Image Recoloring using Generative Adversarial Networks

Cecilia Rossi
Dept. of Information Engineering
University of Padova
Padova, Italy
cecilia.rossi.3@studenti.unipd.it

Alberto Merotto
Dept. of Information Engineering
University of Padova
Padova, Italy
alberto.merotto@studenti.unipd.it

Filippo Canderle
Dept. of Information Engineering
University of Padova
Padova, Italy
filippo.canderle@studenti.unipd.it

***Abstract*—Image recoloring poses a central challenge with wide-ranging implications, and the "Pix2Pix" approach has significantly changed the way to tackle this kind of tasks. This project aims to implement a GAN by using a UNet generator and a CNN discriminator, in order to enhance image recoloring capabilities. By applying different training configurations, we exploited and compared the achieved results, yielding some interesting insights.**

The proposed UNet-CNN GAN architecture shows promising results, underscoring its effectiveness in the complex task of image recoloring.

This study aims to test the evolving landscape of GAN applications and lays the groundwork for future efforts in refining and optimizing image recoloring techniques.

***Index Terms*—Image recoloring, Pix2Pix, GAN**

I. INTRODUCTION

Image Recoloring is a challenging task that can be met in the field of computer vision, photography and computer graphics with several applications such as: restoring historical images, acquiring initially black and white images for subsequent colorization, or revisiting partially damaged images.

The Machine Learning-based approach to this field was firstly brought to research in the early 2000s, when some initial algorithms were proposed. However, these early models had several limitations: they required extensive manual intervention and couldn't achieve results comparable to directly captured color images quality. [2]

The advent of Convolutional Neural Networks (CNN) simplified the automation of the image recoloring process by developing models that learn color patterns from extensive datasets. Further advancements were then introduced by the implementation of Generative Adversarial Networks (GAN) [3], enabling a more realistic color generation through the competition between a generative and a discriminative network. Recent techniques are also able to incorporate user interaction elements or hints to guide the colorization process.

This project aimed to implement a GAN approach to address the image recoloring task, by training, in several configurations, a model capable of automatically recoloring gray-scale images. Obviously, we didn't expect to achieve state-of-the-art results, but we wished to understand which could have been the best parameters configuration to accomplish it.

This report is structured as follows:

- In Section II, we will describe the state of art for this purpose;
- In Section III, we will present the dataset used in this work from a high-level perspective;
- In Section IV, the pre-processing steps will be illustrated;
- In Section V, the learning frameworks will be described in detail, comparing different configuration used;
- While in Section VI and VII, the performance evaluation of the model and the final conclusions will be reported.

II. RELATED WORKS

Impressive results were achieved trough a Pix2Pix GAN approach by Isola et Al, in a paper published in 2017, that we used as principal reference for our work [1]. Pix2Pix is a Generative Adversarial Network technique that focuses on conditional image generation. It operates by training the model on pairs of corresponding input and target images, enabling it to learn how to generate realistic results. Pix2Pix is particularly effective for tasks like image-to-image translation, where the goal is to transform images from one domain to another, such as turning grayscale images into colored versions or converting satellite images to maps. This technique utilizes adversarial training, involving a generator network that creates images, without checking directly the input data, but based on the output given by another discriminator network that distinguishes between real and generated images, refining the generator's output over time.

In the Pix2Pix technique, different loss functions were used to guide the training of the generator and discriminator.

Adversarial Loss

The adversarial loss function used was the Binary Cross Entropy with Logits (BCEwithLogits) , which returns a measure of the generator's ability to deceive the discriminator, or the overall model performance. It can be expressed as:

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

This loss computes the Binary Cross Entropy and combines it with a Sigmoid, which is the reason why this last layer is missing from our architectures implementation.

L1 Loss

The Binary Cross Entropy was then combined with a L1 loss, also called Mean Absolute Error, which measures the pixel-wise difference between the generated output and the target image:

$$\mathcal{L}_{\text{L1}}(G) = \mathbb{E}_{x,y,z}\left[\|y - G(x,z)\|_1\right]$$

The weighted application of the L1 measure in the generator loss, approach also implemented in the original paper, can help mitigate the blurring effect in image generation tasks, encouraging the generator to produce outputs that are not only realistic but also visually similar to the target images.

Pix2Pix Total Loss

While, in order to train the discriminator network, we merely back-propagated the Binary Cross Entropy loss, for the back-propagation of the generator's error and the evaluation of the performance of out model we applied a combination between the Adversarial Loss and L1 Loss:

$$\mathcal{L}_{\text{Pix2Pix}}(G,D) = \mathcal{L}_{\text{CGAN}}(G,D) + \lambda \cdot \mathcal{L}_{\text{L1}}(G)$$

where λ is a balancing parameter.

The final optimization problem is then depicted as minimax zero-sum game:

$$G^* = \underset{G}{\operatorname{argmin}} \max_D \mathcal{L}_{\text{Pix2Pix}}(G,D)$$

In addition to the Pix2Pix technique, there have been several advancements and variations in the field of conditional image generation and GANs.

