Super Resolution with Adaptive Data Augmentation using Generative Adversarial Networks

Alberto Baldrati
alberto.baldrati@stud.unifi.it

Giovanni Berti
giovanni.berti2@stud.unifi.it

Abstract

In this work we implement a novel data augmentation technique tailored to Generative Adversarial Networks in order to reduce discriminator overfitting and stabilize training. This technique was first described in [1] and applied in an image generation from latent space task. We experiment such approach in a super resolution setting using a slightly modified SRGAN [2] achieving promising results when using a small amount of data.

1 Introduction

Training a Generative Adversarial Network can be very challenging. This is in part because in order to achieve high quality generated data big training datasets are often necessary. The main problem with small datasets is that the discriminator overfits to the training examples; its feedback to the generator becomes meaningless and training starts to diverge. Therefore in domains where sufficient training samples are not available GANs without any kind of regularization or augmentation tend to perform badly. In almost all areas of deep learning, especially in computer vision, *data augmentation* is a powerful technique that allows training models with little data helping to alleviate the negative effects of overfitting. However such standard data augmentation pipelines are not directly applicable to GANs because the generator learns the augmented distribution of the original data. Such *leaking* is often undesirable, for example a noisy augmentation leads to noisy results, even if there is none in the dataset. To overcome such issues in [1] is presented a new augmentation pipeline tailored to GANs which mitigate the *leaking* effects of augmentation. Moreover, in contrast to traditional data augmentation pipelines, the technique designed in [1] is adaptive, meaning that it follows an adaptive control scheme that enables the same approach to be used regardless of the amount of training data, properties of the dataset, or the exact training setup.

In [2] is shown that Generative Adversarial Networks can reach better performance on super resolution task compared to other approaches, mostly by being able to generate finer details and higher frequency components with better fidelity. In our work we will describe such novel data augmentation technique applied to a super resolution GAN which closely modeled after the original SRGAN [2]. We will later detail how and why the modifications we performed were chosen and how they relate to the network's performance when combined with the data augmentation scheme.

2 Network Achitecture

In this section we briefly describe the original SRGAN architecture [2] and the changes we made in our implementation in order to better adapt to data augmentation.

2.1 Original SRGAN Architecture

The original SRGAN architecture is illustrated in figure 1 on the following page.

The generator consists in B residual blocks, each made up of two composed 2D convolutions with batch normalization and Parametric ReLU as activation function. This block also employs an input-output residual connection. After these residual blocks there is another convolutional layer with a skip connection followed by two upsampling blocks which are composed to reach a $4\times$ upsampling factor. These upsampling blocks are made of a sub-pixel convolutional layer[3] activated by a PReLU. Finally, another convolutional layer is present before the network's output.

The discriminator consists in eight convolutional layers (four of which are strided in order to reduce image size without any pooling) with an increasing number of 3×3 filter kernels, increasing by a factor of 2 from

