Building an Automatic Sprite Generator with Deep Convolutional Generative Adversarial Networks

Lewis Horsley
School of Computer Science
and Electronic Engineering,
University of Essex, Colchester, UK
lhorsl@essex.ac.uk

Diego Perez-Liebana
School of Computer Science
and Electronic Engineering,
University of Essex, Colchester, UK
dperez@essex.ac.uk

***Abstract*—In recent years, image generation using Convolutional Neural Networks (CNNs) has become increasingly popular in the computer vision domain. However, there is less attention on using CNNs for sprite generation for games. A possible reason for this is that the amount of available sprite data in games is significantly less than in other domains, which typically use hundreds of thousands of images, or even more. In this work, we provide some beginning evidence that CNNs can be utilized for game-style sprite generation, even with small input datasets. We utilize a class of Generative Adversarial Network (GAN) known as a Deep Convolutional Generative Adversarial Network (DCGAN) for unsupervised learning and generation of new sprites. We have trained our network on various custom datasets, which contains human-like characters, faces and creatures in general. Results show evidence that CNNs can generate unique sprites from the input data that was provided as input.**

I. Introduction

Recent developments in Deep Neural Networks (DNN) have crafted the foundations for unique image creation tools. Several implementations utilizing related technologies have proven that, among others, DNNs, Convolutional Neural Networks (CNN), Generative Adversarial Networks (GAN) are capable of producing very eye pleasing synthesized images.

Fine artwork crafted by very skilled individuals have been key identifiers of culture throughout the ages. Many artists such as Pablo Picasso, Vincent van Gogh and Leonardo da Vinci have placed their mark in time with exquisite mixes of content and style within their artwork. The ability of an artificial system being capable of creating such unique and beautiful artwork is a captivating and exciting concept. An example of a method that builds an artificial neural system to generate fine art is that of Neural Style Transfer. This method consists of separating the style of one image, and the content from another arbitrary target image so that the content and style representations can be combined to form new images. Gradient descent is applied from random noise in order to minimize deviation from the content representation of the target image and the style representation of the style image.

There have been several research articles and implementations created for Neural Style Transfer. Various implementations incorporate interesting and diverse methods to provide a refreshing experience of style transfer. An example of this diversity can be seen in a style transfer implementation by Alex J. Champandard [1], where semantic segmentation works with style transfer in order to take a basic two-bit image and enhance the image into fine artwork.

Artwork is one of the most most important assets within games: models, textures, graphical user interfaces and simply 2D art are core components of every endeavor in digital games. The creation of these models are time consuming and typically subject to the creativity of the artist or designer. A system that automatically generates new graphical assets can be of a great aid in the daily work of the creative team. The objective of this work is to conduct research into these technologies in order to produce a reliable and accurate synthetic 2D sprite style generator. To the knowledge of the authors, this is the first sprite generation tool that utilizes DNN technologies to create unique sprite style images. There is another exciting piece of research which generates sprites, however this research by S. Reed [2] introduces a method of forming different animations for provided sprite images. Concretely, this work approaches this problem by adopting a Deep Convolutional Generative Adversarial Network (DCGAN, [3]) framework, which combines the properties of Generative Adversarial Networks (GAN) with the CNN architectural constraints in order to stabilize the learning process.

This paper is structured as follows. First, Section II provides an overview of usages of Deep Learning for automatic generation of art assets. Then, Section III describes the theory behind Generative Adversarial Networks, to later explain the approach followed in this work to generate 2D Sprites in Section IV. The experimental process followed for sprite generation is described in Section V. Results are shown and discussed in Section VI, and conclusions presented in Section VII.

II. Deep Learning for Art

Utilizing Convolutional Neural Networks for image synthesis and style transfer has become a fascinating and exciting line of work of Deep Learning within the recent years. Several implementations of style transfer have been created so that new ways of producing fine art using Artificial Intelligence (AI) can be explored by thousands of people. For example, a twitter

978-1-5386-3233-8/17/$31.00 ©2017 IEEE

bot known as DeepForger[1] was created in order to provide the general public with an accessible and effortless way of interacting with this research and introducing the masses to this surprisingly elegant method of creating interesting artwork using AI.

