INFO8010: Report template

Julien Gustin,[1] **Lucas Michel**,[2] and **Joachim Houyon**[3]
[1] julien.gustin@student.uliege.be (s180337)
[2] lucas.michel@student.uliege.be (s170492)
[3] joachim.houyon@student.uliege.be (s181539)

I. INTRODUCTION

In the early days of photography, the images captured by a camera were only in black and white. Given such a grayscale image, it is usually an easy task for a human to predict color of each object appearing in the picture with a strong degree of confidence. For example, if one recognizes grass on a certain area of a photo, one can predict the color of this area as green. Throughout history, many works have been done to manually add plausible colors to monochrome photographs. The main goal of this project is to automate such a procedure.

Given a grayscale image, we want to automatically colorize it without any user input. To achieve so, we will build and train a *conditional generative adversarial network* (cGAN) from scratch. Our neural network has to understand the different elements present on an image (segmentation) and recolor them in a coherent way for a human. We will start from an architecture that has already proven itself from existing works (cfr. [1]). We will first understand the architecture in depth, then try variations in order to make it ours and improve it.

After a short panoramic view of the current method used to tackle such image colorization problem, we will introduce cGAN. Then, we will present both generator and discriminator architecture used to build our cGAN. After that, we will present some *original*[2] ideas (or tricks) that we've tried to implement in the hope of getting better results. Finally, we give our results together with a qualitative and quantitative analysis. As several ideas of potential improvement came into our mind during the whole project, we conclude this paper by a *To go further* section.

II. RELATED WORK & BACKGROUND

There exist various approaches to tackle the re-colorization task, one of them is proposed in the paper [3]. Given an input lightness $\mathbf{X} \in \mathbb{R}^{H \times W \times 1}$, one could think that the goal is to learn a mapping $\mathcal{F}$ that sends X to $\hat{\mathbf{Y}} = \mathcal{F}(\mathbf{X})$ to the two associated color channels $\mathbf{Y} \in \mathbb{R}^{H \times W \times 2}$, which is the ground truth. A natural objective function would be the L_2 loss. However, this loss lacks of robustness as it would favor to guess the mean of a set of colors, giving grayish, desaturated results. Instead, the problem is treated as a multinomial classification. In other words, we must learn a mapping $\mathcal{G}$ that sends X to $\hat{\mathbf{Z}} = \mathcal{G}(\mathbf{X})$ to a probability function over possible colors $\hat{\mathbf{Z}} \in [0,1]^{H \times W \times Q}$, where Q is the number of quantized ab values. The convenient loss would be the multinomial cross entropy loss, where the loss is strongly engineered such that uses weights that help to rebalance the loss based on the rarity of a color class. The trained architecture is a VGG-styled network with added depth and dilated convolutions.

Another way to tackle the colorization task is given by the article [1]. They solve, in a general fashion, image-to-image translation problems by using conditional generative adversarial networks (cGAN) as a general-purpose solution. These networks not only learn the mapping from input image to output image, but also learn a loss function to train this mapping.

We chose to follow the general solution given by the latter article and apply it to our colorization problem. Indeed, we want to learn a mapping *grayscale image* to *colorful image*.

In the next subsection, we explain the necessary theory concerning cGAN.

A. Conditional generative adversarial network (cGAN)

Generative adversarial networks (GAN) are generative models introduced in the paper [4]. A GAN consists of two 'adversarial' models:

- a generative model G (the *generator*) that captures the data distribution,
- a discriminative model D (the *discriminator*) that estimates the probability that a sample came from the training data rather than G.

The generator G (resp. discriminator D) belongs to a family of function indexed by a parameter $\theta_G \in \mathbb{R}^g$ (resp. $\theta_D \in \mathbb{R}^d$). Typically, G and D are neural networks and θ_G and θ_D denote their respective set of parameters.

More formally, on the same probability space, we consider

- a random variable z of codomain $\mathcal{Z}$ (a latent space) and of probability distribution p_Z,
- a random variable y of codomain $\mathcal{Y}$ (the data space) and of probability distribution p_Y.

The random variable Z is used as noise. The generator is an application

$$G(\cdot;\theta_G) : \mathcal{Z} \to \mathcal{Y}$$

inducing another distibution q_{θ_G} on $\mathcal{Y}$ by

$$y \sim q_{\theta_G} \iff z \sim p_Z \text{ and } y = G(z;\theta_G).$$

The discriminator is an application

$$D(\cdot;\theta_D) : \mathcal{Y} \to [0,1]$$

whose role is to distinguish between true samples $y \sim p_Y$ and generated sample $y \sim q_{\theta_G}$. The closer $D(y;\theta_D)$ is close to 1 (resp. 0), the more the discriminator classify y as a true (resp. generated) sample. We train the discriminator to maximize the expectation of assigning 1 (resp. 0) to true (resp. generated) inputs. Simultaneously, we train the generator in order to fool the discriminator as much as possible. If we consider the value function V defined as

$$\begin{aligned} V(\theta_G,\theta_D) =& \mathbb{E}_{y\sim p_Y}[\log(D(y;\theta_D))] \\ &+ \mathbb{E}_{z\sim p_Z}[\log(1 - D(G(z;\theta_G);\theta_D))], \end{aligned}$$

then we can describe the best generator as the solution of the two-player min-max game with value function V. In other words, we want to find

$$\theta_G^\star = \arg\min_{\theta_G}\max_{\theta_D} V(\theta_G,\theta_D).$$

In practice, one can hope to find an approximation of the solution by using an alternating gradient descent algorithm.

We now transpose what we know about GAN to introduce cGAN. In the conditional settings, we need to consider a new random variable X of codomain $\mathcal{X}$ and of probability distribution p_X. Both the generator and discriminator take as input an additional term x from $\mathcal{X}$. The role of this additional term x is to condition the generator and the discriminator to the information carried by this term. In the section III, we will see an example of how to use this exta term. The generator and discriminator are function of the form

$$\begin{aligned} G(\cdot,\cdot;\theta_G) &: \mathcal{X}\times\mathcal{Z} \to \mathcal{Y} : (x,z) \mapsto G(x,z;\theta_G), \\ D(\cdot,\cdot;\theta_D) &: \mathcal{X}\times\mathcal{Y} \to [0,1] : (x,y) \mapsto D(x,y;\theta_D), \end{aligned}$$

and the value function is now defined as

$$\begin{aligned} V(\theta_G,\theta_D) =& \mathbb{E}_{(x,y)\sim p_{X,Y}}[\log(D(x,y;\theta_D))] \\ &+ \mathbb{E}_{(x,z)\sim p_{X,Z}}[\log(1 - D(x,G(x,z;\theta_G);\theta_D))] \\ =& \mathbb{E}_{x\sim p_X}\big[\mathbb{E}_{y\sim p_{Y|x}}[\log(D(x,y;\theta_D))] \\ &+ \mathbb{E}_{z\sim p_{Z|x}}[\log(1 - D(x,G(x,z;\theta_G);\theta_D))]\big], \end{aligned}$$

where the second equality comes from the law of total expectation. As for GANs, we want to find

$$\theta_G^\star = \arg\min_{\theta_G}\max_{\theta_D} V(\theta_G,\theta_D), \tag{1}$$

and an approximated solution is usually found by using alternating stochastic gradient descent, as discussed in section III. In order to have more concise notations, we will omit the parameters θ_G and θ_D.