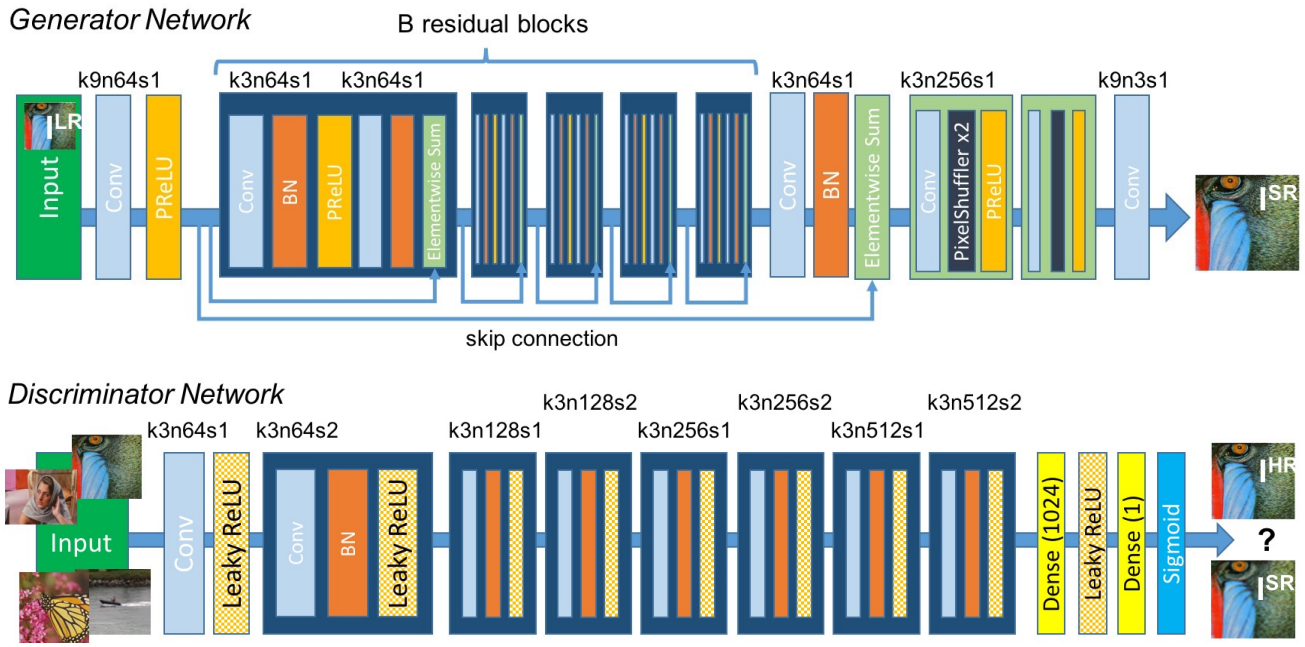


Figure 1: SRGAN original architecture of Generator and Discriminator Networks with corresponding kernel size (k), number of feature maps (n) and stride indicated for each convolutional layer

64 to 512 kernels, as in the VGG network. On top such eight convolutional layers is built a small feed-forward network which classifies the input as either real or generated.

The purely convolutional nature of the generator allows it to be flexible with regard to the input size. This means that whatever the size $n \times n$ of the input image, it will always be able to output a $4n \times 4n$ image. On the other hand the discriminator is tied to a fixed input size, because of the final feed-forward network present in the latter layers. In [2], the proposed input size is 96×96. We will later detail how this property is used during training. Moreover, because the discriminator is used only during the training phase, this does not affect the network's usage at inference time.

2.2 Modified SRGAN Architecture

Our generator architecture differs slightly from the one described in figure 1. Its primary difference is the inclusion of a residual connection. This connection sums the network's output with a bicubic upsampling of the input. This allows the network's initial weights to correspond to approximately an identity function instead of a zero mapping [4]. In other words, the generator has only to learn the residual from a bicubic upsampling to a fully super-resolved image. This technique was employed primarily because it allows faster convergence, but it has also been shown to be effective in avoiding undesired local optima during training. The other main difference is the absence of batch normalization layers in all B residual blocks, such layers has been removed in order to speed up training since their presence didn't bring any noticeable performance improvements. Our discriminator architecture instead does not differ from the one described in figure 1. In appendix A on page 11 we detail other network architectures we experimented with.

3 Training procedure

Our baseline training procedure analogous to the one described in [2]. We start from a ground truth dataset made of high resolution images. Because the discriminator architecture constraints the input size to be fixed, a suitable *patch size* must be chosen. In our case, we chose a patch size of 128×128. During training the images are cropped randomly, and a downsampled copy of the patches is made, with a downsampling factor of 4 (equal to the upsampling factor of the network). The downsampled patches are then fed to the generator while the generator output and the original high resolution image are fed to the discriminator.

3.1 Losses

When applying GANs in super resolution tasks, the training loss is usually made up of two components: a *perceptual* loss and an *adversarial* loss. The perceptual loss signals the generator the overall structure of the output images, while the adversarial loss is computed using the discriminator, and is the primary force that pushes the generator into outputting finer details.

Intuitively, the easiest choice for a perceptual loss would be a pixel-wise MSE loss between the high resolution input patch and the generated output. When used alone this loss produces images that closely resemble the original input, however they suffer from heavy blurriness and overly smooth textures. In [2] the authors propose a *VGG loss*. This loss is computed by taking the mean square error of the feature representation of the ground truth and the generated images, as in equation (1). It can attain a higher degree of perceptual similarity between input and output images, meaning that even without producing an output that matches exactly the ground truth, the overall spatial structure and color features of the image are preserved.

$$l^{SR}_{VGG} = \frac{1}{\dim(\phi(\hat{y}))} \|\phi(\hat{y}) - \phi(y)\|_2^2 \tag{1}$$

Equation 1: VGG loss. $\phi(\cdot)$ denotes a fixed feature map produced by an intermediate layer of the VGG network

In our experiments we observed that the usage of VGG loss led to a greater training instability (*e.g.* the discriminator got too strong too early in the training without letting the generator learn enough to produce high quality images). Because of this, we choose to rely on a simpler MSE loss for our model. In appendix A on page 11 we will detail the effects of this particular loss and what drove us to choose a simpler loss for our model.

In order to avoid local minima and to head start the generator into producing decent quality images, we employ a pre-training phase where only the perceptual loss is active.

A subtle issues that arises when training GANs is how to balance the adversarial loss with the content loss. Training stability and output quality are in fact very sensitive to this balancing factor. Intuitively, a loss that is dominated by the perceptual component will not be able to generate finer details, while a loss dominated by the adversarial component will force the generator to learn to "trick" the discriminator and produce outputs that are close in the discriminator feature space to the ground truth images, while lacking perceptual similarity. In our experiments we chose a balancing factor such that the adversarial loss was $5\% - 10\%$ of the perceptual loss in the first epoch with adversarial loss, as current literature has shown that it gives a good compromise between training stability and output quality.

4 Data augmentation

One of the major problems that occurs when training GANs is when the discriminator becomes overly confident of its prediction too early in the training. When this happens, usually its feedback signal to the generator is either useless or too weak to be useful to improve on image quality. This issue is further exacerbated in low size datasets, where the discriminator can quickly learn the ground truth distribution while the generator is still learning. Therefore we need a way to tackle this problem by employing a non-leaking data augmentation pipeline.

