Image Colorization with Conditional Generative Adversarial Networks

Chrysoula M. Nampouri
c.m.nampouri@student.rug.nl
University of Groningen
Groningen, the Netherlands

Christodoulos Hadjichristodoulou
c.hadjichristodoulou@student.rug.nl
University of Groningen
Groningen, the Netherlands

Filippos Andreadis
f.andreadis@student.rug.nl
University of Groningen
Groningen, the Netherlands

Abstract

Image-to-Image translation corresponds to the task of pixel-to-pixel value mapping. Recent research has shown that Conditional Generative Adversarial Networks, a variant of Generative Adversarial Networks, can be used as a general-purpose solution for a variety of sub-tasks of this domain. Part of the reason for their success is the fact that they are able to learn at the same time a mapping from the input to the output image domain as well as a loss function to train this mapping. In this study, we focus on the problem of image colorization using Conditional Generative Adversarial Networks. We evaluate two different modules used as Generators of the final network, namely a U-Net and a pre-trained ResNet, and we perform a comparative analysis between them. Even though we restrict the size of the input data to a fraction of what has been used in similar works, the generated images do not fall far away from the ground truth.

Keywords: Image-to-Image translation, Image colorization, Conditional Generative Adversarial Network, Convolutional Neural Networks

1 Introduction

The aim of image colorization is to augment a monochromatic image with color in order to produce a colorful result. Eventually, the grayscale input must be mapped to a visually plausible and perceptually meaningful color image. This concept has its roots dating back to the late 19th century when the French film-making pioneer Georges Méliès used hand-colored film to introduce color to his motion pictures. Since then and due to the huge technological leap, sophisticated deep learning systems have been developed that are able to automatically apply color to images with impressive accuracy, making digital colorization an integral tool for restoration of vintage material and art, among others.

Faculty of Science and Engineering, University of Groningen, the Netherlands, April 2023

.

The rest of the paper is structured as follows: Section 2 presents an overview of the approaches proposed in prior research on the topic. In Section 3 we provide a thorough explanation of all the components and techniques involved in building the final image colorization pipeline, while in Section 4 we go into the details of the methodology we followed. In Section 5 we present the dataset that we used for training our models, along with details about the data pre-processing and the training procedure itself. In Section 6 we display the results of our work by reporting the final losses achieved by each one of our models along with a comparison of the output images and the real colored ones. Finally, Sections 7 and 8 include our comments and intuition upon the results, as well as our general final thoughts about the project and possible future improvements.

2 Related work

The task of image colorization has been studied for a couple of decades with many different approaches. Early attempts relied on transferring color between a source, color image and a destination, grayscale image [1] or on propagating the colours provided by a pseudo-annotation in the form of scribbles on top of the image [2].

In later years, research was geared more towards techniques that do not require user interaction. One of these techniques is stating the problem mathematically as an optimization problem with an explicit energy function to minimize [3]. Morimoto, Taguchi and Naemura [4] introduced a method by which the input image is matched to several reference images with similar structure from a large database and then multiple color images are generated by using each reference image. During the past decade, the rapid developments in deep learning motivated researchers to tackle this issue using Convolutional Neural Networks (CNNs) [5–7], allowing the models to automatically discover and exploit higher level characteristics in the images.

In recent years, specialized generative models have become the state-of-the-art for image-to-image tasks such as image colorization. These models learn to generate samples

that look like they belong in the specific distribution that was used to train them. Xiao et al [8] trained a modified version of CycleGAN [9] to perform this task, while Variational Autoencoders have also been used in combination with a multi-modal conditional model [10].

The work that inspired this project is the paper by Isola et al [11], in which they propose a general purpose framework for image-to-image translation problems by leveraging Conditional Generative Adversarial Networks (cGANs) [12]. In their implementation, the generator has a UNet-like [13] architecture with skip connections and the discriminator is multi-layer network with a series of convolutions, ending in a dense layer for classification. They also suggest the addition of the L1 loss between the original image and the output image to cGAN's loss, based on similar previous approaches that found it beneficial. This solution has been applied to many tasks: sketch-to-photo, day-time images to night-time images and of course grayscale images to fully coloured images.

3 Background

This section covers the explanations of the general working of Convolutional Neural Networks (CNNs), including the two particular architectures used in our CGANs, as well as the principles of Transfer Learning. Additionally, we introduce the concept of Generative Adversarial Networks (GANs) and how they can be tailored for the task of Image-to-Image translation.

