Image Colorization using Conditional Generative Adversarial Networks

Manuel Barusco[†], Riccardo Rampon[†]

***Abstract*—Machine Learning and, in particular, Neural Networks and Generative Neural Networks are becoming more and more pervasive in several fields. Generative Adversarial Networks (GANs) and Condition Generative Adversarial Networks (CGANs) have brought an innovative approach to the development of generative models and are used in a lot of fields especially in the Computer Vision and Image Processing fields. The GAN training framework can be used in fact in a lot of "image-to-image translation problems" such as semantic segmentation and grayscale image colorization as reported by [1]. Starting from that work, we are going to test the suggested CGAN architecture (composed by a standard Unet generator [2] and a Patch Discriminator) in the grayscale image colorization task and we try to improve it by using data augmentation, perceptual losses and a new Unet generator composed by a ResNet18 downsampling backbone and PixelShuffle layers in the upsampling path. The used GAN architectures and their training will be described in their entirety starting from the preprocessing phase of the dataset to the training strategies and losses used. Finally, the obtained generators will be evaluated and a final comparison between them will be done in order to see which model is performing better.**

***Index Terms*—Unsupervised Learning, Optimization, Neural Networks, Generative Adversarial Networks, Conditional Generative Adversarial Networks, Image Recoloring, Image Colorization.**

I. Introduction

The advent of Deep Learning in the Computer Vision and Image Processing fields has made possible to solve complex and advanced problems of semantic image analysis and understanding. As noticed by [1], "Many problems in image processing, computer graphics and computer vision can be posed as translating an input image into a corresponding output image", so many tasks involve passing from one image representation to another one. These representations are correlated and are the result of a semantic comprehension of the image. These problems are called "image-to-image translation problems" [1] and in this work we are going to consider one of these: the image colorization problem.

The purpose of this problem/task is to color a given gray-scale image in order to obtain a colorized version of it. This is an ill-posed problem because there exists many ways to color a given gray-scale image, and all of these can be good. Furthermore, the problem of image colorization has numerous practical applications, including photo enhancement, video color correction, animated film and cartoons creation [3] and the creation of alternative versions of existing images.

[†]Department of Information Engineering, University of Padova,
email: `manuel.barusco@studenti.unipd.it`
email: `riccardo.rampon@studenti.unipd.it`

Starting from the work of [1] we trained different generators in a Conditional Generative Adversarial Network framework. These generators will color gray-scale images in order to obtain images that are perceptually good, well colored and with sharp edges by avoiding blurry outputs.

The contributions of this paper are the following:

- we create and train a Vanilla Conditional GAN based on a Unet generator and a Patch Discriminator as suggested by [1] and we evaluate the Unet generator performances after training,
- we try to improve the previous network training by applying some data augmentation techniques on the dataset used,
- we try to improve the Vanilla Conditional GAN training by adding a perceptual loss term during the training,
- we try to improve the Vanilla Conditional GAN Network by using a different architecture: the generator is a Unet where the down-sampling part is composed by a ResNet18 backbone and the up-sampling part is done by using PixelShuffle layers starting from [4] work.

This is also the order of how we dealt with the problem and how we arrived to best solution.

The paper is organized as follows: Section 2 describes related work in the image colorization task, Section 3 describes the entire pipeline of our work, Section 4 describes how data have been processed, treated and augmented, Section 5 describes our models in details, Section 6 presents the results, Section 7 draws the conclusions and presents observations and future research directions, finally Section 8 presents the experience that has been gained through the project and the contributions of each group member.

II. Related Work

During the last years the problem of image colorization has attracted significant attention from the research community and various approaches have been proposed, ranging from traditional optimization methods to deep learning-based techniques.

The first approaches based on traditional optimization methods treated the problem as a regression task where the model try assign to each pixel a color onto the continuous color space or to quantized color values. An interesting approach among these is the Colorization algorithm proposed by Levin et al. [5]. This method uses a graph-based optimization method to find the optimal color assignment to a gray scale image.

With the advent of Deep Learning the usage of Neural

Networks int the task increased. One of the biggest contribution to the problem by using Neural Networks was given by [6] where they dig into the main problems of this task: its uncertainty, its underconstrained nature and the difficulty in testing the final model performances. They approach the problem as a pixel classification task using a CNN and by using some tweaks in the loss functions in order to avoid the desaturation effect and enhance the image colors and their variety.
Furthermore, with the advent of Deep Learning the task began to be semantic-aware and based on a semantic comprehension of the image before the actual colorization phase. This is done in order to color all the shapes and patterns in a meaningful and corrected way and in order to preserve the semantic structure and meaning of the grayscale image in the relative output colored image. One example of this is the Semantic Image Synthesis with Spatially-Adaptive Normalization algorithm by Park et al. [7], which uses a Neural Network to generate the output image while considering the semantic segmentation of the input image.
Another approach, and one of the most interesting, is to view the colorization problem as an "image-to-image translation problem" as viewed by Isola et al. [1], which used Generative Adversarial Networks (GANs) [8] and Conditional Generative Adversarial Networks (CGANs) frameworks to learn the mapping between an input and output image. This is a general approach that can be used for several tasks as semantic segmentation, image colorization and so on and all its power derived from the usage of a GAN training and of a learned task-adaptive GAN loss for the task. Our work is entirely based on this last approach.

