Deepfake Video Detection with Dimensionality Reduction

Hyun Jung, James Zhan
New York University, New York, NY 10003
hj1399@nyu.edu, jz4665@nyu.edu

Abstract

Deepfake detection is a rising problem that deep learning systems still have trouble solving. Some of these challenges include the adversarial nature of deepfakes and the need for heavy monitoring on this task. We aim to build a deepfake detection system that tackles these issues by reducing the time of the model deployment lifecycle. Our approach uses various techniques in handling these issues including implementing MTCNN for face cropping, and an autoencoder for further dimensionality reduction, a transformer for video classification task. We also implement a subsampling of video frames into the pipeline. Our results show that we are able to get competitive performance with very small training time using limited resources.

1 Introduction

Over the past few years deep learning systems has seen a massive increase in societal use. With the rise of big data and computational resources, deep learning has been able to benefit immensely paving the way for many technological breakthroughs and innovations. Transformer models and attention networks have enabled more accessible communication through the use of widely available machine translation platforms. Deep reinforcement learning is laying the foundations for autonomous vehicles. Overall deep learning systems have achieved more benefit than harm. However, there are also many cases where deep learning systems have been used for immoral causes and these issue need to be addressed. An example of this is through deepfakes.

The term "deepfakes" refers to videos that have facial or voice manipulations through the use of deep learning. Not long ago, these manipulations were manually performed from highly advanced video editing. However, with the wide accessibility of high computational resources and deep neural network models, manipulating videos for generating deepfakes is now easily accessible to anyone. And the use of deepfakes have been used for predominantly unethical motives such as identity theft, embezzlement and misinformation [1].

Therefore, for the general safety, it is imperative that there exists effective deepfake detection systems. However, this task comes with many challenges. In general, deep fake detection is extremely hard due to its adversarial nature. This is because deepfakes are generally created from systems that have been trained against deepfake detection systems [2]. Due to the adversarial nature, the need to monitor one's model and retrain based on newly generated deepfake videos that are able to bypass the current state of deepfake systems is crucial. However, deep fake detection is also a very time consuming task. A video is made up of many frames/images so the size of a video dataset can often be extremely large, resulting in long training and inference times. This presents a bottleneck for the deployment lifecycle as the training phase will be very slow.

For handling these challenges, we implement three techniques. The first technique is implementing a Multi-task Cascaded Convolutional Neural Network (MTCNN) into our model architecture [3]. The MTCNN will be used for face detection and face cropping to reduce the original dimension of the frames. Our second technique will be to use an autoencoder for further dimensionality reduction of the frames. And finally, we will experiment with subsampling of frames from the videos in order to observe an optimal amount number of frames to be used for training and inference. The hypothesis is that we do not need to use all the frames in a video file, and reducing the number of frames per video will help decrease the overall wall time. Our overall objective will be to reduce the model deployment lifecycle without sacrificing the performance of the overall classification task.

1.1 Related Work

In the area of deepfake video detection, CNN-based models have been very popular, by taking a frame-by-frame analysis approach, with a relatively shallow network [4]. Other alternative technique exploits the temporal evolution of video frames by utilizing LSTM networks. The authors in [5, 6] first extract a series of frame-based features, and then use a recurrent mechanism. Lastly, instead of pixel-based analysis, there have been attempts to adopt semantic analysis approach where the models try to learn to distinguish between real and fake head poses, inconsistent lighting effects, or eye blinking [7, 8, 9]. In this project, we take the CNN-based approach that utilizes convolutional autoencoder and transformer network.

2 Methodology

2.1 Approach

Our model consists of two parts, the convolutional autoencoder and the transformer. The input data to our network is video and audio data.

