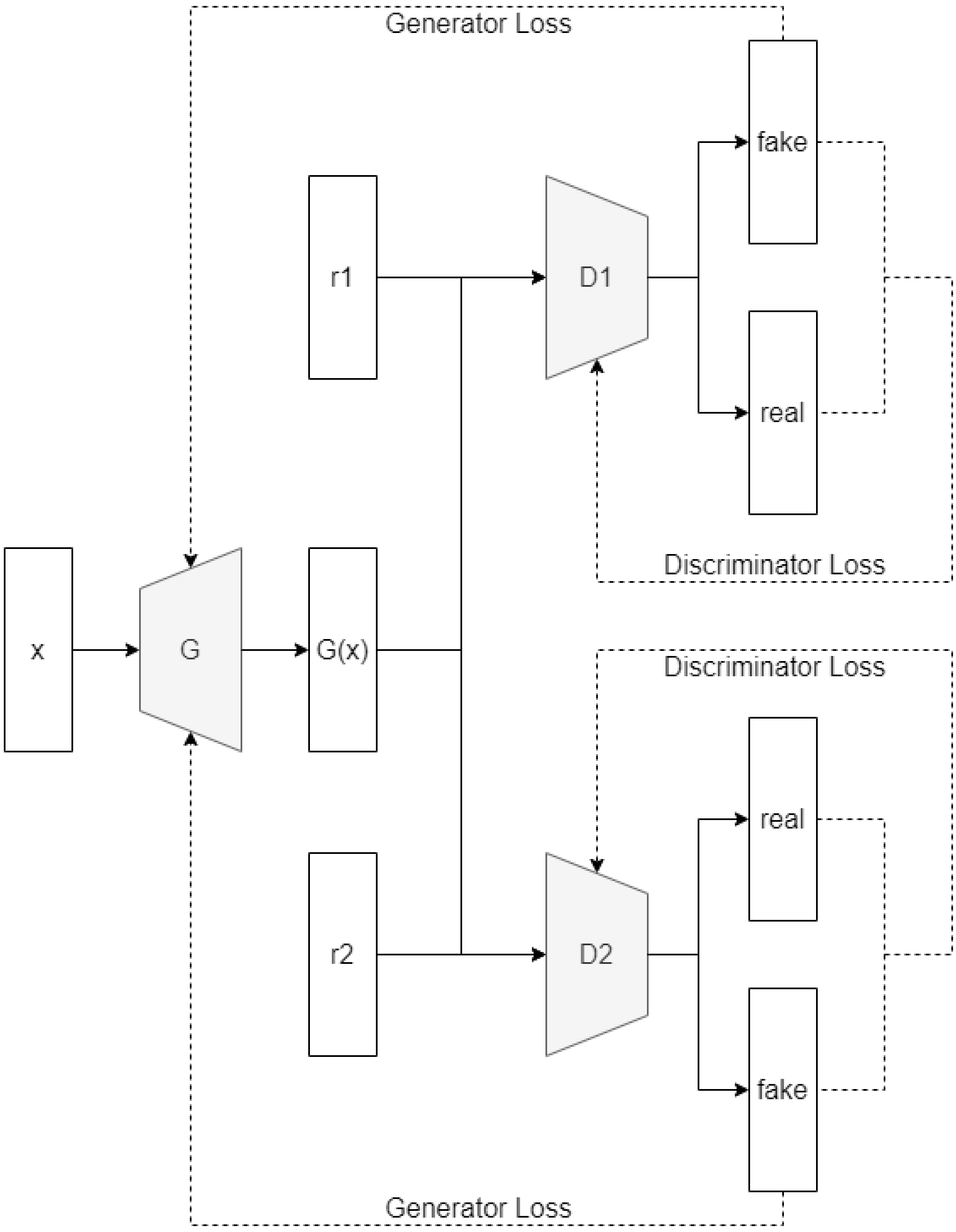

Figure 1. Diagram of the final generative model architecture.

Spectral normalisation has been used to train a generative adversarial network that creates images of winged horses that look like a Pegasus from the CIFAR-10 dataset. As explained in [A], SN-GANs can control the Lipschitz constant of the discriminator function by constraining the spectral norm of each convolution layer. The value of spectral norm used to regularise each layer W^l is set to the largest singular weight $\sigma(W^l)$. As applying singular value decomposition in each step becomes computationally expensive, a very computationally cheap power iteration method is used to estimate $\sigma(W^l)$. This normalisation of weights stabilises the training of the discriminator and helps mitigate the degree of mode collapse.

