Senior Project Final Report

# Introduction and Problem Identification

In today's digital age, the rise of deepfakes has become a growing concern. These manipulated videos or images, created using machine learning algorithms, have the potential to harm individuals and society by spreading false information and undermining trust in journalism and media. With deepfakes becoming increasingly sophisticated, it is becoming more difficult to differentiate between what is real and fake. The project aims to develop a "Deepfake detection" model that utilizes artificial intelligence to accurately identify and detect deepfakes.

It is crucial to preserve trust in digital media, safeguard reputations, and protect individuals from harm. This work seeks to improve the problem of deepfake manipulation and provide a reliable method to detect deepfakes. The target audience includes anyone who consumes digital media, including individuals, organizations, and media outlets.

The project will involve various stages, including researching and understanding deepfake technology, selecting and preparing a video dataset, training a deep learning model to identify manipulated content, and testing the accuracy of the model. The decision to undertake this project was made to address the growing concerns around deepfakes and provide a reliable solution to the problem of misinformation in digital media.

# Project Objectives and Modifications

The aim of our project is to develop a "deep fake detection" application that utilizes computer vision and machine learning to identify deep fakes. Initially, we planned to create a solution that could detect both audio and video deepfakes, but due to time constraints, we focused solely on detecting video manipulation. To make our project unique, we used a large dataset and evaluated our model with high accuracy and prediction.

To detect deepfakes, we employed Res-Next convolutional neural networks to extract frame-level features from videos. These features were then used to train a long-short-term memory (LSTM)-based recurrent neural network (RNN) to determine whether a video has been manipulated or not. We tested our approach on a variety of deep-fake videos obtained from various websites, and we aimed to improve the model's performance on real-time data. To achieve this, we trained our model on a combination of available datasets, enabling it to learn features from different types of images. We collected a sufficient number of videos from datasets like the Deepfake Detection Challenge, and we also evaluated our model using a large volume of real-time data, including YouTube datasets, to achieve competitive results in real-world scenarios.

1. Top of Form

|  |  |  |
| --- | --- | --- |
| **Original and Modified Objectives** | **Delivered?** | **Explanatory Notes** |
| Audio Extraction from video- Audio Signal Processing | NO | We decided to prioritize the development of a model that could identify video deep fakes with a high degree of accuracy, rather than incorporating audio into the analysis. This was due to the significant time it took to process the videos alone. Our goal was to achieve a prediction confidence level of at least 95%. |
| Data Collection | Yes | We utilized the DFDC data set from the Deep Fake Challenge to develop our accurate model. The data set encompasses different characteristics, such as age and ethnicity, which were crucial in the creation of our model. |
| Data Pre-Processing | Yes | In this stage, several libraries such as cv2 were used to preprocess the DFDC dataset. Specifically, it reads all video files in a given directory, checks if each video has at least 150 frames, and removes any corrupted videos. Additionally, it extracts faces from the remaining videos using face\_recognition library and saves them in a new directory. |
| Training the Model data set | Yes | We used Google Collab pro to train the model. The programming language used to write the codes of the training model is python. The model is training based on some parameters (learning rate) to give the maximum accuracy. |
| Create model using python programming language | Yes | We have used python for almost every part of the project. For example, for data pre-processing and training the model. |
| Incorporate packages for the model to run | Yes | We downloaded many packages, for example torch, python, cmake for pytorch to work. This all was heavy so we change computer to higher computational GPU to train the model by using the following packages. |
| Front - end development (website) | Yes | We have used Django application to develop the user interface for testing the model. |
| Testing the whole application | Yes | A complete website that allows users to upload and receive their results was created. We used unit testing to test the complete application. |

# Description of the Delivered Product/Service

**3.1 Explanation**

There are numerous tools available for creating deep fakes, but few tools exist for detecting them. And those tools are mostly accessible by computer vision and machine learning expertise giving limited access to the public. A web-based platform for detecting deep fakes will be a valuable contribution to preventing the spread of deep fakes on the internet and enable users to upload a video to the platform and classify it as real or fake.

