user-freindly Gradio [8] interface is made to demo realtime detection capablities.**

***Index Terms*—Deepfake Detection, EfficientNet-B0, ResNeXt50, LSTM, Swin Transformer,Ensemble Learning, MTCNN**

I. Introduction

A. Background and Motivation

Deepfake synthetic media that look almost like real people have quickly grown into a major concern in today's digital world. Using deep learning techniques like GANs and autoencoders, these fake images and videos are realistic enough to fool both humans and many automated systems. Unfortunately, they've been used in harmful ways from fake political videos to scams and impersonation. This makes the ability to detect deepfakes really important for things like security, journalism, and public trust.

The problem is, deepfake generation tools are improving faster than the detection methods. Many of the existing approaches do not work well when tested on different types of data, or they only work on images but not videos. So, there is a big need for something more reliable that works on both static and moving content. Our project tries to fill that gap by designing a pipeline that can catch deepfakes in both images and videos using newer deep learning models.

B. Project Scope and Phases

We broke the work into two parts:

Phase 1: Deepfake Image Detection

This part was focused on spotting fake face images. We used the Kaggle dataset of 140K real and fake faces. At first, we tried a bunch of different CNNs like VGGFace, XceptionNet, and DenseNet. Some were too slow or didn't perform well enough. Finally, we picked EfficientNet-B0 as it had the best balance of speed and accuracy. We tested it under various noisy or altered image conditions to see how well it holds up, and it performed well enough to go ahead with it.

Phase 2: Deepfake Video Detection

Here, things got more complex since videos need both spatial and temporal analysis. We chose the c23 Low Quality videos which are part of the FaceForensics++ dataset. The idea was to use CNNs to pull features from frames, and then feed that into LSTM layers to learn changes over time. EfficientNet-B0 and ResNeXt50 were used as the CNN backbones. We also added the Swin Transformer model, which is good at handling both space and time parts of the video using attention mechanisms.

We also tried combining all three models to see if that improved the results:

- EfficientNetB0 + LSTM
- ResNeXt50 + LSTM
- Swin Transformer

For better face accuracy, we used a two-step face detection and verification process with MTCNN [12] with varying thresholds of 0.7, 0.8 and 0.9:

1) Extract frames from video frames over full length with open-cv [5] and apply MTCNN to extract faces
2) Filter them to keep only the clean and consistent ones

All models were built and trained with PyTorch [6] torchvision [11], with mixed precision training to save memory and train faster. We also tested our models not just on FF++, but also tested for generalization behaviour on Celeb-DFv1 and UADFV, which are more challenging and realistic datasets. Similar to out train-val-test split of 70-20-10, we tested on 10% CelebDFv1 dataset and since UADFV is a way smaller dataset compared to other two so we tested on 70% of it.

C. Implementation Strategy

We tried to keep the project practical and solid by doing the following:

- Make the models fast and simple enough to use by utilizing pretrained CNN models
- Did fine-tuning of models and looked for overfitting and saved weights
- Did experiments with variations and studied which ones worked best
- Make sure our models generalize well on other datasets too

At the end, we built a Gradio [8] based web interface utilizing Model Ensembling so users can upload an image or video, and get back a result showing if it's fake or real—plus a confidence score(probability) averaged from all three models.

D. Contributions and Impact

Here's what we contributed in this work:

- A two-step system that handles both image and video deepfake detection
- Used CNNs, LSTMs, and Transformers together to improve detection
- Used smart preprocessing like 2-step facial verification to reduce False positives
- Built a real-time demo interface using Gradio with fast inference.
- Validated on multiple datasets to show it works in different scenarios and generalizes well.

Our project not only hits high test accuracy but also tries to be actually useful in real situations. It lays down a good base for further work in detecting smarter and more realistic deepfakes going forward.

II. Related Work

Deepfake detection has gained significant attention in recent years due to the rise in manipulated videos and images. Various researchers have proposed different deep learning models for identifying such fake media. In this section, we discuss some important papers related to this area and how their methods compare with ours.

One of the earlier works was the *Two-Stream Network* [10] which used both RGB frames and optical flow for detecting fakes. It was trained on FF++ dataset and had decent accuracy but didn't generalize well, especially to datasets like CelebDFv1. The performance dropped significantly in cross-dataset testing.

Another popular method is *FWA (Face Warping Artifacts)* [14], which focused on detecting artifacts created during face warping. It was trained and tested only on UADFV dataset and achieved very high accuracy there, but didn't perform well when used on other datasets like FF++ or CelebDF due to overfitting to UADFV-specific artifacts.

Multi-task CNN with LSTM [12] tried to improve results by combining spatial and temporal features. This model was trained on FF++ and tested on CelebDFv1 but showed low accuracy on new data. This suggests that simply using LSTM does not help much if the training dataset lacks diversity.

In 2022, *EfficientNet and Vision Transformer based ensemble* [3] was proposed. It used EfficientNet as the backbone CNN and combined it with a Vision Transformer to better capture spatial-temporal patterns. While it showed good performance on FF++, it was computationally heavy and not very practical for real-time systems.

Another recent work is *Multi-modal Multi-scale Transformer (MMFT)* [2] which used an advanced transformer architecture to extract both low and high scale features. It performed quite well on FF++ but again failed to generalize well to CelebDFv1.

A different approach was taken in *On the Detection of Digital Face Manipulation* [1], where researchers analyzed digital manipulation patterns using large-scale face data and proposed detection mechanisms that could work across different manipulations. While effective in detecting common deepfakes, it still faced challenges on unseen manipulations and compression levels.

Our method is mainly trained on the **FF++ (LQ c23 compressed)** dataset and includes a combination of CNN backbones (EffB0 and ResNext50) combined with LSTM, along with another model involving Swin Transformer. While training was done only on FF++, we tested the same model on **UADFV** and **CelebDFv1** to evaluate generalization. Not a single video from these datasets was used for training. The results were significantly better than many past works, especially without any fine-tuning on these new datasets.

