CPSC 452 Final Project: Deepfake Detection with Transformer-Based Architectures

Carlo Abelli, Jason Chen, Chris Hays

May 6, 2020

Abstract

1 Introduction and background

Deepfakes are artificially doctored images or videos, usually involving a famous individual's image being imposed onto a realistic background to portray them as saying or doing things that they did not do. In recent years, advancements in AI have resulted in deepfakes becoming significantly more realistic and difficult to distinguish from undoctored videos. Most existing methods to generate deepfakes use autoencoders or generative adversarial networks (GANs). As deepfakes become more widespread on the internet, they can create a variety of social problems, from political hoaxes and fake news, to fraud and invasions of privacy. A variety of industry and governmental groups, including Amazon, Facebook, Microsoft, and the Partnership on AI's Media Integrity Steering Committee are invested in solutions to accurately distinguish deepfakes from real videos. As it becomes more difficult for humans to distinguish between artificially generated images and real-life images, it becomes increasingly important to develop AI mechanisms for detecting real from fake.

In this project, we trained a deep neural network to distinguish real videos from artificially generated deepfakes. Deepfake detection mechanisms look for salient features including lighting, shadows, and facial movements, as well as temporal features or inconsistencies between frames in the video. As such, we applied state of the art deep learning approaches, including convolutional autoencoder and transformer models, for feature detection and classification of deepfakes.

Convolutional autoencoders are comprised of a encoder that contains convolutional layers and a decoder that takes the output of the encoder to reproduce the original image. The output of the encoder, or the code, is a lower-dimensionality representation of the image. For the decoder to reconstruct the original image effectively, the encoder must capture the most salient features of the image in the lower dimension code. Thus, convolutional autoencoders are a nonlinear method for obtaining a low-dimensional embedding of images and should allow us to preserve the most relevant features while balancing for memory and runtime constraints.

Transformer models have recently been proposed in deep learning. These models typically take in a sequence of inputs and trains encoder and decoder networks with attention mechanisms to produce a sequence of outputs [4]. By focusing the multi-headed attention mechanism on a sequence of embedded inputs, such as through word2vec embeddings or a convolutional autoencoder mentioned earlier, it can be used to extract relevant features temporally within a video, which is a sequence of individual image frames. Therefore, it is particularly useful for our task of detecting deepfakes. Transformers typically produce a series of outputs, but we can modify the transformer architecture for the problem of classification instead, similar to appoaches by others [1]. In this project, we trained a transformer-based classifier on both the audio and the video dimension.

2 Dataset

2.1 Description

For this project we used the dataset provided by the "Deepfake Detection Challenge" on Kaggle. The training dataset consists of 119,154 videos, each with 300 frames (in color) with resolution which we scaled to 1920 by 1080. We sampled 5000 amplitude values from the audio of each video. A total of 100,000 (or 84%) of these videos are fake. This constitutes 470GB of content.

Each deepfake video is associated with a real video. The deepfake versions of the videos differed from the real versions by small alterations to the image frames or audio: adding glasses where there were none, changing a video subject's speech, making a subject's facial features look older. All of the video footage contains a person facing the camera, usually speaking.

Initially, we thought it might be possible to create four training labels (real, fake audio, fake video, fake audio/video) rather than just the two that came with the training data (real, fake). This would allow us to separately train audio and visual classifiers. This proved difficult, however, because numerical issues prevented us from comparing the audio or video frames between the original videos and derived fake videos.

2.2 Modifications

The imbalance between real and fake videos presented a challenge for training the model, since guessing "fake" every time gives about 16% error. At one point, our model learned to do just that, and it was difficult to tell the true performance of our model. We constructed modified train and validation datasets consisting of a 50/50 split between real and fake videos.

Another issue is that each video has 300 frames and 50027 audio samples. We discovered that decreasing the length of the inputs lead to increases in training speed. From manually inspecting examples of video and audio, we observed that it was quite easy to distinguish deepfake videos from nearly the beginning of the video. Because of this, and since training on shorter sequences of frames and audio data was faster, we clipped the first 1s (30 frames) of video images, and the first half second, or 500 frames, of audio.

3 Method

3.1 Architecture Design

Figure 1 shows the broad outline of our architecture which processes both sequential audio and visual data, and the specific components will be discussed in the following sections.

