Classification of Artificially Generated Faces Using Transfer Learning

I recommend reading this write-up while reading the Jupyter Notebook. While the pdf provided contains 24 pages, most of it is not actual code, but rather graphs so I hope it will not be too hard to evaluate.

I started off the project having high hopes and wanting to do something 'cool' with the knowledge gained from class. There were a lot of datasets available on Kaggle, but a lot of the data fell within the domain of not being able to understand the labels of the dataset or being too difficult for me to do. Eventually I stumbled upon a dataset titled 'real and fake face detection'. This was a dataset containing 2041 images of real and fake faces that were artificially generated. The caveat however was that the fakes were obvious fakes, and to a human observer, it would be very simple to classify between which faces were real or fake. Having also spent a couple days learning about pre-trained convolutional neural networks, I wanted to use some well-known models instead of training my own, because transfer learning sounded interesting. Essentially, you use a pre-trained model and train that model with your new data. So, I found a model named VGG19, which boasts of 19 deep layers with weights trained from the ImageNet database. Having then trained my model, when I evaluated my results on the ~2,000 images, I was only getting about ~60% validation accuracy at most with 5-10 epochs. This value would not get better despite the introduction of MaxPooling2D, batch normalization, tuning the Dropout layer, using different activation functions, and different loss functions. Ultimately, I realized the downfall of neural networks — its need for large amounts of data. I then looked towards another similar dataset off Kaggle.

The dataset I found was the current dataset I trained my model with. It contains 140k facial images, half of which are generated from a generative adversial network (GAN). This

meant 70k real faces, 70k fake, so no class imbalance. To explain GANs would be out of the scope of the report, and instead I think it is simple enough to understand GANs as two neural networks that play a game of cat-and-mouse with one neural network trying to create a very good fake images based on random inputs, and the other neural network trying to distinguish if the inputted image is real. If you look at the dataset of the 140k images, you can see that the faces are very similar, and that one could not simply just see if a face were real or artificially generated, raising the concern of someone constructing a false identity with these images.

So, the initial framework I had in mind was to load the data and convert each image into a NumPy array of the shape (224,224,3), freeze some of the layers of the VGG19 model (only train certain layer weights), evaluate the performance. Then I realized that seemed a bit too simple, so I introduced some of my own models and introduced another well known model: MobileNetV3, which boasts of very lightweight CNN model.

Then began the coding part. Having worked with Keras for a little bit now, I realized that there was a generator function available named ImageDataGenerator() which allows me to streamline the code and allowed importing of the folders containing the different images to automatically be classified and automatically be placed into a train/valid/test dataset. While this was very convenient, it later led on to issues with my model training as I realized I had to set the test dataset to ‘shuffle = False’, otherwise the ordering of the class changes. I spent a couple days looking into this.

I had results from a very simple CNN model (simple_model), a slightly more complex CNN model (simple_model1), and then finally the MobileNetV3 and VGG19. The following architectures can be seen in the code. I spent 10 epochs on each of the models, however I quickly switched it to 3-5 epochs at most, due to the time spent on the VGG19 model.

From the simple model, I was achieving a validation accuracy of 74.7% and no signs of any significant overfitting as my loss function was continually decreasing and the validation accuracy was still increasing. These three epochs also took about 7.5 minutes to train. Based on the confusion matrix, it seemed that my model was classifying more images as real rather than fake as the false positive count was also very high.

From the slightly more complex model which contained a Conv2D and MaxPooling2D layer, I was getting lower validation accuracy at 66.6%, as well as equal high levels of false positive and false negative classifications. This leads me to think there is something wrong with the model, or that it's simply underfitting, and more training time would serve to benefit the model, as the validation accuracy was still increasing. This model also took 7 minutes to train.

Next was the MobileNetV3 pre-trained model. I froze all the layers meaning the weights would not change, and I added on a 'softmax' layer for classification. The initial results were very good as I started at 92% validation accuracy and having spent 5 epochs, I achieved a validation accuracy of 96.5%. The F1 score showed that the model had high levels of accuracy and precision with identifying both classes at 0.96 and 0.97 for the fake and real class respectively. Based on the confusion matrix, it seemed like I was getting a magnitude higher false positives when compared to false negatives. Again, this seems to imply that my model was classifying more fake images as real. Looking at the training times, each epoch also took about similar times with the previous models. Observing the matplotlib plots, it seemed that further training times would improve my model as the validation accuracy was still increasing, indicating that overfitting was not of concern.

Finally, looking at the VGG16 model, I only trained for 3 epochs as each epoch took about 13 minutes. This was nearly double the time to train an entire MobileNet model. This

however did result in a validation accuracy of 98.8% however lower F1 scores, as well as looking at the confusion matrix, led to higher misclassifications of both false positives and false negatives. Near the bottom I have a combined plot of all the models, showcasing that VGG16 has better accuracy, but that it takes very long time to train the model, and that MobileNetV3 achieves almost similar levels of accuracy and is very fast to train. Finally, showing the superiority of these pre-trained models as my initial trial models had relatively lower levels of validation accuracy. I then tested the MobileNetV3 predictive powers at the bottom of the Notebook, by extracting known real and fake images from the test dataset (which was not trained on) and observing the classifications, which all came back as correctly classified.

I did not include in the notebook the first dataset I explored, as I was running low on space, however I tried testing my model on the initial dataset of obvious fake faces. The results were astonishing as my MobileNet model failed to classify any of the fake images. This means that my models fail to generalize and that more work should be done and that some more obvious fakes should be incorporated into the training datasets to account for more obvious fakes.

In conclusion, transfer learning is a very viable approach that allows for faster and highly accurate models. MobileNetV3 achieves high levels of accuracy while also gets trained very fast, allowing for fast prototyping and scalability. While the GAN generated nearly indistinguishable fake faces that were easily classified by the CNN models, it seems that with more obvious fake images, the model suffers. This indicates future work is required and that a broader dataset should be used instead.