3.1 Convolutional Neural Networks

CNNs are a category of artificial feed-forward neural network, where the matrix multiplications among the fully connected layers are replaced by the convolution operation. This type of network is specialized in Computer Vision tasks and there is a plethora of specifically designed CNN architectures for a variety of problems including image classification, segmentation, and recognition, among others. The concepts of weight sharing and sparse connectivity have allowed for a much lower computational complexity and efficiency compared to that of Multi-Layer Perceptron (MLP). The network learns to optimize a set of filter kernels through consecutive applications of the convolution function with the input. This way the input is reduced to a feature space of lower dimensionality, which the models uses as a representation of the entire input after learning all its important spacial features. This means that the network optimizes the filter kernels and automatically learns all the important parts of the input in an end-to-end fashion, erasing the need for hand-crafted features.

Residual Neural Network (ResNet). Even though CNNs have yielded state-of-the-art results in big range of tasks, previous research has shown that after increasing the number of layers beyond a certain point not only halts the improvement in performance, but rather has a negative impact. This is mainly due to the vanishing/exploding gradient effect, a problem that typically occurs in very deep architectures. He *et al.* introduced the ResNet in [14] in order to overcome this issue and push further the capabilities of CNNs. The novelty of this architecture is the skip-connection, which allows the gradient to be propagated to the next layers and mitigate saturation. ResNet consists of multiple instances of a residual block, which is illustrated in Figure 1.

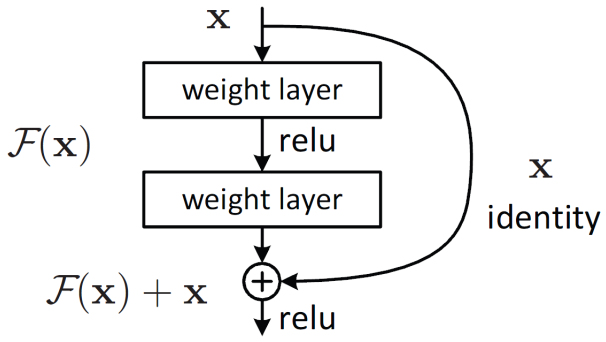

Figure 1. Residual Block [14]

Given an input x and the corresponding output of a layer $H(x)$, we define the residual $F(x)$ of the layer as shown in Equation 1. A traditional network would try to learn the mapping between these two, however since a residual block includes an identity connection coming from x, the layers of ResNet are actually trying to learn the residual, $F(x)$. An important constraint is that x and $F(x)$ must have the same dimensionality.

$$F(x) = H(x) - x \tag{1}$$

U-Net. Ronnenberger *et al.* proposed in 2015 a CNN variant that introduced a significant improvement in performance for the task of image segmentation. U-Net [13] is a convolutional network tailored for biomedical imagery segmentation and has improved upon the state-of-the-art of the time, while being trained on much fewer data compared to its predecessors. Typically, it is considered an encoder-decoder model and its core idea is to exploit feature representations

learned in the encoder, by directly using them in the decoding stage. An overview of the U-Net architecture can be seen in Figure 2.

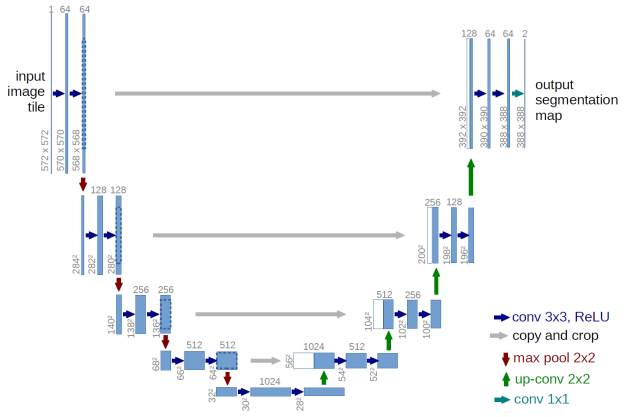

Figure 2. U-Net architecture [13]

The architecture consists of a contracting path to capture context and a symmetric expanding path that enables precise localization. Additionally, a third bottleneck component acts as a link between these two. The contraction block essentially consists of multiple convolution layers that allow the network to learn complex structures of the input, while the dimensionality of the feature space is reduced as we move deeper. The expanding component consists of symmetrical transposed convolution layers so that the dimensions of the feature map are increased again. An integral part of the U-Net are the connections between the corresponding contraction and expanding blocks. The feature representations of each contraction block are concatenated with the feature map of the respective expanding block. These connections have shown to benefit the quality of the reconstructed output and also help to move the gradient along the layers during back-propagation.