III. METHODS

A. Dataset & Color space

As soon as we can easily remove colors from any image, we can chose any existing image database containing many different scenes and locations. In this way, we can hope our neural network to learn to colorize plausibly a plethora of different situations. For that purpose we used the Coco dataset from 2017 split. We also use the unlabeled images for training giving us 241k images which is only 20% of the dataset size use in [1].

We resize all images to 256x256 and we use the CIELAB colorspace instead of RGB. This colorspace allows to only predict the color channels *a, b* given the image's L channel as input [5]. We use batch of 4 images where random horizontal flip has been applied during training as data augmentation.

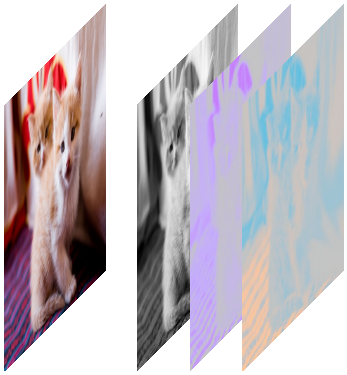

FIG. 1. Colored image of a cat along with its respective channel L, a and b

In order to build a cGAN to solve the colorization problem and with the notations of the previous section, we will take $\mathcal{X} = \mathbb{R}^{256\times256}$ to encode the luminance channel L and $\mathcal{Y} = \mathbb{R}^{2\times256\times256}$ to encode ab channels. Note that our neural networks are fed with L,a,b values normalized in $[-1,1]$.

B. Noise

This paper [1] suggests that using Gaussian noise z as input of the generator together with x is not significantly effective since the generator learns to ignore that noise. Instead, they *simulate* noise using dropout layer on the first layers of the decoder.

Following this, we have done the same choices regarding the noise.

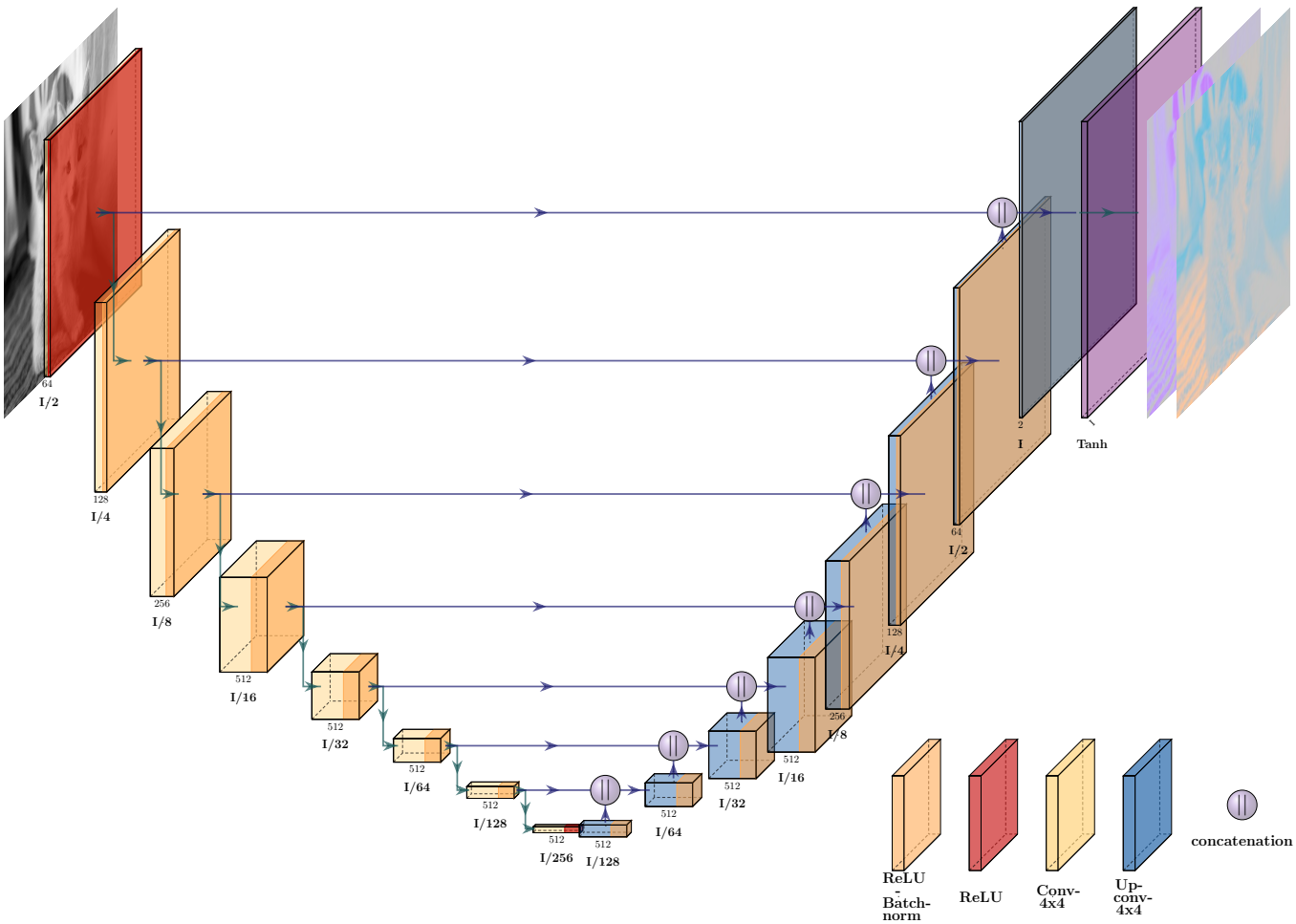

FIG. 2. Generator architecture

C. Generator architecture

We first present the U-Net architecture then focus on our specific implementation for our generator.

U-Net is a convolutional neural network architecture initially developed for biomedical segmentation [6]. It has recently proven to be very effective for multiple task consisting in picture to picture task such as segmentation and colorization.

U-Net architecture consists of two main parts: the encoder and the decoder. The encoder is a succession of convolutions and pooling layers that allows to capture features of an image and reduce the number of parameters to optimize by reducing the size of the input. It is then followed by the decoder part that consists in up-sampling and a concatenation with layer of the encoder having the same size for localization.

Our generator is an adaptation of U-Net, following the description of [1]. The encoder uses a succession of 4x4 *convolutions-batchnorm-ReLU* [7] layers with padding of 1 and stride of 2. This choice of kernel size, stride and padding allow to divide the size of the input by 2 at each layer instead of using max-pooling layer as U-Net was initially designed.

Below are the computation of the output size n_{out} given the input size n_{in} after a convolution layer of kernel size k, padding p and stride s. We suppose that n_{in} is even.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} + 1 \right\rfloor = \left\lfloor \frac{n_{in} + 2 - 4}{2} + 1 \right\rfloor$$
$$= \left\lfloor \frac{n_{in}}{2} + \frac{-2}{2} + 1 \right\rfloor = \frac{n_{in}}{2}$$

At each stage of down-sampling, the number of feature channels are multiplied by a factor of 2 until reaching 512 channels starting from 64 channels. When reaching the bottleneck, meaning having a layer of size 1x1, the decoder takes the role of retrieving the initial image size. Each layer is up-sampled using transposed convolution with the same kernel size, padding and stride than the encoder in order to double the input size. Then, in order to avoid loosing information from the input to the output, we add skip connections across layers of same size; each skip connection concatenates all channels

of layer i with those at layer $n-i$ where n is the number of layers and $i \in \{1, .., \frac{n}{2} - 1\}$. Finally, the last transposed convolution outputs 2 channels that are then passed through a *tanh* activation function. Given a grey scale image, represented by it's L's normalized channel, out generator outputs the predicted color encoded as normalized a and b.

