DEQUANTIZERNET

GENERATIVE ADVERSARIAL NETWORK FOR IMAGE QUALITY RESTORATION

Margarita Geleta & Pablo Martin Redondo
Department of Computer Science
University of California at Irvine
Irvine, CA 92697, USA
{mgeleta,pmartinr}@uci.edu
21804939,

ABSTRACT

Lossy compression algorithms for digital images, commonly used in photography, such as JPEG, may introduce compression artifacts, as a consequence of quantization. In particular, these compression artifacts downgrade the image quality through posterization and loss of image detail. In this work, we introduce **DequantizerNet**, a super-resolution imaging-based generative adversarial neural network for quality restoration of highly degraded JPEG images.

1 INTRODUCTION

In the last years, we have seen models that are able to generate details out of data. These include applications such as: super-resolution, coloring, video frame rates augmentation, etc. The potential of this approach is enormous. Being able to generate details out of undetailed, fuzzy or noisy information opens an avenue to reconstruct the lost information of data. This leads directly to an optimization opportunity for data storage and transmission. Why would data be stored in its original state when it can be compressed and then decompressed when it is needed? Why would 1 GB of pictures be sent when 100 MB can be sent and use such model to restore their quality? The focus of this project is data reconstruction, specifically in image reconstruction or dequantization, for the purpose of image quality restoration.

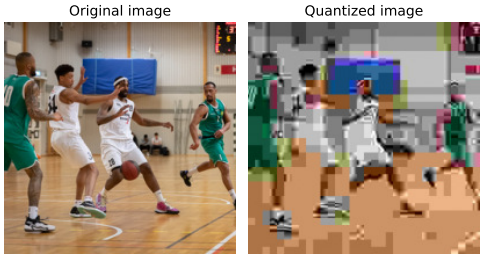


Figure 1: An example of our dataset images.

Compression is a very settled topic. There are very settled algorithms to compress images. The most common digital compression artifacts are DCT blocks, caused by the discrete cosine transform (DCT) compression algorithm used in many lossy digital media standards, such as JPEG Wikipedia contributors (2022), extensively used in photography. Only a specific level of quantization is acceptable without an enormous loss of quality. Nevertheless, having a robust tool able to reconstruct quantized images, allows for the optimizing of both, storage and channel capacities. Moreover, this tool can be potentially used to create an image processing method for image quality enhancement in mobile pictures and web video conferences.

In this project, we aim to develop that tool. Using a generative adversarial network (GAN), we will train a model capable of inferring the details of quantized pictures with the JPEG compression algorithm. First of all, we will create our own dataset downloading pictures from the popular social

media Flickr. Following this, we will process the images. The goal is to have a dataset with quantized images as an input, and the original images downloaded from Flickr as an output. The next step will be to design a deep neural network able to learn how to add detail (dequantize) images and restore their quality. Our goal is to get to a robust model that is able to retrieve the losses caused by JPEG compression. The final step will be to join all together in a tool that has a very simple API: quantize and dequantize.

2 Related Work

Starting from the definition, image super-resolution (abbreviated as SR) is the process of recovering high-resolution images, $\mathcal{I}$, from low-resolution images, $\hat{\mathcal{I}}$. Even though, quantized images are not exactly low-resolution images as defined in the SR works, we will adopt SR models for our task.

2.1 Frameworks

In the literature there exist several approaches for SR. One of the first ideas in the field has been **pre-upsampling super resolution** (SRCNN by Dong et al. (2015); VDSR by Kim et al. (2015)), which consists in the application of upsampling algorithms (such as bicubic interpolation) followed by deep convolutional neural networks (CNNs) for feature extraction and refinement. Later on, learnable upsampling layers have been introduced (i.e., deconvolutions by Long et al. (2014)) into SR architectures. *Very Deep Super Resolution* (VDSR) is an improvement on SRCNN, with the addition of residual connections, i.e., instead of learning the direct mapping from $\hat{\mathcal{I}}$ to $\mathcal{I}$ like SRCNN, VDSR learns the residual so that the initial low resolution image is added to the network output to get the final high resolution image (Kim et al., 2015).

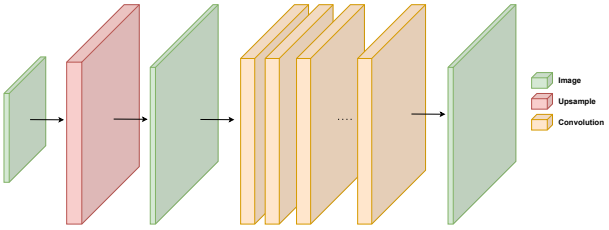


Figure 2: Pre-upsampling super resolution model.

