

Documentation

DeepFake Recognition

The Sentinels



Index

1. Introduction
2. User Manual
3. Operational Manual
4. API Manual
5. Model Architecture
6. Use Cases
7. Technology Stack
8. Requirements
9. Future Scope
10. Conclusion



Introduction

Our Team:

Our team “The Sentinels” consists of 6 members. Sakshi Doshi(Leader), Parag Ghorpade, Varun Irani, Hrishikesh Mane, Akash Lende and Richa Maurya. We are a group of students pursuing(pursued) Bachelor in Engineering from Modern Education Society’s College of Engineering, Pune.

The problem we aim to solve through this project:

Real-time image processing and forensic verification of Fake Videos:- Cyber
Criminals are using Image processing tools and techniques for producing a variety of crimes, including Image Modification and fabrication using Cheap and deep Fake Videos/Images.
Desired Solution: The solution should focus on helping the Image/Video verifier/examiner find out and differentiate a fabricated Image/Video from an original one. Technology that can help address the issue: AI/ML techniques can be used.



User Manual

Accessing services through the website:

The user can access the website and click on the classify tab on the menu. Once that is done, he can either select the video by browsing or dragging and dropping the file. Then he should wait until the video is classified. Once the video is classified, the user will get a push notification to inform him that the file is done processing. He will get his result in a downloadable pdf format.

Accessing services through the app:

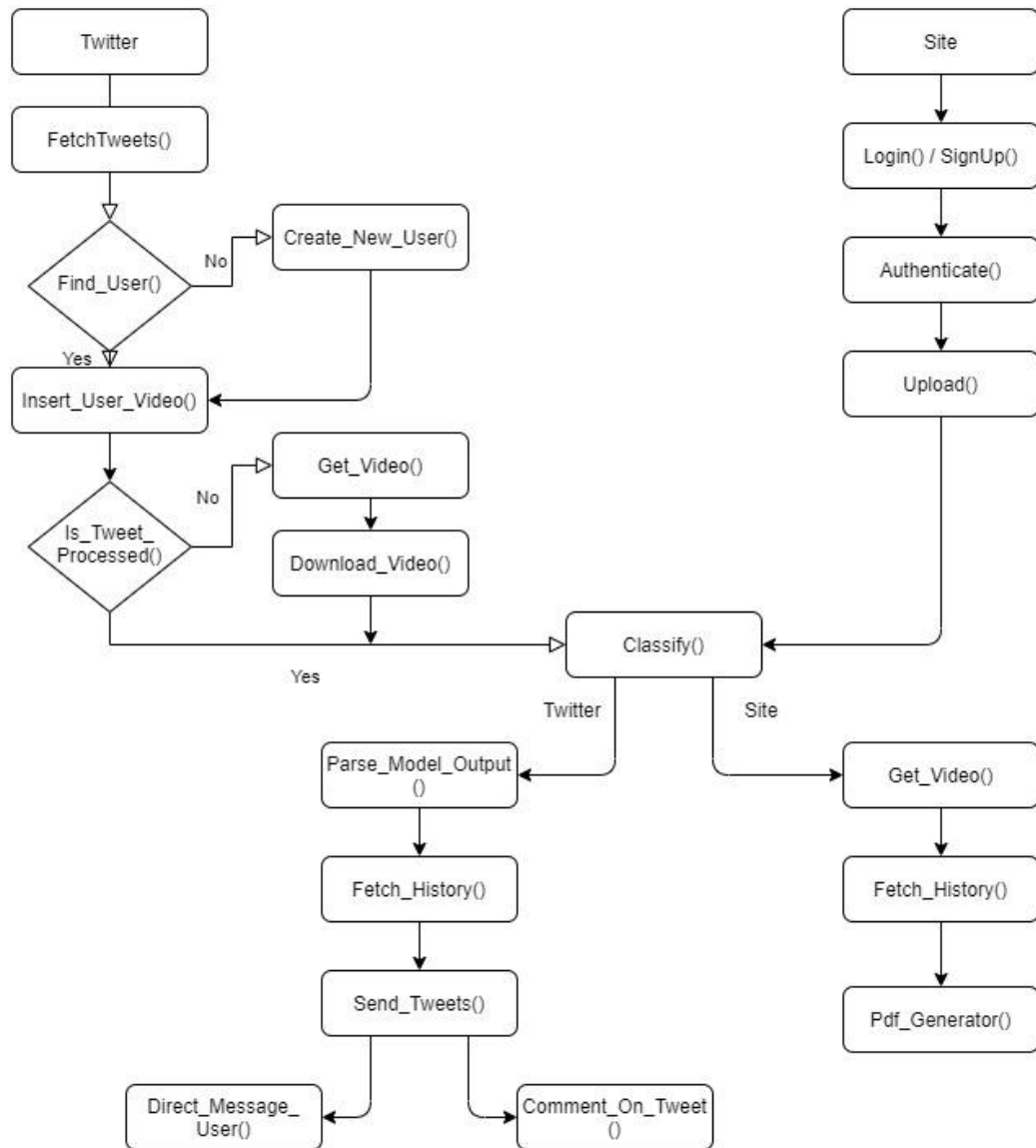
The user can access the website and click on the classify tab on the menu. Once that is done, he can either select the video by browsing or dragging and dropping the file. Then he should wait until the video is classified. Once the video is classified, the user will get a push notification to inform him that the file is done processing.