TABLE I
DEEPFAKE DETECTION METHODS – ARCHITECTURE AND DATASET

Work / Paper	**Architecture**	**Training Dataset**
Two-Stream Network [10]	CNN (RGB + Optical Flow)	FF++
FWA (Face Warping Artifacts) [14]	CNN (Artifact-based)	UADFV
Multi-task CNN + LSTM [12]	CNN + LSTM	FF++
EfficientNet + ViT Ensemble [3]	CNN + Transformer	FF++ (HQ)
MMFT [2]	Multi-modal Transformer	FF++ (HQ)
Ours (EffB0+LSTM, ResNeXt+LSTM, Swin-T)	CNN + LSTM Transformer	FF++ (LQ)

TABLE II
ACCURACY ON BENCHMARK DATASETS

Work / Paper	Accuracy (FF++)	Accuracy (CelebDFv1)	Accuracy (UADFV)
Two-Stream Network [10]	70.1%	55.7%	–
FWA (Face Warping Artifacts) [14]	–	–	**97.4%**
Multi-task CNN + LSTM [12]	65.8%	39.6%	–
EfficientNet + ViT Ensemble [3]	95.1% (AUC)	–	–
MMFT [2]	97.6%	∼54%	–
Ours	**96.76%**	**76%**	**95%**

Our pipeline performs better in terms of generalization while being more memory efficient and real-time ready due to mixed precision training and linear interpolation-based uniform sampling across entire video duration for frames. Unlike other works that focus only on spatial features or only on temporal ones, our model combines both and uses an ensemble of three models.

III. PROPOSED METHODOLOGY

Dataset and Preprocessing Pipeline for Images: For the image-based deepfake detection task, a subset of 80,000 face images (real and fake) was selected from a larger Kaggle dataset containing 140,000 samples. The dataset was divided into 60,000 images for training (30K real and 30K fake), 10,000 for validation, and 10,000 for testing. To ensure consistency and optimal model performance, all images were resized to 128×128 pixels and normalized using ImageNet statistics (mean: [0.485, 0.486, 0.406], std: [0.229, 0.224, 0.225]) to align with the input requirements of the EfficientNet-B0 model.

Link to the dataset can be found at this link.

Training loop for CNN for Image Classification: For the image-based deepfake detection task, EfficientNet-B0, pretrained on ImageNet, was used as the base model, with its classifier head modified for binary classification (real vs fake). The model was trained using PyTorch over 10 epochs with a batch size of 64. The training setup included CrossEntropyLoss as the loss function and the Adam optimizer with a learning rate of 0.001. Mixed-precision training was employed using torch.amp for improved efficiency, and the training was performed on a CUDA-enabled GPU. The pipeline included loading image paths and labels from a CSV file, preprocessing, creating DataLoaders, training the model while tracking metrics, applying early stopping based on validation loss, and finally evaluating the saved model on the test set.

Dataset and Preprocessing Pipeline for Videos:

We have utilized 1362 real videos (YT and actors) and 3997 fake videos (Deepfakes, Face2Face, FaceSwap, FaceShifter) and with extracted 32 frames over whole video lengths and utilized MTCNN for face detection cropping and verification to reduce false face detections that plague many deepfake detectors. Cleaner input data are obtained compared to single-pass MTCNN detections. Futhermore, each of the faces are resized to 224x224 pixels (each of our models takes a 224x224 input image) and aggressive transformations are done on train data compared to validation and test sets for robustness and to simulate real-world distortions.

Each input sample will consists of a sequence of extracted faces from videos. Frames are loaded using a custom Dataset class that ensures temporal consistency (equally spaced frames). The final shapes of the data loaders is *(batch_size,num_frames,C,H,W)* where num_frames is the number of frames(≤ 32) we are using per video for training , C,H,W represents the images (3,224,224).

Fig. 8 shows the flow of our preprocessing pipeline.

Link to the dataset can be found at this link.

Training loop for CNN+LSTM:

The training loop for the proposed CNN+LSTM-based deepfake detection model is designed to process spatio-temporal information from video frames. This is accomplished by combining pretrained CNN backbone (either ResNeXt-50 or EfficientNet-B0) for spatial feature extraction followed by an LSTM for temporal modeling. The workflow is visually depicted in Fig 9.

Each extracted face is individually passed through the pretrained CNN. The model extracts spatial features from each frame resulting in shape: *(batch_size × num_frames, C_out, H', W')*. The spatial dimension is reduced using Adaptive Average Pooling and converts to frame-wise features of shape: *(batch_size, num_frames, C_out)* to pass it as a sequence to an LSTM to captures temporal dependencies across frames. Input dimension of LSTM will be decided by C_out which is 2048 for ResNext and 1280 for EfficientNetB0. LSTM can be unidirectional or bidirectional. Output shape will be *(batch_size, num_frames, hidden_dim)* (hidden_dimx2 if bidirectional). After the LSTM, temporal aggregation is done via mean pooling across the num_frames and normalized with LayerNorm. The aggregated features are passed through a fully connected layer with Dropout for final prediction activated using LeakyReLU: *(batch_size, num_classes)*.

Training loop for SWIN Transformer: Similarly, the training pipeline for the Swin Transformer-based deepfake detection model leverages temporal consistency across video frames and spatial feature richness from transformer backbones. The workflow is visually depicted in Fig 10.

Frames are processed using a pretrained Swin Transformer model *(swin_tiny_patch4_window7_224)* returning multi-scale feature maps. Only the last feature stage is selected resulting in shape *(batch_size × num_frames, C_out, H', W')*. Like earlier Average Pooling is applied over H' and W' and the tensor is reshaped to *(batch_size, num_frames, C_out)*. Now we aggregates features directly across the temporal axis and perform LayerNorm leading to *(batch_size, C_out)*.Like before a fully connected layer activated with LeakyReLU and Dropout is used for final prediction : *(batch_size, num_classes)*.

Loss computation for both is done using weighted CrossEn-

tropyLoss to handle class imbalance.

$$\text{weight}_{\text{class}} = \frac{\frac{N_{\text{total}}}{num_{\text{classes}}}}{N_{\text{class}}}$$

Training is done with AMP mixed-precision training (via autocast and GradScaler) and accuracy computation is done using max prediction over logits.