As shown in figure 1, the architecture implemented consists of one generator and two discriminators. When importing the CIFAR-10 dataset into the program, two training sets are formed by filtering out all bird and horse tags. The network begins by training the discriminator for horses (D1). 64 images from the training set of horses and 64 fake images of a Pegasus taken from Gaussian noise are then fed into the discriminator. The discriminator downsamples the images using four convolution layers, and outputs a value representing the realness of the input. Upon calculating the discriminator's binary cross entropy losses, its weights are updated through backpropagation. Next, the discriminator for birds (D2) is trained using the same process, replacing the 64 real images of horses with birds. To encourage randomness in the fake images and reduce the likelihood of the GAN getting stuck, a dropout rate of 0.2 has been added to the discriminator. Label smoothing had additionally been applied to the discriminator's real image inputs to tackle overconfidence. To train the generator, four transposed convolution layers are used to upscale noise taken from a Gaussian distribution. The fake output image is then passed into both discriminators (D1 and D2) and two BCE loss functions are computed. The generator loss function combines the two results and updates its weights through backpropagation based on this value.

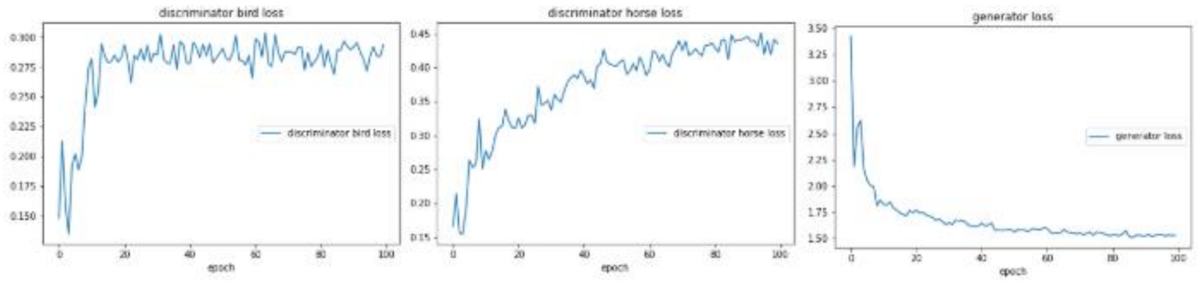

Figure 2. Example loss results from the final solution

Having run the solution over 100 epochs, the discriminator functions looked very stable (figure 2), with the horse discriminator approaching 0.5 and the bird discriminator approaching 0.3. This suggests that the discriminator was somewhat accurate at distinguishing between bird and Pegasus images but struggled to distinguish the difference between horse and Pegasus images.

To improve the network, the generator loss function was inspected. Initially, the BCE loss calculated by both discriminators had an equal 0.5 weighting on the overall loss function. As shown in figure 3, the algorithm was tested on a variety of different weights for the loss function. For all the tests, similar curves were output to the curves in figure 2 and as expected, the greater the weighting towards one animal i.e. horses, the better the discriminator was at distinguishing the other animal i.e. birds. Consequently, the final weighting was decided on the best visual output, which appeared to be a ratio $0.7 : 0.3$. As a Pegasus is arguably more horse-like than bird-like, the weighting towards horses feels justified.

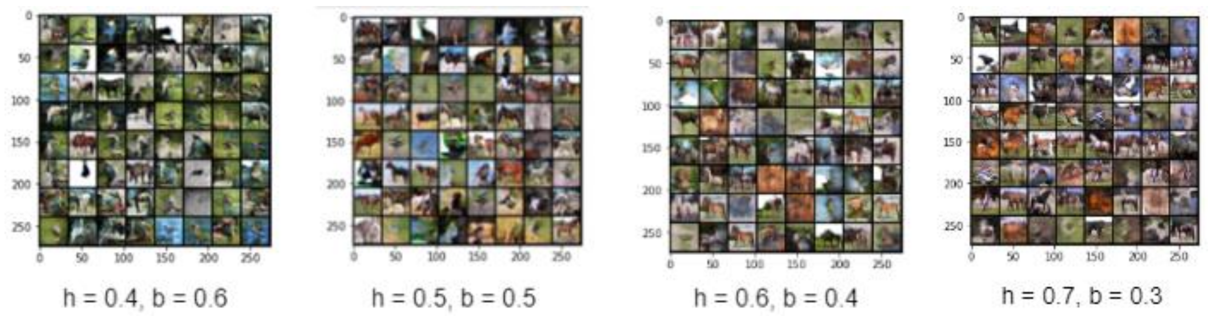

Figure 3. Images of a Pegasus after 100 epochs based on different loss ratios

With the intention to reduce the accuracy of the bird discriminator, the number discriminator iterations over the number of generator iterations was reduced from five times more than the generator, to three times more and four times more (figure 4). Disappointingly, this made the training of the bird discriminator far more unstable and had little effect overall and hence were left at the default five.

