DeepFake Detection: Progress Report

Jesse Schmidt (jbschmid)

Ellen Kalvan (ekalvan)

Anon Cheewakarn (acheewak)

Summary

The objective of this project is to detect a ‘Deepfake.’ Deepfakes have become a buzzword in the past year. A Deepfake is video content that is either visually or auditorily manipulated

To train the model, a number of visual and audio features are extracted and then prepared before being fed in. The main categories of these features are video frames and audio frames. A more detailed description of features that are fed into the model can be found below.

The main components of this model include 2D Convolutions, 2D Max Poolings, Long Short-Term Memory (LSTM) and, of course, some Dense layers. The model is expected to be able to detect fake videos correctly 75% of the time, at the very least.

Dataset

We have about 470 GB of training data from the data science website Kaggle. Each file is a 10-second video with one or more people having a dialogue.

The deep fake manipulations come in a few forms. The video manipulations are very obvious when they are present. Faces have extra noise surrounding them, skin tones do not match, or simply do not look human.

The audio manipulations are harder for a human to pick up on, but still noticeable. Features of fake audio include unusual changes in pitch, volume, and timbre of speech.

Models and Algorithms

The classification of audio/video as real or fake is quite difficult mainly because the provided labels do not specify whether there has been audio manipulation or video manipulation. Due to the different physical properties of audio and video data, the features that can be extracted are thus of different dimensions.

A data generator was created that inherits from the Keras Sequence object to handle the loading of video data into memory, in batches. The generator does some preliminary data processing to extract audio features, and reframe video data to only the areas with faces.
The video is processed following these steps: (using these packages)

1. Video and audio are split into separate variables. (moviepy)
2. Audio is split into ~300 chunks equal to that of a single video frame or 1/30th of a second. (moviepy)
3. Video is split into ~300 frames given a 10-second video. (open-cv)
4. The MFCC (Mel-frequency cepstral coefficients) is extracted from each audio chunk creating an array of [300, 13]. (python_speech_features)
5. Using a pre-trained python module the images are reduced to the individual's face with a buffer of 50 pixels then resized to 150x150x3. This creates an array of size [300, 150, 150, 3]. (face-recognition)
6. Return [video data, audio data], and labels for training.

That preliminary processing is slow and possible improvements are discussed in a later section.

Now video and audio data are separated. The video frames are now run through a CNN. The CNN largely resembles that which was used in Assignment 2. Each video frame will need to be processed by the CNN individually. The hope is that the CNN will be able to locate certain irregularities in the facial features that will alert that the video is fake.

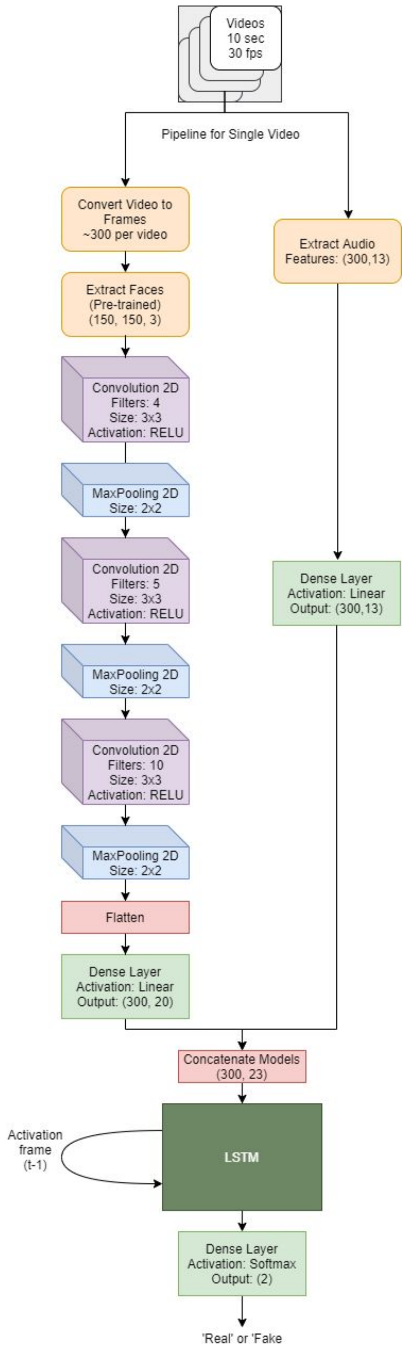
The CNN has these 6 layers:

