Deep Learning Deepfake Detection

Hlumelo Notshe
Department of Computer Science
Stanford University
no6279@stanford.edu

Hlumelo Notshe
Department of Computer Science
Stanford University
tsvoboda@stanford.edu

Shannon Xiao
Department of Computer Science
Stanford University
sxiao1@stanford.edu

Abstract

In the pursuit of effective deepfake detection, this study delves into the comparative effectiveness of various deep learning architectures across multiple levels of granularity—from individual frame analysis to whole-video synthesis. We evaluate a multitude of models including Baseline 2D CNNs, Two-Stream CNNs for both per-frame and whole-voideo analysis, CNN-RNN hybrids, and 3D CNNs. Our results highlight the importance of integrating both spatial and temporal data to enhance detection accuracy. The Two-Stream models, which concurrently process spatial and temporal features, demonstrate superior performance in identifying deepfake videos, thereby underscoring the significance of temporal dynamics in deepfake detection technology.

1. Introduction

Deep Fake images and videos are a technology that uses advanced algorithms to create convincing, artificial copies of a person's likeness. These technologies have improved drastically over the years, making it nearly impossible for the human eye to discern a deepfake in some cases. This technology can harm reputations, raise legal and ethical issues, threaten privacy, and undermine overall trust in information. Even leveraging modern algorithms, the task of accurately and quickly identifying deepfakes has become increasingly challenging. This is particularly true for deepfake videos where current deepfake algorithms have found unique ways of applying deepfake masking on both a frame-level and/or a whole video level. Detecting deepfake videos with a high accuracy often requires a lot of time and computing resources. This project focuses on creating a deepfake video detection system to help combat this problem. We will experiment with multiple deep learning model architectures as well as various preprocessing methods on our input dataset.

We hope to evaluate various methods for deepfake video detection. We are particularly interested in the class of ”overlay” deepfake videos whereby authentic videos are overlaid with deepfake artifacts. In our research, we found that the two most effective components in video classification, and more specifically deepfake detection, are: 1. Identifying Spatial Anomalies and 2. Identifying Temporal Anomalies. We hope to experiment with various architectures that capture these two classes of features in different ways. Namely, we want to look at models ranging from ones that do detection on a ”frame-by-frame” level to ones that look to capture ”whole-video” detection at once. To do so we explore ”traditional” video classification techniques that look at spatial and temporal features separately before combining them for classification (e.g. Multi CNN Architectures, CNN-RNN Architectures). By comparing these various architectures, we hope to gain insight into the importance of Spatial and Temporal features in deepfake video detection and compare the effectiveness of how well traditional 2D image classification techniques compare with 3D video classification techniques in this space.

For all of these models, our input will be a set of potential deepfake videos split into frames. We'll then use variations of a CNN model to output a predicted score between 0 (Real) and 1(Fake), which we will then binarize to interpret as a prediction of a real or fake video. In our frame-by-frame models, this prediction is done at a frame-level, where we'll then use thresholding to determine the ultimate prediction for the entire video. For our whole-video model, there is single prediction assigned to the entire video at one time.

2. Related Works

To start, we referred to a recent survey by Rehman et al. [6] that provided a comprehensive review of deep learning approaches applied to video classification, covering key architecture and methodologies that have been tested. The paper emphasized the importance of processing both spatial and temporal video information to learn robust video detection models. The different categories of approaches presented include 2D image-based models that involve frame level feature extraction with CNNs and classification models like SVM; 3D CNNs that expand a 2D image classification account for time; spatiotemporal convolutional networks that use convolution and pooling layer to aggregate temporal and spatial information; recurrent spatial networks (i.e. RNNs such as LSTM or GRU); two/multi-stream networks that use layered optical flow to identify movements in addition to the context frames; mixed convolutional models utilize 3D convolution in the bottom or top layers but 2D elsewhere; and hybrid approaches that are a combination of CNN and RNN architectures. The paper also discussed fusion strategies such as concatenation, product, summation, maximum, and weights for combining features from different models or feature sets.

