Generating Faces using Generative Adversarial Networks

Daniel Sanche
University of Saskatchewan
drs801
11123105
drs801@mail.usask.ca

1. ABSTRACT

A significant amount of time and effort is expended in the entertainment industry to generate background assets, which add substance to an environment without being the focus of the viewer's attention. For artificially rendered scenes of crowds in particular, background characters may need to be designed individually, which creates a substantial burden on the character artists for relatively minor scenes. Machine learning can attempt to alleviate this problem by creating a model that can generate realistic human faces on demand. This paper explores using a Generative Adversarial Network (GAN) for this task, modified to allow sex and age features to be specified for individual outputs. In this way, faces can be generated from data alone, with no input from artists required. The model follows the GAN convention of having a discriminator and a generator working as adversaries to improve outputs over time. Images are drawn from a heavily filtered subset of the IMDB-WIKI dataset. This dataset is made up of images of celebrities scraped from the internet, along with useful metadata. After generation, the output images were evaluated by running an OpenCV face detection function over all outputs. The evaluation found faces in the output images at a comparable rate to the images in the original dataset. Although current results are promising, more work is required to generate faces that are truly indistinguishable from real ones to the human eye.

2. INTRODUCTION

In the entertainment industry, background assets are often used to add substance to imaginary world. These assets can include crowds of people, cities, or forests. They are designed to make the world feel more alive, without actually being the focus of the audience's attention. The film industry has traditionally solved this problem by hiring actors and designing set pieces, which can be costly to produce. While CGI can occasionally cut down on costs, it is often not an ideal solution for complex assets such as background characters in a crowd, which may need to each be created individually.

The games industry has attempted to mitigate this problem through the use of procedural generation, in which assets can be created infinitely on demand. Still, a significant amount of work is required to design the procedural generation algorithms themselves. Programmers often have to describe how to properly generate an object through math, which is typically a non-trivial problem. Using standard methods, there is no simple way to generate a large quantity of realistic background assets, which leads to expensive productions or mediocre assets.

This paper will specifically focus on the problem of generating realistic human faces from data alone. Face generation is particularly interesting, because it has been observed by psychologists that the brain has a special "face-specific system" fine-tuned for human facial recognition [1]. Because faces are so important to the brain, any irregularity in artificial faces is immediately noticed by the human visual system. This leads to an effect commonly known as the uncanny valley [2], in which a sense of uneasiness is caused by artificial faces that are close to lifelike, but imperfect. This sensitivity to errors makes face generation a particularly interesting image generation problem.

Although data-generated faces may not be suitable for important characters, they can be ideal for creating background characters to populate a world. A machine learning approach would be capable of creating large crowds of faces for a fraction of the cost of manual animation. Even low budget productions could use a pre-trained model to generate assets at will. This would save considerable time and budget, that could then be reallocated towards the more important assets in the production.

To solve this problem, I have used a Generative Adversarial Network (GAN), as proposed by Goodfellow et al [3]. At a high level, GANs are a neural net architecture that can take in random noise as input, and output an image; in this case a realistic human face. For this project, I have modified the canonical GAN structure to add additional labels for the gender and sex. When these labels are input to the network, it will generate faces that conform to the input labels. In this way, the user of the program can control the properties of the output image.

This project consisted of the following steps:

- Create a script to filter out the usable files out of the IMDB-WIKI dataset
- Create a script to load batches of images into memory for use by the model, using multiple CPU threads so as not to block the training process
- Build a discriminator network that can read an image, and output a confidence value
- Build a generator network that can create images based on a series of inputs
- Train networks until they produce acceptable results
- Create script to create sample images from generator, so we can extract images after training
- Create a script to evaluate the outputs

Due to the subjective nature of the model's outputs, it would be desirable to conduct a user study to evaluate the model's outputs. In this study, human participants would attempt to evaluate results by attempting to determine which images were generated, and which ones were real faces. Due to the lack of time, I instead opted to use the standard face detection module in OpenCV to evaluate the model. This module was designed to efficiently recognize faces in photos, so it can provide a simple, quantifiable metric to indicate the number of recognizable faces being generated.

