COMP3419 Assignment Option 3 - PokéGAN

Hamish Ivison

Introduction

In this report, we investigate Wasserstein Generative Adversarial Networks (WGANs) in relation to pokémon image generation, researching the effects of different architectures and data augmentation on their performance. We also briefly research how recent improvements to WGAN training improve on these results, focussing on the Gradient Penalty WGAN. We find that more data is always better, with data augmentation significantly improving results. We also find that using deconvolution layers results in sub-optimal performance for baseline WGANs, and that the Gradient Penalty WGAN greatly stabilises and improves WGAN training and results.

For the assignment, we improved WGAN performance in two main ways, using a new generator architecture that greatly stabilised performance, and using the WGAN-GP algorithm over the WGAN algorithm, which provided far better results.

What are GANs?

Generative Adversarial Networks (GANs) are an innovative network made up of two neural nets, introduced in (Goodfellow et al. 2014). By training on a dataset of images, GANs learn to mimic the dataset and generate similar images[1]. The network consists of two neural networks: a generator, which learns to generate fake versions of the images fed into the network, and a discriminator, which learns to tell apart the real images and the generated images. In training both networks, with the discriminator's output being used to train the generator, the generator slowly learns to 'fool' the discriminator, while the discriminator still learns to distinguish the fake images of the improving generator. Hence, the discriminator and generator play a two-player minimax game, with the discriminator learning more and more features of the input dataset as time goes on, and the generator learning to replicate these features better as time goes

[1] Of course, datasets other than images could be applied too.

on. It has been shown[2] that this results in the loss function of a GAN being equivalent to minimising the Jensen-Shannon divergence:

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r||\mathbb{P}_m) + KL(\mathbb{P}_g||\mathbb{P}_m) \tag{1}$$

where $\mathbb{P}_r$ is the real probability distribution (that is, the probability distribution of the input set of images) and $\mathbb{P}_g$ is the generated image's probability distribution[3]. KL is the Kullback-Leibler divergence, and $\mathbb{P}_m$ is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$. As the Jensen-Shannon divergence is always non-negative and only zero when the two distributions are equal, minimising it is equivalent to the generator learning the real distribution over the image space - that is, learning how to produce images exactly similar to the real set of images. More intuitively, this distance is the difference between the probability distribution of images produced by the generator, and the probability distribution of the images within the input dataset. Therefore, by minimising this difference the network gets closer to representing the true distribution of images in the input dataset, thereby producing images closer to the input dataset.

This has achieved fantastic results - however, GANs can be very unstable, with parameters requiring fine-tuning to produce reasonable results. Two larger issues are mode collapse, where a generator learns to produce samples with low variety, due to it not fully learning a complex multimodal distribution[4], and the vanishing gradient problem, where the discriminator learns too well and the generator can not learn from it. Therefore, traditional GANs require that the generator and discriminator be trained slowly and at similar rates - lest the discriminator become too powerful and the generator fails to learn from it. This is obviously undesirable, and (Arjovsky, Chintala, and Bottou 2017) introduces a solution to this problem: the Wasserstein distance metric.

WGANs

In this report, we examine this particular recent improvement on traditional GANs, as proposed in (Arjovsky, Chintala, and Bottou 2017): the Wasserstein Generative Adversarial Network (WGAN), which improves the performance and stability of GANs through its use of a novel loss function - the Wasserstein distance. The use of this new loss function allows the 'discriminator' of a GAN to become a 'critic', giving a score to the generated images - the lower the score, the more obviously fake the image, and the better the score, the less fake the image seems. This provides a way for the generator to improve even if it cannot provide images that can fool the discriminator, thereby eliminating the need to

[2] see (Goodfellow et al. 2014) for more details.

[3] This is a highly simplified explanation- please see (Arjovsky, Chintala, and Bottou 2017) for a more technical treatment.

[4] see (Nibali 2017) for a more in-depth explanation.

keep the discriminator weak enough that the generator can fool it. In theory, this removes the risk of mode collapse and vanishing gradients.

The Wasserstein Distance

The core of the WGAN is its application of a new loss metric: the Wasserstein distance. This distance, also known as the "earth mover's distance", can be expressed as:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}[||x - y||] \tag{2}$$

here $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively $\mathbb{P}_r$ (the real distribution of input images) and $\mathbb{P}_g$ (the generator's distribution). $\mathbb{E}$ simply means we take the average of the results from an infinite number of samples of (x,y) from the given joint distribution γ.

Intuitively, if we imagine both probability distributions $\mathbb{P}_r$ and $\mathbb{P}_g$ as histograms, this distance is the amount of 'earth' (or 'mass') required to move to make $\mathbb{P}_r$ resemble $\mathbb{P}_g$, with an optimal movement plan. This serves as the cost function for the WGAN, providing a meaningful representation of the divergence even in difficult scenarios, thus providing better performance than the Jensen-Shannon distance[5]. By assuming K-lipschitz continuity for some K, this cost function can be transformed into a more usable form for neural network training, providing a suitable cost function for the network[6].

In order to use this new cost function, we transform the discriminator in a GAN into an estimator of the Wasserstein distance. This has the effect of turning the 'discriminator' into a 'critic', the Wasserstein distance of each image serving as a score, where lower scores are better. When the generator is worse, its distribution will be further from the real distribution, and so the Wasserstein distance will be large, while if the generator is closer to the real distribution, the distance will be smaller. This prevents the vanishing gradient and mode collapse problems present in prior GANs, as now the generator always gains useful information from the critic regardless of how far off it is (assuming the critic has been trained to optimality). Hence, we actually train the critic in a WGAN much more at the start of training in order to ensure it provides meaningful data to the generator from the beginning, unlike previous GAN systems (which tried to keep the discriminator weak so it could be partially fooled by the generator, preventing the vanishing gradient problem). By using the Wasserstein distance as the cost function instead of the Jensen-Shannon divergence, WGANs completely remove the chance of model collapse or vanishing gradients occurring, and remove the

[5] Arjovsky, Chintala, and Bottou (2017), p4-6.

[6] If you wish to know more about the mathematics, please read (Arjovsky, Chintala, and Bottou 2017) for details. we also have found (Weng 2017), (Hui 2018), (Lai 2017), and (Yurasov 2017) quite useful explanations.

need for the discriminator and generator to be trained at a roughly equal rate, further stabilising GAN performance. Furthermore, it provides a nice way to correlate loss with image quality - the larger the loss of the WGAN's critic, the worse the generated image is.

Weight Clipping

However, there is one large issue with using the Wasserstein distance: in order to use the metric, K-lipschitz continuity must be upheld, meaning the weights of the critic have to lie in a compact space. In order to ensure this, we simply clip the weights to be within a fixed range (usually [-0.01, 0.01]) after every gradient update. This is noted by the creators of the WGAN to be an awful way to enforce the lipschitz condition, and new networks, such as WGAN-GP, have managed to find better solutions to this problem[7]. However, for this assignment, we chose to focus on the normal WGAN setup, and so used weight clipping to ensure lipschitz continuity. Without this, the discriminator ceases to be a good estimator of the Wasserstein distance and so does not provide useful information to the generator.

WGAN-GP

In order to improve results, an implementation of the WGAN-GP (gradient penalty), as introduced in (Gulrajani et al. 2017), was used. As the WGAN is very sensitive to the weight clipping parameter (and the creators of the WGAN themselves admit it is an inadequate method for enforcing the lipschitz condition), the WGAN-GP improves this by removing the need for weight clipping entirely. Instead, the WGAN-GP punishes the GAN if its gradient norm moves away from 1. This is because if the gradient norm of the model is 1 everywhere, then it is 1-lipschitz. Hence, by enforcing this penalty the WGAN-GP ensures the lipschitz condition without requiring weight clipping.

This results in far improved performance - a tricky parameter (the weight clipping) is removed, and this new WGAN-GP is also usable with the Adam optimiser[8], providing improved results in fewer iterations. Hence, an implementation of this model was used to boost the performance of the WGAN.