**3.2 Accomplishment**

**An LSTM** stands for Long Short-Term Memory, which is a type of artificial neural network that is commonly used for processing sequential data. An LSTM can analyze the sequential pattern of the video frames to identify any inconsistencies or anomalies such if there is a sudden change in lighting or the background looks unusual, an LSTM can recognize that these changes are not consistent with the previous frames of the video and raise an alert that the video may be a deepfake.

**ResNext CNN** is a specific type of CNN that has an improved accuracy in recognizing patterns. In this project it is used to analyze each frame of a video and identify specific features or patterns that may indicate whether the video is real or fake. These features are then fed into an LSTM, which uses them to make a prediction about the authenticity of the video.

At the start of the project, we analyzed how deep videos are created and came to the conclusion that deepfakes created using pre-trained neural network models are so convincing that it is almost impossible to detect them with the naked eye. However, the deepfake creation tools leave traces or artifacts such as wrinkles on faces, lighting, double chin, edges, and teeth enhancement. in the video that are imperceptible to the naked eye. And the main objective is for the model to correctly identify those inconsistencies.

We have divided the work of what we did throughout the project as follows:

**Module 1. Data Set Gathering**

Collecting data manually for this project, which requires a diverse set of individuals with varying ages, races, ethnicities, and skin tones, within the given time constraints was impractical. However, there are existing datasets that can be used, such as FaceForensic++, Deepfake detection challenge, and Celeb-DF. For this project, we used the dataset from the Deepfake detection challenge (DFDC), dataset was created by Facebook AI and academic partners, and it contains both face-swapping and facial reenactment types of deepfakes, as well as a diverse set of real videos from various sources of almost 6,000 videos. Based on a rough approximation, the dataset includes 74% female and 26% male participants, with 68% Caucasian, 20% African American, 9% East Asian, and 3% South Asian participants [1].

**Module 2. Data Pre-processing**

In this step, the videos are preprocessed to remove any unnecessary noise or frames and to crop with the focus only on the face. After preprocessing we took 4200 real video and 1300 fake videos as some were corrupted. The first step is to split each video into frames, so that each frame can be analyzed individually. Then, using a facial detection algorithm, the faces in each frame are identified and cropped as shown in figure 1. The cropped frames are then combined to create a new video with only the face visible. To ensure uniformity in the number of frames in each video, a threshold value is selected based on the mean of the total frames count of each video. This threshold value is chosen because of limited computational power - processing a video with a large number of frames can be computationally difficult. For example, a 10-second video at 30 frames per second (fps) would have a total of 300 frames, and it may not be possible to process all 300 frames at once in the experimental environment.

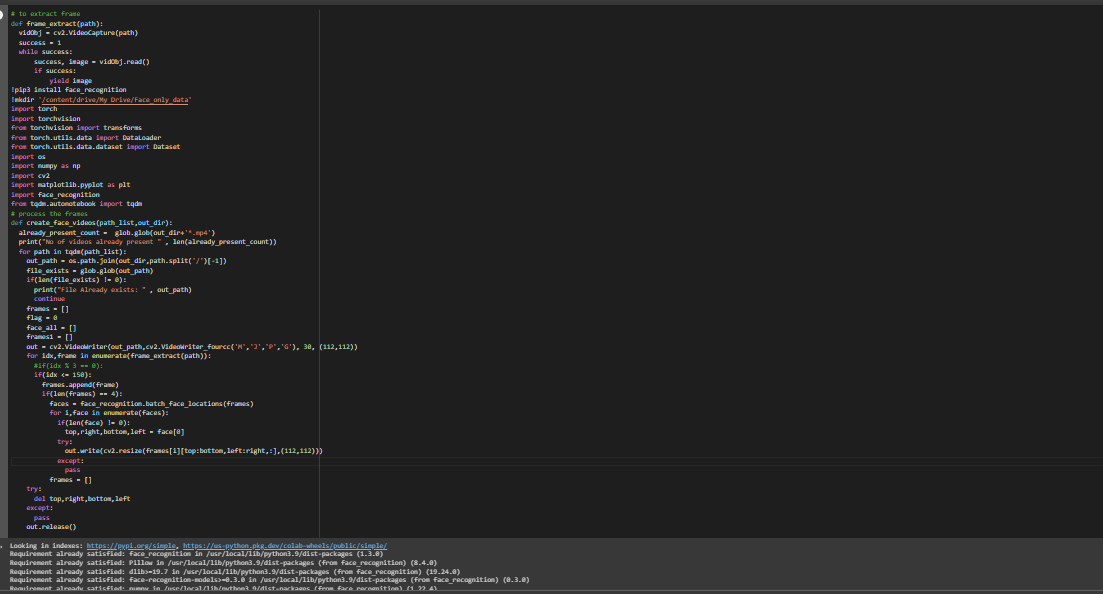
The code shown in Figure 3.1 selects a threshold value of 150 frames based on the GPU's computational power. When creating a new dataset, only the first 150 frames of each video are saved. This guarantees that the processing is feasible and there are enough frames for analysis. The code employs the OpenCV library to read and write video files and the face\_recognition library for identifying faces.