As introduced earlier (section III B), noises are simulated by the use dropouts, which are applied to the three first layers of the decoder with a dropout rate of 50% for both training and test time. Following the choice made in [1] during test time batch norm uses the statistics of the test batch, rather than the one of training batch. All the ReLUs activation in the encoder are leaky, with a slope 0.2, while ReLUs in the decoder are not.

See FIG.2 for a detailed graphical representation of the generator.

D. Discriminator architecture

Since the $L1$ loss enforces the correctness at the low frequencies, the GAN discriminator can focus on modelling high-frequency pixels. This modelling can be restricted at the scale of patches; the discriminator's job is to determine if each $N \times N$ patch in an image is real or fake.

In this setting, an image (more precisely, the normalized channels L, a and b) is given to the discriminator which outputs a $M \times M$ matrix, where each element of that matrix is a value between 0 and 1 determining how confident the discriminator is towards the veracity of the patch associated to that element.

The discriminator architecture is a succession of 4x4 *convolutions-batchnorm-LeakyReLU* blocks, except for the first and the last layer. The first layer is a block with no *batchnorm* and the last layer only does a convolution. All *LeakyRelu's* have a slope of 0.2. After the last layer, we apply (element-wise) a sigmoid function to the $M \times M$ matrix in order to get a matrix whose of the same dimension whose all elements between are 0 and 1.

The desired patch size highly depends on the number of layers of the discriminator. In this work, the discriminator works on patches of size 70×70. Hence, the discriminator architecture is made of 5 blocks. The first block starts with 64 channels and the number of channels is multiplied by a factor of 2 from one channel to another. The last channel maps the input into a one dimensional output. In the case of $256 \times 256 \times 3$ images, the output of the last convolution is of size 30×30.

A detailed description of the architecture is available at FIG.3.

In theory, the discriminator of a cGAN needs to output a number between $[0, 1]$. In our case, the discriminator outputs a 30×30 matrix whose elements are all in $[0, 1]$. To adress this output, we consider that, for each image, the target is a 30×30 matrix filled with 1 or 0 depending on if the image is real or fake. Therefore, each of these elements is forwarded to the loss function, giving a scalar at the end.

E. Objective functions

As a reminder of sub-section II A, the objective function of a cGAN can be expressed as

$$\mathcal{L}_{\text{cGAN}}(G, D) = \mathbb{E}_{x \sim p_X}\big[\mathbb{E}_{y \sim p_{Y|x}}[\log(D(x, y; \theta_D))] + \mathbb{E}_{z \sim p_{Z|x}}[\log(1 - D(x, G(x, z; \theta_G); \theta_D))]\big],$$

where the respective parameters θ_G and θ_D of G and D are implicit.

This objective functions can be seen as the binary cross-entropy loss.

As it was already the case in [1], we want our generator to fool the discriminator (ie: minimize $\mathcal{L}_{\text{cGAN}}(G, D)$) but also to be near the ground truth output ($L1$ norm pixelwise). The $L1$ norm is preferred as $L2$ norm because $L1$ norm encourages sparsity, resulting in less blurry generated images. Thus, we also consider the objective function

$$\mathcal{L}_{L1}(G) = E_{(x,y,z) \sim p_{X,Y,Z}}\big[||y - G(x, z)||_1\big].$$

The final objective considered in [1] (which is not **our** final objective. See V A for more details.) is given by

$$G^\star = \arg\min_G \max_D \mathcal{L}(G, D), \tag{2}$$

where $\mathcal{L} = \mathcal{L}_{\text{cGAN}} + \lambda \mathcal{L}_{L1}$ and where $\lambda > 0$ is an hyperparameter.

F. Optimization

As already introduced in subsection II A, a solution of the minimax optimization problem (2) is approximated by using alternating stochastic gradient descent, such as

- $\theta_G \leftarrow \theta_G - \alpha \nabla_{\theta_G} \mathcal{L}(G, D)$ (descent step),
- $\theta_D \leftarrow \theta_D + \alpha \nabla_{\theta_D} \mathcal{L}(G, D)$ (ascent step),

where $\alpha > 0$ is a learning rate. Note that the convergence to a global solution is not guaranteed.
As proposed in [4] and [1], we will do the following trick: we train G to minimize $-\mathbb{E}[\log(D(X, (G(X, Z))))]$ instead of $\mathbb{E}[\log(1 - D(X, (G(X, Z))))]$. Indeed, when G is poor, the generated sample are highly different from the true data. Thus, the discriminator can reject generated samples with high confidence. This will induce a saturated gradient. Furthemore, the functions f_1 and f_2 defined as

$$f_1(t) = -log(t), f_2(t) = log(1 - t),$$

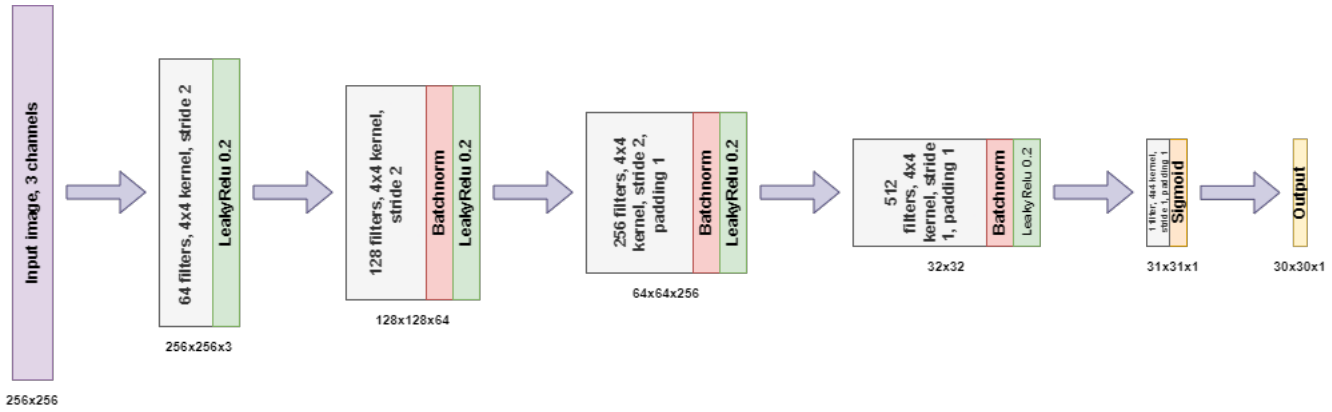

FIG. 3. Discriminator architecture

are both stricly deacreasing on $]0, 1[$. All in all, having recourse to this trick induce the same dynamics between the generator and the discriminator, but provide stronger gradient when G is poor (ex : early in the learning).

In addition, we divide the objective by 2 while optimizing D, which slows down the rate at which D learns relative to G.

G. Metrics

To assess the quality of the produced images, converted back to RGB, two metrics are used:

- **Peak Signal-to-Noise Ratio (PSNR):** The PSNR is computed as

$$\text{PSNR} = 20log_{10}(\frac{256}{\sqrt{\text{MSE}}})$$

  where MSE is the *Mean Squared Error* between the original image and the reconstructed image. The higher the PSNR is, the better the image has been reconstructed.

  The purpose of this metric is to compare how correct the generator has reconstructed the original image. However, image reconstruction is not the task that the architecture must do; the model must re-colorize in a logical manner, not necessarily recover the right color with respect to the real image.

