Deep Fake Face Detection

Rahul Torlapati
rahult1@umbc.edu
University of Maryland Baltimore County

***Abstract*—"Deep Fakes" is a popular image synthesis technique based on artificial intelligence. The goal of "Deep Fakes" is to capture common characteristics from a collection of existed images and to figure out a way of enduing other images with those characteristics, e.g. shapes and styles. Generative adversarial networks (GANs) is the one of most frequently used ways to implement a "Deep Fake".**

I. Introduction

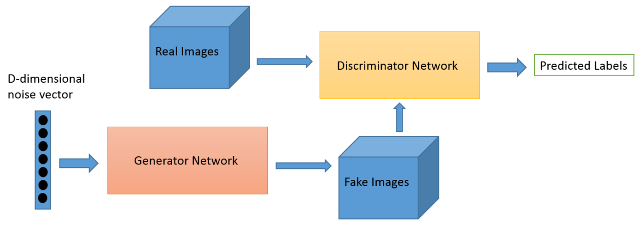
The 21st century's answer to Photoshopping, **Deepfakes (**a portmanteau of "deep learning" and "fake") use a form of artificial intelligence called deep learning to make images of fake events, hence the name **Deepfake**. They are fake videos, images or audio recordings that look and sound like the real thing. There's a lot of confusion around the term "deepfake," though, and computer vision and graphics researchers are united in their hatred of the word. It has become a catchall to describe everything from state-of-the-art videos generated by Artificial Intelligence to any image that seems potentially fraudulent.

To make a deepfake video of someone, first train a neural network on many hours of real video footage of the person to give it a realistic "understanding" of what he or she looks like from many angles and under different lighting. Then combine the trained network with computer graphics techniques to superimpose a copy of the person onto a different actor. While this a technique, the most used and easiest technique of all must be the usage of Generative Adversarial Networks (GANs). Generative adversarial networks (GANs) are algorithmic architectures that use two neural networks, pitting one against the other (thus the "adversarial") in order to generate new, synthetic instances of data that can pass for real data. GANs were introduced in a paper by Ian Goodfellow and other researchers at the University of Montreal in 2014[1].

One neural network, called the *generator*, generates new data instances, while the other, the *discriminator*, evaluates them for authenticity; i.e. the discriminator decides whether each instance of data that it reviews belongs to the actual training dataset or not. In the case of images, generator generates new images from existing images and the discriminator, evaluate the authenticity of these images and decided whether they belong to the original training data.

So far, Deepfakes have been limited to amateur hobbyists making politicians say funny things. However, it would be just as easy to create a deepfake of which potentially damage people's lives and reputations.

Hence, I aim to tackle this issue by building neural networks which can be able to detect between real faces and GAN generated. I have decided to use two datasets. CelebA dataset has 202,599 images of different celebrities and Million fake faces by Nvidia Research Lab. Due to limiting computational resources I have decided to only use 10,000 images from each dataset.



II. Tools Used

We used Python3 as the base coding language and Jupyter Notebook as the editor and environment. In Python3 we used libraries like TensorFlow, Keras, Metrics, Pandas, scikit-learn, Matplotlib etc..

III. Related work and Literature review

Previous research in detecting fake images relied heavily on handcrafted features to analyze tampered regions which were inefficient and time-consuming. Currently a large number of approaches are being designed to detect fake images. GoogLeNet (Inception v1) is a pretrained convolutional neural network that is 22 layers deep is one of such approaches. Although GoogLeNet has 22 layers it consists of 12x less parameters [2]. GoogLeNet was a significant improvement in image classification when compared to ZFNet and AlexNet which were considered previous benchmark. Another such approach is MANFA which is a hybrid framework which that uses Adaptive Boosting (Adaboost) and eXtreme Gradient Boost (XGBoost) [3]. SwapMe, an iOS application and an open source application called FaceSwap utilize this technique.

Pixel-level analysis is another such approach which tackles the problem of failing to capture high quality features, which generally lead to sub-optimal solutions. Pixel-level analysis is a pixel-level segmentation task which evaluates multiple architectures on both segmentation and classification tasks [5]. Another approach is to train GoogLeNet to detect tampering artifacts in a face classification stream and train a patch-based triplet network to leverage features capturing local noise residuals and camera characteristics as a second stream [4].