III. Processing Pipeline

Our CGANs have been developed following this sequence of steps:

1) Processing of the dataset before the training phase and division into training and test set. This phase can contain also the data augmentation part.
2) Definition of the CGAN architectures: generators and discriminators models definition.
3) Loss functions definition and training of the models.
4) Visualization of the generators performances in the test set.

Dataset: The dataset that we use is a shortened version of COCO Dataset [9]. The dataset is composed by 21837 RGB images of a lot of subjects and scenarios so it's quite good for our task. The dimension of every image is not fixed.
Processing: The dataset is a general purpose dataset so it is processed in order to fit our task and our CGAN models. In one of our contributions we also add a data augmentation phase in this part. All the details will be given in Section 4.
Models: All our work is based on Conditional Generative Adversarial Networks (CGANs) that are composed by a generator and a discriminator. We test two generators: a Unet generator and a Unet [2] with a downsampling backbone composed by a ResNet18 and up-sampling part done by using PixelShuffle layers. The discriminator is instead composed by a Patch Discriminator. All the details will be given in Section 5.
Evaluation: After the training the CGAN generators are tested and evaluated on the test set. The metric used is the Pixel Accuracy. All the details will be given in Section 6.

IV. Signals and Features

The dataset used, as said before, is composed by 21837 RGB images of varying sizes. We decided to split it into 2 parts: 19900 images for the training set and 100 for the test set. Then, these images were resized to the 256 x 256 dimension in order to fit our networks and were transform from the RGB color space to the L*a*b color space. As in the RGB color space, also in the L*a*b color space we have three numbers for each image pixel but these numbers have different meanings. The first number (channel) L, encodes the lightness of each pixel and when we visualize this channel (the second image in figure below) it appears as a grayscale image. The a and b channels encode instead how much green-red and yellow-blue each pixel is respectively. In the following image you can see each channel of L*a*b color space separately.

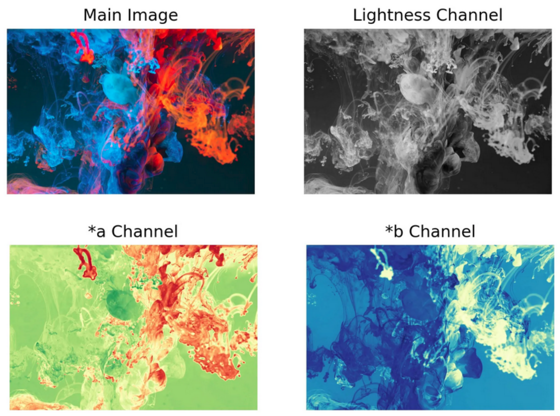

Fig. 1: The L*a*b color space

The main reasons for using this color space are:

- For every image we can split it into the three channels and use the L channel as the input of our generator. Our generator will then output the ab channels and by concatenating the L and ab channels we will get our final generated image in a very fast way.
- For every image pixel the generator has to predict two numbers and not three as in the case of the RGB space. So the task in the RGB color space is more difficult and unstable due to the more possible combinations of three numbers compared to two numbers that a pixel can assume.

This is the standard processing that we apply in our dataset but we also tried to do some data augmentation on the dataset in order to obtain a bigger dataset and with more image variance. We decided to apply an online data augmentation, so the image transformations defined in our data augmentation strategy are applied during the batch sampling, in order to save memory (we are not creating the augmented images and storing them in the dataset). We decided to apply some simple random horizontal and vertical flipping to the images in order to obtain more variance. The choice to perform data augmentation was made not only because it is a step almost always used in Deep Learning but also because we noticed that the Unet generator during the training could not recognize humans body parts always in the correct way. For example, in one epoch the Unet generator colored human arms in the correct way, but after a few epochs, if the arm was in a strange position (for example a bent arm) or if the human was photographed upside down, the generator colored the arm not in a good way, as if it doesn't recognize it. So we did data augmentation in order to make the Unet generator stronger in the patterns and shapes recognition.

V. Learning Framework

In this section we are going to explain which models, losses and training strategies we choose to train our Conditional GANs.

Models: The tested Conditional GAN architectures are the following:

- Vanilla CGAN: standard Unet generator and Patch Discriminator
- Advanced CGAN: Unet with Resnet18 downsampling backbone and up-sampling part composed by PixelShuffle layers as generator and Patch Discriminator.

