Advanced Technologies Report - Asset Generation

Ricardo Heath
15000805

University of the West of England

July 17, 2019

The task was to use machine learning to generate assets for a game using frameworks available in Python. The solution was to use a combination of a conditional GAN and a deep convolutional GAN to create a conditional deep convolutional GAN using the Keras library which could generate textures, and then creating supporting scripts to create a minecraft texture pack.

1 Introduction

This project sets out to apply labels to a DCGAN to generate specific textures that are based on existing minecraft textures. This is done by modifying the layers of the generator and discriminator of the DCGAN to take categorical label data as input. The generator should then be able to generate any learnt texture that can be found in a minecraft texture pack in order to compose a new one entirely from scratch.

2 Related Work

A new framework for generative models via an adversarial process of two networks was proposed by Goodfellow et al., 2014. This framework is known as Generative Adversarial Network, or GAN. This features semi-supervised learning, improving performance of classifiers with limited label data and was used successfully on the MNIST dataset.

This work was followed up on by Mirza and Osindero, 2014 which proposed a modification that makes the generation conditional based on labelled inputs fed to the generator and discriminator. The conditional GAN (CGAN) successfully generates MNIST digits by using the digit labels.

A new form of GAN was introduced by Radford, Metz, and Chintala, 2015 which is based on the work done with convolutional networks. The proposed architecture is a Deep Convolutional GAN (DCGAN) which is capable of unsupervised learning. DCGANs were not as successful at unsupervised learning as convolutional networks.

A network similar to the one in this prototype is described by Gauthier, 2014 which applies conditional labelling to a GAN that uses convolutional layers to generate faces. Their model demonstrated that conditional information could be used to deterministically control the output of their generator which used convolutional layers.

The work done by these authors forms the basis of the CDCGAN and its feasibility as a working model for generating conditional images in an unsupervised learning environment.

3 Method

3.1 GAN

Generative adversarial networks are a type of neural network architecture which are composed of two independent networks that are pitted against each other. These networks are the discriminator and the generator. The generator produces some output, which is marked as fake. The discriminator takes the generators output as input and determines if it's real or fake.

If the discriminator classifies the generators output as fake then the generator has made a mistake and tries to compensate for it in future tests, if it classifies it as real then the discriminator has made a mistake

and becomes the one to compensate. This back and forth is what causes the networks to fight each other and improve over time, attempting to outdo the other.

A combined model of the discriminator and generator is created in order to provide a feedback loop to the generator for training.

3.2 DCGAN

Deep convolutional GAN's build on standard GAN's by applying convolutional filters to the layers of the discriminator and generator. These convolutional filters are better able to discern features in the data.

3.3 CGAN

Conditional GAN's build on standard GAN's by supplying an additional input to the generator and discriminator. This input is used to categorise the other input with a label. Using this labelling technique allows for specific features to be detected independently of others.

3.4 CDCGAN

A CDCGAN was created by applying label inputs to the generator for the DCGAN. Combining these two GAN's allows for specific textures to be generated with a higher quality than a traditional CGAN.

3.5 Layers

Neural networks use layers which are a collection of nodes that work together in the network at a specified level. The input layer is the raw data of the network, it is not modified in this layer. The hidden layers are attempting to pick up certain features of the previous layer and learn from them, with their weights being adjusted based on the loss function. The output layer contains the final result of the hidden layers. The output layer of the generator will be a $(64, 64, 3)$ image while the output of the discriminator will be a single node between 0 and 1.

3.5.1 Convolution

Convolutional filters are used to transform inputs in to a smaller and more specific set of outputs. It is an element matrix multiplication operation in which the image is multiplied by the kernel or filter, resulting in the transformed output. A stride can be used to create gaps in the filter, as it will step over a certain number of nodes. A convolutional layer can be used for deconvolutions by varying the filter size compared to its inputs. The output of a convolutional layer is smaller than that of its input, while the output of a deconvolution is larger.