IV. Preprocessing

CelebA dataset has 202,599 images and Fake Faces dataset has about a Million image of which 10,000 will be used in order to maintain a balanced dataset. The images from CelebA dataset and Fake Faces dataset are in different dimensions. Hence the images from Fake Faces dataset are resized into images of size 224*224.

A. ImageData Generator Class

This class generates batches of tensor image data with real-time data augmentation. The data will be looped over in batches.

```
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)
train_gen = datagen.flow_from_dataframe(
    train_df,
    target_size=|(224, 224),
    batch_size=64,
)
```

V. Network Layers Used

A. Conv2D

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If `use_bias` is True, a bias vector is created and added to the outputs. Finally, if `activation` is not `None`, it is applied to the outputs as well.

B. Flatten

Flattens a tensor and reshaped into 1-D array

C. MaxPooling2D

Max pooling is a sample-based discretization process. The objective is to down-sample an input representation (image, hidden-layer output matrix, etc.), reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. Max pooling operation for spatial data.

D. GlobalAveragePooling2D

Global average pooling operation for spatial data. Output shape is 2D tensor with shape `(batch_size, channels)`.

E. Dense

Dense implements the operation: `output = activation(dot(input, kernel) + bias)` where `activation` is the element-wise activation function passed as the `activation` argument, `kernel` is a weights matrix created by the layer, and `bias` is a bias vector created by the layer (only applicable if `use_bias` is `True`).

F. Dropout

Applies Dropout layer to the input. Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

G. BatchNormalization

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

VI. Models

The model type that we will be using is Sequential. Sequential is the easiest way to build a model in Keras. It allows you to build a model layer by layer. We use the `add()` function to add layers to our model.

A. Model – I

First two layers are Convo2D layers, these will deal with the input images which are 2 dimensional matrices. Kernel size is the size of the filter size for our convolution. The activation function used for the first 2 layers is Rectified Linear Activation function. This function has been proven to work well with neural networks. Flatten layers serves as a connection between convolution and dense layers. Dense layer is the standard layer used in many types of neural networks. The Dense layer has 2 nodes because the goal of the neural network is the distinguish if an image is real or GAN generated. The activation is 'SoftMax'. SoftMax makes the output sum up to 1 so the output can be interpreted as probabilities.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 222, 222, 64)      1792
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 220, 220, 32)      18464
_________________________________________________________________
flatten_1 (Flatten)          (None, 1548800)           0
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 3097602
=================================================================
Total params: 3,117,858
Trainable params: 3,117,858
Non-trainable params: 0
_________________________________________________________________
```

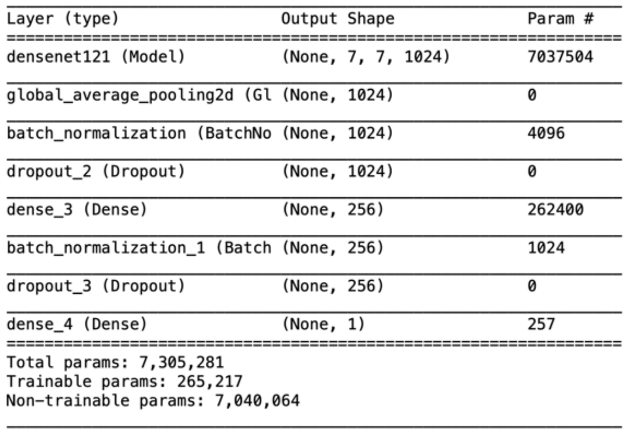
B. Model – II

First two layers are Convo2D layers, these will deal with the input images which are 2 dimensional matrices. Kernel size is the size of the filter size for our convolution. The activation function used for the first 2 layers is Rectified Linear Activation function. This function has been proven to work well with neural networks. Maxpooling operation for spatial data of the image and is followed by a dropout layer. The output is flattened and passed onto a dense layer. The last layer is a Dense layer with activation as 'SoftMax'. SoftMax makes the output sum up to 1 so the output can be interpreted as probabilities.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 222, 222, 32)      896
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 220, 220, 64)      18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 110, 110, 64)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 110, 110, 64)      0
_________________________________________________________________
flatten_2 (Flatten)          (None, 774400)            0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               99123328
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 99,142,978
Trainable params: 99,142,978
Non-trainable params: 0
_________________________________________________________________
```

