In Search of a Generic Deepfake Detector

Sigurthor Bjorgvinsson
Department of Computer Science
University of Maryland
College Park, MD 20742
sigurtho@cs.umd.edu

Abstract

Detecting Deepfakes uploaded to the internet will become a crucial task in the coming years as Deepfakes continue to improve. An undetected Deepfake video could have immense negative consequences on our society because of the pace and spread of information consumption. A general Deepfake could be applied to videos across information platforms to identify whether a video is authentic. Wanting to tackle this problem, I attempted to create a general detector using Deep Learning techniques. I was able to improve the accuracy of today's best detection models on this particular dataset using two different models. In addition I implemented a recurrent network inspired by other authors to evaluate its generalizability on the multi-faker dataset I was working with. A generalized Deepfake detector was not found.

1. Introduction

In this paper, I describe the methods I used to try to create a generic Deepfake detector. Deepfakes are videos that have been manipulated in some way that changes the perception of the watcher. Deepfakes fall under a big umbrella of general manipulation to videos but this paper focuses on Deepfakes in which human faces have been edited (here on referred to as Deepfakes). Deepfakes that edit faces in/into videos can be very damaging to him or her. An actor (i.e. the origin) can make a victim (i.e. the face that is being edited in/into a video) say anything. The audience would then perceive that the actor/origin is doing something he or she never would. Given how important faces are for humans, this type of fake video can have a significant impact on public opinion of the victim and could forever damage their reputation.

There are two main approaches of creating Deepfakes today. These are facial expression manipulation and facial identity manipulation. Facial expression manipulation allows transferring facial expressions of an actor to victim while preserving the victim's identity in the video. Facial identity manipulation is where an victim's face is placed on top of the actors faces so that it seems that the victim is the one in the video.

In the following sections, I describe the dataset used in this research, previous work in this field that provide a benchmark for working toward this goal, the training pipeline developed, methodologies, results, and finally, future work.

2. Dataset

Face Forensics [1] provides a video dataset of news anchors with 1000 videos. Each video has 4 versions; Pristine, Face2Face, FaceSwap and DeepFakes (note the big F here for the faker), where the 3 changed versions is the name of the 'faker' that was used to manipulate/replace the face in the video. This dataset exists in 3 versions where the video can be Raw, High Quality (compression rate of 23) or Low Quality (compression rate of 40). All training, testing and validation was done using the High Quality dataset. Every faker has their own unique way of creating Deepfakes.

Face2Face is a facial reenactment system that performs facial expression manipulation. Face2Face transfers facial expressions from an actor to a victim while keeping the identity of the victim in the video the same using computer graphics techniques.[2]

FaceSwap is a facial identity manipulation tool where the entire face of an actor is replaced with a victims face. FaceSwap only needs a set of images to extract the victims face which makes it very easy to use. FaceSwap applies face alignment, Gauss Newton optimization and image blending when swapping faces.[3]

DeepFakes uses an encoder-decoder network trained to generate the victim's face that matches the facial expressions of the actor. Because Deepfakes applies a

Deep Learning approach, DeepFakes requires much more data than the computer graphics approaches above to get reasonable results.[4]

Because this dataset is built using multiple different fakers, it allows us to use it to evaluate how well the models generalize by training on one faker and testing it on another.

Each approach described in later sections required different data formats and inputs. All the methods described in this paper, except 'RNN on Facial Landmarks', shared the following preprocessing method. Any additional preprocessing methods will be described in their respective sections.

Figure 1. Examples of the same frame for different fakes; Top Left: Pristine, Top Right: Face2Face, Bot Left: FaceSwap, Bot Right: DeepFakes

2.1. Preprocessing

Working with videos is generally considered difficult because of the amount of data and processing required. The amount of processing needed to decode the video repeatedly required looking into other options. Because I only care about the faces in the video, the dataset would end up being cropped images of the face in the video. To prepare a video the following steps were performed:

1. Extract facial landmark locations for every frame in the videos.
2. Compute a square face detection bounding box while maintaining the aspect ratio of the face.
3. Crop the square face detection bounding box from the image frame.
4. Resize cropped face to $128x128$ pixels.
5. Save cropped face to disk.

An example cropped image from the same frame from the same video but a different faker can be seen in figure 1.

2.2. Data split

The dataset includes 1000 distinct Pristine videos. The dataset is split $70/15/15$ for Training, Validation and Testing sets. Having a well balanced dataset is important when training a network. Imbalances might lead to the network being more inclined to predict the class that has more datapoints and would need to be accounted for by using regularization terms during training.