Because the feature extraction process in pre-upsampling SR tackles inputs of full resolution, the computational expense is relatively large. **Post-upsampling super resolution** (FSRCNN by Dong et al. (2016); ESPCN by Shi et al. (2016)) is the next framework introduced to mitigate the latter issue. Essentially, raw low-resolution images $\hat{\mathcal{I}}$ are fed into deep CNNs to perform the feature extraction in the lower resolution space and, then, upsampling layers are applied at the end, significantly reducing computation.

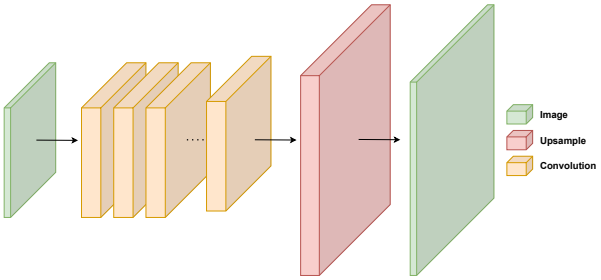


Figure 3: Post-upsampling super resolution model.

Even though the previous framework, overall, reduces the computational complexity, using just a single upsampling layer makes the learning process harder for large scaling factors. **Progressive**

upsampling super resolution (LapSRN by Lai et al. (2017); Progressive SR by Wang et al. (2018)) is another framework which addresses the aforementioned issue. This family of models makes use of a cascade of CNNs to progressively reconstruct high resolution images at smaller scaling factors at each step.

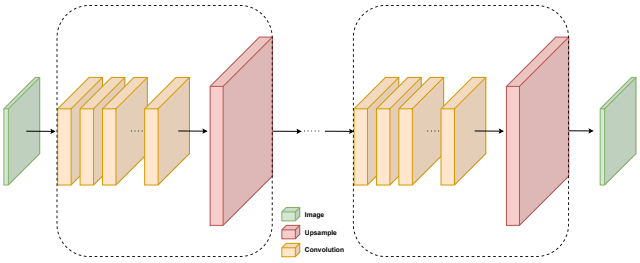


Figure 4: Progressive upsampling super resolution model.

Finally, another approach for addressing SR is the **Iterative Up and Down Sampling super resolution** (inspired by U-Net by Ronneberger et al. (2015) and Stacked Hourglass network by Newell et al. (2016)). Initially, the design of the hourglass has been motivated by the need to capture information at every scale in human pose estimation Newell et al. (2016), because local details were essential for identifying features like faces and hands, but a final pose estimate required an understanding of the full body. The same argument can be applied to SR tasks. SR models under this framework can define better the relationship between $\mathcal{I}$ and $\hat{\mathcal{I}}$.

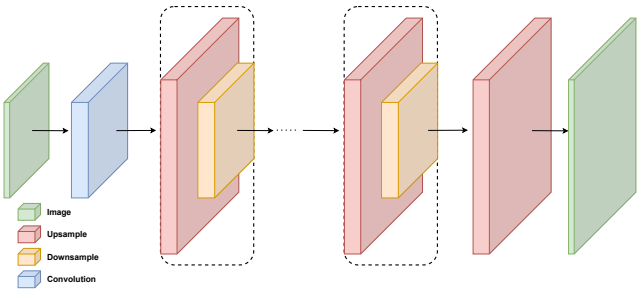


Figure 5: Iterative Up and Down Sampling super resolution model.

2.2 Loss functions

A loss function $\mathcal{L}$ is used to measure the difference between the ground truth high-resolution image $\mathcal{I}$ and the SR model (denoted as f) output $f(\hat{\mathcal{I}})$, which is the low-resolution or quantized image $\hat{\mathcal{I}}$ with restored quality. A loss function is used to optimize upon, this way tuning the model f parameters. Specifically for SR, several kinds of loss functions do exist, each of them is crafted to penalize a different aspect of the generated image. In general, several loss functions are combined into one by weighting each term and summing up all of them. This weighted sum of individual loss functions enables the model to learn different enforced aspects simultaneously.

Given target image I and predicted image $\hat{\mathcal{I}}$ of width W and height H, we will define the loss of a sample $\mathcal{L}$ as the score to measure the difference between them.

Pixel Loss – these loss functions compute the difference between each pixel ground-truth image compared to each pixel in the generated image. Examples of pixel loss functions include L_1, L_2 and Smooth L_1.

$$\mathcal{L} = \frac{1}{W \cdot H} ||\mathcal{I} - \hat{\mathcal{I}}||_p^p \tag{1}$$

Content Loss or Perceptual Loss – this loss functions evaluates the image quality based on its perceptual quality. One way to make it possible is comparing the high level features (of a pre-trained image classification network, such as VGG by Simonyan & Zisserman (2014) or ResNet by He et al. (2015)) of the generated image and the ground truth image. This loss function encourages the generated image to be perceptually similar to the ground-truth image. Given $\mathcal{VGG}_{f_i}$ the output of the VGG network at feature layer f_i:

$$\mathcal{L} = \frac{1}{W \cdot H} ||\mathcal{VGG}_{f_i}(\mathcal{I}) - \mathcal{VGG}_{f_i}(\hat{\mathcal{I}})||_p^p \tag{2}$$