Figure 3.2.1: Data preprocessing

In order to effectively use Long Short-Term Memory (LSTM) for detecting deepfake videos, the frames of the video are analyzed in a sequential manner, starting from the first frame and proceeding in order. To facilitate this process, a new video is created that only includes the cropped face, which is saved at a frame rate of 30 frames per second and a resolution of 112 x 112. These videos are stored in a folder named "Face\_only\_data", as illustrated in the figure below.

Graphical user interface, text

Description automatically generated

Figure 3.2.2 A code snippet that shows where the pre-processed videos are saved.

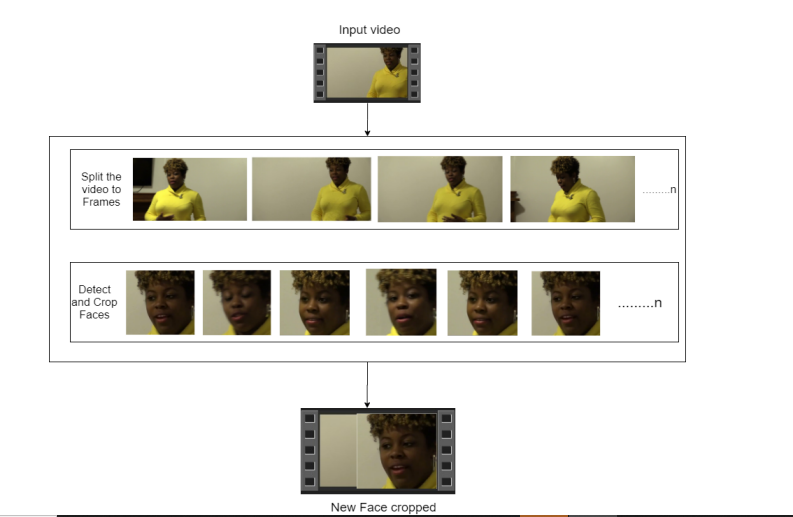


Fig 3.3: Pre-Processed face in a video

**Module 3: Data-Set Split**

The dataset is divided into two parts: a training dataset with 70% of the videos (4,200), and a testing dataset with 30% of the videos (1,300). The distribution of real and fake videos is equal in both sets, with each set containing 50% of each.

**Module 4: Model Architecture**

The most important steps while creating the model were:

**4.1 Data Pre-Processing continuation(model)**

Before the data is loaded, there was another data preprocessing that took place. In this code, the facial recognition library is installed, which is used for facial detection. The video files are read, and frames are extracted using OpenCV's "cv2.VideoCapture" function. Data transformations are then applied to each frame using PyTorch's "transforms" module. The resulting frames are converted into tensors and returned as output from the "validate\_video" function. Then frames extracted from the videos are normalized by subtracting the mean and dividing by the standard deviation. This helps to center the data and scale it appropriately, making it easier for the model to learn from. The number of frames extracted from each video is also checked to ensure that it is greater than or equal to 100. If it is less than 100, the video is considered corrupted and is removed from the list of video files.

Text

Description automatically generated

Text

Description automatically generated

Fig 3.2.4: Data pre-processing before the model is trained.

