Image Recoloring with conditional GANs

Riccardo De Monte[†], Simone Cecchinato[†], Mattia Secchiero [†]

***Abstract*—We investigate the image recoloring problem by using of adversarial training with both cGAN and WGAN approaches. In particular, we obtain interesting results considering the importance of choice for both the dimension of the dataset and the batch size. In fact, the typical approach is to use a batch size as big as possible. However this is not always the case, especially when dealing with GANs. Moreover, in order to compare the models obtained with different hyper-parameters and training, we have implemented three metrics. Since these metrics are not evaluating the recoloring process in the same way, we don't have a concordant result for all three of them, but we need to weight the results together to determine the best model.**
All the code and models are available on GitHub at: Image_Colorization_code

***Index Terms*—Deep Learning, Image Colorization, Neural Networks, cGAN, WGAN, Adversial training.**

I. Introduction

Nowadays deep learning is becoming the state of the art for generating synthetic data, in particular GANs and diffusion models are the most popular techniques used for image generation. Image recoloring, namely generating an RGB image from a gray-scale one, is one possible example of task that can be solved by exploiting these approaches. In particular, impressive results have been reached in [1] by use of conditional GANs, namely cGANs, proving the effectiveness of Adversarial training in enriching colorization obtained by only using CNNs and classical regression approaches.

One of the problems of adopting GANs, but also other kind of generative models, is evaluating the results objectively by use of formal metrics like the accuracy for a classification task. The most used metrics for image generation are the *Inception Score* (IS) and *Frechet Inception Distance* (FID). However these metrics are not specifically designed for our purpose, namely they do not take into account the importance of color for some images.

Moreover, a key feature needed to develop a well performing generator, is to have access to a very big training dataset, such as *ImageNet*, but this is not always the case.

In this work we try both to find a reliable metric for image evaluation and to obtain good results even with a small amount of image data by investigating the role of the batch size. Moreover, also WGAN ([2]) training have been taken into account as an alternative training procedure since in the literature has been considered to be very effective independently by the hyper-parameters and architectures for the NNs used.

[†]Department of Information Engineering, University of Padova, email: {riccardo.demonte, simone.cecchinato2, mattia.secchiero}@studenti.unipd.it

II. Related Work

In [1] in order to evaluate the resulting generators, for image recoloring, the Amazon service "Amazon Mechanical Turk" have been used. In particular a series of pair images, "real" and "fake", have been presented to the Turkers and they were given unlimited time to respond as to which was fake. Finally the percentage of fooled participants have been presented as evaluation for the resulting model.

The problem of this kind of evaluation is that unbiased results can be obtained; moreover if the aim is to compare a large number of different models this approach might be not suitable in practice. For this reason metrics like the FID and the IS might be more appropriate for large comparison.

Our evaluation methods differ from the before cited because they can be implemented very easily and at the same time they provide objective scores. Three metrics have been designed by following the intuitive idea that if a model has been trained for solving a classification task with a dataset with real images, then it should perform the same on synthetic data. In particular, one of them has been designed specifically for the image recoloring task.

III. Processing pipeline and Datasets

III.I Datasets

In order to train and test our models the images have been chosen from two datasets: a smaller version of COCO dataset with 21,837 images and a Kaggle dataset with 17,178 images containing 12 categories of animals. The different animal categories are: "butterfly", "cats", "cow", "dogs", "elephant", "hen", "horse", "monkey", "panda", "sheep", "spider", "squirrel".
The following datasets have been derived:

- $\mathcal{D}_{\text{GAN\_s}}$: this dataset contains 6,600 images from COCO and 3,000 images of animals, more specifically 250 images per class. The resolution used is 256×256.
- $\mathcal{D}_{\text{GAN\_b}}$: this dataset contains 10,800 images from COCO and 4,200 images of animals, more specifically 350 images per class. The resolution used is 256×256. These two last datasets have been used to train the generator in order to see the effects of using a bigger dataset.
- $\mathcal{D}_{\text{test}}$: this dataset contains 2400 images from the animals dataset, in particular 200 images per class. It is used to test the model performances. Those images are not included in the previous datasets, namely they are disjointed.

- $\mathcal{T}_{\text{test}}$: contains the corresponding labels for the test dateset $\mathcal{D}_{\text{test}}$
- $\mathcal{D}_{\text{tr-animals}}$, $\mathcal{T}_{\text{tr-animals}}$: these two sets contains 12,800 images and the corresponding labels used for training a classifier. In particular $\mathcal{D}_{\text{tr-animals}} \cap \mathcal{D}_{\text{test}} = \emptyset$
- $\mathcal{D}_{\text{test-2}}$: contains all the images from the animals dataset except for the ones used for training the generator, namely $\mathcal{D}_{\text{test-2}} \cap \mathcal{D}_{\text{GAN\_b}} = \emptyset$. $\mathcal{T}_{\text{test-2}}$ contains the corresponding labels. In total this dataset includes 12,978 images

The images provided as input to the classifier have a resolution of 224×224 pixels and no data-augmentation has been applied. Instead, for the images employed for the training task of the generators, it has been applied a random horizontal flipping.