Another survey was conducted on current deepfake generation and detection methods [8]. By highlighting various deepfake video use cases, this paper helped us find the scope of our project, i.e. it outlined that the most common deepfake method is an "overlay" method where existing authentic videos are overlaid with deepfake artifacts. Furthermore, this article outlined 5 general methods used for deepfake detection: General Network-Based methods (frame-level classification task), Temporal Consistency-based methods (finds inconsistencies between adjacent frames), Visual Artifacts-based methods:(Use CNN-based methods to identify artifacts left behind in the blending operation done during deepfake generation), Camera Fingerprints-Based methods, Biological Signals-Based methods. Due to the results discussed in this paper, we choose to focus on comparing the effectiveness of the first two methods – i.e. General Network-Based Methods and Temporal Consistency- Based Methods.

Al-Dhabi and Zhang [3] looked into a deepfake detection algorithm that uses CNNs and RNNs using the Resnext50 model for feature extraction, as well as LSTM nets for finding temporal anomalies. The researchers tested their model on multiple deepfake datasets, which tests the adaptability of their model. This hybrid model showed a superior performance when compared to solo CNN/RNN models. In particular, we see that FP and FN were greatly reduced when compared to standalone models. The combined approach identified deepfakes with an accuracy rate of around 95%, which is a significant improvement over previous models, which found 80%-90% accuracies. This shows that a merged CNN-RNN model have a very high potential for countering deepfakes.

Another way to detect deepfakes uses a Multi-modal Multi-scale Transformer (M2TR) [7]. This model is designed to capture perceptually convincing forgeries at multiple scales by combining transformer models, frequency filters, and cross modality fusion blocks. This multi-dimensional method aims to address the limitations of previous models that primarily detected inconsistencies in images at only one scale or modality. By focusing on both spatial and frequency domains, M2TR enhances the robustness of deepfake detection, which is much better than prior methods which operate on just RGB color data and tend to overlook subtle forgery artifacts in the frequency domain. The results presented in the paper showcase the effectiveness of M2TR, particularly highlighting its superior performance across several datasets. M2TR achieved a 99.9% detection accuracy on the Celeb-DF dataset and 90.5% on their SR-DF dataset, showing a strong performance in both controlled and more challenging real-world scenarios. M2TR also does better than existing deepfake detection methods like the F3-Net, by noticeable margins, such as a 1.92% higher accuracy in low-quality settings of the FF++ dataset. These evaluations demonstrate the potential of M2TR in significantly advancing the field of deepfake detection technology.

3. Dataset and Features

We are using the Kaggle Deepfake Detection Challenge dataset as the source for our project's training, validation and testing data [1]. The original dataset contained 471.84 GB of labeled .mp4 videos. Some videos were labeled as "FAKE", representing deepfake AI generated videos, and others "REAL", which represent unaltered videos recorded with real-life subjects. Since the original dataset size was too large, we downloaded a subset of the data for our project. The sample we selected includes 825 10-second video clips. The original dataset included videos in both a horizontal and vertical format. We choose only to select the horizontal videos which have dimensions of 1080x1920. Once we downloaded our set of videos, we discretized our data by using the OpenCV library [4] to cut each video into 16 equally spaced out frames, which were saved and exported to .jpg files. The original videos' labels (0/1 for REAL/FAKE) were then attached to the frames of each video and exported to a .json file. The 13,200 jpg files and the labels json file were then uploaded to Google Drive for it to be accessible by our models, which were implemented on Google Collab.

3.1. Preprocessing

After reading in the image files, we preprocessed the images before feeding the data into our spatial and temporal models.