Texture Loss – this loss function enforces the generated image to have the same style as the ground truth image in terms of texture, color, contrast, etc. One method to enable this is by optimizing over the correlation between different feature channels (Gatys et al., 2016), which are usually obtained from a feature map extracted using a pre-trained image classification network (e.g., VGG or ResNet).

$$\mathcal{L} = 1 - |\rho(\mathcal{VGG}_{f_i}(\mathcal{I}), \mathcal{VGG}_{f_i}(\hat{\mathcal{I}}))| \tag{3}$$

Total Variation Loss – this loss function is used to suppress noise in the generated images (i.e., make the regions more homogeneous). A way to compute it is to take the sum of the absolute differences between neighboring pixels. Given the pixel neighborhood $\mathcal{N}$ and total variation $\mathcal{V}$:

$$\mathcal{V}(\mathcal{I}) = \sum_{i,j \epsilon \mathcal{N}} ||x_i - x_j||_p^p \;\; \mathcal{L} = \frac{1}{W \cdot H} ||\mathcal{V}_{f_i}(\mathcal{I}) - \mathcal{V}_{f_i}(\hat{\mathcal{I}})||_p^p \tag{4}$$

Adversarial Loss – in the case of using a Generative Adversarial Network (GAN) by Goodfellow et al. (2014), a model is usually trained with adversarial loss, which includes losses of the two networks which compose the GAN system: the generator (G) and the discriminator(D).

$$\mathcal{L} = -\log D(G(\hat{\mathcal{I}})) \tag{5}$$

2.3 METRICS

To quantitatively assess the performance of our model, perceptual metrics are used in image tasks, two of which we use in this work and them introduce next:

Peak Signal-to-Noise Ratio or PSNR – it measures the reconstruction quality of images subject to a lossy compression. PSNR is inversely proportional to the logarithm of the Mean Squared Error (MSE) between the ground truth image and the generated image. Given MSE the mean squared error and MAX_I the maximum value a pixel can take, we define the PSNR as:

$$\mathcal{PSNR} = 10 \log_{10} \left(\frac{MAX_I^2}{MSE(\mathcal{I}, \hat{\mathcal{I}})} \right) \tag{6}$$

Structural Similarity or SSIM – is a subjective metric used for measuring the structural similarity between images, based on three relatively independent comparisons, namely luminance, contrast, and structure, computed as the weighted product of the aformentioned terms. Let represent two windows of NxN of the images $\mathcal{I}$ and $\hat{\mathcal{I}}$ as x and y, and let be μ the average, σ_i the variance and σ_{ij} the covariance, the SSIM is defined as:

$$\mathcal{SSIM} = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{7}$$

Where $c_1 = (k_1 \cdot L)^2$ and $c_2 = (k_2 \cdot L)^2$ are stabilizers of the formula, with L being the range of the pixel values and $k_1 = 0.01$ and $k_2 = 0.02$.

3 METHODOLOGY

3.1 DATASET

We generated our own dataset. For this matter, we used the available public API of the social network Flickr. We downloaded pictures out of 1000 different categories (extracted from ILSVRC

2012 by Krizhevsky et al. (2012)). For each category, we downloaded 10 pictures. The way we do this is by requesting a search of the given category to the Flickr API to then retrieve the 10 most recent images of the category.

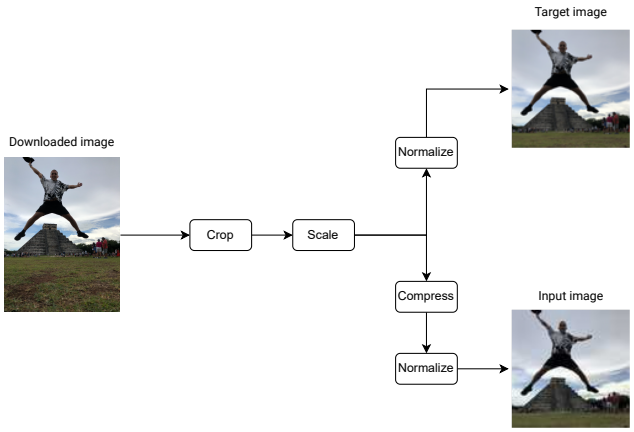


Figure 6: Image processing.

Once we have our images, we crop them to be squared and we resize them to have a size of 96×96 pixels. These images will be the output images to predict. For the input we compress the image with the JPEG compression algorithm. The final size of the dataset is of 10,000 images, 80% (8,000 images) will be used for training and 20% for test (2,000 images) for the testing phase.