III.II **Processing pipeline**

In order to evaluate our models we have used three different metrics. That implies the fact that three processing pipelines have been followed.

For the first metric, two models are needed: the generator to be tested, that we denote with G, and an image classifiers used to solve the classification problem for the 12 animal categories. We denote with C_c the classifier that takes as input colored images in the RGB format and predicts the corresponding class (animal category). The classifier will be trained by use of $\mathcal{D}_{\text{tr-animals}}$, $\mathcal{T}_{\text{tr-training}}$. Figure 1 shows a block diagram explaining which dataset is used for the different tasks

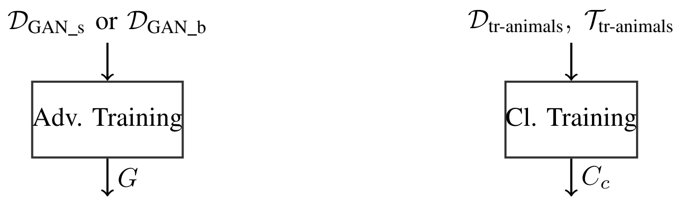
$\mathcal{D}_{\text{GAN\_s}}$ or $\mathcal{D}_{\text{GAN\_b}}$
Adv. Training
G
$\mathcal{D}_{\text{tr-animals}}$, $\mathcal{T}_{\text{tr-animals}}$
Cl. Training
C_c

Fig. 1: Models training scheme

More details on how the training blocks have been implemented, namely how the different models have been trained, will be clearly explained later.

Then by use of the dataset $\mathcal{D}_{\text{test}}$ and the corresponding labels $\mathcal{T}_{\text{test}}$, the accuracy can be computed. To analyze the effectiveness of the recoloring provided by G, the accuracy can be computed again by using as input images the ones in $\mathcal{D}^{\text{gray}}_{\text{test}}$ (gray images) recolored by G. A block diagram resuming this process is presented in Figure 2

Also for the second metric C_c is needed. In Figure 3 we preset the evaluation of a given model G by use of this metric. In particular, the testing images from $\mathcal{D}_{\text{test}}$ are transformed to a single channel gray-scale images and provided as input

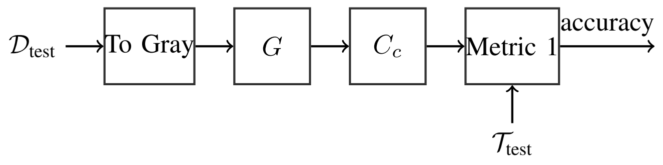

Fig. 2: Pipeline for metric 1

to a trained generator G. Then the fake colored images are classified by C_c and the corresponding outputs are used to evaluate the model.

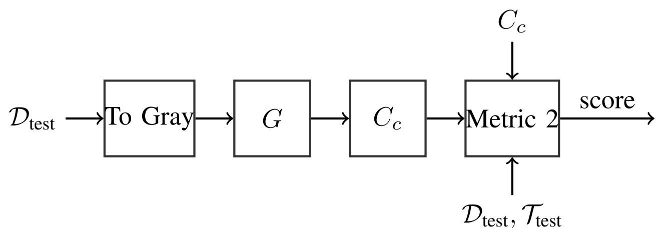
C_c
$\mathcal{D}_{\text{test}}$
To Gray
G
C_c
Metric 2
score
$\mathcal{D}_{\text{test}}, \mathcal{T}_{\text{test}}$

Fig. 3: Pipeline for metric 2

As presented in Figure 3, also the classifier C_c and the original testing images (and the corresponding labels) are needed for the computation of the score, a real number in the interval $[0, 1]$. A more detailed explanation of how the block "Metric 2" is implemented will follow.

For the computation of the third metric, denoted with $\mathbf{\Omega}$, the dataset $\mathcal{D}_{\text{test-2}}$ (and the corresponding labels) are used to train a classifier C^*. Then to compare different generators G_1 and G_2, two classifiers C_1, C_2 are trained using the fake images provided by the processing of $\mathcal{D}_{\text{test-2}}$ with G_1 and G_2 (namely the images have been converted to gray-scale and then recolored by the generators).

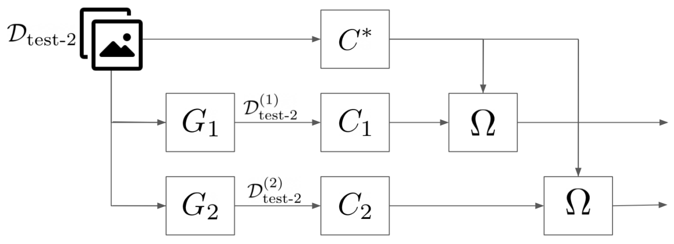
$\mathcal{D}_{\text{test-2}}$
C^*
G_1
$\mathcal{D}^{(1)}_{\text{test-2}}$
C_1
Ω
G_2
$\mathcal{D}^{(2)}_{\text{test-2}}$
C_2
Ω

Fig. 4: Scheme for the metric $\mathbf{\Omega}$

IV. **Method**

Image recoloring is the task of assigning RGB values to gray-scale pixels. Given the fact that we used the Lab color space we will actually have to find the ab values instead of the RGB ones. Since this is an ill posed problem (multiple colored images can produce the same gray-scale image) it can be tackled by a generative framework. In particular, conditional generative adversarial networks (cGANs) can be used as solution to image colorization. In this implementation a cGAN and a WGAN will be used to solve this task.

