CS 230 Final Report: Deepfake Video Detection

Aleksander Dash, Nolan Handali
{adash, nolanh}@stanford.edu

Abstract

In this paper, we investigate several approaches towards detecting deepfake videos. Misinformation is rampant in today's world, and one method of spreading misinformation gaining in popularity is deepfakes, or facial manipulation in images and videos. We experiment with the usage of combining a CNN to generate an embedding from facial features with an RNN for temporal structure between frames within a video. Additionally, we make use of facial detection systems to focus solely on faces within frames of video, and ignore the rest of the image. Our model managed to achieve a 72.5% accuracy on the validation set.

1 Introduction

Deepfake detection is becoming a much more popular topic among today's computer vision world. Deepfakes refer to when a performance by an actor is superimposed onto a photo or video of a target person to make it appear like the target is performing the actions that the actor is doing. The creation of deepfakes has been enabled by recent AI/ML advances, and modern deepfakes are virtually imperceptible from real people to human eyes. This technology is devastating to people targeted by them, as politicians can be made to give speeches they never would have, archive footage can be doctored, or celebrities can be superimposed onto pornographic footage. It is therefore important that there exist robust algorithms to distinguish real photos or footage from deepfakes. Detecting deepfakes is interesting, as they are rapidly becoming more prevalent in today's world, have serious potential for harm, and is an extremely difficult task for humans to perform unaided.

2 Prior work

In Sabir et al [1], the authors use a recurrent-convolutional network to process video frames, followed by a simple feed-forward net to classify the video as real or fake. We plan on modifying this approach by experimenting with different networks, as the original paper used ResNet and DenseNet as their base networks. Additionally, for detecting manipulated still images, Hsu et al [2] use a Common fake feature network to extract facial features of the images, followed by a classification network to detect deepfakes.

3 Data

The dataset comes from Kaggle, which is currently running a competition where they provide a 500GB dataset of real and deepfake videos along with a label indicating whether the video is a deepfake or not. In the interest of computational feasibility, we are currently working on smaller segments of the dataset for computational viability.

For our baseline that trains on a single frame per video, we extract the first frame of video. For our baseline model that trains on multiple frames within a single video, we extract every six frames of video. Since all videos in the training and test sets are 10 seconds long at 30fps, this corresponds to 300 frames in total, of which we extract 50 per video. Each video is either 1920x1080 (landscape) or 1080x1920 (portrait). Since this is more detail than necessary, we will locate just the faces of the subjects in the video and rescale them to 224x224 pixels for training and testing.

In our final model, we train on a 60GB subset of the dataset. Since our model makes use of the Inception network for facial embeddings, we pass video frames through a facial detector and rescale the face with the highest confidence to be 299x299 pixels. We select every 30 frames of video, meaning that for each video file we train on 10 frames. This was done for purposes of computational feasibility – the facial detection could only run on the CPU and was easily bottlenecking training, so we needed to select a trade-off between number of frames per video and number of videos in order to ensure we had a varied enough dataset to train a robust model on, while still preserving enough input signal per training example.

The dataset is imbalanced. In the training set of 400 videos, 323 of the videos are fake and only 77 videos are real. In a slightly larger 11GB slice of the dataset we have 1248 fake videos and 86 real videos. In general, a very small fraction of the dataset consists of real videos, and each real video has been deepfaked multiple times, which explains the imbalance.

Our final model was trained on a 60GB slice of the dataset. Since the original dataset was imbalanced, for ease of training we created a balanced training set with 2400 real videos and 2400 fake videos. The validation set was similarly balanced, with 240 videos of each class contained within.

4 Approach

4.1 Baseline

We created two baseline models.

For the first baseline model, we extract the first frame of each video and pass it through a pretrained VGG-16 network to extract 512-dimensional embeddings. We pass these embeddings through a 4-layer Linear-ReLU network and predict two classes – real (0) or fake (1).

For the second baseline model, we extract every six frames of video. Since the videos are all 10 seconds long at 30fps, this corresponds to 50 frames per video. We detect a face on every frame of the video (see the next section for details) and crop the input image to just contain the face, before resizing it to a size of 224x224 using Lanczos resampling (which gives a higher image quality than nearest neighbor or bilinear interpolation). We then pass all frames through a VGG16 network to get a 512-dimensional embedding, pass these frames through a 4-layer network with ReLU activations and BatchNorm, and average the predictions across all the frames of each video to get the final predicted value for that video. Both models were trained with BCE Loss.