- **Structural Similarity (SSIM):** The SSIM of two images, in constrast to PSNR, doesn't evaluate the difference pixel-wise. Instead, SSIM compares the structures of two images. In fact, the approach is made under the hypothesis that pixels that are close to eachother represent structural information. The SSIM of two images x and y with 1 channel is computed based on the luminance l, the contrast c and the structure s between the two images:

$$l(x,y) = \frac{2\mu_x\mu_y + c_1}{\mu_x^2\mu_y^2 + c_1},$$
$$c(x,y) = \frac{2\sigma_x\sigma_y + c_2}{2\sigma_x^2\sigma_y^2 + c_2},$$
$$s(x,y) = \frac{\sigma_{x,y} + c_3}{\sigma_x\sigma_y + c_3},$$
$$SSIM(x,y) = l(x,y)c(x,y)s(x,y)$$

  where μ_x and μ_y are the means, σ_x and σ_y the variances, σ_{xy} the covariance, and $c1$, $c2$, $c3$ are constants.

  The SSIM between two images x and y of 3 channels is computed by computing the SSIM channel-wise and average the results.

IV. OUR CONTRIBUTIONS

We keep the same notation as subsection II A.

In this section, we present our *contributions* concerning the paper [1] for which we drown our inspiration. Indeed we wanted to include some tricks to regulate and ease the convergence of our cGAN.

A. Regularization term

As shown in the article [8], adding a penalty on the quadratic norm of the gradients of the discriminator forces the local convergence of the alternating gradient descent algorithm used to train a GAN. Without any formal proof to transpose this trick to our settings [9], we've tested to add a similar regularization term to our objective function $\mathcal{L}$ defined below equation (2). Precisely, we've considered the regularization term given by

$$R_1(D;\gamma) = \frac{\gamma}{2}\mathbb{E}_{(x,y)\sim p_{X,Y}}\left[||\nabla_y D(x,y)||_2^2\right],$$

where $\gamma > 0$ is an hyper-parameter. The results and choise of hyper-parameter associated to this regularization term are discussed in the section VI.

B. One Sided Label Smoothing

According to [10] and [11] Neural networks tend to produce extremely confident outputs to identify the correct class but with too extreme probability.

It has been shown in [12] that using *one-sided label smoothing*, meaning using 0.9 instead of 1 for the positive label, reduce the vulnerability of adversarial neural networks and is a excellent regulator for encouraging the discriminator to estimate soft probabilities.

However by looking at TABLE. II this trick does not seems to be quite effective here.

V. OUR FINAL MODEL

In this section we first provide an ablation studies to validate our design choices V A. Then, we summarize our final model and explain precisely how we effectively proceed the training of our cGAN.

A. Ablation study

We validate our hyper-parameters choices with a quantitative and qualitative ablation study performed on the validation set (see Table II, FIG. 15 and FIG. 16).

We observe in Table II that the best *PSNR/SSIM* scores is achieved for the model trained only with the L1 loss. Unfortunately this result does not have such meaning since these metrics will privilege low MSE and pixel-wise reconstruction, which is not exactly what we want. Indeed, some reconstructed color may be plausible but different from the initial image while *PSNR/SSIM* will give higher value for brown/grey color which is kind of average color.

In the light of this remark, we have decided to choose our final model according to qualitative measurement, by lack of time and *human ressources*, we were not able to perform a Turing test to find the best model. Therefore, we have arbitrarily chosen the model that give the best results (visually) according to us in the validation set. We first tuned the parameters γ from IV A. The one that give, according to us, the best results is when γ is set to 0.001. Higher values give more conservative colors while lower values result introduce some strange color artefact in the recolorization. Samples of the experiment are available at FIG.15.

Then, experiments has been made on other parameters such as including or not the one Sided Label Smoothing trick IV B, only using $\mathcal{L}_{cGAN}$ or only the $\mathcal{L}_{L1}$ loss. From FIG. 16 clearly we can directly notice that when using only $\mathcal{L}_{cGAN}$ resulting images does not seems at all realistic while using only $\mathcal{L}_{L1}$ give more conservative and bland colors but still realistic image. The original architecture of our reference paper [1], called Pix2Pix, is a kind of mix between these two but does not look quite stable.

B. Summary of our final model

Given the previous ablation study, our final model makes use of only one of the two tricks explained in IV : the R1 regularization, using as hyper-parameters $\gamma = 0.001$ and $\lambda = 100$. Using or not the one-sided label smoothing does not significantly impact the result. We chose to not use this trick.

Finally, our final objective function is

$$\mathcal{L}(G, D) = \mathcal{L}_{cGAN}(G, D) + 100\mathcal{L}_{L1}(G) - R_1(D; 0.001)$$

according to (2). Given our trick explained in section III F, the alternating stochastic gradient steps are explicitly given by

1. $\theta_G \leftarrow \theta_G - \alpha\nabla_{\theta_G}\left(\mathcal{L}_G(G, D) + 100\mathcal{L}_{L1}(G)\right)$
2. $\theta_D \leftarrow \theta_D - \alpha\nabla_{\theta_D}\left(\mathcal{L}_D(G, D) + R_1(D; 0.001)\right)$

where

$$\begin{cases} \mathcal{L}_G(G, D) & = -\mathbb{E}[\log(D(X, (G(X, Z))))], \\ \mathcal{L}_D(G, D) & = -\frac{1}{2}\mathcal{L}_{\text{cGAN}}(G, D), \end{cases}$$

are respectively called the generator and discriminator loss.

For the rest of this report, we analyze qualitatively and quantitatively this final model.

VI. RESULTS

In this section, we compare qualitatively (VI A) and quantitatively (VI B) our final model. We also try to use a petrain backbone for the encorder part of our generator in our final model (VI C).

A. Qualitative analysis

When looking at FIG. 4., we can notice that the model makes a decent job when it comes to recolorize the sky and the greenery. Indeed, the model has learned to segment objects and to recolorize them.

FIG. 4. Examples of detecting grass and sky.

However, this is not always the case for some objects; FIG. 5. shows that the model colorizes in brown/gray objects for which it has poor confidence. This brown/gray re-colorization happens due to the L1 loss which favors more conservative colors.

Concerning human re-colorization at FIG. 6., the model performs well to re-colorize the whole face, but it struggles to recognize body parts at different angles such as the hands and the arms.

Other examples are available at FIG.17 and FIG.18.

FIG. 5. Examples of bad results

FIG. 6. Examples of re-colorizing humans

B. Quantitative analysis

Training generative adversarial network is hard due to some major problems as the non-convergence, model collapse or diminished gradient. Fortunately looking at the loss plot FIG. 7 can give some inside of the health of our model. Indeed, a good generator would fool completely the discriminator therefore that one would not be able to distinguish between real and fake images. If the generator succeeds this task the discriminator should have $\mathcal{L}_D = -(\log(0.5) + log(0.5))/2 = 0.69$.

The figure FIG. 7 shows that our model is quite stable and converge slowly toward 0.69 for both, generator and discriminator.

