# Image Recoloring with Conditional Adversarial Network

Lorenzo Borella, 2012266, Unipd

***Abstract*—The possibility of learning specific features of images, as well as generating completely new samples is nowadays becoming a skill of major importance for many scientific and technological fields. The problem of Image Recoloring has been so far approached by Autoencoder Neural Networks, but a more general and reliable method must be searched. In this paper I explore the usage of Conditional Generative Adversarial Networks to tackle image translation problems. I study the characteristics of this learning technique by playing with its main structures and exploring its hyperparameters. Three different UNET architectures are used to generate samples of colored images, while the patchGAN approach is studied for the discriminating section of the DCGAN. Surprising results are reported regarding the behaviours of the cited architectures with respect to the usage of Adam and SGD optimizers. Pillar concepts are built in order to define the best fitting architecture for the Image Recoloring problem, eventually delivering satisfying experimental results.**

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

## I. Introduction

Image-to-image translation's field of applicability is extremely wide, as well as the number of techniques to tackle different aspects of this same issue. Due to the variety of the single case problems, they have been so far resolved by specific and dedicated Neural Network structures and learning algorithms. Nevertheless, the majority of these challenges can be formally reduced to the same concept of Image Translation, therefore it must be possible to solve this heterogeneity of issues with one single approach. The necessity to build such learning framework has reached its climax.

Conditional Generative Adversarial Networks (shortly **CGANs**) have recently shown to be a good example of Neural Network architectures that can tackle a great variety of problems in image translation and also reach satisfying results. The adversarial nature of this Neural Networks guarantees a constant and long lasting learning; the engine of such mechanism relies in the corresponding machine-level concept of healthy competition.

Image Translation problems often reduce to learning a proper pixel-to-pixel mapping function that can be used to highlight or select specific features of some images, as well as changing them completely or using them as a seed to generate something completely new. The **CGAN** architecture can be applied in several of the above cases. In this paper we will present its usage and applicability when dealing with the Image recoloring problem, namely the necessity to reconstruct a colored image starting from its black and white version.

## II. Related Work

Conditional Adversarial Networks have already been used for image translation problems. Day to Night image transformation, as well as selecting the edges of photos or labelling the objects in a street-view picture are all problems that have been approached and tested with Conditional Generative Adversarial Networks. The Image Recoloring has been of course one of them, therefore I exploit what the state of the art studies have previously derived.

By making the Generative Adversarial Network Conditional, consistent gains are present in the quality of the generated images and in the learning process of the Discriminator. Without this Conditional feature the Discriminator would only have to classify fake vs real images by confronting them one to another. In this way instead, a ground-truth information is given in input to the Discriminator, which can more easily detect the distances between original and generated images. Moreover, by inserting an $L_1$ regularization term in the loss for the Generator, we guarantee that the euclidean $L_1$ distance between the generated and original colored images is not increasing too much. This procedure also allows to lower the blurring effect on the generated images and to keep the edges well defined; an $L_2$ term would not guarantee this specific feature [1].

The usage of a U-Net architecture has been suggested by the previously derived results in Image Segmentation. In some medical fields it might be of paramount importance to correctly draw the edges of objects in biological imaging pictures. This can be tackled by the specific U-Net architecture. Its peculiarity is that it is able to store the processed information in a Convolutional Neural Network and propagate it directly by skipping connections with in-between layers. At the same time, it correctly compresses the information into a feature space and expands it back to generate a new sample, exactely as an Autoencoder would do. The only difference is that the skip connection allow a more homogeneous image translation, avoiding the bottleneck problem typical of Autoencoders [2]. This approach has brought consistently positive results and will be used in my study to solve the Image Recoloring problem.

## III. Processing Pipeline

The defining characteristic of a Generative Adversarial Network is to have two different NN structures that work as rivals in a zero-sum game. In my case, the role of the Generator is to

produce the most plausible colored images starting from BW inputs, while the Discriminator's aim is to correctly detect differences in the original and synthesized colored images, therefore evaluating the performances of the Generator. The learning process and the generating efficiency rely on such adversarial behaviour: the Generator must return increasingly believable images in order to fool the Discriminator, while the latter must improve its classification ability in order not to be fooled. To solve the image recoloring problem in this paper the **UNET** architecture is used as a Generator, while the Discriminator exploits the frame-by-frame approach of the **patchGAN** architecture.

### A. *Generator*

The generator's aim is to produce believable images, starting from BW input data. It performs such action exploiting an U-shaped Network, in which skip connections lay in between convolutional and downsampling layers. This feature happens to be particularly useful when transferring information and mapping input pixels to output pixels. In past experiments, the naive approach was to exploit an Encode-Decoder generative model, but it was soon realized that the bottleneck of information in such architecture might be too strict to properly transfer the right amount of data when building the input-output mapping. The **UNET** method solves the problem of a too small size of the feature space by introducing the above cited skip connections, hence explained the so called U-shaped Network. Just as the Encoder-Decoder network, the **UNET** has two different sections, in which the information is firstly reduced and then expanded. The skip connections consist of cropping the output of specific layers in the shrinking section and propagating them to the corresponding layers in the expanding section; the channels are then concatenated and upsampled. In this way, we are able to encode the input information into the deepest layers of the **UNET** while maintaining some of the original features of the input images.

### B. *Discriminator*

The discriminating section of the **patchGAN** has the role of understanding if the input images received are coming from the real dataset or if they are instead produced by the Generator. In order to do so, each picture is analyzed and processed in a patch-per-patch manner. The 2D data are divided into smaller frames of specified size, each frame is then processed by a Convolutional Neural Network that eventually outputs a value in the range $[0, 1]$, respectively representing the evaluation of fake and real images. When all the frames have been processed, a final mean is computed on the output values of each patch, producing a response on the origin of the complete image. Since the architectures tested in this paper are built in a conditional manner, the **patchGAN** is receiving the same BW images that the Generator receives, together with their respective colored layers coming from the original pictures and from the Generator's products. The Discriminator is therefore called to process 3-channeled images (2 colored + 1 BW) and correctly classify them into Fake and Real pairs.