After each epoch, the model is evaluated on the validation set. Early stopping is monitored with patience. If no improvement is seen for patience epochs, training halts and the best checkpoint is restored. Once training stops (either by convergence or early stopping), the best model weights are loaded and evaluation is performed on the test set.

Approximate Time Complexity Analysis for CNN+LSTM based model

$$\text{T}_{\text{total}} = \underbrace{B \cdot F \cdot \mathcal{T}_{\text{CNN}}}_{\text{CNN Cost}} + \underbrace{B \cdot F \cdot \mathcal{T}_{\text{LSTM}}}_{\text{LSTM Cost}}$$

$$\text{T}_{\text{CNN}} = \sum_{\ell=1}^{D} \underbrace{C_{in}^{\ell} \cdot C_{out}^{\ell} \cdot k_{\ell}^{2} \cdot H_{\ell} \cdot W_{\ell}}_{\text{Layer } \ell}$$

$$\text{T}_{\text{LSTM}} = 4 \cdot \underbrace{h^2}_{\text{Hidden-Hidden}} + \underbrace{h \cdot C_{\text{out}}}_{\text{Input-Hidden}}$$

$$\text{T}_{\text{total}} = B \cdot F \cdot \left(\sum_{\ell=1}^{D} C_{in}^{\ell} C_{out}^{\ell} k_{\ell}^{2} H_{\ell} W_{\ell} + 4h(h + C_{\text{out}}) \right)$$

B is the Batch Size. F is Frames per Video extracted. D is the CNN Depth. C_{in}^{ℓ} and C_{out}^{ℓ} are the number of input and output channels in layer ℓ. k^{ℓ} is Kernel Size for Layer ℓ. H,W are the Spatial Dimensions of Layer ℓ. h is the Dimension of hidden state in LSTM. C_{out} is the Final feature dimension from the CNN backbone. 4 Gates of LSTM contribute 4×4 matrix operations for the recurrent connection.

Inference Pipeline with Ensemble of Deepfake Detection Models A web based Gradio interface is designed to accept a wide range of input types (videos, images, or zip archives of frames) and outputs both prediction scores and visual result. The workflow is visually depicted in Fig 3.

Inputs are prioritized in the order: ZIP folder ¿ Image ¿ Video. Only one source is processed at a time. When a video is provided , Faces are extracted from uniformly sampled frames using MTCNN. Detections are filtered by a confidence threshold (user-controlled slider). If the input is a zip of frames or a single image, this step is skipped.

The system loads the three trained model weights and the extracted images are passed to the models. All models output softmax probabilities over two classes. The system performs soft probability ensembling by averaging the real/fake scores from all three models and decides the label.

The user receives a gallery of annotated face crops, a summary of probabilities and model-level predictions and processing statistics (e.g., face detection rate, inference time) as seen in Fig 11 and Fig 12.

IV. Experimental setup

The experiments were primarily conducted on a local machine equipped with an NVIDIA® RTX 4060 GPU with 8GB of VRAM. For the Swin transformer model, experiments were conducted on Kaggle T4 GPU with 15GB of VRAM. The project was implemented using the PyTorch deep learning framework, chosen for its flexibility and strong support for custom architectures. For training, the hyper-parameters in Table 3 were fixed, while batch size and Number of Frames were varied. For each "Number of frames", we tried training the model on the highest possible "Batch size" which did not lead to a CUDAOutOfMemory error. This was done to reduce the training time since experimentation involved training multiple models and evaluating their accuracies. Training was conducted on the following three different models and then the training graphs, confusion matrix and test accuracies were generated.

Parameters estimation for CNN+LSTM based model

1) **CNN backbone**

- **EfficientNet-B0**
  - Total model ≈ 5.29M parameters
  - Classifier (linear for ImageNet-1k) has 1.28M ⇒ features only ≈ 5.29M – 1.28M = **4.01M**
- **ResNeXt-50 32×4d**
  - Total model ≈ 25.03M parameters
  - Final FC layer has (2048 × 1000 + 1000) ≈ 2.049M ⇒ features only ≈ 25.03M – 2.049M = **22.99M**

2) **LSTM** (single-layer, hidden = 256, bidirectional = False)

- Input→hidden weights: $4 \times$hidden_dim$\times$latent_dim
- Hidden→hidden weights: $4 \times \text{hidden\_dim}^2$
- Biases: $2 \times (4 \times \text{hidden\_dim})$
- **EFB0** (*latent_dim* = 1280): = 1,310,720 + 262,144 + 2,048 = **1.574,912**
- **ResNeXt** (*latent_dim* = 2048): = 2,097,152 + 262,144 + 2,048 = **2.361,344**

3) **LayerNorm**

- $2 \times \text{hidden\_dim} = 2 \times 256 =$ **512**

4) **Linear head** ($256 \rightarrow 2$ classes)

- Weights: $256 \times 2 = 512$
- Bias: 2
- Total = **514**

Adding them up gives the totals in the table 4.

TABLE III
TRAINING HYPER PARAMETERS USED IN THE EXPERIMENTS.

Hyperparameter	Value
Batch size	2 or 4 or 8
Number of frames	10 or 20 or 30
Number of epochs	20
Early stopping patience	5
Learning rate	1e-4
Optimizer	AdamW
LR Scheduler	StepLR (step=5, gamma=0.5)
Loss function	Weighted CrossEntropyLoss
Class weights	[1.966, 0.67]
Mixed precision	Enabled (AMP)
Dropout	0.4

TABLE IV
APPROXIMATE PARAMETER COUNT BREAKDOWN FOR CNN+LSTM VARIANTS

Component	EFB0 + LSTM	ResNeXt50 + LSTM
CNN backbone	≈ 4.01 M	≈ 22.99 M
LSTM	1.574 M	2.361 M
LayerNorm	0.000512 M	0.000512 M
Linear head	0.000514 M	0.000514 M
Total	**≈ 5.58 M**	**≈ 25.35 M**

TABLE V
MODELS USED IN THE EXPERIMENTS.

