CS230

DeepFake Image Detection

Omkar Salpekar
omkars@stanford.edu

Abstract

AI-generated fake images, also known as DeepFakes, are designed to spread abusive content and misinformation amongst millions of people, exacerbated by their inherently controversial nature and the reach of modern media. In this paper, we focus on detecting DeepFake images and propose a binary classifier based on a 2-phase learning architecture. The first phase consists of a ResNet-based CNN trained as a Siamese Neural Network, designed to find distinguishing features for fake images. The second phase is a 2-layer CNN that takes the feature encodings from the first phase and outputs a REAL/FAKE classification. We build on top of prior work exploring this architecture and demonstrate 91% validation accuracy on a large, diverse dataset of sophisticated GAN-generated DeepFake images.

1 Introduction

DeepFakes are manipulated pieces of media generated to spread misinformation, hoaxes, or otherwise abusive content. With the reach of modern social media platforms, DeepFakes' inherently viral nature gives them the potential to negatively influence the opinions of millions of people, making their detection a very serious problem. Due to recent advancements in architectures like Generative Adversarial Networks (GANs) [1], DeepFake generation has become much simpler, only requiring a source image and set of intended distortions, to generate believable manipulated images. These GAN-generated DeepFakes, however, leave noticeable visual artifacts that can be analyzed using Convolutional Neural Networks (CNNs). In this paper, we explore a 2-phase learning architecture using Siamese Neural Networks [2] with CNNs for DeepFake detection. This paper constrains the problem to binary image classification, with an image as the model input and a prediction of whether the image is real or fake as the output.

2 Related work

DeepFake detection by hand is an extremely difficult task, so analytical approaches have always been far more practical. The earliest generation of work focused on non-deep learning approaches for detecting manipulated images before the rise of GANs, and included analyzing low-level features in images such as JPEG compression artifacts and chromatic aberrations [3]. Other approaches have included featurizing image data using bag-of-words and feeding those features to statistical classifiers like SVMs [4] and examining image features in the frequency domain by training classifiers on the Discrete Fourier Transforms of the images [5]. These approaches are typically too susceptible to differences in low-level image features such as lens-type and camera settings as well as specific compression formats [6].

The next generation of approaches used CNNs, as with the MesoNet by Afchar *et al.* [7], which uses dilated convolutions [8] to encode multi-scale information (due to multiple differently-sized convolutions occurring at the same layer) and learn richer contextual information. Yet other CNN's focus on

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

face-warping artifacts left by common DeepFake-generating GANs [9] and usage of high-pass filters for detecting statistical artifacts left by GANs [10]. Classical CNN-based approaches demonstrate significantly better accuracy with more sophisticated DeepFakes, yet the lack of standardization in the datasets used by each technique makes it difficult to compare their accuracies and ascertain a state-of-the-art [11]. Standardized data such as the DFDC dataset [12] and the FaceForensics++ [13] benchmark should improve this.

Lastly, the approach we build upon is the 2-phase learning architecture proposed by Hsu *et al.* [14], which recognizes the need for comparing images across classes to learn features that distinguish DeepFakes from real images. We describe this method further in Section 4.

3 Dataset and Features

The dataset used for training is derived from the DeepFake Detection Challenge (DFDC) dataset on Kaggle [12]. The original DFDC dataset contains over 470GB of mp4 videos, and based on an analysis done on a 20GB sample of the dataset, approximately 83% of the examples are deepfakes. The reason for this imbalance is that each real example has been deepfaked anywhere from 1 to 22 times, with an average of 5.19 fakes per real image, giving us a great diversity of different deepfake-generation techniques that we must detect. Each video has a frame rate of 30fps and was exactly 10 seconds long. The videos contain people of a variety of races and ages, and the backgrounds vary from bright indoors to dark outdoors.

Having constrained the problem to image classification, we sought to create a large dataset of uniformly-sized images, each with a REAL/FAKE label and roughly a 80-20 split between real images and deepfakes. More insight into this intentional imbalance can be found in Section 4. We sampled 5 frames from each video (at a frequency of 2 seconds or every 60 frames) from a 100GB slice of the original video dataset. Each image frame was resized to (224x224) pixels, normalized using the standard division by 255, and mirrored or transformed (in brightness, contrast, and saturation) at random. Further, we included 3-4 deepfakes corresponding to each real image in the dataset. These transformed raw images were fed into our ResNet-based model. An pre-processing step of performing face detection and cropping was attempted, but ultimately not incorporated into the final model. We finally used a training set of approximately 100K images with a validation set of around 5K images. Example data can be found in Figure 4 in Appendix A.