Video Data Video data is passed to CNN autoencoder and then to the transformer [Figure 1]. The original input frame size is $3 \times 1920 \times 1080$. Since the input size is quite large, we use Facenet's pre-trained model MTCNN [10] to crop the face portion of the frame, effectively reducing the dimension to $3 \times 160 \times 160$. We also observe that most cases show that the artificial alteration is focused on the facial area. The reduced data is then passed to the convolutional autoencoder for further dimensionality reduction and the output of the autoencoder is then passed to the transformer. First, the positional encoding is added to each item in varying frequencies obtained from multiplication of sines and cosines based on the position of the input. Then, the instances are passed to the transformer network, which outputs the sequence. Finally, we take the last value of the sequence output to train the classifier. In training, binary cross entropy loss is used, and at the inference stage, the sigmoid ouput is rounded to produce the prediction.

Audio Data For audio data processing, FFmpeg is first used to extract audio from video data [11] [Figure 1]. After the positional encoding, audio embeddings are passed to the transformer network. The sigmoid prediction from video and audio embeddings are then combined together and passed to the final classifier to produce the final prediction.

2.2 Models

Convolutional Autoencoder The autoencoder is composed of three convolutional layers with ReLU activations, each with the kernel size of 5. Two dropout layers and max-pooling layers are also incorporated. The decoder is very similar to the encoder but in reverse. The decoder takes the output of the encoder to reconstruct the original image. For the decoder to be able to reconstruct the original image well, the encoder must capture the most salient features of the image in the lower dimension. We expect that this would help represent and preserve the features in the lower dimension.

Transformer The transformer model takes a sequence of inputs and trains the encoder-decoder with an attention mechanism to produce a sequence of outputs. For this project, the output sequence is passed to the sigmoid layer as it is more suitable for our classification task. This transformer-based classifier is used to train both types of input data. We use the standard Pytorch transformer model with a multi-headed attention and a mask, which has eight heads with six encoder layers as well as dropout layers (default 0.1).

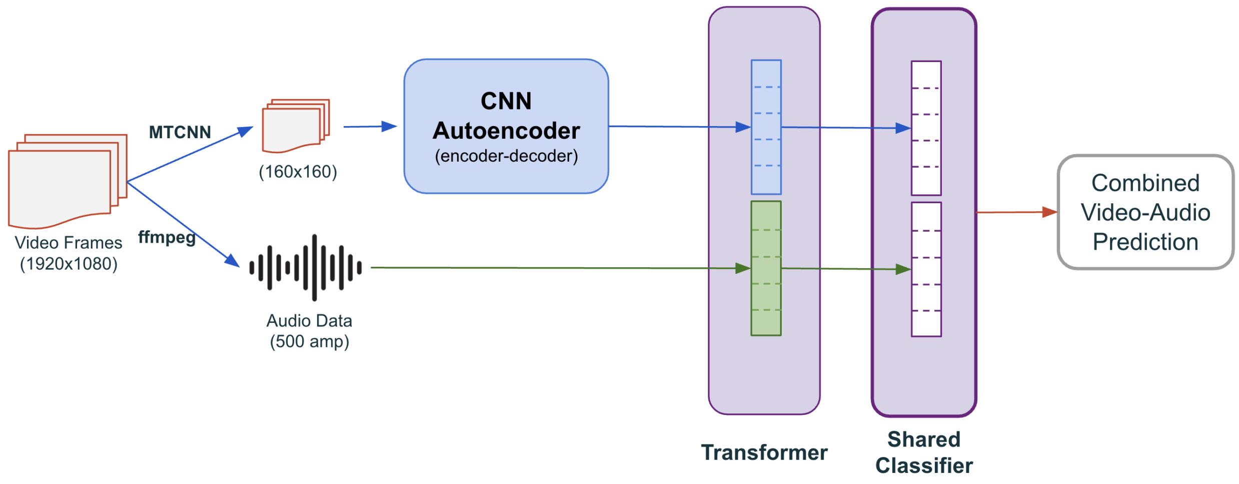

Figure 1: Diagram of CNN Autoencoder-Trainsformer DeepFake Detection Model.