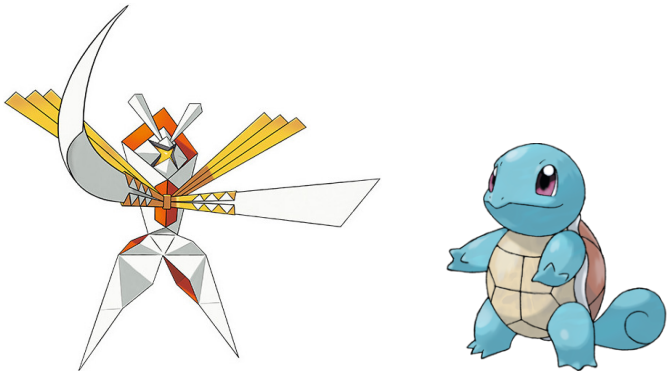
Pokémon and WGANs

Pokémon provide a fantastic dataset for investigating the behaviour of GANs: large datasets are easy to find, and the pokémon themselves are quite varied,

[7] see Gulrajani et al. (2017)

[8] An algorithm for stochastic gradient-based optimisation, providing better performance than RMSprop, the optimiser used for the standard WGAN. See (Kingma and Ba 2014) for details.

especially with the introduction of ultra beasts in Pokémon Sun and Moon, which break many traditional pokémon design norms. Hence, we used a dataset collated from veekun (a great resource for pokémon images) for testing.



(a) Kartana (b) Squirtle

Figure 1: Kartana and Squirtle, examples of ultra beast and traditional pokémon design respectively

In particular, the dataset of Sugimori art was used - watercolour drawings of every pokémon and every pokémon form, by Ken Sugimori (or drawn in his style). This artwork is the official artwork of pokémon, and so serves as the best dataset of correct pokemon images, as opposed to mixing art styles from Pokémon spin-off games or sprites.

Our WGAN Implementation

For this assignment, we examined two main additions to a base WGAN: using upsampling combined with convolutional layers to reduce checkerboarding in the generator, and using residual layers in the discriminator. This was based on the work done in (Odena, Dumoulin, and Olah 2016) and (He et al. 2015) respectively, though the critic architecture used is not a full residual net. Rather, a shallower version of the architecture was used. A baseline WGAN, based on the design described in the assignment specification, was also implemented. Both implementations are discussed below. The improved generator and baseline discriminator architectures were also used in our WGAN-GP experiment.

Method

Data Augmentation

Despite the thousands of pokémon images, this is still not enough to train a WGAN - rather, successful GANs need millions and millions of images to properly train. Hence, we use data augmentation techniques to pad out our dataset. Several data augmentation techniques have been applied:

- random cropping
- random transforms (padding random edges and then cropping back to the usual size)
- horizontal flips
- vertical flips

Applying these not only augments our dataset with more images, but also will ideally result in WGAN learning features of pokémon invariant of translation or rotation, since more pokémon images feature the pokémon facing left or right in the centre of a white background. By randomly shifting the pokémon and flipping it, the WGAN will have to learn how to recognise and produce such features even when they are flipped or moved around, rather than learning them in a particular place in a particular orientation. This serves to make the WGAN more robust on new data, helping the network generalise better about features (as opposed to 'overfitting' on the place and orientation of the pokeémon).

Data augmentation was implemented through the Keras *ImageDataGenerator*[9] class, which allows the user to set what augmentations they wish, then apply them randomly to a dataset stored within a folder. This creates a generator which produces batches of image data with the augmentations randomly applied. To demonstrate this, run `data_prep.py` from our codebase. This will generate some images with the data augmentations randomly applied.

The base WGAN and WGAN-GP used the following data augmentation parameters:

```
{
    rescale=1./255,
    zoom_range=0.2,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
}
```

The rescale was required to bring image colour values to a [0,1] range. Zoom refers to cropping (selecting random sections of the image and zooming into

[9] see the keras documentation for more details.

them), while shifting refers to random padding. Horizontal flip and vertical flip refer to flipping the images horizontally and vertically respectively.

For the data augmentation experiments, these parameters were removed or tinkered with. More details on this can be found below, within the experiments section.

Generator

Two generator architectures were used: *(i)* a base architecture utilising repeated layers of deconvolution, batch normalisation, and activation layers, and *(ii)* an augmented architecture consisting of repeated convolution, batch normalisation, activation, and upsampling layers.

The augmentation was modelled after work done in (Odena, Dumoulin, and Olah 2016), which suggested that deconvolution layers caused checkerboard artifacts due to uneven patterning in architectures, and suggesting using convolution and upsampling to avoid this issue. This is due to the deconvolutional layers operating on overlapping image segments, thereby causing the generator to 'care' more about certain pixels. This overlap occurs in a checkerboard fashion, thus causing the checkboard artifacts. Hence, applying upsampling and then convolution still allows recognition and generation of features, while avoiding this overlapping issue and thus reducing checkerboard artifacts.

In order to test the generator architectures, an image segmentation task was performed using the AD20K dataset. Results are given alongside the diagrams below.

Base Generator Architecture

Below is a diagram of the baseline generator architecture. Rather than show the full architecture, we instead show the first few layers. 6 deconvolutional blocks were used in the architecture, along with input and output layers.

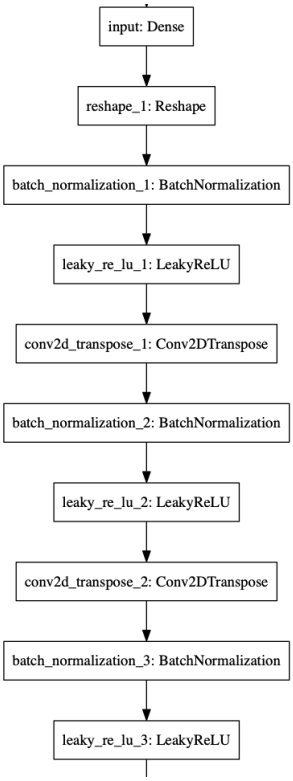


Figure 2: Baseline generator architecture

This generator architecture achieved 60.68% accuracy on the AD20K dataset without parameter tuning and paired with a simple encoder structure[10], after 50 epochs of training. This is not the best result, but clearly shows the generator has some baseline capacity for segmentation - otherwise its results would be far worse. Furthermore, the generator is clearly segmenting the image to some degree on closer inspection, as seen below. Furthermore, the architecture clearly overfitted to some degree, as there was a substantial gap between test and training accuracy, suggesting that with more tuning the model could have better results. The lower score makes sense - image segmentation is a complex task, often requiring quite deep architectures, and so it stands to reason that our simpler generator architectures would not necessarily perform as well without fine-tuning. Memory limitations were also an issue - we simply could not fit larger generator architectures onto our GPU, and so had to work with simpler architectures. This was not a focus of the report, as it merely served as a sanity check that our architectures could work, but we include an example of the segmentation performed below.

[10]See our codebase for this encoder - it is simply a few convolution layers.

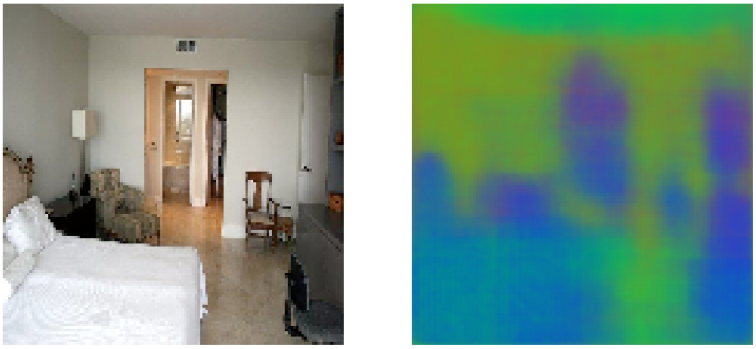


Figure 3: Baseline generator segmentation. Note the artifacts in the image, likely due to overlapping deconvolution layers.