Alex J. Champandard [1] introduced the concept of applying semantic annotations to images in order to create a content-aware algorithm that offers control over the outcome image. Even though a Convolutional Network routinely extracts semantic information for classification, this information is poorly exploited within several gram-based and patch-based style transfer implementations. Through introducing this concept, the research aims to reduce the unpredictable behaviour that is currently associated with relevant implementations of style transfer as well as bridging the gap between generative algorithms and pixel labelling neural networks. By achieving these goals, an overall increase in the quality of the images produced by the system would be recognised.

There are various research articles that have become the inspiration and guidelines for this method of Neural Style Transfer. Firstly, this contribution to style transfer has been built upon a neural network approach to style transfer which involves a patch-based method of understanding patterns in images. This patch-based approach described in a report by Chuan Li and Michael Wand [4] operates on the output of convolutional layers of a ConvNet. Neural patches of 3×3 are matched between style and content using a nearest neighbour calculation. These patches give the algorithm an understanding of the patterns within the image. Secondly, research on semantic segmentation was the foundation for the semantic annotations feature of Champandard's system. The fundamental concept of semantic segmentation is to cluster parts of images together that belong in the same class. A more detailed report on the topic can be found within Martin Thomas survey on semantic segmentation [5].

A CNN model [6] (named *VGG* after the authors' team name) is utilized within this system. More specifically the pooling and convolutional layers of the network are used alongside a 3×3 kernel in order to provide intermediate post-activation results. These results consist of N channels, which capture patterns from the images. Patterns captured include features such as colours, textures, strokes etc. Additional semantic channels are concatenated onto the *VGG* network; these concatenated channels are of size M, where M is computed by down-sampling a static semantic map provided as input by the user. The result of this concatenation is an output of $N + M$ channels.

Another important line of work on generative art is the usage of Generative Adversarial Networks (GAN). Several variations of GANs have been researched and implemented throughout the recent years. Most implementations have found major success in face generation tasks. A report by Jon Gauthier [7] details how he was able to produce realistic images of faces by utilizing a GAN. Figure 1 shows some final outputs from a GAN trained in the CIFAR-10 dataset, consists of 60000 32×32 colour images uniformly distributed in 10 different classes (airplanes, birds, cats, trucks, etc).

[1]https://twitter.com/deepforger

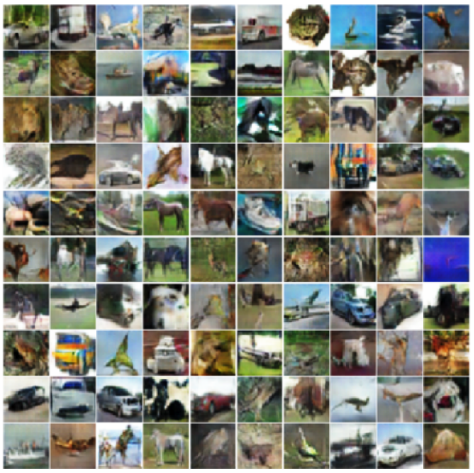

Fig. 1: Samples generated by GANs during training in the CIFAR-10 dataset [8]

Several attempts of up-scaling GANs by utilizing Convolutional Networks (ConvNets) have been attempted in the past and most often resulted in failure. One historical approach was able to iteratively upscale low resolution generated images. This lead to a conclusion that invoking this method would provide more reliable generated output than what could be expected from a general GAN. This alternative approach to up-scaling GANs was labelled LAPGAN, and is detailed further in a paper by E. Denton et al. [9].

DCGANs has become an important role in many GAN based implementations. Many programs tend to use DCGANs over GANs due to the up-scaling having next to no downsides. For example, a recent program utilized DCGANs to generate very accurate human faces with a variety of different characteristics. This program is called Neural Faces and was developed by Facebook AI Research [10]. This program was capable of producing very accurate representations of human faces.

Figure 2 shows some of the representations generated. These stability improvements introduced by DCGANs have paved a way to a much more stable and reliable GAN for developers to utilize. Faults within regular GANs have been identified and resolved in order to improve upon the GANs introduced by I. Goodfellow et al. [11].

There are several reasons why GANs have been chosen over a Neural Style Transfer approach to automatically generate sprites from a given set in this work. Firstly, due to the competition of the generator and discrimination networks (see Section III for a description of GANs), the quality of the images are consistently improved until the images are deemed just as good as the test data. Therefore, utilizing training data that applies the sprite style would imply the generated images would be good quality sprite style images. Neural Style Transfer does not offer this method of checking and enhancing the image. Thus, if the style transfer wasn't as good

Fig. 2: Faces generated by Neural Face [10]. This program implements DCGANs as a method of generating faces with a variety of diverse characteristics, such as gender, smiling and wearing glasses.

as expected there is no method of enhancing that image other than to run the style transfer process again. Secondly, GANs are trained and provide output from utilizing the training data. This means that no input from the user is required. This favours towards the objective of requiring as little user input as possible and therefore more favourable than the Neural Style Transfer approach.