**4.2 Data Loading and Preparation**

Once the data is preprocessed, it needs to be loaded and prepared for use in the machine learning model. This involves creating a PyTorch dataset class that can read the video frames and labels from a CSV file. The dataset class also applies data transformations to the frames if specified. The video files are split into training and validation sets, and transformations are defined using PyTorch's "transforms" module. As shown in figure 5, there are some libraries used for making this happen, for example, it extracts frames from video files using OpenCV and applies a data transformation function (if specified). The "im\_plot" function displays an image using Matplotlib after converting the image from PyTorch tensor format to NumPy array format and applying some colour transformations.

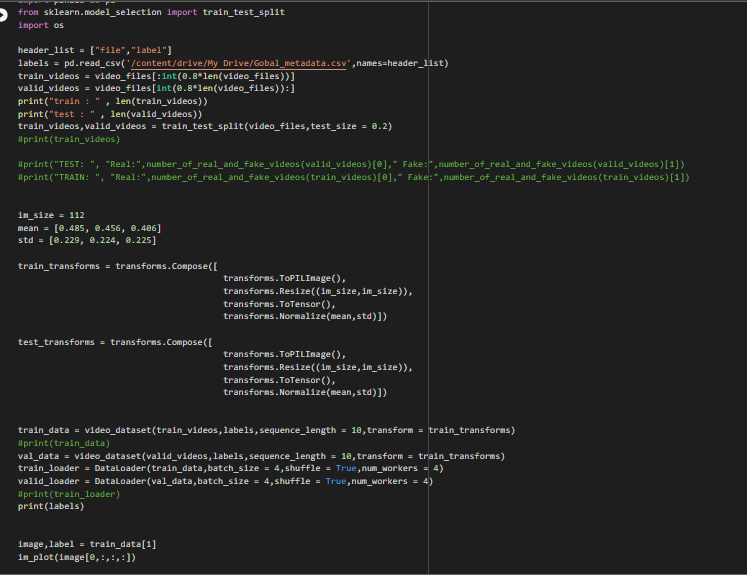


Figure 3.2.5: Applying transformations to data

Afterwards, data loaders are created to load the data in batches during the training and evaluation phases. These data loaders facilitate automatic and efficient batching of the data and can handle a range of data loading scenarios, including shuffling and parallel loading. As depicted in Figure 6, the loaded dataset is presented along with the number of videos used for training and testing, the label assigned to each video indicating whether it is real or fake, and a frame from one of the videos.

Text

Description automatically generated with medium confidence

Figure 3.2.6: The output of the code shown in Figure 5 is the extracted video from the train\_videos dataset, with each video labelled as either "Real" or "Fake".

**4.3 Training and Evaluating the Model**

The model is a combination of CNN and RNN. This model consists of three main components: a pre-trained ResNet50 CNN, a bidirectional LSTM layer, and a linear layer.

During training as shown in figure 7, the train\_epoch function trains the Model on a given dataset using a specified loss function and optimizer. The test function evaluates the Model on a separate dataset and reports the loss and accuracy of the model. Additionally, the print\_confusion\_matrix function generates a confusion matrix for the predicted and actual labels.

Throughout the code, PyTorch functions and classes such as nn.Module, nn.LSTM, nn.Linear, nn.AdaptiveAvgPool2d, nn.LeakyReLU, and nn.Dropout are used to define the custom model and compute the loss and accuracy of the model. Additionally, PyTorch data loaders are used to load the training and testing datasets, and the torch.optim module is used to define the optimizer.

Text

Description automatically generated

Figure 3.2.7: This is a portion of the code that exclusively prints out the confusion matrix, which is a performance metric for classification algorithms. It summarizes the model's performance by comparing the predicted and actual values of the target variable, displaying information about the number of true positives, true negatives, false positives, and false negatives that the model has predicted.