Augmented Generator Architecture

Below is a diagram of the augmented generator architecture. Rather than show the full architecture, we instead show the first few layers. 4 upsampling/convolutional blocks were used in the architecture, along with input and output layers.

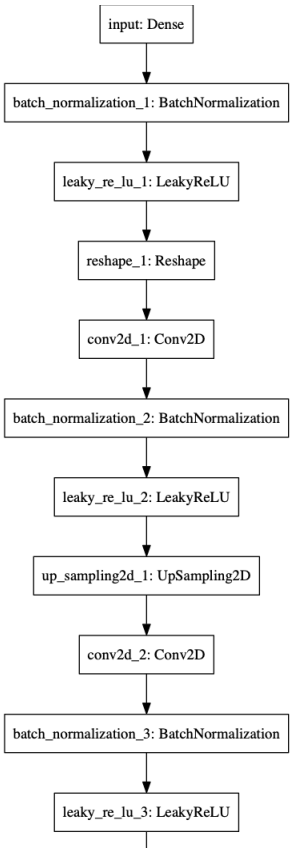


Figure 4: Augmented generator architecture

This generator architecture achieved 65.24% accuracy on the AD20K dataset without parameter tuning and paired with a simple encoder structure[11], after 50 epochs of training. The segmented image generated is freer of artifacts when compared to the baseline above, but also seems to be less accurate when segmenting the image, providing only very poorly defined outlines- likely due to the use of upsampling. The higher accuracy suggests it is merely the 'blobby' style that makes the results look worse- the architecture is performing better than the baseline, but is worse at creating definite edges, resulting in seemingly inferior performance on purely visual inspection. While this more 'blobby' image is clearly a drawback for the image segmentation task, it provides better results in the pokémon generation task, due to the watercolour pokémon art making more use of blobs of colour. This displays how architectures suited for one task may trade off performance in another task - a 'one size fits all' architecture is difficult to find. Again, as image segmentation was not a focus of this report,

[11] See our codebase for this encoder - it is simply a few convolution layers.

we simply include an example of the segmentation results below and focus on pokémon generation in the rest of the report.

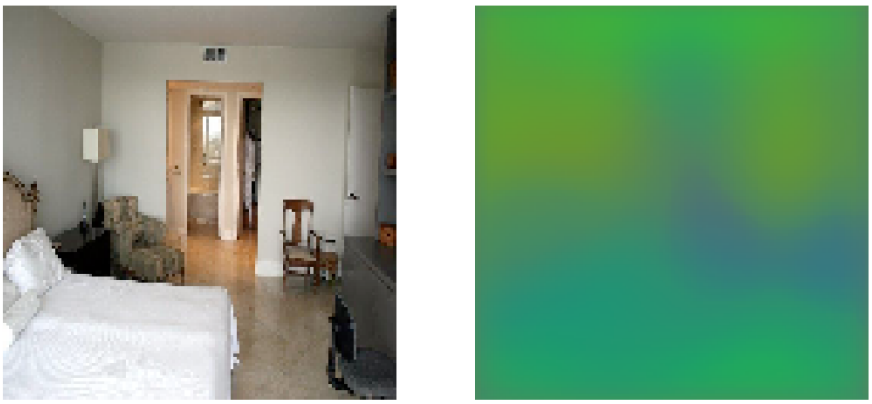


Figure 5: Augmented generator segmentation. Note the generator seems to only be able to vaguely segment the scene.

Discriminator

Two discriminator architectures were used: *(i)* a base architecture consisting of repeated convolution, batch normalisation, and activation layers, and *(ii)* an augmented architecture based off resnet[12]. The augmented architecture uses residual blocks, which allow the network to 'skip' convolutional layers. The intuition here is that the network is simultaneously very simple and quite deep, depending on if you take these shortcuts or not. Hence, the network can learn simple features fast and easily using only a few layers, and then slowly expand to use its entire depth when determining more complex features. As our computer GPU's memory was limited, we reduced the full resnet architecture to fewer layers, while still using the residual blocks. Each residual block is simply a set of standard convolutional blocks (repeated convolution, batch normalisation, and activation layers), with a shortcut from the first convolutional block to the end of the residual block. These residual blocks either have their own convolutions in the shortcut path (in order match the longer path's downsampling), or no convolution.

To test the discriminator architectures, a simple digit recognition task using the MNIST dataset was performed. Results are given alongside the diagrams below.

[12]The architecture proposed in (He et al. 2015)

Base Discriminator Architecture

Below is a diagram of the base discriminator architecture. Again, rather than showing the full network, we instead show the first few layers. In the full architecture, 5 convolution blocks were used, with an output dense layer. This number of blocks was decided on after testing different amounts of blocks, with 5 blocks providing the best results for pokémon generation. This architecture achieved 98.64% accuracy on the MNIST dataset after 12 epochs of training, without parameter tuning, showing it is fairly good at classification, but not incredibly powerful.

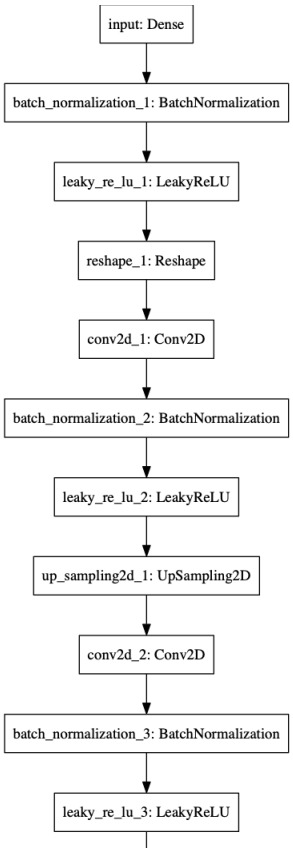


Figure 6: Baseline generator architecture

Augmented Discriminator Architecture

As the full architecture is quite deep, again, instead of showing the full network, below there is a diagram of the two main blocks of this network. Four of these blocks, as well as an input convolutional layer, and an output dense layer, made up the entire network. This architecture achieved 98.13% accuracy on the MNIST

dataset after 12 epochs of training, without parameter tuning, showing it is fairly good at classification, with limitations in results likely due to overfitting. Since the MNIST dataset is fairly simple, it makes sense a more complex architecture would achieve similar, if not worse, results to a simpler one.

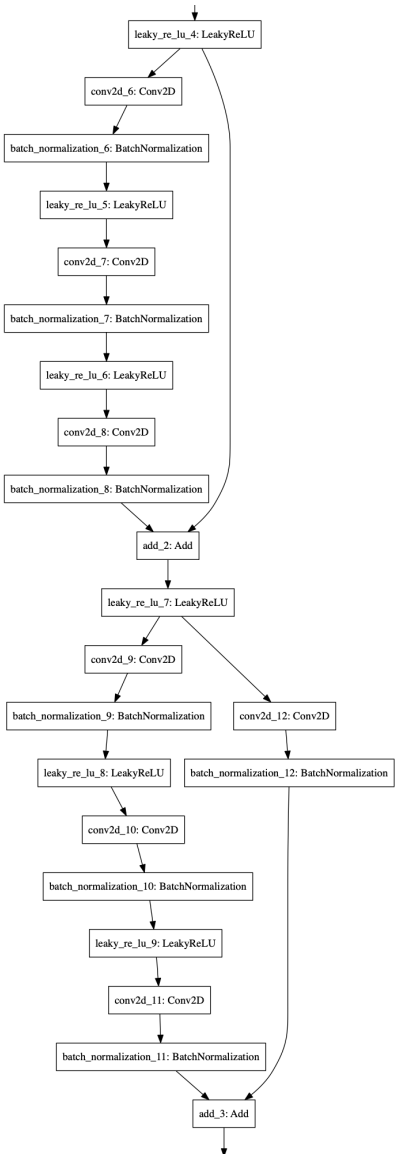


Figure 7: Resnet-inspired discriminator architecture

Overall WGAN Setup

We implemented a custom training program, as the Keras library did not natively support WGAN training. The discriminator was trained on 10 batches of data for every 1 batch on the generator. This is based on advice in (Arjovsky, Chintala, and Bottou 2017), with the number of discriminator iterations increased from the suggestions in the paper to ensure it converged. The exception was the first 25 epochs (or every 500 epochs thereafter), in which the discriminator was trained on 100 batches for every one batch on the generator. Note that here, an epoch is not a full run through the dataset, but rather a single batch being trained on the generator. Weights were clipped to the range [-0.01, 0.01]. Images were also rescaled so that RGB values were in the range [-1.0, 1.0], rather than [0.0, 1.0]. This was done both for input images and the generator output, meaning images must be rescaled to the [0, 1] range before they can be saved as images. The RMSProp optimiser was used with parameters as described in (Arjovsky, Chintala, and Bottou 2017). This is based on the original WGAN code, found here[13].

