CS230

You're Not Real: Deep Fake Detection

Final Report

John Kustin
Stanford University
kustinj@stanford.edu

Isaac Schaider
Stanford University
schaider@stanford.edu

Andy Wang
Stanford University
andy2000@stanford.edu

Abstract

The rise of AI in the recent decade has given way for deep fakes. In particular, deep fake, manipulated images/voice snippets/video, can now be produced with high precision using deep generative models. These fakes can be extremely damaging to individuals and society by artificially engineering people into compromising situations. It is critical that we find methods to defend against the proliferation of deep fakes by identifying them with high accuracy.

1 Introduction

Information is key in every modern endeavor. If the integrity of our information is jeopardized then many of the assumptions we make on a daily basis would be swept out from underneath us. A modern, *but already classic*, example of this disruption is BuzzFeed's YouTube sensation *You Won't Believe What Obama Says In This Video!* The video demonstrates how *deep fakes*, a computer generated image or video tuned to mimic a real individual, can be used to imitate global leaders. The principle motivation for pursuing this problem is the scale of disruption that a good deep fake can bring; the break of trust between a nation's leader and the populous can be a catalyst for mayhem. The future is not as gloomy as it may seem. Because deep fakes are *made* from generative models, we believe it is possible to detect video or imagery features indicative of deep fake generation. The input for our algorithm is a video. We then use a deep convolutional neural network to output a predicted binary classification: fake or real. [5]

2 Related work

In a 2019 report by Koshy and Mahmood [5], the researchers developed deep architectures for face liveness detection that use a combination of texture analysis and a convolutional neural network (CNN). Face liveliness detection is used for facial recognition for identity management and secure access control for many web and mobile-related software applications. An impostor can gain access to the system by presenting a copy of the image to the camera. Therefore, prior to face recognition authentication, face liveness detection is important to detect whether the captured face is live or fake. The researchers built 3 different architectures to test on the NUAA dataset: CNN-5, ResNet50, Inception v4. The team found that the Inception-v4 produced the best results with a test accuracy of 100%.

In a 2016 report by Szegedy, Ioffe, and Vanhoucke [8], the researchers give clear empirical evidence that training with residual connections accelerates the training of Inception networks significantly. They also provide some evidence of residual Inception networks outperforming similarly

CS230: Deep Learning, Spring 2020, Stanford University, CA. (LaTeX template borrowed from NIPS 2017.)

expensive Inception networks without residual connections by a thin margin. Additionally, they present several new streamlined architectures for both residual and non-residual Inception networks. These variations improve the single-frame recognition performance on the ILSVRC 2012 classification task significantly. We further demonstrate how proper activation scaling stabilizes the training of very wide residual Inception networks. With an ensemble of three residual and one Inception-v4, we achieve 3.08% top-5 error on the test set of the ImageNet classification (CLS) challenge.

3 Dataset and Features

We pulled our dataset from the online Kaggle Deepfake Detection challenge. [3] The dataset consisted of videos 4,529 videos, which we then sliced into individual frames within our model. Within these frames we particularly focused on the face and our model was trained to then recognize these faces as real or fake within the video dataset. The video resolution was 128 by 128 and within each extracted frame we normalized the values (Figure 1).

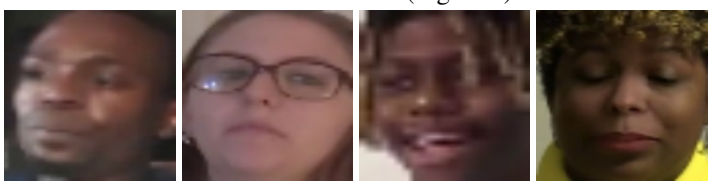


Figure 1: Example images from our data set

4 Methods

For our project we have tested 3 different models, the the Tensorflow implementation of the DNN from Coursera, the CNN implementation from Coursera with some minor modifications to hyper parameters, and the InceptionV3 network.

Deep Neural Network Model

Our first model to investigate the task of deep fake detection was a DNN since DNN's are great at learning high-level features. The DNN is fully connected so the model can learn complex features of our deep fake database. The DNN model follows the structure LINEAR -> RELU -> LINEAR -> RELU -> LINEAR -> SIGMOID. We suspected that the DNN would require a lot of computational resources and time due to its fully connected structure; indeed it did. Despite the time required to train and run the model, the model was accurate with respect to the distribution of our training set. Our choice of hyperparameters (minibatch size and learning rate) enabled linear regression with gradient descent to tune the inner layer weights in order to classify whether a given input image was fake (0) or real (1). We initially implemented the DNN using just python and numpy but eventually upgraded to TensorFlow in order to use a more optimized code base.