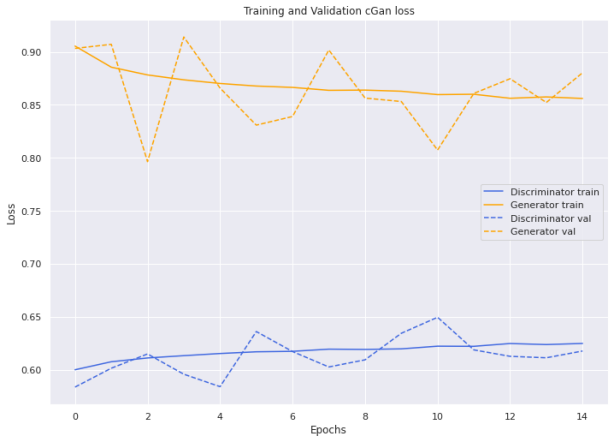

FIG. 7. Plot of the $\mathcal{L}_G$ and $\mathcal{L}_D$ on both the train and validation set.

Concerning the discriminator FIG. 8, the behavior of its loss over the epochs shows that little by little the discriminator loss increase it means that the generator directly improves from the response of the discriminator. On the other hand, the L1 loss FIG. 9 decreases over the epochs until the 8th epoch where it seems stabilizes. Moreover, the SSIM and the PSNR FIG: 10 improve slightly over the epochs, meaning that the generator somehow learns to achieve its task. FIG. 11 show the *complete* loss of the generator

As a comparison FIG. 19, 20, 21, 22, and 23 are the plot of a model trained with the same configuration but without the two trick explained in IV (same model as [1]). Clearly without the regularisation the GAN is way more unstable and the discriminator tend to learn to quickly compare to the generator.

On the test set our model give the following metrics, using batch of size 4 where the results are averaged along with its standard deviation.

PSNR	SSIM
24.3086 +- 2.1187	0.9222 +- 0.0260

FIG. 8. Plot of $\mathcal{L}_D$ along with $R_1(D, 0.001)$ on both the train and validation set.

FIG. 9. Plot of the $L1$ loss of the generator on both the train and validation set.

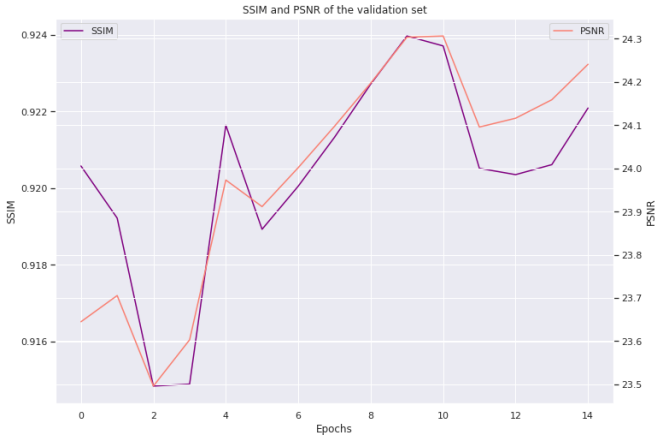

FIG. 10. Plot $SSIM$ and $PSNR$ metrics on the validation set.

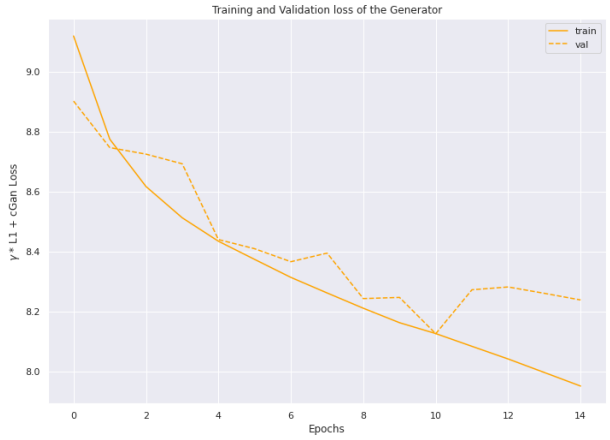

FIG. 11. Plot of the (full) loss of the generator ($\mathcal{L} + 100L1$) on both the train and validation set.

1. *Turing test*

It is known that evaluating the quality of generated image is an hard task. Since our goal is to recolorize image in a coherent way for humans, we did a poll with 20 pairs of images[13] available in annexe VIII A. Each pair of images contains one colorless image and either the original colorized version of that image or a colorized version of this image generated by our final model. Successively, poll participants were given a pair of such images and had to say whether the coloured image had been generated by our model or not. We've collected about 220 responses for each pair of images. The results are transcribed into the following confusion matrix.

		Answer	
	Truth	Original	Generated
	Original	1292	491
	Generated	988	1231

Given this table, we can compute the following value:

$$\text{True positive rate (TPR)} = \frac{1292}{1292 + 491} \approx 0.7246,$$
$$\text{True negative rate (TNR)} = \frac{1231}{988 + 1231} \approx 0.5548.$$
$$\text{False postive rate (FPR)} = \frac{988}{988 + 1231} \approx 0.4452.$$

Thus, in general, one can observe that

- poll participants classify original images from the dataset as original images in $\approx 72\%$ of the cases.
- Given generated image, poll participants classify it as original image in $\approx 45\%$ of the cases.

C. Pretrain model - Transfer learning

As experiment we have also make use of a pretrain backbone for the encoder part of our generator with the same hyper-parameters/trick than our model. We wanted to see how well the transfer learning can increase our performance using a resnet18, without the last two layer, as the encoder part of a U-Net. Note that the resnet had been trained for classification on imagenet.

As the task is relatively different (i.e. trained on RGB images for classification while we use LAB for a segmentation task) we did not freeze the weights but just used them as starting point.

From our experiments we have not noticed much different between our model and the pretrain one apart from the fact that it is less likely to give images with random color artefact. However we have seen that it give relatively good results after few epochs in contrast to a no-pretrained model that will require more epoch.

Quantitatively

PSNR	SSIM
24.24.4454 +- 1.1382	0.9257 +- 0.0133

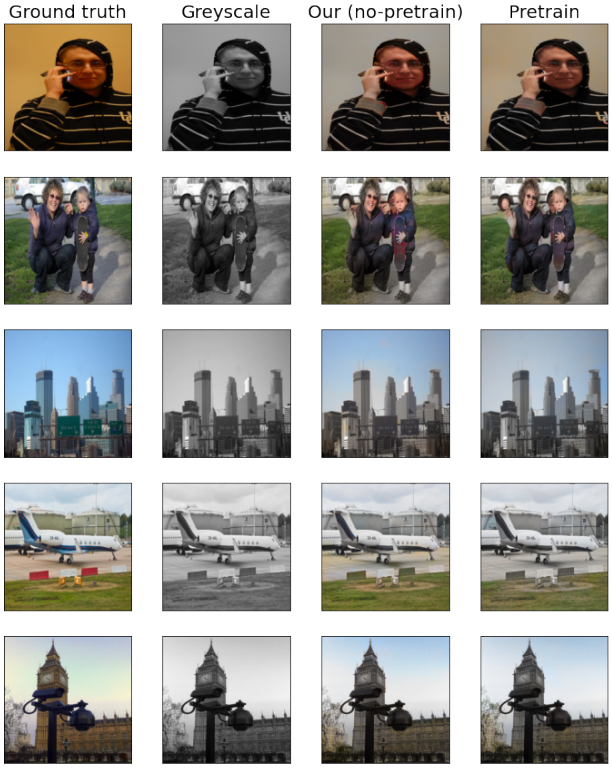

FIG. 12. Pretrain model - qualitative results

VII. DISCUSSION

Firstly, evaluating the quality of the results of the generator is not trivial. Even though *SSIM* and *PSNR* have been chosen to assess our results, they are not adequate; the task of our model is not meant to reconstruct the original image, but to realistically reconstruct it such that each element of the image makes sense (i.e. a car might be recolored in blue whereas the car is in red in the original image). Hence, the best metric this architecture could get is to see how much we can fool humans from making them think that this image is not reconstructed.