The first result of the code (figure 7) as shown in figure 8, is that there are 2 graphs on the first. During each epoch, the model is trained on a batch of data, and the loss is computed based on the difference between the predicted output and the actual output. The training loss is the average loss over all the training examples in a given epoch. The training loss is used to assess the performance of the model during training. As seen in the first graph, the goal is to minimize the training loss so that the model can make accurate predictions on new, unseen data. Typically, as the number of epochs increases, the training loss decreases, and the model's accuracy improves. For the second graph, regarding the accuracy, it is not uncommon for the validation accuracy to remain constant while the training accuracy increases. This may be because the model is starting to overfit the training data, meaning it is becoming too specialized to the training examples and losing its ability to generalize to new, unseen data.

Graphical user interface, application

Description automatically generated

Figure 3.2.8: The line graph results of the confusion\_matrix that was stated in figure 7: showing the relationship between epochs, training and validation loss and accuracy.

The second result of the code is shown in figure 9, which can be interpreted as follows:

* True Positive (TP): 43 cases were predicted as positive and were actually positive.
* False Positive (FP): 0 cases were predicted as positive but were actually negative.
* False Negative (FN): 17 cases were predicted as negative but were actually positive.
* True Negative (TN): 0 cases were predicted as negative and were actually negative.

To calculate the accuracy, you can use the formula:

accuracy = (TP + TN) / (TP + FP + FN + TN)

In this case, the accuracy would be 43 / 60 = 0.7167 or approximately 72%.

This means that the model correctly classified 72% of the validation set data points.

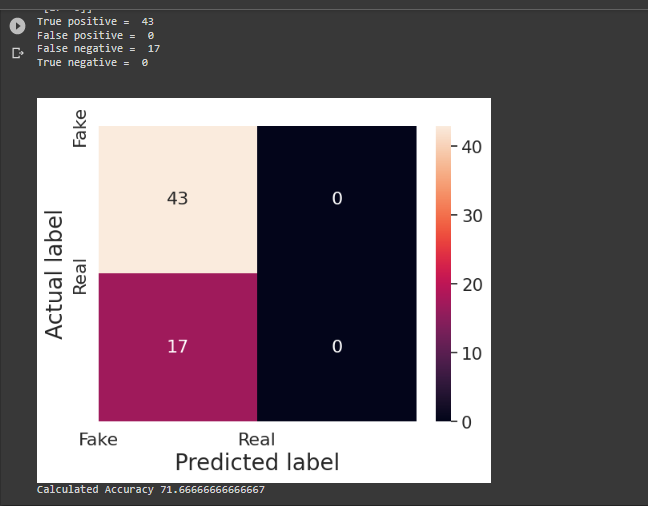


Fig 3.2.9: The result of figure 7: that shows the accuracy of how the model predicts.

**Module 5: Hyper Parameter tuning**

Hyper-parameter tuning is a crucial step in achieving maximum accuracy for the model. The process involves iterating several times on the model to identify the best hyper-parameters for our dataset. As shown in figure 7, the Adam optimizer with the model parameters and a learning rate of 1e-5 (0.00001) was utilized to ensure a better global minimum of gradient descent. Additionally, a weight decay of 1e-3 was utilized. As this is a classification problem, the loss was calculated using the cross-entropy approach. To optimize the use of available computation power, batch training was implemented with a batch size of 4, which was determined to be the ideal size for training in our development environment.

**Module 6: The User Interface**

The Django framework was utilized to develop the User Interface for the application, ensuring its scalability in the future. The index.html page of the User Interface includes a tab that allows users to browse and upload videos. The uploaded video is then processed by the model, which predicts whether the video is real or fake, and provides the confidence level of the prediction. The results are displayed on the predict.html page, superimposed on the playing video.

The Django application was installed and modifications were made to the url.py, settings.py, and view.py files to enable the back end of the website. The project settings were primarily stored in the project\_setting.setting file. To launch the server, a virtual environment was created and the command shown in Figure 10 was executed. For the front end, HTML and JavaScript were used and linked to Python files such as usr.py and settings.py. The website was developed using Visual Studio. To ensure proper functioning, the necessary requirement lists were downloaded, as shown in Figure 11. Connecting the back end and front end was achieved by running "python manage.py migrate" and including the models under the model folder found in the Django application. The software developed by the Deep Fake Challenge was utilized to check the accuracy of the models.