Using Twitter to classify:

To use this feature, the user need not upload the file to the app or the website. We have used the Twitter API to save time. If the user finds any video on Twitter suspicious, he can just retweet it by tagging us in the tweet (Username - @theSentinels_20). We will then process the video and the result will be sent to the user via Twitter direct messaging. A summary of the remaining API calls will also be given to the user.

Operational Manual

A flowchart depicting the operations performed in the project:





FetchTweets():

This function helps the program find the tweet when any user tags us (@theSentinels_20) in their tweet.

FindUser():

This function helps determine the program if the user who called the API by tagging us is a legitimate one or not.

Create_New_User():

Sometimes, if the user is not found we cannot just stop the code from running. Instead, we create a new user in the database with a new UserId and then pass on the video to the Insert_New_User function. This function is dependent on the FindUser() function.

Insert_User_Video():

This function is used to insert the video provided to us by the user through the FindUser() function directly or through Create_New_User(). This function also checks if the tweet exists instead of if the user exists. If it writes only on the basis of the user's existence, the new tweets of the user are not written in the user's collection.

Is_Tweet_Processed():

This function basically cross-checks if we have processed the tweet or not. If we have finished processing the tweet and downloaded the video, the video is further sent to the classify function. If it is not processed, the video goes through another set of functions and is then sent to the Classify() function.

Get_Video():

This function accesses the video from the tweet in which our handle is tagged.

Download_Video():

This function downloads the video from the path provided to it by the get_Video() function. This function is dependent on the Get_Video() function.

Classify():

This function calls the trained model made in Python and classifies if the media is deepfake or not. This function also checks if the max media size limit i.e. 100 MB is not exceeded. If the media is a video, this function also checks if the max fps (frames per second) is not above 30.



Parse_Model_Output():

This function parses the results if the media is fake or not which is given to us by the model. This result is passed to us by the express server. This function then sends the result to the Fetch_History_Twitter function.

Fetch_History_Twitter():

This function stores the results given to us by the Parse_Model_Output() function and thus keeps a record of all the media which was given for classification before. Thus, it maintains the history of the user.

Send_Tweets():

This function gets the entry from the Fetch_History_Twitter() function and sends this to the Direct_Message_User() function or the Comment_On_Tweet() function.

Direct_Message_User():

This function is used to direct the message to the user. A direct message can be sent to the user if his Classification limit is over or if the media in the tweet does not match the desirable configuration.

Comment_On_Tweet():

This function is used when the classification of the media mentioned in the tweet is successful. Once it is done, the results are then posted on Twitter in the form of a comment on the original tweet.

Login() / SignUp():

This API endpoint will help the user to log in or sign up with a new account on the website or the mobile application.

Authenticate():

This function authenticates if the details entered by the user on the Login/ Signup page are correct or not. If the credentials are correct, this function gives the user access to the website.

Upload():

This API endpoint helps the user who has logged on directly via the website to upload his media. This media will then be sent to the classify() function. Hence, the result will then be sent to the front end again.



GetVideo():

While the video is being processed by the Python backend, we also send the video to the front end and stream it real time on the screen. Hence the GetVideo() functions help us in doing this.

Fetch_History():

Once the video is done being classified, the result will be displayed in the History tab in the menu. Hence, the Fetch_History() function fetches the history from the GetVideo() function and displays it on the History tab. This function is dependent on the GetVideo() function.

Pdf_Generator():

The result in the History tab will be generated in a downloadable pdf format. Hence, for doing so, the Pdf_Generator() function is used. This pdf contains information about the media such as UserID, VideoID, File Checksum, Confidence, Duration, Size, Average fps(for video) and real-to-fake ratio.