The structure of the Unet generator used in the Vanilla CGAN is the following:

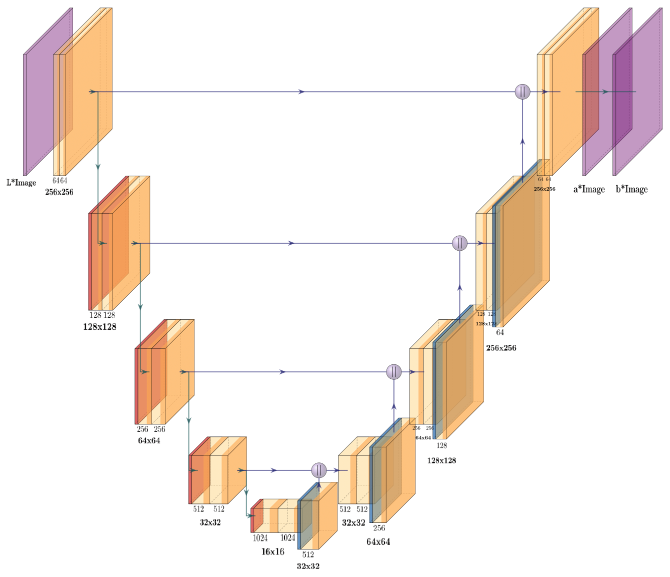

Fig. 2: Unet generator structure

The Unet generator structure is the standard one [2] but with some changes in order to fit our needs. The input layer accept 256 x 256 pixels one channel (the L channel) images and the down-sampling path is composed by a series of encoder blocks. Each of these blocks consists of 3 x 3 convolution layers with stride 1 for downsampling, each followed by batch normalization, ReLU activation function and Max Pooling with 2 x 2 kernel size. The number of channels are doubled after each block. The upsampling part is composed by a series of decoder blocks. Each of these blocks consists of a 2 x 2 transposed convolution layer with stride 2 for upsampling, concatenation with the activation map of the mirroring layer in the contracting path (skip connection), followed by batch normalization and ReLU activation function. The last layer of the network is a 1 x 1 convolution which is equivalent to cross-channel parametric pooling layer. We use the Tanh function for the last layer activation function as proposed by [1]. The output of the Unet is a tensor (image) with the same dimensions of the input tensor (image) but with two channels that are the a,b channels predicted by the network. The dimension of every tensor after every block is indicated in the figure.

As reported in [1], for many image translation problems, there is a great deal of low-level information shared between the input and output, and it would be desirable to shuttle these information directly across the net. In our case of image colorizaton, the input and output share the location of prominent edges, so, to give the generator a means to circumvent the bottleneck for information like this, the Unet generator has skip connections (previously mentioned) [2]. Specifically, there are skip connections between each layer i and layer $n-i$, where n is the total number of layers. Each skip connection simply concatenates all channels at layer i with those at layer $n-i$.

The colored image is composed by concatenating the L channel (given in input to the generator) with the a,b channels generated. The colored image is then passed as input to the Patch Discriminator.

The structure of the Patch Discriminator is the following:

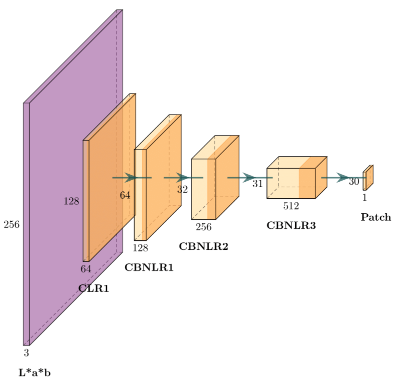

Fig. 3: Patch discriminator structure

The main point of the Patch Discriminator is that it states for every N x N pixels patch in the image if it is real or fake and it is a quite interesting implementation for a CGAN Discriminator. In fact, for this task, a normal discriminator that states if the input image is real or fake is not so precise in stating if the image is well colored and it doesn't take care about how well the image is colored locally for every little patch. The Patch Discriminator is composed by a series of convolution layers with 4 x 4 kernels, stride = 2 and 1 padding, LeakyReLU with negative slope = 0.2 and batch normalization. The dimension of every tensor after every block is indicated in the figure. The output of the Patch Discriminator is a tensor of dimensions 16 x 1 x 30 x 30 and the area of its receptive fields (the area of every square patch that it analyzes) is given by the equation:

$$area = \frac{image\,dimension}{30 \cdot 30} = \frac{256 \cdot 256}{30 \cdot\ 30} \approx 70 \qquad (1)$$

So our Patch discriminator analyzes patches of 70 pixels each. In both the Unet generator and Patch Discriminator we use batch normalization in order to make the training more stable, we don't use Dropout layers because we saw in [4] work that it doesn't improve the model performances and we observed in [1] that the introduction of noise with Dropout layers in the network produce only small stochasticity in the network output.