For the majority of our models, we compressed the images by cutting the dimensions of the height and width by 10 for more efficient model training; the final resolution of each image was 108x192 with 3 RGB channels. For our 3D CNN, due to the constraints of the pre-trained model in the first layer, we resized our image to size 224x224. We then normalized the input tensors for all the video frames. For the input of our Two-Stream CNN, we also took the batch of frames for each video to calculate optical flow to represent the temporal features of the input video. We used the OpenCV library to extract the optical flow between consecutive frames in each video and used the flow graphs as inputs to represent the temporal information of the video. Since we are looking at pairs of frames to calculate optical flow this means for every batch of 16 frames we get 15 flow graphs. We decided to duplicate the first flow graph to thus ensure the input pairs of ”spatial” frames and flow graphs matched. An example of the video frames our input training dataset and the generated optical flow graph used in the Two-Stream CNN model is shown here:

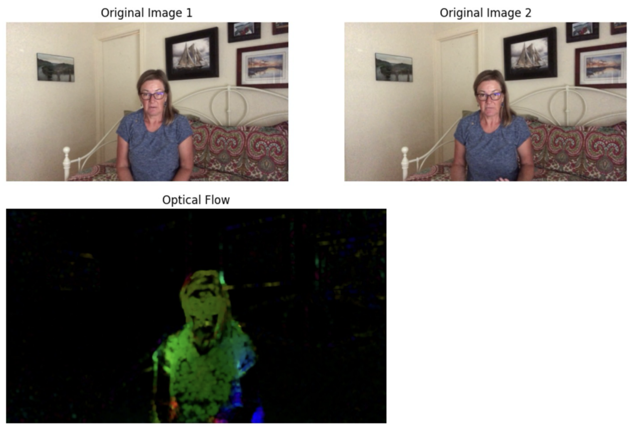

Figure 1: Visualized example input data and temporal feature extraction between two consecutive frames using optical flow

The inputs were then randomized and split into three subsets: 80% was used for the training set, 10% split for the validation set, and the last 10% split was used for testing. In our final split, we had 660 videos (10560 frames) in the training set, 82 videos (1,312 frames) in the validation set, and 83 videos (1,328 frames) in our test set.

For our local frame-by-frame models (e.g. baseline) we treated our resulting dataset as a ”bag of frames” and shuffled the data at a frame level. For our two-stream per frame model, we paired a ”spatial” frame with its corresponding ”temporal” flow graph and then treated the resulting dataset as a ”bag of spatial-flow pairs” and shuffled the data at a pair level. For our whole-video models (e.g. 3D CNN), we kept the batches of frames from the same video together and shuffled the batches on a batch level.

4. Methods

As discussed in our introduction, we wanted to compare several models that work on different levels of locality– from frame-by-frame analysis to whole video detectionz– and capture the spatial and temporal features of the videos in various ways. We used four different model architectures to experiment with different localities and on various granularities to yield 5 separate detection models. A summary of the detection models is consolidated in table 1 below.

4.1. Models

All of these architectures look to capture and combine spatial and temporal information in various ways.

As our baseline, we implemented one 2D CNN which solely extracts spatial features from the input frames. This model was intended to observe if high-accuracy local spatial detection abstracts well to whole video classification. Taking in an input video split into 16 frames, the baseline model then makes a prediction on each frame. We then average the predictions across frames to classify the entire video as deepfake or not. Table 2 summarizes the input and output details of the different layers of the baseline model.

The Two-Stream networks extract spatial and temporal features before concatenating the learned features for classification. We offer two approaches – one where this concatenation is done at a frame level and another where this concatenation is done on a whole video level. The Two-Steam per Frame model is fed a pair of spatial-temporal pairs with each frame of the video being paired with an optical flow graph. The spatial CNN looks at the individual frames in isolation and extracts spatial features. The second CNN takes temporal consistency data to separately extract temporal features. These features are then concatenated and pushed through a linear classifier for classification. On the other hand, the Two-Steam per Video model will take in the entire set of frames from a given video and corresponding set of optical flows. The first CNN looks at individual frames in isolation, extracts spatial features, then stacks the spatial features of the video batch. The second CNN extracts temporal consistency features from each optical flow graph and then stacks these temporal features for the video batch. These stacked spatial and temporal features are then concatenated and pushed through a linear classifier for classification. Table 3 and 4 describes the details of the layers of the three parts of the two-stream per frame model and the two-stream per video model, respectively.