4.1 Previous work

In [5] the authors proposed a data augmentation technique, called *balanced consistency regularization* (bCR), which is supposed to be non-leaking. Consistency regularization states that two sets of augmentations, applied to the same input image, should yield the same output. bCR adds a regularization term to the discriminator loss, enforcing discriminator consistency for both ground truth and generated images. This forces the discriminator to being able to distinguish ground truth and generated images both in the augmented and non-augmented case. However, no augmentation or consistency loss are applied when training the generator. When combined these two factors allow the generator to generate images that already contain the transformations applied during augmentation without any penalty. In figure 2 on the following page is outlined such augmentation pipeline

4.2 Stochastic discriminator augmentation

In [1] the authors design a variant of bCR, where the discriminator only sees the augmented images both when training the discriminator and the generator. Such approach is outlined in figure 3 on the next page and is named *stochastic discriminator augmentation*. The proper functioning of such technique is not obvious, in fact this means that the discriminator only sees the augmented distributions (respectively of ground

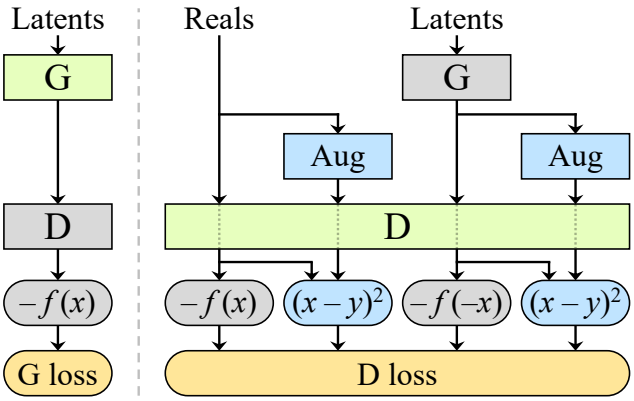


Figure 2: Flowchart of balanced consistency regularization (bCR)

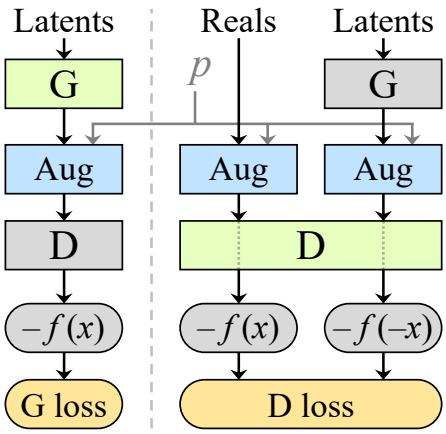


Figure 3: Flowchart of stochastic discriminator augmentation

truth images and generated images) and never sees the "true" distributions. When considering the effects of these transformations over the image distributions, as long as these transformations are invertible the training process is able to learn the correct underlying unaugmented distribution.

Note that this does not mean that the transformation itself must be invertible, for example an augmentation as extreme as setting the input image to zero 90% of the time is invertible in the probability distribution sense: it would be easy, even for a human, to reason about the original distribution by ignoring black images until only 10% of the images remain.

Many classical data augmentation transformations (*e.g.* rotation, scaling, pixel blitting, etc.) often are not invertible in the distribution sense. However, when applying a transformation stochastically (that is, by introducing a probability p of applying the transformation), the resulting distribution becomes skewed towards the original data. This makes the overall transformation invertible (in the distribution sense), and allows it to be used effectively as an augmentation during training without leaking artifacts into the generated images. Composition of stochastic transformation is not straightforward, however the authors provide a framework under which arbitrary transformations can be composed together, and develop a pipeline using well-known transformations that is guaranteed to be non-leaking. Due to finite sampling and to finite representation of the network dynamics, leaks are able to occur even when p is lower than 1. Experiments suggest that as long as p stays under 0.8 leaks are unlikely to happen in practice.

4.3 Adaptive discriminator augmentation

Up until now, the probability of transformation p is an additional hyperparameter that ought to be fine tuned for each network architecture and training dataset instance. The original authors propose an adaptive method that allows to automatically change p in relation to how much the discriminator is overfitting at any given point during training. The proposed measure of overfitting is called r_t, and is computed with

$$r_t = \mathbb{E}\left[\text{sign}\left(D_{\text{train}}\right)\right] \tag{2}$$

Equation 2: r_t overfitting metric

For this heuristic a value of $r_t = 0$ means no overfitting, while a value of 1 indicates complete overfitting. In equation (2) we denote with D_{train} the discriminator output on the ground truth dataset before the activation function. The aim of this adaptive augmentation is to tune p in order to keep r_t close to a suitable target value. Taking the sign insted of the actual value makes the overall metric less sensitive to chosen target values and to other hyperparameters. This adaptive variant increments or decrements p by a fixed amount whenever r_t is above or below the target value. Usually the computation of the expected value for r_t (and the subsequent update of p) is done every k minibatches instead of every minibatch. This allows the heuristic to be less sensitive to noise. In our experiments we have used directly such adaptive augmentation instead of the non-adaptive one.