Because the length of the videos in our dataset can vary between fakers, it could make the dataset unbalanced. To ensure a $50/50$ balance of classes (Pristine/Fake), I only extract frames that exist in both fakers. For example, if Pristine video has 120 frames but faker video has 100, only 100 frames were extracted from the Pristine.

3. Related Work

Previous work in this area is well summarized in FaceForcensics++[1]. The authors compile the results of previous methods which shows that they currently have the best accuracy. In that paper, the authors use a XceptionNet based on separable convolutions with short-cut connections. They fine tune their network after initializing it with pretrained ImageNet weights. A comparison of their results and my results will be discussed in the results section.

The authors of [5] report good results when training a Recurrent Neural Network (RNN) on a feature map from an InceptionV3 network pretrained on ImageNet. They report results on a custom dataset which makes it hard to compare their results to mine and evaluate their ability to generalize between fakers. In my search for a generic Deepfake detector I implemented their network to compare it to other methods and see if they had indeed created a generic Deepfake detector.

4. Training pipeline

All the methods that are described in this paper were implemented using Python and TensorFlow [6]. Some methods take advantage of existing architectures like InceptionV4[7], ResNet50[8] and I3D[9]. The models all have in common that their outputs are a softmax of two output nodes what predict whether the input was Pristine or Fake.

When training, a train faker was selected and the paths to that fakers dataset. One iteration over that dataset (one

epoch) was followed by a prediction pass using the validation set. The validation sets for all fakers were passed through one by one so that the accuracy for different fakers could be monitored. Early stopping was used by selecting the epoch with the best validation accuracy.

5. Methodologies

In this section I will describe the methods I took to work toward a generic Deepfake detector. For each method, I describe the general idea, preprocessing step, the model used, and then the results.

5.1. RNN on facial landmarks

One method attempted was training a Recurrent neural network (RNN) on the facial landmarks in the video. Facial landmarks are points (x, y coordinates) in the frame that build up a face, like the tip of the nose, points around the jaw, mouth, eyes and eyebrows. A sequence of these coordinates would be the input to the model and the output would be a prediction if the sequence was Pristine or Fake.

Preprocessing: To extract all of the coordinates from the 4000 videos, each frame had to be passed through a face detector. The face detector generated 64 facial landmarks for each face in the frame. If the detector did not detect a face in the frame, no further frames were processed of that video so that only consecutive frames of datapoints were included. If the face detector detected multiple faces, the largest face was always selected. These points were then dumped into a pickle binary storage file.

Model: As input, the model took in a sequence of 64 coordinates (128 vector). The sequence length was configured in my tests to 24 which was passed through a 25 node wide RNN layer followed by a single fully-connected layer with 25 nodes. Different configurations of the layer sizes were attempted without any notable change in the results.

Results: This model unfortunately did not end up learning anything. In hindsight, a much larger network was most likely needed here. This network needed to be able to capture what natural facial expressions and the transition between expressions are from the facial landmarks coordinates. When this approach was implemented and trained, the model was unable to learn what these natural face movements looked like. A network of this size most likely had no chance to be able to capture that and was most likely the reason for it's failure. This will be part of future work to attempt this approach again.

Another reason for why this model did not work could have been the video quality. When a frame is passed one by one through the face detector, the low quality video makes it difficult to get precise landmark coordinates. The low quality might introduce jitters into the data, making it quite noisy.

Finally, data normalization would have been necessary. The coordinates individually have no real meaning to the network; only the relation of the points between sequences matters in this approach. It should not matter if the face is in the top left or top right but they would have drastically different coordinates. Subtracting the first 64 coordinates from all of the coordinates in the sequence would have been the first approach to attempt.

5.2. InceptionV4

Following the same ideas as in [1], I implemented a network to predict Fakes for frame-by-frame cropped faces using an InceptionV4 model from the slim models package [10]. Before training, the model was initialized using the pretrained weights that were trained on ImageNet.

Preprocessing: The cropped face was resized to $299x299$ and the RGB values were normalized to the range $[-1, 1]$. During a prediction pass, no data augmentation was performed. During training, the input was randomly aspect ratio distorted, cropped, flipped horizontally and color distorted.

Model: As stated earlier, the model used here was the InceptionV4 model but where the last layer (output layer) was cut off and fitted with two output nodes. In between the InceptionV4 model and the output nodes, I put a 0.5 dropout layer followed by a batch normalization layer. When I trained this model I did not freeze any of the layers, on the contrary of what was done in [1], and trained all of the weights in the network from the first epoch.