## IV. Signals and Features

### A. *ImageNet1000 Dataset*

The analysis is performed exploiting a subset of the `ImageNet1000` dataset, containing around $20.000$ samples between Training and Validation images. The pictures represent a large variety of subjects, *e.g. animals, musical instruments, cars etc.*, corresponding to a total number of $1000$ different classes. Horizontal, vertical and squared images are stored in such dataset, representing a huge variety in the 2D dimensions of each data sample; in order to standardize the input data, the images are reduced to a fixed squared size (200x200), leading to consistent modifications in the original quality of some of them.

The original images in the dataset are colored, therefore they present 3 RGB channels per 2D picture. In order to obtain greyscale images as starting input, the data are moved to the **L*a*b*** representation space (see Fig. 1), in which the Luminosity layer **L*** can assume the role of 1-channel BW image. The **a*** and **b*** represent the colored pixels respectively transitioning from red to green and from blue to yellow. While the luminosity values are usually reported in the range [0,1], the colored pixels have no range limitations and therefore are cut in the [-128,127] range for computational needs. The generated images in the code need to be rescaled in the proper **L*a*b*** ranges and eventually moved into the RGB space for representation purposes; the latter is performed via the `color.lab2rgb` transformation. Processing images exploiting the **L*a*b*** space results to be particularly useful from the Generator's learning point of view. Instead of receiving a single channel BW image in input and trying to learn 3 different RGB channels to generate the colored image, in this case we are just feeding the UNET with the luminosity layer **L*** and asking him to learn the mapping from it to the colored layers **a*** and **b***, reducing consistently the computational complexity of the Image Recoloring problem.

Due to the specific structure of such a framework, processing data through the **UNET** results in cropping of the 2D dimensions of the input images. The initial quality of the pictures is therefore lowered when propagated through the Generator section, as a price to pay in order for it to properly learn the necessary features to generate believable images. The size of the processed images changes accordingly with the structure of the network, therefore a dedicated definition of the **patchGAN** happens to be necessary. Before feeding the **patchGAN** with colored and greyscale images, the original figures must be cropped to the same size as the generated pictures, as when pairing the **UNET** outputs with original BW images, they must be of the same size. If this was not the case, the software would automatically fill the missing dimensions

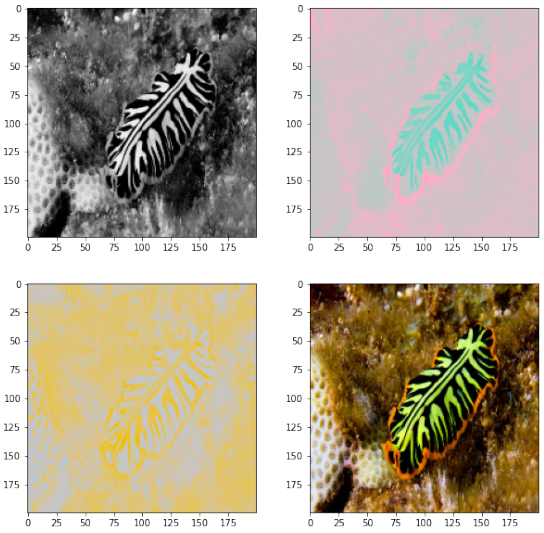


Fig. 1: Image in **L*a*b*** space.

with 0 values, creating a black frame that would prevent the correct Discriminator's learning.

### *B. Preprocessing*

The data are uploaded on Google Drive, where the Google Colab notebook can easily reach them. The samples are divided into Training, Validation and Test datasets, exploiting the `ImageFolder` method from `torchvision.datasets` as reported in the following table.

| Set | Percentage | Samples |
|---|---|---|
| Total | 100% | 20426 |
| Training | $\sim 80\%$ | 16340 |
| Validation | $\sim 10\%$ | 2042 |
| Test | $\sim 10\%$ | 2044 |

TABLE 1: Initial Data Splitting.

Since not all the images present the same pixel distributions, a study of the "colorization" of each image is performed, in order to understand which samples are more or less colored. This analysis is performed by loading the images as **RGB** tensors and by defining a custom function `image_colorfulness` which is able to compute a specific metric that reflects the amount of colors in an image. By considering separately each colored layer R, G and B, the function computes the matrices $rg = |R - G|$ and $yb = |\frac{R+G}{2} - B|$. The mean and standard deviation of such matrices are evaluated and combined as a sum of squares and the final value for the colorfulness of the image is computed according to **colorfulness** $= 0.3 * \mu + \sigma$. Eventually the colorization of all the images in the datasets are computed and their distributions plotted (Fig.2), in order to inspect which dataset contained the most colorful images.

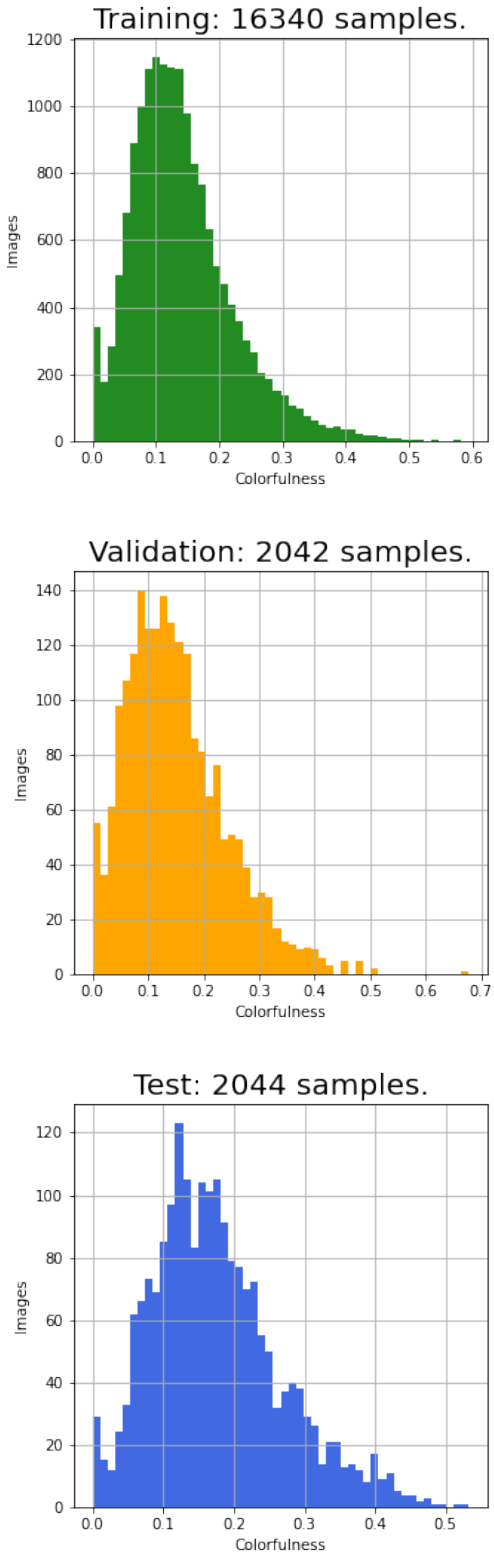



Fig. 2: Colorization distributions before cut.