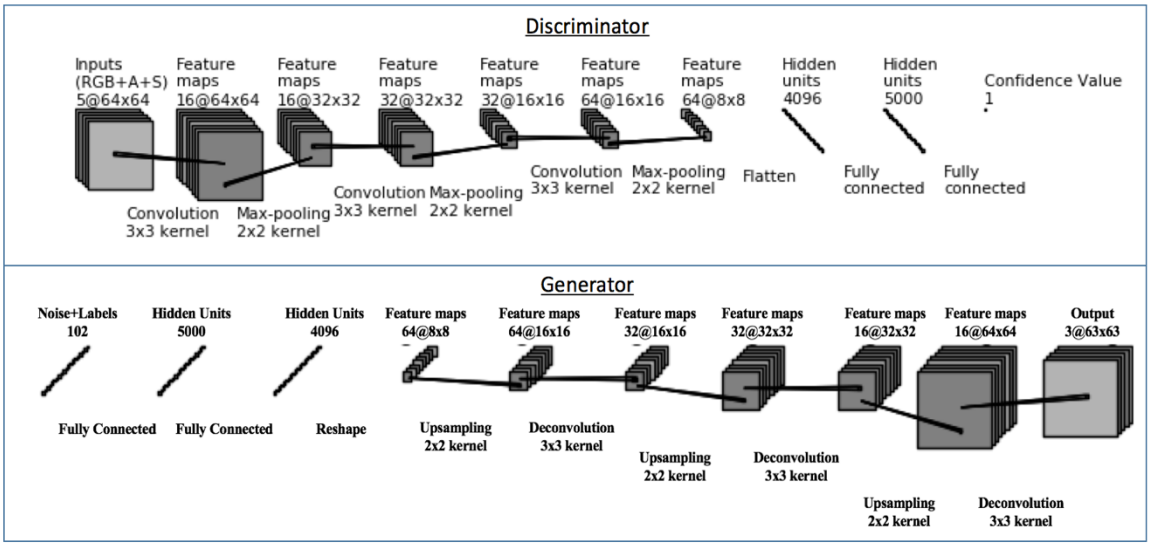

Figure 1. Architecture diagram. Top box is discriminator; bottom box is generator.

3. ALGORITHM

The model I used for this problem was based on the Generative Adversarial Network (GAN) model [3]. At a high level, this model consists of two convolutional neural networks designed with competing goals. The first network is the generator, which can generate artificial images on demand. The second network is the discriminator, which takes in an image, and tries to determine whether it came from the dataset, or the generator. At the beginning of training, the generator will only output static noise, but over time the discriminator will learn relevant facial features, and the generator will learn how to replicate them.

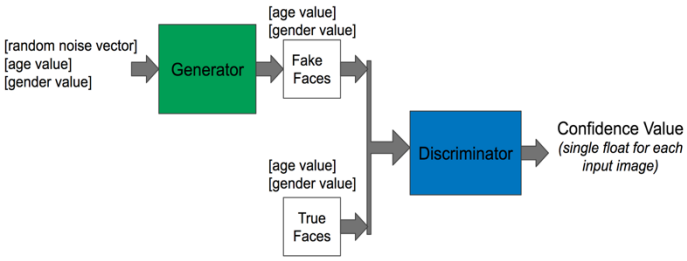

Figure 2. High-level architecture diagram

The discriminator network was constructed similarly to a typical convolutional neural network classifier model. It takes in a batch of input images, and runs them through three convolutional layers, separated by max pooling layers. The convolutional layers scan through the image, learning to recognize important features in the input. The max pool layers perform a down sampling operation, shrining their input by a factor of 4. They do this by separating pixels into blocks of 4, and keeping the largest value in each block. The discriminator then ends with two fully connected layers, with the final layer having only a float value.

The discriminator's convolutional layers use a patch size of 3, with strides of 1. The first convolutional layer produces an image with 16 channels, the next produces 32, and the last 64. This is done so that more feature information can be stored in latter layers, to prevent information from being lost due to a bottleneck effect. Each convolutional layer is followed by a rectified linear unit (ReLU) non-linearity function, as this function become popular over recent years to prevent the vanishing gradient problem [4].