Model	Number of Parameters	Size
Effb0+Lstm	5.583M	23MB
ResNext+Lstm	25.342M	102MB
Swin	27.52M	110MB

V. RESULTS ON DEEPFAKE IMAGE DETECTION USING EFFICIENTNETB0

Test accuracy achieved on Kaggle 140K dataset was 98.84%. Time take to train the model 99.68 minutes. Refer Fig 1 and Fig. 2.

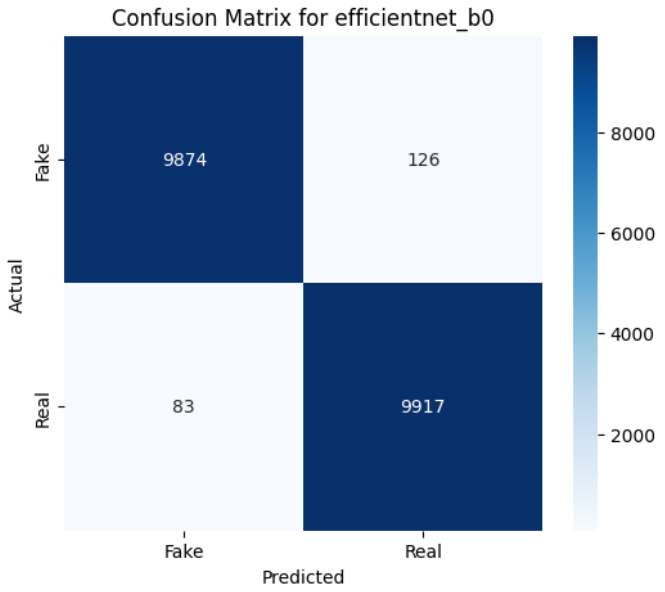

Fig. 1. Confusion matrix of model trained on EfficientNetb0 for image deepfake detection

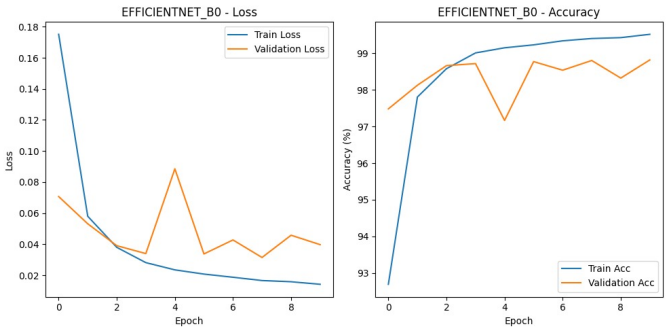

Fig. 2. Train and validation loss and accuracy over epochs of model trained on EfficientNetB0 for image deepfake detection

TABLE VI
TEST ACCURACY ON VARIOUS CHOICES OF BATCH SIZE AND FRAMES FOR EFFICIENTNETB0 + LSTM

Batch Size (BS)	Number of Frames (F)	Test Accuracy	Training time(minutes)
8	10	94.79%	53.955
4	20	96.08%	144.913
4	30	96.76%	153.738

VI. RESULTS ON DEEPFAKE VIDEO DETECTION

A. Results on EfficientNetB0 + LSTM

The following three configurations gave us the top-3 test accuracies, out of which the model trained on Batch size = 4 and Number of Frames = 30 gave us the best test accuracy of 96.76%. The training time of this model was 153.738 minutes on local machine. Refer Fig. 3 and Fig. 4.

B. Results on ResNext50 + LSTM

The following three configurations gave us the top-3 test accuracies, out of which the model trained on Batch size = 2 and Number of Frames = 30 gave us the best test accuracy of 96.56%. The training time of this model was 224.652 minutes on local machine. Refer Fig. 5 and Fig. 6.

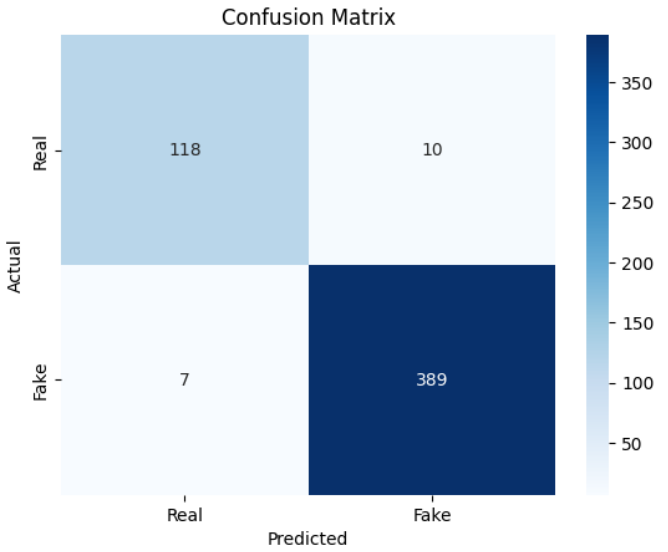

Fig. 3. Confusion matrix of model trained on EfficientNetB0+LSTM with Batch Size = 4 and Number of Frames = 30

Fig. 4. Train and validation loss and accuracy over epochs of model trained on on EfficientNetB0+LSTM with Batch Size = 4 and Number of Frames = 30

TABLE VII
TEST ACCURACY ON VARIOUS CHOICES OF BATCH SIZE AND FRAMES FOR RESNEXT50 + LSTM

Batch Size (BS)	Number of Frames (F)	Test Accuracy	Training time(minutes)
8	10	95.16%	99.01
4	20	95.52%	127.192
2	30	96.56%	224.652

C. Results on Swin

The following three configurations gave us the top-2 test accuracies, out of which the model trained on Batch size = 8 and Number of Frames = 10 gave us the best test accuracy of 93.85%. The training time of this model was 205.298 minutes on Kaggle T4 GPU, while training the Batch size = 4,Number of Frames=20 took 403.409 minutes. We couldn't perform much experimentation as the time taken to train the model was much higher compared to the other models even on lowest configurations. Refer Fig. 7.