Ideally we would want to have a Training Set richer in colors than the Validation and Test sets, since we want the model to study more complex samples and learn a variety of colors and objects before testing it on brand new pictures. With this idea in mind, we tried to select the most colorful samples in the Training set, by applying a cut on the colorization values (Fig.3). With a final selected cut of 0.08, the rejected samples are redistributed equally on the Validation and Test sets. The resulting splitting is reported in Tab.2 (the final distributions are reported in the Appendix Fig.16).

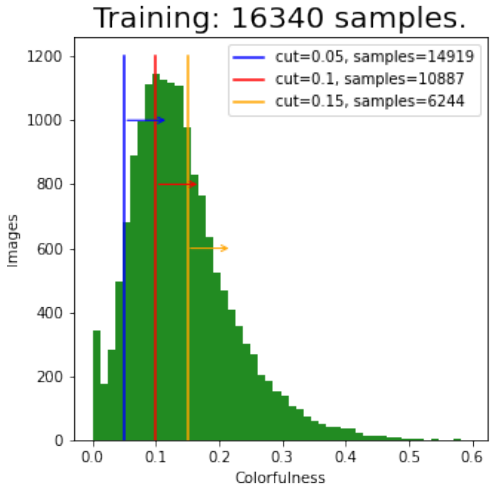



Fig. 3: Possible cuts in Training Set.

| Set | Percentage | Samples |
|---|---|---|
| Total | 100% | 20426 |
| Training | $\sim$ 60% | 12739 |
| Validation | $\sim$ 20% | 3842 |
| Test | $\sim$ 20% | 3845 |

TABLE 2: Final Data Splitting.

`Dataloader` objects are then created for the three datasets, with 12 workers each. The batch size is a fundamental parameter for tuning learning accuracy and computational time, since each batch will be sent to the GPU and processed by the **DCGAN**. Its choice depends on the type of architecture in use: a larger `batch_size` can be used when training a simpler model but a smaller `batch_size` is needed when the number of trainable parameters become quite large, in order not to put too much weight on the GPU RAM. The final choosen values are reported in Tab. 3.

| Dataset | Batch Size | Batches |
|---|---|---|
| Training | 128 | 100 |
| Validation | 128 | 31 |
| Test | 128 | 31 |

TABLE 3: Batch Division

## V. Learning Framework

In the following pages, a series of comparisons are made in order to properly understand the dependencies of the learning behaviour on the structure of the **DCGAN**, as well as on its main hyperparameters. Three different architectures are studied, which I will refer to as *Simple*, *Normal* and *Complex*, depending on the total number of their trainable parameters. SGD and Adam optimizers are used in order to inspect possible changes in the behaviour of the networks. Considering the best combination of the above cited modifications, the model is eventually trained and tested, reporting consistently satisfying results.

### A. *UNET*

In the following tables the detailed structures of the two *Simple* and *Normal* **UNET** architectures are reported. The colored lines in the tables represent the layers in which the skip connections are performed: the outputs of a Conv2d layer in the shrinking section are convoluted to those of a ConvTranspose2d in the enlarging section. In between Conv2d layers, BatchNorm2d and LeakyReLU sections are present, even though not reported in the tables for simplicity. The following parameters remain fixed for all the blocks: `kernel_size` = 3,`stride=1` and `padding` = 0 and noise is inserted in the architectures by applying dropout with probability $p = 0.2$ at each block. The output of the final Conv2d section represents the colored **a*b*** layers.

| Level | Block | In-Out Channels |
|---|---|---|
| 1 | Conv2d | (1,64) |
| 1 | Conv2d | (64, 64) |
| 1 | Conv2d | (64, 64) |
| 2 | Conv2d | (64,128) |
| 2 | Conv2d | (128,128) |
| 2 | Conv2d | (128,128) |
| 1 | ConvTranspose2d | (128,64) |
| 1 | Conv2d | (128,64) |
| 1 | Conv2d | (64, 32) |
| 1 | Conv2d | (32, 2) |
| | **Total Parameters** | 570.825 |

TABLE 4: Simple UNET.

Without training the models but just propagating the images through **UNET** Generator (with weights properly initialized), we can already witness a strong difference in the output images. Since the models have not been trained yet, the colouring is of course not matching the original samples for both models, but it's possible to already visually appreciate quality of the generated images; while the *Normal* model still correctly shows edges and shapes, the *Simple* one fails to do so, creating almost unrecognizable figures (see Fig. 4).
The structure of the *Complex* architecture is reported in the Appendix. The main difference with the previous ones is that it reaches a depth of 8 levels in shrinking the images information and that it analyzes images of size (256,256). The results

| Level | Block | In-Out Channels |
|---|---|---|
| 1 | Conv2d | (1,64) |
| 1 | Conv2d | (64,64) |
| 2 | Conv2d | (64,128) |
| 2 | Conv2d | (128, 128) |
| 3 | Conv2d | (128,256) |
| 3 | Conv2d | (256,256) |
| 3 | Conv2d | (256,256) |
| 2 | ConvTranspose2d | (256,128) |
| 2 | Conv2d | (256,128) |
| 2 | Conv2d | (128, 64) |
| 1 | ConvTranspose2d | (64, 64) |
| 1 | Conv2d | (128,64) |
| 1 | Conv2d | (64, 2) |
| | **Total Parameters** | 2.327.558 |

TABLE 5: Normal UNET.