III. Generative Adversarial Networks

Within the past couple of years, researchers in computer vision have become increasingly more interested in generative models. These models are capable of constructing data similar to the data provided by the user. The key idea behind generative models is that each model should have a good internal representation of the data it is learning from. For example, if a generative model was learning from a dataset consisting of images of cats, then the model should have a good internal representation of a cat. Thus, this representation can be utilized for other related tasks such as image creation and classification. A relatively new and promising approach is using Generative Adversarial Networks (GANs).

GANs were originally proposed and discussed by Ian J. Goodfellow et al. [11]. Within their work, the authors proposed an Adversarial Net Framework. This framework has a generative model (G) that is pitted against an adversary: a discriminative model (D) that learns to distinguish samples from the model distribution against samples from the data distribution. To visualize this framework, G can be seen as a group of counterfeiters. These counterfeiters are trying to produce fake currency and use it without detection. D can be visualized as the police, trying to detect the counterfeit currency. Competition between the two models strives each model to enhance their methods until finally the model distribution samples is indistinguishable from the data distribution samples.

The adversarial networks are the generator network and the discriminator network, they are both multilayer perceptrons (MLP). In order for the generator network and discrimination network to improve their methods, the generators distribution p_g over data x must be computed. Firstly, a prior on input noise variables $p_z(z)$ is defined, then a mapping to data space is defined as $G(z;\theta_g)$, where G is a differentiable function with parameters θ_g. A second neural network model is then introduced as $D(z;\theta_d)$, which outputs a single scalar. $D(x)$ is used to represent the probability that x came from the data distribution rather than the generator distribution p_g. Therefore, D is trained to maximise the possibility of correctly labelling the training samples and the generator samples from G. During the training of D, G is simultaneously trained to minimize $\log(1-D(G(z))$. As indicated in [11], D and G play a two-player minimax game with value function as described in Equation 1.

$$\min_G \max_D \; V(G,D) = E_{x-p_{data}(x)}\left[\log D(x)\right] + E_{z-p_z(x)}\left[\log 1 - D(G(z))\right] \tag{1}$$

Figure 3 presents a visual explanation of this approach. The Generative Adversarial Networks are trained simultaneously by updating the discriminative distribution (D, blue dotted line). The graphs show how D discriminates between the samples from the data distribution p_x (black dotted line) and the samples from the generator distribution p_g (G, green solid line). The lower line represents the domain in which z is sampled. The upper line is the domain x. The upwards arrows highlight the mapping $x = G(z)$ which establishes the non-uniform distribution p_g on transformed samples. (a) shows an adversarial pair close to convergence, meaning G is becoming more accurate in its samples but D can still distinguish between the two samples. (b) Within the inner loop of the algorithm, D is being trained to discriminate samples from data. (c) After updating G, the gradient of D has guided the generator to regions that are more likely to be classified as data. (d) G and D can no longer be improved as they have reached a point where $p_g = p_{data}$. Meaning the discriminator model cannot differentiate between the two samples.

There are advantages and disadvantages to using GANs for image creation. One disadvantage is that D networks training must be synchronised with the G networks training. Thus, the G network must not be trained too much without updating the D network in order to avoid the *Helvetica scenario*. This scenario occurs when G collapses too many values of z to the same value of x, meaning there is a sufficient lack in diversity to model p_{data} without just copying the data. This is a common failure often observed in GANs. As an advantage, GANs do not require Markov chains. These chains are not required because only backpropagation is used to obtain a gradient, therefore no interference during learning is required. This makes the training of the adversarial networks slightly simpler. That being said, there are several improvements that have been suggested in regards to training GANs. Some of these improvements include introducing minibatch discrimination, assessment of image quality and semi-supervised learning. These improvements amongst others are described in more detail by Tim Salimans et al. in [8].