Secondly, the hyper-parameter γ associated to the $R1$ regularization has a strong impact on the training procedure. Indeed, when γ is too high, the gradient of the discriminator is flattened such that the discriminator struggles to learn anything. Therefore, the generator won't improve. In the other hand, the hyper-parameter λ associated to the $L1$ loss regulates how conservative the choice of the colors must be. In other words, if these hyper-parameters are too high, the minimization problem will likely result in only minimizing the $L1$ loss, and completely ignore the output of the discriminator.

Thirdly, the model is trained on many images, where a vast amount of elements of different nature appears in these images. For some elements, the generator might not realistically reconstruct them. However, in overall, the architecture remains efficient. In fact, the architecture could be trained on more specific datasets (i.e. images of humans) which would likely give better results for reconstructing the image.

A main limitation of our model is that it rely mainly on the distribution of the dataset. For instance, if the dataset does not contain any parrots, then it will struggle to correctly colorize a colorful parrots. Note that the fact that our model rely significantly on the distribution is also a strength of cGAN.

In overall, our final model produces coherent results with respect to the segmentation and colorization of images In some cases, when the grayscale image given as input is too intricate or incoherent for a human, the output tends to be either colorless or with too colorful artifacts.

Finally, knowing how to stop training is quite complicated. Previously, the early stop was made based on the value of *SSIM*. However, we observe that this was not relevant due to the unsuitability of this metric. One issue to train too longer was to overfit the L1 loss. Indeed, if one trains too longer, the L1 loss will continuously diminish on the training set while being stable on the validation set. Therefore, we decided to arbitrary set the number of epoch to 15, as it was producing satisfactory results. Empirically, 15 epochs seems to be the limit point where the L1 loss became stable in the validation phase.

A. To go further

In this project, we've build from scratch a conditional generative adversarial network. We implement the generator (U-Net) and discriminator as described in section III in a similar fashion as [1]. We've tried two tricks (section IV) and analyze the effect of them in an ablation study V A.

In order to continue this work, here are several possibilities. Firstly, we would like to tune the hyperparameters of our loss more precisely. Secondly, we would like to see what happens if we choose to predict the three L, a and b channel instead of only a and b. We would also like to see what happens if we chose to take patches of different sizes in the discriminator architecture. In a more theoretical way, we would like to see if a regularization exists for the generator (ex: [14]), understand why does it work. Finally, in a more general setting, we would like to use the general power of our cGAN architecture to solve a completely different task. For instance, train our cGAN create a map given a satelite view of an area (see [1] for more example).

[1] Tinghui Zhou Alexei A. Efros Phillip Isola, Jun-Yan Zhu. Image-to-image translation with conditional adversarial networks.

[2] because not present in the original article [1].

[3] Alexei A. Efros Richard Zhang, Phillip Isola. Colorful image colorization.

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

[5] Alexei A. Efros Richard Zhang, Phillip Isola. Colorful image colorization.

[6] Thomas Brox Olaf Ronneberger, Philipp Fischer. U-net: Convolutional networks for biomedical image segmentation.

[7] except for the first layer which is a *convolution-batchnorm* one.

[8] Lars M. Mescheder. On the convergence properties of GAN training. *CoRR*, abs/1801.04406, 2018.

[9] Our loss consists of the classical cGAN's loss plus a L1 penality, as described in subsection III E.

[10] Eric Ng Kamyar Nazeri and Mehran Ebrahimi. Image colorization using generative adversarial networks.

[11] Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks.

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

[13] After the end of the poll, we discovered that 2 images from the dataset that we used as *true* color were actually fake one. Therefore, the participants in the survey were usually wrong for these 2 images. Since this affected our results, we decided to ignore them in our confusion matrix.

[14] Yufeng Zheng, Yunkai Zhang, and Zeyu Zheng. Continuous conditional generative adversarial networks (cgan) with generator regularization, 2021.

[15] Moein Shariatnia. Colorizing black white images with u-net and conditional gan — a tutorial.

[16] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

VIII. APPENDIX

A. Turing test

FIG. 13. Original images

FIG. 14. Generated images

No pretrain	**Lambda**	**One sided label smoothing**	**R1**	**Only L1**	*PSNR*	*SSIM*
1	100	✓	1	-	24.4431 +- 0.0448	0.9272 +- 0.0002
2	100	✓	-	-	22.8674 +- 0.0521	0.9016 +- 0.0033
3	100	-	1	-	24.4491 +- 0.0537	0.9272 +- 0.0002
4	100	-	-	-	22.9265 +- 0.0734	0.9036 +- 0.0008
5	100	-	0.01	-	*24.3824*	*0.9256*
6 (our)	100	-	0.001	-	*24.2536*	*0.9214*
7	100	-	0.0001	-	*23.5766*	*0.9115*
8	100	✓	0.001	-	*24.2407*	*0.9221*
9	100	✓	-	-	*22.5860*	*0.8926*
10 (gan only)	0	-	-	-	*21.3014*	*0.8751*
11 (L1 only)	1	-	-	✓	**24.4775 +- 0.0097**	**0.9274 +- 0.0001**
Pretrain						
1	100	✓	1	-	**24.6284 +- 0.023**	**0.9289 +- 0.0003**
2	100	-	1	-	24.558 +- 0.0652	0.9288 +- 0.0002
3	100	-	-	-	22.8156 +- 0.034	0.9075 +- 0.0009
4	100	-	0.001	-	*24.5437*	*0.9273*
5	100	✓	0.001	-	*24.3423*	*0.9276*

TABLE II. Ablation Study, average results over 3 runs along with its standard deviation. Results in italic mean that the experiment had been perform only one time.