The Sequential CNN-RNN network first extracts 2D spatial features from each frame and stacks the features. The model then pushes the stacked features through an RNN to deepen spatial feature extraction and build up temporal dependencies. Table 5 summarizes the input and output details

Table 1. Summary of our tested deepfake detection models.

Model Name	Architecture	Locality	Granularity
Baseline	2D CNN	Spatial	Frame-by-Frame
Two-Stream by Frame	CNN-CNN	Spatial-Temporal	Frame-by-Frame
Two-Stream by Video	CNN-CNN	Spatial-Temporal	Whole-Video
Sequential	CNN-RNN	Spatial-Temporal	Whole-Video
3D CNN	CNN	Spatial-Temporal	Whole-Video

Table 2. Baseline model layer specifications

Layer	Input	Output
ResNet50	(108,192,3)	(7, 7, 2048)
GlobalAveragePool	(7, 7, 2048)	(1, 2048)
Linear Layers	(1, 2048)	0/1

Table 3. Two-Stream by Fram model layer specifications

Layer	Input	Output
	Spatial Stream	
ResNet50	(108, 192, 3)	(7, 7, 2048)
GlobalAveragePool	(7, 7, 2048)	(1, 2048)
Linear Layers	(1, 2048)	(1,128)
	Temporal Stream	
Conv2D Layers	(108,192,2)	(25, 46, 64)
Flatten Layer	(25, 46, 64)	(1, 73600)
Linear Layers	(1, 73600)	(1, 128)
	Concatenation	
Concatenation	((1, 128), (1, 128))	(1, 256)
Linear Layers	(1, 256)	0/1

Table 4. Two-Stream by Video model layer specifications

Layer	Input	Output
	Spatial Stream	
ResNet50	(16, 108,192,3)	(16, 7, 7, 2048)
GlobalAveragePool	(16, 7, 7, 2048)	(16, 2048)
Linear Layers	(16, 2048)	(16,128)
Flatten Layers	(16, 128)	(1,2048)
Linear Layers	(1, 2048)	(1,512)
	Temporal Stream	
Conv2D Layers	(16, 108,192,2)	(16, 25, 46, 64)
Flatten Layer	(16, 25, 46, 64)	(16, 73600)
Linear Layers	(16, 73600)	(16, 128)
Flatten Layer	(16, 128)	(1, 2048)
Linear Layers	(1, 2048)	(1, 512)
	Concatenation	
Concatenation	((1, 512), (1, 512))	(1, 1024)
Linear Layers	(1, 1024)	0/1

Table 5. Sequential model layer specifications

Layer	Input	Output
ResNet50	(16, 108, 192, 3)	(16, 7, 7, 2048)
GlobalAveragePool	(16, 7, 7, 2048)	(16, 2048)
Linear Layers	(16, 2048)	(16, 512)
LSTM Layers	(16, 512)	(1, 512)
Linear Layers	(1, 512)	0/1

of the various layers of the sequential model.

Finally, the 3D CNN hopes to capture both spatial and temporal features at one time. The model adds convolutions over the time dimension in addition to the spatial dimensions, and it hopes to perform the same function as our previous recurrent and multi-stream approaches. Table 6 summarizes the details of the different layers of the 3D CNN model.

We tried to keep the model architectures as similar as possible in order to make a more fair comparison of our methods. For example, we used ResNet50 as our base 2D CNN across all architectures (except for the 3D model) and we kept the type and dimensions of the added hidden layers in each model similar to each other. This is to keep as many variables as close as possible.