API Manual

1. POST Sign-Up:

localhost:3000/users/signup

Body raw

```
{
  "username": "Akash",
  "email": "akash.lende12@gmail.com",
  "password": "12345678"
}
```

Example request:

```
curl --location --request POST 'localhost:3000/users/signup' \
--data-raw '{
  "username": "Akash",
  "email": "akash.lende12@gmail.com",
  "password": "12345678" }'
```

Example response:

```
{
  "status": "Registration Successful!",
  "success": true
}
```

2. POST Login:

localhost:3000/users/login

BODY raw


```
{
  "username": "Akash",
  "email": "akash.lende12@gmail.com",
  "password": "12345678"
}
```

Example request:

```
curl --location --request POST 'localhost:3000/users/login' \
--data-raw '{
"username": "Akash",
"email": "akash.lende12@gmail.com",
"password": "12345678" }'
```

Example response:

```
{
  "status": "Login Successful!",
  "success": true,
  "token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJfaWQiOiI1ZjEyZjQzNmY4NWMzMzMwMzBlYTMyY2EiLCJpYXQiOiJlOTUwNzgyNjQsImV4cCI6MTU5NTUxMDI2NH0.tZTUFTraQ5MaziOrDy1edKZKbm
bMc_lE0cuXN1R2Z6Q",
```

```
  
    "id": "5f12f436f85c333030ea30ca",  
    "name": "Akash"  
}
```

3. POST Classify Video

localhost:3000/classify

AUTHORIZATION: Bearer Token

BODY formdata

video

userId

5f0ec570dc8e2b3f885b2bd6

Example request:

```
curl --location --request POST 'localhost:3000/classify' \  
--form 'video=@/C:/wamp64/www/SIH2020/Express  
Server/video-cache/video-0fc3b28a.mp4' \ --form  
'userId=5f0ec570dc8e2b3f885b2bd6'
```

Example response:

```
{  
    "code": 200,  
    "message": "Video is being processed "  
}
```



```
{
  {
    "code": 200,
    "message": "User found",
    "data": {
      "id": "5f12f436f85c333030ea30ca",
      "vid": [
        {
          "feedback": "",
          "tweetId": "",
          "_id": "5f12f98cf85c333030ea30cb"
        }
      ],
      "img": [],
      "remaining": 7
    },
    "vdata": [
      {
        "fileChecksum": "f76f8c4d2eceb6d8079e5cb18ae81839",
        "timestamps": [],
        "duration": 45.037622,
        "bitrate": 1080,
        "fileSize": 3413818,
        "_id": "5f12f98cf85c333030ea30cb",
        "fileName": "video-2ea48ce3.mp4",
```

```

    "filePath": "video-results\\video\\video-393167a9.mp4",
    "videoExists": "true",
    "timeToProcess": 573331.9449010044,
    "confidence": "0.81",
    "realToFakeRatio": 34,
    "status": "FAKE",
    "createdAt": "2020-07-18T13:30:52.669Z",
    "updatedAt": "2020-07-18T13:30:52.669Z",
    "__v": 0
  }
],
  "idata": []
}
}
```

5. GET Fetch PDF for Video:

localhost:3000/pdf?userId=5f0ec570dc8e2b3f885b2bd6&videoId=5f1150710aed29705c63992d

AUTHORIZATION: Bearer Token

Example request:

```
curl --location --request GET
'localhost:3000/pdf?userId=5f0ec570dc8e2b3f885b2bd6&videoId=5f1150710aed
```

Example response:

```
{
```

```
"report": "5f12f98cf85c333030ea30cb.pdf"
}
```

Error Codes	Description
403	Video doesn't belong to the user
404	Video not found
404	User not found

6. POST Remove video:

http://52.225.219.201:3000/remove

AUTHORIZATION: Bearer Token

BODY raw

```
{
  "userId": "5f0b219e2181a238c8ca8baa",
  "videoId": "5f08fa6569cca212947e2bfa"
}
```

Example request:

```
curl --location --request POST 'http://52.225.219.201:3000/remove' \
--data-raw '{ "userId": "5f0b219e2181a238c8ca8baa", "videoId":
"5f08fa6569cca212947e2bfa" }'
```

Example response:

```
{
  "code": 200,
  "message": "deletion successful",
  "success": true
}
```

7. POST Classify Image:

http://localhost:3000/get-image/

AUTHORIZATION: Bearer Token

BODY form-data

userId

5f144f6fd60cde6ad8c54fd1

image

Example request: curl --location --request
POST'http://localhost:3000/get-image/' \
--form 'userId=5f144f6fd60cde6ad8c54fd1' \
'image=@/C:/Users/Akash/Pictures/VideoCapture_20200616-183141.jpg'