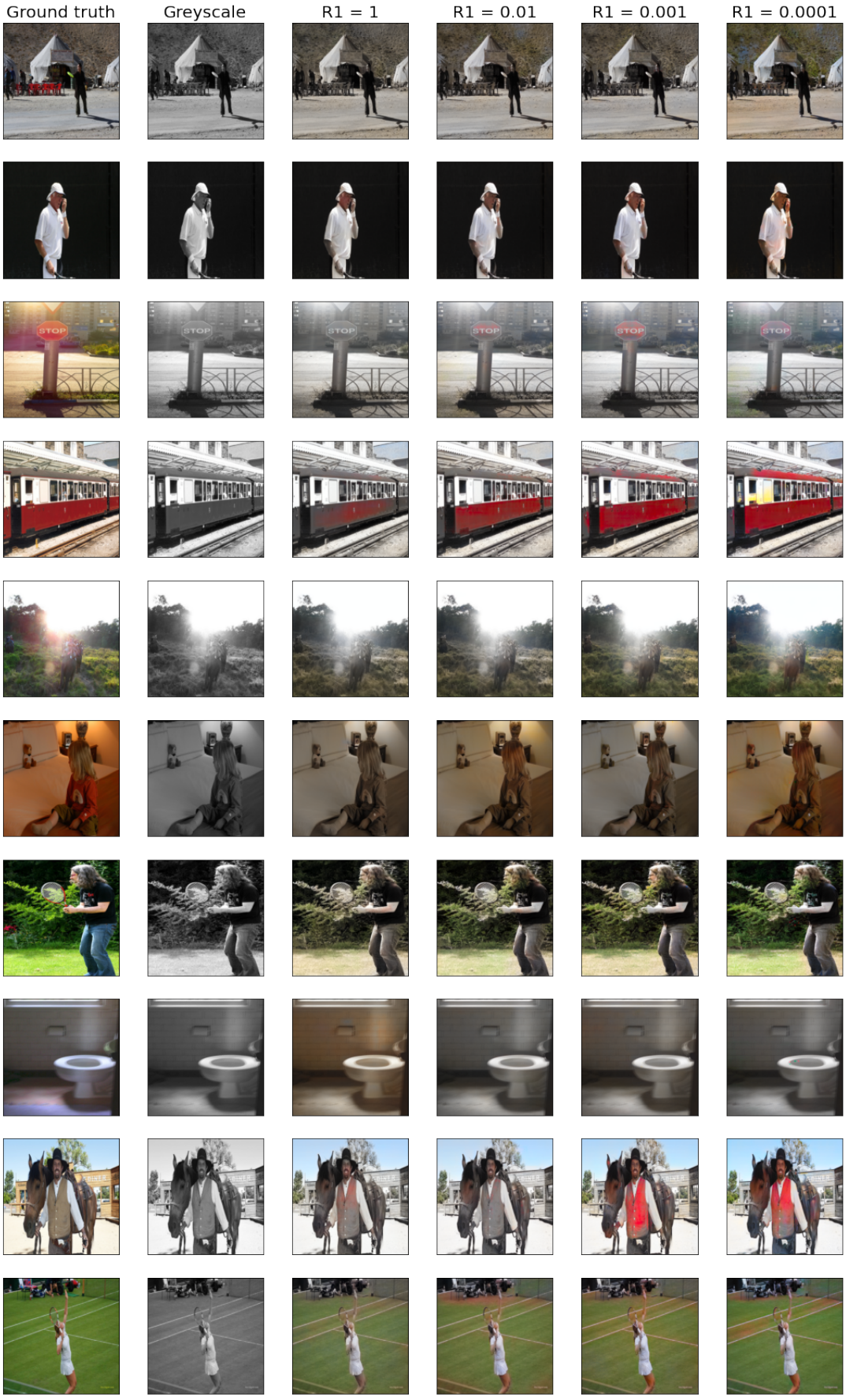

FIG. 15. Comparison among different values of γ (R1 regularization) and the ground truth. In the figures the value of γ is represented by "R1"

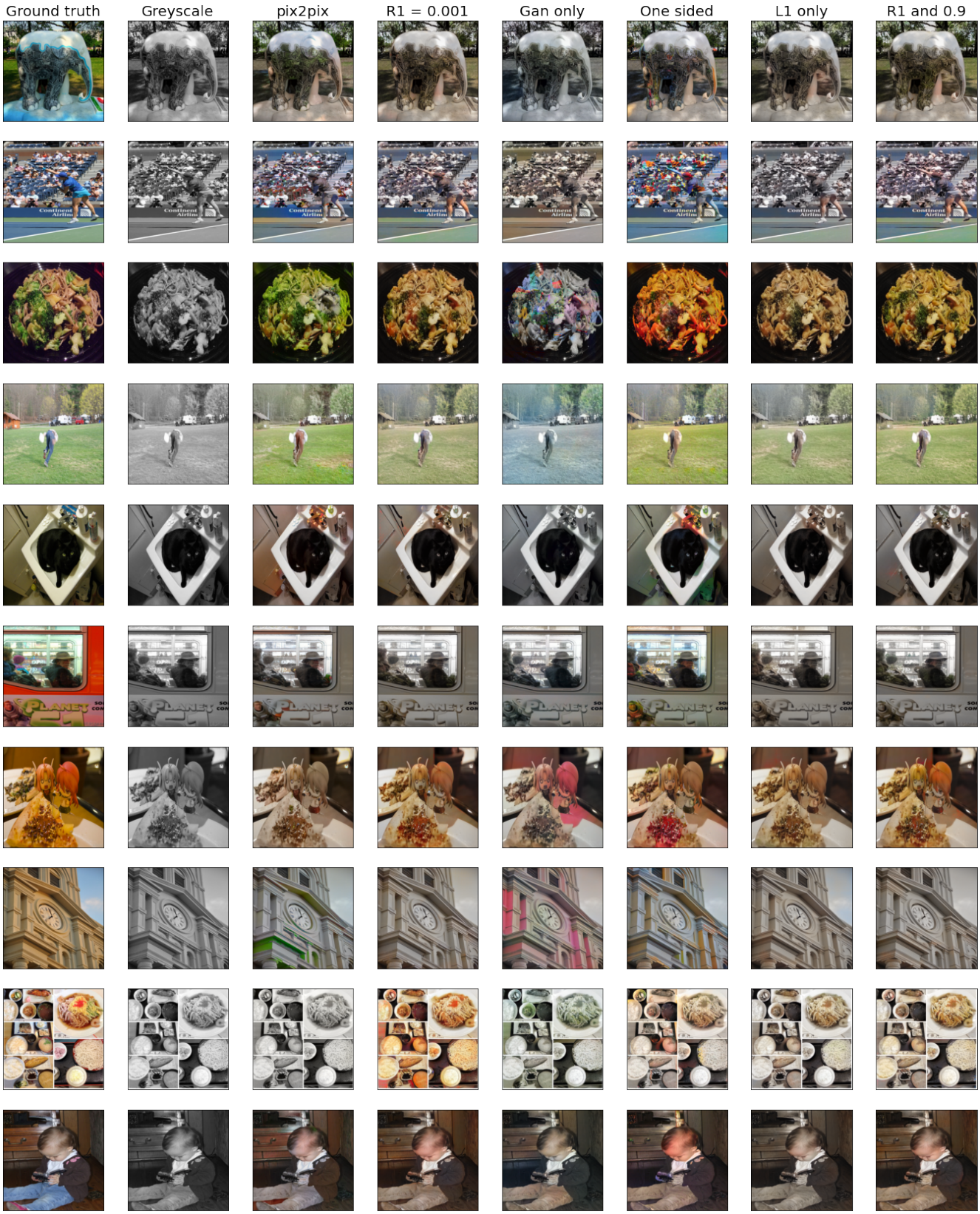

FIG. 16. Comparison of different models/hyper-parameters. *pix2pix* is the model train with the same hyper-parameters as [1], *One sided* is the same as *pix2pix* but including the one-sided label smoothing trick, and *R1 and 0.9* uses the R1 regularization with $\gamma = 0.001$ with the one-sided label smoothing trick.