Our per-frame models take in individual frames (or frame-graph pairs), whereas our 3D model takes in ”video chunks” and ultimately all models output a single prediction

Table 6. 3D CNN model layer specifications

Layer	Input	Output
Inflated 3D ConvNet	(16, 224, 224, 3)	(4, 7, 7, 2048)
GlobalAveragePool3D	(4, 7, 7, 2048)	(1, 2048)
Linear Layers	(1, 2048)	0/1

of REAL (0) or FAKE (1).

4.2. Training

4.2.1 Loss Functions

For our per-frame models we used standard Binary Cross Entropy Loss, calculated using the following formula (y is the correct label and p is the predicted output of the model):

$$\mathcal{L}_{\text{BCE}} = -\left(y\log(p) + (1-y)\log(1-p)\right) \tag{1}$$

For our whole-video models, we used a custom loss function that combines BCE and F1 loss, in order to get competitive performance. The formula for F1 loss [5] is given below. Note, precision is the ratio of true positives over the total number of true and false positive predictions, and recall is the ratio of true positives over the sum of true positive and false negative predictions.

$$\text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2}$$

$$\mathcal{L}_{\text{F1}} = 1 - \text{F1} \tag{3}$$

$$\mathcal{L} = \alpha\mathcal{L}_{\text{BCE}} + \beta\mathcal{L}_{\text{F1}} \tag{4}$$

We also found our dataset we imbalanced and so we incorporate class weighting in our training (using tf.keras [2] ”class weighting” parameter in fit method). This ensured that the model gave more weighting to the minority class during training.

4.2.2 Hyperparameters

In optimizing our model configurations, we carefully considered the batch size and learning parameters to ensure efficient learning and computational feasibility. For frame-by-frame analysis, we set the batch size to 16. This choice was driven by the intent to simulate the group dynamics of video frame batches used in whole video detection, thereby maintaining consistency in data handling across different model applications.

For whole-video analysis, we conducted a comprehensive grid search with batch sizes of 4, 16, and 32. Our findings indicated that processing 16 frames per input led to computational constraints when using batch sizes larger than 4. Consequently, we selected a batch size of 4 for whole-video analysis to balance between computational load and effective learning.

Regarding optimization techniques, we employed the Adam optimizer across our experiments. After experimenting with base learning rates of 10e-3 and 10e-6 through another grid search, we determined that a learning rate of 10e-3 yielded the most favorable results. This learning rate facilitated a faster convergence while maintaining stability in the training process, thereby optimizing the performance of our models under varying computational and data conditions.

Although we used an optimizer, we were hesitant to cross-validate because our dataset was imbalanced. Thus, it was challenging to ensure that we carried enough of the minority class in each fold.

During testing of our per-frame models on whole video classification, we use thresholding to determine which percentage of frames determines whether or not the entire video is deepfake. To determine the appropriate threshold we performed a grid search between 0.3-0.7 incrementing in steps of 0.1 – only the results from the best performance (accuracy-wise) are reported below.

5. Experiments/Results/Discussion

To tune our models, we tracked our training and validation loss and accuracy during model training after each epoch, and used a test set for final evaluation. We also evaluated our results with quantitative metrics such as accuracy, precision, and recall scores.

5.1. Results

Table 7. Prediction results for frame-level detection methods

Method	Accuracy	Precision	Recall
Baseline	0.97	0.95	0.97
Two-Stream Per-Frame	0.97	0.91	0.98

Table 8. Prediction results for whole-video detection methods

Method	Accuracy	Precision	Recall
Baseline	0.60	0.11	0.2
3D CNN	0.62	0.28	1.0
CNN-RNN	0.59	0.36	1.0
Two-Stream Per-Frame	0.83	0.36	0.48
Two-Stream Whole-Video	0.91	0.80	0.94