C. Model – III

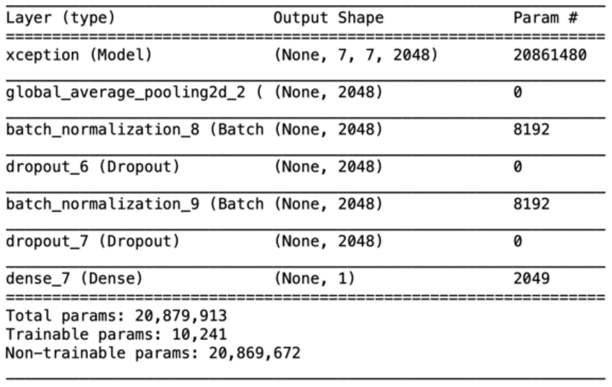
DenseNet is a network architecture where each layer is directly connected to every other layer in a feed-forward fashion (within each *dense block*). For each layer, the feature maps of all preceding layers are treated as separate inputs whereas its own feature maps are passed on as inputs to all subsequent layers. In order to use DenseNet121, the pretrained network and its ‘imagenet’ weights and trainable layers are set to false, for ‘trainable_weights’ property to resolve to an empty list. Since the base convolution network already contains features that are generically useful for classifying pictures, we can un-freeze the top layers of the frozen model base and jointly train both the newly added classifier layers and the last layers of the base model. In addition to the base layer a couple of dense, dropout and normalization layers are added. The resultant neural network has 7,305,281 parameters.



```
Layer (type)                 Output Shape              Param #
=================================================================
densenet121 (Model)          (None, 7, 7, 1024)        7037504
_________________________________________________________________
global_average_pooling2d (Gl (None, 1024)              0
_________________________________________________________________
batch_normalization (BatchNo (None, 1024)              4096
_________________________________________________________________
dropout_2 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_3 (Dense)              (None, 256)               262400
_________________________________________________________________
batch_normalization_1 (Batch (None, 256)               1024
_________________________________________________________________
dropout_3 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 257
=================================================================
Total params: 7,305,281
Trainable params: 265,217
Non-trainable params: 7,040,064
_________________________________________________________________
```

D. Model – IV

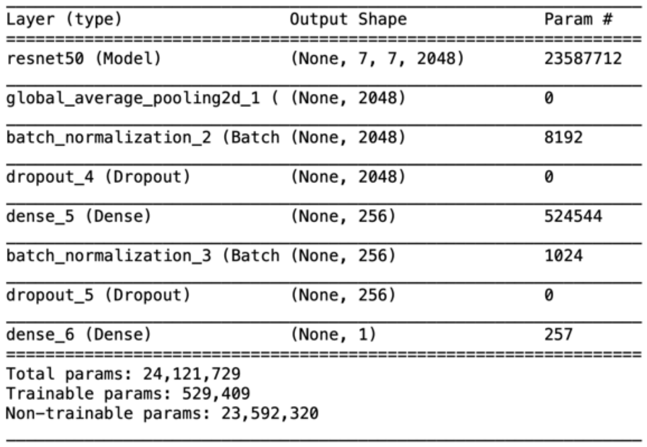
Xception is an extension of the Inception architecture which replaces the standard Inception modules with depth wise separable convolutions. In order to use Xception, the pretrained network and its ‘imagenet’ weights and trainable layers are set to false, for ‘trainable_weights’ property to resolve to an empty list. Since the base convolution network already contains features that are generically useful for classifying pictures, we can un-freeze the top layers of the frozen model base and jointly train both the newly added classifier layers and the last layers of the base model. In addition to the base layer a couple of dense, dropout and normalization layers are added. The resultant neural network has 20,879,913 parameters.



```
Layer (type)                 Output Shape              Param #
=================================================================
xception (Model)             (None, 7, 7, 2048)        20861480
_________________________________________________________________
global_average_pooling2d_2 ( (None, 2048)              0
_________________________________________________________________
batch_normalization_8 (Batch (None, 2048)              8192
_________________________________________________________________
dropout_6 (Dropout)          (None, 2048)              0
_________________________________________________________________
batch_normalization_9 (Batch (None, 2048)              8192
_________________________________________________________________
dropout_7 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 2049
=================================================================
Total params: 20,879,913
Trainable params: 10,241
Non-trainable params: 20,869,672
_________________________________________________________________
```