- *CycleGAN*, proposed by Zhu et al. in 2017 [7], introduces a cycle consistency loss to enable unpaired image-to-image translation. It does not require paired training data, making it suitable for tasks where corresponding input and target images are not available.
- *StarGAN*, introduced by Choi et al. in 2018 [8], extends conditional GANs to handle multiple domains with a single model. It allows for the generation of images across different domains without the need for domain-specific models.
- *UNIT (Unsupervised Image-to-Image Translation).* Introduced by Liu et al. (2017) [4], it employs a novel approach for unsupervised image translation, where a shared latent space is utilized across different domains, allowing the model to learn mappings between domains without the need for paired training data.

In our project, we used the "pix2pix" approach as a foundation. Relying on this structure, developed in Python using the PyTorch framework, we slightly modified the original architecture and tried experimenting with some training parameters, to draw conclusions regarding the general performance of the model with respect to the image recoloring specific task.

III. SIGNALS AND FEATURES

A. COCO Dataset

The COCO dataset, short for Common Objects in Context, is a widely used image dataset in computer vision research. It can be applied as a benchmark for various tasks within the scientific community, providing a diverse and challenging set of images for evaluation. The dataset consists of high-quality, high-resolution images that depict complex scenes with multiple objects. Each image is annotated with object bounding boxes, segmentation masks, and captions, making it suitable for a range of tasks such as object detection, instance segmentation, and image captioning.

The dataset contains images with varying dimensions, color distributions, and object complexities, ensuring a comprehensive assessment of algorithms across diverse scenarios.

In our project, we employed a subset of the COCO dataset, striking a balance between achieving high-quality results and minimizing training times. This choice aimed to address computational constraints while still obtaining meaningful insights from the dataset. To facilitate the data acquisition process, we used the FiftyOne library, a powerful tool for exploring and managing large-scale image datasets. Before training our models, we performed pre-processing on the images, ensuring consistency and compatibility with our specific experimental setup.

Specifically, the number of images used for the entire project, in all configurations, was the following:

Configuration	Training Images
1,2,3	10,000
4,5,6	20,000

TABLE I
NUMBER OF GENERIC IMAGES

B. Car Dataset

In addition, another dataset consisting solely of car images was exploited. This dataset, obtained from Kaggle [9], aims to train the model in the task of image recoloring for a specific category of images. By employing this dataset, the goal is to evaluate the GANs performance in a particular context.

We choose this particular dataset because, while the majority of images are of simple cars with a blank background, it also contains a portion of images of interior of cars and cars on the road with a natural background, allowing the model to learn particularly well the structure of the cars and then try to generalize it to a more realistic scenario.

The final dataset comprises 60,000 pictures, but for this project, a subset was downloaded and used.

Configuration	Training Images
7,8,9	10,000

TABLE II
NUMBER OF CAR IMAGES

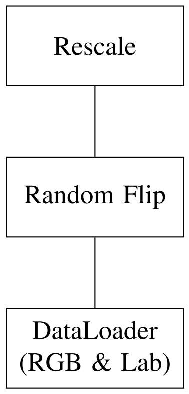
IV. PROCESSING PIPELINE

The pre-processing adopted in this project was kept very simple with the aim of preserving the images as similar as possible to the original ones. After acquiring the training and test datasets, all images underwent the following processing steps:

- **Rescaling**: It was decided to scale the images to a predefined size to efficiently use them within our GAN. Both the U-net (Generator) and CNN (Discriminator) architectures require standardized dimensions. We experimented with various image sizes in order to find the best compromise, between performance on the final architecture and computational complexity. The image dimension that allowed us to achieve the best tradeoff was 128x128. This size allowed us to perform a downscaling that could be applied uniformly to all images, while for larger dimensions (e.g., 512), we might encountered issues with upscaling certain images. Also, the applied downscaling was not overly aggressive, allowing us to preserve the main details of the image. Other image sizes were tested (e.g., 256), but they did not contribute to a satisfactory final result during the image recoloring testing phase and required an excessive computational power.
- **Flip**: Additionally, a random flip was applied along both dimensions of the image (horizontal and vertical) with a 50 percent probability. This transformation increased the variability of our dataset and allowed the architectures to be trained on different scenarios, aiming for better generalization.
- **Dataloaders**: Subsequently, the images were transferred to "*DataLoaders*", Python structures capable of efficiently handling datasets. It was chosen to create the DataLoaders in two color spaces.
  1) RGB: The standard color space, representing colors by combining different intensities of red, green, and blue shades. Therefore, each pixel in an RGB image is defined by three values.
  2) Lab: It's an alternative color representation, consisting of three components: L* (lightness), a* (green to red), and b* (blue to yellow). The L* component represents brightness(in our case the grayscale image), while a* and b* encode color information.