IV.I **Training**

By means of [3] we can get an accurate understanding of the GANs structure and how they work. In particular we see that generative adversarial nets consists of two "adversarial" models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came either from the training data or G. To learn a generator distribution p_G over the data $\boldsymbol{y}$, the generator builds a mapping function from a prior noise distribution $p_z(\boldsymbol{z})$ to data space as $G(\boldsymbol{z}, \boldsymbol{\theta})$, while the discriminator, $D(\boldsymbol{y}, \boldsymbol{w})$, outputs a single scalar representing the probability that $\boldsymbol{y}$ came form training data rather than p_G (i.e. 0 if the image is fake/generated, 1 if the image is true/not generated). G and D are both trained simultaneously: parameters for G are adjusted to minimize $\log(1-D(G(\boldsymbol{y}))$ and parameters for D are adjusted to maximize $\log D(\boldsymbol{y}) + \log(1 - D(G(\boldsymbol{z})),$ as if they are following the two-player min-max game with value function $V(G, D)$:

$$\min_G \max_D V(G,D) = \mathbb{E}_{\boldsymbol{y}\sim p(\boldsymbol{y})}[\log D(\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z}))] \tag{1}$$

In the following figure it is presented a simple scheme of the GAN training.

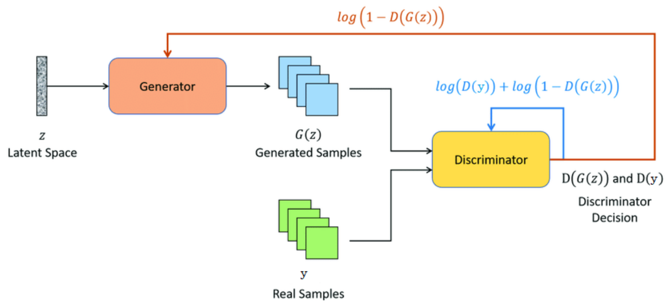

Fig. 5: GAN training

IV.I.I **cGAN**

Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra side information $\boldsymbol{x}$. $\boldsymbol{x}$ could be any kind of auxiliary information, but in our specific case it is the b&w image, from which start to begin the colourization. To be more precise, since we are using the *Lab* colour space, the output of the generator will actually be a tensor with the ab components, that together with x (the b&w L-part) will create the overall coloured image. By feeding $\boldsymbol{x}$ to the network as a side information we enforce it to produce a colorized version of that b&w image.

We can perform the conditioning by feeding $\boldsymbol{x}$ into both the discriminator and generator as additional input layer. In the generator the prior input noise $p_z(\boldsymbol{z})$ and $\boldsymbol{x}$ are combined into a joint hidden representation. In the discriminator $\boldsymbol{y}$ and $\boldsymbol{x}$ are presented as inputs to a discriminative function. The objective function of a two-player min-max game would be rewritten as :

$$\min_G \max_D V(G,D) = \mathbb{E}_{\boldsymbol{y}\sim p(\boldsymbol{y}|\boldsymbol{x})}[\log D(\boldsymbol{y},\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_z(\boldsymbol{z})}[\log(1-D(G(\boldsymbol{z},\boldsymbol{x}))] \tag{2}$$

By considering this min-max game, as just presented in (2), conditional GANs learn a mapping from the observed image $\boldsymbol{x}$ and random noise vector $\boldsymbol{z}$, to $\boldsymbol{y}$ as $G : \{\boldsymbol{x}, \boldsymbol{z}\} \rightarrow \boldsymbol{y}$.

It is also beneficial to mix the GAN objective with a more traditional loss, such as L1 distance. The discriminator's job remains unchanged, but in that way the generator is tasked to not only fool the discriminator but also to be near the ground truth output in an L1 sense.

As presented in [1] the final objective is

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

with

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{\boldsymbol{x},\boldsymbol{y},\boldsymbol{z}}[||\boldsymbol{y} - G(\boldsymbol{x},\boldsymbol{z})||_1] \tag{4}$$

It is worth noticing that without $\boldsymbol{z}$, the net could still learn a mapping from $\boldsymbol{x}$ to $\boldsymbol{y}$, but would produce deterministic outputs. To avoid that we can provide Gaussian noise $\boldsymbol{z}$ as an input to the generator, in addition to $\boldsymbol{x}$. In this implementation the noise is provided only in the form of dropout, applied on several layers of the generator.

