Image Recoloring with conditional GANs

Luca Agosti[†], Nicola Bee[‡], Lorenzo Saccaro[⋆]

***Abstract*—The task of colorizing grayscale images is a challenging problem in computer vision, and the development of effective generative models is crucial in solving this task. This report presents a comprehensive study of various generative models and their impact on the quality and diversity of colorized images. The study focuses on using conditional Generative Adversarial Networks (cGANs) in combination with U-NET autoencoders and PatchGAN models, with the implementation of perceptual losses, Wasserstein Generative Adversarial Networks (W-GAN), and Pixel Convolutional Neural Networks (PixelCNN). The results of this study will provide valuable insights into the strengths and limitations of these models in solving the colorization problem and their potential for practical applications in the field of computer vision. This research is significant as it provides a thorough evaluation of various generative models for the colorization task and highlights their impact on the quality and diversity of the generated images. The findings of this study will be valuable for researchers and practitioners in the field of computer vision, who are interested in developing effective solutions for image colorization.**

I. Introduction

Image colorization is a crucial task in computer vision that aims to convert a grayscale image into a color image. While conventional methods for image colorization can produce unrealistic colors and lack of visual quality, recent advancements in deep learning have shown promising results in this area. In this report, we analyze some of the approaches for image colorization that leverages the strengths of patch generative adversarial networks (Patch GANs) [1] and U-Net autoencoders [2]. Our method utilizes the ability of Patch GANs to generate high-resolution colorized images and the power of U-Net autoencoders to preserve the details and structure of the original image.

Our starting point is the baseline conditional GANs (cGANs) architecture implemented by Philip Isola et al. [3]. We aim to improve their results by modifying our initial framework with the following refinements:

- Analyze different perceptual losses to quantify the quality of generated images.
- Implement the Wasserstein loss, replacing the standard GAN objective.
- Modify the generator architecture to produce higher-quality images using an autoregressive decoder such as the PixelCNN.

In this report, we will first introduce some key concepts like Generative models, GANs architecture, and the Wasserstein loss. Then we will see how we adapted our learning framework to the chosen dataset and solved the problems that rose. Lastly, we will examine the results obtained from our study and discuss their possible implications.

Department of Physics and Astronomy "Galileo Galilei", University of Padova

[†]luca.agosti@studenti.unipd.it

[‡]nicola.bee.1@studenti.unipd.it

[⋆]lorenzo.saccaro@studenti.unipd.it

II. Related Work

Recoloring images has become over the years a classic problem to whom many approaches have been proposed. These approaches vary both in the definition of the problem (user-guided, supervised, or unsupervised) and the architecture used to solve them. An extended taxonomy can be found in [4]. One of the state of the art strategies is to use conditional generative adversarial networks (cGAN) [5]. This kind of architecture is used by Isola et al. [3] for image-to-image translation. In that work, the cGAN exploits the fact that the generator's input (which is always another image) can be fed to the discriminator as side information. The model is built using a U-Net-based architecture for the generator and a convolutional Patch-GAN for the discriminator. This solution is applied to a wide variety of tasks, including image recoloring. The latter task is deepened by Nazeri et al. [6], where the cGAN is studied specifically for image recoloring taking into account two different datasets: CIFAR10, which contains $50\,000$ images of resolution 32×32, and Places365, that is a more challenging set since it has 1.8 million of pictures with a resolution of 256×256. Moreover, this paper highlights the problem that metrics that describe *quantitatively* the image quality are difficult to define. Another approach is presented by Cao et al. [7] where the U-Net generator is changed in favor of an architecture without down-sampling that resembles DenseNet [8], to better focus on local features. The model is applied to the recoloring task of the LSUN bedroom dataset (using for the training $503\ 900$ bedroom images of resolution 64×64), training the model both in RGB and YUV colorspaces. The evaluation of the performance is done by providing surveys to real people. This methodology, however, is hardly applicable and lies beyond the purposes of our study. These last two papers reach satisfactory results in image recoloring, taking into account that the datasets used have sometimes a low resolution and other times a small variety of subjects. Our work aims to expand the study in the application of cGAN to image recoloring considering a different dataset, structure, and loss function. Despite this discussion, it should be noticed (for intellectual honesty) that standard cGAN is no longer the state of the art in unsupervised image recoloring and new and more articulate methods, such as MemoPainter [9] or Colorization Transformer [10].

III. Theoretical Background

A. Introduction on Generative Models

Generative models in machine learning are a class of models that aim to generate new data points by learning the underlying structure of the training data. They can be used for a wide range of applications, such as image synthesis, text generation, audio synthesis, and anomaly detection, among others. Generative models can be divided into two main categories: generative algorithms and generative neural networks. We will focus mainly on the latter. Generative neural networks use deep neural networks to model the data distribution and generate new data points. Some of the most popular generative neural networks include Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs).

Generative Adversarial Networks (GANs) consist of two components: a generator network and a discriminator network. The generator creates new data points and the discriminator evaluates the realism of the generated data. The two networks are trained in an adversarial manner, where the generator tries to produce data that the discriminator cannot distinguish from real data, while the discriminator tries to correctly identify whether a sample is real or fake.