		Validation Set		
		Deepfakes	Face2Face	FaceSwap
Train Set	Deepfakes	**99.4%**	52.07%	49.29%
	Face2Face	53.73%	**98.31%**	50.34%
	FaceSwap	50.84%	51.15%	**98.65%**

Table 1. Validation accuracy from training the InceptionV4 network on a single faker, frame-by-frame

Results: Table 1, shows the accuracy for the validation set on InceptionV4 model. The prediction is which ever node has a higher value after performing softmax on the two output nodes. Accuracy is the percentage of predictions that match the datapoints label. These results shown in in table 1 were the best validation accuracy and it only took 3-4 epochs of training, or 13 hours, depending on the faker. I only needed 4 epochs because I only needed to fine-tune the model.

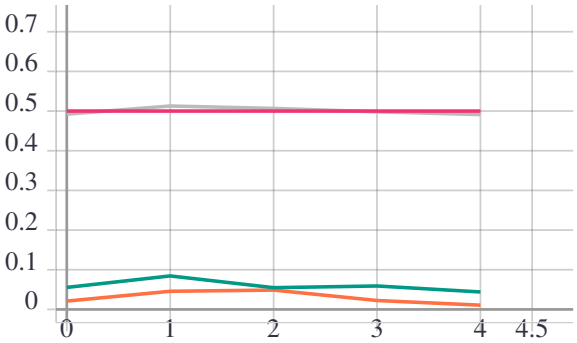

Figure 2. Graph showing the average prediction (1 on left axis means all predictions would be Fake while 0 would mean all predictions are Pristine) over all the Face2Face training set and all of the fakers validation set when training the InceptionV4 model. Pink: Face2Face-Train, Gray: Face2Face-Val, Green: DeepFakes-Val, Orange: FaceSwap-Val

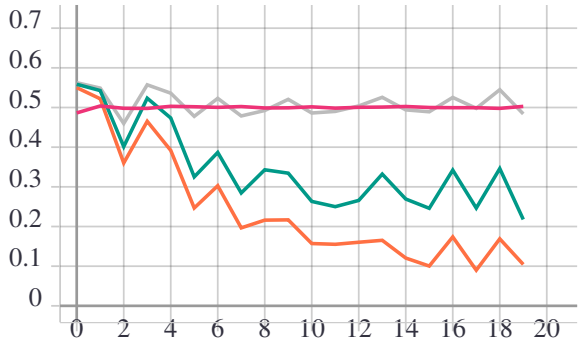

Figure 3. Graph showing the average prediction over all the Face2Face training set and all of the fakers validation set when training the I3D model. Pink: Face2Face-Train, Gray: Face2Face-Val, Green: DeepFakes-Val, Orange: FaceSwap-Val

Given a dataset set of a specific faker, training a model to detect that faker is fairly easy. But the model does not generalize between fakers at all.

The trained model is only staying around 50% for fakers it is not being trained on because it starts to only predict Pristine. This can be visualized in Figure 2 where the average prediction is plotted. The average prediction tells us what average Fake prediction is over the entire fake dataset for a given epoch. If the average prediction goes close to 0, almost none of the datapoints are being predicted as Fake.

This, and the accuracy, tells us that the model latches on to features of the faker that characterizes only that faker and does not characterize any other faker.

Note: The average prediction start at 0 in figure 2 for non-training fakers because after the first epoch, the model is already at 97.82% validation accuracy. At that point the model has already been tuned to detect the specific faker and predicts only Fake for that faker.

5.3. Kinetic-I3D

In section 5.2, an entire dimension of the dataset was not used at all. When frames are passed though the network one by one, the time dimension is completely ignored. The Kinetics-I3D[9] is a network that takes in sequences of RGB frames as a single datapoint.

Preprocessing: The number of frames in a sequence used was 79 with an image size of $224x224$. During training, custom data augmentation was applied which included a 50% random chance to do a horizontal flip, random rotation, random scaling and random cropping, random color distortion and random Gaussian noise.

Model: The original Kinetic-I3D model takes in 2 streams of inputs, a sequence of frames stacked in time order and the optical flow of those frames. In action classification, which Kinetic-I3D is used for, optical flow adds a lot of information because it allows for easier tracking of movement between frames.