The same parameters were used for the WGAN-GP, with the exception of weight clipping. For every generator batch training, the discriminator was trained 5 times (even for the first 25 iterations, unlike the regular WGAN). The Adam optimiser was also used with parameters as described in (Gulrajani et al. 2017). Images were similarly rescaled to the [-1.0, 1.0] range.

Experiment

Generator and Discriminator Experiments

In order to determine the best setup, we implemented and ran three different WGAN configurations for 10,000 iterations each:

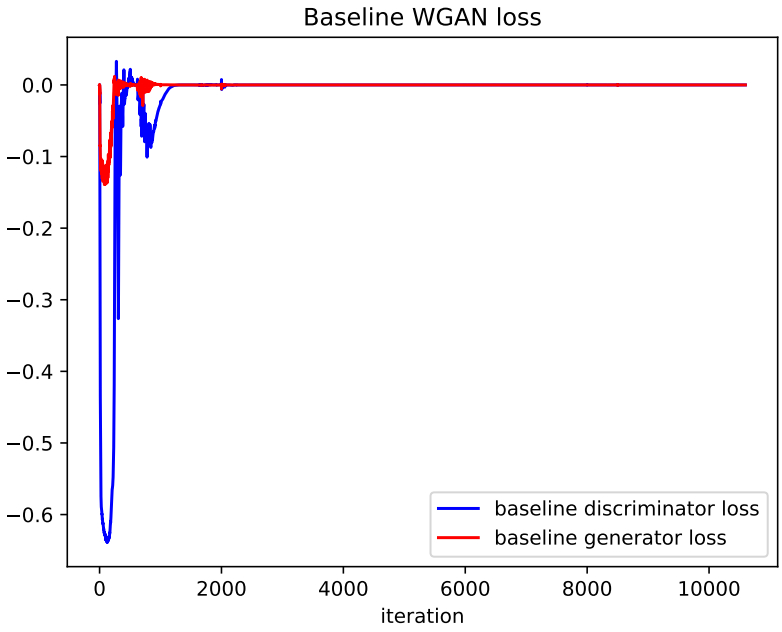
- baseline discriminator and baseline generator (the baseline WGAN)
- baseline discriminator and improved generator (the improved generator WGAN)
- resnet discriminator and improved generator (the improved WGAN)

Other configuration details were set as described in the above section. Note that for the baseline discriminator, the discriminator was altered slightly - the filter values started at 16 and doubled for 4 layers, rather than starting at 64 and doubling for 4 layers as in the current codebase. This was required to ensure the WGAN was able to train properly. The dataset used for all experiments was the Sugimori pokémon art dataset, described above.

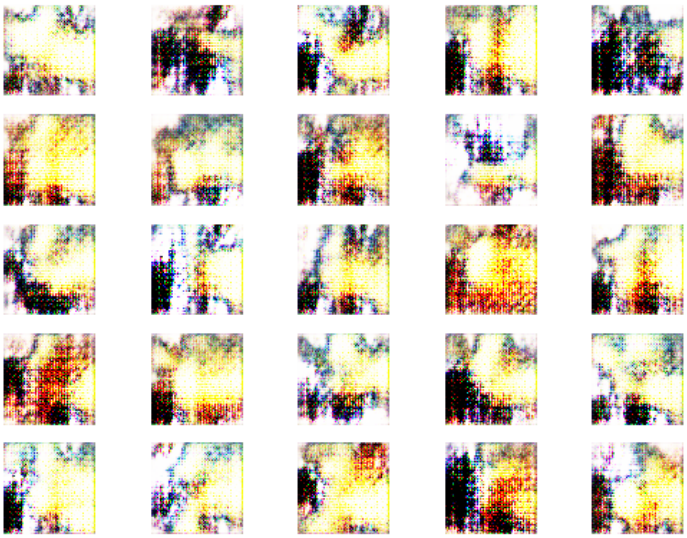
[13] (Team 2018) and (Linder-Norén 2018) were also used as references when writing our code.

Results

Below are the loss graphs and generated image samples for each configuration, respectively:

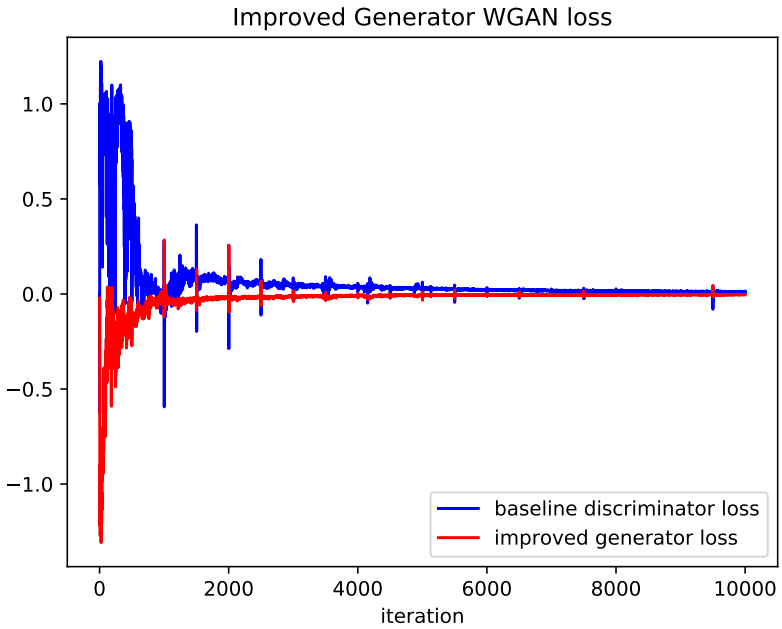


(a) Baseline loss graph



(b) Baseline generated images

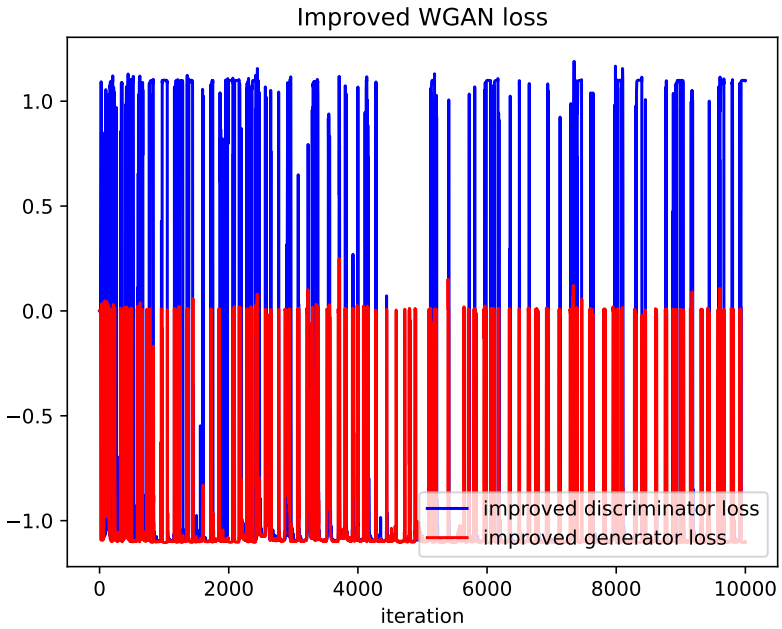
Figure 8: The results of the baseline WGAN configuration