  We decided to create a DataLoader in Lab to test it on our U-net (the generator of the GAN) and see if there could have been some advantages. Since the architecture is required to generate only 2 channels of the recolored matrix (a and b), instead of 3 as required in the RGB space, we expected to find better results both in term of model performance and computational effort.

In summary, our processing pipeline can be seen as follow:



V. LEARNING FRAMEWORK

Our learning framework, as previously mentioned, is based on a Generative Adversarial Network (GAN), which consists of two antagonistic structures, a generator, and a discriminator, working competitively in a zero-sum game.

We have invested the majority of our effort in implementing a GAN composed of a U-Net [10] as generator and a Patch-CNN as discriminator. This can be defined as the "main architecture" of our project. Subsequently, after experimenting with this architecture using the configurations described in section 5.C "Configurations," we tested several modifications. Initially, we attempted the same architecture in a specific configuration with images in the "RGB" image format, aiming to compare the results with the same configuration in "Lab" color space. Finally, we explored different types of generators while keeping the discriminator unchanged to assess the GAN's performance and compare it with the "main architecture".

A. Main Architecture

- **Generator**: The generator architecture was assembled following a U-Net scaffold. It consists of down-sampling (encoder) and up-sampling (decoder) layers, emphasizing skip connections and batch normalization for stable and effective training.
  The 7 down-sampling layers employ LeakyReLU activation and Batch Normalization to reduce spatial dimensions progressively. On the other hand, the 7 up-sampling layers use ReLU activation and Batch Normalization to increase spatial dimensions.
  Dropout layers are strategically applied during up-sampling with a specified dropout rate of 0.5.
  Notably, Skip Connections are established through concatenation, preserving high-level features by combining feature maps from the down-sampling path and the corresponding up-sampling path. The final output is produced by the last up-convolutional layer, followed by a hyperbolic tangent activation (tanh) to ensure output values fall within the range [-1, 1].
  For all the convolutional layers (both downsampling and upsampling) a kernel size of 4 has been used, with a stride of 2 and padding equal to 1.

The "LeakyRelu" activation function used through the convolution layers is:

$$\text{LeakyReLU}(x) = \begin{cases} x & \text{if } x \geq 0, \\ \text{negative\_slope} \cdot x & \text{otherwise.} \end{cases}$$

and, at the end, the activation function used is:

$$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$$

- **Discriminator:** The discriminator is a Convolutional Neural Network(CNN) implemented with the aim of classifying between real colored images (the original training images) and generated colored images coming out from the U-net structure. Exploiting the fact that convolution has a local receptive field, it will aim to assign a boolean classification not strictly to each pixel of the generated images (or real images, since the discriminator is the only part of the model to which a direct access to the train data is allowed), but only to the underneath 14x14 patches of the image. In this way, we are actually able to take advantage of the ability of convolutional layers to capture spatial correlation in the images. The discriminator architecture comprises five convolutional layers, progressively reducing spatial dimensions while utilizing Leaky ReLU activation and batch normalization. The first layer takes the input channel and returns 64 channels using a 4x4 kernel, a stride of 2, and Leaky ReLU activation with a slope of 0.2.
  Subsequently, the second and third layers continue this down-sampling process, increasing the channel size to 128 and 256, respectively. Batch normalization is applied after each convolutional layer to enhance training stability. Leaky ReLU activation with a slope of 0.2 persists throughout.
  The fourth layer maintains 256 channels and adopts a 4x4 kernel with a stride of 1, further refining feature representation. Batch normalization and Leaky ReLU activation are applied.
  The final down-sampling layer, prepares for the binary classification task. It employs a 4x4 kernel with a stride of 1 and Leaky ReLU activation.
  Finally, no Sigmoid activation function is applied, since Binary Cross Entropy loss is employed.

The training procedure, like the majority of Machine Learning tasks, is aimed to solve an optimization problem. The loss function that we want to minimize is a linear combination of Binary Cross Entropy (BCE) and Mean Absolute Error (L1), as already attempted by researchers in [1].

$$\mathcal{L}(\theta_G, \theta_D) = \mathcal{L}_{\text{BCE}}(\theta_G, \theta_D) + \lambda \cdot \mathcal{L}_{\text{L1}}(\theta_G),$$

With a λ value of 100.
Adam optimizer [6] was employed in stochastic optimization of this GAN, with learning rates of 0.0002 for both the Generator and Discriminator.

B. Further Architectures

As previously mentioned, other generator architectures were employed to deeply investigate how to improve the GAN performances or, at least, make a comparison with the "*Main architecture*".