After generating optical flows for a couple of videos in the dataset, it was decided that it would likely not add much information when detecting Deepfakes. The computational requirements of creating the optical flow sequence for all the videos was also a factor. The sequence of images becomes quite large in memory so I could only fit a batch size of 3 on my RTX 2080 TI at one time. Keeping the model simple and runnable was more important for me than using the optical flow frames. The code for the Kinetic-I3D was cloned from deepminds github repository[11] with minor edits to only use RGB frames and support for 2 output nodes.

		Validation Set		
		DeepFakes	Face2Face	FaceSwap
Train Set	DeepFakes	**99.17%**	56.61%	50.26%
	Face2Face	67.92%	**98.41%**	46.56%
	FaceSwap	51.25%	53.44%	**98.41%**

Table 2. Results from training the Kinect-I3D network on a single faker

Results: The I3D model was not able to generalize either but did get much better results than expected. I3D does generalize marginally better between Face2Face and DeepFakes as can be seen on the first two rows in table 2.

The better generalization can also be seen in the average prediction plot in figure 3 when compared to figure 2. The

Train Set \ Validation Set	Deepfakes	Face2Face	FaceSwap
Deepfakes	**83.40%**	52.34%	48.49%
Face2Face	56.72%	**75.78%**	56.04%
FaceSwap	53.63%	55.78%	**77.40%**

Table 3. Validation accuracy from training the RNN on Feature Maps on a single faker with feature maps from ResNet50

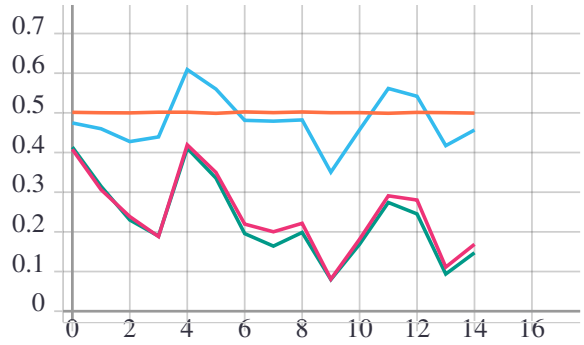

Figure 4. Graph showing the average prediction over the DeepFakes training set and all the fakers validation set when training the RNN model. Orange: DeepFakes-Train, Blue: DeepFakes-Val, Pink: Face2Face-Train, Green: FaceSwap-Val

average prediction for the fakers that are not being trained are not as close to 0 as when InceptionV4 was trained.

5.4. RNN on Feature Maps

Authors of [5] reported great results using a very simple model. In their paper, they only report a single accuracy on their entire custom dataset. Wanting to investigate their generalizability, I implemented and tested their approach on the FaceForensics++ dataset so that it could be compared fairly. Because I was not getting the same results they were getting, I decided to run multiple different experiments with different pretrained models and different model configurations to figure out what worked best for my dataset.

Preprocessing: Additional data preprocessing was required for this approach as the RNN's input was not images, but rather the feature map (intermediate output) from a pretrained network. For each pretrained network, I passed all of the cropped faces through the model and stored the feature maps into TFRecords. TFRecords are binary storage formats which offer great speed and is a go-to storage format when working with Tensorflow.

Finally the feature maps needed to be combined into sequences to form the datapoints. For that I decided that the length of the sequence would be 100 because they reported in [5] that the longer the sequence, the more accurate their model was. The sequences were created with a step of 50, meaning that the 50^{th} frame of a video would be the 1^{st} frame in the second datapoint.

Since I had been working with other models than the authors of [5] I wanted to try using feature maps from other models. Using InceptionV4, ResNet50 and VGG, loaded with pretrained weights trained on ImageNet, I generated feature maps for every frame in my dataset from all 3 models. Out of all experiments, ResNet50 was the fastest to train, showed the most stable learning plots, and best accuracy.

Model: After running the experiments described above, the best performing model was the following. Using feature maps from ResNet50, the dimensions of the input to my RNN model was $[100, 2048]$. The RNN layer had 2048 hidden layers followed by a 0.5 dropout layer. Next came a 512 fully connected with 0.01 L2 weight regularizer followed by a 0.5 dropout layer and then finally the output layer. For training a learning rate of $01e \quad 5$ with the Adam Optimizer.

Results: Table 3 shows the results from training the best performing model at its best performing epoch on different fakers. After training on all fakers individually, this model was not able to generalize either. Even though the trained models do not start exclusively predicting Pristine for the other fakers they were not trained on, the RNN model does not get any better at predicting other fakers. As seen in figure 4 the average prediction for other fakes gets down to about the same as the I3D model except it does not show any indication of generalizability.