E. Model – V

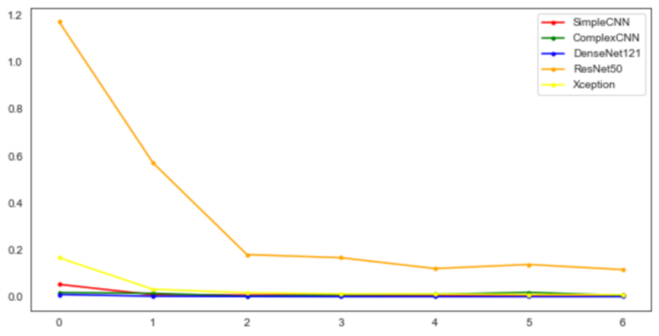
ResNet architecture has a fundamental building block (Identity) where you merge (additive) a previous layer into a future layer. Reasoning here is by adding additive merges we are forcing the network to learn residuals (errors i.e. differences between some previous layer and current one). In order to use ResNet50, the pretrained network and its ‘imagenet’ weights and trainable layers are set to false, for ‘trainable_weights’ property to resolve to an empty list. Since the base convolution network already contains features that are generically useful for classifying pictures, we can un-freeze the top layers of the frozen model base and jointly train both the newly added classifier layers and the last layers of the base model. In addition to the base layer a couple of dense, dropout and normalization layers are added. The resultant neural network has 24,121,729 parameters.



```
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Model)             (None, 7, 7, 2048)        23587712
_________________________________________________________________
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
batch_normalization_2 (Batch (None, 2048)              8192
_________________________________________________________________
dropout_4 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_5 (Dense)              (None, 256)               524544
_________________________________________________________________
batch_normalization_3 (Batch (None, 256)               1024
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 257
=================================================================
Total params: 24,121,729
Trainable params: 529,409
Non-trainable params: 23,592,320
_________________________________________________________________
```

VII. Results

Validation loss is a good metric as in some sense it is some measure of how many of the predictions differ from the original before they’re put through the threshold value. Each of the models run for 7 epochs each and their validation losses are compared. Although ResNet50 has the greatest number of parameters, this model performs significantly worse than all other models. Of all the models DenseNet121 performs the best as it has the lowest validation loss when compared to all other models. Approximately each epoch for all models took 600sec to run, with Xception being the fastest model to run. Together all the models collectively took over 9 hours to run. Keeping in mind the limited computational resources available only 10,000 instances of CelebA and Fake faces are considered. With additional resources, this project can be performed better.



For future work, better quality images can be used, and more complex neural networks and better performing can be built. Frame by frame images of a video can be fed into a better performing neural network to assess if that particular video is real or machine generated.

Acknowledgment

Foremost, I would like to express my sincere gratitude to my advisor Dr. Ergun Simsek for the continuous support of my Masters' study and work, for his patience, motivation, enthusiasm, and immense knowledge. His guidance and feedback helped me in all the time of my work at UMBC and on this project. I could not have imagined having a better advisor.

References

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Yoshua Bengio, (2014) "Generative Adversarial Networks"

[2] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2017) "Going Deeper with Convolutions."

[3] L. Minh Dang, Syed Ibrahim Hassan, Suhyeon Im and Hyeonjoon Moon (2019) "Face image manipulation detection based on a convolutional neural network."

[4] Zhou, P., Han, X., Morariu, V.I., & Davis, L.S. (2017). "Two-Stream Neural Networks for Tampered Face Detection." *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 1831-1839.

[5] Jia Li, Tong Shen, Wei Zhang, Hui Ren, Dan Zeng, Tao Mei (2019) "Zooming into Face Forensics: A Pixel-level Analysis."

[6] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger "Densely Connected Convolution Networks" 2016

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun (2015) "Deep Residual Learning for Image Recognition."

[8] Liu, Ziwei and Luo, Ping and Wang, Xiaogang and Tang, Xiaoou (2015) "Deep Learning Face Attributes in the Wild."

[9] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, Timo Aila (2019) "Analyzing and Improving the Image Quality of StyleGAN."

[10] François Chollet (2016) "Xception: Deep Learning with Depthwise Separable Convolutions."