5 Experiments

We conducted experiments on two datasets: FFHQ[6] and DIV2K[7]. FFHQ dataset contains 70k high resolution images of human faces. Such dataset is often used in human face generation from latent spaces, but it also fits well for super resolution purposes. The peculiar structure of faces makes the image super resolution task easier than in the general case. The other dataset we used is tailored for super resolution tasks. It contains 1000 very high resolution images in 2K resolution. The images depict various artificial and natural contexts and are more diversified than in the FFHQ dataset.

As stated above in order to avoid local minima and to head start the generator into producing decent quality images, in all experiments we employ a pre-training phase where only the perceptual loss is active. When testing the data augmentation algorithm we considered several random subsamplings of the original dataset (50%, 25%, 10%, 5% and 1%). When subsampling the training dataset, we took care of properly scaling the number of training and pre-training iterations such that the total number of iterations was the same (keeping the same ratio between two phases).

In table 1 we detail the data augmentation transformations parameters for the augmentation pipeline shown in [1]. Because we employ a patch based training strategy we removed from the original pipeline both cutout and translations. In our first experiments we discovered that image filtering (based on wavelets) and image rotation both contributed to artifacts in supersampled generated images (see appendix B on page 12), thus we also removed these transformations from the pipeline.

Transformation	**Parameters**	**Type**
x flip	N/A	$\mathscr{U}\{0,1\}$
90 degree rotation	N/A	$\mathscr{U}\{0,1,2,3\}$
Isotropic scaling	$\sigma = 0.2 \log 2$	Lognormal$(1, \sigma^2)$
Anisotropic scaling	$\sigma = 0.2 \log 2$	Lognormal$(1, \sigma^2)$
Brightness	$\sigma = 0.2$	$\mathscr{N}(0, \sigma^2)$
Contrast	$\sigma = 0.5 \log 2$	Lognormal$(0, \sigma^2)$
Saturation	$\sigma = 0.2 \log 2$	Lognormal$(1, \sigma^2)$
Hue rotation	N/A	$\mathscr{U}(-\pi, +\pi)$
Luma flip	N/A	$\mathscr{U}\{0,1\}$
Noise	$\sigma = 0.1$	Halfnormal$(0, \sigma^2)$

Table 1: Data augmentation transformations parameters.

5.1 FFHQ Experimental setup

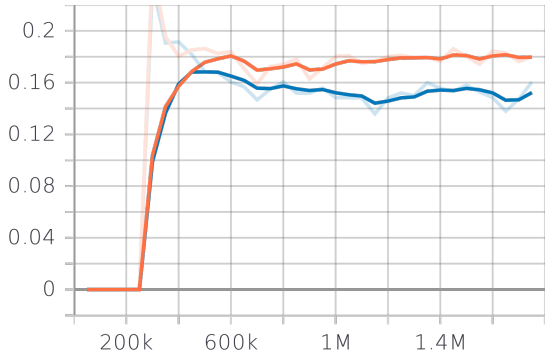
In this section we describe our experimental setup regarding training hyperparameters. The dataset we considered is FFHQ[6], downsampled to a 512×512 size. We split the dataset into a 50k portion for training purposes and two 10k portions for validation and testing respectively. All networks were pre-trained for **250k** iterations and trained in an adversarial setting for **1.5M** iterations, for a total of 1.75M iterations. We used the following hyperparameters:

- **Optimizer**: Adam (both for the generator and the discriminator), with a learning rate $\eta = 1 \times 10^{-4}$
- **Patch size**: 128×128
- **Number of residual blocks B**: 16
- **Batch size**: 32
- $\boldsymbol{k}$ **minibatch update frequency**: 8 minibatches
- $\boldsymbol{p}$ **update amount**: 1×10^{-3}
- **Adversarial loss balancing factor**: 4×10^{-5}
- $\boldsymbol{r_t}$ **target value**: 0.6

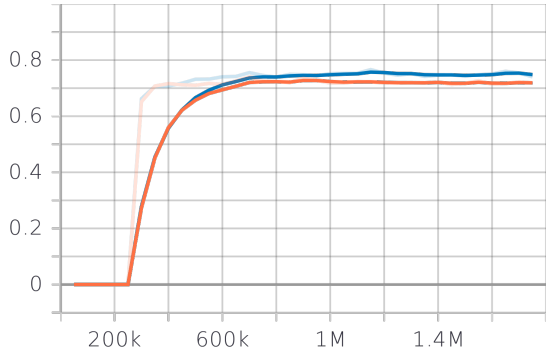
Moreover, as a supplementary measure to further limit early discriminator overconfidence, we applied one-sided label smoothing on ground truth images with a label value of 0.9 as suggested in [8]. To evaluate the performance of the model we considered the LPIPS[9] and FID[10] metrics, based on deep features, as well as the more traditional SSIM and PSNR.