- **Autoencoder:** This alternative architecture is pretty similar to our original model, with which it shares the encoder-decoder structure. It also exploits bidimensional convolutional layers, Batch Normalization, Leaky Relu (or ReLu in the up-sampling part) and the insertion of dropout layers during the decoding process, to enhance the network's generalization capabilities. The final output is generated by the last transposed convolutional layer, with a subsequent application of hyperbolic tangent activation (tanh). Consistently across all convolutional layers, both in down-sampling and up-sampling, is the use of a kernel size of 4, a stride of 2, and padding set to 1. The main difference with respect to our starting architecture is the insertion of a middle block and the use of a Max Pooling layer, in order to effectively decreasing spatial dimension.
- **ResNet:** With its residual connections, ResNet is designed to address the vanishing gradient problem in deep neural networks. It is commonly used in classification tasks and has been adapted for image-to-image translation tasks. Furthermore, the residual connections enable the model to learn identity mappings, facilitating the training of deeper networks. Our architecture consist of an initial 7x7 convolutional layer with batch normalization. Then, a downsampling module reduces spatial dimensions. After that, residual blocks capture and propagate essential image details.Then, an upsampling module gradually restores spatial dimensions. Finally, a 7x7 convolutional layer produces RGB colorized output with Tanh activation.

C. Configurations

To pursue our goal of image recoloring with the above-mentioned architecture, we tried various hyperparameter combinations, aiming to understand which could have been more convenient in terms of computational cost, training time, and the required number of samples. We decided to keep the batch size fixed at 8 for each configuration, varying the number of training samples (10k and 20k) and the number of epochs (15, 30, 45). Then, we focused on comparing a model trained with the same number of images (10k), this time varying the type of dataset: on the one hand, we maintained a dataset of generic images (the COCO ones), while on the other hand, we selected images belonging to a specific category (the cars dataset). What we expected from this comparison was to achieve better generalization capabilities in models with the first type of dataset, while obtaining better performance on the test set of images belonging to the second dataset. At the same time, we wondered if, by applying the model trained on the specific dataset to more general images, we could have reaches good generalization and results, anyway.

More specifically, the configurations we will focus on are:

Configuration	Batch size	Nr. samples	Epochs	Dataset
1	8	10K	15	generic
2	8	10K	30	generic
3	8	10K	45	generic
4	8	20K	15	generic
5	8	20K	30	generic
6	8	20K	45	generic
7	8	10K	15	Cars
8	8	10K	30	Cars
9	8	10K	45	Cars

TABLE III
TRAINING CONFIGURATIONS

After looking at the results, we decided to train the alternative architectures (with Autoencoder and ResNet as generators) only using a number of train data equal to 10K and 45 epochs, since this specific configuration was the one that returned better results using the original architecture. The same idea was enforced also in the comparison between training on data belonging to the Lab color space and the RGB color space, for computational and time-efficiency reasons.

D. Technology Architecture

In our project, we implemented the code using Python, in particular exploiting Pytorch library, a widely applied Machine Learning tool. Other Python libraries(e.g. Numpy, PIL, matplotlib) were used during the development of the pipeline.

VI. Results

In the following section, we will illustrate the obtained results, with particular focus on the comparison between different architectures and hyperparameters/data configurations:

- Firstly, we will show the different train losses and performance on a subset of the test set, of out principal model (U-Net with a Patch CNN Discriminator) trained on a different amount of data;
- We will, then, face a brief digression about the same comparison done by maintaining a constant the number of training images, but varying the number of epochs used in the training of the model;
- After that, we will reveal the results obtained by the same model if trained on data in a different color space, e.g. the Lab color space;
- Finally, we will display in parallel the performances and train losses by using alternative architectures, with respect to the ones returned by our original model.

While in the first two steps we exploited both the Generic (COCO) Dataset and the Specific one (Cars Dataset), in the following two we focused only on the Specific dataset, since it was the one that returned better results in the early analysis.

A. Comparison in U-net

1) Different Amount of Data: By training our original U-Net model on a different amount of data, we realized that the generator behaviour remained quite similar and the shape of the generator loss seemed to follow the same pattern. By using a double amount of data [20k images, instead of 10k], we started from a lower initial loss value, but in the end the average loss converged to an higher value compared to the one reached by exploiting a lower amount of data. We assumed that this was due to a higher change of overfitting, confirmed also by the average losses computed on the different batches of the test set, in which, after training the model for 45 epochs on 20k images, we seems to reach comparable performances than by exploiting 10k.

By using the more specific Cars dataset (that we trained only on 10k images), we reached a much lower loss value both regarding the initial training epoch and once reached convergence.

This is explainable by the characteristics of the dataset, in which black, white and primary color details are dominant and, therefore, easier for the network to learn. Also, the fact that all the images share the same object enables the network to reach better results on the training, since a lower extent of generalization is required.

The amount of data exploited in the training seems, instead, more impacting on the loss of the discriminator. In fact, by using a larger number of images, the discriminator loss drops to a lower value. In both cases, the generic dataset doesn't enable the generator to actually trick the discriminator into recognizing the generated images as real, increasing its loss value. Instead, by using images coming from the specific dataset, we notice how the discriminator loss tends show a plateau, even if in quite unstable way, with a slight tendency to increase after 30 epochs or so. This behaviour is more in line with what we would expect in a Adversarial Network model.