Variational Autoencoders (VAEs) consist of two components: an encoder network and a decoder network. The encoder network maps the input data to a lower-dimensional representation, called the latent code, and the decoder network maps the latent code back to the original space. During training, the VAE tries to learn a compact representation of the data and a probabilistic model of the data distribution.

B. Differences between GANs and VAEs

Both VAE and GAN are used mainly for image generation, but their approach to this task is different. VAE uses an unsupervised approach for training, while GAN uses a supervised technique. During training, VAE tries to compress the input into a latent space and generate output from a target distribution. Meanwhile, GAN finds a balance between the generator and discriminator in a two-player game. The loss function for VAE is KL-divergence, while GAN uses two loss functions, one for the generator and one for the discriminator. VAE is generally easier to train than GAN, but once the balance is found, GAN can recognize more complex insights of the input and generate higher-quality data. Therefore GAN is used for demanding tasks such as super-resolution and image-to-image translation and this is we choose this type of network for our image recoloring task.

C. GANs Architecture

The initial architecture that we implement is a patch GAN with U-Net autoencoder. It consists of two main components: the generator network (U-Net autoencoder) and the discriminator network (Patch GAN).

The U-Net autoencoder acts as the generator network and is responsible for converting a grayscale image into a color image. It consists of two main parts: the encoder and the decoder. The encoder compresses the input grayscale image into a lower-dimensional representation, while the decoder expands this representation back into the color image. The U-Net architecture is chosen for the autoencoder due to its ability to preserve the details and structure of the input image during the encoding-decoding process.

The Patch GAN acts as the discriminator network and is responsible for evaluating the quality of the colorized images generated by the U-Net autoencoder. It consists of several convolutional layers that receive overlapping patches of the colorized image as input. The Patch GAN then predicts the probability of each patch being real (i.e., from a training dataset) or fake (i.e., generated by the U-Net autoencoder). The Patch GAN and U-Net autoencoder are trained together in a game-theoretic manner, where the goal of the U-Net autoencoder is to generate colorized images that the Patch GAN can't distinguish from real images, while the goal of the Patch GAN is to accurately distinguish between real and fake images. Regarding the losses it is widely recognized that using the L2 loss (and also the L1 loss) leads to blurry results in image generation problems. Despite these losses failing to encourage sharpness at high frequencies, they still effectively capture low frequencies in many cases. When this is the case, we do not need to establish a completely new framework to ensure accuracy at low frequencies; the L1 loss will suffice. This motivates the restriction of the GAN discriminator to only model high-frequency structures, relying on the L1 term to enforce low-frequency accuracy. To model high frequencies, it is enough to focus on the structure within local image patches. This is the reason why the discriminator architecture, which we refer to as a PatchGAN, only penalizes structures at the patch level. This discriminator attempts to classify each N x N patch in an image as either real or fake. We run this discriminator convolutionally across the image, taking the average of all responses to produce the final output of D.

D. Objective

The objective of our network during training is to make the generator G produce images that the discriminator D cannot distinguish from real ones. This task is described by the following loss function:

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))] \tag{1}$$

where the generator and the discriminator play a min-max game described by:

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

However, our actual loss function includes an additional $L1$ term because previous work in the literature shows beneficial effects resulting from this addition. The $L1$ term is favoured over $L2$ because it encourages less blurry images:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{3}$$

E. WGAN and WGAN-GP

The Wasserstein GAN (WGAN), introduced in [11], proposes a different objective function. Instead of minimizing the *Jensen-Shannon* (JS) divergence

$$JS(\mathbb{P}_r, \mathbb{P}_g) = KL(\mathbb{P}_r||\mathbb{P}_m) + KL(\mathbb{P}_g||\mathbb{P}_m) \tag{4}$$

where $\mathbb{P}_m$ is the mixture $(\mathbb{P}_r + \mathbb{P}_g)/2$, the authors suggest to use the Wasserstein-1 or *Earth-Mover* (EM) distance

$$W(\mathbb{P}_r, \mathbb{P}_g) = \inf_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)} \mathbb{E}_{(x,y)\sim\gamma}\Big[||x, y||\Big] \tag{5}$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ denotes the set of all joint distribution $\gamma(x, y)$ whose marginal are respectively $\mathbb{P}_r$ and $\mathbb{P}_g$. Informally, this represents the minimum cost of transporting mass in order to transform the distribution $\mathbb{P}_r$ into the distribution $\mathbb{P}_g$. From this distance, the WGAN value function can be constructed using the Kantorovich-Rubinstein duality [12]

$$\min_G \max_{D\in\mathcal{D}} \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})] - \mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] \tag{6}$$

where $\mathcal{D}$ is the set of 1-Lipschitz functions. In this framework, the discriminator is also called *critic* since it no longer classifies samples as real or fake but outputs a scalar value that measures the realism of a sample, with the goal of producing low values for real samples and high values for generated samples. To enforce the Lipschitz constraint on the critic, the authors suggest clipping the weights of the critic to lie within a compact space [-c, c]. The WGAN value function results in a critic function whose gradient with respect to its input is better behaved than its GAN counterpart, making optimization of the generator easier (although longer) and more stable. Moreover, the authors observe that the WGAN value function appears to correlate with sample quality, which is not the case for GANs.