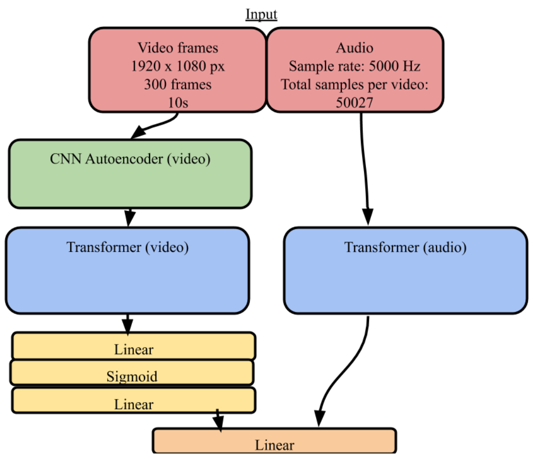

Figure 1: Deepfake Detection Classifier Architecture

3.2 Overview

We trained separate models on the video and audio data and then used a linear layer to combine their outputs into a single real/fake prediction.

For the video data, we used a convolutional autoencoder to reduce the number of dimensions of each video frame from millions to thousands. As input to the CAE, we used two separate strategies: inputting the whole video frame or just the face of the person in the video. We then used the encoded versions of the sequence of video frames as tokens to input into a transformer encoder. Then we used the output of the transformer encoder as input to a linear model which produced a single output value. This output value was combined with the output of the audio transformer to create a prediction.

For the audio data, we input the first 500 amplitude readings into a transformer encoder. Since we only had sound wave amplitude readings to work with, the number of features in the encoder input was just 1. Then we used the output of the transformer encoder to feed into a linear layer with the output of the visual transformer to create a single real/fake prediction. In what follows, we describe the architecture and design choices in more detail.

3.3 Visual data preprocessing

We trained the model on visual data with and without preprocessing.

To train the video component of the model without preprocessing, we input the 1920x1080 pixel images directly into a convolutional autoencoder, resulting in CAE inputs of 3x1920x1080 features.

We also trained the model using only a 160x160 frame around the face of the speaker for each frame. From visual inspection, we noticed that a high proportion, if not all, of the fake videos included alterations to the face of the person in the video. We used facenet-pytorch's implementation of MTCNN, a pre-trained face detection CNN for this purpose [5]. MTCNN crops faces around the frame, and we feed this input into the convolutional autoencoder, inputting frames with 3x160x160 features.

3.4 Convolutional autoencoder architecture

Even with face cropping, we guessed that 3x160x160 features would be too many to input directly into a transformer. (Most of the seq-to-seq text translation transformer models use around 1000 features.) So each frame in the video was encoded using a convolutional autoencoder, with the hope of extracting out useful features in a lower-dimensional embedding. Without face cropping, the encoded frames had 3600 features, and with face cropping they had 1296 features.

The CAE encoder consists of 3 convolutional layers with ReLU activation, each with a kernel size of 5. We also included downsampling layers between the convolutional layers. The choice of 3 layers came from the desire to strike a balance between allowing the network to learn more complex features while allowing it to process fairly quickly given the sheer amount of video that will eventually be input. The decoder mostly reversed the steps of the encoder.

3.5 Hyperparameter selection for the convolutional autoencoder

In training the convolutional autoencoder on video, we ran several experiments to determine the optimal hyperparameters. We included these in our progress report. However, in our initial parameter tuning, we were using a CAE with a hidden layer about 1/8th the size of the input. We soon realized that we wanted a hidden layer about 1/1000th the size of the input, so we redesigned the CAE. We used mean squared error for the loss function.

We trained the autoencoder for about 10 minutes on a NVIDIA GeForce GTX 1080 Ti, 10 epochs of 250 images each. The test set consisted of

400 unseen images. The face cropped version of the autoencoder achieved training error of 0.039 and test error of 0.045. The autoencoder for the videos without preprocessing achieved training error of 0.014 and test error of 0.014.

	Train MSE	Test MSE
Face cropped CAE (hidden dim = 1296)	0.039	0.045
Un-preprocessed CAE (hidden dim = 3600)	0.014	0.014

3.6 Transformer architecture

In this section, we discuss the architecture used for the transformer. The input to this is a sequence of embeddings of video frames from the convolutional autoencoder.

The transformer consists of a multi-headed attention mechanism and a mask that allows each input to only see inputs that have passed before.

We further add positional encoding to each item in the input sequence, in the form of multiplication by sines and cosines of varying frequencies based on the position of the input.

The transformer outputs a generated sequence based on the input sequence. We take the last value of the sequence and train a classifier on that.