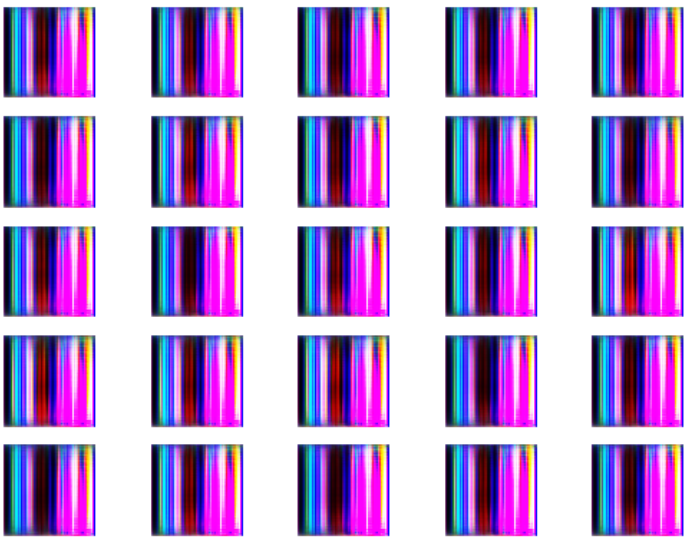
(a) Improved generator loss graph

(b) Improved generator images

Figure 9: The results of the improved generator WGAN configuration



(a) Improved WGAN loss graph



(b) Improved WGAN images

Figure 10: The results of the improved WGAN configuration

Discussion

As we can clearly see, the best results were found in the configuration with the baseline discriminator and improved generator, providing a clear improvement over the baseline. The baseline WGAN, while it does display some rough shapes, has clearly not learnt proper pokémon colours nor more complex shapes. This lack of learning is reflected in the loss graph of the baseline WGAN: rather than the discriminator loss growing large and steadily decreasing, it increases initially, and then collapses completely. This suggests the discriminator may be overfitting, or is too weak relative to the generator (or both). In the first case, the discriminator may be relying on the checkerboard artifacts produced by the generator, due to its use of deconvolution layers[14]. The discriminator learns to use these checkerboard artifacts, and as the generator unlearns them, the discriminator's loss collapses due to it relying too strongly on those artifacts. In the second case, the generator is able to fool the discriminator with junk images, and while the discriminator does eventually learn some features (how else would the generator learn to produce shapes), the learning process is much slower due to this weaker discriminator. Furthermore, the discriminator may even stop being able to learn at some point due to weight clipping hindering its ability greatly (it can only choose from a small range of weights). While a stronger discriminator, and parameter tuning, may improve these results, there are more stable and much better ways to improve our performance from this baseline.

In contrast, the improved generator WGAN shows much better performance. The loss graph shows a clear convergence: the discriminator loss slowly decreases over time, showing that the generator is slowly improving in quality, though the discriminator is still able to differentiate between generated and real images. This downward trend continues through the entire 10,000 iterations, and if we had more time for training it likely would have continued for much longer. This shows that the improved generator and stronger discriminator are far better matched. The improved generator is able to produce images without checkerboard artifacts from the very beginning, and so the discriminator is forced to learn to distinguish between truly random images and the real images. This results in much faster learning- shapes start emerging around 2,500 iterations, much earlier than other WGAN configurations. Furthermore, we no longer see the collapse issue present in the baseline configuration, which may have been due to those checkerboard artifacts being 'overfitted' on by the discriminator. With this stronger generator, the slightly stronger discriminator[15] is also able to keep up, resulting in images that are clearly emulating the watercolour style of the dataset.

Finally, the improved WGAN setup has clearly completely failed. The loss fluctuates wildly, and does not converge. This is likely due to the weight clipping parameter: in enforcing the weights to be within a certain range, the WGAN is biased towards simpler models as opposed to more complex ones, and so the discriminator either fails to learn well, or even stabilises on a simple and useless

[14] see (Odena, Dumoulin, and Olah 2016) for more.
[15] see the beginning of the section for details on this

model. The full capacity and power of more complex networks cannot be used, due to this weight clipping. This bias is noted in (Gulrajani et al. 2017), and was clearly observed across our experiments with other improved architectures. Even using pretrained models did not help with this[16], showing that it is clearly not an issue with the discriminator not having converged[17]. Hence, this clearly shows the instability of the WGAN: it is highly dependent on this weight clipping parameter, and is not a simple 'plug and play' architecture suitable for all types of generator and discriminator networks.

Thus, we can see that when it comes to WGANs, simpler models perform better. The improving of the generator yielded excellent rewards, speeding up the training of the WGAN, but adding more complexity to the discriminator failed. This suggests that WGAN architectures should be kept simple and straightforward to achieve the best results, rather than relying on stacking many layers. Likewise, more care should be put towards the weight clipping parameter.

Data Augmentation Experiments

In order to determine the effect of data augmentation on WGAN performance[18], we used the improved generator WGAN setup (detailed above) on the following data augmentation setups for 10,000 iterations:

- no data augmentation
- flipping only
- padding and random zoom only

All other configuration parameters were kept constant from the previous improved generator WGAN experiment.

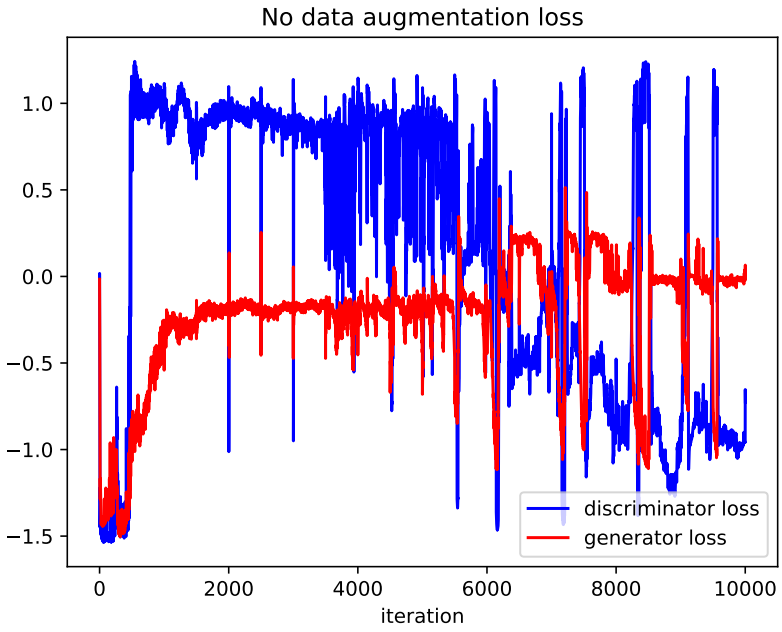
Results

Below are the loss graphs and generated image samples for each data augmentation setup, respectively:

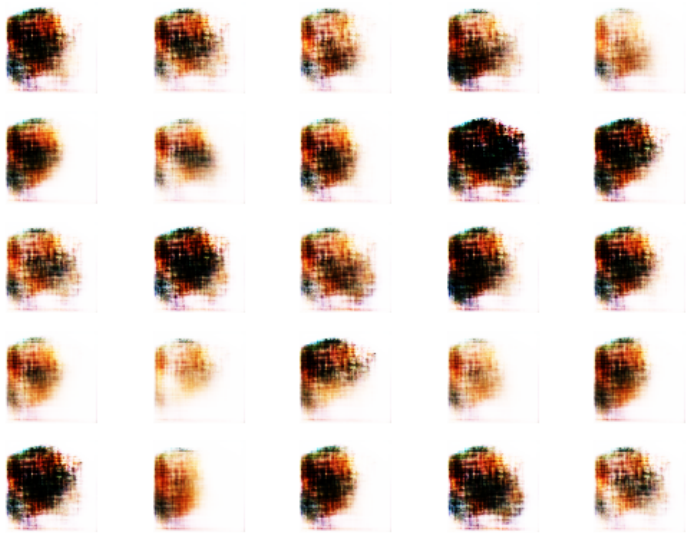
[16] We ran experiments with several pretrained models available in the Keras library: NasNet, ResNet50, Inception-ResNet-v2, and more, without any luck. See Appendix A for the loss graphs generated when running NasNet Mobile and ResNet50. These experiments were not fully run due to their clearly subpar results.