4 Methods

In this section we explain the 2-phase learning approach for DeepFake detection, first proposed by Hsu *et al* [14]. Phase 1 of training involves a ResNet18-based CNN model that is trained as a Siamese Network. This network, called the Common Fake Feature Network (CFFN), is trained with triplet loss for the first several epochs to learn feature-level distinctions between fake and real images.

$$L(A, P, N) = max(0, ||f(A) - f(P)||_2^2 - ||f(A) - f(N)||_2^2 + \alpha) \tag{1}$$

The triplet loss, as seen in Equation 1 above, involves 3 input examples: Anchor (A), Positive (P), and Negative (N). We choose all possible triplets per batch where the anchor and positive examples are from the same class, and the negative image from the other class. (Examples are chosen per batch in order to avoid precomputing $\binom{n}{3}$ triples where $n \approx 100$K.) The triplet loss is minimized when the difference between the positive and anchor encodings $||f(A) - f(P)||_2^2$ is minimized and the difference between negative and anchor encodings $||f(A) - f(N)||_2^2$ is maximized. Thus training the CFFN with gradient descent drives the model weights to produce vastly differing encodings for real and fake images, and the large number of convolutional filters will learn feature-level differences between the two classes.

In Phase 2 of training, we have a small CNN (the Classifier Network) appended to the end of the CFFN, that takes the concatenated encodings from the final convolutional layers of the CFFN and passes it through more convolutional and linear layers to output a binary class prediction. The entire network (CFFN and Classifier network) is trained with cross-entropy loss in Phase 2 to create a high-performance end-to-end classifier. Figure 1 provides a visualization of the entire network architecture including the specifics for the CFFN and Classifier networks.

Intuitively, training the CFFN as a Siamese Network for the first several epochs gives us a deep network of robust feature detectors for properties that commonly distinguish DeepFakes such as occlusions and pixelation near the face, mismatched color gradients, and abnormal shadowing. The structure also suits our dataset, which contains far more fake examples than real, allowing us to create multiple triples with different fakes for each real image. The classifier network then fine-tunes the earlier CFFN layers and trains the last two layers for several epochs. This model was implemented with TensorFlow [15].

5 Experiments/Results/Discussion

5.1 Hyperparameter/Architecture Choices

The final model uses the 2-phase training architecture described in Section 4. Preprocessed images (with the transformations described in Section 3) are resized to 224x224x3 and then passed into the CFFN network. The CFFN uses two separate Dense Residual Blocks with filter sizes of 3 and 5. As seen in Figure 2, each Dense Residual block consists of 15 separate residual units, with 64, 96, 128, and 256 channels. Each residual unit in the dense block is a standard residual unit [16] with 2 sets of BatchNorm->Swish->Conv and a skip connection. (See Equation 2 for the swish activation [17] formula). The output volumes of each dense block are concatenated, as inspired by the DenseNet architecture [18], and then reshaped before being passed to the Softmax layer. The encodings returned by Softmax are used for the triplet loss computation in the Phase-1 Siamese training.

In Phase-2, input images follow the same processing steps until the dense block outputs are concatenated, after which they are passed through the classifier network consisting of BatchNorm, Conv, and Linear layers to output a binary prediction. The architecture differs from the original CFFN model by adding 12 additional residual units: 6 more residual units in each of the 2 dense blocks (see Figure 2). These added residual units have 256 channels each, greatly increasing the number of parameters and allowing the CFFN to better learn distinguishing features between real and fake images. The skip connections in the residual blocks helps prevent these additional layers from overfitting.

$$f(x) = x \cdot sigmoid(x) = x \cdot \frac{1}{1 + e^{-x}} \tag{2}$$

Phase-1 Training (CFFN as a Siamese Network) lasted the first 5 epochs, while Phase-2 (whole network with cross-entropy loss) lasted the next 15 epochs. Both phases used the Adam Optimizer and a learning rate scheduling policy that started at $1e^{-4}$ and divided by 10 until $1e^{-6}$ based on certain thresholds such as completing 16 epochs of training and reaching a validation accuracy of 89%. Additionally, a batch size of 128, regularization λ of 0.001 for the cross-entropy loss function, and a margin α of 0.8 for the triplet loss function were further hyperparameters that were determined to produce the best performing model. Early stopping was used to prevent overfitting with a validation threshold of 94%.