5.2. Discussion

In our experiments, it became evident that employing a spatial-only approach allows for effective local detection of features within individual frames. However, when these per-frame analyses are aggregated to represent entire video sequences, we observe a significant drop in accuracy, particularly evident in the baseline models. This decline underscores the critical need for incorporating temporal components into our models to enhance overall video understanding. This assertion is further supported by the comparative performance of the TwoStream model on whole-video analysis. Unlike the baseline, the TwoStream model incorporates both spatial and temporal information, thereby retaining a degree of temporal continuity across frames, which likely contributes to its superior performance in whole-video contexts. The confusion matrices for our baseline and Two Stream models are shown below:

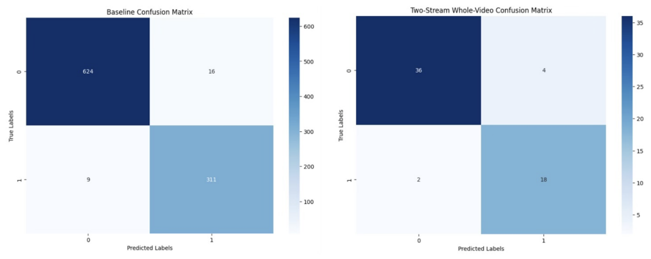

Figure 2: Confusion matrices for baseline (left) and Two-Stream whole-video (right) methods

Additionally, our frame-by-frame analysis exposed potential vulnerabilities to overfitting, despite the implementation of high regularization and dropout layers. The lack of diversity in our dataset may predispose the models to learn noise-specific features rather than generalizable patterns. This risk of overfitting highlights the importance of not only enhancing model architecture with mechanisms to capture temporal dynamics but also ensuring a diverse dataset that can challenge the model to learn more robust and universally applicable features. These insights suggest that future work should not only explore more sophisticated temporal integration strategies but also emphasize the acquisition and incorporation of a more varied set of video data to train our models.

We also noticed that our 3D CNN and CNN-RNN struggled. This is likely due to a relatively small dataset and dataset imbalance and homogeneity (i.e. needed more diversity). We tried to counteract this by using a custom loss score which optimizes for precision and recall, and also by incorporating class-weights during training. Unfortunately, these attempted fixes did not work too well. We got very low precisions, which suggest that the model didn't learning the features. This is further backed by accuracy of these models sitting at 0.59 and 0.62 which is the rough class split of the dataset. The high recall we observed can be attributed to our custom loss function and the use of class weighting.

Compounding these issues, the application of deepfake video techniques introduces additional complexity. Some frames are not consistently altered with deepfake masks, yet are labeled as deepfakes, leading to incorrect training data. The CNN-RNN model, which processes frames sequentially and extracts spatial features before passing them through the RNN, appears particularly susceptible to this issue. It often misinterprets unaltered frames as deepfakes, which exacerbates the model's performance issues. This highlights the necessity for more sophisticated handling of data inconsistencies and reinforces the need for a more robust dataset in training models to detect deepfake videos effectively.

However, this does not completely rule-out the viability a 3D CNN or Sequential model. If we look at the below heat-map looking at the features of the last convolution layer of our 3D CNN and match it up with the the corresponding temporal flow graph shown below – we can already see that the 3D model is picking up features extracted from areas of ”high” temporal flow (namely around the right hand and right shoulder area). This suggests that these models can extract temporal features alongside spatial features.Thus we could see improved performance with more diverse and balanced training data and more train time.