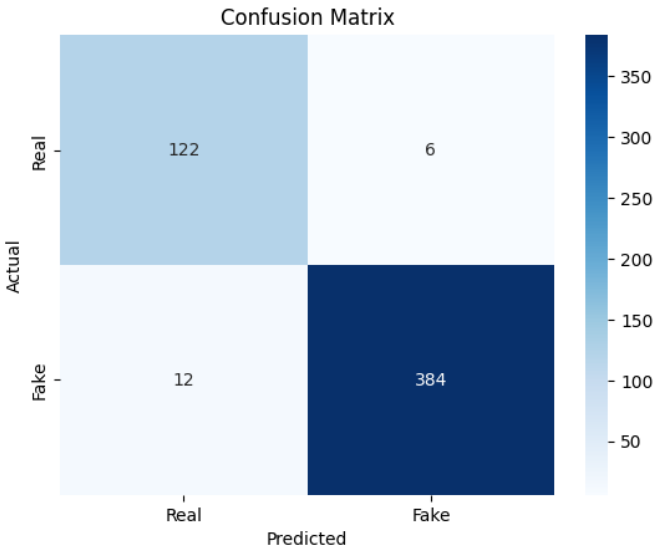

Fig. 5. Confusion matrix of model trained on ResNext50+LSTM with Batch Size = 2 and Number of Frames = 30

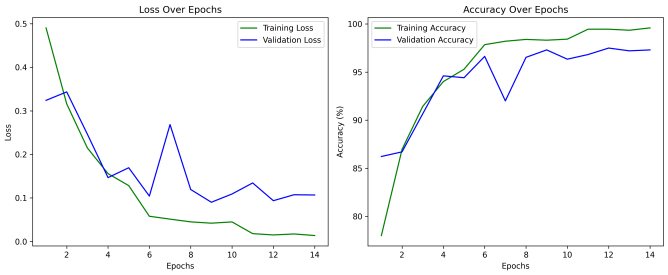

Fig. 6. Train and validation loss and accuracy over epochs of model trained on ResNext50+LSTM with Batch Size = 2 and Number of Frames = 30

TABLE VIII
TEST ACCURACY ON VARIOUS CHOICES OF BATCH SIZE AND FRAMES FOR SWIN TRANSFORMER

Batch Size (BS)	Number of Frames (F)	Test Accuracy	Training time(minutes)
8	10	93.85%	205.298
4	20	75.19%	403.409

TABLE IX
CONFUSION MATRIX OF MODEL TRAINED ON SWIN WITH BATCH SIZE = 8 AND NUMBER OF FRAMES = 10

	Predicted Positive	Predicted Negative
Actual Positive	117 (TP)	20 (FN)
Actual Negative	13 (FP)	387 (TN)

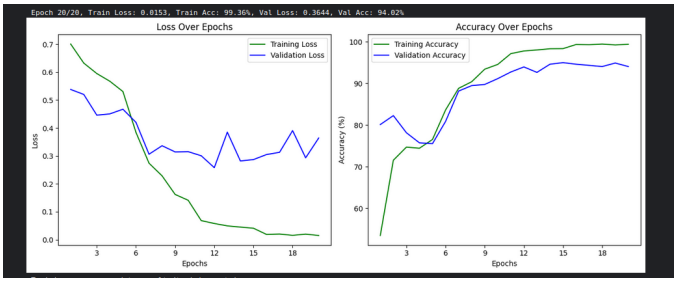

Fig. 7. Train and validation loss and accuracy over epochs of model trained on Swin with Batch Size = 8 and Number of Frames = 10

D. Observations

Since we were achieving almost 99& test accuracy on deepfake image dataset, we focused deeply on video deepfake. Following things were observed while conducting the experiments of training and testing the models:

- Training the Swin model took way longer than training the other two models, while getting no noticable improvements on the test accuracies.
- Test accuracies on UADFV were higher compared to CelebDFV1 datasets.
- There is no overfitting observed in any of the models as evident from the curves. To reduce the chances of overfitting we have applied many strategies like Early stopping, dropout layers etc.,.

VII. Discussion and Future Work

Due to time limitations and limited compute power, we couldn't try the following:

- We have trained our model only on FaceForensics++ datasets, we could've trained the models on other datasets such as DFDC or CelebDF. Strategies like data augmentation from multiple datasets were not explored.
- We have used only Unidirectional LSTM, we could've tried training with Bidirectional LSTM.
- Instead of LSTM we could've also explored GRU or xLSTM.
- Acoustic feature analysis for audio related deepfakes could've been integrated with current work.

VIII. Conclusion

The implemented model successfully detects deepfake images and videos with a test accuracy of 96.76% with combination of EfficientNetB0 and LSTM. To make the predictions more robust, the ensemble of three different models (EfficientNetB0+LSTM , ResNext50+LSTM and SWIN Transformer) have been used and the same has been integrated in the user interface.

IX. Generalization of Models on UADFV and CelebDFv1 and the Overall Training Summary

Model	Parameters	Dataset		Precision	Recall	F1Score	Accuracy
ResNext-50	BS:2 F:10	UADFV	Real	0.825	0.97	0.89	0.88
			Fake	0.96	0.79	0.87	
		CelebDFv1	Real	0.53	0.90	0.67	0.70
			Fake	0.92	0.60	0.72	
	BS:2 F:20	UADFV	Real	0.84	0.82	0.83	0.83
			Fake	0.82	0.85	0.84	
		CelebDFv1	Real	0.51	0.90	0.65	0.67
			Fake	0.91	0.55	0.69	
	BS:2 F:30	UADFV	Real	0.96	0.93	0.95	0.95
			Fake	0.93	0.96	0.95	
		CelebDFv1	Real	0.47	0.97	0.63	0.61
			Fake	0.97	0.43	0.59	