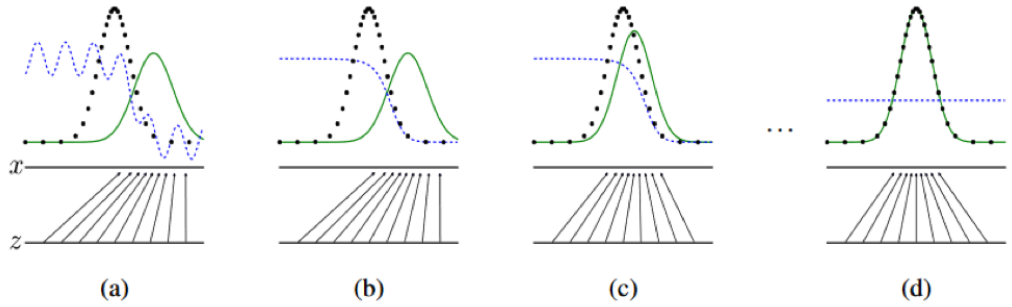

Fig. 3: Generative Adversarial Networks during training [11]

IV. A Deep Convolutional Generative Adversarial Network

The proposed architecture consists of two neural models which are trained simultaneously: a generative model Generator (G) and a discriminative model Discriminator (D). During training, G tries to maximize the possibility of D making a mistake when trying to discriminate the sample data from the training data. This leads to a training procedure which follows the pattern of D trying to detect G samples, whilst G is trying to generate samples indistinguishable from the training data. The constant competition between the two models drives both models to improve their methods of detection and generation.

As mentioned in Section III, DCGANs approach to up-scaling GANs uses ConvNets as the network architecture for the generator model and the discriminator models. In this work, several adjustments to the ConvNet architecture have been adopted in order to stabilise the DCGAN approach.

A. Generator

1) Strided Convolutions: The first of these was to replace deterministic spatial pooling functions (such as *maxpooling*), with strided convolutions for the discriminator model, and fractional-strided convolutions for the generator model. This allows the discriminator network to learn its own spatial downsampling, whilst allowing for the generator network to learn its own spatial upsampling. This method of replacing pooling functions is introduced and detailed further in J. T. Springenberg et al.'s work on re-evaluating ConvNets [12].

2) Global Average Pooling: The second concept is to eliminate any fully connected layers that are on top of convolutional features. Removing fully connected layers in a ConvNet architecture is by no means a new concept, this idea has been implemented by various different image classification and creation tool. One example of this would be the Network in Network (NIN) which is alternative deep neural network structure proposed by M. Lin et al [13]. Within the NIN architecture, fully connected layers are replaced with global average pooling in order to improve stability and reduce overfitting. Global average pooling was adopted by DCGAN but even though it did improve stability, it also damaged convergence speed. In order to correct this in-balance, DCGAN directly connects the highest convolutional features to the input and output of the respectively for the generator and discriminator models.

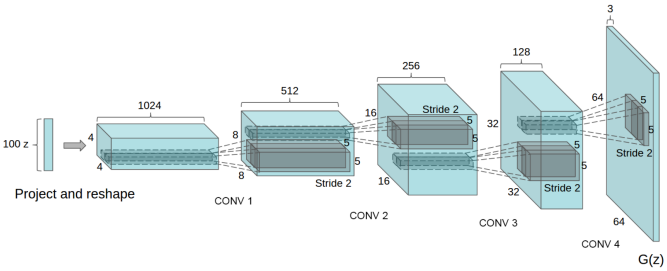

Fig. 4: Structure of the DCGAN Generator [3]

3) Batch Normalization: Finally, the third concept is to utilize Batch Normalization (Batchnorm) in order to stabilise learning. The key idea behind Batchnorm, which was introduced by S. Ioffe and C. Szegey [14], is to normalize each unit input to have zero mean and unit variance. Implementing this normalization within the generator and discriminator networks helps to deal with training problems that arise when poor initialization is in place. Gradient flow is also improved in the deeper networks. As mentioned in [3], this normalization was critical for the generator to begin learning. When batchnorm was implemented, the generator model was capable of avoiding the *Helvetica Scenario* (described in Section III). However, applying the batchnorm to all layers only resulted in a more unstable model. Therefore, batchnorm was only applied to the generator output layer and the discriminator input layer.

Finally, ReLU [15] activation is used throughout the generator, except from the output layer, which instead uses the Tanh function. Implementing ReLU allowed the generator to learn how to quickly saturate and cover the colour space of the training distribution. Figure 4 in shows a visual representation of G network.