5.2 Architecture Tuning Process

This section describes how we arrived at the chosen model architecture and hyperparameters. In the first implementation, we saw a training accuracy of $\sim$0.8 and a validation accuracy of $\sim$0.6 on a 10GB slice of the dataset (with around 10K images). The low training accuracy suggested our model underfit the dataset. To improve training accuracy, we experimented with several architectural changes including increasing the number of channels in the residual units, using longer residual units (such as 3-4 sets of BatchNorm->Swish->Conv instead of 2), using more residual units in each dense block (See Section 5.1 and Figure 1 for details), stacking multiple parallel convolutions per residual unit with 7x7 or 9x9 filters as with Inception Net [19], and using the activations from all previous units in each layer similar to DenseNet. Through experimentation, we determined that adding 6 more residual blocks with 256 channels to each Dense Residual Block performed best, with the nearly 24 additional convolutional filters greatly increasing the model's expressiveness.

Secondly, we tried training the model for more epochs, which is determined by the number of training epochs for each phase. Whereas the Phase-1 Siamese network was previously only trained for one epoch, we observed significant decreases in triplet loss only around 5 epochs. The length of Phase-2 training was empirically chosen to be 15 epochs.

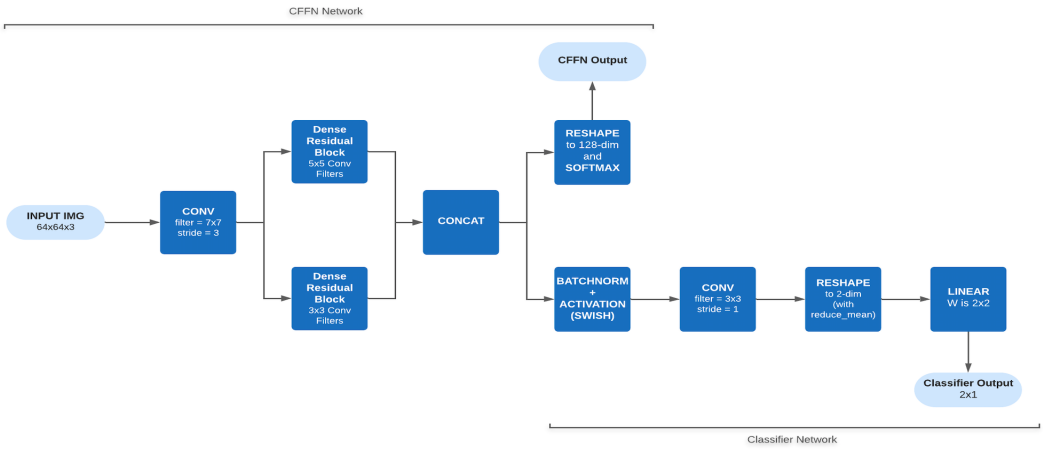


Figure 1: End-to-end model architecture. Note the labels demarcating the CFFN Network, the Classifier network, and their respective outputs.

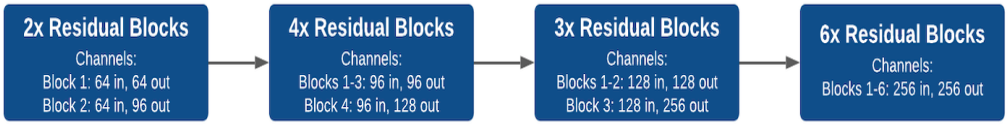


Figure 2: A more detailed look at the Dense Residual Block units referenced in Figure 1.