[17] If this was the case, then using pretrained models, which have already converged on a dataset and training them for a few iterations to adapt to the new dataset should allow fast convergence of the discriminator and thus good results.

[18] This experiment was performed as the assignment specification stated: "Compare pokémon image generation results with and without different data augmentation techniques." Hence, we compared three different data augmentation techniques.

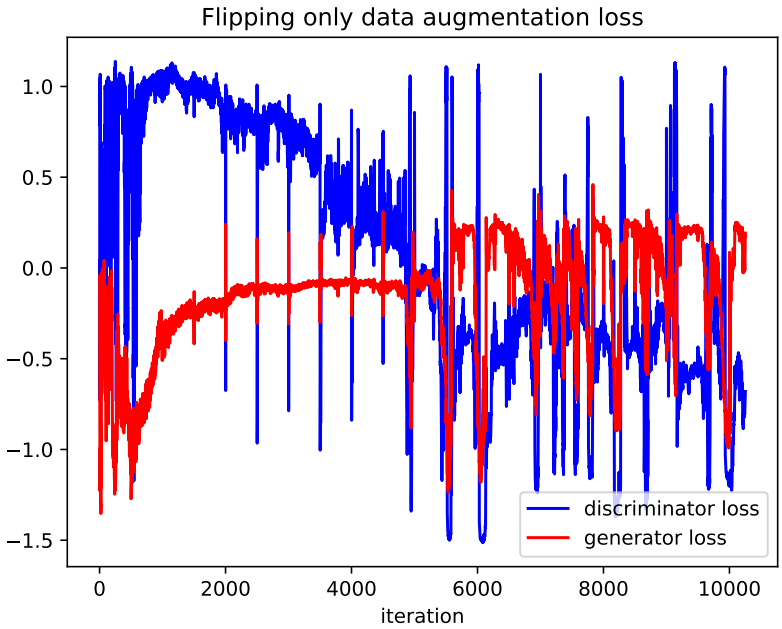


(a) No augmentation loss graph

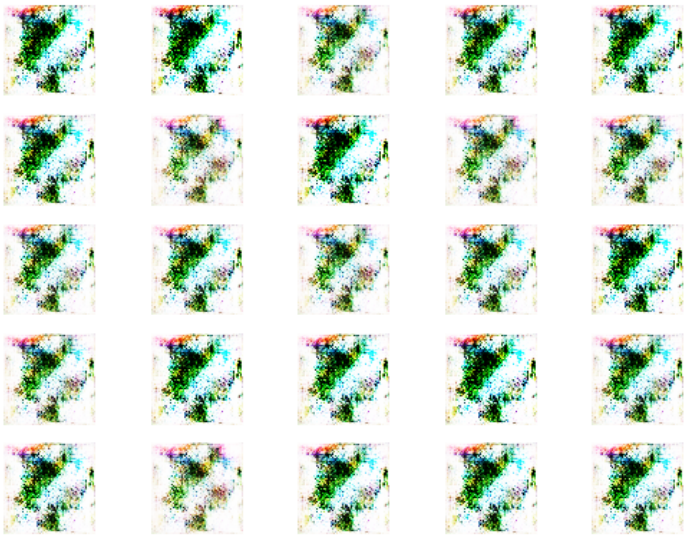


(b) No augmentation images

Figure 11: The results of not using any form of data augmentation

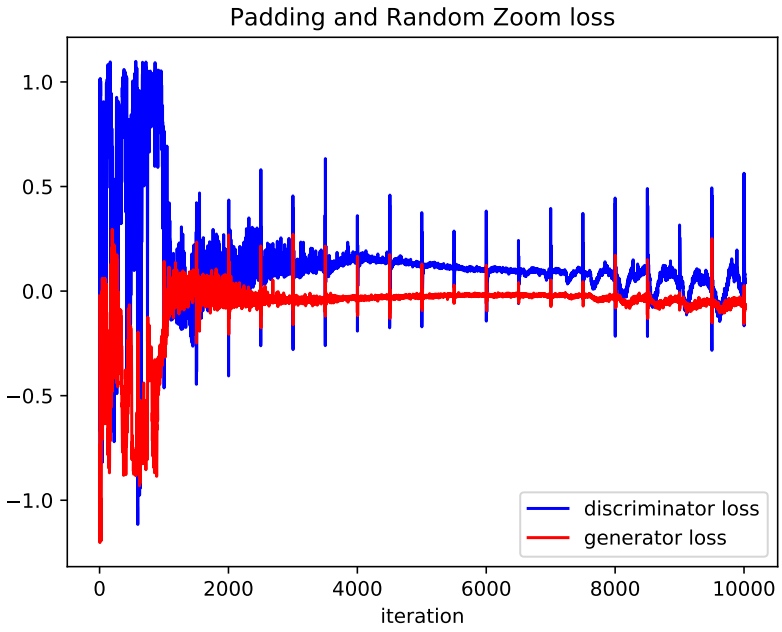


(a) Flipping only loss graph

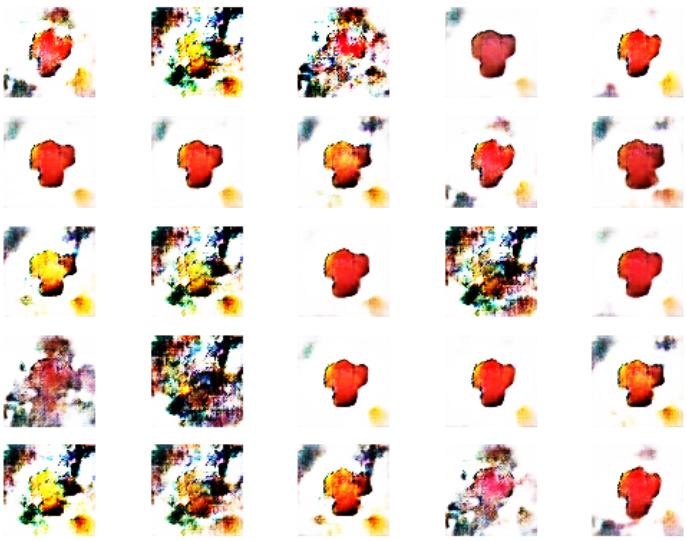


(b) Flipping only images

Figure 12: The results of only using vertical and horizontal flipping for data augmentation



(a) Padding and random zoom loss graph



(b) Padding and random zoom images

Figure 13: The results of using padding and random zoom for data augmentation only

Discussion

As we can see, the use of padding and random zoom gives the best results, with no augmentation and flipping only providing poor results. This is likely due to the tiny dataset: with only around 1000 images, the dataset of pokémon images is well below that of typical datasets for GANs. Hence, the discriminator is more prone to overfitting on the dataset, resulting in slower training. As the discriminator overfits, focussing on specific elements of the input dataset, the generator learns this and takes advantage of it. The discriminator then has to 'unlearn' this overfitting. This can be seen in the loss graph for the no data augmentation setup - while the discriminator is still learning normal features (the first section of the graph), it is steadily decreasing in loss, and the generator slowly converging with it, as expected. However, over time the loss grows more erratic, mirroring the overfitting of the discriminator - a generator that relies on very specific features of the input dataset is easier to fool, resulting in the wild fluctuations as it overfits, unlearns the overfit feature, then overfits again, and so on.