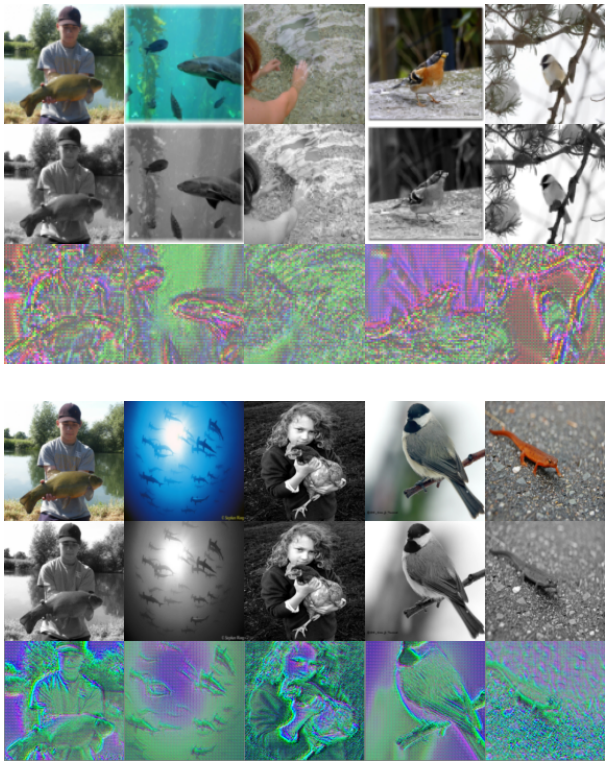


Fig. 4: *Simple* (top) and *Normal* (bottom) image propagation.

delivered by such extremely complex architecture are of much higher quality than those reported by the previously cited generators, since the NN is capable of learning more complicated patterns, objects and colors. Having more trainable parameters and connections corresponds to a smarter Neural Network. Even in this case, BatchNorm2d and Dropout blocks (with probability $p = 0.5$) are inserted in between Conv2d layers; the following parameters are kept fixed `kernel_size` = 4,`stride=2` and `padding` = 1.

### B. *patchGAN*

To inspect the efficiency of the Discriminator, we wanted to explore its dependence on the number of total trainable parameters. Therefore, even in this case, three *Simple*, *Normal* and *Complex* architectures have been defined; they will eventually be used paired with the **UNET** corresponding architectures and not mixing *Simple* and *Complex* examples.

Due to the conditional characteristic of the **CGAN**, the Discriminator receives in input a 3-channel image, corresponding to the **L*a*b*** layers of each image. The Discriminator receives a `divisor` parameter as input, which describes how to split the incoming image into its $\texttt{divisor}^2$ patches. At this point, each patch is processed through a CNN. The *Simple* Discriminator architecture has 3 convolutional layer and 1 final linear layer, while the *Normal* one has 5 convolutional layers and 1 final linear layer; the *Complex* architecture is reported in the table below. All the above networks eventually output one single value in the range $[0, 1]$ that should reflect the veridicity of the images.

| Level | Block | In-Out Channels |
|---|---|---|
| 1 | Conv2d | (3,64) |
| 1 | Conv2d | (64, 128) |
| 1 | Conv2d | (128, 256) |
| 1 | Conv2d | (256,512) |
| 1 | Conv2d | (512,1) |
| | **Total Parameters** | 2.765.633 |

TABLE 6: Complex PatchGAN.

### C. *Learning details*

From the **DCGAN** theory we know that the objective function can be expressed as:

$$\mathcal{L}_{cGAN}(G, D) = logD(x, y) + log(1 - D(x, G(x)))$$

with G that tries to minimize such function while D tries to maximize it. Most of the time unfortunately, one has to deal with the problem of vanishing gradients, which prevents the model to learn properly. A possible improvement to prevent this from happening is to redefine the objective function above: instead of asking the Generator to minimize the probability that the Discriminator is correct ($\log 1 - D(G(x))$), we should ask him to maximize the probability for it to be wrong ($-\log D(G(x))$). The learning process in our code has followed this principle.
In general, the Discriminator's aim is to perfectly recognize when an image is real and when it is fake, while the Generator's objective is to fool the discriminator by producing a believable image; its loss is directly related to the Discriminator's ability to detect a fake image. Additionally, an $L_1$ regularization term is added to guarantee the Euclidean distance between y and G(x) not to increase too much. The learning rate multiplied to the $L_1$ term to weight on the total Generator's loss is kept fixed at $\lambda = 0.005$

for all the runs. All architectures are trained with the same procedure. Firstly, the Discriminator processes the real and fake pairs, its loss is computed and its weights updated. At the end of the procedure, the Generator produces a fake image and its loss is computed with respect to the previously updated weights of the Discriminator. As it is required in a zero-sum game, the gain of one player is the loss of the other, therefore the Generator's loss is computed considering the produced images as real ones and feeding them to the Discriminator. I used the `nn.BCEWithLogitsLoss()` criterion for the evaluation of D and G losses. As a form of Binary Cross Entropy, it naturally measures distances between distributions, in this case for values in the range $[0, 1]$. I also tested the `nn.BCELoss()` option, but in the majority of cases it resulted into a divergent learning process, causing the two sections of the **CGAN** not to learn at the same pace, eventually saturating G's loss and producing a perfect Discriminator.

A variation on the traditional training method is also explored. The **WGAN** present some difference with respect to those cited above; first of all, the discriminator is built to return values in the range [-1,1] instead of [0,1], where 1 stands for "fake image" and -1 for "real image". Moreover, the WGAN discriminator is trained more than the generator; each batch is analyzed `disc_steps` times by the Discriminator and only once by the Generator. Then the **WGAN** exploits the usage of the Wasserstein loss, from which it takes the name. This function multiplies the reconstructed labels with the true labels and computes their mean, returning a value that represents the veridicity of the image. In this sense, the **WGAN** discriminator has the role of a critic rather than a classifier, producing higher results for images that are more believable to be true and lower evaluations for poor quality images. For this training process, the optimizer used is `RMSProp` and the insertion of clipped gradients it's mandatory, in order not to fall into the exploding gradients problem.

