APPLICATION PROJECT: DEEP FAKE DETECTION

Katrina Greene

Data 696
American University

ABSTRACT

For my final project for Data 696, I did an application project for deep fake detection. Deep fake technology has had major advancements in the last few years; this rapid growth in the accessibility and quality of deep fakes has outpaced legislation to address it. As such, misinformation caused by digital forgeries is a real danger that we are facing. The goal for my project is to use Google's publicly available data set to train a classification model that detects whether a video is original or manipulated. I tested three different model architectures: 3D CNN, a pre-trained ElasticNet CNN, and a combination of CNN and RNN. I used Tensorflow and Keras on a Google Colab GPU isntance, and I found that the pre-trained model performed the best.

1. MOTIVATION

"Deepfake" technology is a branch of AI software that allows users to swap a video's subjects faces or create fictitious people entirely to create digital forgeries [1]. In the past decade, this technology has advanced too quickly for laws to keep up with. A lack of legislation means that there are no guardrails or regulations to manage this spread. As a result, the AI software to create these forgeries is easily accessible and can be used to spread harmful misinformation- for example, a video circulated on social media last year in which it appeared that Ukrainian President Volodymyr Zelensky surrendered to Russia [2]. Since this technology is so fast-moving, law enforcement officials have been struggling to detect deep fake videos [1]. My motivation for this application project is to apply deep learning to classify whether a given video is authentic or whether it was manipulated with deep fake technology.

2. DATA

The data I used for this project is a set of .mp4 videos created by Google, which was provided to me by Professor Boukouvalas. This data set consists of 3068 manipulated videos and 363 original videos. In order to pass an observation, or one video, into a classification model, I need to transform these videos into frames with 5 dimensions: samples, frames, height, width, and channels. I used Keras's documentation on how to how to load video data to Tensorflow models as a guide [3]. During this process, I selected a subset of the data set from the zip file to extract and load. I selected a subset instead of the entire zip file in order to reduce the possibility of computationally expensive models. In this subset, I selected an equal amount of original and manipulated videos in order to keep the classes balanced- I selected 100 videos in total. After I selected the file names for my training/validation/test subset with a 60/20/20 split respectively (30/10/10 videos per class), I unzipped these files to my Google Colab instance. This is a still from a "Manipulated" video from my training set:

Once these files were there, I used a function that splits the videos into frames, reads a randomly chosen span of n frames out of a video file, and returns them as a NumPy array. I chose 10 frames in order to not use too much memory or computation overhead. I then passed this data into a Tensorflow data input pipeline that created a Dataset object where a set of frames is associated with a label. My resulting training, validation, and test data set objects have the following

dimensions:

```
Shape of training set of frames: (2, 2, 2, 10, 224, 224, 3)
Shape of training labels: (2, 2, 2)
Shape of validation set of frames: (2, 2, 2, 10, 224, 224, 3)
Shape of validation labels: (2, 2, 2)
Shape of test set of frames: (2, 10, 224, 224, 3)
Shape of test labels: (2,)
```

3. METHODS

I tested three different model architectures to see which how each performs. I tested a model composed of 3D Convolutional Neural Networks, a pre-trained CNN, and a model composed of a combination of a CNN and RNN layers. For each model, I used the Adam optimizer, the binary cross entropy loss function, and accuracy as the performance metric.

3.1. 3D CNNs

I started out with a straightforward model with just 3 dimensional CNNs because at the heart of this is essentially an image classification problem, and CNNs handle computer vision well. There is a temporal element to this application, however, so I decided to used 3D CNNs since dealing with the individual slices independently in 2D CNNs deliberately discards the depth information which results in poor performance for the intended task [4]. 3D CNNs deal with this by utilizing a three-dimensional filter to perform convolutions, so it is able to process a sequence of 2D images as input. The following is the architecture that I compiled and tested for my 3D CNN model:

Layer (type)	Output Shape	Param #
conv3d_3 (Conv3D)	(None, 8, 222, 222, 32)	2624
max_pooling3d_3 (MaxPooling 3D)	(None, 4, 111, 111, 32)	0
dropout_4 (Dropout)	(None, 4, 111, 111, 32)	0
conv3d_4 (Conv3D)	(None, 2, 109, 109, 64)	55360
max_pooling3d_4 (MaxPooling 3D)	(None, 1, 54, 54, 64)	0
dropout_5 (Dropout)	(None, 1, 54, 54, 64)	0
flatten_2 (Flatten)	(None, 186624)	0
dense_5 (Dense)	(None, 64)	11944000
dense_6 (Dense)	(None, 1)	65

3.2. Pre-Trained CNN

The second architecture I implemented was a pre-trained CNN. Transfer learning is a powerful tool in which models are trained on large amounts of data and the weights from that pre-training can be loaded and used for a new application. In this project, I tried out Keras's pre-trained EfficientNet offering with the 'imagenet' weights. EfficientNet is a CNN model that was designed to achieve high accuracy with fewer parameters and less computation compared to other models. [5]. Because of this, they are a popular model for video classification since training heavy video data sets can be time consuming and computationally expensive. [3]. I chose to try this model since for these reasons, and this is the final architecture I tested:

Layer (type)	Output Shape	Param #
rescaling_5 (Rescaling)	(None, None, None, None, 3)	0
time_distributed_3 (TimeDistributed)	(None, None, None, None, 1280)	4049571
dense_4 (Dense)	(None, None, None, None, 1)	1281
global_average_pooling3d_1 (GlobalAveragePooling3D)	(None, 1)	0

3.3. CNN-RNN

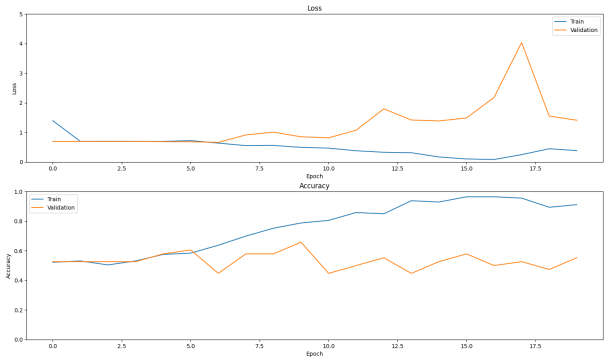
The final model architecture I tested was a pre-trained CNN whose outputs are fed to an RNN. Since RNNs allow information from previous inputs to persist, adding layers with RNNs after a CNN allows the model to remember information from past frames. In my model, I tried using Long Short Term Memory RNN layers since they address RNN's "vanishing gradient" problem where context for the earlier parts of the inputs are lost as the RNN progresses [6]. For the pre-trained CNN layer, I tried using the VGG16 model to see if that created different results than EfficientNet; I chose VGG16 since it is another model that is frequently used in image classification tasks and performs well in transfer learning applications [7]. The following is the architecture from my CNN-RNN model:

Layer (type)	Output Shape	Param #
time_distributed_7 (TimeDistributed)	(None, 10, 7, 7, 512)	14714688
time_distributed_8 (TimeDistributed)	(None, 10, 25088)	0
lstm_4 (LSTM)	(None, 10, 256)	25953280
dropout_10 (Dropout)	(None, 10, 256)	0
lstm_5 (LSTM)	(None, 256)	525312
dropout_11 (Dropout)	(None, 256)	0
dense_11 (Dense)	(None, 1)	257

4. RESULTS

4.1. 3D CNNs

After compiling the above model architecture for 3D CNNs, I fit the model with the training and validation data sets. Below are the plots for the training and validation loss and accuracy:

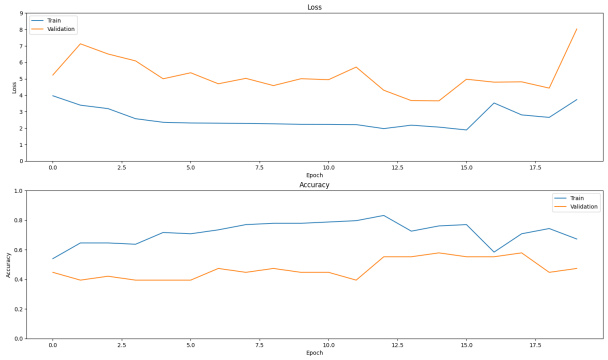


These plots show the training for 25 epochs, and I included them to show how the training and validation scores start to diverge significantly starting at epoch 5. When I fit this model with the early stopping callback function, it usually exited around epoch 3. When I added the dropout layers with 0.25 dropout after the maxpooling layers, it would make it to epoch 5.

The validation accuracy always hovers around 50 percent, and this is present in the test accuracy as well. When I tested the model on the test set, I got 0.50 as the accuracy score. This is not a great result because 50 percent is exactly what we would expect from a random guess.

4.2. Pre-Trained CNN

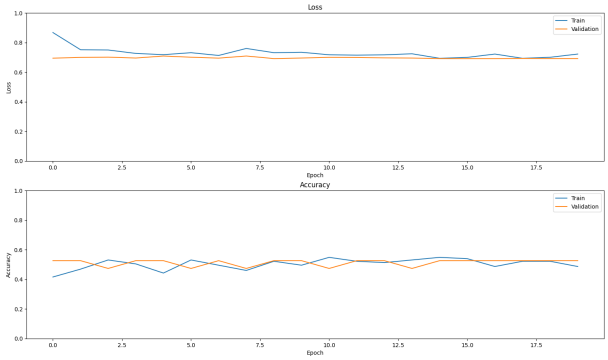
The pre-trained EfficientNet CNN model also sees a large gap between the training and validation data sets in the loss and accuracy score:



This model is similar to the prior 3D CNN model in that the validation accuracy score does not reach much higher that 0.60 while the training climbs closer to 1. In both models, the validation accuracy is a relatively linear line, which the EfficientNet score ticks slightly closer towards 0.60 around epoch 12. The test accuracy score for this model is higher than the previous model at 0.60. This is better than a random guess but not by much.

4.3. CNN-RNN

My final model architecture is different from the last two in that the training and validation loss and accuracy scores remain extremely close to each other throughout the epochs. While it is usually good to not see overfitting happening, it does not matter too much in this context since the model seems to just be making random guesses at each trial. The test accuracy score of 0.50 also reflects this.



5. DISCUSSION

After training three separate architectures for video classification with deep fake data, I found that the pre-trained CNN model EfficientNet with 'imagenet' weights performed the best with a test accuracy score of 0.60. This score is slightly better than a random guess, which is the equivalent of what the 3D CNN and CNN-RNN models outputted. I am surprised by this outcome- I was expecting the CNN-RNN model to perform the best given that it is a combination of feature detection with pre-trained CNN weights and temporal memory with LSTM RNN layers. I expect that these models would perform better with a larger amount of training data. I used a subset of the available deep fake video data set because I was worried about memory and computation cost when training on my instance. The subset that I chose did not cause problems on the Google Colab GPU I used, so I should have selected a larger amount of videos. If I were to try training these models again, I would select a larger amount of training data and see if that improves the accuracy scores. This might also address the overfitting I saw in the 3D CNN and EfficientNet models, since a more diverse selection of training data would make generalizing to the validation and test data easier.

6. REFERENCES

[1] Tiffany Hsu, "As deepfakes flourish, countries struggle with response," Jan 2023.

[2] Adam Satariano and Paul Mozur, "The people onscreen are fake. the disinformation is real.," Feb 2023.

[3] "Load video data nbsp;: nbsp; tensorflow core," .

[4] Hasib Zunair, Aimon Rahman, Nabeel Mohammed, and Joseph Paul Cohen, "Uniformizing techniques to process ct scans with 3d cnns for tuberculosis prediction," 2020.

[5] Mingxing Tan and Quoc V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[6] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," 1997.

[7] Karen Simonyan and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.