An additional improvement to the WGAN is proposed by [13]. The authors suggest adding a new term to the objective to overcome some of the problems of WGANs caused by the weight clipping, such as capacity underuse and exploding or vanishing gradients. Leveraging the fact that a differentiable function is 1-Lipschtiz if and only if it has gradients with norm at most 1 everywhere, a gradient penalty (GP) term is added to the critic loss. The objective of the WGAN-GP is as follows

$$\mathbb{E}_{\tilde{\boldsymbol{x}}\sim\mathbb{P}_g}[D(\tilde{\boldsymbol{x}})] - \mathbb{E}_{\boldsymbol{x}\sim\mathbb{P}_r}[D(\boldsymbol{x})] + \lambda\mathbb{E}_{\hat{\boldsymbol{x}}\sim\mathbb{P}_{\hat{\boldsymbol{x}}}}\Big[\left(\|\nabla_{\hat{\boldsymbol{x}}} D(\hat{\boldsymbol{x}})\|_2 - 1\right)^2\Big] \tag{7}$$

This solves the issues of weight clipping while conserving the good properties of WGAN.

F. Perceptual losses

Perceptual loss functions have been widely used for various image generation tasks such as colorization and style transfer. They are necessary for our task because we need to establish some kind of objective metric to evaluate the performance of our network during each epoch of the training process. Unfortunately, in our case the best perceptual loss is not clear a priori, therefore we compute each of the following five metrics during each epoch of the training process and then do a comparison between them to see which ones are a true reflection of an improvement in the generated image quality.

- Frechet Inception Distance (FID): A metric used to assess the quality of images created by GANs. The FID compares the distribution of generated images with the distribution of a set of real images ("ground truth"). The higher the value of FID is, the better the similarities between the images are. The metric was originally proposed in [14].
- Structural Similarity Index Measure (SSIM): A method for predicting the perceived quality of digital television and cinematic pictures, as well as other kinds of digital images and videos. The resultant SSIM index is a decimal value between -1 and 1, where 1 indicates perfect similarity, 0 indicates no similarity, and -1 indicates perfect anti-correlation [15].
- Peak Signal Noise Ratio: The ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. Higher values of PSNR imply smaller colorizing errors [16].
- Learned Perceptual Image Patch Similarity (LPIPS): Used to judge the perceptual similarity between two images. LPIPS essentially computes the similarity between the activations of two image patches for some pre-defined network. This measure has been shown to match human perception well. A low LPIPS score means that image patches are perceptual similar [17].
- Universal Image Quality Index (UIQI): Evaluates the quality of an image using loss of correlation, luminance distortion, and contrast distortion. The range of values for the index Q is [-1, 1]. The best value 1 is achieved if and only if the images are identical. It was initially proposed in [18].

IV. Dataset

The dataset that we choose for our analysis is the COCO-2017 dataset containing more than 160 000 different images of various resolutions, divided as:

- Training set: 115 045
- Validation set: 5 000
- Test set: 40 670

Since the training set is huge and due to the fact that multiple experiments need to be carried out we opt to use only 20% of it. After an initial attempt in studying our data using the RGB color space, it is immediately noticeable that using the L^*a^*b color space is a more suitable choice for our problem, as explained in the next section.

A. The $L^*a^*b^*$ color space

The acronym $L^*a^*b^*$ stands for:

- L^*: Lightness
- a^*: Red/Green Value

- b^*: Blue/Yellow Value

and there are five main reasons that make the $L^*a^*b^*$ color space more suited for our colorization task:

- When using $L^*a^*b^*$, we can give the L channel to the model (which is the grayscale image) and want it to predict the other two channels (*a, *b), and after its prediction, we concatenate all the channels and obtain our colored image. If RGB is used instead, we have to first convert the image to grayscale, feed the grayscale image to the model, and hope it will predict 3 numbers, which is a way more difficult and unstable task due to the many more possible combinations of 3 numbers compared to two numbers.
- $L^*a^*b^*$ color space is perceptually uniform: The $L^*a^*b^*$ color space separates color information (A and B channels) from lightness information (L channel), which allows for more accurate color representation. This can be particularly useful for image colorization, as it allows for more precise control over the colorization process.
- $L^*a^*b^*$ color space is more suitable for image processing: $L^*a^*b^*$ color space is designed to be perceptually uniform and it separates the lightness information from the color information. This makes it more suitable for image processing tasks such as colorization, where it is important to maintain the relationship between lightness and color.
- $L^*a^*b^*$ color space is more robust to lighting changes: The L channel of the $L^*a^*b^*$ color space represents the lightness of the image, which is relatively robust to changes in lighting conditions. This can be useful when colorizing images taken under different lighting conditions, as it allows for more consistent results.
- $L^*a^*b^*$ color space is more similar to human vision: The $L^*a^*b^*$ color space is based on the way human eyes perceive color, which means that the results of colorization in $L^*a^*b^*$ space are more similar to what a human would perceive.