Model	Parameters	Dataset		Precision	Recall	F1 Score	Accuracy
EfficientNetB0	BS:8 F:10	UADFV	Real	0.93	0.91	0.92	0.92
			Fake	0.91	0.94	0.92	
		CelebDFv1	Real	0.47	0.82	0.68	0.62
			Fake	0.85	0.52	0.65	
	BS:8 F:20	UADFV	Real	0.96	0.82	0.88	0.89
			Fake	0.84	0.97	0.94	
		CelebDFv1	Real	0.62	0.80	0.70	0.76
			Fake	0.88	0.74	0.80	
	BS:4 F:30	UADFV	Real	0.93	0.87	0.90	0.90
			Fake	0.88	0.93	0.90	
		CelebDFv1	Real	0.52	0.73	0.61	0.69
			Fake	0.83	0.66	0.74	

Model	Parameters	Dataset		Precision	Recall	F1 Score	Accuracy
SWIN	BS:2 F:10	UADFV	Real	1	0.85	0.92	0.92
			Fake	0.87	1	0.93	
		CelebDFV1	Real	0.47	0.70	0.56	0.63
			Fake	0.80	0.6	0.68	
	BS:2 F:20	UADFV	Real	0.96	0.85	0.90	0.91
			Fake	0.86	0.97	0.91	
		CelebDFv1	Real	0.51	0.80	0.62	0.67
			Fake	0.85	0.60	0.71	

Model	Parameters	Accuracy		Precision	Recall	F1 Score
EfficientNetB0	F=10 BS:8	94.79%	Real	0.90	0.92	0.91
			Fake	0.97	0.96	0.96
	F=20 BS:4	96.08%	Real	0.91	0.86	0.88
			Fake	0.95	0.97	0.96
	F=30 BS:4	96.76%	Real	0.85	0.92	0.88
			Fake	0.97	0.95	0.96
ResNext-50	F=10 BS:8	95.16%	Real	0.91	0.89	0.90
			Fake	0.96	0.97	0.96
	F=20 BS:4	95.52%	Real	0.88	0.94	0.91
			Fake	0.97	0.96	0.97
	F=30 BS:2	96.56%	Real	0.91	0.95	0.93
			Fake	0.98	0.97	0.98
SWIN	F=10 BS:8	93.85%	Real	0.90	0.85	0.87
			Fake	0.95	0.97	0.96
	F=20 BS:4	75.19%	Real	0.51	0.45	0.48
			Fake	0.81	0.85	0.84

REFERENCES

[1] Y. Dang, Z. Zhang, W. Liu, and H. Jia, "On the Detection of Digital Face Manipulation," in *Proc. 2022 IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pp. 4652–4661, 2022.

[2] Y. Zhang, L. Ma, X. Li, and S. Wang, "Multi-modal Multi-scale Transformers for Deepfake Detection," in *Proc. 2022 European Conf. Computer Vision (ECCV)*, pp. 123–139, 2022.

[3] S. Kumar, A. Verma, and R. Sharma, "Combining EfficientNet and Vision Transformers for Video Deepfake Detection," in *Proc. 2022 Int. Conf. Pattern Recognition (ICPR)*, pp. 2847–2854, 2022.

[4] Li, Y., Yang, X., Sun, P., Qi, H., Lyu, S. (2020). Celeb-DF: A Large-scale Chal-lenging Dataset for DeepFake Forensics. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition.

[5] OpenCV Team. (n.d.). OpenCV: Open Source Computer Vision Library. Retrieved from https://opencv.org/.

[6] Torchvision. (n.d.). Torchvision: Datasets, Transforms and Models for PyTorch. Retrieved from https://github.com/pytorch/vision.

[7] Karandikar, A., Thakare, Y., Sah, O., Sah, R. K., Nafde, S., Kumar, S. (2023). Detection of Deepfake Video Using Residual Neural Network and Long Short-Term Memory. International Journal of Next-Generation Computing. https://doi.org/10.47164/ijngc.v14i1.1046

[8] Abid, A., Abdelaal, M., Muhammad, N., Choi, J. H., Mustahsan, Z., Alfozan, A.(2019). Gradio: Hassle-free sharing and testing of ML models in the wild. Retrieved from https://github.com/gradio-app/gradio

[9] Hochreiter, S., Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780.

[10] Rössler, A., Cozzolino, D., Verdoliva, L., Riess, C., Thies, J., Nießner, M. (2019) FaceForensics++: Learning to Detect Manipulated Facial Images. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition

[11] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T.Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.(2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. InAdvances in Neural Information Processing Systems 32 (pp. 8024–8035). Retrieved from https://pytorch.org.

[12] Zhang, Kaipeng and Zhang, Zhanpeng and Li, Zhifeng and Qiao, Yu (2016). Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks, IEEE Signal Processing Letters

[13] Saining Xie and Ross Girshick and Piotr Dollár and Zhuowen Tu and Kaiming He (2017).Aggregated Residual Transformations for Deep Neural Networks. arXiv eprint 1611.05431

[14] Xin Yang and Yuezun Li and Siwei Lyu (2018). Exposing Deep Fakes Using Inconsistent Head Poses. arXiv eprint 1811.00661

[15] Mingxing Tan and Quoc V. Le (2020). EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks arXiv eprint 1905.11946