3.2 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [15] are a type of machine learning models that involve two neural networks, the generator and the discriminator, working together in a competitive process to generate new data that resembles a training dataset.

GANs have many applications, including generating realistic images, video, and audio, as well as generating text and other types of data. They have been used in a variety of fields, including computer vision, natural language processing, and music synthesis, among others. Despite their many applications and successes, GANs also have some limitations and challenges. One major challenge is that they can be difficult to train and can suffer from instability and mode collapse, where the generator produces only a limited range of data samples. Additionally, GANs can be computationally intensive and require large amounts of training data.

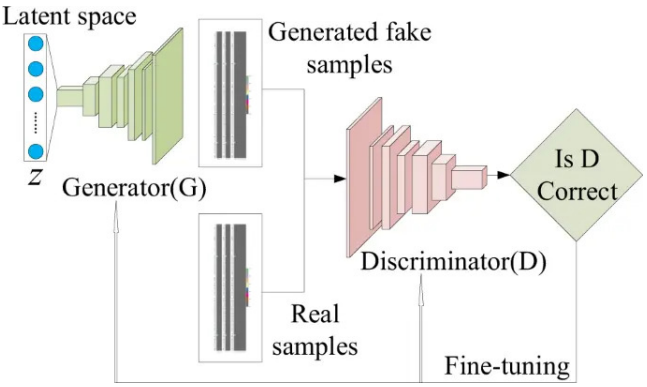

Figure 3. General architecture of GAN [16]

Generator Network. The generator network is a key component of a GAN model that is responsible for creating new data samples that resemble the training data. The generator network typically consists of multiple layers of neural network nodes, such as convolutional or dense layers, that transform the input noise into a higher-dimensional representation that can be reshaped into the desired output format, such as an image or a sequence of text. During the training process, the generator network's goal is to produce output that is similar enough to the real training data to fool the discriminator network.

Discriminator Network. The discriminator network is another key component of a GAN that is responsible for distinguishing between real training data and data generated by the generator network. The discriminator network takes an input data sample and produces an output that represents the probability that the input sample is real or fake. Similarly to the generator, it is built with multiple layers, usually of downsampling operations and convolutions in order to lower the dimensionality of the input data and a dense layer at the end for the classification.

Adversarial Training. The generator and discriminator networks are trained in an adversarial manner. The training process for the GAN starts with the generator network receiving random noise as input and generating a fake data sample. The discriminator network then receives both the real data sample from the training data and the fake data sample generated by the generator network as input and predicts whether each sample is real or fake.

The loss function for the discriminator network is typically a binary cross-entropy loss that measures the difference between the predicted and the actual label (1 for real data and 0 for fake data). The loss function for the generator network is typically based on the discriminator's output for the fake data sample and aims to minimize the difference between the predicted and the actual label (1 for real data). So, the general form of the loss function of a GAN is:

$$\mathcal{L}_{GAN} = \min_G \max_D \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (2)$$

The training process for a GAN can be challenging, as the two networks are constantly competing with each other. If the generator produces samples that are too easy to distinguish from the real data, the discriminator will learn to distinguish them quickly and the generator will need to produce better samples to continue improving. Conversely, if the generator produces samples that are too difficult to distinguish, the discriminator may not provide useful feedback to the generator.

3.3 Image-to-Image Translation with Conditional Adversarial Networks

Image-to-image translation is a computer vision task that involves converting an image from one domain to another domain, while preserving important characteristics of the input image. In recent years, deep learning models, such as GANs, have shown great success in performing image-to-image translation tasks.