The first feed-forward layer has 5000 neurons, which makes up the majority of the weights in the network. While this number of neurons is likely unnecessarily large, it was chosen to ensure that the network can make the appropriate inferences from the convolutional output. The output layer is a single neuron, which can be interpreted as the probability that the input image came from the dataset, rather than the generator network While every other layer used ReLU as the non-linearity function, the output layer uses a sigmoid to constrain the value to be between 0 and 1 for easy analysis.

The most significant difference between the discriminator network and a typical convolutional neural network is how the sex and age labels are incorporated into the network. The age and sex labels are specified as inputs, along with the input images. Before the first convolutional layer, there is a step that creates a 64 x 64 matrix for both the sex and gender label. These matrices are then interpreted as image channels, and concatenated with the input image's 3 RGB channels, leaving it with 5 channels. In this way, the labels are baked into the input image. Now, as the network is learning to associate certain features with the dataset, it must also learn to associate specific features with specific labels.

The generator network was constructed to be the inverse of the discriminator. As its input, the generator takes in a 100 value noise vector, along with age and sex labels. All inputs are 32 bit floats between -1 and 1. The inputs are then concatenated into a single vector, and fed the network. The generator is made up of two feed forward layers, followed by three deconvolution and upsampling layers. The deconvolution layers work as the opposite of a typical convolutional layer. They look at a window of the image in the previous layer, and use the values in the window to construct a new pixel in the next layer. Similarity, the upsampling layers work as the reverse of the max pool layers in the discriminator; rather than shrinking an image and keeping the largest pixel value, the upsampling layers enlarge the image by repeating each pixel into a block of four identical ones.

Like the discriminator, the generator's deconvolution layers use a patch size of 3, a stride size of 1, and ReLU non-linarites between layers. Unlike the discriminator, deconvolution layers decrease the number of channels of the input, rather than increase it. The first deconvolution layer takes an input depth of 64 channels and converts it to 32, then the second takes 32 and converts it to 16, then the output layer one converts from 16 to 3. In this way, we are condensing information over the dimensions over time, and the final layer can be interpreted as a traditional 3 channel RGB image.

Typically, RGB images are represented using 3 color channels with pixels ranging from 0 to 1. For this reason, a sigmoid function would be natural to apply to the generator's output layer. For this network, however, images are represented using the range from -1 to 1 to make use of the extra precision. For this reason, a tanh function is applied to the output layer instead. The image is later scaled to the typical 0 to 1 range before being displayed to the user, however.

To train the network, cost functions are built using the confidence value output returned by the discriminator. Individual cost functions for each network are required because the networks are optimizing for opposite goals. For the discriminator to minimize its cost, it must output 0 for every image created by the generator, and 1 for every image from the IMDB-WIKI dataset. For the generator to minimize its cost, the discriminator must output 1 for every generated image, indicating that it couldn't discriminate the generated batch from the true batch.

Both cost functions use the TensorFlow's standard sigmoid cross entropy loss function to calculate the total cost value. This function will return higher cost values the further away the output value is from the goal value. Each cost tensor was then fed into TensorFlow's Adam optimizer for the training step. Importantly, each training function was set to only optimize the weights belonging to the network being trained. This is an essential point, because the discriminator's task would be much easier if it could simply sabotage the performance of the generator. Instead, the discriminator can only train the weights related to its own task.

During training time, the program will alternate between training each network. It is important that both networks have relatively equal cost values at all times to ensure that they are training on relevant features. For example, if the discriminator were to become too powerful relative to the generator, it may simply learn to exploit flaws in the weak generator rather, than learn true facial features. If this were to occur, the generator would not be able to back-propagate useful information back to the generator, which would only cause the gap between the networks to increase. To alleviate this issue, the training script keeps track of the loss values of each network as training occurs, and temporarily stops training one network if it becomes too powerful, to give the other a chance to catch up. The discriminator will stop training if its cost is ever half the cost of the generator. The generator was given a bit more flexibility; it will only stop training if its cost is a third of the discriminator's. In practice, this rarely happens, as the discriminator tends to have a much lower cost than the generator. The costs for each network during a typical training process can be seen in Figure 3.