Example response:

```
{  
  "code": 200,  
  "message": "Video is being processed "  
}
```

Error codes	Description
412	Size of the file exceeded max permissible size 100MB

413	Video length should not exceed 1 minute
422	No video codec found
429	Rate limit exceeded

8. GET Classified Image File Name:

localhost:3000/get-image?userId=5f0ec570dc8e2b3f885b2bd6&imageId=5f1328d1f85c333030ea30cd

AUTHORIZATION: Bearer Token

PARAMS

userId

5f144f6fd60cde6ad8c54fd1

imageId

5f1328d1f85c333030ea30cd

Example request:

```
curl --location --request
```

```
GET 'localhost:3000/get-image?userId=5f0ec570dc8e2b3f885b2bd6&imageId=5f1328
```

9. GET Fetch Labelled Image:

http://localhost:3000/get-image/image?imageFile=image-09ed1670e.jpg

AUTHORIZATION: Bearer Token

PARAMS

imageFile

Image-09ed1670e.jpg

Example request:

```
curl --location --request  
GET 'http://localhost:3000/get-image/image?imageFile=image-09ed1670e.jpg'
```

10. POST Remove Image:

http://localhost:3000/remove/image

AUTHORIZATION: Bearer Token

BODY urlencoded

imageFile image-09ed1670e.jpg

imageId 5f1452fb095bf920b037d720

userId 5f144f6fd60cde6ad8c54fd1

Example request:

```
curl --location --request POST 'http://localhost:3000/remove/image' \  
--data-urlencode 'imageId=5f1452fb095bf920b037d720' \  
--data-urlencode 'userId=5f144f6fd60cde6ad8c54fd1'
```

Example response:

```
{  
  "code": 200,  
}
```

```
"message": "deletion successful",  
"success": true  
}
```

11. GET Fetch PDF for Image:

http://localhost:3000/pdf-image?imageId=5f145343c04d3d2c5c7d1c65&userId=5f144f6fd60cde6ad8c54fd1

AUTHORIZATION: Bearer Token

PARAMS

imageId 5f145343c04d3d2c5c7d1c65

userId 5f144f6fd60cde6ad8c54fd1

Example request:

```
curl --location --request
```

```
GET 'http://localhost:3000/pdf-image?imageId=5f145343c04d3d2c5c7d1c65&userId
```

Example Response:

```
{  
  "report": "5f12f98cf85c333030ea30cb.pdf"}  
}
```

Error Codes	Description
403	Video doesn't belong to the user
404	Video not found
404	User not found

Model Architecture

Solution Description:

In general, the solution is based on frame-by-frame classification approach.

Face-Detector

MTCNN detector is chosen due to kernel time limits. It would be better to use the S3FD detector as more precise and robust, but open-source Pytorch implementations don't have a license.

The input size for the face detector was calculated for each video depending on video resolution.

- 2x scale for videos with less than 300 pixels wider side
- No rescale for videos with wider side between 300 and 1000
- 0.5x scale for videos with wider side > 1000 pixels
- 0.33x scale for videos with wider side > 1900 pixels

Input size

As soon as we discovered that EfficientNets significantly outperform other encoders we used only them in our solution. As we started with B4 we decided to use "native" size for that network (380x380). Due to memory constraints, we did not increase input size even for B7 encoder.

Margin

When we generated crops for training we added 30% of face crop size from each side and used only this setting during the competition. See `extract_crops.py` for the details.

Encoders

The winning encoder is the current state-of-the-art model (EfficientNet B7) pre-trained with ImageNet and noisy student Self-training with Noisy Student improves ImageNet classification

Averaging predictions

I used 32 frames for each video. For each model output instead of simple averaging, I used the following heuristic which worked quite well on the public leaderboard (0.25 -> 0.22 solo B5).

```
import numpy as np

def confident_strategy(pred, t=0.8):

    pred = np.array(pred)

    sz = len(pred)

    fakes = np.count_nonzero(pred > t)

    # 11 frames are detected as fakes with high probability

    if fakes > sz // 2.5 and fakes > 11:

        return np.mean(pred[pred > t])

    elif np.count_nonzero(pred < 0.2) > 0.9 * sz:

        return np.mean(pred[pred < 0.2])

    else:

        return np.mean(pred)
```