For our experiments, we created two separated datasets. They differ in the quality the image is compressed in. In one, images are compressed to 1% of their quality, while in the other one they are compressed to 50% of it.

3.2 Architecture

Taking current state-of-the art super-resolution architectures as our starting point, our model is based on a Generative Adversarial Network Goodfellow et al. (2014). In this kind of architectures, two models compete with each other and, in the process, they learn. A generator generates the information that will be served as the input with the intention of fooling it. The discriminator will be trained in order to distinguish between real information and generated one. The output of both generator and discriminator will be used in order to evaluate the performance of the models and train them in the next step of the learning algorithm.

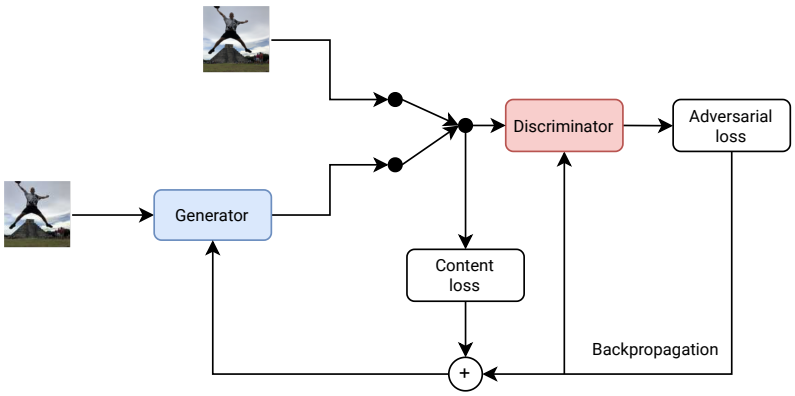
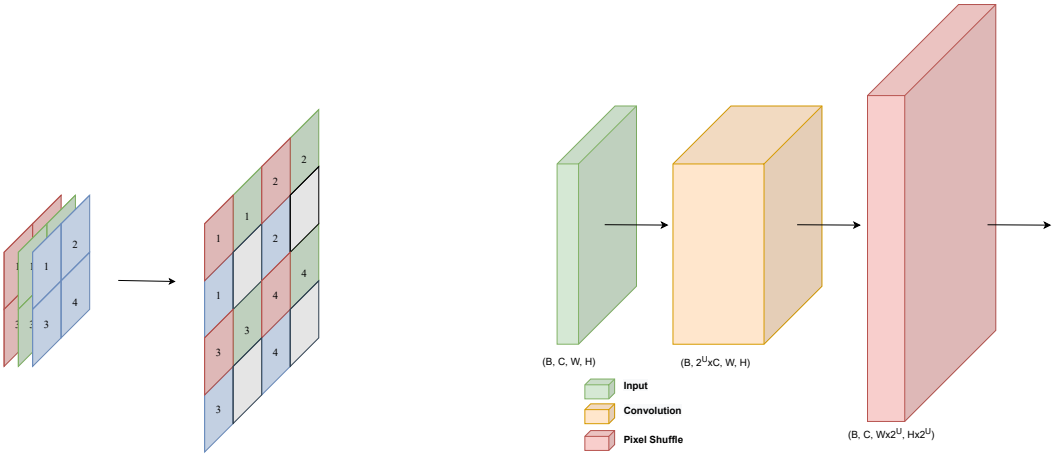


Figure 7: Our Generative Adversarial network for image decompression.

3.2.1 GENERATOR

The generator is a Post-upsampling Deep Convolutional Neural network. In order to make the network deep, we use Residual connections. The Upsample Block is based on the sub-pixel convolution operation, known in Pytorch as pixel shuffle, that combines the channels to convert depth to space (Shi et al., 2016). Hence, to upsample we first increase the depth of the feature map and then we compress the channels, increasing the width and height. The number of channels in the input and output of the block are kept invariant after combining both operations.



(a) Pixel Shuffle.

(b) Upsample block.

Other main building blocks of the generator are the Residual Blocks. These blocks maintain invariant the dimensions of the feature map at the input while performing non-linear operations (convolution and normalization) to its values. It includes a residual connection that performs an addition operation of the input and output of the block.

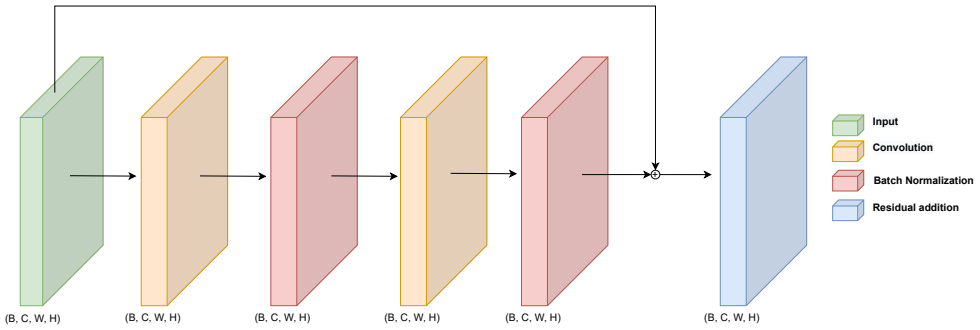


Figure 9: Residual block.