3 Experiments

In this section, we will talk about our experimental setup and our computational setting.

3.1 Dataset

Our experiments were done using the Deepfake Detection Challenge Dataset (DFDC) [12]. This dataset was released by Facebook AI as a Kaggle competition that took place from March 2020-June 2020. The creation of the dataset was done through the use of over 3500 paid actors. The real videos show people talking while facing the camera, while the fake videos were created by manipulating the real videos through artificial networks. Overall the size of the full training set is over 470 GB of video data and consists of about 115K videos. Each video in the dataset spans 10 seconds and has 300 visual frames per video.

3.2 Computational Setting

Our experiments are implemented using PyTorch 1.8. The training of our models is done on the NYU Cassio Cluster using a single V100 GPU. Given our limited resources, we only train using 10 GB (about 3400 videos) of the full training set. Our test set is evaluated on the sample training videos provided in the dataset, which consists of 413 videos.

4 Results

For all of our experiments, we use stochastic gradient descent without momentum and our loss function was BCE with logits loss. Furthermore, we use a multi-step learning rate scheduler for improving the convergence of our model.

4.1 Subsampling Results

Our subsampling experiments are done by training the first N frames for $N \in \{30, 60, 90\}$. Note that every 30 frames corresponds to 1 second of the video. The results of our experiments is shown in Table 1.

Subsample	Train Loss	Validation Loss
30	0.574	1.338
60	0.521	1.217
90	0.648	1.513

Table 1: Table of Subsampling Results

Table 1 displays the train loss and validation loss of our experiments at the end of 20 epochs. Our results show that subsampling the first 90 frames had the worst performance while subsampling the first 60 frames has the best performance. From our subsampling experiments, there doesn't seem to be a clear pattern in validation or train loss from increasing or decreasing the number of frames. Therefore, our results indicate that deciding on the optimal number of frames to use for training seems arbitrary and is dependent on the dataset.

4.2 Hyperparameter Tuning Results

In addition to subsampling, we also conduct hyperparameter tuning on batch size and the initial learning rate. The results of our experiments at the end of 20 epochs are shown in Table 2.

Hyperparameter	Value	Train Loss	Validation Loss
Batch Size	10	0.521	1.217
	32	0.632	1.422
	64	0.657	1.643
	128	0.648	1.944
Learning Rate	0.005	0.574	1.339
	0.01	0.521	1.217
	0.05	0.510	1.190
	0.1	0.495	1.156

Table 2: Table of Hyperparameter Tuning Results

When tuning the batch size, we observe that the performance is better as we decrease the batch size. The reason for this is because with a smaller batch size, the model is able to make more gradient updates per epoch and is able to converge faster. In addition, we also implement a learning rate scheduler which doesn't take advantage of the larger batch sizes. Thus, while a larger batch size is able to make more precise estimates of the true gradient, pairing it with the scheduler will not make use of the precise estimates as later on in the training, the model would be taking too small of steps. Overall, our best observed batch size is 10 with a validation loss of 1.217 for BCE with logits loss.

When tuning the initial learning rate, we observe that a learning rate of 0.1 achieves the best performance amongst our tested values. Additionally, the pattern that we observe was that as we increase the learning rate, the performances also increases. The explanation is that pairing a higher learning rate with the learning rate scheduler allows the model to be trained with a good variety of step sizes. In general, if the model does not converge well with the initial high learning rate, then the learning rate scheduler will eventually decay the learning rate with a more favorable value for convergence.

4.3 Final Model Results

After finding the optimal set of hyperparameters to use for our model, our model is trained using a 10 GB sample of the full training data and performs a final evaluation on our test set. The results of our final model is that it achieves about 93.6% training accuracy and a 80.8% test accuracy. The amount of training time is approximately 35 minutes per epoch.