5.2 FFHQ Experimental results

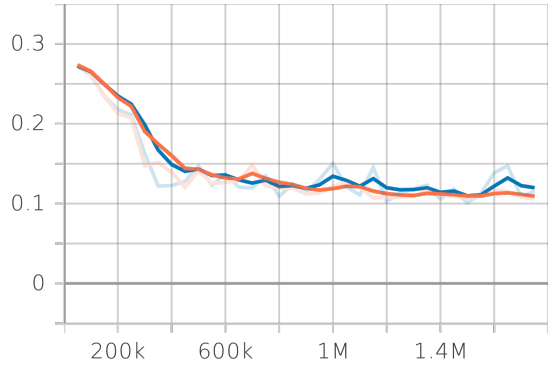
In table 2 on the following page are summarized the quantitative test results using FFHQ datasets. We can notice that the performance are comparable in almost all the experimental setups, meaning that in such tests the outcome does not depend neither on dataset size nor on the usage of the data augmentation described above. The most evident effect of augmentation is training stabilization: both evaluation metrics and discriminator output are much less noisier in the augmented case. The generator also manages to better fool the discriminator when augmentation is present, as in figure 4 on the next page. Both PSNR and SSIM are better in the augmented case, while LPIPS and FID are slightly worse. Usually PSNR and SSIM are less sensitive to overall image blurriness, and this would suggest



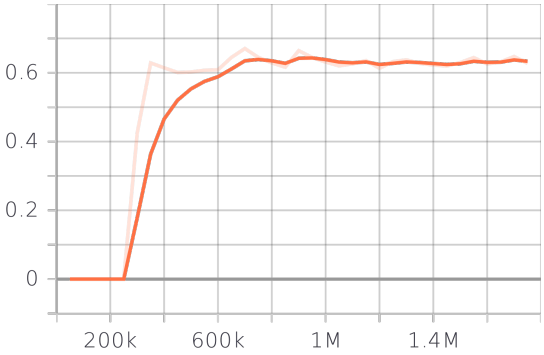
(a) Average discriminator output when classifying generated images (0 equals perfect classification). In orange●: with augmentation. In blue●: without augmentation.



(b) Average discriminator output when classifying real images. In orange●: with augmentation. In blue●: without augmentation.



(c) LPIPS metric (lower is better). In orange●: with augmentation. In blue●: without augmentation.



(d) r_t overfitting metric.

Figure 4: Plots of training behaviour and evaluation metrics on the non-subsampled FFHQ dataset, with and without augmentation. Semitransparent lines are the original data points, while opaque lines are their exponential moving average with a smoothing factor of 0.6, and provided to aid understanding. All the FFHQ subsamplings show similar behaviour, and are thus omitted.

that the generated images in the augmented case would be blurrier. However, we could not find any significant difference from a qualitative point of view between the two sets of generated images (see figure 5 on the following page). We hypothesize that this behaviour is mainly due to the very regular structure of the images in the dataset, and that the model is capable of learning the underlying data distribution correctly and with high accuracy, whether with augmentation or without. A key marker of successful application of the augmentation pipeline is the stabilization of the r_t metric around its threshold value, as we can see in figure 4.
The selection criteria for choosing among models of different epochs is the following: we select the model which perform better on LPIPS metric on the validation set.

Dataset Percentage	Data Aug	Metric FID	LPIPS	PSNR	SSIM
100%	**No**	2.62	0.0962	82.94	0.826
50%	**No**	2.91	0.0981	82.85	0.825
25%	**No**	2.81	0.1008	82.80	0.826
10%	**No**	2.99	0.1002	82.72	0.825
5%	**No**	2.71	0.0984	82.99	0.827
1%	**No**	2.69	0.1002	83.08	0.828
100%	**Yes**	2.87	0.0981	83.51	0.834
50%	**Yes**	2.93	0.1010	83.38	0.837
25%	**Yes**	2.81	0.1007	83.41	0.831
10%	**Yes**	3.10	0.1024	83.21	0.834
5%	**Yes**	2.81	0.0994	82.98	0.831
1%	**Yes**	2.70	0.1001	83.02	0.830

Table 2: FFHQ experimental results varying dataset size

(a) Output example of model trained on full dataset without data augmentation

(b) Output example of model trained on full dataset with data augmentation

(c) Output example of model trained on 1% subsampling of dataset without data augmentation

(d) Output example of model trained on 1% subsampling of dataset with data augmentation

Figure 5: Qualitative performance comparison on FFHQ. Original downsampled images on the left, ground truth images in the center and generated images on the right

5.3 DIV2K Experimental setup

The setup we employed to train and evaluate our model was almost the same as when evaluating on the FFHQ dataset. The dataset if officially split into a 800 portion for training purposes and two 100 portions for validation and testing respectively. However, because the ground truth images for the testing split have not been released, we use the official validation set as a testing set. All networks were pre-trained for **3.2M** iterations and trained in an adversarial setting for **2.4M** iterations. The hyperparameters we used differ from the ones used in the training of FFHQ for:

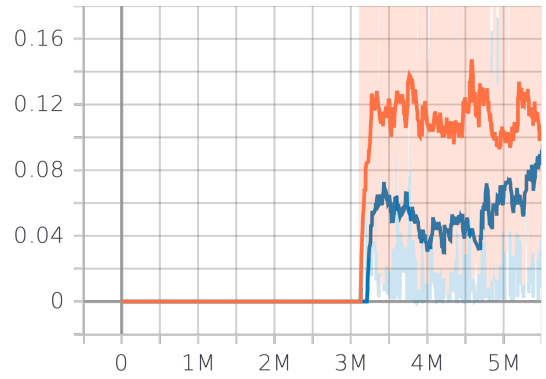
- ***p* update amount**: 5×10^{-4}
- **Adversarial loss balancing factor**: 6×10^{-5}

When training with the DIV2K dataset we perform one-sided label smoothing on ground truth images with a label value of 0.9 as well, and the metrics used for performance evaluation are the same.

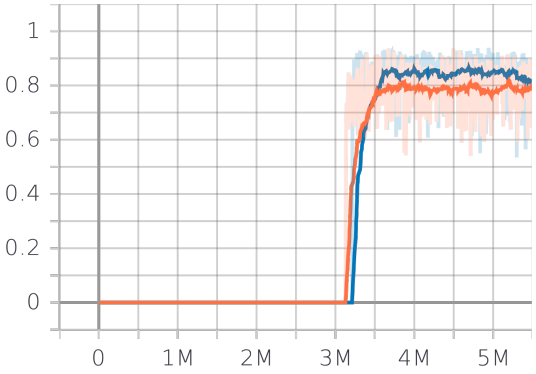
5.4 DIV2K Experimental results

Unlike the FFHQ dataset, which consisted in only faces, this much more varied dataset proved to be difficult for our model to correctly train on. As we can see in figure 6a and figure 7a on page 9), the generator struggles to learn the underlying data distribution and does not manage to fool the discriminator successfully.

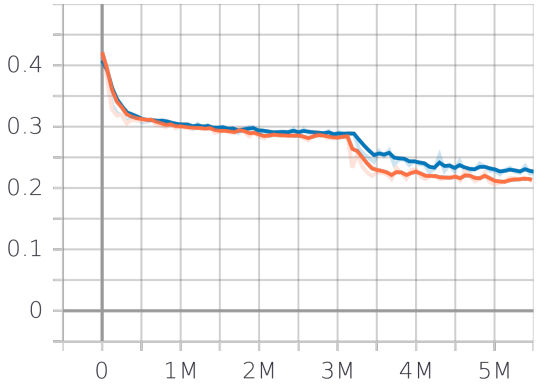
The original work[1] did suggest to cap the augmentation probability controlling the whole pipeline to a $0.85 - 0.9$ threshold in order to prevent leaking (this kind of leaking is due to finite numerical precision and finite sampling rather than from the algorithm per se, which from a mathematical standpoint is leaking-proof). In properly trained networks r_t should quickly stabilize to the chosen threshold value (in our case 0.6), however because our discriminator quickly overpowered the generator r_t was almost always above this value, causing an uncontrolled increasing in the augmentation probability through the whole training process. To avoid leaking in such a badly conditioned



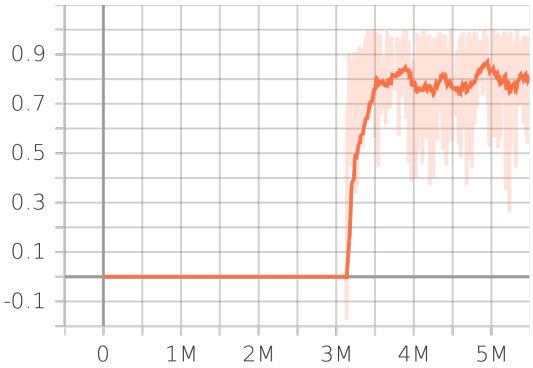
(a) Average discriminator output when classifying generated images (0 equals perfect classification). In orange●: with augmentation. In blue●: without augmentation.



(b) Average discriminator output when classifying real images. In orange●: with augmentation. In blue●: without augmentation.



(c) LPIPS metric (lower is better). In orange●: with augmentation. In blue●: without augmentation.



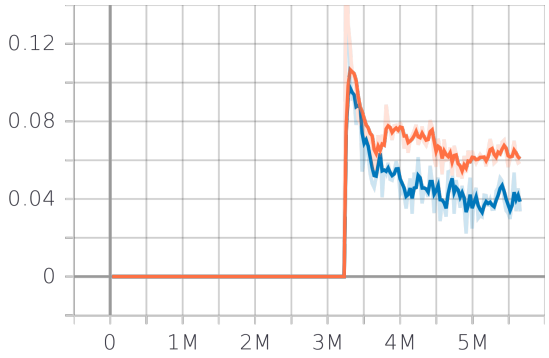
(d) r_t overfitting metric.

Figure 6: Plots of training behaviour and evaluation metrics on 1% subsampling of DIV2K dataset, with and without augmentation. Semitransparent lines are the original data points, while opaque lines are their exponential moving average with a smoothing factor of 0.6, and provided to aid understanding.