Moreover, given that L2 and L1 losses fail to encourage high-frequency crispness, (but capture the low-frequency) it will be used another framework (*patchGAN*) to enforce correctness at the high frequencies. More details will follow in IV.II.I.

IV.I.II **WGAN**

In [2] an alternative training procedure for GANs has been presented. In fact as highlighted by the authors, the usual GANs training based on the minimax game is delicate and unstable. The main reason why GAN training can result unstable is the the use of the Jensen-Shannon (JS) divergence

$$\mathrm{JS}(p_G, p) = \mathrm{KL}(p_G||p_m) + \mathrm{KL}(p_G||p_m) \tag{5}$$

where p denotes the true distribution of data, p_G denotes the resulting distribution implicitly learnt by the generator G, $p_m = (p_G + p)/2$ and KL denotes the Kullback-Leibler divergence. In fact by assuming that the optimal discriminator has been obtained, minimizing the value function for GANs is equivalent to minimize the JS diveregence, but this can bring to a vanishing gradient, namely the generator is not able anymore to learn. For this reason the Earth-Mover (EM) or Wasserstein-1 distance has been proposed to substitute the JS divergence

$$W(p, p_G) = \inf_{\gamma \in \Pi(p_G, p)} \mathbb{E}_{\boldsymbol{y}_1, \boldsymbol{y}_2 \sim \gamma}[||\boldsymbol{y}_1 - \boldsymbol{y}_2||] \tag{6}$$

Even though this distance has better properties in terms of convergence, it is intractable and for this reason the following optimization problem has been derived by exploiting the Kantorovich-Rubinstein duality

$$\max_{\boldsymbol{w}\,:\,||D_{\boldsymbol{w}}||_L \le K} \mathbb{E}_{\boldsymbol{y}\sim p}[D_{\boldsymbol{w}}(\boldsymbol{y})] - \mathbb{E}_{\boldsymbol{y}\sim p_{G_{\boldsymbol{\theta}}}}[D_{\boldsymbol{w}}(\boldsymbol{y})] \tag{7}$$

where $||D_{\boldsymbol{w}}||_L \le K$ for some $K > 0$ means that the critic $D_{\boldsymbol{w}}$ (the discriminator) has to be K-Lipschitz. To force the critic to satisfy this property the authors proposes the use of weight clipping, but the clipping threshold c has to be tuned carefully otherwise it might bring to vanishing or exploding gradients. To tackle this problem, in [4] it is proposed to add a soft version of a new constraint called Gradient Penalty (GP) and the resulting function to maximize by the critic is

$$\mathbb{E}_{\boldsymbol{y}\sim p}[D_{\boldsymbol{w}}(\boldsymbol{y})] - \mathbb{E}_{\boldsymbol{y}\sim p_{G_{\boldsymbol{\theta}}}}[D_{\boldsymbol{w}}(\boldsymbol{y})] - \xi \mathbb{E}_{\hat{\boldsymbol{y}}}\left[(||\nabla_{\hat{\boldsymbol{y}}} D_{\boldsymbol{w}}(\hat{\boldsymbol{y}})||_2 - 1)^2\right]$$

where $\hat{\boldsymbol{y}}$ is uniformly sampled from the possible convex combinations of two samples, one from $p_{G_{\boldsymbol{\theta}}}$ and one from p.

For our purposes we would like to learn $p(\cdot|\boldsymbol{x})$, namely a conditional distribution, and for this reason the generator still maps couples $(\boldsymbol{z}, \boldsymbol{x})$ (noise and gray-scale image) to $\boldsymbol{y}$ (ab channels to Lab image) and the critic maps couples $(\boldsymbol{x}, \boldsymbol{y})$ to a real number. As done with the traditional GANs training, the loss for the generator is a mix of the WGAN loss introduced before and the L1 distance

$$-\mathbb{E}_{\boldsymbol{x},\boldsymbol{z}}[D_{\boldsymbol{w}}(\boldsymbol{x}, G_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{z}))] + \lambda \mathbb{E}_{\boldsymbol{x},\boldsymbol{z},\boldsymbol{y}}[||\boldsymbol{y} - G_{\boldsymbol{\theta}}(\boldsymbol{z}, \boldsymbol{x})||_1] \tag{8}$$

Similarly, also the loss for the critic is changed for the conditional framework.

IV.II **Architectures**

IV.II.I **Generator**

For the generator a variation of the *UNet* model, proposed by [5], is used. It is called in this way because it has a sort of "U" shape. The architectures is symmetric and consists of two major parts. A first path, the contraction path, (also called as the encoder) which is used to capture the context in the image. A second path, the symmetric expanding path, (also called as the decoder) which is used to enable precise localization using transposed convolutions. The layers in the encoder part are skip connected and concatenated with layers in the decoder part. This make the U-Nets use fine-grained details, learned in the encoder part, to construct an image in the decoder part.