1. 2D Convolutions, 4 filters, size: 3X3, Activation: RELU
2. 2D Max Pooling, size: 2x2
3. 2D Convolutions, 5 filters, size: 3X3, Activation: RELU
4. 2D Max Pooling, size: 2x2
5. 2D Convolutions, 10 filters, size: 3X3, Activation: RELU
6. 2D Max Pooling, size: 2x2

After each frame of video is processed, they are flattened in preparation to be concatenated with the audio features that were extracted earlier. After flattening, the data is connected to a dense layer with a linear activation function and an output of size

20. The video frame features are now appended to the audio features giving us an array of [300, 23] where 300 represents the number of time steps **t** in the LSTM.

The plan is that the LSTM will be able to detect interesting variations in both the video and the audio frames. Examples of such a change are differences in skin tone or audio pitch over time. The output of the LSTM is a categorical array of size 2. In order to decrease training, a softmax will be used to increase losses where needed. A full diagram of the model can be seen below.


Videos
10 sec
30 fps
Pipeline for Single Video
Convert Video to Frames ~300 per video
Extract Faces (Pre-trained) (150, 150, 3)
Convolution 2D Filters: 4 Size: 3x3 Activation: RELU
MaxPooling 2D Size: 2x2
Convolution 2D Filters: 5 Size: 3x3 Activation: RELU
MaxPooling 2D Size: 2x2
Convolution 2D Filters: 10 Size: 3x3 Activation: RELU
MaxPooling 2D Size: 2x2
Flatten
Dense Layer Activation: Linear Output: (300, 20)
Extract Audio Features: (300,13)
Dense Layer Activation: Linear Output: (300,13)
Concatenate Models (300, 23)
Activation frame (t-1)
LSTM
Dense Layer Activation: Softmax Output: (2)
'Real' or 'Fake

Preliminary Results

As of February 16th, there are no test results available from the model. There are several roadblocks as a result of limited processing power and the enormous amount of data that have to be preprocessed before being fed to the training model.

Currently, it takes roughly 5 minutes to extract and obtain meaningful data from a 10-second video. This data, as mentioned in some sections above, include facial recognition per video frame and the MFCC value from audio frames. With a total of over 470GB of 10-second videos, this process alone will take more time than what is available.

There are several potential solutions to help mitigate the impact caused by these issues; they can be found in the section below.