One of the most promising approaches for image-to-image tasks is the Conditional Generative Adversarial Network model (cGAN) [12]. CGANs are a type of GANs that use additional input information, called conditioning information, to generate output that is based on it. The condition could be any additional input such as labels, images, text, etc. In a cGAN, the generator takes two inputs: a random noise vector and the condition. The discriminator takes both the generated output and the conditioning information as input. The loss of the cGAN is calculated as shown in equation 3

$$\mathcal{L}_{cGAN} = \min_G \max_D \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x,z)))] \quad (3)$$

Conditional adversarial networks have been investigated as a general-purpose solution to image-to-image translation problems [11]. As in the most common framework, the generator has an encoder-decoder architecture for processing the image condition and generating synthetic samples. The loss function of the network is also enhanced with a new component, the L1 loss between the generated image and the real image in order to encourage the generator to produce output that is close to the ground truth.

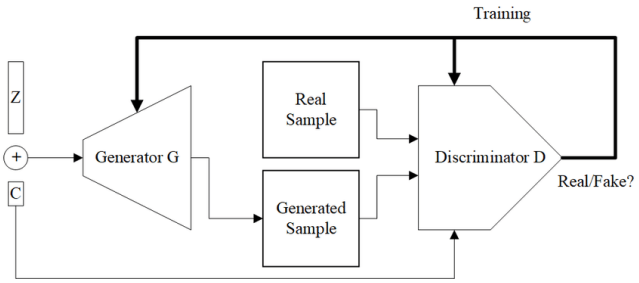

Figure 4. Inner workings of the conditional GAN. Z represents the noise vector and C the condition [17].

3.4 Transfer Learning

Although we have witnessed significant advances in the capabilities and the computational efficiency of deep learning models, a large portion of the cutting-edge techniques are still hard to handle or even remain inaccessible for some researches due to their immense scale. Hundreds of millions of trainable parameters and proportionally large collections of image data translate to the need for massive computational power, something that many research groups and individuals struggle to have at their dispose. Transfer learning is a training paradigm that has attenuated this limitation, to an extent. In principle, a machine learning model trained on certain domain for a particular task can be reused to tackle a different task, given that there is some common ground between them. It is extensively used throughout the Computer Vision field as the learnt low-level structural features of images are usually shared among different problems. For instance, a model that has gained the knowledge of learning to discriminate cats and dogs can be applied to the recognition of other species with little adaptation, as basic shapes and textures information of animals is already acquired by the model.

There are several benefits to applying transfer learning. For one, it can significantly reduce the time and computational resources required for training a model from scratch, while also boosting the performance of a model on a new task by providing a starting point with better initial weights than random initialization. It can also reduce the amount of data required to train a model on a new task, which is particularly useful when there is limited data available. In such cases, transfer learning helps in preventing overfitting, acting as a regularization method as well.

4 Methodology

In this section, the representation of images used, the pipeline of the system, the objective function, and the different components and methods implemented are introduced.

Our approach involves three major processes. First, we establish the color space that we will attempt to learn and apply some basic transformations to the input images in order to make them compatible with our models. Then, we experiment with two different backbone networks for the Generator of our cGAN models. The first is adopted by Isola *et al.* [11] and refers to a U-Net-style Generator that we will train from scratch. The second is about training an already pre-trained ResNet-18 model on ImageNet dataset [18], again following a U-Net-style architecture for generating synthetic images. Finally, we test both architectures on novel data that were not used during the training phase.

4.1 Image Representation

The choice of color space can have a significant impact on the performance of a network and the quality of the generated output. Therefore, we carefully choose the appropriate color space based on the characteristics of the input images, the desired output, and the capabilities of the models themselves.

Specifically, for our experiments, we adopt the CIELAB color space, also referred to as $L^*a^*b^*$. This color system expresses colors with three values, encompassing the entire gamut of human color perception. The L^* channel represents the *Lightness* of each pixel and appears as a grayscale image when visualized. The a^* channel represents the *green-red* color components, where negative values indicate greenish hues and positive values indicate reddish hues. Lastly, the b^* channel encodes the *blue-yellow* color components, where negative values represent bluish hues and positive values represent yellowish hues. Figure 5 illustrates the scaled $L^*a^*b^*$ color space along with each individual channel in an example color image.

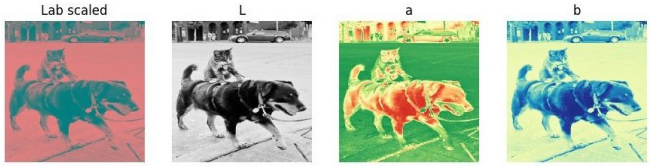

Figure 5. (From left to right) Scaled $L^*a^*b^*$ image, Lightness L^*, a^*, and b^* channels of the $L^*a^*b^*$ color space for an example color image.