Each block of the contracting part is formed by a convolutional layer (down-sampling), a batch normalization and a LeackyReLU layers. As is shown in Fig. 6, we start with an image input of size $256 \times 256 \times 1$, the L channel of the Lab format, and we reach the bottonmost layer, which mediates between the contraction layer and the expansion layer, with a depth of 512.

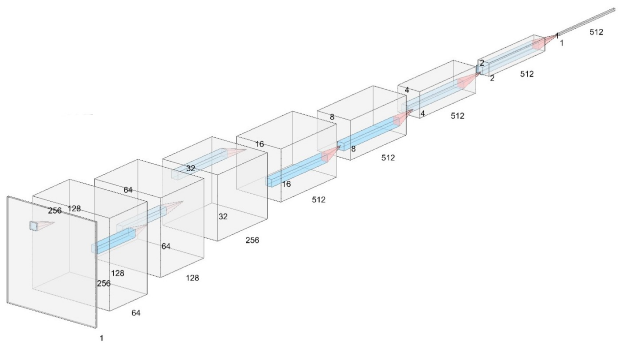

Fig. 6: Architecture contraction path for G

The expansive path is symmetric in the other direction. Every step consists of an up-sampling of the feature map followed by a convolution (up-convolution) that halves the number of feature channels. After the convolution layer, each block of the expansive path include a batch normalization and a ReLu layer. To introduce stochasticity, some blocks in the expansive layers also have Dropout, with probability $\rho = 0.5$. The purpose of this path is to enable precise localization combined with contextual information from the contracting path, using the skip connections. Our generator implementation differs from the one proposed in [1] only by the skip-connection: in our case they are such that the resulting contextual information from the contracting path is the output of the activation function (LeakyReLu) and then it is concatenated with the output of the corresponding ReLU. Using this architecture the output of the net is of size $256 \times 256 \times 2$ (two channels for the predicted ab).

Both the down-convolution and the up-convolution is applied with kernel size $k = 4$, stride $s = 2$ and padding $p = 1$.

IV.II.I **Discriminator**

As introduced in IV.I.I, it is known that the L1 and L2 losses are able only to model accurately low frequencies. In order to model high-frequencies, it is sufficient to restrict the attention to the structure in local image patches. Therefore, in [1], *patchGAN* is proposed: a discriminator that only model high-frequency structure, relying on the L1 term to force low-frequency correctness. This model evaluates every patch of $N \times N$ pixels of the input image and for each of them decides whether it is fake or not, separately. This discriminator is runned convolutationally across the image, averaging all responses to provide the ultimate output of D.

The implementation of the discriminator is composed by stacking blocks of Conv-BatchNorm-LeackyReLU. The convolution is applied with a kernel of size $k = 4$, a stride of $s = 2$ (for the last two conv layers is set to 1) and a padding $p = 1$. In Figure 7 a scheme representing the architecture of D is presented. The input of the discriminator is a $256 \times 256 \times 3$ image (in Lab format) and the patch size is $N = 70$. The resulting output is a 30×30 patch.

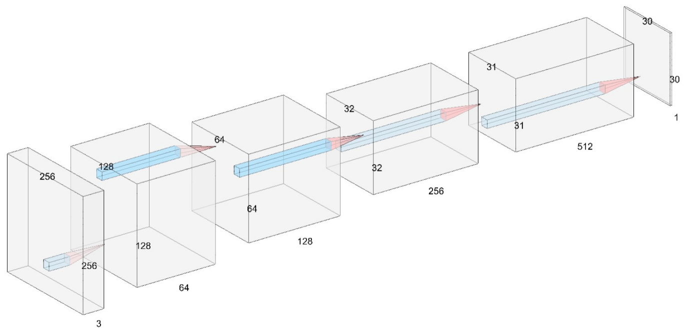

Fig. 7: Discriminator architecture

For the WGAN training, the critic D has the same architecture used by the discriminator in the cGAN training, but instance normalization is applied instead of batch normalization. This is done by following what has been suggested in [4] (they suggest to use layer normalization).

IV.III **Optimization and Hyperparamters**

To optimize the networks, we follow the approach implemented in [1] by adopting Adam as optimizer. For the cGAN training the parameters for the optimizers are $\beta_1 = 0.5$ and $\beta_2 = 0.999$ for both the generator and the discriminator. For the WGAN training the optimizer for the critic has $\beta_1 = 0$ and $\beta_2 = 0.9$, namely RMSProp with initial bias correction as suggested in [4].

During the cGAN training the batch size, denoted with m, has been set either to 8 or 32. In fact in [1] they suggest to adopt m between 1 and 10, but since we are using smaller datasets we have decided to change it and analyze the role of this hyper-paramter in this context. For the L1 loss we imposed a constant value of $\lambda = 100$. This is done to make the L1 loss and the GAN loss (the one for the adversarial training) comparable in terms of importance. The networks have been trained for 100 epochs.

The hyper-parameters for the WGAN training are set to the ones suggested in [2] and [4]. In particular n_c, the number of iterations of the critic per generator iteration, has been set to 5 and ξ, for the GP, has been set to 10.