4.2 Face Detection

We make use of the OpenCV CascadeClassifier to detect faces in frames of video. For each video, we extract a certain number of frames from it. We pass each frame of video through the classifier and it returns a list of candidate bounding boxes for faces with corresponding confidence scores. If we are searching for faces on the first frame of video, we choose whichever bounding box has the highest confidence score (and if the face detector did not detect any faces on the frame, we arbitrarily pick a bounding box in the center of the image). If we are searching for faces on any subsequent frame of video, there are two possibilities:

1. The classifier did not detect any potential faces on this frame of video, so re-use the bounding box from the previous frame. (This heuristic works well in practice: most of the videos feature stationary subjects, so if we correctly detected the person's face in a previously analyzed frame, then it is likely that their face is still contained within that bounding box several frames later).
2. The classifier detected one or multiple potential faces on this frame of video. If (p_x, p_y) is the center of the bounding box we used for the previous frame of video and we detected K potential bounding boxes for the current frame of video, then for every triple $(cx_i, cy_i, \text{conf}_i)$ where (cx_i, cy_i) is the center of a potential bounding box and conf_i is the face detector's confidence that this bounding box corresponds to a face, we compute a score

for all $1 \leq i \leq K$:

$$\text{score}_i = \frac{\sqrt{(p_x - cx_i)^2 + (p_y - cy_i)^2}}{\text{conf}_i}$$

Essentially, we compute the Euclidean distance between the center of last frame's bounding box and the current frame's bounding box. Since people don't move too much from frame to frame we assume the position of their face will be relatively similar across frames, so in general we would choose the bounding box whose center is closest to the previous frame's bounding box, but we scale this distance by the model's confidence that the bounding box corresponds to a face so that if the model previously didn't see the face and classified a background element as a face and only on this frame detects a face, we're not stuck forever tracking the background element.

Here is a sample image with the face detection added.