B. Discriminator

Similar to the Generator, the Discriminator is also an MLP with various changes to the CNN architecture. The discriminator also incorporates the all-convolutional net, but

rather than replacing pooling functions with fracual-strided convolutions, the discriminator replaces pooling functions with strided convolutions. This allows the network to learn its own special downsampling. All fully connected hidden layers have been removed for deeper architectures. However, the last layer of the discriminator is flattened and fed to a single sigmoid output. Finally, batch normalization is also introduced in the discriminator network's input layer, which provides the same benefits as described in the generator. Following the DCGAN model, batch norm was not given to all layers as this would cause sample oscillation and instability. Finally, LeakyReLU activation [16] was implemented in the discriminator.

C. DCGAN

When the system is run, the generator attempts to create a sample from random noise input using what information it currently holds. The generator produces samples with the intent of magnifying the possibility of the Discriminator deciding that this sample is authentic (see Figure 5 for a visual example). At the beginning of training the generator will produce samples that are very distinguishable from the training data. Once the sample is produced, it is passed to the discriminator network. The discriminator then attempts to distinguish between the two sources (generator sample and training data) and reduce the probability that the generated image is authentic. Backpropagation of gradient information then takes place from the discriminator to the generator so that the generator can improve its method and thus learn how to generate better samples. Through various iterations of training the generator can eventually reach a point where it is simply reproducing the training data. Though this is unlikely to happen with large datasets, it has been discovered in small datasets of less than 50 samples.

Figure 5 shows a diagram with both networks, generator and discriminator. $P(A)$ holds the probability that *Generated Image* `A` is an authentic image, while $P(B)$ shows the probability that *Training Data* `B` is an authentic image. Firstly, the Generator produces a sample from random noise input with the intent of maximising P(A). The Discriminator takes the generated sample along with a sample from the training data and attempts to maximise the probability of P(B) while minimizing P(A).

V. Experimental work

The system described here was trained on three custom datasets: Human-like Characters, Environments and Faces. Further details on each dataset are given below. There was no pre-processing on the training images, other than ensuring each dataset had a standard image size (i.e. 23×23 pixels). As suggested in [3], a learning rate of 0.0002 was chosen. In all models, the slope of the leak within LeakyReLU was set to 0.2. The Adam Optimiser [17] was used to accelerate training. Finally, all weights were initialized from a zero-centred normal distribution with a standard deviation of 0.02.

No online datasets were used for training for this experiment. This is due to most dataset online using natural images like human faces (celebA [18]) and rooms (LSUN Bedrooms [19]). As these datasets do not fit the purpose of this experiment, customs datasets were created instead:

- Human-like Characters: The most valuable dataset for this experiment is Human-like Characters. This dataset consists of $1,210$ sprites which resemble humans (two legs, two arms, a face etc.). The variety of different characters is vast, some characters having capes and swords, others having guns and jetpacks. Each sprite was 23×23 pixels and no cropping was used. We trained this dataset to $15,000$ epochs with a batch-size of 32.[2]
- Faces: This dataset consists of sprites of character faces. Like the Human-like Characters dataset, this dataset has quite a lot of diversity but still sticks to some similarities (eyes, ears, face shape etc.). The diversity of the sprite range from skin colour, hoods, scars, eye patches and more. Faces only has 36 sprites for training, which is very small, but relevant to analyze the output that can be achieved. The dataset was trained to 5000 epoch and a batch-size of 15.
- Creatures: This dataset contains 517 sprites which represent various different creatures typically present within arcade games (trolls, demons, spiders). This dataset is very experimental as most sprites have next to no similarities. This dataset was trained to 10000 epochs with a batch size of 32. Each sprite had a 48×48 pixel size.

Figure 6 shows examples of the input data used in this work, from these three categories. The first row shows sprites from the Human-like Characters dataset, the second one from the Creatures collection, and the last one from the Faces set. It is worth mentioning that the second and third cases are purely experimental, in order to address (respectively) two of the main problems that could be found when generating sprites with DCGANs in a games company: the diversity of available sprite types and the small datasets of a particular kind. All sprites were purchased from *Oryx Design Labs*[3], acquired with the aim of portraying real sprite sets that a small company could have access to.

VI. Results and Discussion

This section describes the results obtained in the three custom datasets, showing some examples generated from them.