Text

Description automatically generated

Figure 3.2.10: This code shows the command to run the server and the application, the link provided is the link for the website.

Text

Description automatically generated with medium confidence

Figure 3.2.11 : This is the list of the requirements installed for the application to run

**Tools and Technologies Used**

To complete this project, adherence to certain hardware specifications was necessary, as well as the use of various software components.

|  |  |  |
| --- | --- | --- |
| **Programming Languages**  1. Python  2. JavaScript | **Programming Frameworks**  1. PyTorch  2. Django | **IDE**  1. Google Colab Pro  2. Visual Studio Code |
| **Hardware Components**  1. Nvidia GeForce GTX 1060  2. Cuda 116 |

# Review/Analysis/Test of Product/Service

To test the functionality of the created models, a prediction file was developed that determines if the model works correctly when a real or fake video is uploaded. Only the crucial parts of the code are explained in detail, as the remaining code follows the same structure as described. The code snippet depicted in Figure 11 is utilized to make a prediction. The video is uploaded through the "path\_to\_videos" variable, while the location of the model is specified in the "path\_to\_model" variable. The output of this code is the classification of the video as real or fake, as well as the confidence level of the prediction.

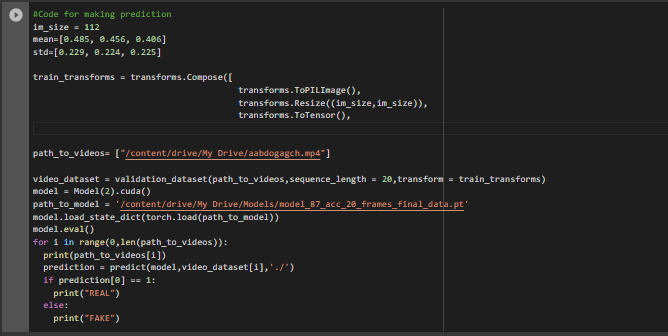


Figure 4.1: A code for making predictions.

To illustrate the output of the prediction code discussed earlier, an example is presented using a video shown in Figure 11. The prediction result for this video is depicted in Figure 12. As shown, the result indicates that the video is real, and the label is also checked to confirm the correctness of the model's identification. The prediction confidence for this result is approximately 95.

Figure 4.2: Showing the result of one of the models that have been trained.

**Test of Product/Service**

For the test of the application, we have implemented two kinds of testing which are functional and non-functional testing.

In this table the test case and test results will be explained in detail:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Case number | Test Case Description | Expected Result | Actual Result | Status |
| 1 | load a word file instead of video | Error message: Only video files allowed | Error message: Only video files allowed | Pass |
| 2 | Upload a 200MB video file | Error message: Max limit 100MB | Error message: Max limit 100MB | Pass |
| 3 | Upload a file without any faces | Error Message: No faces detected. Cannot process the video. | Error message: No faces detected. Cannot process the video. | Pass |
| 4 | Videos with many faces | Fake / Real | Fake | Pass |
| 5 | Deepfake video | Fake | Fake | Pass |
| 6 | Press upload button without selecting video | Alert message: Please select video | Alert message: Please select video | Pass |
| 7 | Upload a Real video | Real | Real | Pass |
| 8 | Upload a face cropped real video | Real | Real | Pass |
| 9 | Upload a face cropped fake video | Fake | Fake | Pass |

Table 4.1: Test Case Report

The website's visual user interface is illustrated in the diagram. It is a single-page website with an option to upload videos. We conducted unit testing for the website, and the outcome is presented in the table below.

|  |  |
| --- | --- |
| **Description** | **Diagram** |
| When the video is uploaded, the result shows it is real, with a thumbs up image. In the video, above the man’s head shows the confidence of prediction. In this instance, the accuracy rates the model detects the video is 99.9%. |  |
| When the video is uploaded, the result shows it is fake, with a thumbs down image. In the video, above the man’s head shows the confidence of prediction. In this instance, the accuracy rates the model detects the video is 98.7%. |  |
| When a video without any face is uploaded, it shows an error message that says “No faces detected.  cannot process the video.” |  |
| When there is no video uploaded, there is a pop-up error that says, “Please upload a file”. It is also impossible to upload any type of file other than video. |  |
| This figure illustrates what happens at the back end which is the server of the Django. It shows the steps of preprocessing, predicting and the results. |  |