## VI. Results

### A. *Simple architecture, Adam Optimizer*

The training process for the *Simple* architecture has been performed using both Adam and SGD optimizers, in order to inspect the possible changes in the learning behaviour. The `learning_rate` $= 0.0001$ has been fixed for both optimizers. The model has been trained for `epochs` $= 30$, resulting into a run of $\sim 1h6min$ for the SGD case and $\sim 1h3min$ for the Adam case.
When training the *Simple* architecture with Adam optimizer, the learning curves report a behaviour that shows that the Generator and the Discriminator seem to have engaged into a zero-sum game. Their losses are oscillating, with values compatible with eachother: when the Generator has a higher loss, the Discriminator has a lower one, meaning that the Generator is producing low quality images and the Discriminator is easily capable of differentiating them from the original ones. When the Discriminator has instead a higher loss, the Generator has

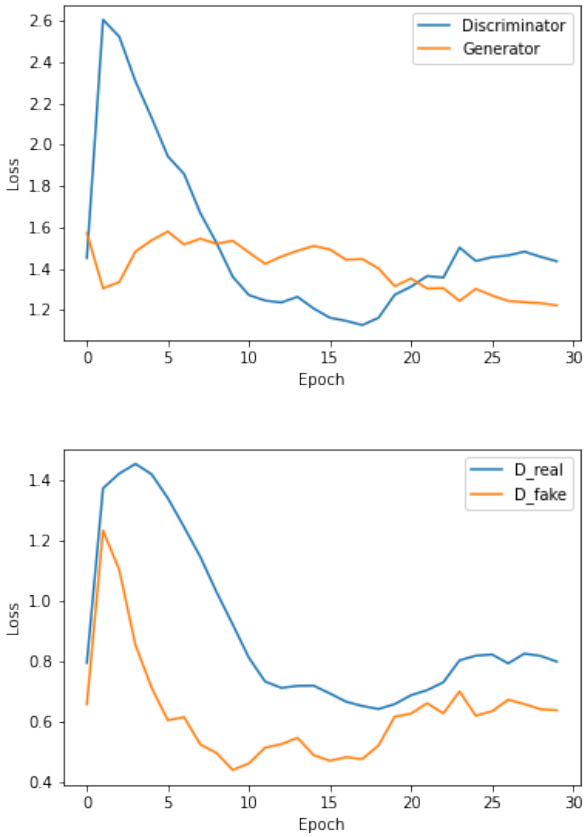



Fig. 5: *Simple* architecture learning curves.

a smaller one, meaning that either the Generator is good at fooling the Discriminator, or that the Discriminator has not learned properly from the original colored images. In the plots in Fig. 5, these tendencies seem to alternate, as if the zero-sum game in which Discriminator and Generator fight against each other was properly starting. Nevertheless these are not the expected results, as it is confirmed by the high loss value that the Discriminator makes when trying to identify real images; it means that it has not properly learned how to do that.
Even though the last values report a smaller loss for the Generator than for the Discriminator, the produced images are of extremely bad quality (Fig.6). Ideally one would like to see a constantly decreasing Generator's Loss and a constantly increasing Discriminator's loss, but that it's not the case for the *Simple* Architectures within the range of only 30 epochs.

### B. *Simple architecture, SGD Optimizer*

For the SGD optimizer case, the results are even worse than those of Adam. As we can see if Fig.7, the learning behaviour is not at all what one should hope to derive. The Discriminator and Generator's losses rapidly increase after only 5 epochs and they remain constant for the whole training. This time the zero-sum game was not engaged by the two architectures. Both of them fail to learn their respective objectives, therefore they diverge at an enormous loss value. We can see that the Discriminator is failing at identifying the real images,

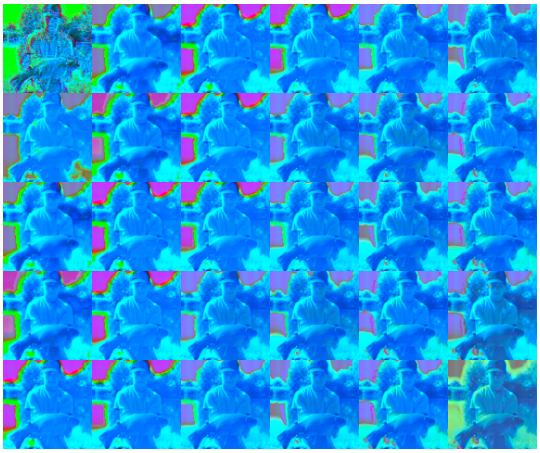


Fig. 6: Images produced by *Simple* architecture with Adam optimizer.

therefore it cannot "motivate" the generator into producing better results. It is clear that the SGD optimizer does not allow the architecture to move in the proper minimization direction, impeaching the learning process.

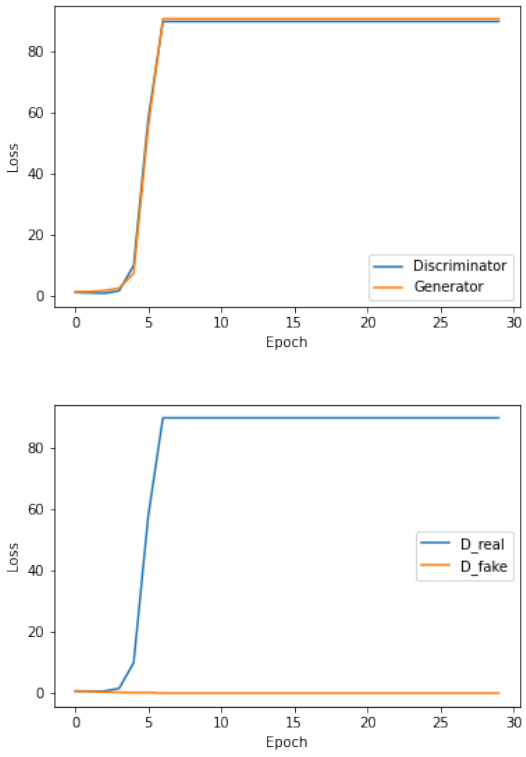



Fig. 7: Learning curves with SGD optimizer.