Augmentations

I used heavy augmentations by default. [Albumentations](#) library supports most of the augmentations out of the box. Only needed to add IsotropicResize augmentation.

```
def create_train_transforms(size=300):

    return Compose([

        ImageCompression(quality_lower=60, quality_upper=100, p=0.5),

        GaussNoise(p=0.1),
```

```

GaussianBlur(blur_limit=3, p=0.05),

HorizontalFlip(),

OneOf([

    IsotropicResize(max_side=size, interpolation_down=cv2.INTER_AREA,
interpolation_up=cv2.INTER_CUBIC),

    IsotropicResize(max_side=size, interpolation_down=cv2.INTER_AREA,
interpolation_up=cv2.INTER_LINEAR),

    IsotropicResize(max_side=size,
interpolation_down=cv2.INTER_LINEAR, interpolation_up=cv2.INTER_LINEAR),

], p=1),

PadIfNeeded(min_height=size, min_width=size,
border_mode=cv2.BORDER_CONSTANT),

OneOf([RandomBrightnessContrast(), FancyPCA(), HueSaturationValue()],
p=0.7),

ToGray(p=0.2),

ShiftScaleRotate(shift_limit=0.1, scale_limit=0.2, rotate_limit=10,
border_mode=cv2.BORDER_CONSTANT, p=0.5),

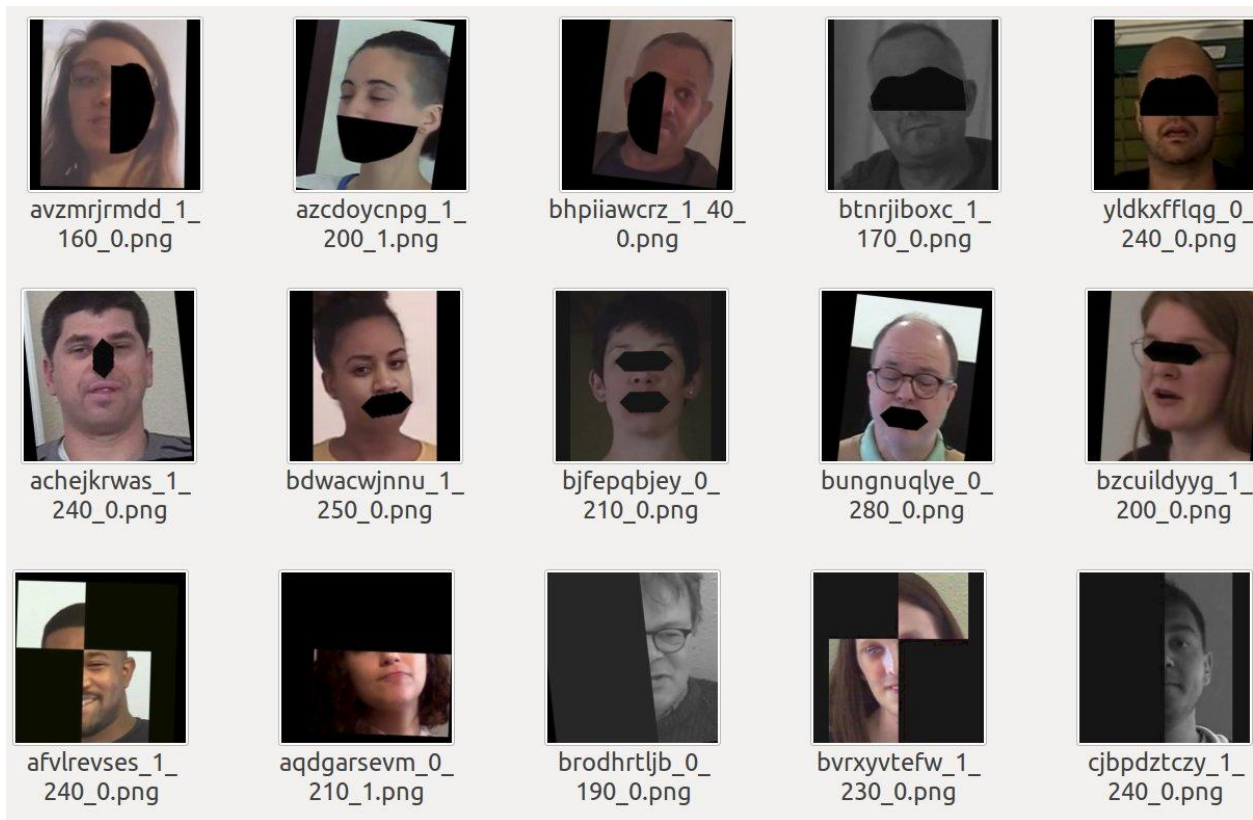
]

)

```

In addition to these augmentations, I wanted to achieve better generalization with

- Cutout like augmentations (dropping artefacts and parts of the face)
- Dropout part of the image, inspired by [GridMask](#) and [Severstal Winning Solution](#)



Data preparation

Once DFDC dataset is downloaded all the scripts expect to have dfdc_train_xxx folders under the data root directory.