B. Preprocessing

After an initial check of the dataset, we can notice that some of the images are already in greyscale. However having a non-insignificant percentage of greyscale images in our dataset could have a negative impact on the training process of our network, suggesting to the generator to not add any colors when creating an image. Therefore we could improve the training performance by writing a script to rule out those outlier images, resulting in a total of 3% of the dataset being deleted. Subsequently, the images are resized to 256×256, converted to the $L^*a^*b^*$ color space, and then scaled to the $[-1, 1]$ interval before being fed to the model.

V. Learning Framework

A. Generator

The U-Net generator built for our purposes consists of an encoder and a decoder composed of sequences of similar blocks. Inside the encoder, each block contains a convolutional layer with a stride of 2, a batch normalization [19] layer, and a Leaky ReLU activation function [20]. For the decoder instead, the structure consists of a transposed convolutional layer, a batch normalization layer, and ReLU activation function [21]. The different choice of the activation function is due to the fact that Leaky ReLU allows for better managing the vanishing gradient problem in the encoder part. Both encoder and decoder are composed of 8 blocks, that progressively halve the size of the image from 256×256 down to a 1×1 feature vector and then up again to the original input size. This architecture is different from the original one, which makes the input pass to three convolutional blocks of the same dimension before applying a max pool or a transposed convolution to respectively reduce or increase the input size. The peculiarity of U-Net is that it contains skip connections between layers of the same size. This is obtained by concatenating at each decoder's step the result of the whole process with the vector of features encoded up to that size. In the decoder branch, a dropout of 50% is applied starting from the fourth block. The very last layer of the U-Net architecture uses $\tanh$ as an activation function to get a prediction in $(-1, 1)$, that emulates the input values. The whole architecture, shown in Fig. 1, has around 54 million parameters.

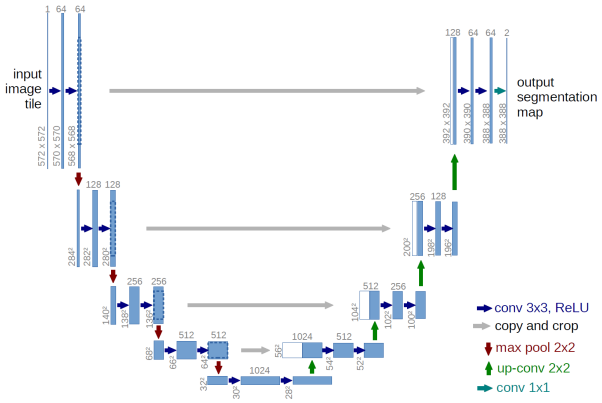

Fig. 1: U-Net original architecture [2]. Our architecture differs for numbers and shapes of the layers, but the main structure is the same

Autoregressive Decoder: A modification of the standard U-Net architecture is obtained by implementing an autoregressive decoder, which means that each output of the network is generated based on previously generated output. A well-known approach in convolutional networks is to apply PixelCNN [22], which performs a masked convolution to predict each pixel based on previous ones, replacing standard convolutional layers. In our decoder, however, the layers can not be easily substituted since they are transposed-convolutional layers. To implement an autoregressive decoder we use instead Pixel-transposed-convolutional-layers (PixelTCL) as described by Gao et al. [23]. In standard TCL the input is double-sized by nesting together four intermediate layers obtained by independent kernels; in PixelTCL, instead, these intermediate layers are generated sequentially, so that they develop interdependence. This is achieved using three masked

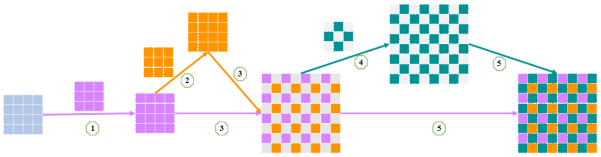

Fig. 2: Operations done by the Pixel transposed convolutional layers. The first two features layer are created sequentially with two 3×3 kernels (① and ②) and then are dilated and added together ③. The third and fourth feature layers are produced together with a masked kernel ④. Lastly, all the layers are combined together ⑤. Image taken from the original paper [23]

3×3 convolutions as shown in Fig. 2 Note that PixelCNN and PixelTCL's result differ since the first aim to create a sequential correlation between already-considered pixels and new ones, while the second correlates the pixels that will be neighbor in the next (expanded) layer. This new type of layer replaces the standard transposed convolution without having to modify the U-net shape and the skip connections, but has a higher computational cost, which leads the whole amount of parameters to 69 million.