1) Human-Like Characters: Figure 7 shows generated sprites after 5000, 10000 and 15000 epochs. After 5000 epochs, there is still a strong presence of noise in the images generated, especially surrounding the main character. Some sprites, however, have developed a rather good representation at this stage (i.e. on the bottom left corner of the upper image).

After 10000 epochs, some good representations have formed along with some which are still mainly noise. There appears to be evidence of two representations of training data being morphed into one in the bottom row. Note that these are new,

[2]All experiments where tun in a PC with a GeoForce GTX 908Ti graphics card, 32GB of RAM and an Intel Core i7 6700k (Skylake) processor.

[3]oryxdesignlab.com

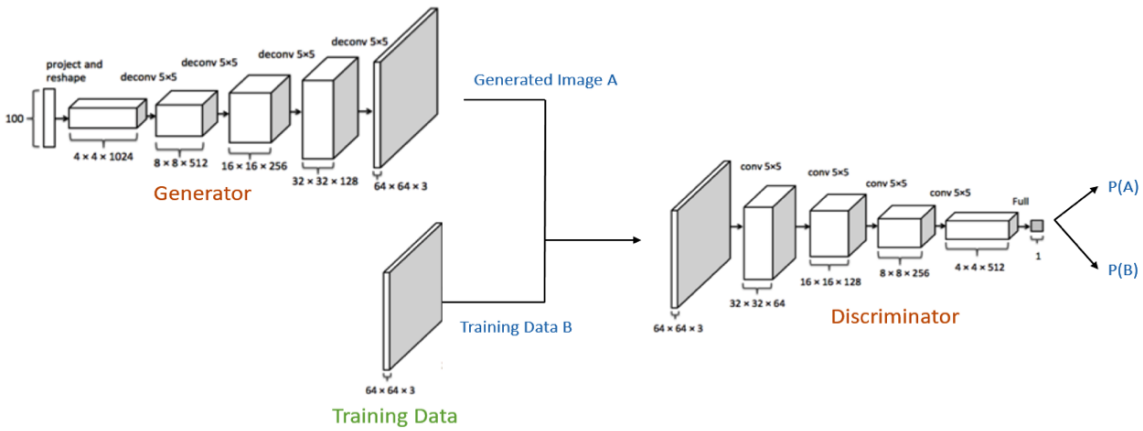

Fig. 5: Representation of a DCGAN.

Fig. 6: Examples of Sprites used as input for the DCGAN.

unique sprites, which have been formed at this stage. Finally, after 15000, much more refined representations have been formed, with most of them having only a small amount of background noise. Training on this dataset to 15000 epochs took around 16 hours. The different sprites shown in each image correspond to the different batches the DCGAN uses to generate them. The amount of noise shown in some of them is quite significant, but some sprites clearly show a decent morph between two of the inputs, generating a unique new image. Figure 8 shows a couple of examples from the Human-like and Faces datasets. In the case of the human-like sprite, it seems like a solider character has been generated with some morphing with a spaceman humanoid.

2) Faces: Figure 9 shows faces generated sprites after 1000, 2500 and 5000 epochs. Due to the smaller quantity of training data, less epochs are required to achieve the same quality of output. 5000 epochs on this dataset took merely 2 hours and 30 minutes. Once again there is some clear evidence of training data representations being morphed to form a new sprite. It is possible to appreciate that the generated sprites for Faces are less satisfying to the eye than the ones from human-like characters. This is most likely due to the small dataset size, meaning that the DCGAN has less information to learn from. After 2500 epochs, clear representations are

Fig. 7: Generated Human-Like Characters after 5000, 10000 and 15000 epochs (up, middle and bottom, respectively).

forming alongside some morphing of representations. There is little diversity from the training data, but the output still shows that with reduced sets it is possible to train the DCGAN to produce morphing representations and unique sprites.

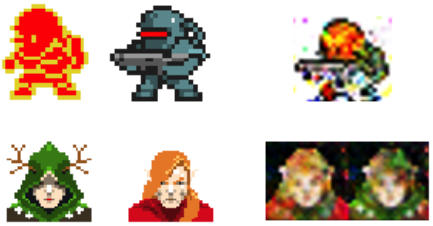

Fig. 8: Samples of new generated sprites for the Human-like (above) and the Faces (below) datasets. The two first sprites of each column on the left are the original images from the input datasets. The images on the right are created by the DCGAN.

Fig. 9: Generated Faces after 1000, 2500 and 5000 epochs (up, middle and bottom, respectively).

