Real Face Detector

Mohammadamin Aliari

Abstract

We are introducing a system that can detect real human faces from fake ones. We think having an image authentication system is necessary because with recent advancements in Machine Learning the current systems are being undermined and detecting fake media is becoming increasingly challenging. Our focus is to build a model that can generalize well and is not only sensitive to a specific source of fake faces but the most prevalent ones. Our approach is to design a novel Convolutional Neural Network that can identify non-genuine aspects of human images. The system's primary metric is accuracy; our model can detect a wide range of fake generators from StyleGAN, PGGAN to Face2Face with an average of 94% accuracy on the test dataset.

1. Introduction

The main goal of the project is to detect real human faces from fake ones. Human identities can be easily altered in social networks or other more sensitive platforms. Also, one of the most important aspects of human identity is their faces. Therefore, there is a need for a system that can help confront fraudulent behaviors. Our system expects a 256x256 RGB image as input, and classifies it as real (label 0) or fake (label 1). We will mainly compare the accuracy of our model with ATMEN[3] which is the state-of-the-art binary classifier for the task with an average of 98.5% on test data. We will also use ATMEN dataset called Hybrid Fake Face (HFF)[3]. For real faces, this dataset uses CelebA[8], CelebA-HQ[7] and YouTube-Frame that have frames of human faces. For fake ones it uses StyleGAN[6], Glow, StarGAN[2], PGGAN[5], and also Face2Face[9] which swaps YouTube-Frame with fake faces to generate DeepFake videos. More details about the number of each source's images present in the dataset are mentioned in Table 1. Also, Figure 1 shows some examples of the kind of data we will work with. The final goal is to have a model that can correctly label the images with at least 90% accuracy on the test set.

	Data Type	Image Size	Count
Real Faces	CelebA	178x218	25k
	CelebA-HQ	1024x1024	10k
	YouTube-Frame	Random size	25k
Fake Faces	PGGAN	1024x1024	10k
	StyleGAN	1024x1024	10k
	Glow	256x256	25k
	Face2Face	Random size	25k
	StarGAN	256x256	25k

Table 1. The Hybrid Fake Face (HFF) dataset.

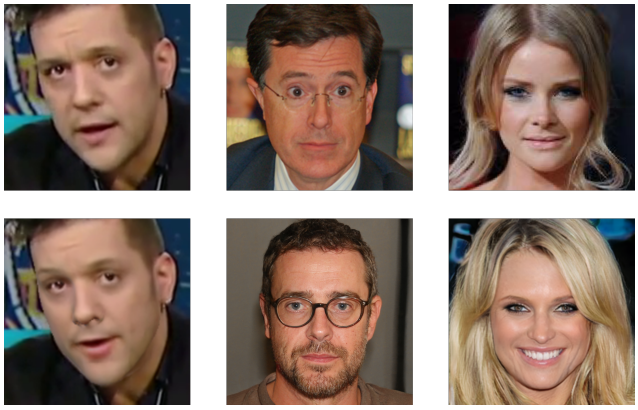

Figure 1. First Row: Real images. Second Row: Fake images.

2. Methodology

2.1. Preprocessing

Our dataset consists of images of different sizes. In the preprocessing stage, we choose 256x256 as image dimensions and resize all of the images to that size. This is in contrast to ATMEN which uses 128x128, but we have seen better results with our choice.