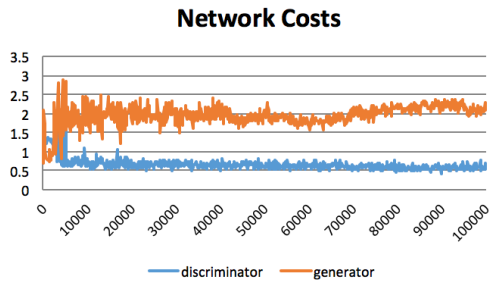

Figure 3. Training costs over 100,000 training batches

While training the networks, I did some adjustments on the learning rate. I found that early on in the training cycle, it was useful to have a relatively larger rate, to allow the general face shapes to emerge from noise faster. As training continued, I found that smaller learning rates were preferred, to allow existing faces to improve themselves, without the possibility that they would completely morph into other faces. For this reason, I decided to decay the learning rate over time. The learning rate started at an initial value of 5e-4, and then multiplies itself by 0.996 after every 100 training batches. Additionally, I added a floor to the learning rate, so it will stop decreasing when it reaches 5e-5. Figure 4 shows the evolution of the learning rate over time.

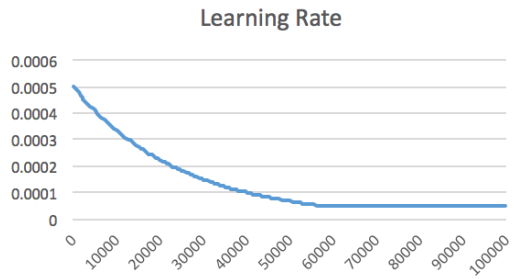

Figure 4. Learning rate degradation over 100,000 training batches.

One of the major risks for this project was the potential for overfitting the dataset. If the generator managed to find a way to reproduce the dataset exactly, there would be no way for the discriminator to tell the images apart. Although the generator would be achieving a perfect score, it wouldn't actually be learning anything interesting. One way this project attempts to avoid overfitting is by limiting the number of weights in the model. If a deep network was used with a large number of, it would simply learn to memorize individual pixels in order to identify images. Because the size of the model was limited, it forces the networks to learn meaningful abstractions from the dataset, rather than relying on memorization. Another technique used to avoid overfitting was to use a small learning rate. Because the learning rate was kept low, the network will only learn a small portion from any individual batch, so memorization should be avoided. Finally, batch normalization layers are used in the generator, which have been found to aid in regularization [6].

4. DATASET

For this project, I used the IMDB-WIKI dataset of faces. This dataset consists of images of celebrities scraped from public databases on the internet, along with relevant metadata. In total, the dataset contained 460,723 images from IMDB and 62,328 images from Wikipedia. The data was labeled with fields such as name gender, birthdate, picture capture date, and the face quality score. The age for each subject could be obtained by subtracting the birthdate from the picture capture date.

Figure 5. Sample images from IMDB-WIKI.

Although the dataset provided a large quantity of images, not all were usable. Many images were low resolution, contained no faces (or multiple), or were grayscale images. Additionally, this project required labeled gender and age information, so any images with missing or incorrect labels were also removed. After filtering out unusable data, I was left with only 198,467 images.

After filtering, the remaining images are all 8 bit RGB images. They have varying resolutions, ranging between 64x57 and 501x501. The resolution differences are largely irrelevant, however, due to the fact that all images were scaled to a consistent 64x64 resolution for training.

The image data is categorized using two labels: age and gender. Each image is tagged with a binary gender label of male or female, and an integer between 10 and 100 representing the subject's age. The overall gender distribution is relatively close; the dataset is 57.7% male and 42.3% female. The data does tend to skew toward a younger demographic, however. The overall mean age is 37.35, with a slightly younger skew for women than men, as seen in Figure 6. This means that, without proper care, the classifier network would tend to favour ages in this range.

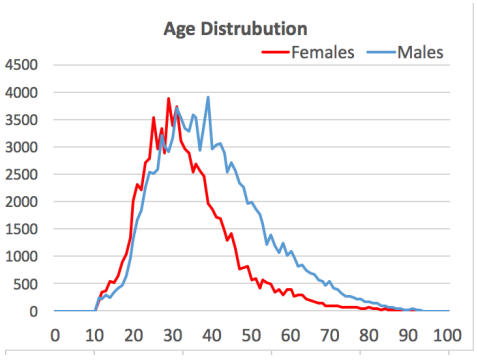

Figure 6. IMDB-WIKI age distribution.