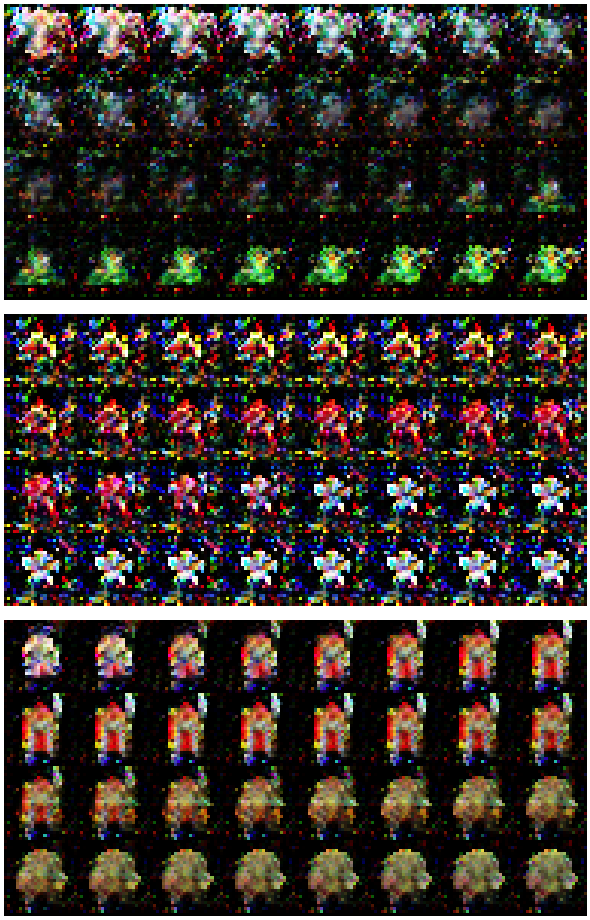

Fig. 10: Generated Creatures after 5000, 10000 and 15000 epochs (up, middle and bottom, respectively).

3) Creatures: Figure 10 shows creatures generated after 5000, 10000 and 15000 epochs. As expected, the output from 5000 and 10000 epochs still have a large amount of background noise and accurate representations have not yet been formed. The output from 15000 epochs begins to show some progression on representation forming, although there appears to be little evidence of correct morphing between the sprites. This could be due to the sheer amount of diversity in the sprites, therefore meaning that when two sprites begin to morph their characteristics (wings, legs, face etc.) collide with one another. This leaves the output sprite as an uneven mismatch of components and very displeasing to the eye (see third row of Figure 10 for evidence of this hypothesis).

4) Loss progression: Figure 11 shows the changes in the discriminator and generator loss through the epochs for the longer experiments of the three datasets. From left to right, it shows Human-like characters, Faces and Creatures. The three plots share some common characteristics, such as the gap between loss of the Generator and loss of the Discriminator.

As can be seen, the Generator loss is always higher, due to the fact the Generator has much more to learn over the Discriminator. This can be seen as the general nature of a DCGAN, since the Generator must learn from scratch whilst the Discriminator has a clear reference to the dataset. All plots have an increasing Generator loss through the experiments, although it is less pronounced in the Faces experiments, probably due to this being the smallest input dataset.

VII. Conclusions and Future Work

We propose a method of utilizing Deep Convolutional Generative Adversarial Networks (DCGAN) for automatic sprite generation and provide evidence that adversarial networks are capable of generating unique new sprites from relatively small datasets. In this work, we have generated sprites for three different categories: Human-like sprites, faces and creatures.

Each one of these categories showcases a particular issue within the problem tackled in this paper. First, the results obtained within the first category show that, even with reduced datasets (of around a thousand sprites), it is possible to generate new unique sprites with the network employed. Secondly, for the Faces dataset, it is clear that the lack of data (this tiny dataset size contains only 36 sprites) signifies more the hazards of generating new clean sprites, but some results are promising at this stage. The third category, creatures, showcases the problem of feeding the network with an non