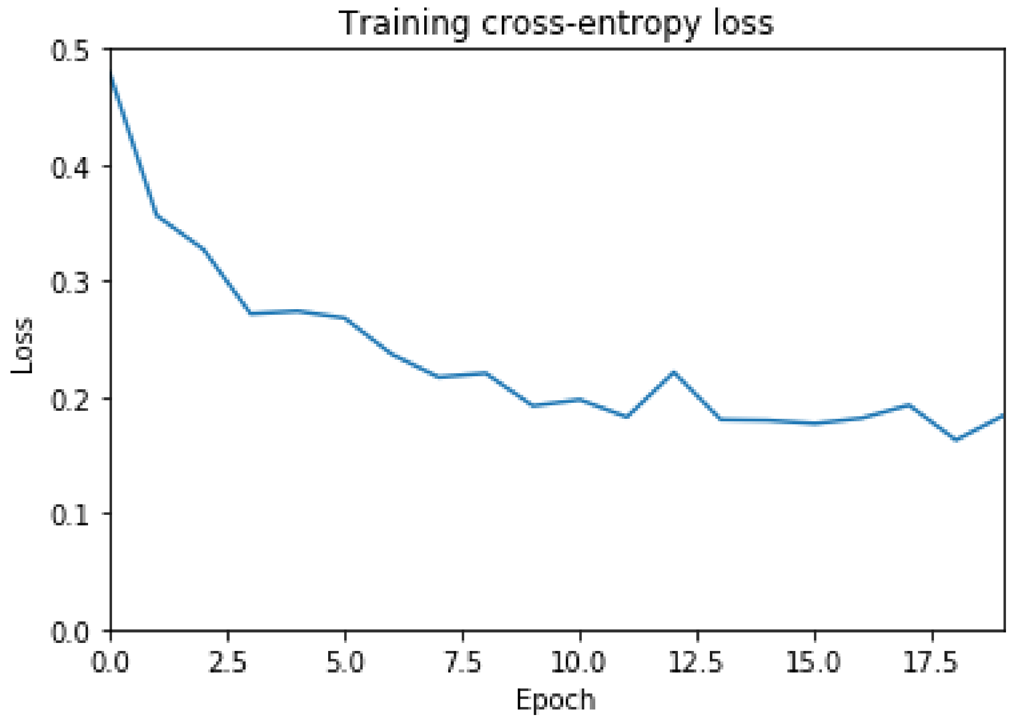
4.3 Model

The main model that we ended up focusing on used a combination of a CNN to extract underlying features from the facial images, and then a LSTM to extract temporal information. For our model, we reshaped the boxes extracted by our facial recognition to 299 x 299, which we then pass into InceptionV3. We remove the final linear layer of the Inception model, leaving us with a 2048 dimensional embedding of the facial image. In order to extract temporal information, we extract this embedding for all of the frames of the video, and we then pass this sequence of frames into an LSTM with hidden size of 512. Finally, we take the final output from the LSTM, and pass that into a linear layer with output size 2 to get the final classification. When training this model, we used binary cross entropy loss, and froze the weights in the Inception model, using the weights that were pretrained on Imagenet, and taken from pytorch's pretrained models.

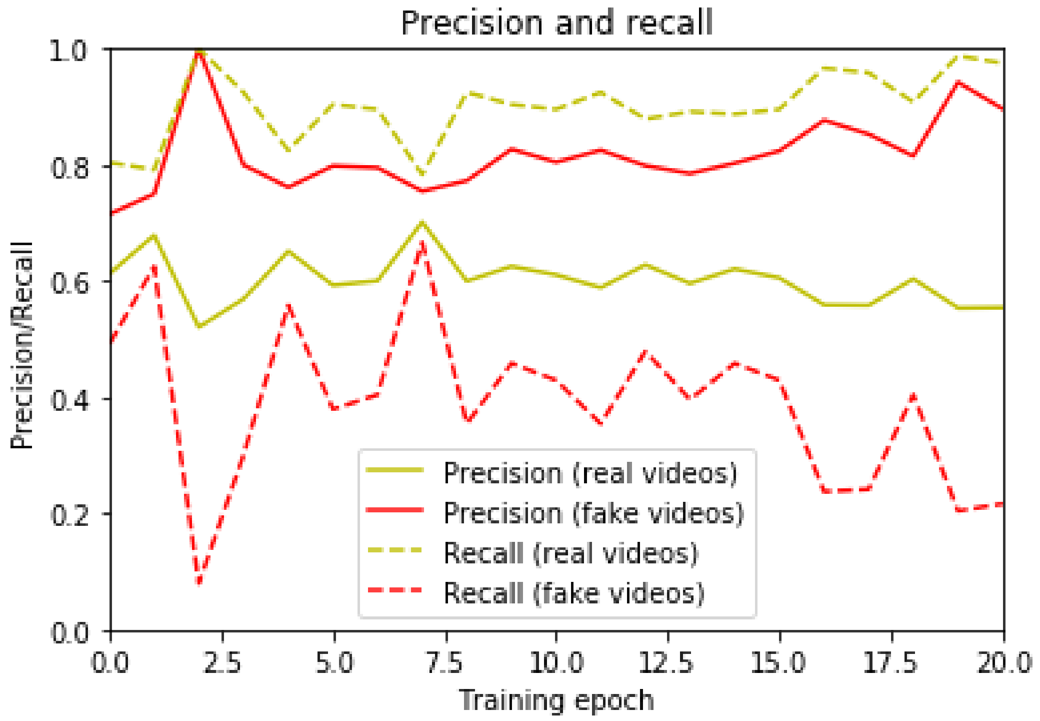
5 Evaluation and Success

Our baseline models did not perform well on the balanced training sets. They were both able to achieve around 85% accuracy on the original training sets we used, however, since around 80% of the videos were fake this is not a noteworthy accomplishment. Even with a penalised loss function to attempt to rebalance the models' priorities, the baseline models did no better than random guessing on a balanced validation dataset, as they only learned to predict the majority class of fake videos on the training set.