Since many hyper-parameters are involved in the WGAN training, choosing properly λ has been a difficult task. For this reason we have decided to train the generator for 60 epochs only considering the L1 loss, namely the adversarial training hasn't been taken into account. Then the resulting weights have been used to initialize the generator for completing the training with the loss introduced in equation (8) with $\lambda = 1$. This second phase of the training has last 60 epochs.

IV.IV **Evaluation metrics**

As well-known the aim of conditional generative models is to learn explicitly or implicitly $p(\cdot|\boldsymbol{x})$ where in our case $\boldsymbol{x}$ is the gray-scale image. Ideally the optimal generator $G_{\boldsymbol{\theta}^*}$ is such that $p_{G_{\boldsymbol{\theta}^*}}(\cdot|\boldsymbol{x}) \approx p(\cdot|\boldsymbol{x})$. This means that if a CNN is trained for solving a classification problem by use of a dataset $\mathcal{D}$ (collection of both images and corresponding labels), the resulting performances should be the same either if we use real images or images recolored by our generator is.

From this intuition, the first metric has been designed. We have trained an image classifier, denoted with C_c, for solving a multi-class classification problem which involves twelve animal categories. C_c is a VGG16 pretrained on *ImageNet* and then fine-tuned on $\mathcal{D}_{\text{tr-animals}}, \mathcal{T}_{\text{tr-animals}}$, namely the last full-connected layer is a linear layer with 12 outputs units. Once C_c has been tested on the test dataset $\mathcal{D}_{\text{test}}$, $\mathcal{T}_{\text{test}}$ by computing the resulting accuracy, if $G^*_{\boldsymbol{\theta}}$ is such that $p_{G_{\boldsymbol{\theta}^*}}(\cdot|\boldsymbol{x}) \approx p(\cdot|\boldsymbol{x})$, then we expect that pretty much the same accuracy should be obtained when the recolored images provided by the generator are used instead of the real ones. For this reason, the closer the accuracy obtained by using fake images (the ones recolored by the generator) is to the accuracy obtained by using the real ones, the better the generator.

This kind of approach can also been applied to another classification problem like a segmentation one, as the FCN-score described in [1] (not used for evaluating the recoloring model). However neither the FCN-score neither the accuracy-based metric proposed before weight the images in an appropriate way. In particular, considering the FCN-score we would like to obtain a weighted mean of the pixel-accuracy instead of a classical arithmetic mean. To do so we have to compute some weights such that if for a given image the color plays an important role, then the pixel-accuracy obtained for that image should be weighted more.

The same can be done dealing with an image classification problem. C_c, the same used for the accuracy based metric, is taken into account for this scope. In particular the weight for a given image, denoted with index i, is computed as follows:

$$w_i = |\hat{p}_i^{\text{color}} - \hat{p}_i^{\text{gray}}| \tag{9}$$

where $\hat{p}_i^{\text{color}}$ denotes the predicted probability that the color image comes from the right class, while $\hat{p}_i^{\text{gray}}$ denotes the predicted probability that the gray-scale version of the image comes from the right class. Both the two predicted probabilities are outputs of C_c and for this reason the gray-scale image is still a 3-channels tensor (we just replicate along the depth dimension the single-channel gray-scale image). The role of the weight is to describe the importance of color for classifying that image. Ideally the absolute value is not needed, but in practice it is since in some cases in which the color is not necessary to classify an image, namely the two probabilities should be the same, it might result that $\hat{p}^{\text{gray}} > \hat{p}^{\text{color}}$. Finally the score for a given generator G is given by

$$\text{CIM} = \frac{\sum_{i=1}^{M} w_i \, \hat{p}_i^{\text{fake}}}{\sum_{i=1}^{M} w_i} \tag{10}$$

where $\hat{p}_i^{\text{fake}}$ denotes the predicted probability (by C_c) for the recolored image i and $M = |\mathcal{D}_{\text{test}}|$. This score allows to compare more fairly two given generators, and in particular it allows to observe if the recoloring pushes the probabilities to higher values for those images that are difficult to classify without color.

The third and last metric has been derived by the following concept. A way to introduce regularization in a deep learning model is using the so called parameter tying (as cited in [6], [7]). Suppose to have model A with parameters $\boldsymbol{w}^{(A)}$ and model B with parameters $\boldsymbol{w}^{(B)}$. Let us imagine to have two similar tasks, with similar input and output distributions of the data. The model parameters, after the training phase, should be close to each other: $\forall\, i,\ \boldsymbol{w}_i^{(A)} \approx \boldsymbol{w}_i^{(B)}$. This information is used through regularization imposing a norm penalty of the form

$$\Omega(\boldsymbol{w}^{(A)}, \boldsymbol{w}^{(B)}) = ||\boldsymbol{w}^{(A)} - \boldsymbol{w}^{(B)}||_2^2 \tag{11}$$

Here we used an L2 penalty but other choice are possible. Another way to regularize the parameters is to use some constraints, to force sets of parameters to be equal. This method of regularization is often referred to as parameter sharing.