Preprocessing is done in a single script preprocess_data.sh which requires dataset directory as the first argument. It will execute the steps below:

1. Find face bboxes

To extract face bboxes I used facenet library, basically only MTCNN. python preprocessing/detect_original_faces.py --root-dir DATA_ROOT This script will detect faces in real videos and store them as jsons in DATA_ROOT/bboxes directory

2. Extract crops from videos

To extract image crops I used bboxes saved before. It will use bounding boxes from original videos for face videos as well. python preprocessing/extract_crops.py --root-dir DATA_ROOT

--crops-dir crops This script will extract face crops from videos and save them in DATA_ROOT/crops directory

3. Generate landmarks

From the saved crops it is quite fast to process crops with MTCNN and extract landmarks python preprocessing/generate_landmarks.py --root-dir DATA_ROOT This script will extract landmarks and save them in DATA_ROOT/landmarks directory

4. Generate diff SSIM masks

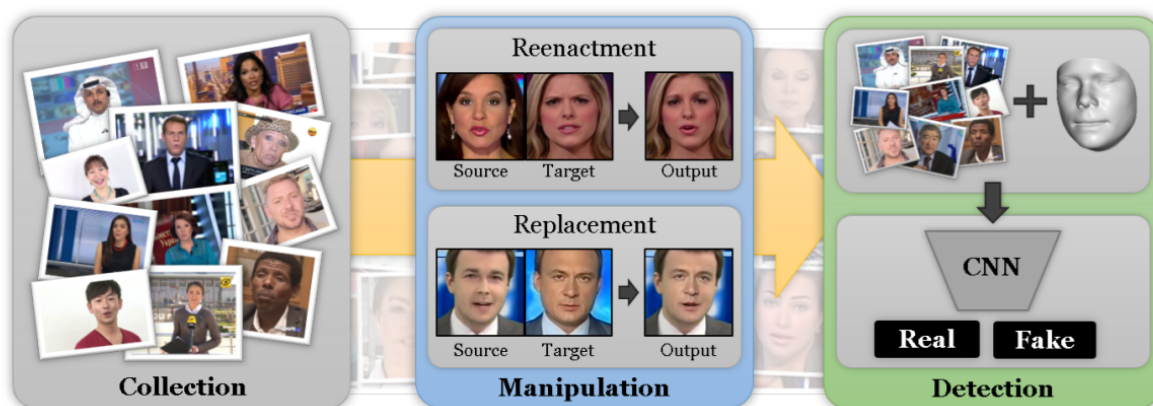
python preprocessing/generate_diffs.py --root-dir DATA_ROOT This script will extract SSIM difference masks between real and fake images and save them in DATA_ROOT/diffs directory

5. Generate folds

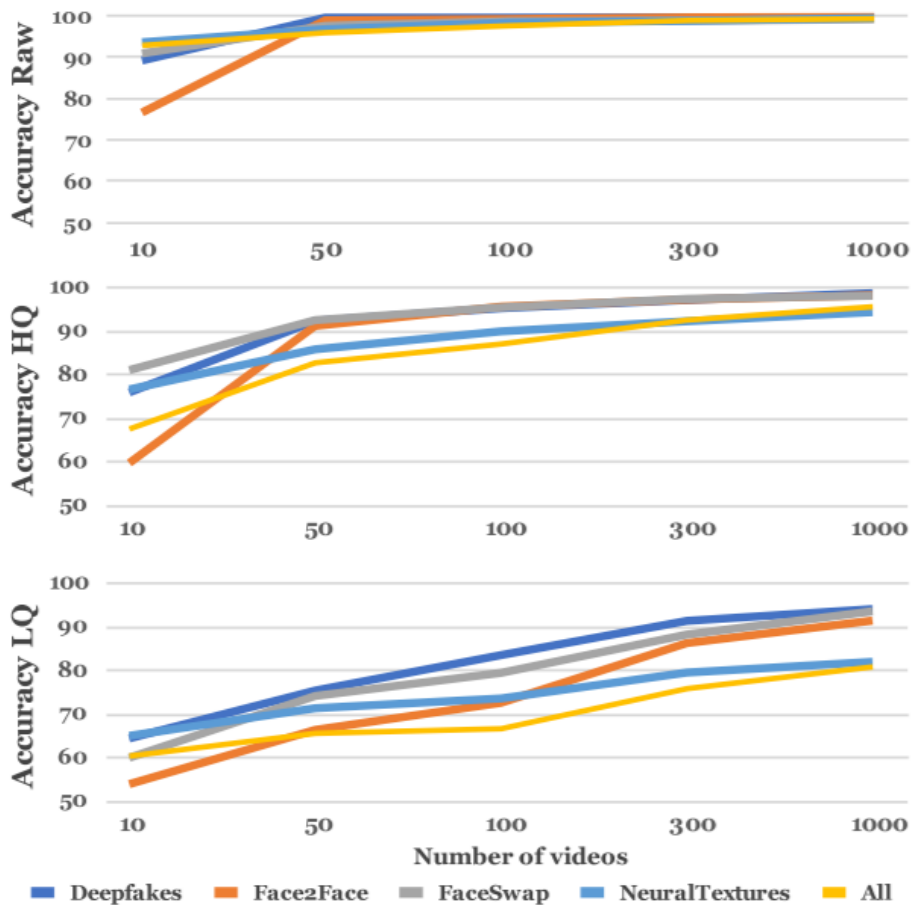
python preprocessing/generate_folds.py --root-dir DATA_ROOT --out folds.csv By default it will use 16 splits to have 0-2 folders as a holdout set. Though only 400 videos can be used for validation as well.