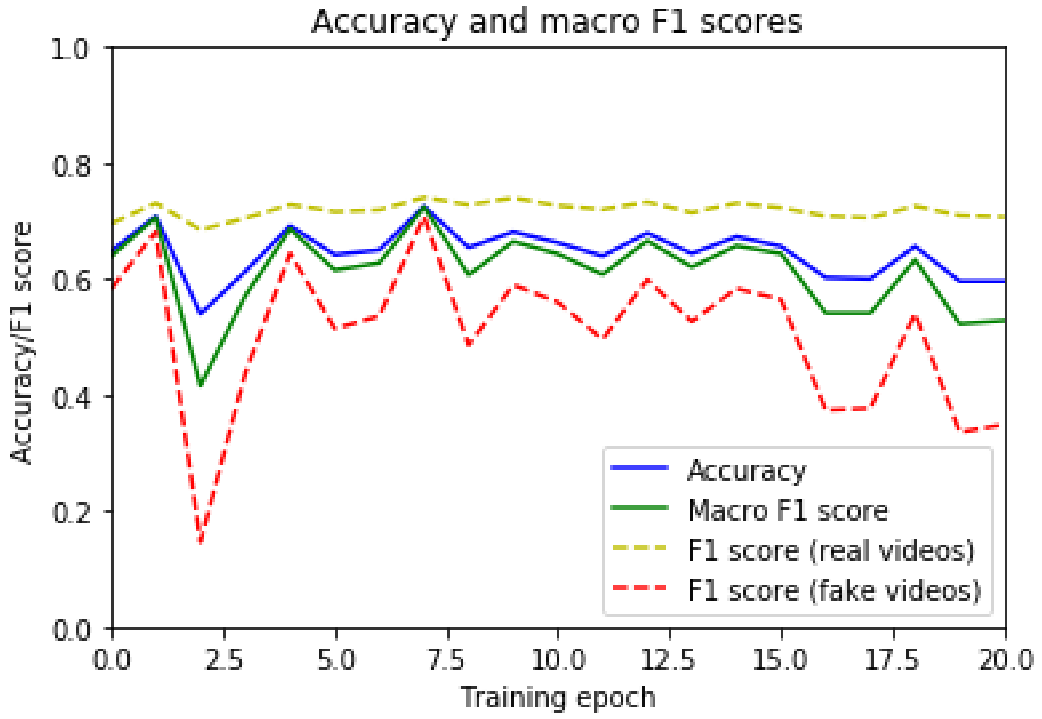
Our main model was trained for 20 epochs with Adam optimization with a learning rate of 0.0005, default β_1 and β_2 parameters, and a batch size of 16. Here is its loss curve:



Past epoch 7 the model starts to overfit to the training data at the expense of its performance on the validation set. We can see this if we look at the model's precision and recall for both the real and fake videos in the validation set:



And here is a plot of the overall accuracy of the model with each training epoch on the validation set, as well as its F1 scores for both the real and fake videos:



Past epoch 7 the model overwhelmingly predicts videos in the validation set as real. Hence, it has a high recall for real videos, and a terrible recall for fake videos. Epoch 7 represents the best balance of the model's predictions as well, as both the macro F1 score, overall accuracy, precision for real videos and recall for fake videos is maximised here, while both precision for fake videos and recall for real videos are still acceptably high. Undoubtedly we could have pushed this even further with hyperparameter optimization, however, since each training and evaluation session took a full day on an AWS p2.xlarge instance, we were happy with the results we obtained.

6 Future Work

With more time to work on this project, we would explore multiple other strategies for detecting deepfake videos. For example, we only tried a single method of extracting temporal information using LSTMs, but other methods, such as 4 dimensional convolutions over time have also been studied, and are something else we could explore. Furthermore, we could make use of GANs to first generate deepfakes, and then use the discrimanator model to try and detect which videos have been deepfaked. Finally, with more time, we could have more time to do hyper parameter tuning and experimentation with the model we did end up choosing.

7 Contributions

Aleksander worked on the facial recognition video processing pipeline, the baseline models, and the model setup and training on AWS. Nolan worked on the final Inception-LSTM-Linear model design, model validation, and creating the plots and diagrams shown throughout this paper.

8 Code

All code can be found at the following github repository: https://github.com/aleksanderdash/cs230

References

[1] Ekraam Sabir, Jiaxin Cheng, Ayush Jaiswal, Wael AbdAlmageed, Iacopo Masi, and Prem Natarajan. sdc. *Interfaces (GUI)*, 3:1, 2019.

[2] Y.; Lee C. Hsu, C.; Zhuang. Deep fake image detection based on pairwise learning. *preprints*, 2019.