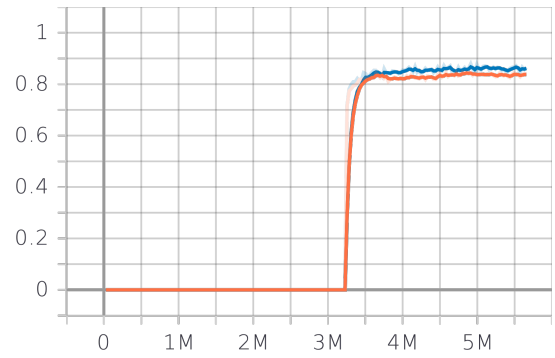
case, we clipped the augmentation probability to 0.85.

As we can see in figure 8 on page 10, the effect of data augmentation is negligible in a large dataset regime, while it becomes significant when training on very small datasets. This effect can be seen from figure 6c and figure 7c, where the gap between the LPIPS metrics with and without augmentation is more visible in the more aggressive subsampling. Moreover from figure 6a and figure 7a on the following page we can also notice that the generator manages to better fool the discriminator when augmentation is present. Another effect of this data augmentation scheme is that it makes the whole training procedure much more stable regarding both discriminator and generator dynamics. We can see this in figure 6 where all metrics have far less severe oscillations throughout the training.

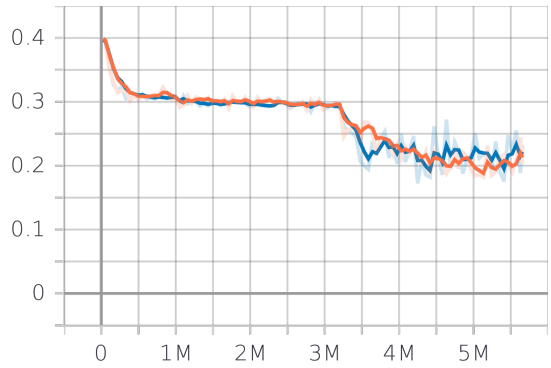
In table 3 on the following page are summarized our experimental results (note that the FID metric tends to be ill-conditioned with very low dataset sizes). Even though the effect of data augmentation on metrics is marginal when the subsampling percentage is relatively high, it becomes consistent when lowering the dataset size peaking at a 1.5 absolute improvement on the LPIPS metric when considering only 1% of data. We can also notice how much larger the FID and LPIPS metrics are when compared to the corresponding results on FFHQ, again suggesting how super resolution on the DIV2K dataset is harder than on FFHQ.



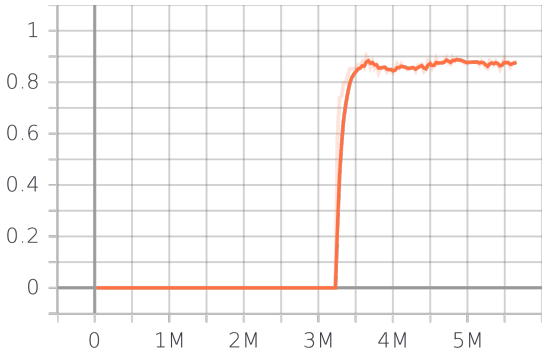
(a) Average discriminator output when classifying generated images (0 equals perfect classification). In orange●: with augmentation. In blue●: without augmentation.



(b) Average discriminator output when classifying real images. In orange●: with augmentation. In blue●: without augmentation.



(c) LPIPS metric (lower is better). In orange●: with augmentation. In blue●: without augmentation.



(d) r_t overfitting metric.

Figure 7: Plots of training behaviour and evaluation metrics on the non-subsampled DIV2K dataset, with and without augmentation. Semitransparent lines are the original data points, while opaque lines are their exponential moving average with a smoothing factor of 0.6, and provided to aid understanding.

Dataset Percentage	Data Aug	Metric FID	LPIPS	PSNR	SSIM
100%	**No**	66.25	0.1923	52.70	0.7503
50%	**No**	66.92	0.1893	52.61	0.7455
25%	**No**	67.08	0.2013	52.51	0.7428
10%	**No**	68.44	0.2024	52.40	0.7409
5%	**No**	68.35	0.2051	52.17	0.7296
1%	**No**	68.78	0.2265	50.36	0.6928
100%	**Yes**	66.44	0.1883	52.65	0.7434
50%	**Yes**	67.62	0.1926	52.67	0.7475
25%	**Yes**	67.17	0.1919	52.42	0.7423
10%	**Yes**	66.93	0.2011	52.30	0.7428
5%	**Yes**	67.14	0.2032	52.32	0.7432
1%	**Yes**	67.88	0.2101	51.03	0.7125

Table 3: DIV2K experimental results varying dataset size

6 Conclusions

In this work we presented and implemented a novel data augmentation technique tailored to Generative Adversarial Networks in order to reduce discriminator overfitting and stabilize training. We applied this technique to a super resolution task and evaluated the performance of a super resolution GAN in two different datasets. We showed how some modifications necessary to make this data augmentation technique not introduce artifacts in generated images. We also quantify how this data augmentation is more useful in very small datasets and loses efficacy the larger the dataset.