# Lessons Learned and Recommendations

5.1 Lesson Learned

|  |  |
| --- | --- |
| 1. Lesson Learned | Lesson Details |
| Computer Vision (CNN and RNN) | Both CNN (Convolutional Neural Networks) and RNN (Recurrent Neural Networks) are important tools in the field of computer vision and are used in various tasks related to image and video processing. This was the superior part to achieve the main objective of our project to work. |
| Deep Learning (framework such as PyTorch) | PyTorch provides tools for building, training, and deploying deep neural networks. PyTorch is chosen as it has good support to CUDA i.e Graphic Processing Unit (GPU) and it is customize-able. |
| Python Programming Language | This was the main programming language that was used to create the models of the project. |
| Different Libraries (Facial Recognition, OpenCV, torch, Scikit-learn and many others) | Overall, these libraries provide a wide range of tools and functionalities that are essential for machine learning and computer vision applications. For example, Scikit-learn is an open-source machine learning library and used in this project as tools for data preprocessing, model selection, and model evaluation. |
| Django Framework | Django is a high-level Python web framework that enables developers to build web applications quickly and easily. |

Table 4.2: Summary of the test cases of the website

**5.2 Recommendation**

During the project it was extremely important to have a clear understanding of the project requirements and expectations. Therefore, it is recommended that future students take time to carefully read and analyze the project instructions before starting to work. Additionally, it is important that have a strong foundation in the area they choose to work in. In our case since we are networking students, we had some background about machine learning and computer vision, but it took a significant amount of time to get familiar with the new terms and topics. It is better to start the project as early as possible to avoid feeling rushed and overwhelmed and it is also essential to keep track of project milestones.

In hindsight, we acknowledge that the process of completing our project could have been improved by conducting more thorough research before selecting the topic. Due to the size and complexity of the project, we found it challenging to complete within the given time frame of 3-4 months. Nonetheless, we believe that our approach of carefully analyzing the requirements, breaking down tasks, and seeking guidance from various instructors and peers, was effective in keeping us organized and on track. Going forward, we plan to apply the same strategies of effective time management and collaboration with team members in our future work situations. We also recognize the importance of documenting our progress more extensively, as it will not only help us stay organized but also make it easier to troubleshoot issues that may arise. Overall, we recommend that others facing similar challenges take advantage of the resources available at their universities and seek guidance from different sources when needed.

# Conclusion and Future Work

**6.1 Conclusion**

The proliferation of deepfake videos in contemporary society has become a major concern, as these videos have the potential to spread disinformation and deceive viewers. The primary objective of this project was to develop a robust and effective deepfake detection system utilizing AI technology to address this issue. To achieve this goal, an approach based on neural networks was employed, utilizing LSTM-based artificial neural networks and pre-trained ResNext CNNs for the classification of videos as either deepfakes or genuine. This approach processed one second of video with ten frames per second and achieved a high level of accuracy. Additionally, the model was trained on large and diverse datasets from the data set of the Deepfake Detection Challenge to ensure that it could perform well in real-world scenarios. The success of this project is reflected in the high level of accuracy achieved by the deepfake detection method. Advanced techniques such as LSTM-based neural networks and pre-trained ResNext CNNs were utilized to improve the robustness and accuracy of the model. Furthermore, the user-friendly front-end application enhances the accessibility of the system to users who want to upload and classify videos. Overall, it is believed that this deepfake detection method can significantly reduce the spread of deepfake videos and provide users with the tools to distinguish between reality and fake videos. This approach represents a significant advancement in the field of AI-based video analysis and detection.