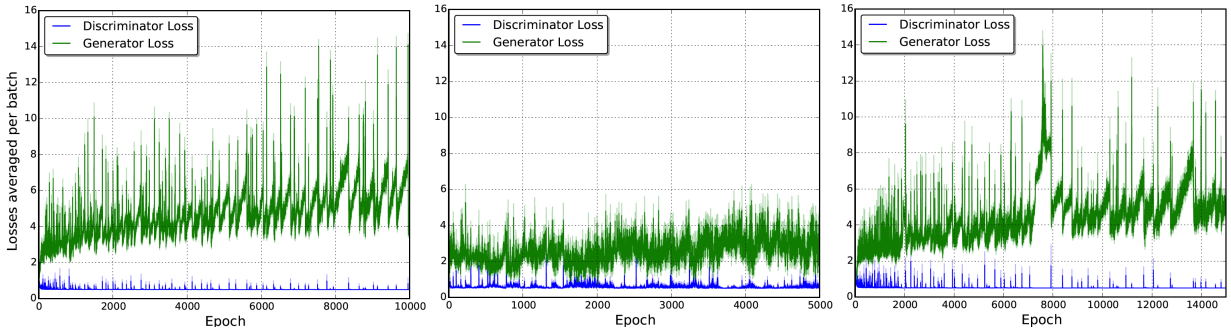

Fig. 11: Discriminator and Generator Loss in three experiments run for each one of the three cases explored. From left to right: Human-Like Characters, Faces and Creatures.

extensive (517 images) but diverse set of sprites, having a reduced number of entities per type: results are extremely noisy and a high number of epochs is required to start achieving interesting images.Finally, there is one problem which was solved throughout all datasets. The uncertainty of establishing the bold, colourful and diverse nature of sprite images. Through outputs observed in all datasets utilized for trraining, there is strong evidence to support the fact DCGAN technology is suitable for accurate sprite generation.

The fact that the lack of training data prevents the network from generating new sprites of good quality is not subject to appeal. It is clear that this implies that in some cases the network simply reproduces training data, and some outputs have an excessive noise. We have shown that increasing the quantity of training data within a dataset can assist in reducing the probability of the network reproducing the input.

However, the lack of training data is in itself why we consider this research interesting. Game companies, especially small and indie members of the industry, may not have the access or the resources needed to purchase or manually generate excessive amounts of sprites to use for training. The final goal of this line of research is to have a reduced set of sprites (say, for instance, soldiers of an army) that serves as a source for new and unique images that fit the theme (that army), automatically generated from the available inputs.

In order to reach that goal, however, further work is needed. The results shown in this paper suggest that the generation of new sprites via DCGANs is possible, but more efficiency is needed in order to produce novelty from reduced datasets. An analysis of the network variables, such as batch sizes and learning rates, could be a first subject of study.

Last but not least, it would be interesting to extend this study to other areas of Procedural Content Generation in games. Examples are the automatic creation of levels, quests, narratives, and even 3 dimensional sprites, which could be generated using Deep Learning techniques.Another interesting advance could be to introduce this research to Deep Visual Analogy Making [2], thus creating a system capable of generating unique sprites with various animations for each sprite.

REFERENCES

[1] A. J. Champandard, "Semantic style transfer and turning two-bit doodles into fine artworks," *arXiv preprint arXiv:1603.01768*, 2016.
[2] S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee, "Deep visual analogy-making," in *Advances in Neural Information Processing Systems*, 2015, pp. 1252–1260.
[3] A. Radford, L. Metz, and S. Chintala, "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks," *arXiv preprint arXiv:1511.06434*, 2015.
[4] C. Li and M. Wand, "Combining markov random fields and convolutional neural networks for image synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2479–2486.
[5] M. Thoma, "A survey of semantic segmentation," *arXiv preprint arXiv:1602.06541*, 2016.
[6] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[7] J. Gauthier, "Conditional generative adversarial nets for convolutional face generation," *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition*, vol. 2014, p. 5.
[8] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Advances in Neural Information Processing Systems*, 2016, pp. 2226–2234.
[9] E. L. Denton, S. Chintala, R. Fergus *et al.*, "Deep generative image models using a laplacian pyramid of adversarial networks," in *Advances in neural information processing systems*, 2015, pp. 1486–1494.
[10] Facebook AI Research, "Neural faces," 2016 (Accessed: 27-03-2017). [Online]. Available: http://carpedm20.github.io/faces/
[11] I. Goodfellow, J. P. Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
[12] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.
[13] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.
[14] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.
[15] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.
[16] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.
[17] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[18] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3730–3738.
[19] F. Yu, A. Seff, Y. Zhang, S. Song, T. Funkhouser, and J. Xiao, "Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop," *arXiv preprint arXiv:1506.03365*, 2015.