### C. *Normal Architecture, Adam optimizer*

For the *Normal* case, the model has been trained with a `batch_size` $= 128$ for `epochs` $= 30$ resulting into a $1h37min$ run. We chose not to train the *Normal* architecture with SGD optimizer, due to the poor results reported in the previous section.
Moving to an architecture with more trainable parameters, we are hoping to reach better results. Since the NN is smarter, it should be able to learn more complex patterns and produce higher quality images. As you can see in Fig. 9, these are slightly better than those produced by the Simple architecture: the patterns and figures are conserved and recognizable, and also the spectrum of explored shades of colors seem to have enlarged. Given these considerations, the results are still not reflecting at all what we should expect to be the recolored images: they do not look at all as the original ones.

The fact that the architecture is still not behaving as expected can be seen in Fig. 8, where we witness a constant decrease of both generator and discriminator's losses. This could be in general a good sign: the generator and the discriminator are both learning at a good pace and in the proper minimizing direction, as a zero-sum game would require. The problem is that we would expect the generator to win such argument over the discriminator, reporting respectively decreasing and increasing losses. In this case instead the game is still going on and the results produced at the end of 30 epochs are way too far from being good. Also the behaviours of the real and fake losses of the discriminator seem to be good: at each epoch the discriminator looses its ability to recognize real and fake images and it's eventually maximally confused, since the loss curves converge at the same value. It would be interesting to run the training for much more epochs than this, in order to explore if the generator would eventually win over the discriminator.

### D. *WGAN training*

Due to computational time, the WGAN has been run only for `num_epochs` $= 10$ and `disc_steps` $= 5$, resulting in a run of $1h40min$. This is due both from the usage of `RMSProp` optimizer and by the fact that the discriminator analyzes data for more iterations than the Generator, slowing down the training process. As we can see from Fig. 10, the generated images usually seem to be monochromatic: even though the model is trying to explore different regions of the color spectrum, it is still not able to produce satisfying results. The learning curves just report this exact behaviour, the discriminator is perfectly able to recognize which image is real and which one is fake; the Generator is therefore not producing good enough images. It would be interesting to test this same training method on the `Normal` and `Complex` architectures, in order to inspect what would happen with much more trainable parameters. This has not been performed in the present paper due to time limitations.

### E. *Best Model*

The Best model derived is the one that exploits the combination of both `Complex` UNET and patchGAN architectures. The model has been run for $epochs = 60$ corresponding to a runtime of $2h30min$, but the loss values reported in the plots

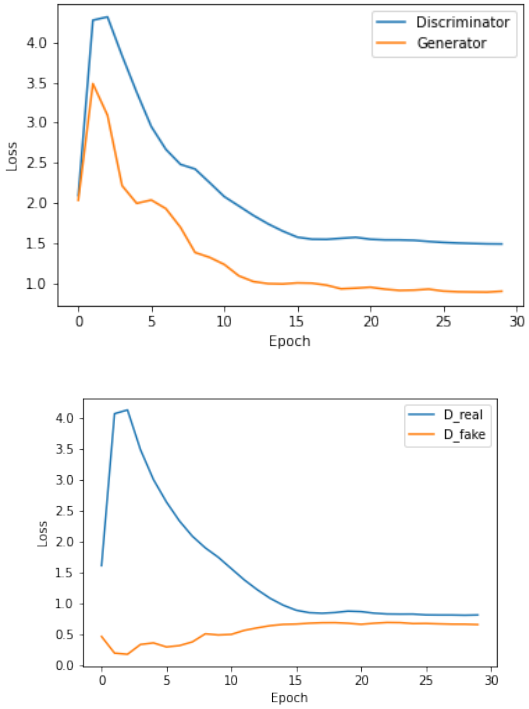



Fig. 8: Learning curves for *Normal* architecture.

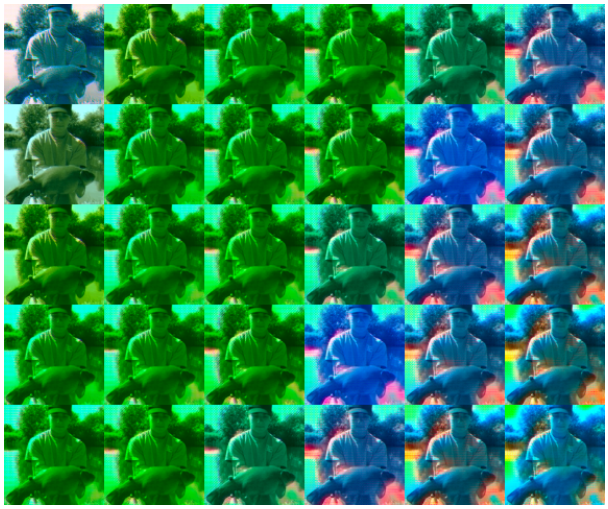


Fig. 9: Images produced by the *Normal* Architecture

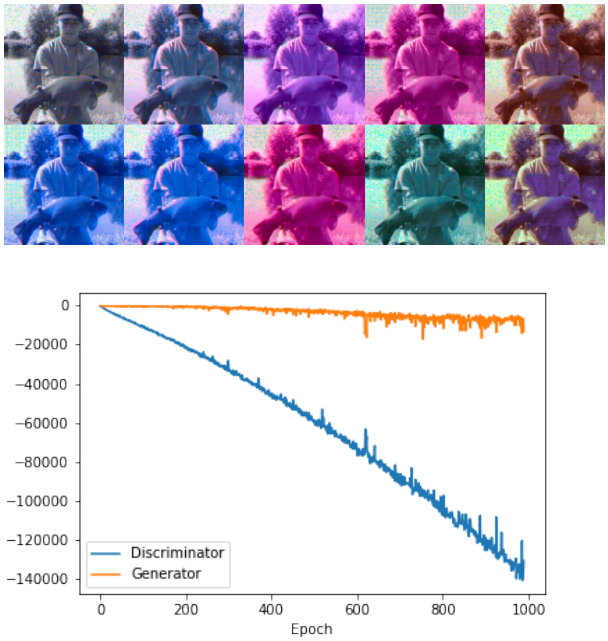



Fig. 10: WGAN Architecture: Images and learning curves.