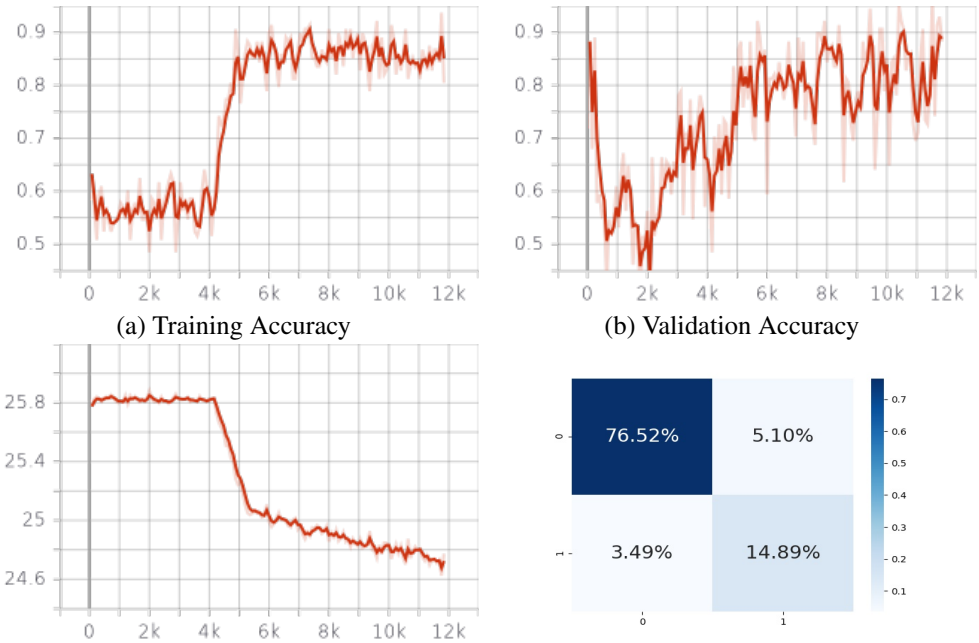
The above optimizations pushed training accuracy to nearly 0.87, but revealed another issue. After several epochs of decreasing cross-entropy loss, it began oscillating between 2 values endlessly and failed to improve classification accuracy. To address this, we implemented learning rate scheduling, and arrived at the policy described in Section 5.1. Finally, various batch sizes from 32 to 256 were tried, and an optimal of 128 was found. These optimizations led to a training accuracy of 0.91 and a validation accuracy of 0.74.

The gap between training and validation accuracies indicated overfitting. We first added regularization to the cross-entropy loss, which closed the gap significantly. We then increased the training data size, first to 20K, then 50K, and finally 100K images. Eventually, these yielded a training accuracy of 0.94 and validation accuracy of 0.91. For the larger training datasets, we observed the validation accuracy sometimes decreasing or oscillating in the latter epochs and implemented early stopping to prevent further overfitting. Additional approaches were attempted to solve the overfitting problem including dropout and varying the placement of BatchNorm layers, however these did not yield better accuracy.

My other experiments have included tweaking the activation functions (swish vs. relu), tuning the regularization constant, increasing the total number of training epochs, tuning the margin constant α in the triplet loss, changing the size of convolutional filters and stride/padding in the CFFN, and preprocessing the training data using face detection and cropping. Many of these yielded varied results and were incorporated into the final model accordingly.

5.3 Analysis

The model had a final training accuracy of 94% and a validation accuracy of 91%. It is difficult to compare this model holistically with existing work, as the original CFFN proposed in Hsu *et al.* used a vastly different self-generated dataset, although our work shows improved precision and recall. Charts depicting the loss curve during training as well as the training and validation accuracies can be seen in Figure 3. While these were the two primary metrics, we have also computed the precision, recall, and F1 score in the Table 1.