Convolutional Model

We decided to use a convolutional model that follows the structure CONV2D -> RELU -> MAXPOOL -> CONV2D -> RELU -> MAXPOOL -> FLATTEN -> FULLYCONNECTED. We think the CNN was a good approach for this project of detecting deep fakes for its ability to recognize features in images that can distinguish it from regular images. Our model works by starting with small windows and detects various simple features and becomes more complex as it moves through each layer ultimately reaching the fully connected layer where it gives a classification of 0 or 1 labeling it as a deep fake or not. We used an Adam Optimizer loss function for this method. [5]

Inception Network

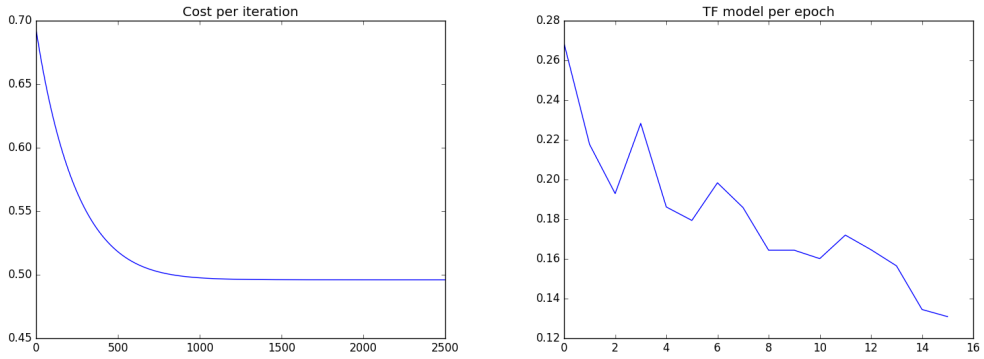
We noticed however that the model was training exceptionally slowly and wouldn't be practical to scale towards larger commercial applications. Thus we decided to do an inception network since it

carries many of the same benefits as the CNN with feature detection but it's ability to use multiple filter sizes within one layer allows it to train much faster while also being able to detect features of similar complexity. Our implementation of the inception network followed closely with the tensorflow public api, with a slight modification to the final layer for a two class classification rather than softmax. Since the original model was trained with image detection we decided many of the pretrained weights in the hidden layers could be reused. However we did play around with both aspects, and trained the model using a smaller dataset but found the pretrained values were superior.

5 Experiments/Results/Discussion

DNN

For our first deep neural net implementation (Vanilla DNN) we had a learning rate of 0.0075 and we ran it for 3000 iterations. Again we used our judgement from prior experience and trial and error to determine the most effective combination of hyperparameters for loss minimization. Our other Deep Net implementation through Tensorflow (TF DNN) used a learning rate of 0.0001, 1500 epochs, and a mini-batch size of 32 (Figure 2). However we decided against pursuing this further as the DNN model lacked the complexity necessary for deepfake detection and had a tendency to overfit. Moreover it's training time was very high as DNN tend to have many parameters with their fully connected layers.



(a) Vanilla DNN loss
Vanilla DNN Accuracy: `0.803`

(b) TF DNN loss
TF DNN Accuracy: `.974`

Figure 2: Performance of Previous Models

CNN

We trained the CNN model for 100 epochs with a mini-batch size of 68. We chose 100 epochs as from testing on smaller sets we saw that loss did not change much more after 100 epochs. Our mini batch size was found through trial and error. We saw that the loss decreased at a reasonable rate when the learning rate set at 0.009. We had an accuracy of 0.98 on the training set and 0.95 on the test set (Figure 3). Thus we think the CNN was doing a good job at feature detection however, the runtime was still very slow especially as we scaled the dataset.

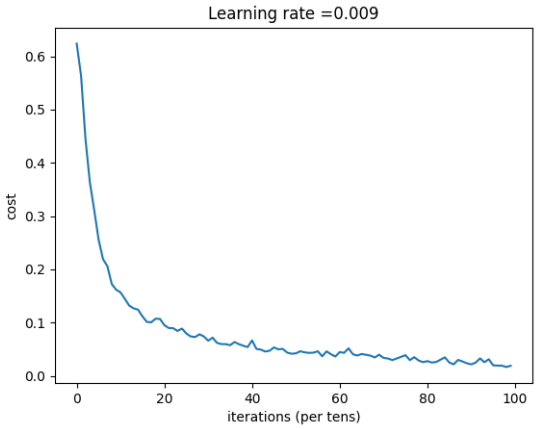


Figure 3: CNN loss

Inception-v3

We trained the Inception-v3 model for 20 epochs with a mini-batch size of 256. We chose our mini-batch size to be 256 because this was the maximum memory capacity of the GPU of our Amazon Web Services Deep Learning instance. We chose 20 epochs because we saw that loss did not change much more after 20 epochs. We saw that the loss decreased at a reasonable rate when the learning rate set at 1.0×10^{-4}. We had an accuracy of 0.99 on the training set and 0.98 on the test set. The results for the Inception-v3 are a significant improvement from the CNN in two ways: first, the accuracy increased for both the training and test sets, secondly, and most importantly, this model trained in about 10 minutes versus the CNN's training time of about three hours (Figures 4, 5, 6). The results here are especially promising as they point to the Inception network's ability to scale towards larger datasets and perhaps commercial application. We believe it was able to train faster because Inception networks allow for multiple filter sizes in each layer as opposed to only a fixed size per layer in the CNN. Moreover, because of it's versatility as we add more layers to the network we are less likely to run into the issue of vanishing gradients or over fitting.

```
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Model)         (None, 2, 2, 2048)        21802784
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 2)                 4098
=================================================================
Total params: 21,806,882
Trainable params: 21,772,450
Non-trainable params: 34,432
```