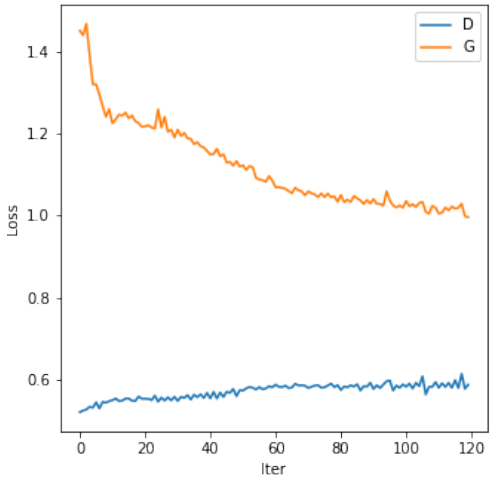



Fig. 11: Complex Architecture learning curves.

are measured twice per epoch, resulting in a total of 120 iterations. As we can see from Fig. 11, the learning curves finally behave as expected. The Generator and the Discriminator are learning at the same pace and the Generator is winning the zero-sum game. A more complex architecture has allowed us to generate increasingly more believable images; in this way, the loss of the Discriminator increases at each iteration, since it cannot distinguish which are the real and fake images, while the loss of the Generator decreases.

Analyzing in detail the behaviour of the Discriminator's loss (Fig.12), we can see how the loss coming from the classification of real images is more or less stable throughout all the iterations, meaning that the Discriminator properly learns how to recognize real images. On the other hand, the loss coming from the fake images is increasing with the iterations, meaning that it's becoming more and more difficult for the Discriminator to differentiate between real and fake images. Eventually, it is maximally confused, since the losses reach almost the same value.

We must remember that also the Generator has two different sources of errors: one coming from the Discriminator and one coming from the L1 regularization term. In order to explain the difference in the orders of magnitude of the losses in Fig. 11, I plotted the separated sources of errors for the Generator (Fig. 13). As we can see, the L1 loss term is almost 10 times

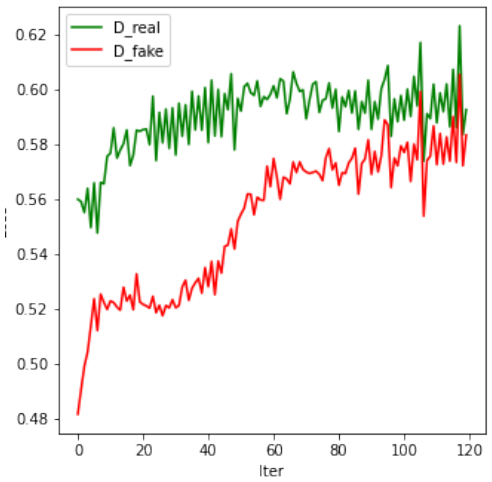



Fig. 12: Real and Fake Discriminator's losses.

higher than the one coming from the Discriminator: this of course depends on the types of images analyzed and their range of representation, as well as on the learning rate $\lambda_1$.

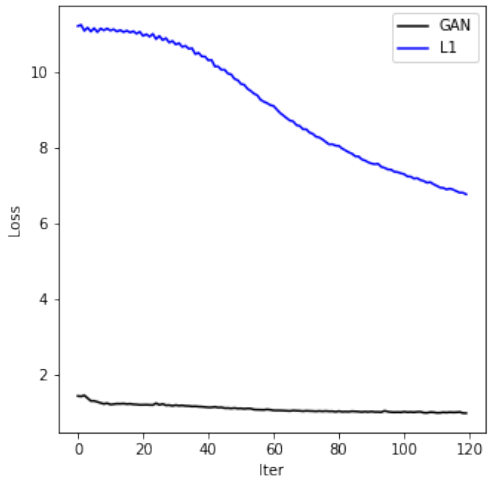



Fig. 13: Generator's losses.

In Fig.14,15 I reported the same pictures produced at the beginning and at the end of the training session. The results are finally what one would expect: the top rows are the BW images, the middle rows the generated ones and the bottom rows the original ones. In most of the cases, the objects are properly colored and are almost identical to the original images. Some mistakes are still being made by the NN, but the overall looking of the generated pictures is extremely believable.

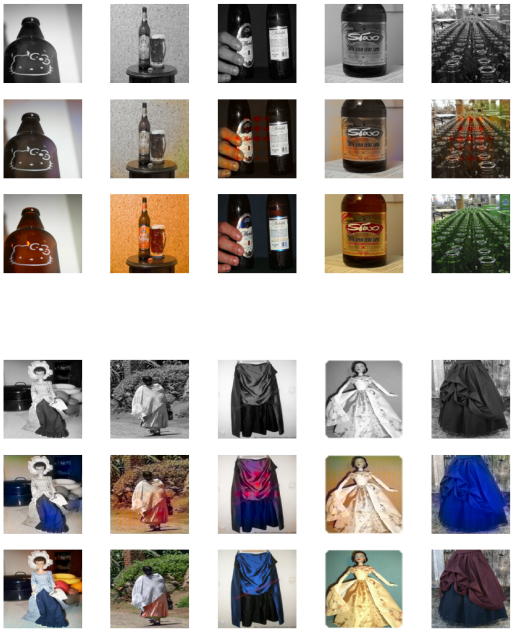


Fig. 14: Pictures at th beginning of training.

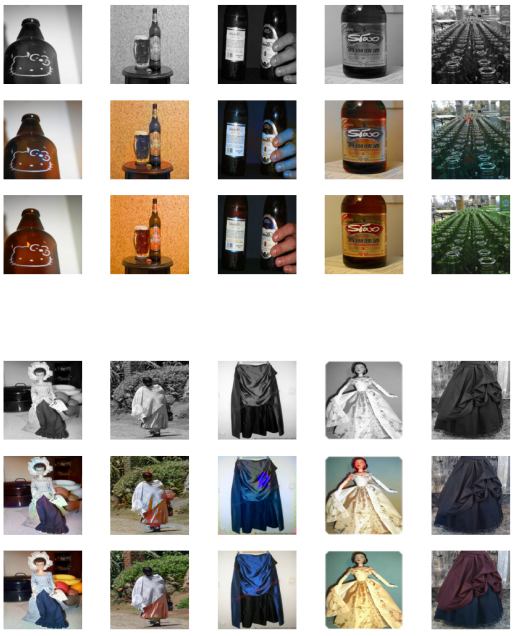
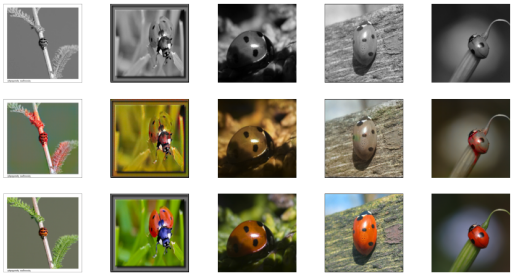