In the Advanced CGAN architecture we use the same Patch Discriminator but we introduce a new Unet generator architecture. This architecture is composed by a ResNet18 [10] as backbone in the downsampling path and the upsampling path is realized by using PixelShuffle layers with ICNR weights initialization. This generator is built by using the FastAI facilities. The ResNet18 backbone is pre-trained in the ImageNet Dataset [11] for a classification task, and we are going to use all its layers except the last two: GlobalAveragePooling and a Linear layer that are used for the ImageNet classification task. The PixelShuffle layers receive the tensors to be upsampled and the activations features from the intermediate layers of the down sampling path. The main idea of this generator implementation is that it exploits two fundamental aspects:

- the ResNet18 is a very deep and powerful residual network that can extract more features w.r.t the Unet downsampling path previously defined. This network can go very deep thanks to the Residual Learning [10] that reduces the effect of Vanishing Gradients.
- the PixelShuffle layers that implement the Sub-pixel convolution layers that were introduced for the first time in the Single Image and Video Super-Resolution task [12].

The PixelShuffle layers receive the output tensors of the bottleneck part, thus tensors of small dimension but with a lot of channels learn how to re-arrange all the elements in a new tensor with less channels but higher dimension in order to obtain an high resolution colored output without checkerboard artifacts. Figure 4 illustrate how these layers work, all the details are present in [12] and [13].

Losses: We based the training of the Vanilla CGAN and Advanced CGAN on the main objective of Conditional GANs

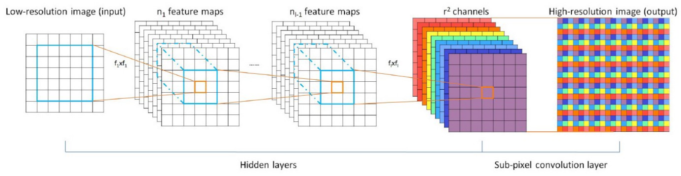

Fig. 4: PixelShuffle layers

[14]. So the main loss is:

$$\begin{aligned}\mathcal{L}_{cGAN}(G, D) =& \mathbb{E}_{x,y}[\log D(x, y)] + \\ & \mathbb{E}_{x,z}[\log 1 - D(x, G(x, z))]\end{aligned} \qquad (2)$$

where z is the random noise vector, x is the conditioning variable (in our case the observed image) and y is the output (in our case the generated image). The generator (G) tries to minimize this objective, instead the discriminator (D) tries to maximize it. So, we want to obtain:

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

We talk about Conditional GANs and not about standard GANs because a standard GAN is not applicable to this task. A standard GAN will in fact produce a colored image from a noise vector and no other side information. We want in fact that the generation process will be conditioned on the grayscale image that is in the input of our GAN. The grayscale image is actually conditioning our models to take care of the L channel that is given in input. In fact we want that the output image preserves all the shapes and features that are present in that channel. Since no noise is actually introduced in the network, the input of the generator is treated as zero noise vector with the grayscale image as a prior: mathematically speaking we write $G(x, \mathbf{0}_z)$. So the two previous formulas become:

$$\begin{aligned}\mathcal{L}_{cGAN}(G, D) =& \mathbb{E}_{x,y}[\log D(x, y)] + \\ & \mathbb{E}_{x}[\log 1 - D(x, G(x, \mathbf{0}_z))]\end{aligned} \qquad (4)$$

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) \qquad (5)$$

The discriminator job is distinguish real samples from the fake ones while the task of the generator is to fool the discriminator. The loss just introduced helps to produce good-looking colorful images that seem real, but to further help the models we combine this loss function with the L1 Loss of the predicted colors compared with the actual colors. So the generator has to fool the discriminator and has also to produce colors that are near the ground truth in a L1 sense:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y}[\|y - G(x, \mathbf{0}_z)\|_1] \qquad (6)$$

Other experiments tried also to use a L2 loss instead, but the results showed the increasing of blurry results [1] [4], caused by the Mean Squared Error factor that is introduced to the loss, and more grayish colors. So the wanted and best generator is:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \qquad (7)$$

If we use L1 loss alone, the model still learns to colorize the images but it will be conservative and most of the times it will use colors like gray or brown because when it doubts which color is the best, it takes the average and uses these colors to reduce the L1 loss as much as possible. By using the L1 loss in combination with the standard CGAN loss we will obtain less blurry, much sharper and colored outputs [1] [4] [15]. In the previous equation the λ parameter balance the contribution of the two losses to the final loss (we use $\lambda = 100$).
The loss used for the discriminator training is the Cross Entropy loss and is calculated calculated as:

$$\mathcal{L}_{CE} = -\sum_{i=1}^{n} (t_i \cdot \log(p_i)) \tag{8}$$

for n classes where t_i is the truth label and p_i is the softmax probability for the i^{th} class. So we calculate the cross entropy loss for our discriminator when we passed to it real and fake images, we calculate the two losses separately and we take the mean of the two values.
In order to improve the perceptual quality of the results we also try to introduce a perceptual loss term in our final loss. The perceptual losses encourages natural and perceptually pleasing in the results [16], these losses are very useful in tasks as image restoration and we decided to use them in our task in order to obtain a good perceptual quality in the generated images. Given a backbone Neural Network that extracts image features, these losses compare the activations features of a reference image to those of a generated image in order to ensure that the generated image is visually similar to the reference image in terms of content, style, and texture. The perceptual loss is usually calculated as the mean squared error (MSE) between the activations of the reference image and generated image in some layer of the backbone Neural Network. Given that the generator is generating the a,b channels and the generated image is built by concatenating the L channel (generator input) with the a,b generated channels, the L channel will impress to the final image a lot of features, that can be preserved or not by the generated a,b channels. So, in order to push our generator to preserve the features of the input image in the a,b generated channels, we calculate the perceptual loss between the generated image and the real image by replacing the L channel of both images with an L channel with constant value (0). This is done in order to push our generator to generate a,b channels that preserve the features of the input grayscale image. By minimizing also the perceptual loss term the network can be trained to produce colored images that are very close to the real one. So we decided to train a version of the Vanilla CGAN also with the addition of this loss term:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda\mathcal{L}_{L1}(G) + \mathcal{L}_{perc}(G) \tag{9}$$

In our implementation we decided to use the VGG network [10] as Neural Newtwork backbone for the perceptual loss.
Training: Each model has been trained for multiple epochs to get the optimal results. In particular, we decided a batch approach having the following characteristics:

- Batch size = 16
- Epochs = 80

In order to have a more stable training we decide to use Adam Optimizer [17] with the default values for its parameters. In general we saw some results after the first 30 epochs. During the train loop the Generator produces a fake image, then the discriminator is used to classify real images (drawn from the training set) and fakes images (produced by the generator). The cross entropy loss is calculated as shown above for the discriminator, and the CGAN loss plus L1 term (plus perceptual loss term eventually) is calculated for the generator. Before the training both discriminator and generator weights are initialiazed by usign a normal initialization instead of a Xavier initialization solution. We noticed in fact that sometimes a Xavier initialization causes the discriminator to become too strong so the generator cannot fool it and consequently it doesn't improve.
In the following we show the training losses obtained by training our solutions. We are going to show the training losses of:

1) Vanilla CGAN: Unet generator and Patch discriminator with standard CGAN loss with L1 term,
2) Vanilla CGAN: Unet generator and Patch discriminator with standard CGAN loss with L1 term and data augmentation,
3) Vanilla CGAN: Unet generator and Patch discriminator with standard CGAN loss with L1 term and perceptual loss term,
4) Advanced CGAN with standard CGAN loss with L1 term.

First Solution: In Fig. 5 you can see how the discriminator and generator CGAN losses vary across the epochs and in Fig. 6 how the Generator CGAN loss + L1 loss term vary across the epochs.

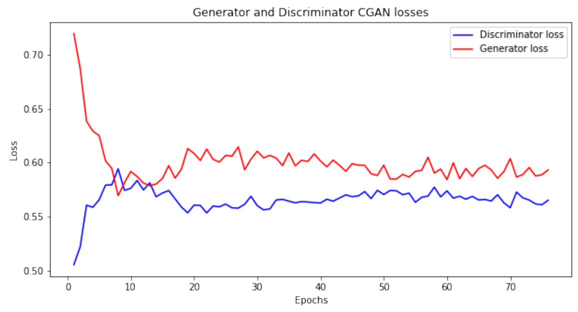

Fig. 5: Discriminator and Generator CGAN losses

Second Solution: In Fig. 7 you can see how the discriminator and generator CGAN losses vary across the epochs and in Fig. 8 how the Generator CGAN loss + L1 loss term vary across the epochs.
Third Solution: In Fig. 9 you can see how the discriminator and generator CGAN losses vary across the epochs and in Fig.

