Detecting GAN Generated Images Using Convolutional Neural Networks

Amil Khan

Deep Learning, UCSB ECE Department

Abstract

Synthetic image generation using *Generative Adversarial Network (GAN)* architectures have become increasingly harder to fail the eye test. With a relatively low-cost GPU and enough time, we have seen images of fake celebrities, bedrooms, and landscapes—to name a few—deceive a reasonable person. Previous use cases of GANs, such as increasing the number of samples in a small dataset, have seen widespread adoption across disciplines. However, over the past year, we have also seen GANs being used for malicious cases as well. Hence, we will feed GAN generated images we produced to a model whose task is to determine whether an image is "Real" or "Fake". We demonstrate GAN generated image detection using five ImageNet classification models for the classification task: classification of real images and fake images presented as inputs to the ImageNet model.

Keywords: Deep Learning, GANs, Image Classification

1. Introduction

The motivation behind this work spawns from the danger of almost everyone having access to large amounts of data. Whether the data is from social media platforms or simply Kaggle, nearly anyone is freely available to generate synthetic images using a GAN architecture. In more concrete terms, someone can claim they bolstered their neuroscience research findings by obtaining 500 brand new fMRI images from patients when, in actuality, s/he created them using GANs. Assuming the images have passed the eye test from numerous highly-skilled experts in the field, there is virtually no other safeguard for this malevolence across all disciplines today.

2. Methods

2.1. Data Generation and Preprocessing

For the data generation portion, we decided to use the Progressive GANs implementation since it produced the best results at the time of this writing. More specifically, however, the Progressive GANs approach of starting with low-resolution images, and then progressively increasing the resolution

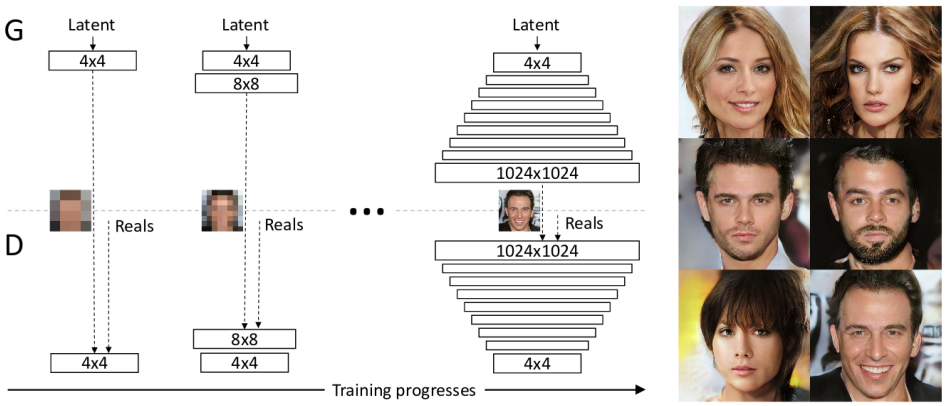

Figure 1: **Progressive GAN.** Training starts with both the generator (G) and discriminator (D) having a low spatial resolution of 4×4 pixels. As the training advances, we incrementally add layers to G and D, thus increasing the spatial resolution of the generated images. All existing layers remain trainable throughout the process. Karras et al. [1]

by adding layers to the networks, allowed greater stability during training, shorter training times, and higher resolution images. Modifying the DCGANs implementation to output larger image sizes made the network unstable–at least in our efforts.

To generate the images, we used the `Celeb-A` and `LSUN` datasets, respectively. For each class, we created approximately one to two thousand images. In total, we had 18105 GAN generated images in our training set, in which we used 30 percent (randomly sampled) of the images as our validation set. Similarly, we had 18109 real images, and 30 percent (randomly sampled) used as validation. For our test data, we included GAN generated image classes that the network has never encountered before, as well as images taken from an iPhone. In total, we have 2585 real images and 2600 fake images.

2.2. Network Architectures

After generating the images and building the datasets, we decided to hand pick five networks that we believed would perform well on our binary classification task: whether the images were GAN generated. We chose these specific models based more on their architecture than their results on ImageNet.

`ResNet`. One of the main reasons we chose ResNet over VGG19 as a somewhat baseline was because of model complexity in terms of floating point operations per second (FLOPs).