**6.2 Future Work**

Moving forward, there are several areas for potential development and improvement of the deepfake detection system. One area of interest is the addition of a deepfake voice detector to the project. With the increasing prevalence of deepfake audio, incorporating a voice detection component to the existing system would enhance its overall effectiveness and accuracy. Furthermore, while the current algorithm is capable of detecting deepfakes in facial features, future research can focus on improving the system's ability to detect deepfakes in full-body features. This would enable the system to detect a wider range of deepfakes and enhance its overall performance. Another potential area of development is the user-interface, which could be scaled up from the current web-based platform to a browser plugin, making it more user-friendly and accessible to a wider audience. Overall, these potential future developments have the potential to further enhance the capabilities and effectiveness of the deepfake detection system, providing users with a powerful tool to combat the spread of deepfake videos and misinformation in society.

# References

[1] B. Dolhansky, R. Howes, B. Pflaum, N. Baram, and C. C. Ferrer, “The deepfake detection challenge (DFDC) preview dataset,” *arXiv.org*, 23-Oct-2019. [Online]. Available: https://arxiv.org/abs/1910.08854. [Accessed: 01-Apr-2023].

[2] Aldwairi, M., & Alwahedi, A. 2018. Detecting Fake News in Social Media Networks. Procedia Computer Science, <https://doi.org/10.1016/j.procs.2018.10.171>

[3] *Academic.oup.com*. Available at: https://academic.oup.com/ct/article/31/1/42/5890673 (2021) (Accessed: February 7, 2023).

[4] Chawla, R. (1970) *[PDF] Deepfakes : How a pervert shook the world: Semantic scholar*, *International Journal for Advance Research and Development*. Available at: https://www.semanticscholar.org/paper/Deepfakes-%3A-How-a-pervert-shook-the-world-Chawla/c3b3a6d27dbbfed4df630b39fc0a8a6692b1828a (Accessed: February 7, 2023).

[5] Cross, C. (1970) *Using artificial intelligence (AI) and deepfakes to deceive victims: The need to rethink current romance fraud prevention messaging: Semantic scholar*, *Crime Prevention and Community Safety*. Available at: https://www.semanticscholar.org/paper/Using-artificial-intelligence-(AI)-and-deepfakes-to-Cross/9f1f1f04d9b9255cc62a2f9db471bfd1a02da780 (Accessed: February 7, 2023).

[6] *Combating deepfake videos using Blockchain and smart contracts | IEEE ...* (no date). Available at: https://ieeexplore.ieee.org/abstract/document/8668407 (Accessed: February 7, 2023).

[7] *Determining authenticity of video evidence in the age of artificial ...* (no date). Available at: https://journals.sagepub.com/doi/10.1177/1365712718807226 (Accessed: February 7, 2023).

[8] Li, Y. and Lyu, S. (2019) *Exposing deepfake videos by detecting face warping artifacts*, *arXiv.org*. Available at: https://arxiv.org/abs/1811.00656 (Accessed: February 7, 2023).

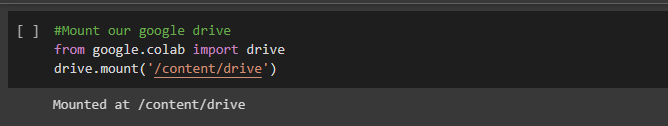
[9] *Determining authenticity of video evidence in the age of artificial ...* (no date). Available at: https://journals.sagepub.com/doi/10.1177/1365712718807226 (Accessed: February 7, 2023).

[10] *A deep learning framework for audio deepfake detection* (no date). Available at: https://www.researchgate.net/publication/356015648\_A\_Deep\_Learning\_Framework\_for\_Audio\_Deepfake\_Detection (Accessed: February 7, 2023).

# Appendices

The screenshots depict the code we developed for pre-processing the data and training the deepfake detection model. Additionally, the code for predicting whether a given video is real or fake is also included. These codes involve various processes, such as loading data, cropping faces, creating data loaders, defining the model architecture, training and evaluating the model, and making predictions.

Pre-Process



Text

Description automatically generated

Text

Description automatically generated

Training the model

Text

Description automatically generated

Text

Description automatically generated

A computer screen capture

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

**Prediction**

**Text

Description automatically generated**

**Text

Description automatically generated**

**Downloading the requirements lists**