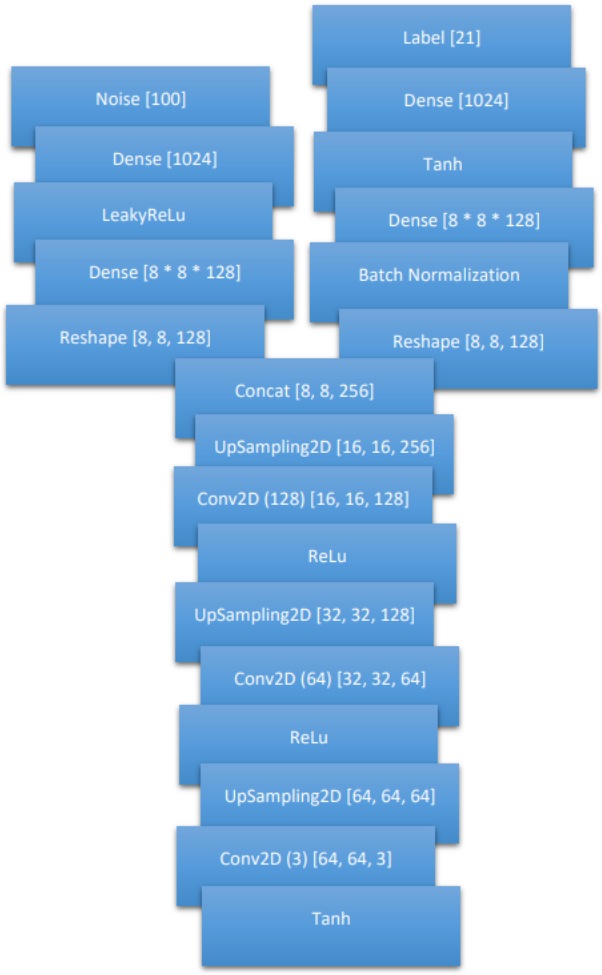

Figure 1: *The layers of the generator model*

3.5.2 Upsampling

Upsampling repeats the elements of its input such that it will be scaled to a larger output. The scaling output can differ based on the interpolation used, such as nearest or bilinear.

3.5.3 Max Pooling

Max pooling is a layer primarily found in DCGAN's inbetween the convolutional layers. It applies a filter to the layers input which takes the max of a region. The region will depend on the given size of the filter and its stride. This results in a downsampled output.

3.6 Activation Functions

Activation functions are used to process the outputs of a layer and map them to specific values. Activation functions are either linear or non-linear. Non-linear functions are used more often as they have specific ranges of values.

3.6.1 Sigmoid

Sigmoid activation maps inputs to a range between 0 and 1. This is useful when predicting probabilities and as such is found as the last layer in discriminators.

$$S(x) = \frac{e^x}{e^x + 1} \tag{1}$$

3.6.2 Tanh

Tanh activation is similar to sigmoid in shape and maps inputs to a range between -1 and 1. The advantage of Tanh compared to sigmoid is in mapping negative inputs to a negative value while ensuring inputs near 0 will remain near 0.

$$T(x) = \frac{e^2x - 1}{e^2x + 1} \tag{2}$$

3.6.3 ReLu

Rectified Linear Unit activation maps inputs to a range between 0 and infinity. Negative inputs are discarded from further layers, which can sometimes be desirable so as to minimize node activations though Dying ReLu is an issue where nodes can always be negative and thus are never trained.

$$R(x) = \begin{bmatrix} x & x > 0 \\ 0 & x <= 0 \end{bmatrix} \tag{3}$$

Leaky ReLu was created to solve this by allowing for negative inputs to be activated using the gradient α with a typical value of 0.01. This alpha value is multiplied with the negative input resulting in small negative outputs.

$$R(x) = \begin{bmatrix} x & x > 0 \\ \alpha x & x <= 0 \end{bmatrix} \tag{4}$$

3.7 Generator

The layers of the generator are shown in Figure 1. Conv2D layers use a kernel size of 5. The layers handling label input use tanh activation as it's better for classification. Upsampling uses factors of 2. 3 Cycles of UpSampling, Conv2D, and Activation are applied in order to pick up features and create the correct output shape for the image. Changing the filter size of the convolutions would result in a different output image resolution, or the number of cycles can be changed to change the amount of detail picked up in an image with more or less detail.

The layers handling the label input ends in batch normalization before reshaping. This involves normalizing the inputs by modifying its activation so that higher learning rates can be used due to the impact of outliers being minimized. Aspects of the network which were unable to train may now have a chance to. The main reason to use batch normalization here is to aid in the over-fitting issue as it adds variance to the activations of the label input. This helps to prevent mode collapse as the labels aren't used as the only basis for generating the output image. Mode collapse can happen when the input noise becomes insignificant to the output result.

The output layer of the generator uses Tanh activation as the pixel values of an image range between -1 and 1.

3.8 Discriminator

The layers of the discriminator are shown in 2. The input is an image where each node has a value between -1 and 1 regardless of its authenticity in order to prevent the discriminator from overfitting through learning the differences in the range of node values.

Conv2D layers use a kernel size of 5 and perform deconvolutions. MaxPooling2D applies a downsampling with a factor of 2 after each deconvolution. The filter size of the decovolutions matches those of the generator so as to try and detect similar features in the two networks.

The discriminators single sigmoid output is used to determine the authenticity of its input image.

3.9 Training

The discriminator is trained by itself in an epoch and then the combined model is trained with discriminator training disabled.