The architecture of the generator is a sequence that includes a convolutional layer, followed by a parameterized number of Residual Blocks. After that, we have another convolutional layer which output is fed into a parameterized number of upsampling blocks. The output of this block is joined with the output of the first convolutional layer with a residual connection and finally we add a convolutional layer. This last layer reduces the channels to the ones of the generated image, that it is served as the output of the generator.

Specifications of the number of blocks used for the residual convolutional phase and the upsampling phase will be determined in the Experiments section. The transfer functions are also parameterized in the code. Nevertheless, in practice we have only used one. For both the Residual and UpSampling blocks we use the Parametric Rectified Linear Unit, or PreLu. This is a Leaky ReLu which slope in the negative side is also a parameter to be trained. At the output of the generator use the Hyperbolic Tangent.

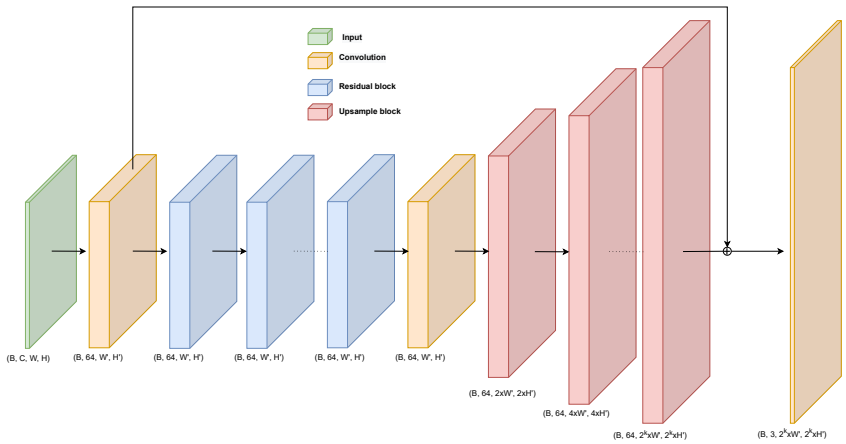


Figure 10: Generator.

3.2.2 DISCRIMINATOR

As the discriminator we have a classical Deep Convolutional Neural Network. The architecture is a sequence of convolutional layers and batch normalization layers that squeeze the width and the height of the features while increasing the channels. In the end, we use average pooling to reduce the dimensions and flatten the features to output a number. This number is the probability of an image being fake or real.

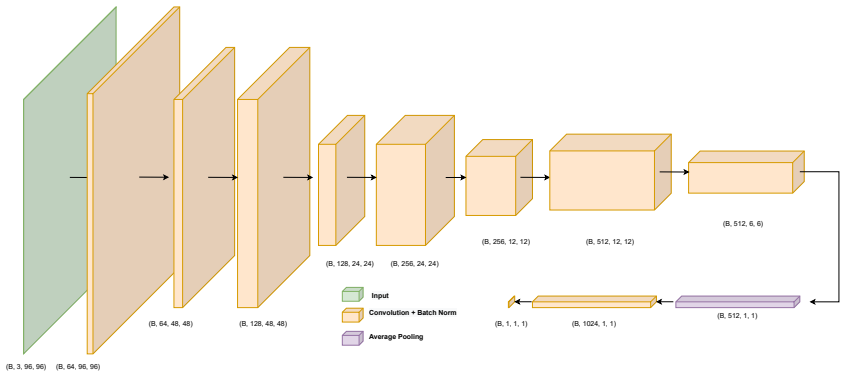


Figure 11: Discriminator.

As in the generator, transfer functions are parameterized and can be change from a configuration file. Nevertheless, due to the time limit of the project we have only used one. The transfer function used between layers have been the Leaky ReLu, while for the output we uses a Sigmoid.

3.3 TRAINING

The source code has been developed in Python using the Pytorch Deep Learning framework. Git, as a version control system, has been used to track the code base development, and the project itself is hosted on GitHub and is publicly accessible[1]. Besides, *Weights & Biases* (wandb) has been used as a complementary set of tools for model versioning, hyper-parameter optimization and experiment tracking.

As explained in the *Related Work* section, in general, in SR, several losses are combined into one as a weighted sum of individual terms, to enforce the model to learn different aspects simultaneously.