2) Different Number of Epochs: As we can appreciate from the Fig from 7 to 10, by using a larger number of epochs the model performance improves. This improvement is much more evident if we are employing the generic dataset, then if we are using the specific one. In this second case, in fact, even by using a smaller amount of epochs, we are actually able to reach quite good results and a satisfactory generalization also on the image details different from the car objects, as the streets and backgrounds. Instead, with the general dataset, an higher number of epochs seems fundamental in order to improve the model generative performance.

3) Test Set Performance Evaluation: The same considerations can be extended by observing the averages loss computed on the different batches of the test set. In particular, while using a higher number of images during the training we are able to lower the Test Loss in both dataset. But, this improvement of the performance becomes negligible if we use 45 epochs. If applying the specific dataset, instead, performance remains steady with respect to the variation of the number of training epochs.

TABLE IV
AVERAGE TRAINING LOSSES IN THE LAST EPOCH

Epochs	Dataset		
	10k	20k	10k Cars
15	14.07	13.75	6.97
30	13.41	13.40	6.65
45	12.79	13.85	6.01

TABLE V
AVERAGE TEST LOSSES

Epochs	Dataset		
	10k	20k	10k Cars
15	13.74	12.67	6.58
30	13.64	14.32	6.61
45	13.02	12.98	6.71

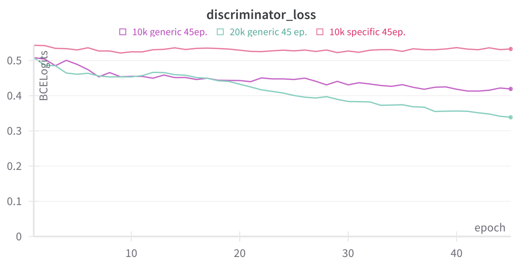


Fig. 1. Discriminator loss of all Datasets, generic and specific (cars images), at 45 epochs

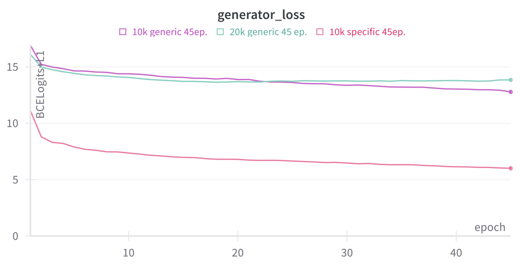


Fig. 2. Generator loss of all Datasets, generic and specific (cars images), at 45 epochs

B. Comparison: RGB vs Lab

In the second part of our analysis, we only took into account a training procedure using 10k images from the specific Cars dataset. In this case, we compared the results gained by our principal GAN model (U-Net as generator and Patch CNN as discriminator), by giving as input the same images, but represented through the Lab color space. This required a slight change in the architecture of the generator, so that it would return a dimensional tensor as output, associated with the a and b channels of the Lab representation, after giving it as input the Brightness (L) channel. The generator output was then combined with the gray-scale image, before giving it into input to the discriminator. We expected, by offering as input to the network images in the Lab color space, to reduce the computational complexity of the image generation and to improve the performance of the model. The obtained results partially confirmed this hypothesis.

Looking at the graphs and table data, it can be observed that the generator loss is better in the Lab color space compared to RGB, with a comparable training time. This is likely due to the fact that the U-net, when trained on Lab images, only needs to learn the chromatic values (A and B). Conversely, training the network on RGB requires the generator to learn how to generate three channels, making it a more challenging task. However, we observed from our test set that some images produced by the Lab model occasionally tend to saturate in the blue tone, making the final output less faithful to the ground truth. It is essential to note, though, by observing the generator loss curves, that they tend to converge to a similar value. Thus, we cannot conclusively state that one color space is superior to the other. As is often the case in engineering, it depends on the specific application.

TABLE VI
RESULTS: RGB AND LAB WITH UNET

Color space	Metrics		
	Training Loss	Test Loss	Training Time
RGB	6.01	6.71	1h 22m 36s
LAB	5.08	3.82	1h 22m 20s

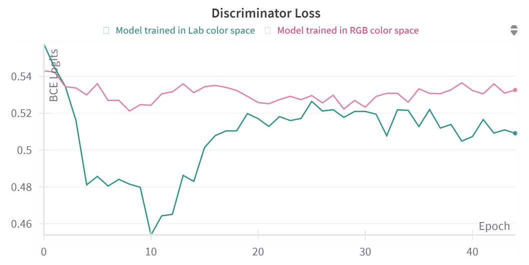


Fig. 3. Discriminator loss - RGB vs Lab

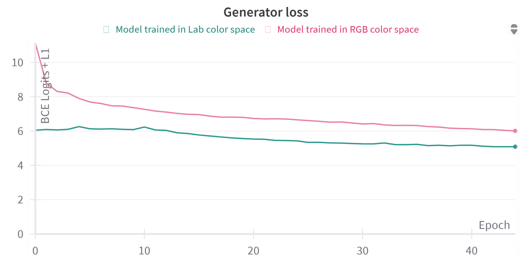


Fig. 4. Generator loss - RGB vs Lab

C. Comparison: Unet vs ResNet

We expected the ResNet, being a more powerful tool, to yield better results. Comparing the test loss values and the images produced by the two architectures, it can be appreciated that the ResNet, leveraging its residual blocks, achieves a lower generator loss value, indicating better performance in the image recoloring task. This observation is further confirmed by examining the output generated.