Another reason I expect I am not getting the same results is that my dataset is of lower quality (lower resolution). Their dataset is from videos online and from movies which can be retrieved of much higher quality. This would play a big factor in the accuracy my implementation is able to produce.

6. Result

In this section we will compare the results between the methods and compare them to the published accuracies. Authors of [5] created their own fake dataset which I did not have access to which makes it hard to compare their results to mine. Their best results were using a sequence of 4080 frames which resulted in test accuracy of 97.1%. Also, the authors make no indication of cropping out faces so the input into the pretrained models for feature map generation vary also.

Table 4 shows the results reported in FaceForensics++ [1] along side my best results. As can be seen there, my

	DeepFakes	Face2Face	FaceSwap	Other
RNN on Feature Map* [5]	N/A	N/A	N/A	97.1%
FaceForensics++ [1]	98.85%	98.36%	98.23%	N/A
Kinetic-I3D (Mine)	99.17%	**98.41%**	98.41%	N/A
RNN on Feature Map (Mine)	83.40%	75.78%	77.40%	N/A
InceptionV4 (Mine)	**99.4%**	98.31%	**98.65%**	N/A

Table 4. Comparison results from two state of the art papers and my methods. Faker column indicates that the model was trained and tested on that faker. This comparison is equivalent to what is being compared in [1]. The video quality is high quality (compression rate is 23). No generalizability is being compared between fakers. * *notes that results reported in paper not entirely comparable but used as a reference point.*

models beat their results by a small margin. Common across all of the approaches attempted is that they all latch on to specifics features of the faker on which they are trained. This can be seen in previous plots where the models stop predicting Fakes for validations sets of fakers that they are not trained on. The reason I believe this happens is that it is easier for the model to latch on to these specific features than it is to latch on to features that identify a non-edited face.

One approach to tackle this issue is to provide the model with more hand picked features. For example the approach described in section 5.1 RNN on Facial Landmarks has potential to work. Using partial facial landmarks has been shown to work in the past. For example, [12] the authors trained a model only using the eyes of the subjects and by that were able to identify fakes.

7. Future Work

As I have been working on this research topic for almost a year, and although the results are great in comparison with current literature, the desired results have not been achieved. At the time of this writing, many different competitions in this field are popping up like Facebooks Deepfake Detection Challenge. [13] I plan to partake in these challenges and continue to work on this topic in the coming future. Future work will also include attempting the 5.1 RNN on Facial Landmarks again using what I have learned through this research.

8. Acknowledgements

I would like to express my very great appreciation to Professor Abhinav Shrivastava for his assistance and guidance through my research on this topic.

References

[1] Rössler, A., Cozzolino, D. , Verdoliva L., Riess, C., Thies, J. & Nießner, M. (2019) FaceForensics++: Learning to Detect Manipulated Facial Images. *arXiv:1901.08971*

[2] Thies, J., Zollhofer, M., Stamminger, M., Theobalt, C. & Nießner, M. (2016) Face2Face: Real-Time Face Capture and Reenactment of RGB Videos. *In IEEE Conference on Computer Vision and Pattern Recognition*. pages 2387–2395, June 2016.

[3] FaceSwap github. `https://github.com/MarekKowalski/FaceSwap/`. Accessed: 2019-12-05.

[4] DeepFakes github. `https://github.com/deepfakes/faceswap`. Accessed: 2019-12-05

[5] Guera, D., & Delp, E.J. (2018) Deepfake Video Detection Using Recurrent Neural Networks. *In 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*. pp. 1-6. 2018

[6] Abadi, M. et al. (2015) TensorFlow: Large-scale machine learning on heterogeneous systems.

[7] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi A. A.. (2017) Inception-v4, inception-ResNet and the impact of residual connections on learning. *In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17)*. AAAI Press 4278-4284.

[8] He, K., Zhang, X., Ren, S., Sun, J. (2015) Deep Residual Learning for Image Recognition *arXiv:1512.03385*

[9] Carreira, J., & Zisserman, A. (2017) Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 4724-4733. 2017

[10] N. Silberman, et al. TensorFlow-Slim image classification model library. 2016. https://github.com/tensorflow/models/tree/master/research/slim

[11] Brian Zhang et. al. Deepmind Kinetics-I3D. https://github.com/deepmind/kinetics-i3d

[12] Li, Y., Chang, M. & Lyu, S. (2018) In Ictu Oculi: Exposing AI Generated FakeFace Videos by Detecting Eye Blinking. *arXiv:1806.02877*

[13] Deepfake Detection Challenge, https://deepfakedetectionchallenge.ai/. Accessed on 12/08/2019