B. Discriminator

Our cGAN uses a PatchGAN as discriminator. This structure is similar to the encoder of the generator and it is made of four blocks each composed of a convolutional layer with a stride of 2, a batch normalization layer and a Leaky ReLU activation function. The last layer is a simple convolutional block that returns a matrix of shape 30×30. Since the discriminator is quite shallow, each pixel's value of the final output is based only on its receptive field, which covers only a small portion of the image (i.e. a *patch*), and represents the classification of the corresponding patch. This means that instead of having a simple binary classifier we obtain 900 values that "vote" to determine if the sample is real or fake. This discriminator's architecture has about 2.5 million parameters

C. Training procedure

Following the recipe of [3], for each iteration the discriminator is trained once on both real and fake samples, then the generator is trained once on the discriminator output of the generated images. We adopt the Adam optimizer [24] with an initial learning rate of $2e^{-}4$ and $\beta_1 = 0.5$ which should help reach an equilibrium between generator and discriminator. We train the model for 100 epochs, during which the learning rate is progressively lowered by a factor of 10 following a cosine annealing scheduler [25]. The batch size is set to 16.

Only a few modifications are needed when moving to the WGAN-GP:

- use the Wasserstein loss, no need to change anything in the architecture since the sigmoid is included in the GAN loss
- disable batch normalization layers in the critic, as suggested by [13]
- train the critic more than the generator (5 times is the amount suggested by both [11] e [13]

The training on a Kaggle instance with an NVIDIA P100 GPU takes about 25 minutes per epoch for the GAN and just shy of half an hour for the PixelTCL.

VI. Results

In Fig. 3 we show the evolution of the discriminator and generator adversarial losses during the training phase for both the baseline model and the PixelTCL. In both cases, the model does not converge and equilibrium between the generator and discriminator is not reached. In fact, the discriminator loss tends to zero, effectively denying any chance of improvement for the generator. Moreover, we observe that the validation loss is very noisy and does not provide, at least in this case, any additional information and it is not useful for managing the training nor for evaluating image quality, which is uncorrelated from the loss. Before examining the behavior of the perceptual losses (Fig. 3), a quick disclaimer: something seems to have happened when training was resumed at around epoch 44[1], probably something related to the metric class provided by torchmetrics. The FID shows the largest gap between training and validation among the other metrics and there is a small but clear difference between the baseline model and the PixelTCL. Moreover, after about 20 epochs the value stabilizes. The other metrics seem to not provide very useful information since training and validation seem to perform identically (with the exception of the PNSR, where the training values keep increasing) with values increasing or decreasing with the number of epochs without a corresponding increase in image quality, which is monitored visually during training.

A separate discussion is in order for the WGAN: despite numerous attempts, we are unable to obtain any decent result. In most cases, the model never starts colorizing the images. Changing the optimizer, its parameter, and the number of iterations of the critic seems to have no effect. The authors of [3] say that probably the reason is that the PatchGAN is too weak and that the fact that it looks at overlapping patches breaks the assumption of WGAN-GP that inputs must be independent[2] [26]. The very disappointing results that we show later are obtained by replacing the PatchGAN with a DCGAN (same structure but the last layer is a dense one with a single output unit).

In Fig. 5 e in Fig. 6 we showcase some generated images from the train and test set respectively. Overall GAN and PixelGAN seem to perform very similarly, the latter is in some instances a little be more precise (see the pizza for instance) and avoids strange choices of color (see the arm of the person which is red in the GAN). However, there are still some artifacts and missing or faded colorization, with brownish and bluish effects in most cases. A human can easily

[1] a session on Kaggle can last for up to 12 hours

[2] this, however, contradicts other results where WGAN-GP was applied successfully in a pix2pix framework

spot the fakes, with the exception of landscapes and animals that are usually well-colored. The model especially suffers with images where multiple objects and details are present.

Following the suggestion of [3] the model is kept in training mode also during evaluation. In this way, the dropout layers present in the generator provide a source of noise that results in slightly different colorization for the same gray image (see Fig. 4).

Finally, in Tab. 1 the perceptual losses are evaluated on the test set: the baseline model and PixelGAN have the same scores with the exception of the FID, while the WGAN is the worst in all of them. This, combined with the visual inspection of the samples, suggest that, at least for the colorization task, the FID is better suited to be a quantitative measure of the quality of the generated samples.

	GAN	PixelGAN	WGAN
FID	2.52	2.28	3.21
SSIM	0.86	0.86	0.86
PSNR	24.44	24.44	23.83
LPIPS	0.14	0.14	0.16
UIQI	0.37	0.37	0.35

TABLE 1: Perceptual losses evaluated on the test set

VII. Concluding Remarks

In this paper we have proposed a solution for the image recoloring task through cGAN architecture, experimenting with the effectiveness of PixelTCL, WGAN, and perceptual losses. The recoloring problem is well known for being challenging and complex overall. Furthermore, our dataset selections did not help to achieve better results both for its subjects' variability and resolution. There have been indeed previous studies that analyzed the difficulty of the problem with an inadequate dataset and try to develop a proper one specific for the task [4]. A possible improvement could be the integration of the FID metric, which we found has a direct correlation with the quality of generated samples, in the training process: directly in the loss function or to reliably monitor the progress of the model. More experiments are required to achieve at least decent results for the WGAN: the train takes lots of time and multiple combinations of parameters must be tested. This was very frustrating but instructive at the same time: failing in achieving good results helped us understand the problem more deeply. Finally, it can be explored how different elements we have used interact one with the others, building networks that include for example both the PixelTCL and the Wasserstein loss.