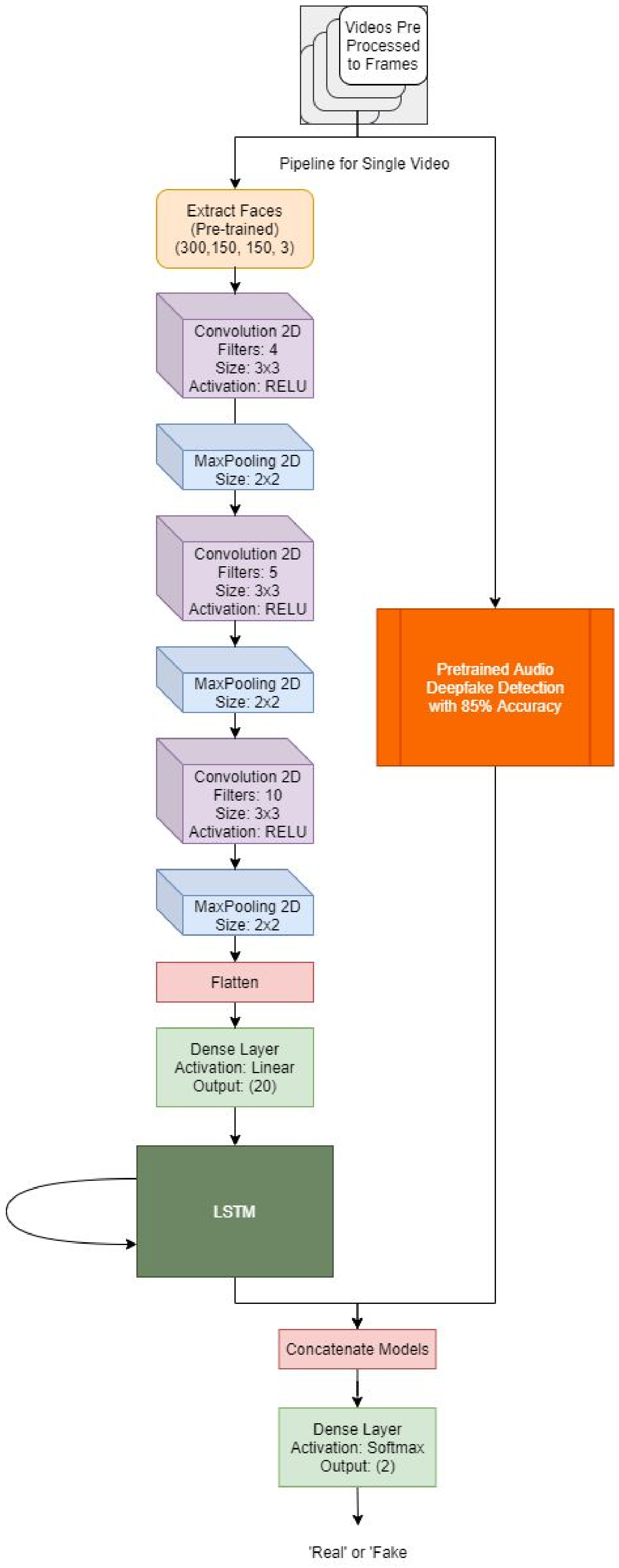
Future Plans

There are a number of issues described in the report that are keeping our team from being able to train our model. The amount of time it takes to process audio/video files and the data produced is beyond what we can process in either Google Colab or our personal machines. However, we have identified some areas where we can remove redundancies, reduce data processes, and increase model accuracy.

Currently, it takes us about five minutes to reduce all 300 frames of a video down to just the areas which contain faces. The current amount of time it takes to detect faces in each video frame makes it impossible to train the model with sufficient data in the given amount of time. To mitigate this we will attempt to reduce the size of the video down to 5 seconds to reduce the number of frames overall. In addition, we will reduce the number of times that we run the face-recognition model to find the face locations and instead reuse the face location for a number of the following frames.

We have found a pre-trained model that is able to determine if audio has been manipulated algorithmically. The authors claim that the model has an accuracy of 85%. Our team is thinking of utilizing this model to preprocess audio information and use the generated output as a feature in our own model. This would greatly reduce the complexity of our design and allow us to focus on video manipulations.

Our future plans can be summarized in the diagram below.


Videos Pre Processed to Frames
Pipeline for Single Video
Extract Faces (Pre-trained) (300,150, 150, 3)
Convolution 2D Filters: 4 Size: 3x3 Activation: RELU
MaxPooling 2D Size: 2x2
Convolution 2D Filters: 5 Size: 3x3 Activation: RELU
MaxPooling 2D Size: 2x2
Convolution 2D Filters: 10 Size: 3x3 Activation: RELU
MaxPooling 2D Size: 2x2
Flatten
Dense Layer Activation: Linear Output: (20)
LSTM
Pretrained Audio Deepfake Detection with 85% Accuracy
Concatenate Models
Dense Layer Activation: Softmax Output: (2)
'Real' or 'Fake