Training

Training 5 B7 models with different seeds are done in the train.sh script. During training, checkpoints are saved for every epoch.



Plotting losses to select checkpoints



Inference

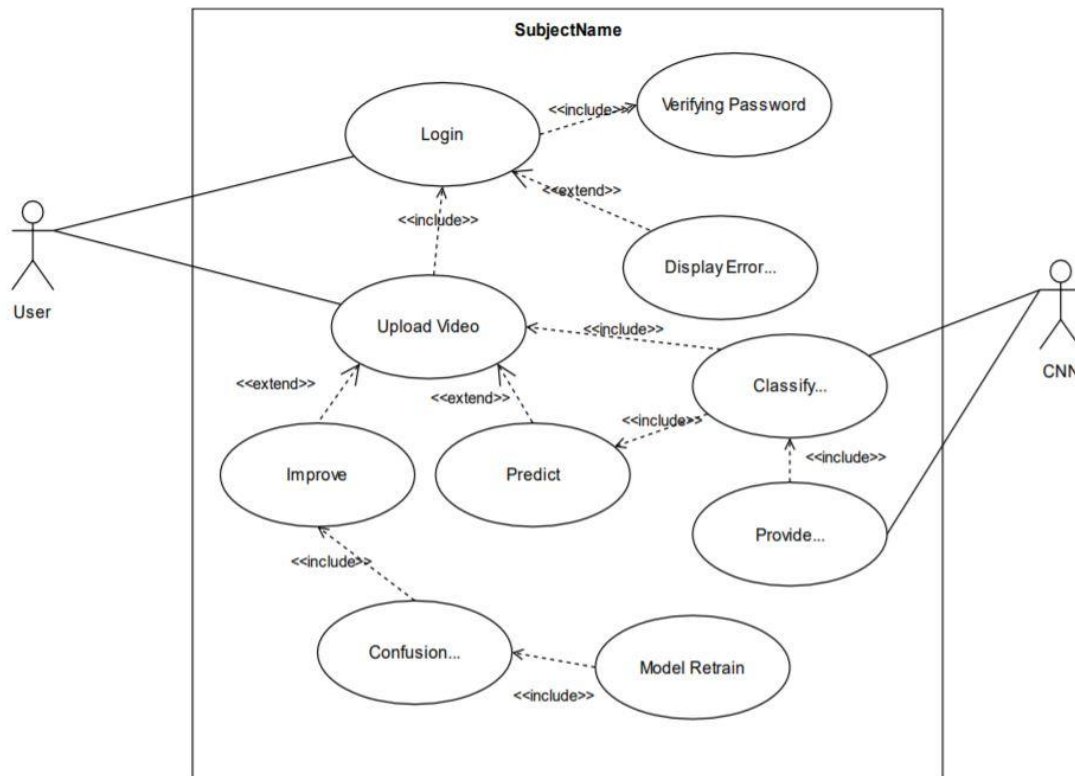
The kernel is reproduced with predict_folder.py script.

Pretrained models

download_weights.sh script will download trained models to weights/ folder. Ensemble inference is already preconfigured with predict_submission.sh bash script. It expects a directory with videos as the first argument and an output CSV file as the second argument.

For example `./predict_submission.sh /mnt/datasets/deepfake/test_videos submission.csv`

Use Cases



Fraud:

Deepfakes have been used as part of social engineering scams, fooling people into thinking they are receiving instructions from a trusted individual. Our idea facilitates detection of these highly intricate patterns and provides instant feedback system for the prevention of misleading information.