Upon close inspection of the images, it becomes evident that the ResNet excels in coloring smaller details with greater fidelity, avoiding the blurring of details observed in the U-net. In the U-net, some details tend to become more coarse. It's important to note that the U-net, specifically designed for image-to-image translation tasks, still manages to achieve excellent results.

However, there is a drawback to this architecture. Comparing the training times, it is noticeable that the ResNet takes 40 % more time than the U-net, making this model more complex. This factor should be taken into consideration, especially when dealing with much larger datasets during training.

TABLE VII
RESULTS: UNET AND RESNET

Model	Metrics		
	Training Loss	Test Loss	Training Time
U-Net	6.01	6.71	1h 22m 36s
ResNet	6.97	6.41	2h 21m 52s

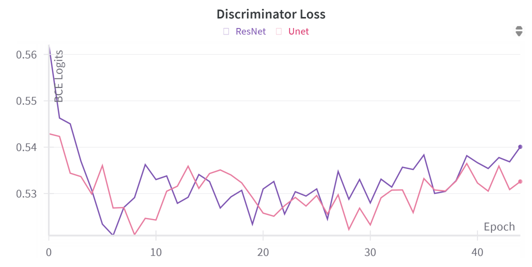


Fig. 5. Discriminator loss of Unet and ResNet

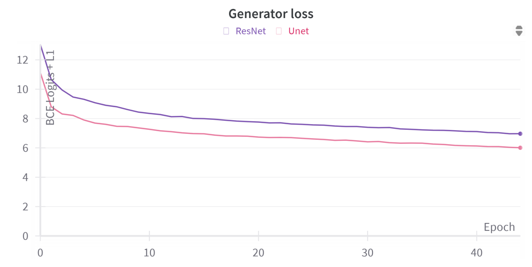


Fig. 6. Generator loss of Unet and ResNet

D. Further results

We didn't insert in this report the plots regarding the comparison between the losses obtained by training the model using the Autoencoder, instead of the U-Net, in the generator. However, also from the images included below, it appears clear that this type or architecture is unable to correctly perform the task, yielding very poor results. This is likely due to the shallow architecture of the model. So, despite using the same number of epochs and input data as we previously did for the U-Net, the generated images were significantly worse. From testing this structure, we can conclude that a shallow architecture is still able to recognize raw and structural details of the grayscale images, but falls short in the image recoloring task.

VII. Concluding Remarks

Throughout this research on the use of GANs for image recoloring, we have observed significantly better results when the model is trained using specific images compared to a generic dataset. However, a clear trade-off has emerged between achieving superior performance with specific images and the model's ability to generalize.

It is interesting observe that all the examined models share common drawbacks, including the presence of a ”greenish prior”, occurrences of ”color bleeding”, and a tendency towards ”grayish colorization”. These aspects are crucial considerations in further refining algorithms for recoloring.

In conclusion, this work serves as a starting point in understanding and optimizing GAN models for image recoloring, and just wanted to make a brief comparison between different models, different datasets and number of epochs. It is important to emphasize that, with additional resources and further research efforts, the model can be significantly improved.

Contribution of Each Author

Retrospectively assigning roles to each component becomes challenging when responsibilities have been distributed organically among the components throughout the project's evolution. This project presented a captivating challenge, blending practical tasks with a learning process aimed at addressing relevant issues.

In reality, collective contributions were fairly distributed across all project stages, with each team member engaging in coding tasks and participating in high-level decision-making. This approach facilitated a comprehensive understanding of every project phase, allowing us to navigate through diverse challenges.

While acknowledging our collaborative efforts, roles could be assigned as follows:

Filippo Canderle primarily focused on the initial stages, concentrating on data scraping, handling, and preprocessing. He dedicated efforts to ensure the datasets were appropriately prepared for use in the architecture.

Cecilia Rossi played a key role in designing the architecture and creating Python classes. Her focus was on implementing the discriminator, various generators, and maintaining overall coherence in the system.

Lastly, Alberto Merotto contributed to preparing and supervising the training procedures. He oversaw the early stages of each training session and acquired final results, including graphical representations and test example images showcased in this report.