Figure 4: Summary of the overall model architecture

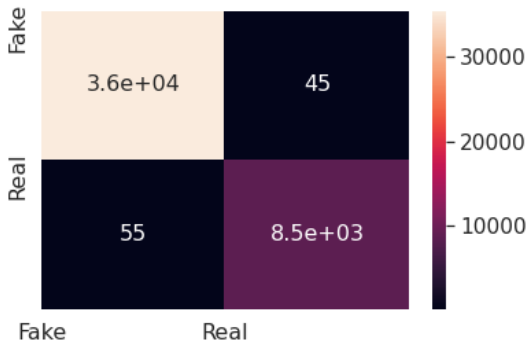


Figure 5: Confusion Matrix from model trained on the above dataset size. Left vertical axis is the actual label. Horizontal Axis is the predicted label. Right vertical axis corresponds to the number of predictions of a type given by the axes.

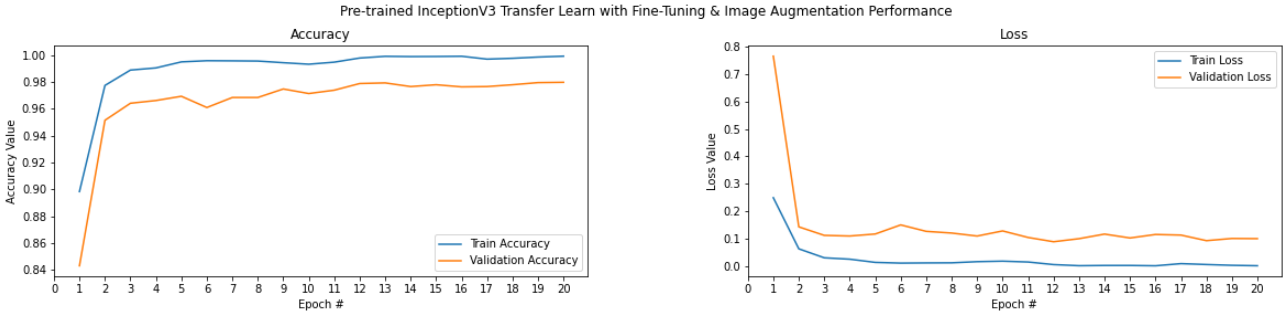


Figure 6: Train/Test Cost and Accuracy

6 Conclusion/Future Work

As our data suggest the Inception network performs well on our image datasets in regards to deepfakes, but we feel that the Kaggle dataset isn't reflective of true deepfake detection on videos. Thus in the future we will look to find more valuable ways of assessing our models on various deepfake videos. We would also like to explore a further integration with RNN strategies to study erratic movements within deepfake videos as another feature for more accurate and robust classification.

7 Contributions

The team worked together to look for valuable datasets and decide which models would perform the best. All discussions were done together on how to adjust hyper parameters. John was responsible for the DNN, Inception work, and much of the computational training was done using his machine. Andy worked on the CNN and data processing pipeline. Isaac worked with the Inception network. The team wrote the report and presentation.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] François Chollet et al. Keras. `https://keras.io`, 2015.

[3] Brian Dolhansky, Russ Howes, Ben Pflaum, Nicole Baram, and Cristian Canton Ferrer. The deepfake detection challenge (dfdc) preview dataset, 2019.

[4] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[5] Ranjana Koshy and Ausif Mahmood. Optimizing deep cnn architectures for face liveness detection. *Entropy*, 21(4):423, Apr 2019.

[6] The pandas development team. pandas-dev/pandas: Pandas, February 2020.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[9] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, CJ Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake Vand erPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1. 0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[10] Michael Waskom, Olga Botvinnik, Drew O'Kane, Paul Hobson, Saulius Lukauskas, David C Gemperline, Tom Augspurger, Yaroslav Halchenko, John B. Cole, Jordi Warmenhoven, Julian de Ruiter, Cameron Pye, Stephan Hoyer, Jake Vanderplas, Santi Villalba, Gero Kunter, Eric Quintero, Pete Bachant, Marcel Martin, Kyle Meyer, Alistair Miles, Yoav Ram, Tal Yarkoni, Mike Lee Williams, Constantine Evans, Clark Fitzgerald, Brian, Chris Fonnesbeck, Antony Lee, and Adel Qalieh. mwaskom/seaborn: v0.8.1 (september 2017), September 2017.

[11] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.