[1] `https://github.com/margaritageleta/dequantizerNet`

To train our models we opted for a sum $\mathcal{L}$ of two terms, weighted by α and β, respectively:

$$\mathcal{L} = \alpha \ell_{\text{ADV}} + \beta \ell_{\text{CONTENT}} \tag{8}$$

Where ℓ_{ADV} is the adversarial loss, which encourages the model to generate images close to the real ones with the aim to trick the discriminator, and where ℓ_{CONTENT} is the content loss, which encourages the model to generate images perceptually similar to ground-truth images. ℓ_{ADV} is computed as binary cross entropy (BCE) between the real labels and the predicted labels, the more the discriminator is tricked (i.e., outputs predictions as real images), the smaller is the BCE. ℓ_{CONTENT} is computed by forwarding the target and dequantized images through a VGG Simonyan & Zisserman (2014), extracting the 35^{th} feature map and finally computing the mean squared error between both feature maps. Adam optimizer Kingma & Ba (2014) has been used for traversing the loss function surface.

Our models have been trained on NVIDIA Tesla T4 (16GB) GPU in Google Colaboratory (referred to as *GColab*). Because our models required a large amount of time to overcome underfitting and because we had time constraints on GColab, we implemented a checkpointing system, which consists in: (1) in each validation procedure, we compute the PSNR metric on the validation split; if that current epoch value is better than the best so far, checkpoints of the discriminator, generator and their respective optimizers are stored in disk; (2) if we restart the training (it may happen for several reasons, e.g., GColab time limit exceeded), we can resume the training retrieving the stored checkpoints. For that reason, for each run we specify an experiment name in the configuration file. We use the following convention: for highly quantized data, we use hebrew letter names and for moderately quantized data we use greek letter names.

4 Experiments and Results

We have trained all our models during 50 epochs, with a batch size $\mathcal{B} = 16$, using a training rate of $\alpha = 10^{-4}$, $\mathcal{U} = 0$ upsampling layers and the Adam optmizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We have used both data compressed to their 50% and to their 1% using the standard JPEG compression. We have done experiments with $\mathcal{R} = [5, 10]$ residual blocks with very similar results.

NAME	QUANT. LEVEL	$\mathcal{R}$	TEST SSIM	TEST PSNR
Alef	1%	5	0.4302	14.985
Beta	50%	5	0.8732	25.578
Gimel	1%	10	0.4067	16.005
Phi	$50\%^*$	10	0.9141	27.651

* Experiment Phi has been trained on 50% compressed data using the Gimel checkpoint weights as a starting point.

Table 1: Perceptual metrics of our experiments. Refer to the Appendix to see the visual results of Alef and Beta experiments.

Due to space limits we cannot elaborate in the perceptual results of each models. Nevertheless, some outcomes are offered in the Appendix.

5 Discussion and Conclusion

In this project we unraveled the power of GANs for image dequantization. The results are surprisingly perceptually good. Even though the input images are tiny 96×96 crops and, thus, lack resolution per se, our dequantization network is capable of smoothing out the compression artifacts and DCT blocks induced by JPEG compression and, in relatively low-resolution, restore some of the lost details and colors. We had limited computational resources and time, we are sure that if we had larger allowances, we could push even further the perceptual metrics. But even with these constraints, these results suggest that dequantization is an operation that can be learned and open up an avenue for further research.

As next directions in this niche research, we envision: (1) architectural changes, deeper networks; (2) a more complex loss function, including additional feature-enforcing terms; (3) include other compression corruptions, to make the model robust not only to JPEG compression; (4) increase the size of the input images.

6 ACKNOWLEDGMENTS

Both participants: Margarita Geleta and Pablo Martin Redondo, collaborated to develop this project and contributed to it equally.

REFERENCES

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks, 2015. URL https://arxiv.org/abs/1501.00092.

Chao Dong, Chen Change Loy, and Xiaoou Tang. Accelerating the super-resolution convolutional neural network, 2016. URL https://arxiv.org/abs/1608.00367.

Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2414–2423, 2016. doi: 10.1109/CVPR.2016.265.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks, 2015. URL https://arxiv.org/abs/1511.04587.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Wei-Sheng Lai, Jia-Bin Huang, Narendra Ahuja, and Ming-Hsuan Yang. Deep laplacian pyramid networks for fast and accurate super-resolution, 2017. URL https://arxiv.org/abs/1704.03915.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation, 2014. URL https://arxiv.org/abs/1411.4038.

Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation, 2016.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

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. URL https://arxiv.org/abs/1609.05158.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. URL https://arxiv.org/abs/1409.1556.

Yifan Wang, Federico Perazzi, Brian McWilliams, Alexander Sorkine-Hornung, Olga Sorkine-Hornung, and Christopher Schroers. A fully progressive approach to single-image super-resolution, 2018. URL https://arxiv.org/abs/1804.02900.

Wikipedia contributors. Discrete cosine transform — Wikipedia, the free encyclopedia, 2022. URL https://en.wikipedia.org/w/index.php?title=Discrete_cosine_transform&oldid=1085819228. [Online; accessed 10-May-2022].