A similar pattern can be seen in the flip only data augmentation loss graph, with a slightly smoother initial convergence. This is because the flipping only increases the dataset marginally (a four times increase, which is not enough to boost it into the millions of images range that GANs usually require). Hence, while the marginal increase does stop overfitting for a bit, after a while the discriminator still does overfit and so creates the wildly fluctuating loss that can be seen in the second half of the loss graph, just as in the no data augmentation case. It is important to note, however, that as evidenced by the images generated, the generator is still learning clear features of pokémon: in both cases, it demonstrates knowledge of a white background, and a central shape. This suggests that learning is still happening, but at a far reduced rate. This is because the GAN does 'unlearn' the overfitting over time, and so does learn correct features of the dataset, just at a much slower rate, since the generator must exploit the overfitting before it can be 'unlearnt' by the discriminator. Hence, it is likely the case these would give similar results to that with the full data augmentation setup, given enough time and training.

In contrast, the padding and random zoom clearly produce the best results, with relatively stable training, as seen in the loss graph. This is clearly because random padding and zoom results in an almost infinite set of possible images being produced from the base dataset, providing a far greater range of images than just flipping or no augmentation. However, as we can see, this does not give results as good as the entire data augmentation setup (see the results of the improved generator WGAN above), and does show some instability towards the end of its training. This is likely due to similar issues described above - while the dataset has effectively been incredibly increased by applying random crop and zoom, it still isn't a particularly large dataset, and so still starts to somewhat overfit towards the end of the training. This results in the 'bouncing' of the discriminator, as it repeatedly overfits and unlearns its overfitting. Hence, the

training of this configuration is still slower than the training of the improved generator WGAN, and so produces slightly worse results.

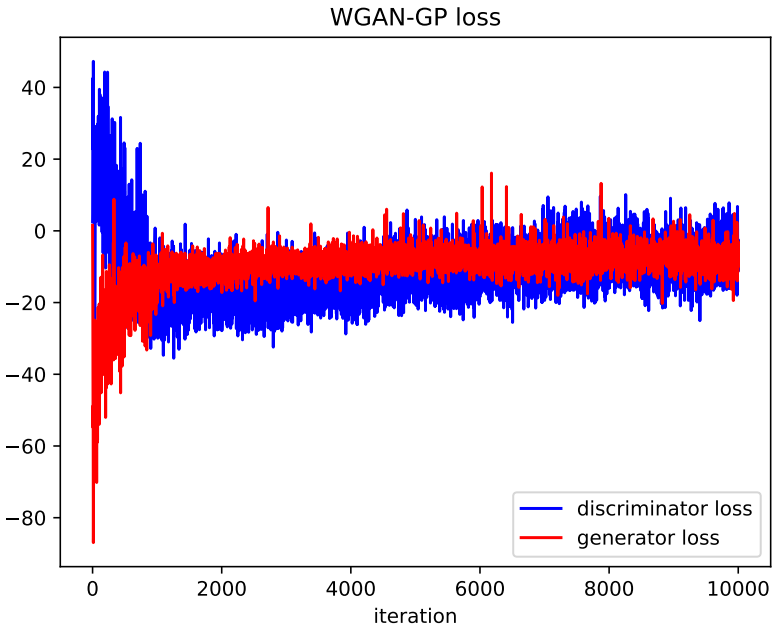
As we can clearly see from the above discussion, more data is better when it comes to WGANs. Applying any sort of data augmentation provides better training, and the more augmentation the better, improving both the overall results and the training stability.

WGAN-GP Experiments

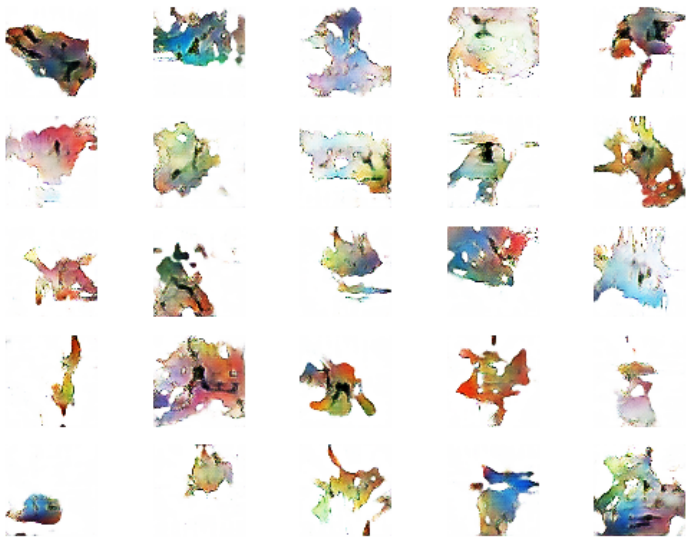
In order to see how much the use of the WGAN-GP improved performance, we performed one experiment with the WGAN-GP, using our improved generator architecture described above. The configuration and parameters are described in the method section, and the WGAN-GP was run for 10,000 iterations.

Results

Below are the results of this experiment.



(a) WGAN-GP loss graph



(b) WGAN-GP images

Figure 14: The results of the improved generator architecture used in with the WGAN-GP.

Discussion

The WGAN-GP clearly gives the best results, but also has a vastly different loss graph to previous experiments. However, the generator clearly improves over time, as the discriminator and generator loss converge quite early on. This matches the experiment itself - as early as iteration 500, the WGAN-GP was generating images with distinct shapes and colour patterns similar to the input dataset. Hence, the less drastic movement during the rest of the loss graph reflects this early convergence, with only minor improvements being slowly made over the rest of the training, as the discriminator and generator learning more complex features. The other noticeable thing about the loss graph is its noise: the WGAN-GP seems to be far less stable, at a glance, than the far more steady loss graphs of the regular WGAN. This is likely due to the use of the Adam optimiser, and increased training speed of the WGAN-GP: rather than making changes very incrementally, the WGAN-GP is free to make full use of the discriminator, and adjust its weights to a far greater extent than the regular WGAN, especially with the removal of weight clipping, which also would have had a dampening effect on the loss graph of the WGAN. Hence, the WGAN-GP shows a significant improvement over our previous results: not only does it converge much earlier, producing good results, it also provides much better results overall, as the severe negatives of weight clipping and a low learning weight are removed in the WGAN-GP training algorithm.

Conclusion

In conclusion, we have clearly shown several things in this report: the importance of data augmentation, the need for simple yet powerful architectures in WGANs, and the vast improvement that the WGAN-GP model brings, with the WGAN-GP with the improved generator providing the best results. However, there is much that could be done to further investigate how performance could be improved.

Future Work

In the future, more work should be done to see how other architectures affect the performance of the WGAN-GP, as was done for the WGAN in this report. We have also shown the importance of having as large a dataset as possible, with data augmentation (especially cropping and zooming) proving invaluable to the performance of the WGAN. Larger pokémon datasets should also be gathered, and their effects on the results examined - would adding pokémon sprites to the input data improve results, or confuse the discriminator, since they use a vastly different art style? From our initial attempts at WGAN training, adding in more data did not speed up learning, and even if training was stable resulted in sub-par images. We hypothesise this is due to the art style clashes (small

pixel images and large watercolour images have very different characteristics), but more formal work should be done here. Furthermore, the experiments we considered above were short for GAN experiments - often networks are left to train for weeks, while our longest experiment lasted two days. Hence, future work should also investigate if more training time improves or otherwise changes the results found in this report. We hypothesise that improvements should be possible with more training time, since the best WGAN experiments were still learning and improving their results at the 10,000 iteration mark.