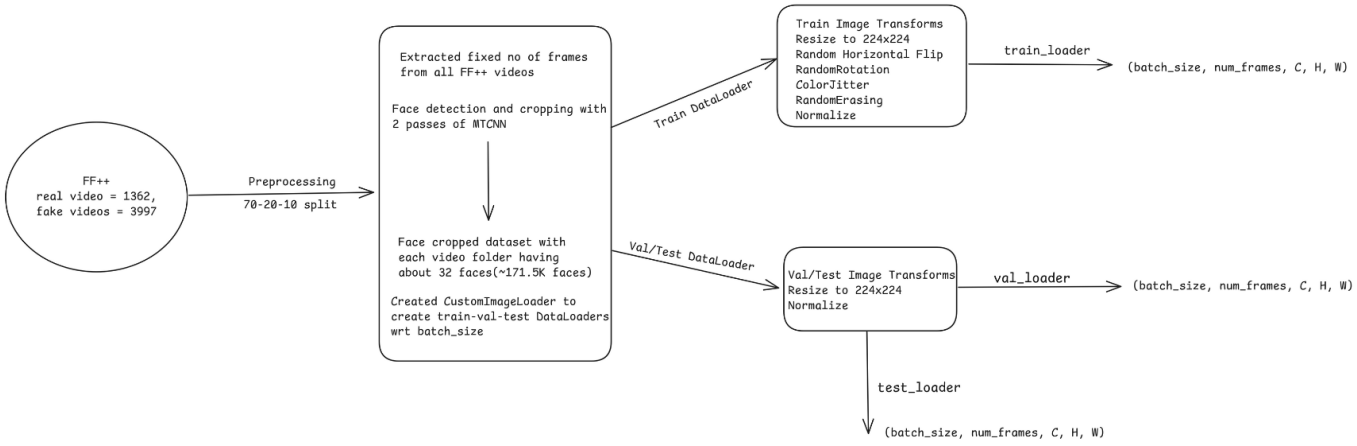

Fig. 8. Preprocessing flowchart

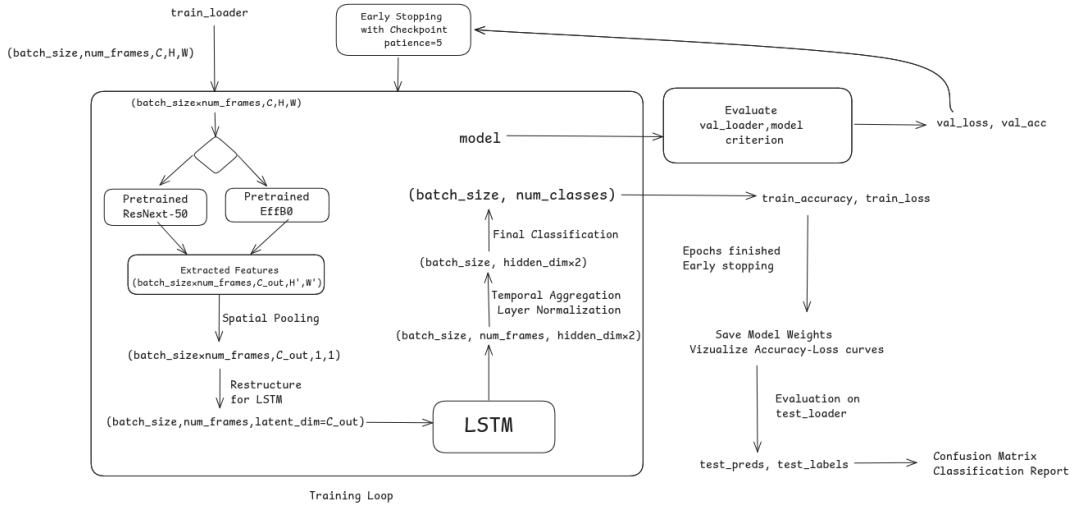

Fig. 9. Training flowchart for CNN+LSTM model

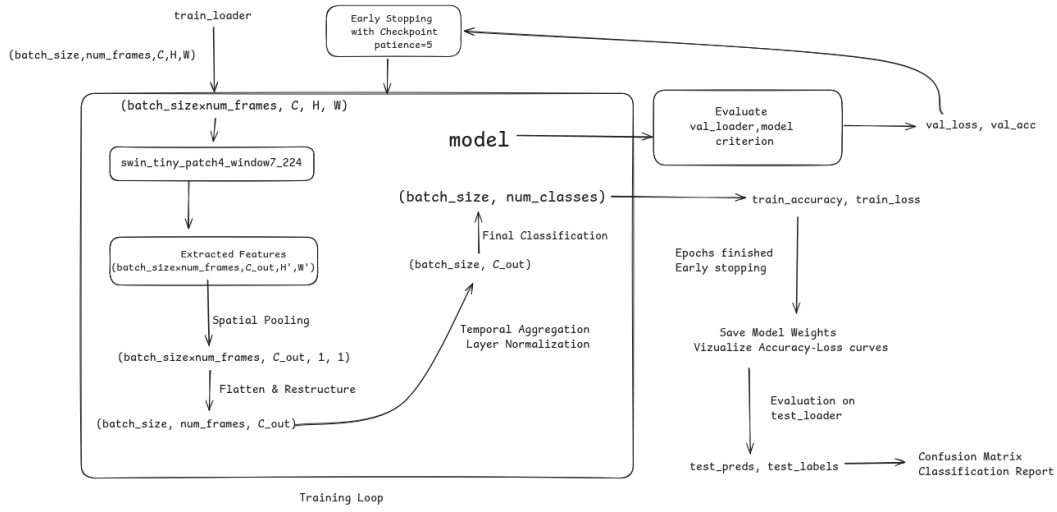