6.1 Resources

Code and further details available at `https://gitlab.com/reddeadrecovery/superaugan`

(a) Output example of model trained on full dataset without data augmentation

(b) Output example of model trained on full dataset with data augmentation

(c) Output example of model trained on 1% subsampling of dataset without data augmentation

(d) Output example of model trained on 1% subsampling of dataset with data augmentation

Figure 8: Qualitative performance comparison on DIV2K. Original downsampled images on the top left, ground truth in the top center and generated images on the top right. Under the three aligned images is shown a detail of the generated images

References

[1] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Training generative adversarial networks with limited data, 2020.

[2] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network, 2017.

[3] Wenzhe Shi, Jose Caballero, Ferenc Huszár, Johannes Totz, Andrew P. Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video

super-resolution using an efficient sub-pixel convolutional neural network, 2016.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[5] Zhengli Zhao, Sameer Singh, Honglak Lee, Zizhao Zhang, Augustus Odena, and Han Zhang. Improved consistency regularization for gans, 2020.

[6] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks, 2019.

[7] Andrey Ignatov, Radu Timofte, et al. Pirm challenge on perceptual image enhancement on smartphones: report. In *European Conference on Computer Vision (ECCV) Workshops*, January 2019.

[8] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[9] Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric, 2018.

[10] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

Appendices

A Other network architectures

Our original network architecture had the following modifications:

- It used tanh as activation function on the final layer
- It did use a learning rate scheduler, both for the generator and the discriminator
- The generator used *VGG loss* (equation (1) on page 3) as a content loss
- It did not use one-sided label smoothing

In this section we describe how in our experiments each of these features (or lack thereof) prevented our model to achieve its full performance capabilities.

While the use of hyperbolic tangent as a final activation function is very popular in modern GAN setups, we found that (a) the removal of the activation function does improve the overall output of the model, and (b) while the hyperbolic tangent allows the output domain to be bounded and thus easier to process (specifically it corresponds to the tanh codomain, *i.e.* $[-1, 1]$), the model is itself more than capable in learning the expected output domain (in our case $[0, 1]$) without any activation function.

The use of *VGG loss* at best introduced tiny artifacts around image details (see figure 9 on the following page), and in the worst cases deeply influenced the generation process, altering image colors in wide patches (mostly preserving gradients and content information). This effect was independent of feature map choice and model choice (*e.g.* VGG16 vs VGG19). Also, the model's behaviour didn't seem to improve even after many iterations, suggesting that something fundamental about training this super resolution model is at play. We hypothesize that the *VGG loss* doesn't provide enough signal for the generator to pick up in order to successfully generate coherent images. This could be a limitation of our model (especially when compared to other models with more parameters), as others has shown the inverse tendency with regard to this loss function.

One-sided label smoothing did slightly improve the convergence speed of the generator without affecting any other metric (either losses or evaluation metrics).

On the other hand, the two learning rate schedulers, originally suggested in [2] did not provide enough significant benefit to include them in our final model.

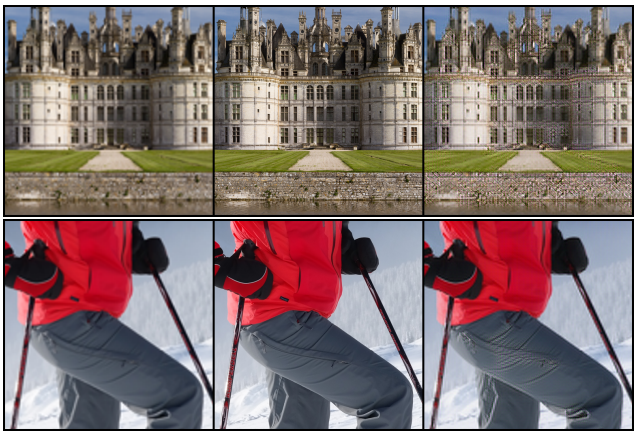


Figure 9: Artifacts introducted when using VGG loss as perceptual loss

B Modifications to the original pipeline

Even though the pipeline described in [1] did include both image filtering and arbitrary angles rotation, we omitted these steps from our pipeline. In our initial experiments we observed that the images produced by the generator did include some diagonal and grid-like high-frequency artifacts, especially around hair and teeth.

We hypothesized that these artifacts were due to either image rotation or image filtering (we thought of the former because of the orientation, always diagonal, and the latter due to the regular structure of these artifacts). In order to find which of these two augmentations was causing these artifacts we tried removing one of them at a time and found that both these two augmentations, even when taken singularly, were the cause. The artifacts produced when only one of them was active were slightly less noticeable, but there was no significant difference suggesting that any of the two had a more pronounced effect than the other. In figure 10 are displayed two examples of such artifacts.

(a) Augmentation artifacts on hair mostly introduced by arbitrary angles image rotation (rightmost part of the image

(b) Augmentation artifacts on teeth mostly introduced by image filtering (center of the image)

Figure 10: Augmentation artifacts introduced by arbitrary angle rotation and image filtering