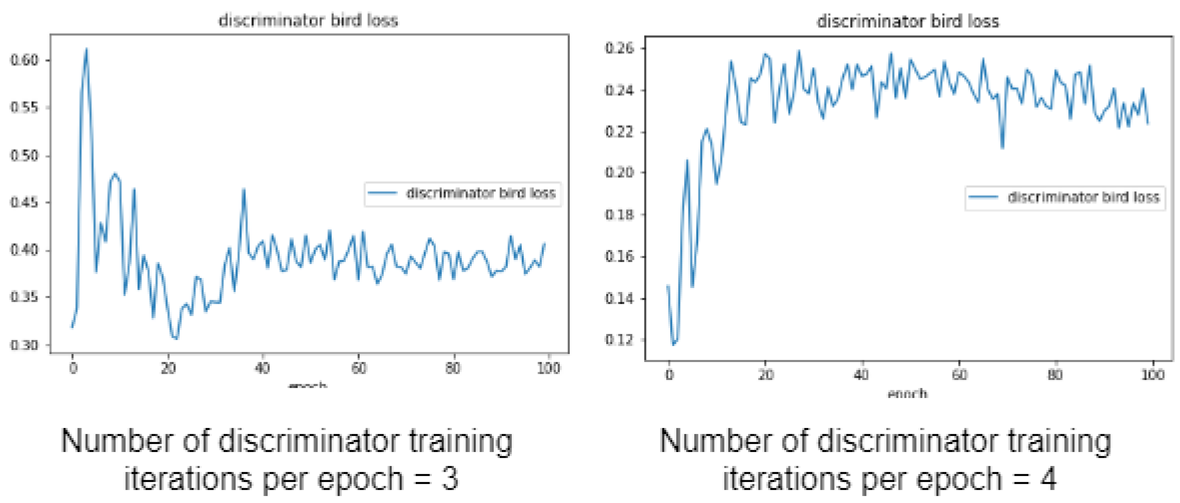

Figure 4. The bird discriminator's loss per epoch

Looking to improve the generator model, different activation functions were considered. Activation functions are mathematical 'gates' that model the firing rate of neurons. As ReLU functions can be implemented by theresholding a matrix of activations at zero, they are more computationally efficient than functions involving more expensive operations i.e. sigmoid, tanh. ReLU functions were therefore initially used as the activation function for all layers in generator. However, as shown in figure 5, using ReLU and LeakyReLU in the final layer of the generator left many blank spots after 100 epochs in the images of a Pegasus. As a result, Sigmoid and Tanh activation functions were experimented with in the final layer of the generator model. From the outputs shown in figure 5, these functions provided better results. Hence, Tanh was selected as the final activation function.

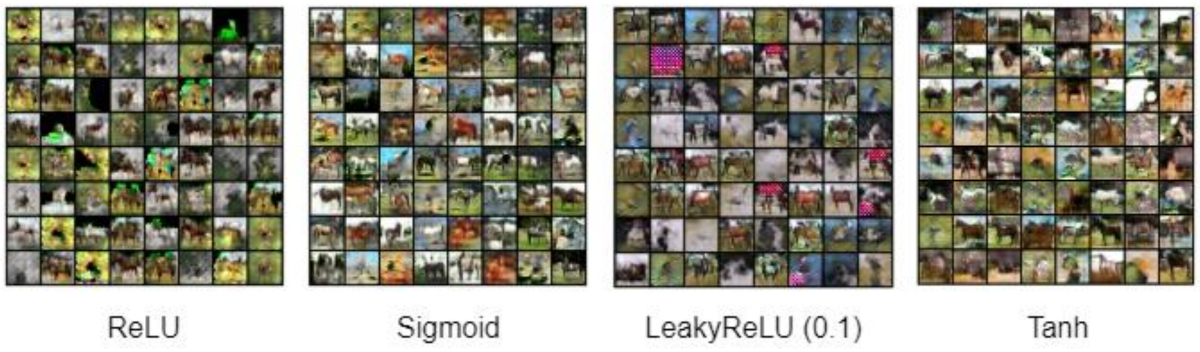

Figure 5. Results after 100 epochs using different activation functions in the final layer of the generator model.

Note: Sections of code taken from SN-GAN found in lecture 5 - https://colab.research.google.com/gist/cwkx/f4b49cd3efc0e624bc22c89c90921931/spectral-norm-gan.ipynb

References:

1. Miyato, T., Kataoka, T., Koyama, M. and Yoshida, Y., 2018. Spectral normalization for generative adversarial networks. arXiv preprint arXiv:1802.05957.