Fig. 10. Training flowchart for SWIN Transformer

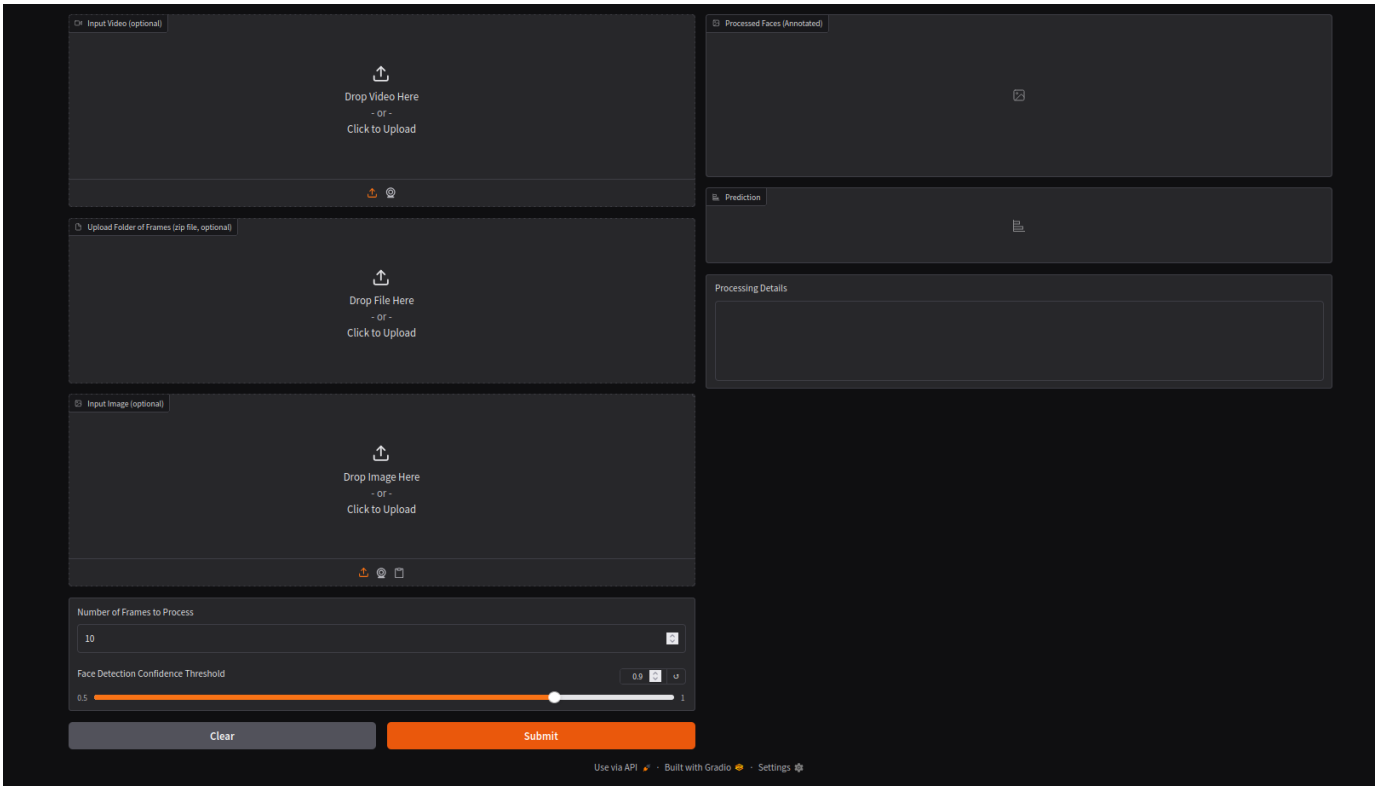

Fig. 11. UI for Deepfake detection

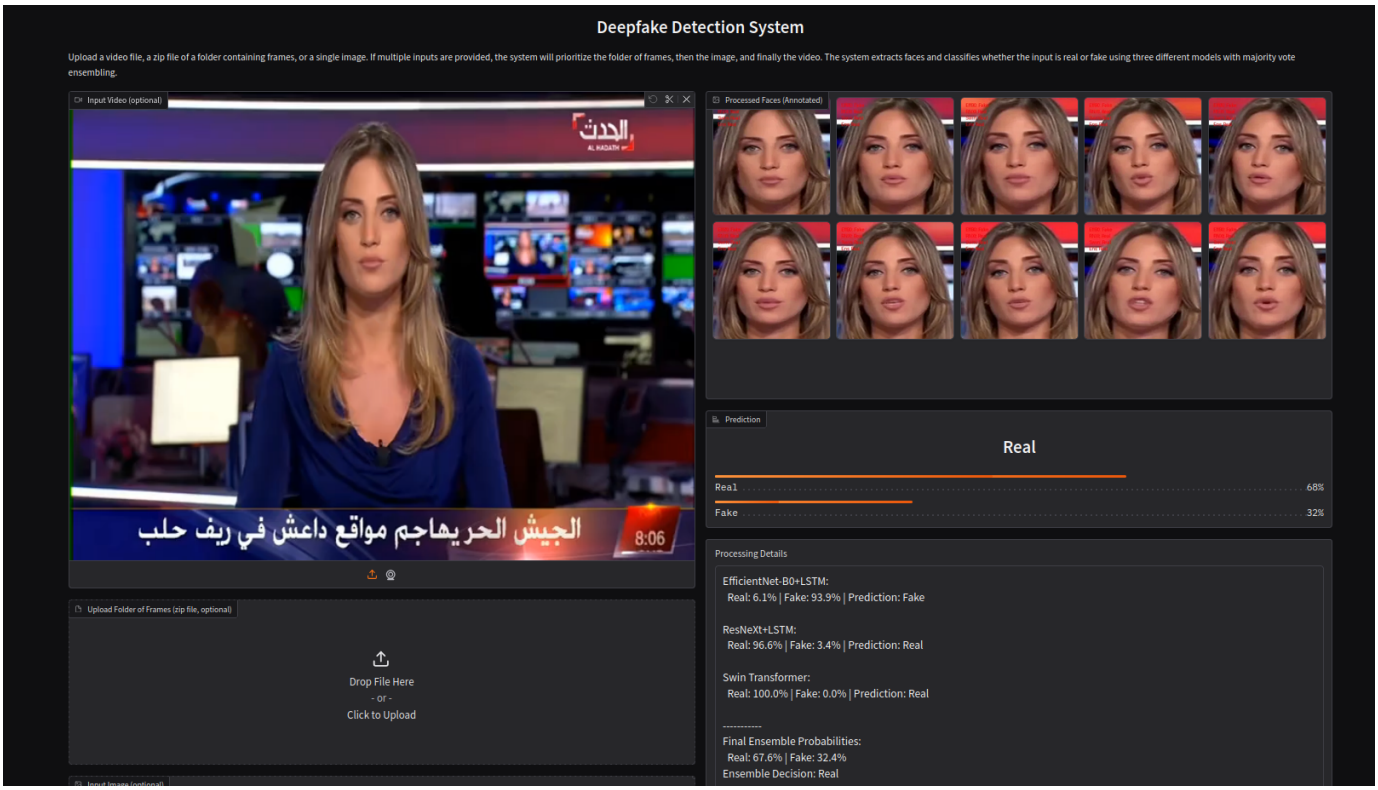

Fig. 12. UI demo for Video Input