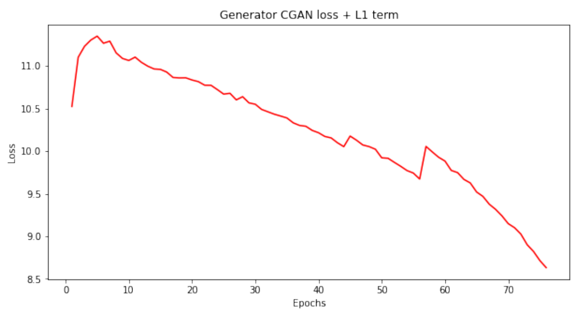

Fig. 6: Generator CGAN loss + L1 loss term

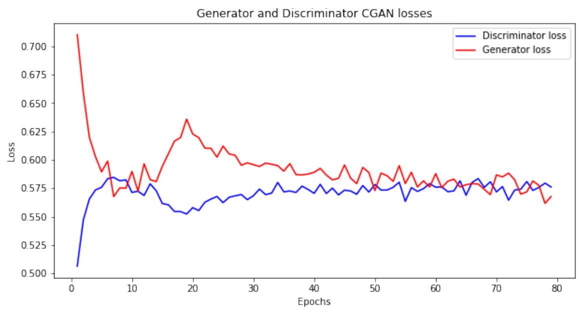

Fig. 7: Discriminator and Generator CGAN losses

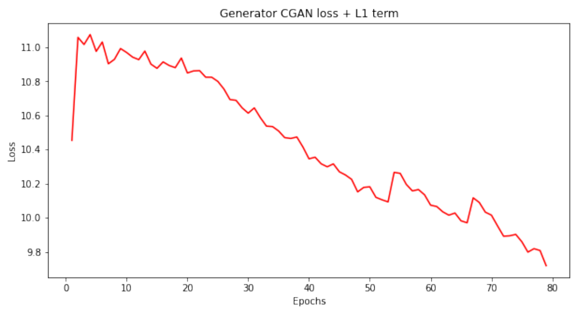

Fig. 8: Generator CGAN loss + L1 loss term

10 how the Generator CGAN loss + L1 loss term + Perceptual loss term vary across the epochs.

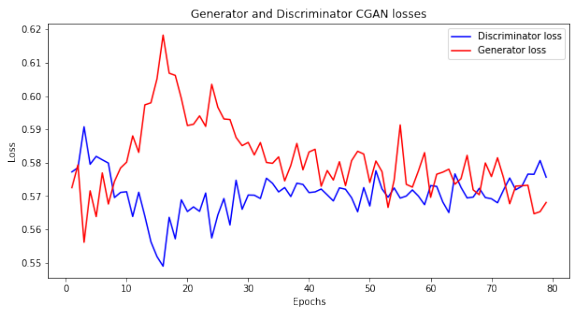

Fig. 9: Discriminator and Generator CGAN losses

Fourth Solution: In Fig. 11 you can see how the discriminator and generator CGAN losses vary across the epochs and in Fig. 12 how the Generator CGAN loss + L1 loss term vary across the epochs.

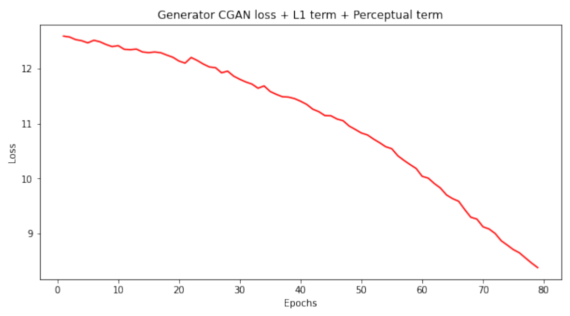

Fig. 10: Generator CGAN loss + L1 loss term

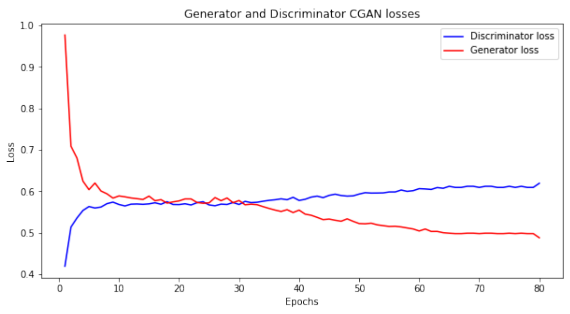

Fig. 11: Discriminator and Generator CGAN losses

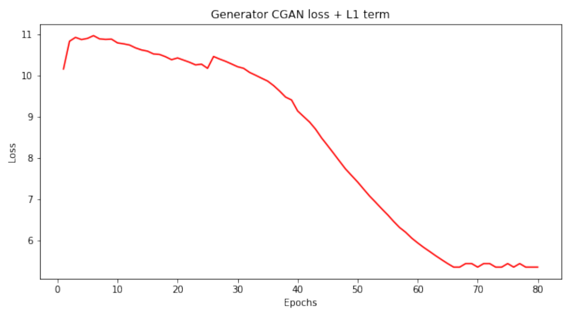

Fig. 12: Generator CGAN loss + L1 loss term

Testing: Testing generative models is not an easy task because we are in an unsupervised task so, in order to have a numeric performance indicator, we decided to evaluate the trained generators by using a Pixel Accuracy metric. The Pixel Accuracy, for our purposes, is given by the ratio between the number of pixels in the generated a,b channels that have the same color values as the source image a,b channels and the total number of pixels. Any two pixels are considered to have the same color if their underlying color channels lie within some threshold distance indicated as a percentage. Mathematically speaking:

$$acc(\boldsymbol{X}, \boldsymbol{Y}) = \frac{1}{2dim(\boldsymbol{Y})} \sum_{c=2}^{3} \sum_{i=1}^{dim(Y)} (\mathbf{1}_{[0,\epsilon_c]}(|\boldsymbol{X}_{i,c} - \boldsymbol{Y}_{i,c}|)) \tag{10}$$

where $\boldsymbol{X}, \boldsymbol{Y}$ are respectively the generated and the real image and $dim(Y)$ is the number of pixels in a channel of the real (and also generated) image. $\mathbf{1}_{[0,\epsilon_c]}$ is the indicator function and ϵ is a threshold distance used for each color channel. The total accuracy is calculated as:

$$acc = \frac{1}{n}\sum_{i=1}^{n}(acc(\boldsymbol{X}_i, \boldsymbol{Y}_i)) \tag{11}$$

where n is the test set dimension.