We piped the outputs of the transformers to a linear layer, a sigmoidal activation function, and another linear layer with output dimension one. This feed-forward neural network takes the outputs of the transformer and outputs one point, corresponding to the classification result of the video transformer. In training, we used binary cross entropy on the sigmoid of the output. In testing, we rounded the sigmoid of the output to binary classification 1 or 0.

We also train an audio transformer encoder with the same architecture. The model takes in a time series of audio input, which only consists of one feature. We used standard parameter choices for the visual and audio transformers: 8 heads with 6 encoder layers and dropout of 0.1. We tried several different parameter settings for the number of heads, encoder layers and dropout, but could not discern a difference in their outputs.

3.7 Loss function

We used cross-entropy log loss to train the model.

$$\sum_n y \log(x) + (1 - y) \log(1 - x)$$

It is well-suited to training on classification problems, since it penalizes incorrect confident classifications and unconfident ($p_{\text{correct}}(x) \approx 1/2$) classifications. We used sigmoid activations to mitigate the vanishing gradients problem.

3.8 LSTM Autoencoder / Audio Processing

We also experimented with an LSTM autoencoder on the audio input in order to create an encoding of the salient features of the audio rather than using a transformer to do this. The architecture consisted of a standard LSTM encoder and decoder.

However, after a few days of experimentation, the LSTM autoencoder was still not learning salient features of the audio. For future work, we think that it may be productive to encode the audio sequentially using Fourier decomposition, or apply some kind of sampling technique on a per-frame basis using logarithmic mel spectogram extraction, as in this paper's approach [1].

3.9 Aggregation on the Transformer Encoder Output

Another experiment was attempting to include an aggregation block on the transformer encoder outputs as seen in some other papers. Specifically, instead of classifying on the final output from the transformer, classification would be run on each individual output from the transformer and then either a mean or max-pooling would determine the final classification. However, this yielded similar results to the previous attempts, with 50% accuracy on the test set.

4 Results

We trained each model for 20 epochs of 500 videos each. We tested on 400 unseen videos.

	Train BCE	Test classification error
Classifier with face crop	0.693	49%
Classifier with face crop and aggregation	0.693	47%
Classifier without face crop	0.704	51%
Classifier without face crop with aggregation	0.697	50%

Recall for comparison that the BCE for a classifier that always outputs 0.5 is 0.693.

5 Discussion / Future Work

We had two goals in creating this project. Firstly, we wanted create an audiovisual classifier which incorporated the sequential nature of video data. Second, we wanted to evaluate the performance of this architecture in a classification setting. We have a few guesses about why the model fails to train. One possible explanation is that the autoencoder for the video frames does not preserve enough of the data important to classification of deepfakes, instead prioritizing reconstruction. To mitigate this, we added the face cropping pre-processing step, with the goal of preserving more salient information in the latent representation. However, we did not see better results.

Another possible way to improve the architecture might be to combine the information in the audio and video together a more meaningful way. For example, [1] uses an encoder-decoder architecture where either the audio or the video frames are passed into the encoder and then the other input is added to the decoder. They claim to achieve the state of the art for video classification.

To see how we could have improved our performance, we also looked through the architectures proposed by contestants in the Kaggle competition that inspired this project. Many of the top architectures just classify each frame and then perform some sort of thresholding/averaging on the various frames. It appears that many of the models tested that attempt to look at frames over time do not perform as well as this more basic frame-by-frame classifier [3][2]. Additionally, it appears that many did not take audio into account significantly, if at all.

Perhaps it could be possible to incorporate these models with our transformer based model, by feeding some of the classification data of each frame over time to perhaps capture even more complexity in the data rather than just static frames.

6 Conclusion

In conclusion, this project designed and trained a transformer-based architecture for task of deepfake detection. Though the classifiers we trained were not able to successfully classify at a signifcantly higher level than random guessing, we learned quite a bit about the limitations of transformer based architectures and areas for potential improvement.

References

[1] Wim Boes and Hugo Van hamme. Audiovisual transformer architectures for large-scale classification and synchronization of weakly labeled audio events. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, page 1961–1969, New York, NY, USA, 2019. Association for Computing Machinery.

[2] Jan Bre. 27th place solution. https://www.kaggle.com/c/deepfake-detection-challenge/discussion/145965, 2020.

[3] Selim Seferbekov. 3rd place solution. https://www.kaggle.com/c/deepfake-detection-challenge/discussion/145721, 2020.

[4] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[5] Kaipeng Zhang, Zhanpeng Zhang, Zhifeng Li, and Yu Qiao. Joint face detection and alignment using multi-task cascaded convolutional networks. *CoRR*, abs/1604.02878, 2016.