- ResNet: 11.3 billion FLOPs
- VGG19: 19.6 billion FLOPs

Additionally with ResNet, we felt that a deeper model utilizing residual functions would be able to capture a larger amount of variance within the data. For each residual function $\mathcal{F}$, we used a stack of three layers. The three layers

are $1 \times 1, 3 \times 3$, and 1×1 convolutions, where the 1×1 layers are responsible for reducing and then increasing (restoring) dimensions, leaving the 3×3 layer a bottleneck with smaller input/output dimensions [2].

DenseNet. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—DenseNet has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. Why is this advantageous? Shorter connections between layers close to the input and close to the output means we stick with the theme of deeper networks and less vanishing gradients.

Xception. Based entirely on depthwise separable convolution layers. We had to test if the "extreme" hypothesis made a difference for GAN generated images. The hypothesis is that the mapping of cross-channel correlations and spatial correlations in the feature maps of convolutional neural networks can be entirely decoupled [3].

NASNet. Google used 500 GPUs across 4 days resulting in 2000 GPU-hours during their neural architecture search [4]. Google claims that their self built architecture has 9 billion fewer FLOPS than the current state of the art human built networks.

InceptionResNetV2. Combining the best of both worlds—Inception architecture with residual connections. Why not InceptionV4? We tested both and observed fairly similar performance.

3. Results

Altogether, we spent about two weeks training and tuning each model. Here are a few parameters that were the same across all models:

- Cross Entropy Loss measures the performance of a classification model whose output is a probability value between 0 and 1. Cross Entropy Loss is robust to class imbalance, which we had.
- Stochastic Gradient Descent as our optimizer. Everything above our last layer, we defined the Learning Rate to be 0.001, Momentum to be 0.9, and L_2 Regularization to be 0.001. For our last layer, we defined the Learning Rate to be 0.00005, Momentum to be 0.9, and L_2 Regularization to be 0.00001.
- Early Stopping if validation loss does not decrease after n epochs.
- Random Horizontal Flipping

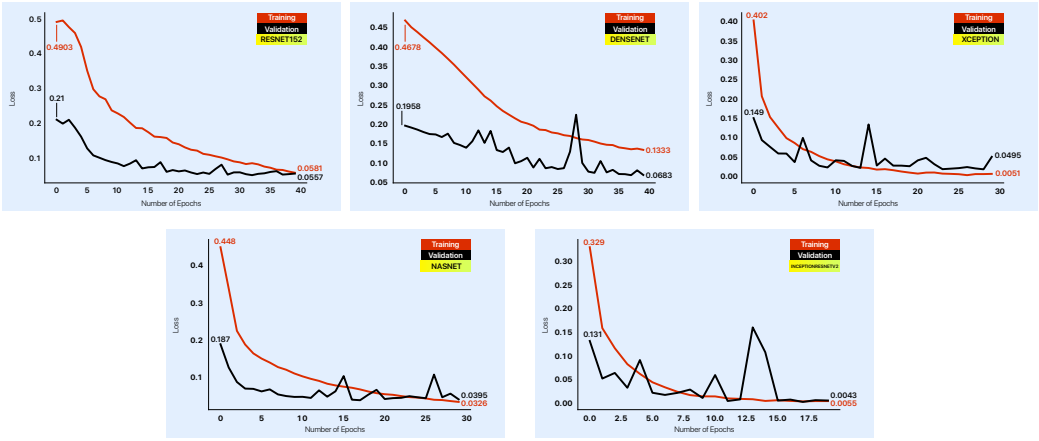

Figure 2: **Model Performance.** One constant feature across all the graphs is an abrupt spike. This could be due to random sampling, but in a future work we can explore the before and after effect of the spike. All models were also trained from scratch.

3.1. ResNet

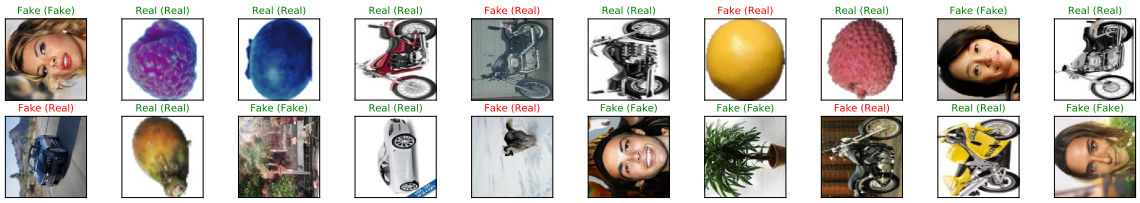

Figure 3: ResNet test set miss-classification images

We were able to achieve 77% accuracy on our test set, but this was after epoch 37. Other models were able to perform better with less epochs.