3.10 Backpropagation

Forward propagation involves the input data of a model being passed throughout all the layers until reaching

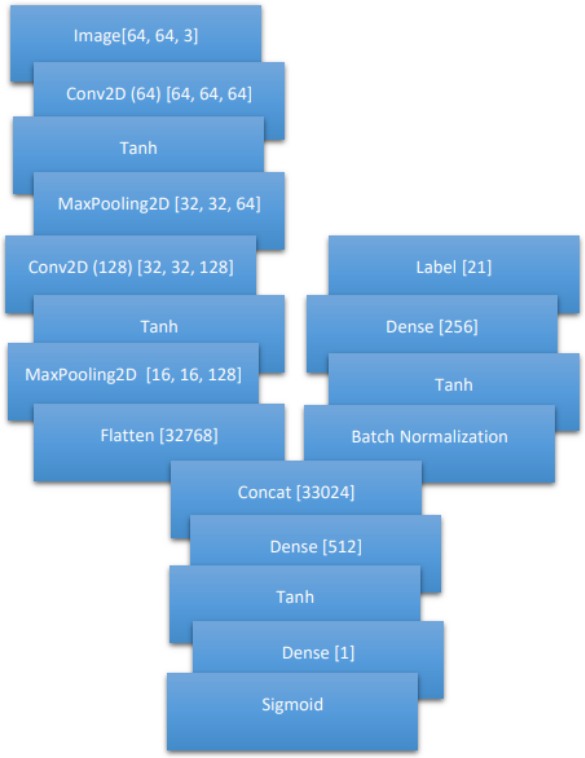

Figure 2: *The layers of the discriminator model*

the output layer, The output layer starts off innacurate due to the weights of previous nodes being incorrect as they lack training. The output layers prediction can be compared to the ground truth to measure the error of that propagation.

Forward propagation forms half of backpropagation. Once the error is calculated for the forward propagation the error is propagated backwards through the layers of the network so that the weights of every node are modified by the optimizer to minimize that error.

3.11 Optimizer

Optimizers are used to train neural networks faster by changing the weights and biases of the nodes in the network. The loss of a network is used by the optimizer to change the weights in a better direction, attempting to make continual progress until a minima is reached. Gradient descent is a simple optimizer in Keras that implements the gradient descent algorithm that calculates what small changes in weight does to the loss function, a general version of which can be seen in Figure 4, and adjusts those weights based on a gradient. Local minima's are undesirable due to not reaching the best possible node weights. Higher learning rates allow the optimizer to escape them.

The Adam optimizer is used for the discriminator and combined model with a learning rate of 0.0002 and a beta 1 of 0.5. Adam uses the momentum of past gradients by using small amounts of them and adding

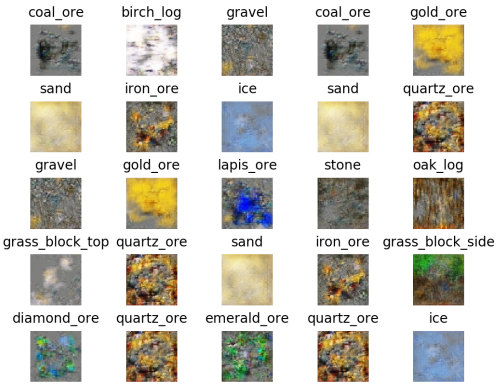

Figure 3: *A sample of the generators output at epoch 1600, with mode collapse*

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$

Figure 4: *General loss function*

them to the current gradient, as well as using previous gradients to calculate the current gradient.

3.11.1 Adversarial Ground Truths

Ground truths are used in training to allow the discriminator and generator know if they succeeded or failed in fooling the other. The ground truths for real data is 1 and fake data 0.

3.11.2 Data

Training data is loaded in batches every epoch using the Keras generator flow_from_directory and with categorical labels. The pixel values of the images are normalized to be between -1 and 1.

3.11.3 Noise

Spherical noise is sampled on every epoch, once for each training image. Spherical noise is used instead of linear as described by White, 2016 for producing sharper samples.

$$H_p(q) = -1\frac{1}{N}\sum_{i=1}^{N} y_i \cdot \log(p(y_i)) + (1-y_i) \cdot log(1-p(y_i))$$