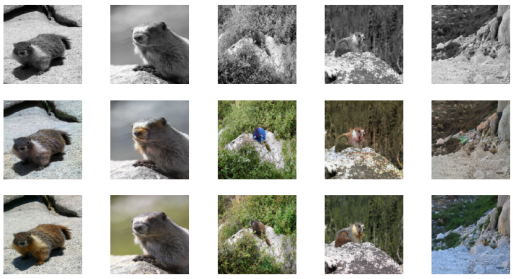
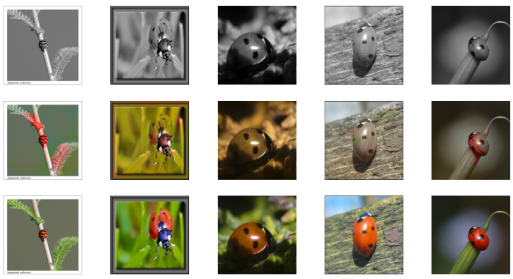
Fig. 15: Pictures at the end of training.

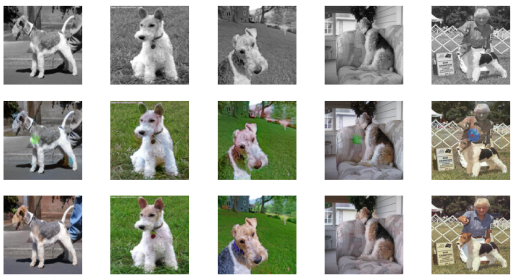
### *F. Test*

The best model has been finally evaluated on the test samples, reporting the following images.





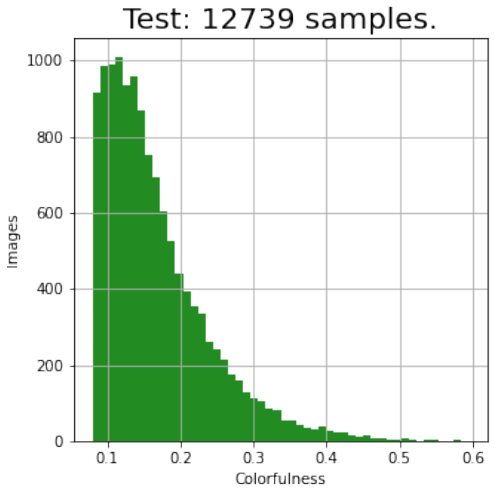



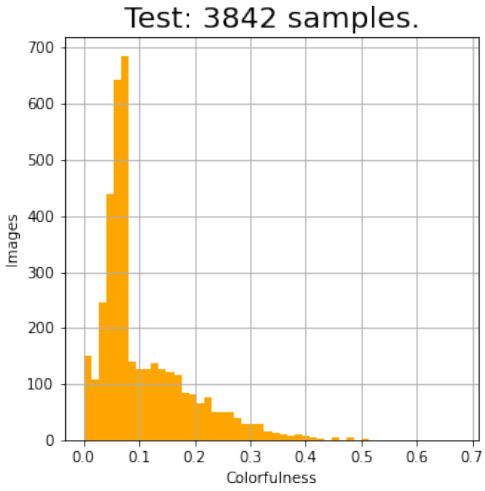


## VII. Concluding Remarks

In the present paper, a series of measurements and comparisons have been explored regarding the Convolutional Generative Adversarial Network usage in the field of image-to-image translation. The nature of the **UNET** and **patchGAN** architectures has been tested, both in terms of structural definition and learning hyperparameters. The problem of Image Recoloring has been tackled successfully, deriving the optimal model to perform such challenging work. Needless to say, a lot of improvements can still be done to reach better results, providing enough time and computational resources at disposal.

## VIII. Appendix



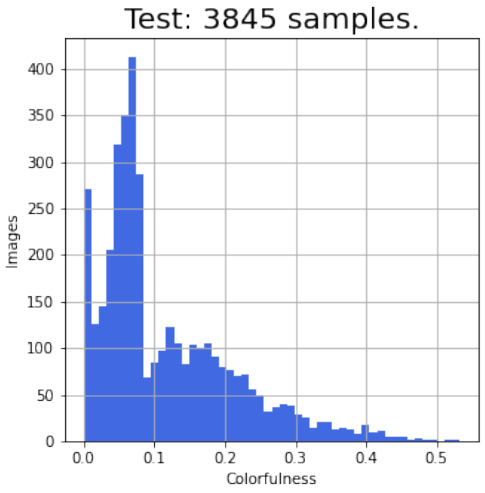




Fig. 16: Colorization distributions after cut.

| Level | Block | In-Out Channels |
|---|---|---|
| 1 | Conv2d | (1,64) |
| 2 | Conv2d | (64,128) |
| 3 | Conv2d | (128,256) |
| 4 | Conv2d | (256, 512) |
| 5 | Conv2d | (512,512) |
| 6 | Conv2d | (512,512) |
| 7 | Conv2d | (512,512) |
| 8 | Conv2d | (512,512) |
| 7 | ConvTranspose2d | (1024,512) |
| 6 | ConvTranspose2d | (1024,512) |
| 5 | ConvTranspose2d | (1024,512) |
| 4 | ConvTranspose2d | (1024,256) |
| 3 | ConvTranspose2d | (512,128) |
| 2 | ConvTranspose2d | (256,64) |
| 1 | ConvTranspose2d | (128,2) |
| | **Total Parameters** | 54.409.858 |

TABLE 7: Complex UNET.

## References


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

[2] Olaf Ronnenberg, Philipp Fischer and Thomas Brox, "U-net: Convolutional networks for biomedical image segmentation," *Computer Science Department and BIOSS CEWntre for Biological Signalling Studies, University of Freiburg, Germany*.