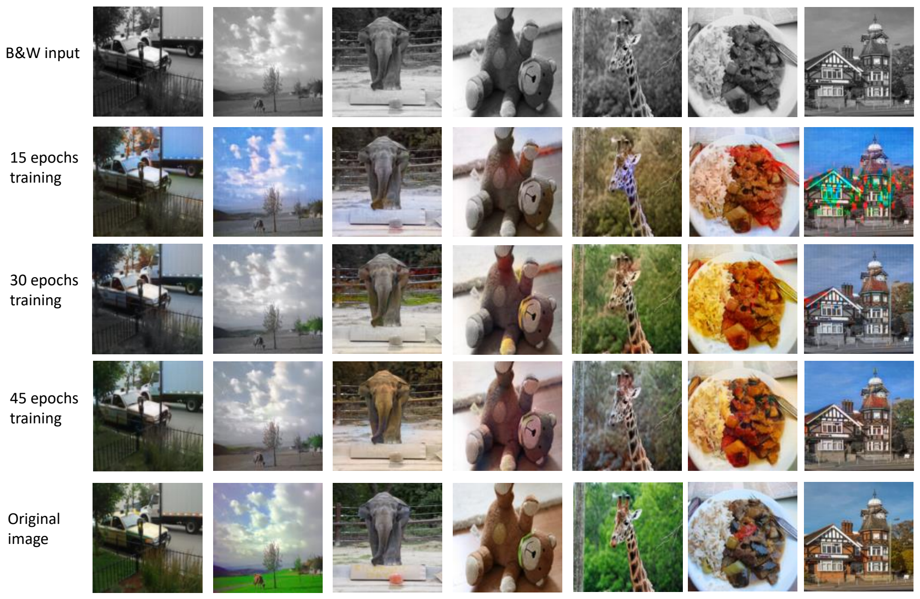


Fig. 7. Results for different numbers of training epochs with a dataset of 10'000 images

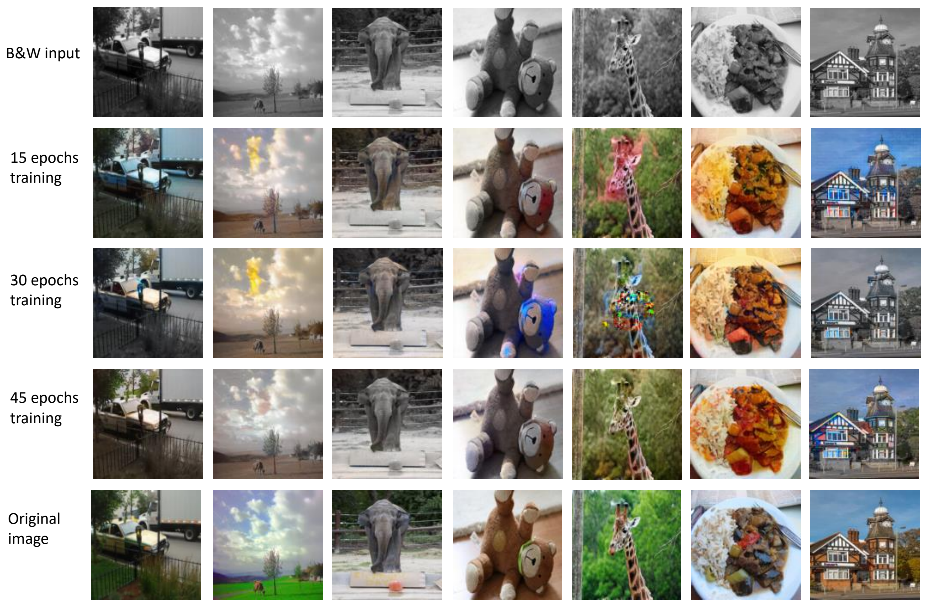


Fig. 8. Results for different numbers of training epochs with a dataset of 20'000 images

Moreover, each team member conducted thorough reviews and corrections of the other's work, in a continuous peer-to-peer review process throughout the project. The manuscript, specifically, was collaboratively written using the Overleaf environment, with each co-author making similar contributions to its development.

References

[1] Isola, P., Zhu, J., Zhou, T., & Efros, A. A. (2016). "Image-to-Image Translation with Conditional Adversarial Networks." ArXiv:1611.07004.

[2] Y. Wang et al., "A Review of Gray-scale Image Recoloring Methods With Neural Network Based Model," 2022 7th International Conference on Image, Vision and Computing (ICIVC), Xi'an, China, 2022, pp. 462-467, doi: 10.1109/ICIVC55077.2022.9886935.

[3] Goodfellow, I. J., Mirza, M., Xu, B., Ozair, S., Courville, A., & Bengio, Y. (2014). "Generative Adversarial Networks." ArXiv:1406.2661.

[4] Liu, M., Breuel, T., Kautz, J. (2017). Unsupervised Image-to-Image Translation Networks. ArXiv. /abs/1703.00848

[5] Lin, T., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., & Dollár, P. (2014). "Microsoft COCO: Common Objects in Context." ArXiv:1405.0312.