More formal experiments with different WGAN-GP architectures should also be considered. (Gulrajani et al. 2017) were able to use ResNet101 as a discriminator and achieve results, suggesting the WGAN-GP is much more stable than the base WGAN, as our experiments found. Due to time and budgetary constraints, and the assignment asking for base WGAN investigations, we did not dig deep into WGAN-GP experiments, but our initial results above are very promising, and could very likely be improved upon. Furthermore, recently more work has been done on improving the WGAN, such as the WGAN-CT[19], all of which should be investigated. GAN research moves fast, and if the past is any indication, in the years to come there will be many innovations that improve on the results found in this report in incredible ways.

Final Thoughts

One clear result of this report is that data is important, and with it, data augmentation: the best results only occurred when all data augmentation parameters were used. This reinforces a fundamental truth in neural network research: the more data, the better.

However, the strong results of the improved generator WGAN, when compared to the baseline, shows how utilising new architectures can add much to the WGAN performance, but the weak results of the improved WGAN shows that it is not as simple as 'plugging in' large architectures - unlike other tasks, deeper does not mean better for WGANs. Rather, much careful thought and tuning must be used in choosing your WGAN architecture. The WGAN-GP shows improvements in this aspect, allowing for better results with the same architecture, and providing a much better algorithm for using larger neural networks in its architecture. This 'plug and play' feature is a great boon, allowing neural networks designed for different purposes be used in the WGAN-GP architecture, allowing for a system of neural networks that not only generates great pokémon, but also provides great image classification and image segmentation results.

[19] see (Wei et al. 2018) for more details

References

Arjovsky, M., S. Chintala, and L. Bottou. 2017. "Wasserstein GAN." *ArXiv E-Prints*, January.

Goodfellow, I. J., J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. "Generative Adversarial Networks." *ArXiv E-Prints*, June.

Gulrajani, I., F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. 2017. "Improved Training of Wasserstein GANs." *ArXiv E-Prints*, March.

He, K., X. Zhang, S. Ren, and J. Sun. 2015. "Deep Residual Learning for Image Recognition." *ArXiv E-Prints*, December.

Hui, Jonathan. 2018. "GAN - Wasserstein Gan & Wgan-Gp – Jonathan Hui – Medium." *Medium*. Medium. https://medium.com/@jonathan_hui/gan-wasserstein-gan-wgan-gp-6a1a2aa1b490.

Kingma, Diederik P., and Jimmy Ba. 2014. "Adam: A Method for Stochastic Optimization." *CoRR* abs/1412.6980. http://arxiv.org/abs/1412.6980.

Lai, Shaofan. 2017. "Implement Improved Wgan with Keras-2.x." *Shaofan Lai's Blog*. http://shaofanlai.com/post/10.

Linder-Norén, Erik. 2018. "Keras-Gan." *Github*. https://github.com/eriklindernoren/Keras-GAN.

Nibali, Aiden. 2017. "Mode Collapse in Gans." *Aiden Nibali's Blog*. http://aiden.nibali.org/blog/2017-01-18-mode-collapse-gans/.

Odena, Augustus, Vincent Dumoulin, and Chris Olah. 2016. "Deconvolution and Checkerboard Artifacts." *Distill*. doi:10.23915/distill.00003.

Team, Keras. 2018. "Improved_wgan.py." *Github*. https://github.com/keras-team/keras-contrib/blob/master/examples/improved_wgan.py.

Wei, Xiang, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. 2018. "Improving the Improved Training of Wasserstein Gans: A Consistency Term and Its Dual Effect." *CoRR* abs/1803.01541. http://arxiv.org/abs/1803.01541.

Weng, Lilian. 2017. "From Gan to Wgan." *Lil'log*. https://lilianweng.github.io/lil-log/2017/08/20/from-GAN-to-WGAN.html.

Yurasov, Mikhail. 2017. "Wasserstein Gan in Keras." *Deeply Random*. https://myurasov.github.io/2017/09/24/wasserstein-gan-keras.html.

Appendix A - Experiment Loss Graphs

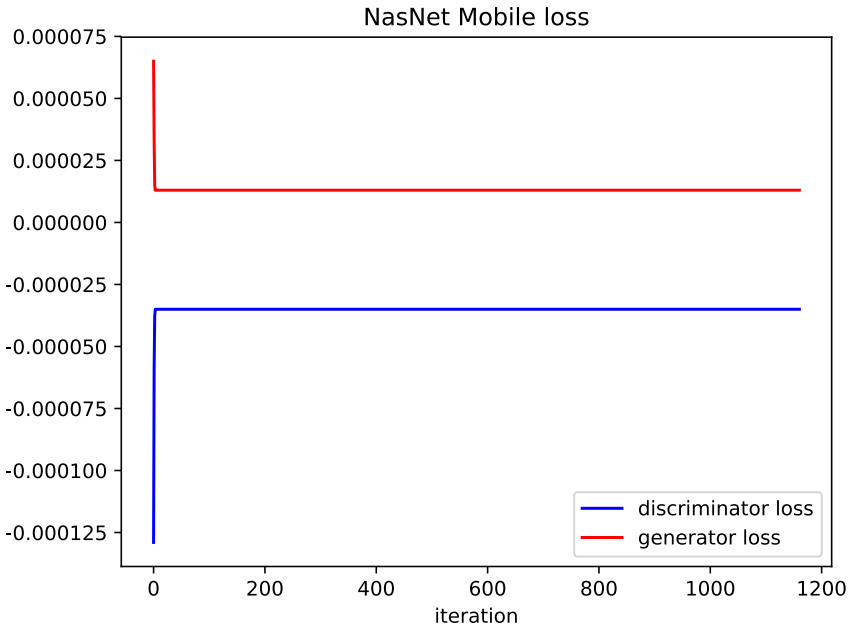


Figure 15: NasNet Mobile loss when used with the improved generator architecture and base WGAN algorithm. Note the loss remains constant after a few iterations, showing a complete lack of training, likely due to weight clipping completely hindering the model, as discussed in this report.

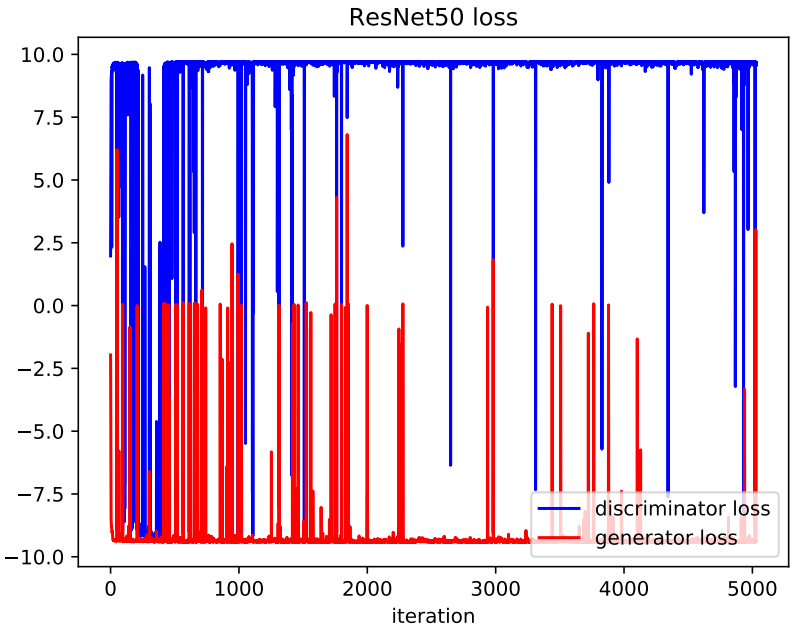


Figure 16: ResNet50 loss when used with the improved generator architecture and base WGAN algorithm. This experiment showed similar results to the resnet experiment discussed in this report, likely due to similar reasons.