VI. Results

In this section we present the generators performances by providing their Pixel Accuracy in the test set with varying thresholds for considering if two pixels have the same color. A threshold equal to 2% means that we are considering a tolerance of the 2% in the given channel color range when considering if two pixel have the same colors. We refer to the various generators by indicating the solution where have been trained (for example we indicate Vanilla CGAN solution and we are actually referring to the Unet generator that we trained in that solution). In the following table you can see the obtained Pixel Accuracies of the trained generators in the test set:

Model	**Threshold 2%**	**Threshold 5%**
Vanilla CGAN	67%	40%
Vanilla CGAN with Data Augmentation	69%	42%
Vanilla CGAN with Perceptual losses	74%	44%
Advanced CGAN	80%	50%

In the Figure 13, 14, 15, 16 you can see how the trained generators actually color the images. In every image the first row is composed by the grayscale images sampled in the dataset, the second row is composed by the colored images and the last row is the ground truth (real colored images sampled in the dataset).

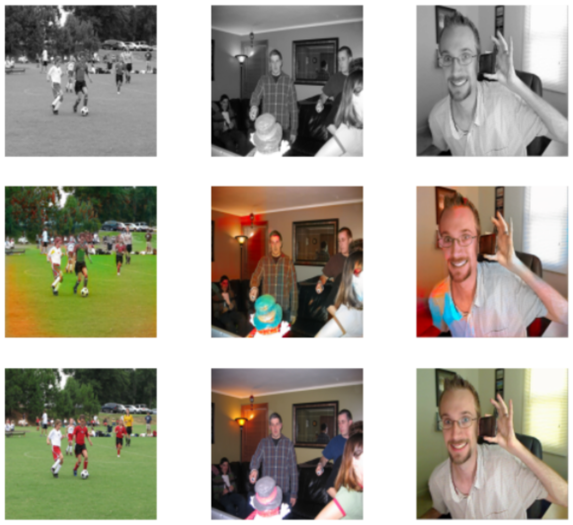
Fig. 13: Vanilla CGAN Unet generator colorization

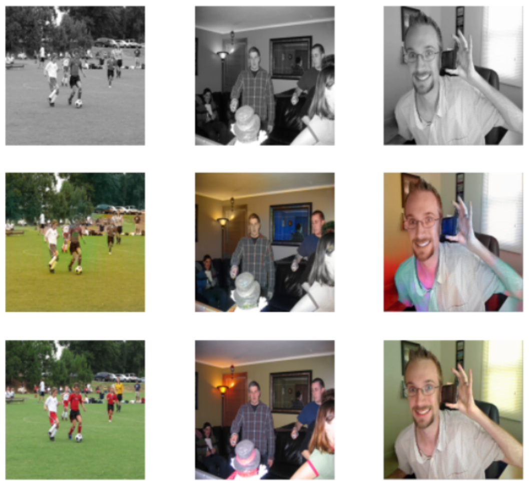
Fig. 14: Vanilla CGAN with Data Augmentation Unet generator colorization

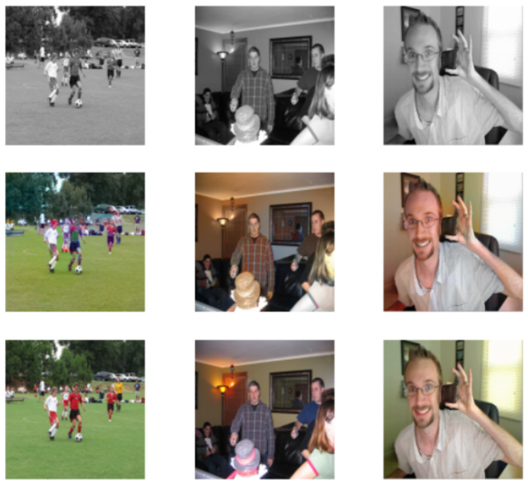
Fig. 15: Vanilla CGAN with perceptual loss Unet generator colorization