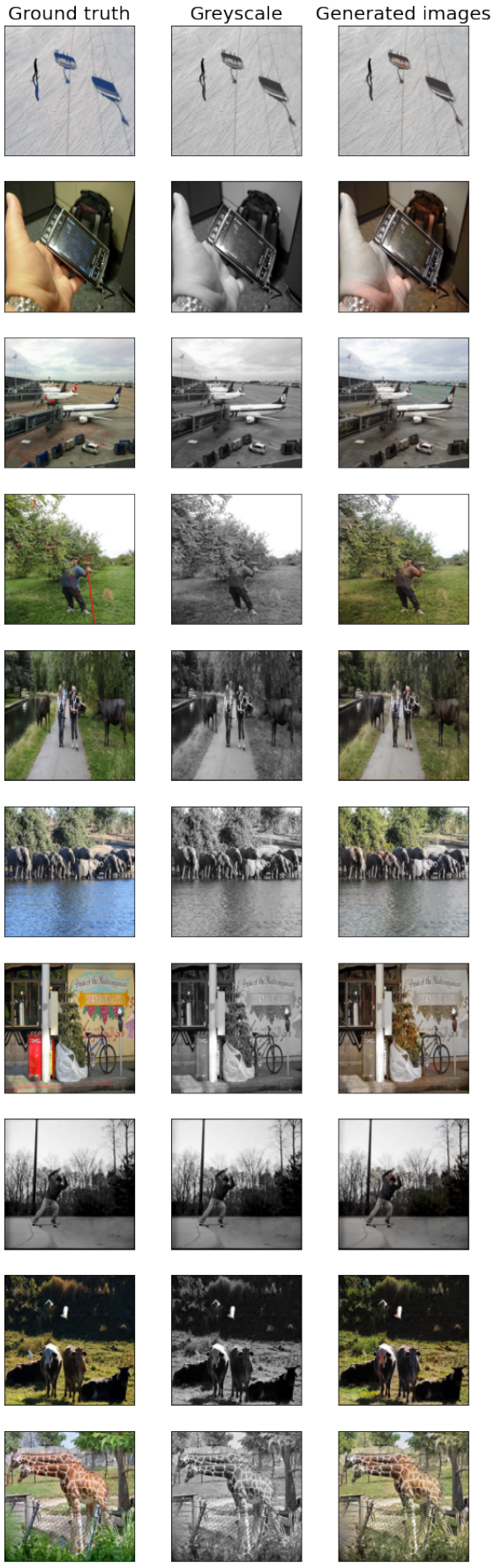

FIG. 17. Samples image from the test set using our model

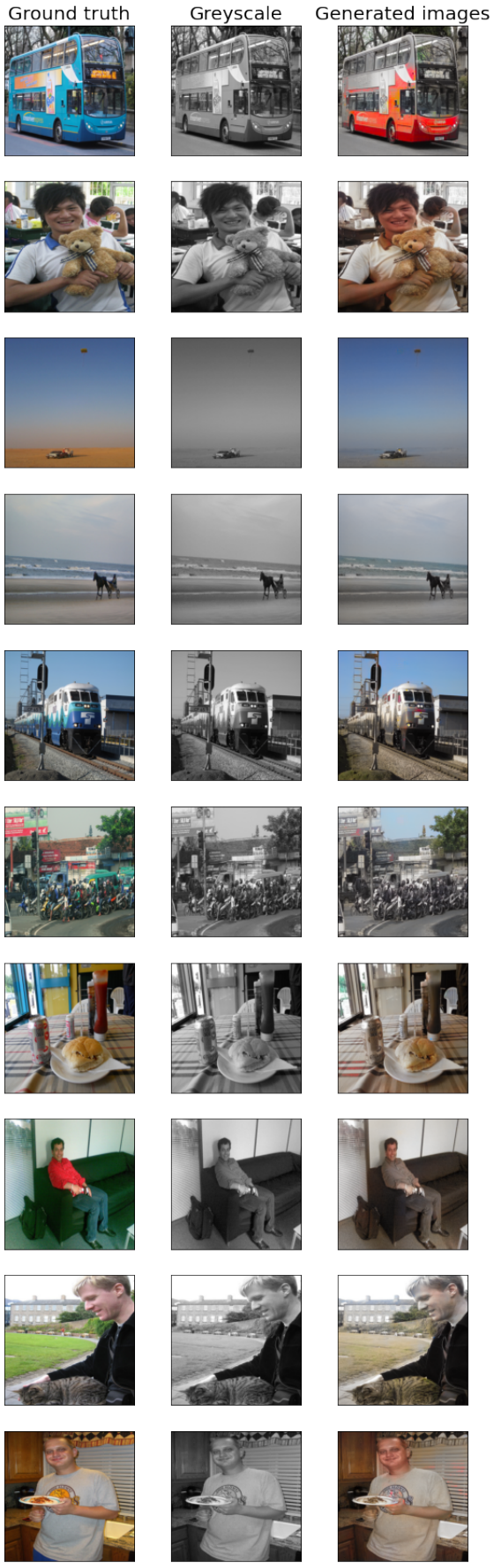

FIG. 18. Samples image from the test set using our model

FIG. 19. Plot of $\mathcal{L}_G$ and $\mathcal{L}_D$ on both the train and validation set with Pix2Pix model

FIG. 22. Plot of the *L1* loss of the generator on both the train and validation set with Pix2Pix model

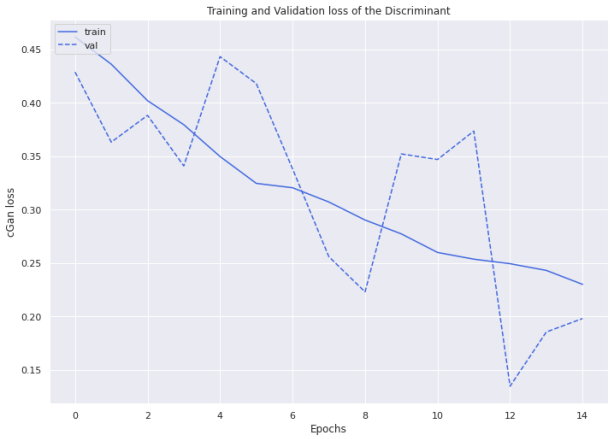

FIG. 20. Plot of $\mathcal{L}_D$ on both the train and validation set with Pix2Pix model

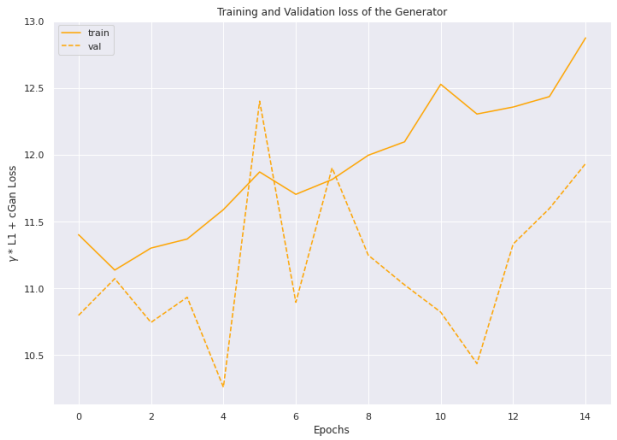

FIG. 23. Plot of the (full) loss of the generator on both the train and validation set with Pix2Pix model

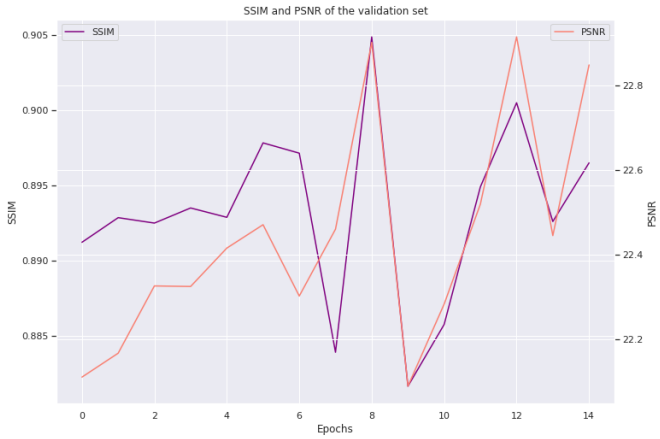

FIG. 21. Plot *SSIM* and *PSNR* metrics on the validation set with Pix2Pix model