For the training set, we pick 48k real and 76k fake faces. Since the goal of our classifier is to generalize well to handle different famous generators, we make sure that each generator is equally represented to avoid bias. For the validation set, we use 7k real and 13k fake faces. The validation set is also balanced and as we explain later in this section, we continually check the model performance on each fake generator in the set. Having balanced data, as indicated in section 3, ensures that our model performs well on all the

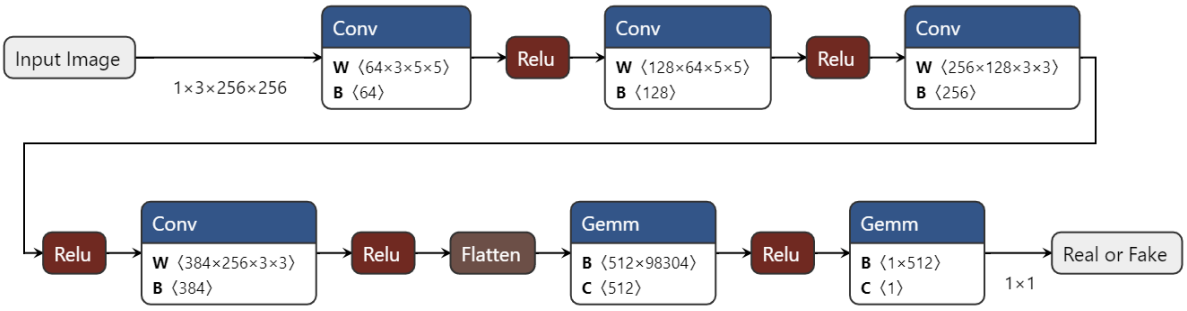

Figure 2. FaceCNN Architecture

generators present in the HFF dataset. The rest of the data is used for the testing set, so our split is about 90% for training and validation and 10% for testing. We should mention that our dataset split is almost the same as ATMAN.

2.2. Baseline Models

We first started with a linear model called logistic regression and achieved 51.8% accuracy which was far from ideal. Next, we moved to non-linear models and used multilayer perceptron which is a neural network with one hidden layer. After tweaking the batch size and other hyperparameters, we achieved 57% accuracy. The increase in accuracy showed us that non-linear models are more promising. Therefore, we chose convolutional neural networks which have been widely used for image data. Specifically, 2D convolutions can capture hierarchical and spatial info of images from high-level abstractions to low-level details.

2.3. FaceCNN

We call our novel model **FaceCNN**. It takes a tensor of size (3x256x256) as input since our images are of size 256x256 and consist of 3 channels of RGB values. We employ 4 convolutions of kernel sizes [5, 5, 3, 3], strides of [2, 2, 2, 2], and padding of [2, 2, 1, 1] respectively for our classifier. We use the mentioned strides and paddings to make sure our feature maps follow powers of two while down-sampling them. We start with 64 output features and gradually increase our output features number by multiples of 2. We increase our output features in each layer to detect finer, lower-level spatial information and patterns. We apply BatchNorm2D to ensure that our neuron outputs stay in a predictable distribution. This also helps our parameter space to stay stable and easier to optimize. At the end of each convolution, we apply Relu for non-linearity as it proved to be good at avoiding vanishing gradients. Finally, to apply softmax and classify the input image, we first linearize our final feature maps to a tensor of size 98304 by torch Flatten layer. We feed this tensor to two fully connected layers (MLP) of sizes 512 and 2 to lower the features count to a point that can comfortably map to our final 2 classes. See Figure 2 for a graphical representation of the architecture.

Despite solving a binary classification problem, we picked 2 neurons for output and used softmax on top of them with categorical cross-entropy loss function, instead of binary cross-entropy with 1 output neuron. This resulted in a much better convergence in practice.

2.4. Training

For the training, we update our model multiple times over all the batches. Furthermore, to enhance our stochastic solution, we shuffle our batches before each epoch. At the end of each epoch, we check the performance over the validation set which consists of never-seen-before data. We use vanilla cross-validation and not K-fold or other more algorithmic approaches since the dataset is huge and the training times are long. However, this was not a problem in general as we picked our hyperparameters based on the well-known values that have been widely used and then fine-tune them manually with a binary search approach. For instance, we figured that the Adam optimizer worked the best with no L2 regularization, or increasing convolutional layers did not help and caused overfitting. Also, we started with 12 base feature outputs and found noticeable improvements by increasing it to 64. We also discovered that early stopping at the 3rd epoch leads to finding the best model which has the lowest validation loss. We save the model at that point and avoid overfitting. Our model-specific tweaks and tunings mentioned here improved the performance from 78% to beyond 90%.