[6] Kingma, D. P., Ba, J. (2014). Adam: A Method for Stochastic Optimization. ArXiv. /abs/1412.6980

[7] Zhu, J., Park, T., Isola, P., Efros, A. A. (2017). Unpaired Image-to-

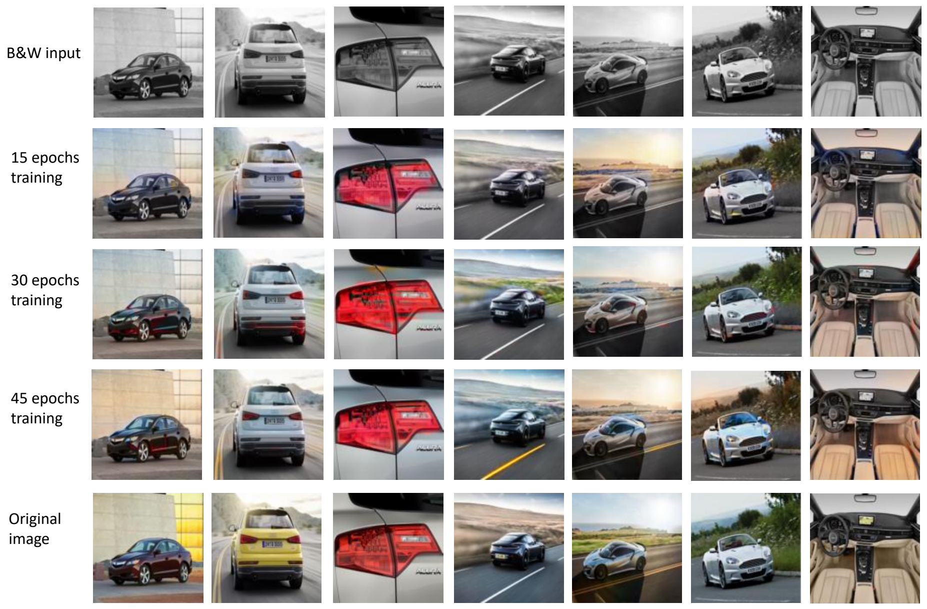


Fig. 9. Results for different number of training epochs with a dataset of 10’000 images of cars

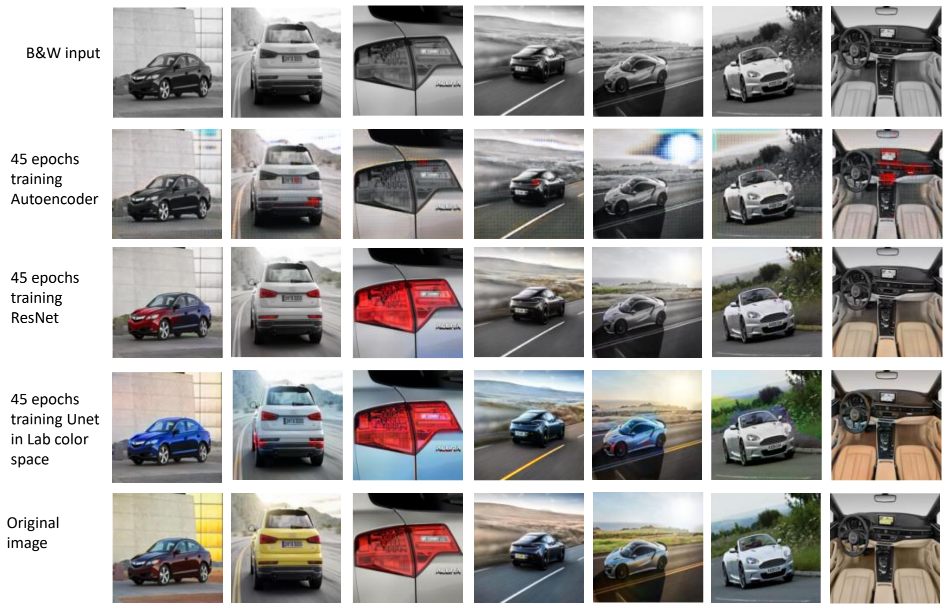

B&W input
45 epochs training Autoencoder
45 epochs training ResNet
45 epochs training Unet in Lab color space
Original image

Fig. 10. Results for different architectures tested with a dataset of 10’000 images of cars

Image Translation using Cycle-Consistent Adversarial Networks. ArXiv. /abs/1703.10593

[8] Choi, Y., Choi, M., Kim, M., Ha, J., Kim, S., Choo, J. (2017). StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation. ArXiv. /abs/1711.09020

[9] https://www.kaggle.com/datasets/prondeau/the-car-connection-picture-dataset

[10] Ronneberger, O., Fischer, P., Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. ArXiv. /abs/1505.04597