We decide to design a comparing method using this idea. Suppose to have two different generators, namely G_1 and G_2. If we consider a dataset $\mathcal{D}_{\text{test-2}}$ and the corresponding labels $\mathcal{T}_{\text{test-2}}$, we can process these images with the G_1 and G_2 obtaining respectively $\mathcal{D}^{(1)}_{\text{test-2}}$ and $\mathcal{D}^{(2)}_{\text{test-2}}$. We can compare these two synthetic dataset to the original ones, using three identical classifiers: C^*, C_1, C_2. Model C^* is the ideal one, trained with the original images, namely $\mathcal{D}_{\text{test-2}}$. C_1 and C_2 are trained using the images processed by G_1 and G_2 respectively, namely $\mathcal{D}^{(1)}_{\text{test-2}}$ and $\mathcal{D}^{(2)}_{\text{test-2}}$. The weights $\boldsymbol{w}^{(*)}$ of C^* will be close to the weights $\boldsymbol{w}^{(i)}$ of C_i with $i = 1, 2$, only if the two models are trained using the same data distribution. Therefore we evaluate $\Omega(\boldsymbol{w}^{(*)}, \boldsymbol{w}^{(1)})$ and $\Omega(\boldsymbol{w}^{(*)}, \boldsymbol{w}^{(2)})$ using the parameter tying formula (11), to get a score of how well the distribution of data is created by the generator with respect the original one. The smaller is the norm of Ω, the better the performance of the generator.

During our tests we use as C^*, C_1, C_2 VGG16 architectures, and we train them respectively with $\mathcal{D}_{\text{test-2}}$, $\mathcal{D}^{(1)}_{\text{test-2}}$ and $\mathcal{D}^{(2)}_{\text{test-2}}$ with batch size 128 for 2 epochs each. A quantitative scheme is shown on Fig. 4

V. Results

Observing the values obtained with the CIM metric, reported in Table 1, we can clearly see that the colorized images have lead to an improvement in the results. In fact, the CIM obtained by providing as input the grays-scale images (3-channels) to C_c is lower than the ones obtained with the synthetic images produced by the generators (in Table 1, with "gray-scale" we mean that the images used are gray-scale images while with "real" we mean that the original images in $\mathcal{D}_{\text{test}}$ are used). This means that they actually embed more information and the colorization has been implemented in a meaningful way.

Model	m	N	**Accuracy**	**CIM**	$\boldsymbol{\Omega}$
WGAN	32	$9.6K$	0.9146	0.5359	4.4197
cGAN	32	$16K$	0.9079	0.5613	4.4747
cGAN	32	$9.6K$	$\underline{0.9263}$	$\underline{0.5764}$	4.4170
cGAN	8	$9.6K$	0.9221	0.5761	$\underline{4.4159}$
real			0.9658	0.7396	
grayscale				0.4613	

TABLE 1: Parameters

Moreover we can analyze the effects of changing the batch size from $m = 8$ to $m = 32$, by using the small dataset $\mathcal{D}_{\text{GAN\_s}}$. As shown in Table 1, the results in terms of accuracy, CIM and $\boldsymbol{\Omega}$ are pretty much the same. This means that when a small dataset is used then it seems that the batch size doesn't play an important role in improving the performance of the resulting generator.

A surprising outcome is that by increasing the number of images N, comparable results in terms of CIM have been obtained (with respect to the accuracy, the generator trained with $N = 16K$ is even worse than the one obtained with the smaller dataset, in particular if we consider the $\boldsymbol{\Omega}$ results). A possible reason is that $m = 32$ is too large when a bigger dataset is employed and probably this is the reason why a smaller batch size dimension is suggested in [1]. From these observations, it seems that if the training dataset is increased then the batch size should be decreased. This is in particular relevant for GANs training since we would like to avoid a situation in which the discriminator is too hard to fool, and from the last observations it result that a combination of big batch size and big dataset might bring to this kind of situation.

This is confirmed if we analyze the G_GAN loss in Figure 8 and 9 (G_GAN loss is the part of the generator loss without the L1 contribution). We see that after some epochs it start increasing again. That might be due by the fact that the discriminator is getting to good in its job hence the generator won't have any more room of improvements. The worst result is obtained by the cGAN with $\mathcal{D}_{GAN\_b}$, where the G_GAN loss keeps increasing. Instead, for the cGAN with

$\mathcal{D}_{GAN\_s}$, even though the oscillatory behaviour, we don't have a continuously increasing loss.