In summary, we found the best convergence with the following hyperparameters: Batch Size of 200, Learning Rate of 1e-4, Adam Optimizer with L2 regularization of 0, Relu Activation, He Initializer, and 2D Batch Normalization with Momentum of 0.1.

Finally, we use GPU for faster training when available and use the smallest scope for on-device variables to minimize memory usage. The result is a predictable memory footprint which makes our hardware requirements predictable. The training process requires 8GB of VRAM.

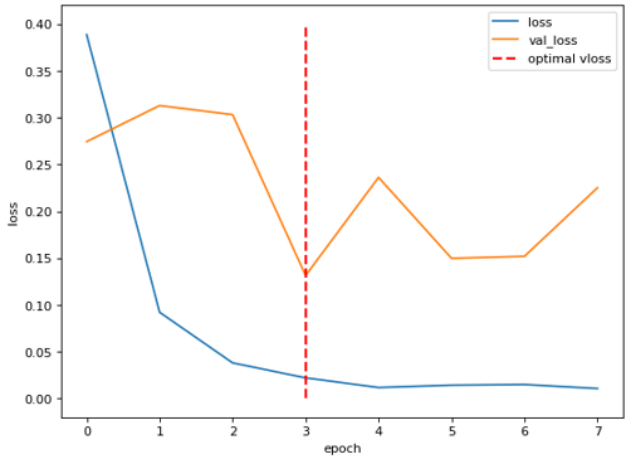

Figure 3. CCE Loss Function History

2.5. Originality

We are using a novel design for our model that can handle real and fake faces tasks on both images and face-swapped video frames similar to deep fake videos. This is in contrast to most of the efforts in academics that had focused only on DeepFake videos or synthetic images. Furthermore, during the training process, we also checked if our model has a bias based on the image source. Finally, in comparison to ATMEN which has multiple preprocessing networks to accentuate modification traces, we only have one network with a significantly smaller size and lower learnable parameters count. Therefore, in practice, it would also require less memory in run-time and can be seen as a lightweight solution in web environments. For example, it could be used as a fast online API to verify the image the user is uploading to a social network platform.

3. Experimental Results

Here we discuss the main experimental results, critically discuss them, and compare them with results available in the literature.

3.1. Metrics

Training and Validation loss history can be observed in Figure 3. We use categorical cross-entropy and minimize the loss for training and validation sets to 0.01 and 0.13 respectively. Therefore, our model does not experience underfitting or overfitting. Our main metric for the performance of the model is accuracy. As indicated in Figure 5, we achieve an average of 94% accuracy on test data. Our model performs well on all different sources of images available in the training set and it does not suffer from bias.

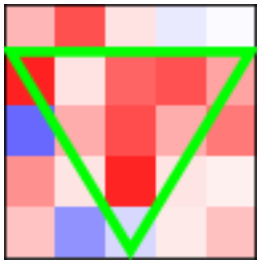

Figure 4. FaceCNN Filter Example.

3.2. Learned Filters

Reverse engineering deep neural networks is a challenging task as they look like black boxes. However, in Figure 4 we can see one of the first layer's weights which has a triangle pattern where eyes and mouth along with other facial features have a higher value (hotter colors). Therefore, our model appears to learn the structure of the face and distinguish real and fake aspects of the image by its trained kernels.

3.3. Comparison to ATMEN

ATMEN as introduced earlier is the state-of-the-art model for HFF dataset. It has 98.5% accuracy. In Table 2, we can see both FaceCNN and ATMEN performance on test data. Our model has an accuracy of 94% which is only 4.5% less than ATMEN's.