While these results may seem promising at first, the reality is that the dataset is extremely imbalanced. The baseline accuracy is already at 80% since 80% of the instances in the test set are deepfakes. This indicates that our model performs only slightly better than a naive model that predicts just the majority class. However, this is the common case for the top model submissions in the DFDC Kaggle competition. The top model from Selim Seferbekov achieved just 82.56% test accuracy, highlighting the difficulties of this task and dataset [13, 14]. The reasons why our model is not able to learn features well could be due to 1) the small training sample size, 2) the class imbalance issue in our dataset, and 3) our autoencoder may have prioritized the image reconstruction instead of learning salient features necessary for the classification task.

5 Conclusion

In summary, our work presents effective solutions to the challenges of deepfake detection. By implementing various techniques for dimensionality reduction such as MTCNN for face cropping, autoencoders for further dimensionality reduction, a transformer classifier, and subsampling of frames, we highlight a lightweight model architecture that can train a deepfake detection system using limited resources and training time while achieving competitive performances. However, the performance of our model, as well as many other models can still be significantly improved on. And the fact that there still does not exist a great solution in terms of performance for deepfake detection just reiterates how difficult of a task this continues to be in deep learning.

5.1 Future Work

For future work, we can resolve the class imbalance issue by upsampling the real video data or downsampling the fake video data. We can also adopt a different state of the art architecture where the encoder takes either video or audio data as input, and the decoder takes the other input. We can also adopt the triplet loss function so that the information from the audio embeddings and video embeddings can share the same latent space and help each other to better learn the features. Another thing we can try to change in the future is the hyperparmeter selection process. In this project, we experiment on one hyperparameter by fixing all the other hyperparameters. In the future, we can obtain all possible combinations of hyperparameters and do the random sampling to experiment on different kinds of combinations.

References

[1] Jesse Damiani. A voice deepfake was used to scam a ceo out of $243,000, Sep 2019.

[2] Shehzeen Hussain, Paarth Neekhara, Malhar Jere, Farinaz Koushanfar, and Julian McAuley. Adversarial deepfakes: Evaluating vulnerability of deepfake detectors to adversarial examples, 2020.

[3] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, Oct 2016.

[4] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen. Mesonet: A compact facial video forgery detection network. *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2018.

[5] P. Korshunov and S. Marcel. Deepfakes: A new threat to face recognition? assessment and detection. *CoRR*, abs/1812.08685, 2018.

[6] D. Guera and E. J. Delp. Deepfake video detection using recurrent "neural networks". *IEEE International Conference on Advanced Video and Signal-Based Surveillance (AVSS)*, abs/1812.08685, 2019.

[7] X. Yang, Y. Li, and S. Lyu. Exposing deep fakes using inconsistent head poses. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019.

[8] F. Matern, C. Riess, and M. Stamminger. Exploiting visual artifacts to expose deepfakes and face manipulations. *IEEE Winter Applications of Computer Vision Workshops (WACVW)*, 2019.

[9] Y. Li, M. Chang, and S. Lyu. In ictu oculi: Exposing ai created fake videos by detecting eye blinking. *IEEE International Workshop on Information Forensics and Security (WIFS)*, 2018.

[10] Tim Esler. Face recognition using pytorch. `https://github.com/timesler/facenet-pytorch`, 2020.

[11] Suramya Tomar. Converting video formats with ffmpeg. *Linux Journal*, 2006(146):10, 2006.

[12] Brian Dolhansky, Joanna Bitton, Ben Pflaum, Jikuo Lu, Russ Howes, Menglin Wang, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) dataset, 2020.

[13] Deepfake detection challenge results: An open initiative to advance ai.

[14] Selim Seferbekov. Deepfake detection (dfdc) solution by @selimsef. `https://github.com/selimsef/dfdc_deepfake_challenge`, 2020.

[15] Chris Hays. Deepfake detection challenge, kaggle. `https://github.com/johnchrishays/deepfake`, 2020.