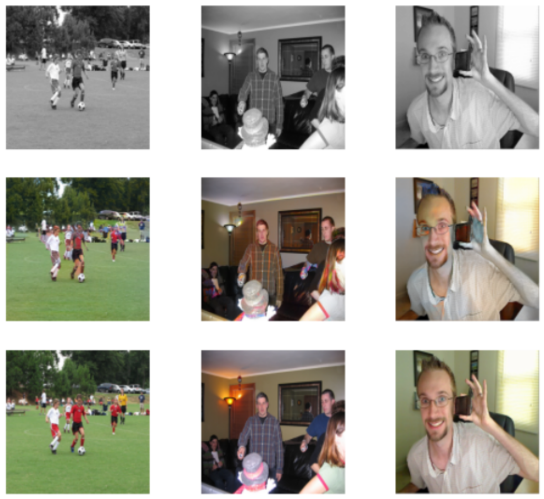
Fig. 16: Advanced CGAN generator colorization

As you can see from these images, from a qualitative point of view, the last two solutions color the images in a quite good way. In general all the trained generators work very well when they color environments such as rooms, houses interiors and images with low light, low color variance and with few subjects. The last two solutions perform well also when there is a lot of color variance, more subjects and they color the image with bright colors, unlike the former two solutions that tends to color the image with a sepia effect or with dull colors when the models are undecided about how to color the scene. Others colored test images can be seen in the Google Colab Notebooks of the project.

VII. Concluding Remarks

In this paper we have investigated how Conditional Generative Adversarial Networks can be applied to solve the problem of Image Colorization. In particular, we have tested various solutions that includes:

- Two different generators: a standard and an advanced Unet generator,
- Data augmentation in the chosen dataset,
- Different losses: standard CGAN loss + L1 term, standard CGAN loss + L1 term + perceptual loss term.

We have compared the solutions performances in the test set and we showed that the Advanced CGAN is the best solution with the highest Pixel Accuracy. In the future, we want to improve our work with these ideas:

- use a bigger dataset or new data augmentation techniques,
- train the models with more epochs,
- introduce a λ_{perc} variable with the perceptual loss term (in the complete training loss function) in order to trade its importance in the generator loss and test the generator performances with λ_{perc} that varies,
- test the quality of the results in the test set by using a well trained discriminator that can states if an image is real or fake, or by involving some people in the classification.

VIII. Experience gained and contributions

What we have learnt: It has been a great pleasure to be part of this project, which has taught us the following:

- CGAN training is very tricky and unstable, we could observe with our eyes how, especially in the first epochs, the generator and discriminator losses swing.
- Training very deep Neural Networks can take a lot of time, specially in the CGAN framework. Furthermore is very difficult to understand if the model is underfitting or overfitting without wasting a lot of time. It is also difficult to find good ideas that can improve the model.
- Writing a good and understandable paper can be really challenging because fitting everything in a small amount of pages requires to be very straight to the point without giving too many concepts for granted.
- Writing modular and easy debuggable code is fundamental in Neural Network development because if there is a bug it is very difficult to find it, specially during the training.
- The memory management is crucial, we have learnt how to treat in a right way by managing the batch sizes, the loss gradients tracking and the CUDA memory cleaning.

Contributions: We report here the contributions from each member of the group:

- Manuel Barusco
  - Dataset pre-processing phase and Data Augmentation
  - Patch Discriminator and Advanced Unet implementation
  - Models training and testing
  - Paper writing
- Riccardo Rampon
  - Dataset pre-processing phase
  - Basic Unet, Perceptual losses and Advanced Unet implementation
  - Models training and Utility functions
  - Paper writing

The Google Drive folder with all the weights of our trained models is available here

References

[1] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” 2016.

[2] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” 2015.

[3] Q. Fu and W.-T. Hsu, “Colorization using convnet and gan,” 2017.

[4] “Colorizing black white images with u-net and conditional gan.” https://towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8. Accessed: 2023-06-02.

[5] A. Levin, D. Lischinski, and Y. Weiss, “Colorization using optimization,” *ACM SIGGRAPH 2004 Papers*, 2004.

[6] R. Zhang, P. Isola, and A. A. Efros, “Colorful image colorization,” 2016.

[7] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu, “Semantic image synthesis with spatially-adaptive normalization,” 2019.

[8] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014.

[9] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. DollÃ¡r, “Microsoft coco: Common objects in context,” 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[12] W. Shi, J. Caballero, F. HuszÃ¡r, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, “Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network,” 2016.

[13] W. Shi, J. Caballero, L. Theis, F. Huszar, A. Aitken, C. Ledig, and Z. Wang, “Is the deconvolution layer the same as a convolutional layer?,” 2016.

[14] M. Mirza and S. Osindero, “Conditional generative adversarial nets,” 2014.

[15] K. Nazeri, E. Ng, and M. Ebrahimi, *Image Colorization Using Generative Adversarial Networks*, pp. 85–94. 06 2018.

[16] “Perceptual losses for deep image restoration.” https://towardsdatascience.com/perceptual-losses-for-image-restoration-dd3c9de4113. Accessed: 2023-06-02.

[17] “Adam: A method for stochastic optimization.” https://arxiv.org/abs/1412.6980.