3.4. Performance on Hand-Edited Images

As an experience, we wanted to see how well our model can generalize on out-of-distribution images[1]; meaning, photoshopped or hand-edited images that are only partially fake. Our model performance drops to 55% which shows that it cannot classify these types of images reliably. To complete our experience, we wanted to test the same out-of-distribution dataset on ATMEN. However, first, we needed to spend a while reviving ATMEN's code which was written in Caffe[4] and used old libraries. After that, as shown in Table 2, ATMEN's accuracy drastically drops and is only 64.5%. Keep in mind that these kind of images are usually easily distinguishable by human eyes, but still, it shows that there is more work to be done even for state-of-the-art models.

Model	Dataset	Accuracy
FaceCNN	Test Dataset	93.8%
	Photoshop	54.8%
ATMEN	Test Dataset	98.5%
	Photoshop	64.5%

Table 2. Models Performance

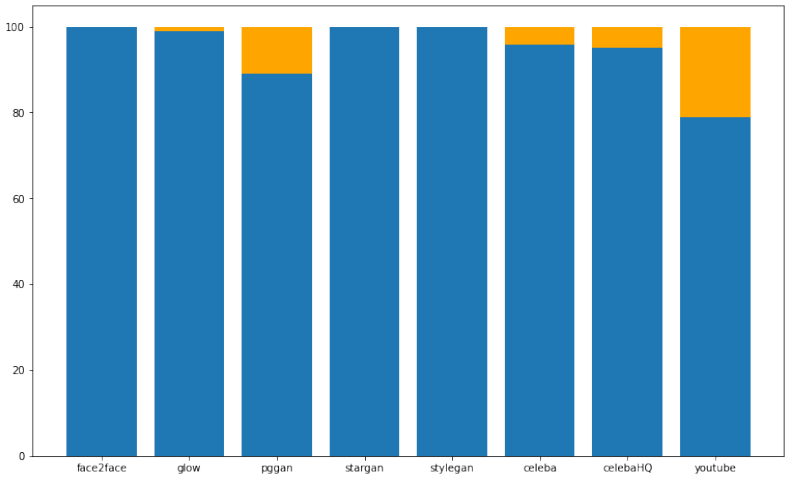

Figure 5. FaceCNN Accuracy on Different Image Sources.

4. Conclusions

4.1. Goals and Experiences

We think we have achieved the main goal of the project as our model has above 90% accuracy on test data. Another goal was to have no bias and perform well on all real and fake images in the dataset. This goal was also reached according to 5. A secondary goal of the project was to see if the model can handle out-of-distribution images. The accuracy was about 55% on manually-edited photos. Therefore, while we did not completely fail and fall under 50%, the model cannot be seen as a solid answer to this kind of problem. We have also seen that even state-of-the-art models like ATMEN could not handle these images and their performance dropped significantly.

4.2. Future Work

For future work, we would like to train our model on images that have been manually edited with tools like Photoshop, so that it can understand partially fake faces as well. AutoEncoders are another valid avenue to investigate. This is because they are strong at compressing the image and extracting the most important features of them. Therefore, if we employ an AE before feeding the image to FaceCNN, we might see improvements in the model's general performance.

References

[1] Fake face photos by photoshop experts - ciplab @ yonsei university. https://www.kaggle.com/datasets/ciplab/real-and-fake-face-detection/. Accessed: 2022-03-20. 3

[2] Yunjey Choi, Min-Je Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. *CoRR*, abs/1711.09020, 2017. 1

[3] Zhiqing Guo, Gaobo Yang, Jiyou Chen, and Xingming Sun. Fake face detection via adaptive manipulation traces extraction network. *Computer Vision and Image Understanding*, 204:103170, 2021. 1

[4] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014. 3

[5] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017. 1

[6] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. *CoRR*, abs/1812.04948, 2018. 1

[7] Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. Maskgan: Towards diverse and interactive facial image manipulation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020. 1

[8] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. 1

[9] Justus Thies, Michael Zollhöfer, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Face2face: Real-time face capture and reenactment of RGB videos. *CoRR*, abs/2007.14808, 2020. 1