Contributions

- **Agosti**: implemented discriminator and researched and integrated perceptual losses
- **Bee**: implemented generator and PixelTCL
- **Saccaro**: implemented dataset and trainer class

Everybody contributed equally to the writing of the project

References

[1] C. Li and M. Wand, "Precomputed real-time texture synthesis with markovian generative adversarial networks," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part III 14*, pp. 702–716, Springer, 2016.

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241, Springer, 2015.

[3] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1125–1134, 2017.

[4] S. Anwar, M. Tahir, C. Li, A. Mian, F. S. Khan, and A. W. Muzaffar, "Image colorization: A survey and dataset," *arXiv preprint arXiv:2008.10774*, 2020.

[5] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[6] K. Nazeri, E. Ng, and M. Ebrahimi, "Image colorization using generative adversarial networks," in *Articulated Motion and Deformable Objects: 10th International Conference, AMDO 2018, Palma de Mallorca, Spain, July 12-13, 2018, Proceedings 10*, pp. 85–94, Springer, 2018.

[7] Y. Cao, Z. Zhou, W. Zhang, and Y. Yu, "Unsupervised diverse colorization via generative adversarial networks," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2017, Skopje, Macedonia, September 18–22, 2017, Proceedings, Part I 10*, pp. 151–166, Springer, 2017.

[8] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[9] S. Yoo, H. Bahng, S. Chung, J. Lee, J. Chang, and J. Choo, "Coloring with limited data: Few-shot colorization via memory augmented networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11283–11292, 2019.

[10] M. Kumar, D. Weissenborn, and N. Kalchbrenner, "Colorization transformer," *arXiv preprint arXiv:2102.04432*, 2021.

[11] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein gan," 2017.

[12] C. Villani, *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften, Springer Berlin Heidelberg, 2008.

[13] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," 2017.

[14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[15] "SSIM documentation." https://torchmetrics.readthedocs.io/en/stable/image/structural_similarity.html. [Online; accessed 10-February-2023].

[16] "PSNR documentation." https://torchmetrics.readthedocs.io/en/v0.8.2/image/peak_signal_noise_ratio.html. [Online; accessed 10-February-2023].

[17] "LPIPS documentation." https://torchmetrics.readthedocs.io/en/stable/image/learned_perceptual_image_patch_similarity.html. [Online; accessed 10-February-2023].

[18] Z. Wang and A. Bovik, "A universal image quality index," *IEEE Signal Processing Letters*, vol. 9, no. 3, pp. 81–84, 2002.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, pmlr, 2015.

[20] A. L. Maas, A. Y. Hannun, A. Y. Ng, *et al.*, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. icml*, vol. 30, p. 3, Atlanta, Georgia, USA, 2013.

[21] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[22] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, *et al.*, "Conditional image generation with pixelcnn decoders," *Advances in neural information processing systems*, vol. 29, 2016.

[23] H. Gao, H. Yuan, Z. Wang, and S. Ji, "Pixel transposed convolutional networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 5, pp. 1218–1227, 2019.