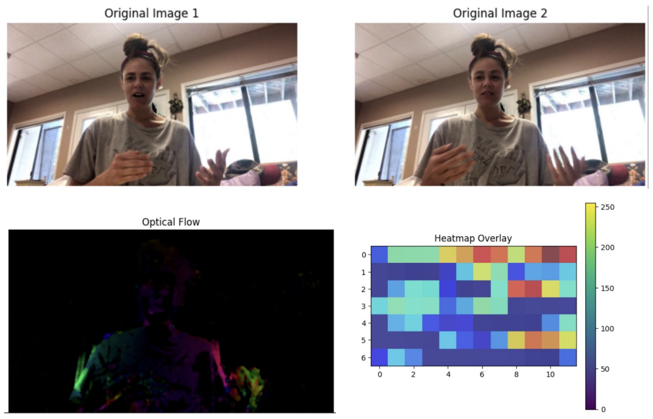

Figure 3: Visualized optical flow graph example with corresponding heat map

On the other hand, we achieved considerable success with the Two-Stream Whole-Video model, which demonstrated enhanced resilience to the issues plaguing the other models. This resilience can be attributed to its ability to capture temporal dynamics across entire video sequences. Even if individual frames are not altered with deepfake masks, the temporal flow component of the Two-Stream model can still detect anomalies, providing a robust mechanism for identifying deepfake content. Analyzing whole batches of frames allows the model to establish longer-term dependencies, further improving its capability to discern genuine sequences from manipulated ones. This approach not only compensates for sporadic inconsistencies in frame manipulation but also leverages the inherent temporal continuity of video to enhance detection accuracy.

6. Conclusion/Future Work

Our experiments strongly underscore the critical role that temporal information plays in the effective detection of deepfake videos. Our purely local spatial models struggled to work on the whole video level without some temporal component. Our experiments provide several ways of capturing temporal information. We found that models that analyze whole videos demonstrate potential but are notably dependent on the availability of extensive and diverse datasets to accurately capture temporal dependencies. This dependency becomes particularly significant when these temporal features are not explicitly extracted through techniques such as optical flow.

For our future work, we would like to train our specifically whole video models (namely 3D CNN, CNN-RNN) on a bigger more diverse dataset and see if/how much this improves performance. Furthermore as mentioned in our introduction,we looked specifically at the a particular type of deepfakes which used ”overlays” over authentic videos. We would want to test our models on other types of deepfakes and see how well they generalize. Furthermore, the Two-Stream CNNs have shown promising results in our study, indicating that integrating spatial and temporal data streams effectively enhances model performance. Given the intrinsic ability of transformers to handle long-term dependencies due to their attention mechanisms, it would be intriguing to investigate the performance of a Two-Stream 2D Transformer model in the future. There is already an increasing trend in using Transformers for vision tasks and such an exploration could potentially leverage the strengths of transformers to further improve the detection of deepfake videos, especially in scenarios where temporal continuity and subtle frame-to-frame variations are key indicators of authenticity. This future work could provide valuable insights into the adaptability and efficiency of advanced model architectures in the ongoing effort to combat deepfake technology. It would also be interesting to experiment

7. Contributions

All three of us contributed to all parts of the coding and final project writeup. However, Tycho led the data preprocessing and sequential model training, Shannon led the baseline model composition and 3D CNN model construction and training, and Hlumelo led the two-stream model creations and worked on model improvements and running experiments. We all worked on the full video prediction pipeline.

References

[1] Kaggle deepfake detection challenge.

[2] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[3] Y. Al-Dhabi and S. Zhang. Deepfake video detection by combining convolutional neural network (cnn) and recurrent neural network (rnn). *2021 IEEE International Conference on Computer Science, Artificial Intelligence and Electronic Engineering (CSAIEE)*, pages 236–241, 2021.

[4] G. Bradski. The opencv library. In *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Z. C. Lipton, C. Elkan, and B. Narayanaswamy. Thresholding classifiers to maximize f1 score. *arXiv preprint arXiv:1402.1892*, February 2014. Last revised on 14 May 2014 (this version, v2).

[6] A. ur Rehman, S. B. Belhaouari, M. A. Kabir, and A. Khan. On the use of deep learning for video classification. *Applied Sciences*, 13(3), 2023.

[7] J. Wang, Z. Wu, W. Ouyang, X. Han, J. Chen, Y. Jiang, and S. Li. M2tr: Multi-modal multi-scale transformers for deepfake detection. *Association for Computing Machinery*, page 615–623, 2022.

[8] P. Yu, Z. Xia, J. Fei, and Y. Lu. A survey on deepfake video detection. *IET Biometrics*, 10(6):607–624, 2021.