To mitigate this effect, I was careful to maintain equal distributions for each label in each training batch. This was accomplished by creating a data loader script to manage image retrieval. This script had functions to would sort the dataset's images into 16 bins based on gender and age, and then form a batch with equal representation of each bin. In this way, the neural network should not learn a bias towards any age or gender features, even if they are over represented in the overall dataset.

Although the dataset had a large quantity of images, it did have some quality issues. Even after removing over half of the images from the dataset due to quality issues, there were still a large number of errors remaining. The biggest issue was incorrectly labeled data. While data with missing labels was simple to filter out, data with the incorrect age or gender labels could only be fixed by manually combing through the data, which would take a significant amount of time on such a large dataset. By drawing random samples, I determined that approximately one in every 50 images fed into the network seemed to have erroneous labels. This could cause the network to have trouble learning certain features. Additionaly, many faces were faced away from the camera or otherwise obscured, which likely also caused training issues.

5. RESULTS

A sample of the images generated by this model can be seen in Figure 7. These images were produced after 480,000 training batches. Looking at the outputs, it can be seen that the results are mixed. While some images are clear and high quality, other images are filled with noise. In some images, no face is visible at all. While the current results are promising, more work is required in order to make outputs that are indistinguishable from real faces.

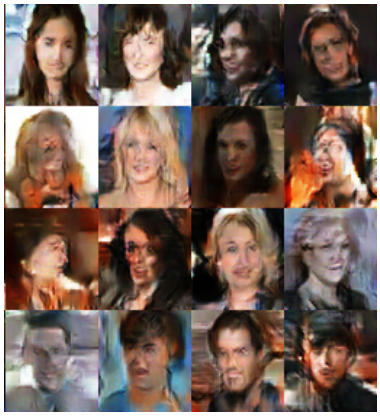

Figure 7. Random samples drawn from generator after 480,000 training batches.

To evaluate the results in a more quantifiable manner, I used the standard OpenCV face detector to determine which portion of generated images had visible faces. This face detector works using Haar cascade classifiers, in which a series of feature filters are slid across the image to determine whether it contains a face [5]. The face detector takes as input an XML file that describes the features that make up a face. For the purposes of face detection, I used the following built-in OpenCV cascade files:

- haarcascade_eye_tree_eyeglasses.xml
- haarcascade_frontalface_alt_tree.xml
- haarcascade_frontalface_default.xml
- lbpcascade_frontalface.xml
- haarcascade_frontalface_alt2.xml
- haarcascade_frontalface_alt.xml
- haarcascade_smile.xml

These files each contain different facial descriptors. If any cascade detects a face, the face is seen as detected for my analysis. The face detector will look for faces between 32x32 to 64x64 pixels, using a scale factor of 1.1.

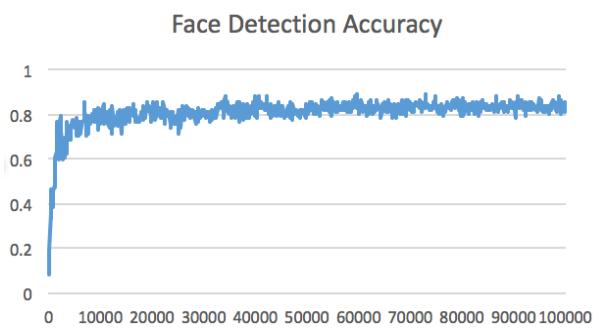

Figure 8. Face recognition improvements during training.

Accuracy was based on percentage of faces recognized in samples of 300 images

After running the OpenCV face detector on a set of 100,000 randomly generated samples, it was able to detect detected faces in 84.83% of the set. When run on the ground truth dataset, it could detect 86.12% of faces. This is likely due to the fact that many faces in the original dataset were turned away from the camera, or otherwise obstructed. The cascade files used for detection were primarily focused on frontal faces, so any obstructed faces would likely be undetected. If any generated images chose to replicate the obstructions found in the dataset, they too would be undetectable by the OpenCV face detection function. For this reason, I interpret the evaluation results to be favorable, because they demonstrate that the majority of generated faces did have the necessary features to trick a conventional face detection algorithm.