Figure 5: *Binary cross entropy loss function*

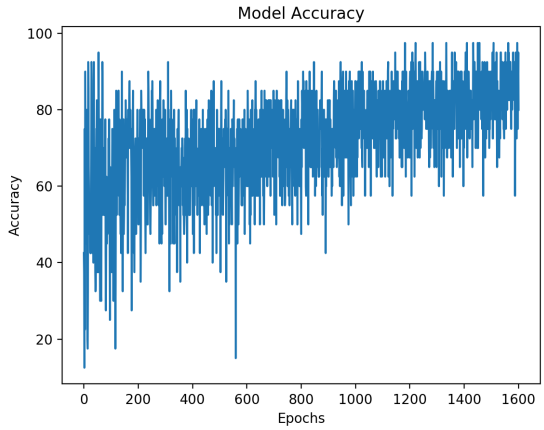

Figure 6: *The discriminators accuracy over 1600 epochs*

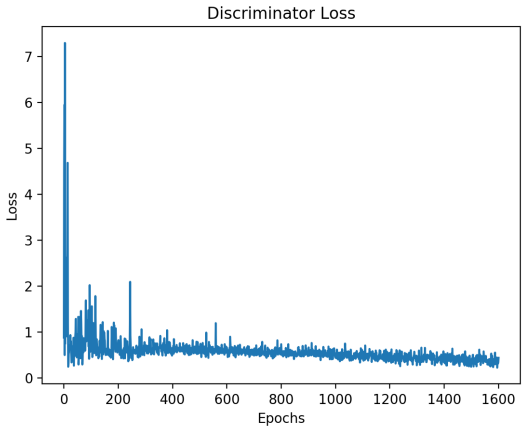

Figure 7: *The discriminators loss over 1600 epochs*

3.11.4 Loss

Binary cross entropy is used as the loss function, as shown in Figure 5, for this CDCGAN and it works to condense the entire set of weights of the network in to a single value such that improvements in that value correspond to a better model. That value is the loss, lower being better. The loss of the discriminators output when dealing with training images and generated images is averaged together to produce the loss shown in Figure 7.

3.12 Saving & Loading

The generator model is saved and loaded using Keras. Compilation must be turned off when loading and the model cannot be trained further.

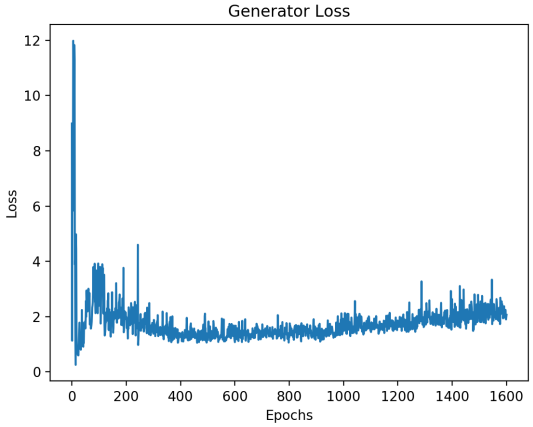

Figure 8: *The generators loss over 1600 epochs*

3.13 Image Output

Images of the training data are output for sanity checking alongside the generators output using matplotlib.

3.14 Texture Packs

A texture per label is generated from the CDCGAN and saved to the correct file structure for minecraft texture packs.

4 Evaluation

4.1 Discriminator Accuracy

The discriminator's ability to detect authentic images is improving over time as can be seen in Figure 6. The accuracy categorical and is calculated using the mean accuracy rate across all predictions of the discriminator for that epoch. The increase in accuracy could signify that the generator is becoming predictable and is being beaten easily but more data is required, as that would be shown with an accuracy nearing close to 100% and with less variance.

4.2 Loss

The discriminator's loss is trending down over time as shown in Figure 7. This trend matches the inverse of the accuracy trend, indicating that the model is continuing to learn and is working correctly. The generators loss shown in Figure 8 is trending upwards, which is expected as the trend should be similar to the inverse of the discriminator. The amount of upward trend is unexpected compared to the amount of downward trend for the discriminator however. The model has visible mode collapse in its output, it's possible that the generator is ignoring the noise input relative to the label input and won't improve as a result, leading to an ever-diverging loss between the two networks when in a working network they should eventually stabilize.

5 Conclusion

Applying the labelling of a CGAN to a DCGAN allowed for generating specific textures with the advantages of the convolutional filters that a DCGAN is known for. Mode collapse is present in the model as shown in Figure 3 which could be prevented in future work by using dropout, label smoothing or applying gaussian noise to the discriminators inputs to prevent overfitting. The combination of these GANs is a simplistic alteration which resulted in good output. The generated textures fit suitably within the minecraft aesthetic despite the mode collapse of the model.

Bibliography

Gauthier, Jon (2014). “Conditional generative adversarial nets for convolutional face generation”. In: *Class Project for Stanford CS231N: Convolutional Neural Networks for Visual Recognition, Winter semester* 2014.5, p. 2.

Goodfellow, Ian et al. (2014). “Generative adversarial nets”. In: *Advances in neural information processing systems*, pp. 2672–2680.

Mirza, Mehdi and Simon Osindero (2014). “Conditional generative adversarial nets”. In: *arXiv preprint arXiv:1411.1784*.

Radford, Alec, Luke Metz, and Soumith Chintala (2015). “Unsupervised representation learning with deep convolutional generative adversarial networks”. In: *arXiv preprint arXiv:1511.06434*.

White, Tom (2016). “Sampling generative networks”. In: *arXiv preprint arXiv:1609.04468*.