Final Project Report for CS 175, Spring 2018

Project Title: Deepfake Detection

Project Number: The Boys

Student Name(s)

Steven Diaz de Leon, 13449791, diazdels@uci.edu

Jesus Zuniga, 32887236, zunigajf@uci.edu

1. Introduction and Problem Statement :

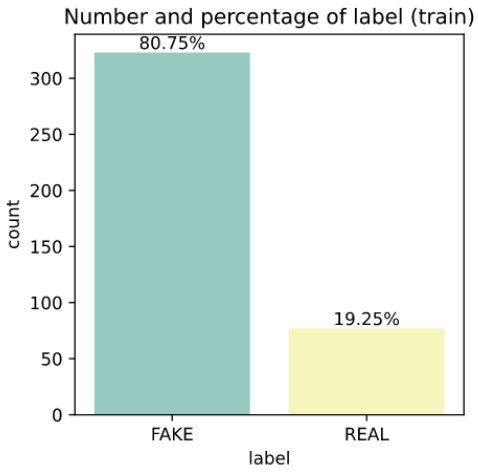
Deepfakes are defined as synthetic media in which a person in an existing image or video is replaced with someone else's likeness. As time progresses, deepfakes are becoming increasingly sophisticated and in some cases hard to identify for the average person. This will become a problem given that deepfakes may be used maliciously as a source of misinformation, manipulation, harassment, and persuasion. In this project we attempted several different techniques with the sole purpose of identifying deepfake videos.

We primarily used a MTCNN (Multi-Task Cascaded Convolutional Network) in order to extract the faces and the feature vectors with a static number of frames for all of the videos in the training set. We then proceeded to feed the face images to an InceptionResNet in order to tune it to our data set and properly classify the videos. The Inception network we used was originally pre-trained using the weights from the "vggface2" dataset, with this information we were able to achieve 80% accuracy on the test data set which consists also of 400 videos. We will go into further detail about the techniques we tried in the next sections.

2. Related Work:

Because Deepfakes are a relatively new creation so too is the problem of detecting them. As a result of this new problem the literature regarding detection is limited. However the

sub-problem of face detection in a video is a known problem and has been worked on. We primarily used the Multi-task Cascaded Convolutional Network(MTCNN) proposed by Zhang et al [https://arxiv.org/abs/1604.02878]. In this paper the authors provide ample results showing how the MTCNN face detection model outperforms previously state of the art facial detection models, FDDB and WIDER FACE. Because MTCNN outperforms the previous top models we decided to implement it in our face detection pipeline.



3. Data Sets

For the project we tried using the whole data set from the kaggle page, https://www.kaggle.com/c/deepfake-detection-challenge/data, but training over 400GB worth of videos was taking too long within google colab. We instead decided to use the provided sample data from the kaggle page. We then performed exploratory data analysis (EDA) on the sample data set in order to get a better understanding of the data we were working with. The data set consisted of 400 training and 400 test videos for a total of 4GB and 800 videos where each video was around 10 seconds of a 1920x1080 resolution. Furthermore, each data set contained a metadata JSON file with the labels {“FAKE”, “REAL”} for each video as shown below.