Figure 8 shows how the accuracy of faces generated improved over time. This chart was generated by instructing the network to run the face detection algorithm on a batch of 300 faces after every 100 training batches. The chart shows the first 100,000 training rounds, after which there is little noticable improvement. This chart shows that accuracy quickly spikes up from 0% to ~75% on average after the first 5000 training rounds, and then slowly increases up to to range reported in the final result. It should be noted that the final 84.83% result recorded is more accurate than the values in Figure 8, because a smaller sample size was used to generate the chart in order to make the data collection tractible.

Figure 9. Images generated with different sex label values, but consistent noise values. Top row is women, bottom row is men

Generation using sex labels worked reasonably well. Figure 9 provides a random sample of images in which all noise and age values are held consistent, with only the sex label changed. In these images, it is clear that gender-specific features are saved and applied to the image. While the overall faces are typically very similar, features such as facial shape, hair, and clothes often differ significantly between images. Although some images are more apparent than others, it is usually clear that the intended gender is being represented in the output.

The age labels, on the other hand, were not captured as successfully, as can be seen in Figure 10. I attempted to visualize the aging of a single face, by keeping the noise and sex consistent, but increasing the age value over time. If you look closely at some images, you can see some grey hairs or bald spots as the sample becomes older. Overall, however, many of these images resulted in only noisy blotches of color, often with no face clearly visible. Because of this, I consider the age label sampling to be unsuccessful at this time.

Figure 10. Images generated with different age labels. Network is input ages going from 15 to 75 from left to right. Each row is a separate individual with consistent noise values.

6. DISCUSSION

Although the results leave something to be desired, I believe they show the potential of Generative Adversarial Networks to produce faces from data alone. By simply training the network on batches of images of celebrities, it was able to adapt from outputting noise, to outputting faces. Models like this could be used to generate crowds of unique faces to populate the background of films or games, without any input from artists.

GAN models are highly generic as well; if the network was fed with a different dataset, it would learn to generate entirely different images, with very little changes in the architecture. The same architecture that was used to generate faces, could later be used to generate trees or buildings, simply by training on a new dataset. This could be a very useful feature for artists, who may have to generate many different objects to populate their worlds.

Additionally, this work shows the potential, and the limitations of generating images constrained to specific labels. While the sex label was typically captured fairly accurately, the age label was not. This is likely due to the fact that aging produces subtler, more complex features than those found between sexes. While sex can be encoded as a simple binary feature, age is represented as a continuous value. This means that there is a much larger range of possible age values and associated features for the network to learn. Additionally, some people age differently than others, and it can often be difficult for even the humans to determine which features to associate with which precise ages. All of this is then complicated by the fact that the dataset had many errors in the age labels associated with each face, which likely caused confusion in the network, preventing it from learning relevant age features.

Working on this project has helped give me a better understanding of how deep learning models work. Working on generative models allowed me to get a good idea of exactly what the network was capturing. When you see the network learning to recognize a facial feature, like hair, you know that somewhere in the network, there is an internal representation of hair. This insight will be useful when working on other networks that are less easy to visualize in the future.

Working on this project also helped me appreciate the difficulty of debugging neural networks. There were many instances where the network was returning unexpected results, and there was no simple way to determine what the issue was. I would have to resort to changing a parameter, and then waiting for the network to train for a while to see if it made a visible difference. To complicate the issue, Tensorflow is inherently difficult to debug, due to the fact that the tensor graph is run in a lower level C component outside of python, and is inaccessible from the debugger. I was unable to set breakpoints and step through the code like a typical python program. I quickly learned that it is

important to carefully think through any changes before testing them.

I also learned the importance of using a high quality dataset. When I first started training, I was having difficulty generating anything that looked like faces. My outputs improved dramatically as soon as I added filtering to the data that would be input to the network. This is because any incorrect labels or faulty images would cause the network to make false correlations in the data, and prevent meaningful training from taking place. Even after performing filtering, the network was still showing signs of having issues due to the low quality dataset. This project has made it clear to me how important clean data is, and not just the machine learning algorithm itself.