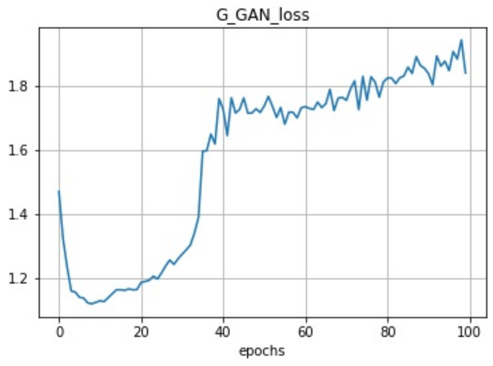

Fig. 8: G_GAN loss of the cGAN with $\mathcal{D}_{GAN\_b}$

The oscillatory behaviour might be caused by a not general and rich enough dataset. But due to the computationally expensiveness of the training, we weren't able to do many tests with bigger datasets or different batch sizes. Having a bigger datasets, as done in [1], could really improve this because we would have much more data to train both the generator and discriminator on, without having to worry that given the small number of pictures, the discriminator becomes too powerful.

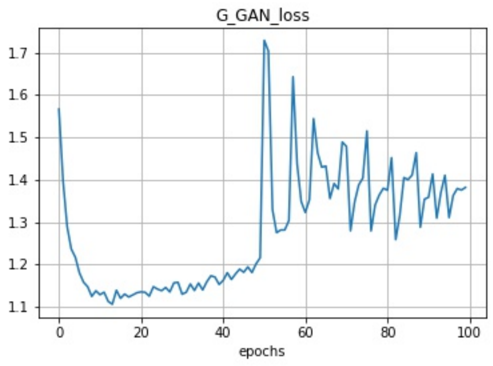

Fig. 9: G_GAN loss of the cGAN with $\mathcal{D}_{GAN\_s}$

Overall by looking at the figures 10 and 11, we can see that the whole generator loss is decreasing. That is a desired result as explained in the formulation of the min-max game of the value function $V(G, D)$.

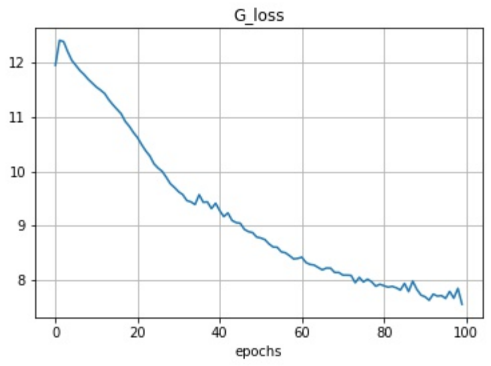

Fig. 10: G loss of the cGAN with $\mathcal{D}_{GAN\_b}$

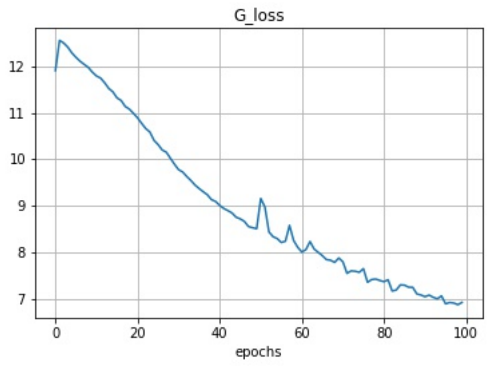

Fig. 11: G loss of the cGAN with $\mathcal{D}_{GAN\_s}$

VI. Concluding Remarks

As shown in the result section, our experiments pointed out how the batch size and the dataset dimension are really important in the adversarial training. For this reason, a more accurate and wide study by use of different dataset dimensions (higher than the ones adopted in this work) should be done. Also with the additions of other metrics like the FID and the IS. Another possible metric can be derived by the Catastrophic forgetting problem, that usually reveals when a model is trained two times for solving different tasks, as explained in [8]. In particular, a possible way for evaluating a generator might be derived as follows: given an image classification task, a classifier can be trained on a real dataset $\mathcal{D}$ and then re-trained on $\mathcal{D}_{\text{fake}}$ generated by the generator. If the generator is capable of learning the distribution of $\mathcal{D}$, then catastrophic forgetting measured by means of the accuracy might either not arise or be present in a negligible way.

As pointed out in the WGAN training, we encountered problems in tuning all the hyper-parameters for the optimization and for the losses. We have partially solved this problem by splitting the training in two phases, but we have not had the opportunity to choose the right numbers of epochs for the two phases. Therefore a more in-depth research is needed.

VII. **Individual contributions and problems encountered**

- **Riccardo De Monte** : WGAN, CIM metric and contributions for cGAN
- **Mattia Secchiero** : cGAN & WGAN contribution
- **Simone Cecchinato** : Evaluation metrics, Classifiers training, contributions for WGAN

The different trials for choosing the hyperparameters and for training the different models have been done by all the three members of the group.
In particular, due to the limited resources of Google Colab, we were not able to make all the trials desired. As a result we couldn't find the proper setup for the WGAN training.

References

[1] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," Nov. 2016.

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

[3] M. Mirza and S. Osindero, "Conditional Generative Adversarial Nets," Nov. 2014.

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

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

[6] I. J. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," MIT, 2016.

[7] J. Lasserre, C. Bishop, and T. Minka, "Principled hybrids of generative and discriminative models," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 1, pp. 87–94, 2006.

[8] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio, "An empirical investigation of catastrophic forgetting in gradient-based neural networks," 2013.