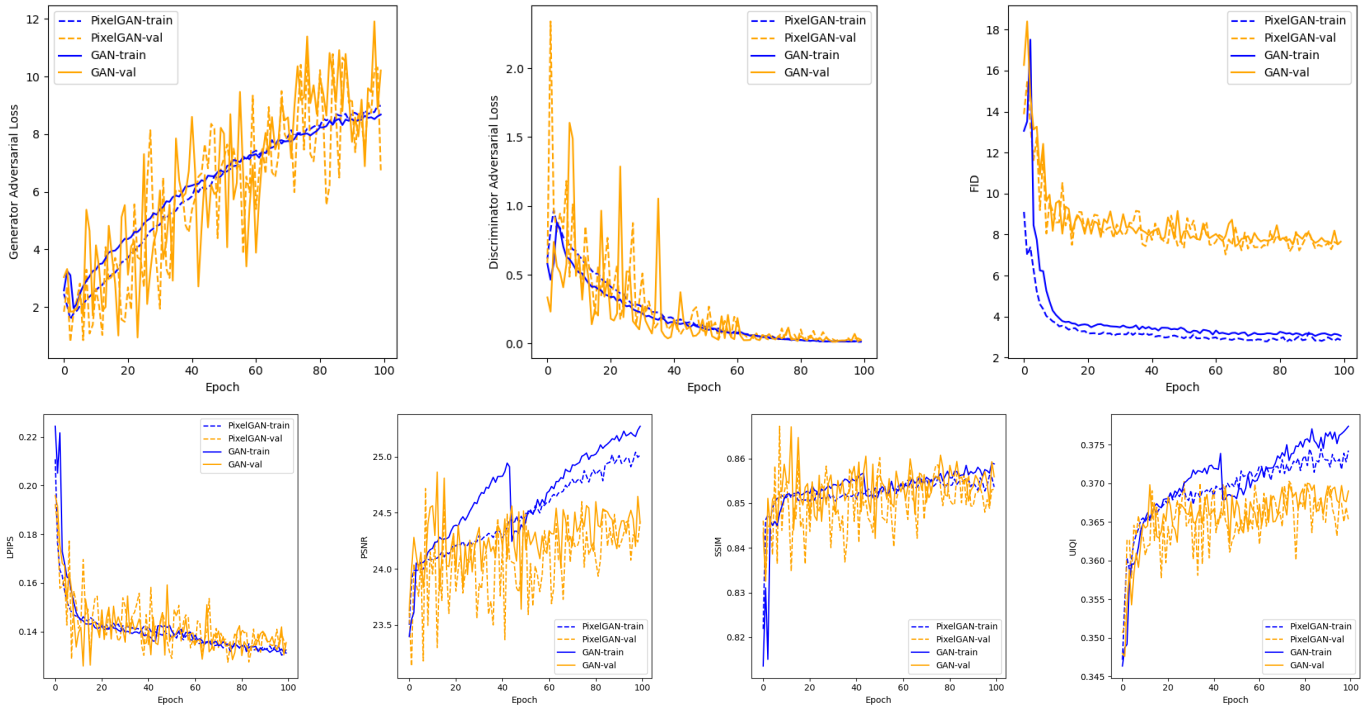

Fig. 3: Generator and discriminator adversarial loss for GAN and PixelTCL (PixelGAN) (top left and center top). Evolution of perceptual losses during training (top right and bottom row)

Fig. 4: From left to right: input, true, and multiple generated outputs of the same image. The top row shows the results of the GAN while the bottom row shows the ones obtained from the PixelTCL

[24] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[25] I. Loshchilov and F. Hutter, "Sgdr: Stochastic gradient descent with warm restarts," 2016.

[26] "WGAN-GP in CycleGAN model." https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/issues/1103#issuecomment-663916779. [Online; accessed 10-February-2023].