Batch normalization also proved to be an essential key to the problem. Earlier attempts to create the network lead to issues where all generated images appeared nearly identical. This lead to an issue where the discriminator would only have to learn a single face to classify all generated outputs. Instead of the generator improving over time, it would have to constantly change the single face it could generate every time the discriminator learned to recognize it. After doing some research, I found the solution in batch normalization, which would expand the variance in the output after each layer. After researching and applying batch normalization, the network results improved dramatically.

7. FUTURE WORK

The next steps for this project would be to increase the quality of results produced. To achieve this goal, it would likely be important to fix issues in the dataset. I anticipate that if the incorrect labels were fixed and images without clear faces were removed, the model would be able to produce much higher results. Resolving this issue could be as simple as writing a script that goes through the dataset, displaying images and allowing the user to indicate whether any errors are present.

Another way to produce higher quality results would be to simplify the age parameter. Currently, the age is represented as a float value, which means that the neural network must be able to infer proper facial features for any possible age in the continuous range. One possible way to achieve improved results would be to threshold different ages into different bins. By limiting possible age values to a small, discrete set of bins, the network may be able to better capture relevant features. In an extreme case, the age ranges could be binned into three simple categories: teenagers, adults, and seniors. This would likely dramatically improve the quality of the age outputs.

After the outputs themselves are improved, the next step for this project would be to create better evaluation methods. While the OpenCV face detector allowed a simple, unbiased method of determining the quality of the outputs, it doesn't accurately capture the main motivation of the project: to create faces that look indistinguishable from real ones to human observers. To solve this, a study could be run where human participants would evaluate results by attempting to determine which images were generated, and which ones were real faces. This evaluation method would be more in line with the goal behind the project.

One way to expand on the work in this project would be to train the network to learn new parameters for the data. Parameters such as skin colour, accessories (hats, scarfs, etc.), or facial expressions could be associated with the data, and then used to selectively generate images. These features weren't labeled in the dataset, so an improved dataset would likely be necessary. Additionally, these features can be subtle and hard to identify, so it is likely such a change would require significant changes to the architecture of the network to be captured properly.

Finally, it would be interesting to explore altering existing images instead of simply generating new ones. In the model used for this project, images were generated by inputting noise to the generator network. In order to use existing images as input, they would need to be converted into a vector encoding that could be input to the generator, rather than the noise vector. This can be accomplished using a new "encoder" network, which takes in an image, and returns a vector encoding. This network could be trained by feeding the encoder's output vector into the generator network's input, and comparing the generator's output against the encoder's input. To receive a cost value of 0, both images should be identical. In this way, the encoder and generator would together form an autoencoder network, in which a unique encoding for each input image is learned and saved in the bottleneck between them. In theory, this technique could allow arbitrary images to be sent as input to the generator, which could then modify features, such as the age or gender labels used in this project.

8. SUMMARY

Although the final outputs of the model have clear room for improvement, this work can be viewed as a proof of concept for what is possible. The OpenCV evaluation proves that important facial features are truly being captured by the network. Furthermore, the success of the sex label indicates that the output results can successfully be controlled, although the failure of the age label proves that careful consideration is needed to produce acceptable results. The completed generator network can now be used to generate faces at will, with the number of possible outputs constrained only by the float capacity of the 100 value noise vector input. In the future, methods like these could be used to generate large crowds of people using only a publicly available dataset, saving the entertainment industry significant time and money, and allowing more immersive worlds to be created by projects with a modest budget.

9. REFERENCES

[1] M. Moscovitch, G. Winocur, and M. Behrmann. What Is Special about Face Recognition? Nineteen Experiments on a Person with Visual Object Agnosia and Dyslexia but Normal Face Recognition. *Journal of Cognitive Neuroscience*, Vol 9, No 5, 555-604, 1997.

[2] M. Masahiro, K. MacDorman, and N. Kageki. The Uncanny Valley [From the field]. *IEEE Robotics & Automation Magazine,* volume 19, issue 2, pages 90-100, 2012

[3] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. *arXiv:1406.2661,* 2014

[4] X. Glorot, A. Bordes, and Y. Bengio. Deep Sparse Rectifier Neural Networks. *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15, pp. 315–323, 2011.

[5] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. *Computer Vision and Pattern Recognition*, 2011.

[6] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167,* 2015