3.2. DenseNet

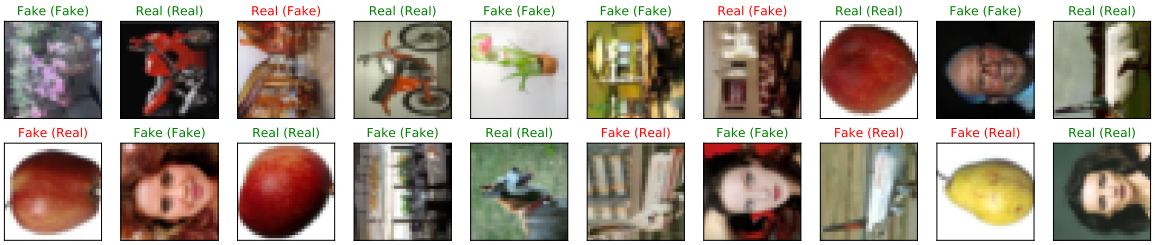

Figure 4: DenseNet test set miss-classification images

Similar to ResNet, DenseNet required several more epochs and gave us 82% test accuracy.

Figure 5: Xception test set miss-classification images

3.3. Xception

At epoch 9, we were able to achieve a test accuracy of 87%. However, the test error increases for the rest of the way.

3.4. NASNet

Figure 6: NASNet test set miss-classification images

Similar to Xception, we observed 87% accuracy but with more epochs.

3.5. InceptionResNetV2

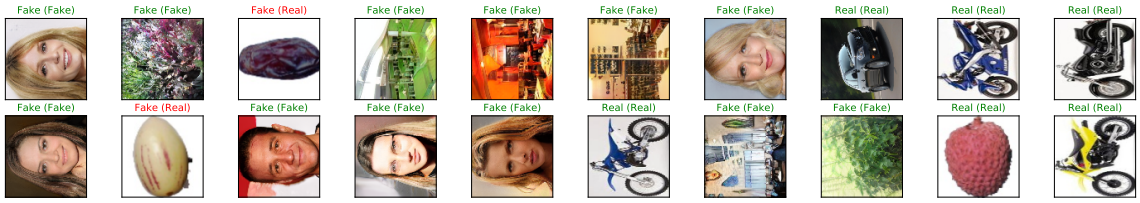

Figure 7: InceptionResNetV2 test set miss-classification images

The best model that was able to balance computational time, complexity, and accuracy. With 96% accuracy on the test set at epoch 11, we were able to obtain extremely low error rates and a generalizable model.

4. Conclusion

After completing this project, we realized that even implementing popular ImageNet networks required large amounts of hyperparameter tuning. More so was the progressive GAN architecture that came with almost endless knob tuning settings. This was because what works on celebrity faces, does not work on dogs or bedrooms or motorcycles. Here is where we learned about the importance of domain knowledge and its application to building efficient network architectures for specific tasks/data. Indeed the ImageNet models will work well for many tasks, but do we really need 152 layers to determine whether a substance promotes carcinogenesis—the formation of cancer.

In a future work, we would like to explore how we might be able to trick ImageNet models into classifying, say, a cat as a motorcycle. With the use of GANs, is it possible to generate images that to our eyes is a cat, but to ResNet is a motorcycle. Is there an underlying distribution such that two or more completely different classes can be wrongly classified. We believe attacking the network architectures might be the next biggest challenge if we were to use these models for real time decision making—i.e. self-driving cars.

References

[1] T. Karras, T. Aila, S. Laine, J. Lehtinen, Progressive growing of gans for improved quality, stability, and variation, CoRR abs/1710.10196 (2017).

[2] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, CoRR abs/1512.03385 (2015).

[3] F. Chollet, Xception: Deep learning with depthwise separable convolutions, CoRR abs/1610.02357 (2016).

[4] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le, Learning transferable architectures for scalable image recognition, CoRR abs/1707.07012 (2017).