Credibility and authenticity:

Though fake photos have long been plentiful, faking motion pictures have been more difficult, and the presence of deepfakes increases the difficulty of classifying videos as genuine or not. Further scope of our project extends to alerting the person/entity involved in the forged video.



Technology Stack

Pytorch / Keras / OpenCV:

PyTorch is an open-source machine learning library based on the Torch library, used for applications such as computer vision and natural language processing.

Keras is an open-source neural network library written in Python. It is capable of running on top of Pytorch. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible.

OpenCV is a library of programming functions mainly aimed at real-time computer vision. These are the libraries used for making the model in Python.

Python:

Python is an interpreted, high-level, general-purpose programming language. Python is the language which we have used in training our entire model with the help of the above-mentioned libraries.

Express.js:

Express.js is a Node.js framework. This framework allows us to host our server and thus establish communications between the frontend, backend and the database.

MongoDB:

MongoDB is a cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. We have used MongoDB to store User Information, information related to the API Limits and so on.

ReactJS:

React is an open-source JavaScript library for building user interfaces. In our project, we have used ReactJS as a frontend for our website.

Flutter:

Flutter is an open-source UI software development kit. It used to make apps for Android as well as iOS. Hence, flutter gives us the ease of using a single code base for both the popular platforms. In our project, we have used flutter for making our mobile app.



Requirements

Software Requirements:

1. Express (Node.js) Server
2. Knowledge of languages such as Python, Javascript and MongoDB
3. Knowledge of libraries and frameworks such as ReactJS, ExpressJS, Flutter, OpenCV, Keras, Pytorch.
4. Chromium supported Browser

Hardware Requirements:

1. Nvidia CUDA parallel computing platform
2. 8GM RAM (Minimum)
3. GPUs

Future Scope

DeepFake media is going to be a great part of cybercrime in the near future. With the availability of such threatful codes directly on platforms like Github, Youtube, etc. it will not be long when normal people will be bombarded with incorrect information through various social media platforms. DeepFakes can make an individual bankrupt, or even make him lose his dignity. It is very important for everybody to understand the gravitas of DeepFake media.

Hence, our team, The Sentinels have come up with an intuitive solution to this huge problem. By using Deep Learning, we can detect the highly intricate patterns and provide instant feedback system for the prevention of misleading information.

This software can be used via the website or the app which will be available in Google / iOS store. We see this software as a necessity rather than want. An application with State of the Art classification along with ready-to-use implementation features is something we all need especially the Police.

In the coming decade, we visualize this software to be a huge part of Police research and development for it has the potential of lessening down the nonsense threats and fake authenticity of a criminal. We see this software solving huge problems which shouldn't have occurred in the first place, like an incident from India in February when one political party used DeepFakes to brainwash the local people to not make them vote for the opposing party.

As of now, there is no explicit law banning DeepFakes. Hence, there is nothing stopping such cheap criminals. We need this software now more than any other time. It is our motive to eradicate DeepFakes from their source. As of now, we are capable of doing that from Twitter, but those days aren't far enough where we are able to do that from every social media possible. We hope this software is useful for everyone, especially our country and the Police as we cannot see any more people suffer from this dangerous application of a promising technology.

Conclusion

DeepFakes can be used in a good way or in a bad way. It is completely up to us how we decide to use it. After completing this project, in conclusion, we can say that deepfakes are not just shaming the individual but they're also torturing him mentally. Crimes made using deepfakes should be banned at all costs. Our recognizer can prove to be a new source of help to all the policemen and individuals who want to eradicate this type of cybercrime.

With the amazing accessibility provided by us, anyone can easily use our platform just in case they come across a deepfake. Women being a major victim to such videos, can easily use our app to clarify if the given video is a deepfake or not, And if it is, they can directly report it to the police officer. Our app and website also support many regional languages. Hence if anyone doesn't really understand English, they can switch to their local languages such as Marathi, Hindi, Bengali, Malayalam, Telugu, etc.

We have also given social media support to our app. Hence, if the user feels like there is a video on Twitter that they really need to classify they can just retweet it by mentioning our handle "@TheSentinels_" in the tweet. We will send the user the result in the comment section and then also send the remaining necessary details in their direct messages. Similarly, the user can do it for YouTube as well. All they have to do is copy the link of the video and paste it on our website. We will then do the remaining part for you.

Deepfakes are something that needs to be detected as it can steal your identity without you even knowing it. By making this detector, we hope we help to make society a better place. We also hope that we can help the victims of such heinous crimes and provide justice to the needful