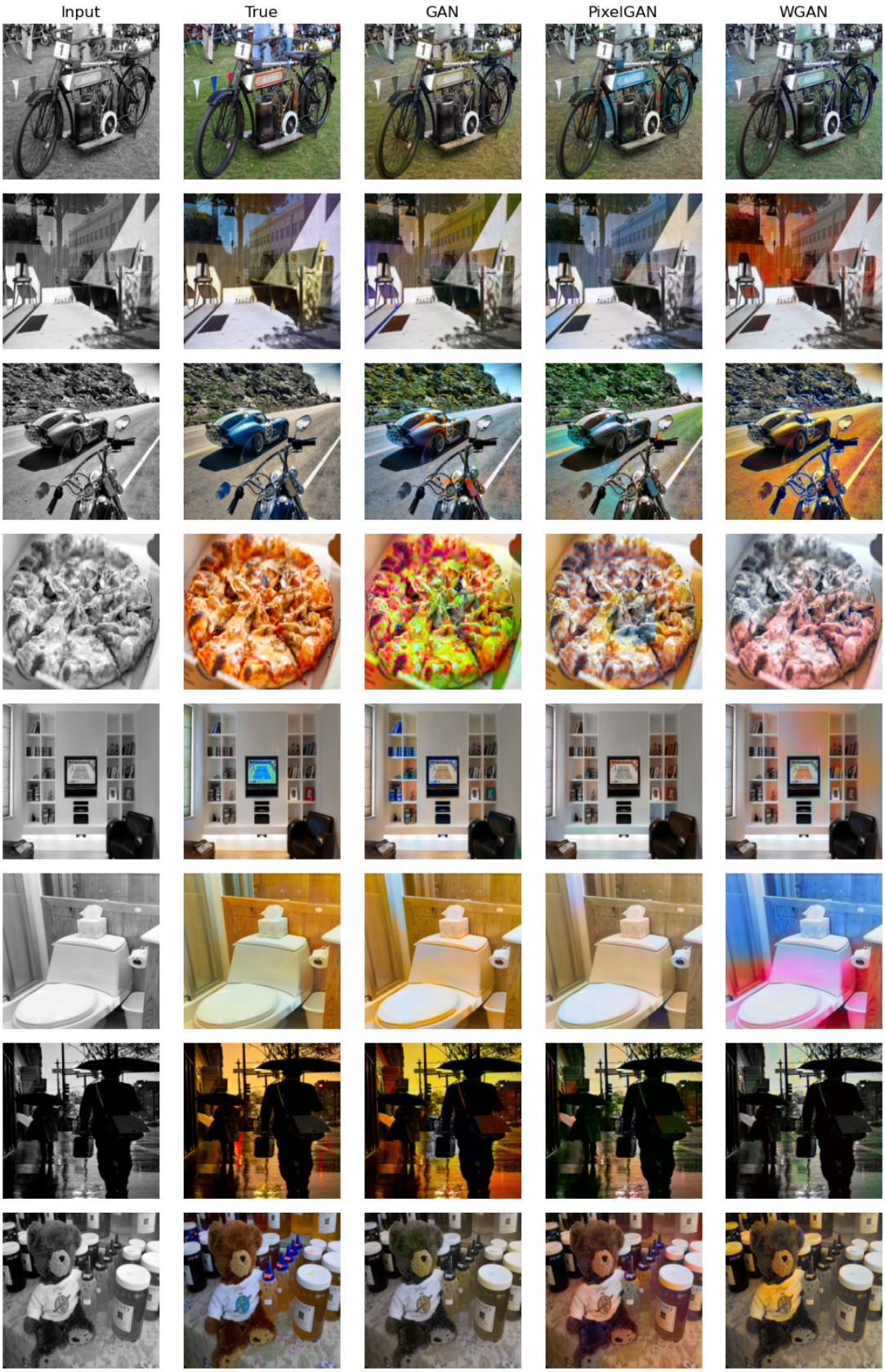

Fig. 5: Sample of generated images from train set for GAN, PixelTCL (PixelGAN) and WGAN

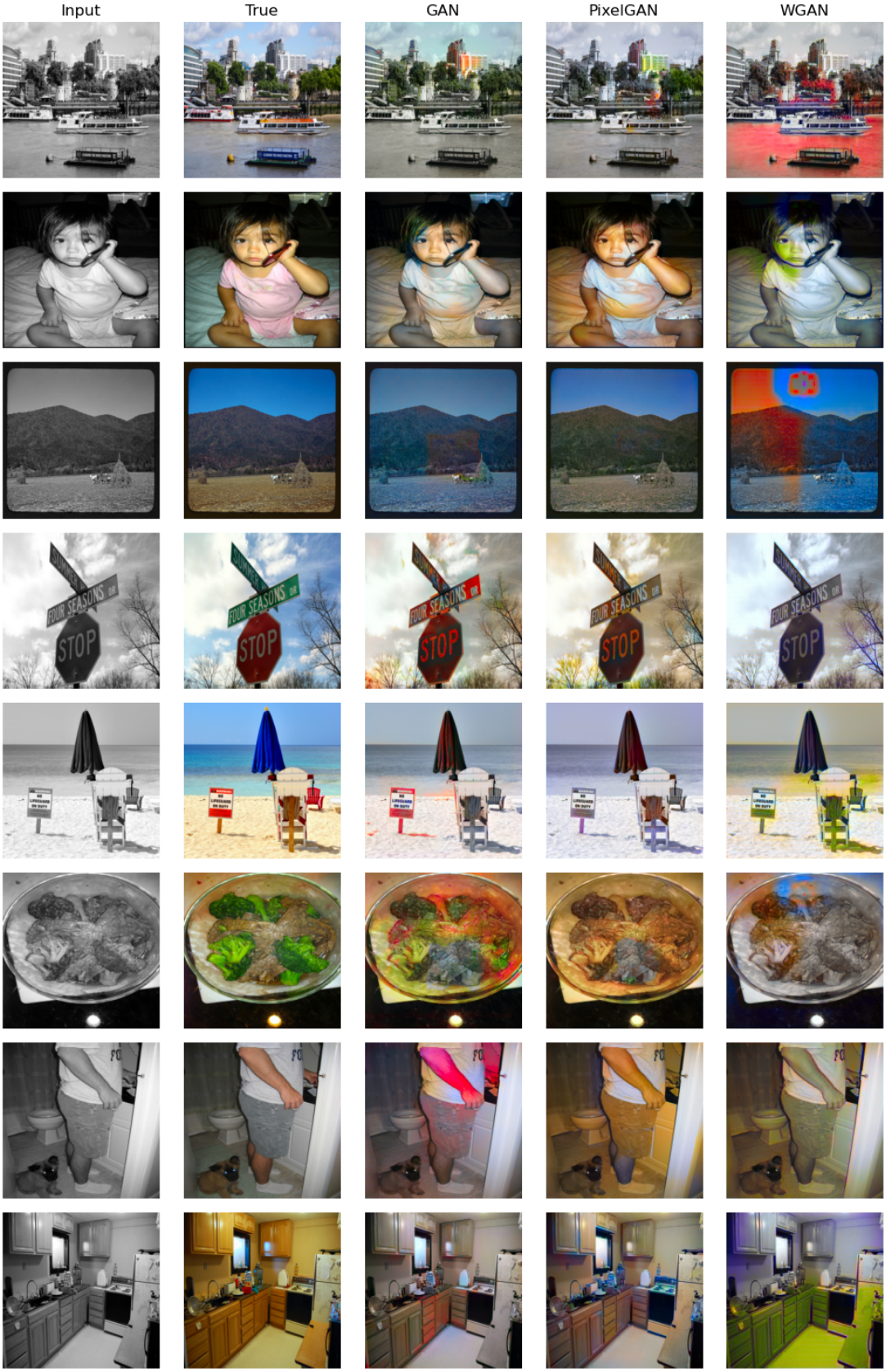

Fig. 6: Sample of generated images from test set for GAN, PixelTCL (PixelGAN) and WGAN