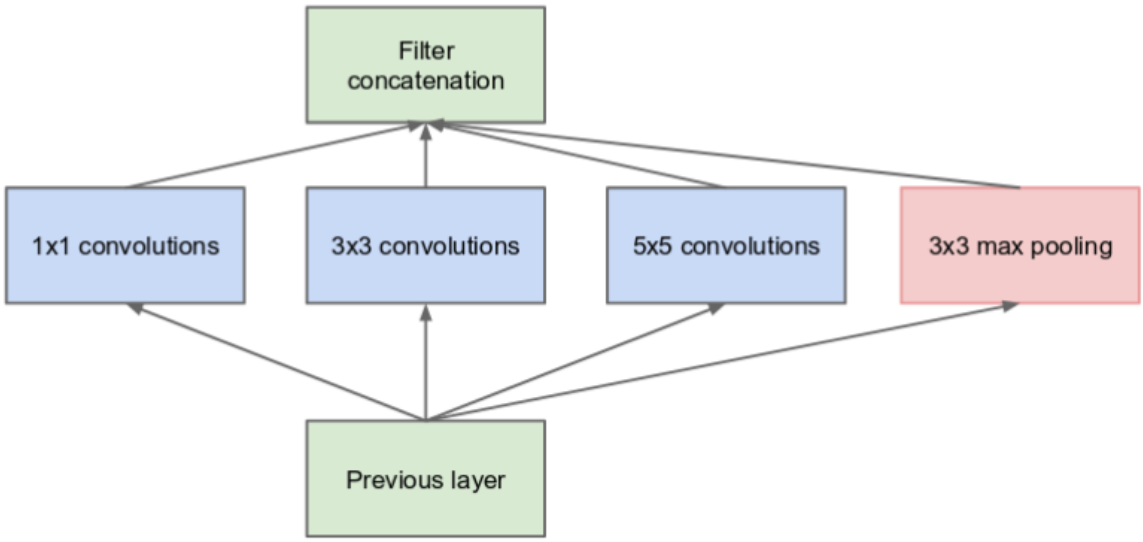
JSON file: metadata.json

	label	split	original
aagfhgtpmv.mp4	FAKE	train	vudstovrck.mp4
aapnvogymq.mp4	FAKE	train	jdubbvfswz.mp4
abarnvbtwb.mp4	REAL	train	None
abofeumbvv.mp4	FAKE	train	atvmxvwyns.mp4
abqwwspghj.mp4	FAKE	train	qzimuostzz.mp4

In addition our implementation uses a pre-trained inception based resnet on the CASIA-Webface and VGGFace2 datasets for facial detection. Cropping faces from the frames of the training videos using MTCNN and then using the pre-trained datasets to detect faces proved to be the best approach.

4. Description of Technical Approach

The detection pipeline uses the MTCNN network previously mentioned in order to crop the faces from each frame of the training video set. Once faces have been cropped they are then processed using the inception resnet from pytorch facenet “InceptionResnetV1”. The MTCNN works by using a cascaded structure with three stages of deep convolutional networks. These three networks consist of a proposal network, a refine network and an O network. They attempt to build candidate images of the face and facial landmarks such as the eyes, nose, and mouth in the provided images. The training of the model consists of three CNN detectors: face/non-face classification, bounding box regression, and facial landmark localization. Because there are multiple tasks used in each of these CNN’s there are different training images being composed in the training step. An overall learning target then needs to be implemented due to the multi tasked training. Finally MTCNN uses Online Hard sample mining, where in each mini-batch 70% of the top samples with the highest loss computed in the forward propagation step are set aside as hard samples. These samples are then used as the main set to compute the gradient. Because of this easy samples are ignored and the harder samples provide us with a stronger detector in training. The InceptionResnetV1 in pytorch is based on the GoogLeNet deep convolutional neural network https://arxiv.org/pdf/1409.4842v1.pdf. The inception model works by using multiple sized filters on the same level making a wider network opposed to a deeper one. Below is a sample naive model from the paper:



(a) Inception module, naïve version

Initially it performs a convolution on the input with 3 different filter sizes (1x1, 3x3, 5x5) followed by a max pooling. The output is then concatenated and sent to the next inception module.

4. Software

Our project was run in Google Colab using facenet-pytorch library for the MTCNN and Inception Resnet implementations. In addition we used sklearn toolkit to compute the accuracy scores and validation. We used a kaggle book linked in the table below as a starting point for our code.

a)

EDA:

Exploratory data analysis scripts used to analyze the data and visualize metadata files

Additional Training:

Script that allowed for fine-tuning the pytorch pretrained network to better fit the datasets.

b)

MTCNN:

Library that allowed us to properly detect faces in a video and return them so we can forward them to the inception network.

Inception Resnet V1:

Using this library we were able to experiment with a pretrained model with different data sets such as “vggface2” in order to gain a better understanding of how CNNs were used to detect anomalies in images.

implementation/code	library/resource
MTCNN	facenet-pytorch
Inception Resnet V1	facenet-pytorch
Sklearn	scikit-learn
Base kaggle notebook	Facial recognition model in pytorch

5. Experiments and Evaluation

For our evaluation we used sklearn and various models to fit the labels. Using the CASIA-Webface pre trained dataset we got the following results for the various listed models

Prediction Model	Accuracy
Gaussian Naive Bayes	0.641
SVM	0.807

Logistic Regression	0.801

For the VGGFACE2 pre trained model:

Prediction Model	Accuracy
Gaussian Naive Bayes	0.682
SVM	0.807
Logistic Regression	0.807

Both models seem to perform about the same with the VGGFACE2 model slightly outperforming the CASIA-Webface dataset. This follows the documentation provided in facenet-pytorch where the models have an LFW accuracy score of 0.9995 and 0.9965 respectively. So the minimal variation in accuracy is expected.

6. Discussion and Conclusion

We learned that detecting deepfakes is still a very difficult problem that will continue to get more difficult as new AI based deepfake techniques arise. As for the algorithms that we used, we got a better understanding of how convolutional networks can be used to solve image classification problems. One major issue with CNNs is that they don't encode the position and orientation of the objects when processing images and making predictions, this makes it difficult for CNN based algorithms to detect deepfakes when the person is facing away from the camera and hence for face detection algorithms to process those frames for the models. When looking at the future directions, one possible way to improve deepfake detection is to look at different features and fuse them together when training a model. One example of this is instead of only looking at facial landmarks and audio features separately, we could possibly experiment with ways of fusing those two kinds of features together in order to get a more accurate predictor.