A Appendix

Figure 12: Our model Alef is able to reconstruct some lost details (e.g., eyes).

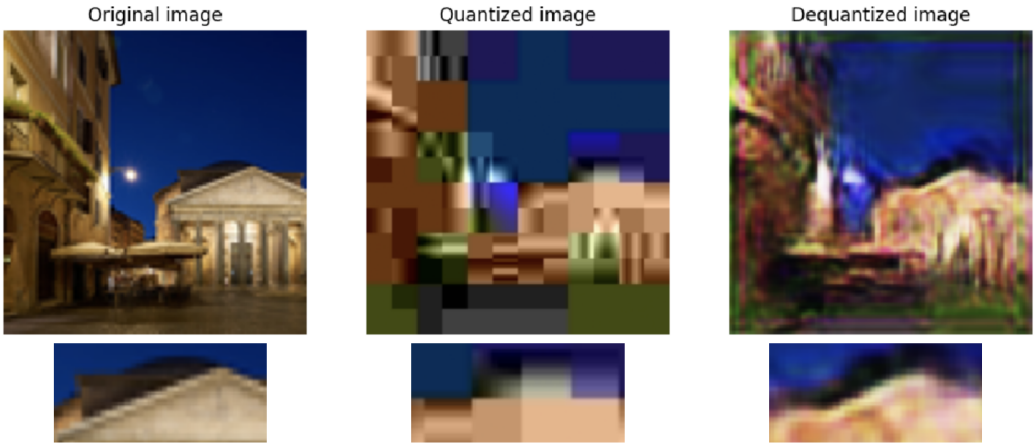


Figure 13: Our model Alef is able to reconstruct some lost details (e.g., edges).

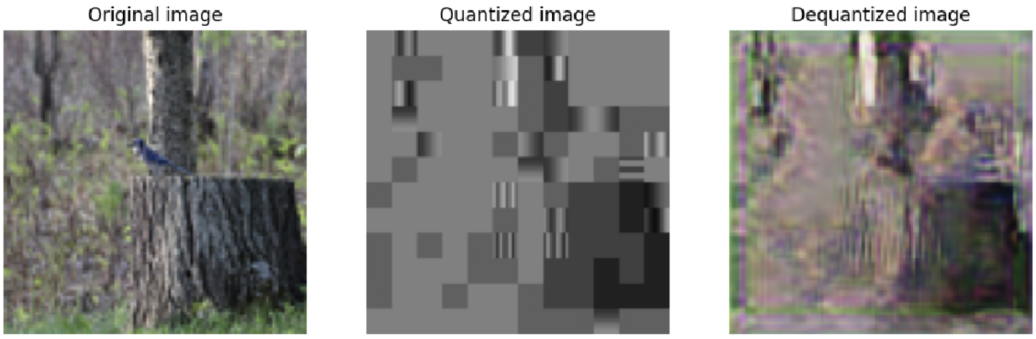


Figure 14: Our model Alef is capable to restore some lost colors during quantization.

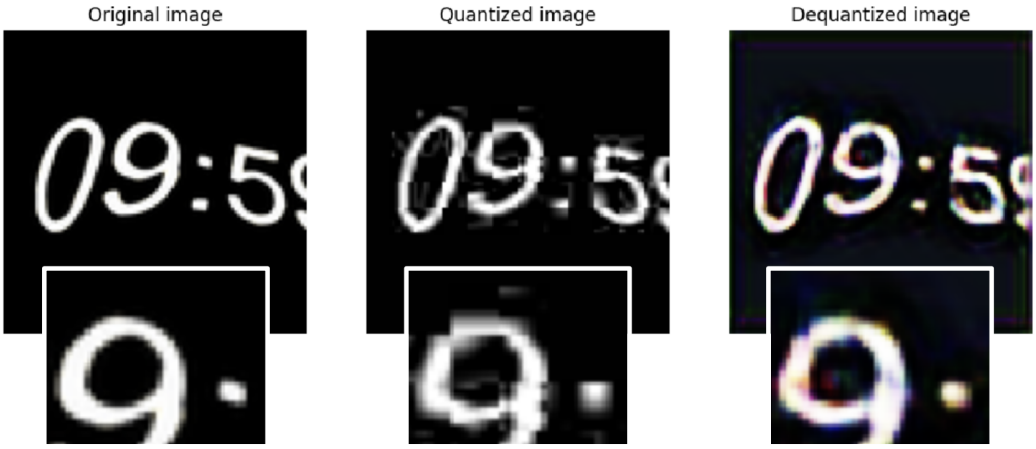


Figure 15: Our model Alef smoothes the edges resulting from DCT blocks.