(a) Training Accuracy

(b) Validation Accuracy

(c) Loss Curve

(d) Confusion Matrix on Validation Set

Figure 3: Training Metrics. y-axis for curves is number of iterations

The most informative chart about the specifics of the model's performance can be found in the confusion matrix in Figure 3(d). The validation set had 4902 examples, 3922 fakes and 980 reals, which is the intended 80-20 split in the validation data that matches the training data distribution. Of these, the model correctly identifies the vast majority of fake examples. The false positive rate being greater than the false negative rate and the larger number of predicted fakes suggests the model is much more likely to err on the side of labelling examples fake. This is the intended behavior since the cost of a false negative (fake image marked as real) is higher than that of a false positive (real image marked as fake), as the former could allow the spread of misinformation and abusive content, whereas the latter would merely inconvenience the end user before eventually being flagged and corrected manually in some real-world setting. Ideally, we would drive the false negative rate even closer to 0. A scan through the false negatives suggests that the model is susceptible to low-quality images, dark backgrounds, and natural occlusions in the facial area due to shadows, and perhaps a more complex model and increased preprocessing could be useful.

Validation Accuracy	Training Accuracy	Precision	Recall	F1 Score
0.9141	0.9375	0.937516	0.956400	0.946864

Table 1: Key Performance metrics for a binary classification model.

6 Conclusion/Future Work

We built a high-performance DeepFake classifier using 2-phase learning and a ResNet-based model with large dense residual blocks. The CFFN was trained as a Siamese Network with an aggressive triplet loss margin, whereas the entire network was fine-tuned in latter epochs with regularized cross-entropy loss to preserve generality in a large, diverse dataset of 100K images. Optimizations such as learning-rate scheduling, early stopping, and batch size tuning improved accuracy.

Given more time, expanding on this work to multi-modal DeepFakes would be interesting. The existing model could be used within the context of an RNN for learning cross-frame relationships. Similar RNN-based approaches could be used to detect audio tampering, and the results of these two models could be synthesized for a complete end-to-end DeepFake video detection model.

7 Contributions/Code

This was a solo project. The code for the project can be found in the following repository: https://github.com/osalpekar/DeepFake-Detection

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," 2014.

[2] G. R. Koch, "Siamese neural networks for one-shot image recognition," 2015.

[3] H. Farid, "Image forgery detection," *IEEE Signal Processing Magazine*, vol. 26, no. 2, pp. 16–25, 2009.

[4] Y. Zhang, L. Zheng, and V. L. L. Thing, "Automated face swapping and its detection," *2017 IEEE 2nd International Conference on Signal and Image Processing (ICSIP)*, pp. 15–19, 2017.

[5] T. Dzanic and F. Witherden, "Fourier spectrum discrepancies in deep network generated images," 2019.

[6] L. Verdoliva, "Media forensics and deepfakes: an overview," 2020.

[7] D. Afchar, V. Nozick, J. Yamagishi, and I. Echizen, "Mesonet: a compact facial video forgery detection network," *CoRR*, vol. abs/1809.00888, 2018.

[8] W. Shi, F. Jiang, and D. Zhao, "Single image super-resolution with dilated convolution based multi-scale information learning inception module," *CoRR*, vol. abs/1707.07128, 2017.

[9] Y. Li and S. Lyu, "Exposing deepfake videos by detecting face warping artifacts," 2018.

[10] H. Mo, B. Chen, and W. Luo, "Fake faces identification via convolutional neural network," in *Proceedings of the 6th ACM Workshop on Information Hiding and Multimedia Security*, IHamp;MMSec '18, (New York, NY, USA), p. 43–47, Association for Computing Machinery, 2018.

[11] T. T. Nguyen, C. M. Nguyen, D. T. Nguyen, D. T. Nguyen, and S. Nahavandi, "Deep learning for deepfakes creation and detection," *ArXiv*, vol. abs/1909.11573, 2019.

[12] B. Dolhansky, R. Howes, B. Pflaum, N. Baram, and C. C. Ferrer, "The deepfake detection challenge (dfdc) preview dataset," 2019.

[13] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, and M. Nießner, "Faceforensics++: Learning to detect manipulated facial images," 2019.

[14] C.-C. Hsu, Y.-X. Zhuang, and C.-Y. Lee, "Deep fake image detection based on pairwise learning," 01 2020.

[15] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pp. 265–283, 2016.

[16] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015.

[17] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," 2017.

[18] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," 2016.

[19] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2014.

Appendix

A. Sample Data

Figure 4: Some Real Images (right) and their corresponding DeepFakes (left) that can be found in the DFDC dataset. These images were presented in the DFDC overview paper [12]. We refrain from using other non-public images in the paper due to the DFDC competition terms and conditions.