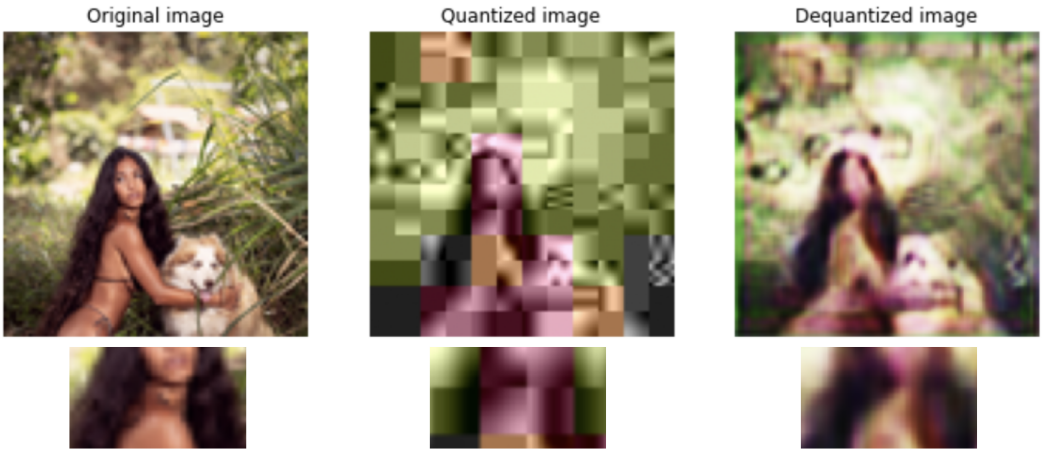


Figure 16: Our model Alef smoothes the edges resulting from DCT blocks.

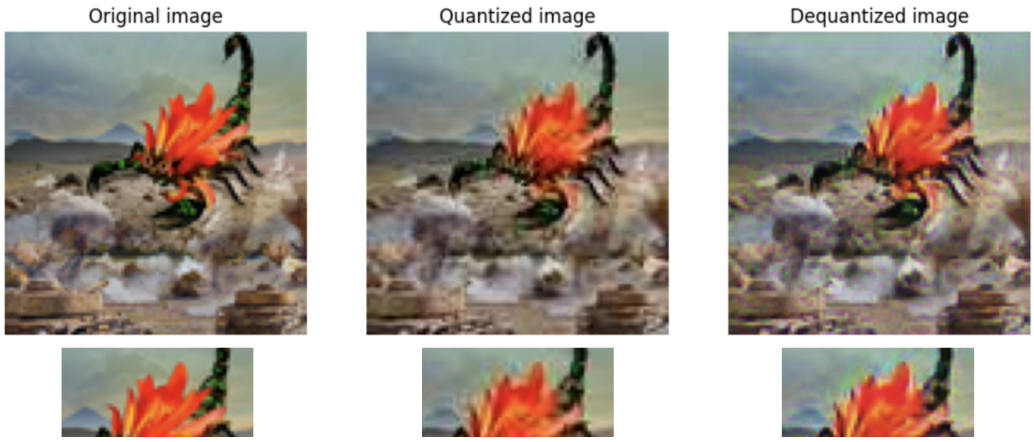


Figure 17: Our model Beta sharpens the edges to increase the resolution.

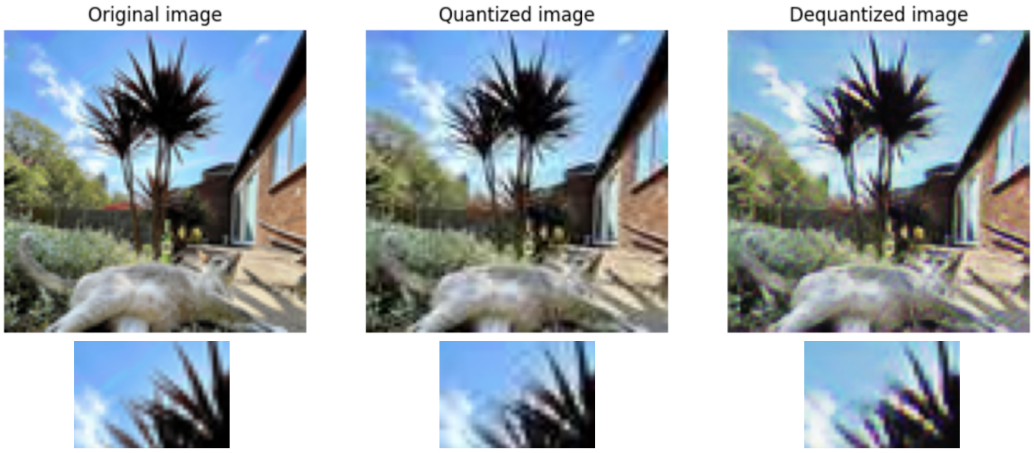


Figure 18: Our model Beta removes the DCT blocks by smoothing.