The main rationale for using the $L^*a^*b^*$ color space instead of the commonly used *RGB* color space is as follows. When performing colorization using the $L^*a^*b^*$ color space, we only need to predict the a^* and b^* channels while using the L^* channel as input, resulting in a 2-channel prediction task. Conversely, for the *RGB* color space, we would need to predict all the three *RGB* channels, given a grayscale image as input, resulting in a 3-channel prediction task. Thus, by using the $L^*a^*b^*$ color space, we reduce the prediction task by one channel, which can lead to improved performance and efficiency in our colorization task.

Let I_L, I_a, and I_b denote the L^* , a^*, and b^* channels of an input color image, respectively. Then $I_L \in [0, 100]^{1\times W\times H}$, $I_a \in [-110, 110]^{1\times W\times H}$, and $I_b \in [-110, 110]^{1\times W\times H}$, with W and H be the width and height of the image, respectively. We normalize each channel to the range $[-1, 1]$ as follows:

$$I_L^{norm} = \frac{I_L}{50} - 1, \tag{4}$$

$$I_a^{norm} = \frac{I_a}{110}, \tag{5}$$

$$I_b^{norm} = \frac{I_b}{110}, \tag{6}$$

The preprocessed normalized channels $I_L^{norm}, I_a^{norm}, I_b^{norm} \in [-1, 1]^{1\times W\times H}$ are to be used for training the Generator and Discriminator networks. That is, I_L^{norm}, representing the grayscale image, will be the input to our Generator network. Then $I_{ab}^{norm} \in [-1, 1]^{2\times W\times H}$ indicates the concatenated a^* and b^* channels along the channel axis and represent the color space we want to learn, given as:

$$I_{ab}^{norm} = I_a^{norm} \bigoplus I_b^{norm}, \tag{7}$$

where $\bigoplus$ indicates the concatenation operation along the channels dimension. This will constitute the desired output of our Generator network. Lastly, we formulate the fully-color target image $I_{Lab} \in [-1, 1]^{3\times W\times H}$, which will also be used in the Discriminator network, as follows:

$$I_{Lab} = I_L^{norm} \bigoplus I_{ab}^{norm} \tag{8}$$

4.2 Networks architectures

We mainly adopt our Generator and Discriminator architectures from those in [11]. For the Generator, we experiment with two different backbone networks, whereas the Discriminator remains the same across both methods. All networks use modules of the form **convolution-BatchNorm-ReLU**. Details of the architectures are provided in the supplementary material at the end of the study, with the key components being directly discussed below.

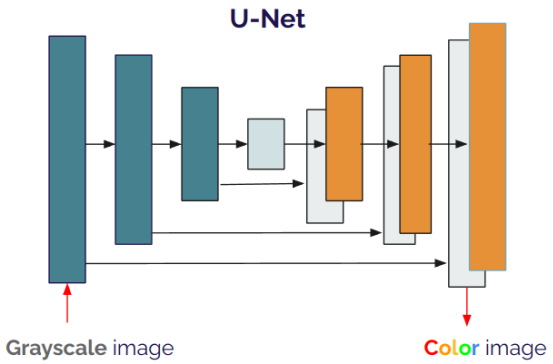

Figure 6. The "U-Net" [13] is an encoder-decoder network with skip connections between mirrored layers in the encoder and decoder stacks.

U-Net Generator with skip connections [11]. One of the key characteristics of image-to-image translation problems, including colorization, is that they map a high-resolution input grid to a high-resolution output grid. Following Isola *et al.* [11], we used a simple U-Net [13] style Generator of encoder-decoder with skip connections that concatenate each layer i of the encoder with each layer $n - i$ of the decoder, where n is the total number of layers. A simplified visualization of U-Net is shown in Figure 6. The input to the Generator network is a single channel L^* that corresponds to a grayscale image and the goal is to generate the color channels a^* and b^* of the image. The task can be specified as follows:

Given an input grayscale image $x \in [-1, 1]^{1\times W\times H}$ and its real color version $y \in [-1, 1]^{2\times W\times H}$, we want to learn a feature mapping $G : x \in [-1, 1]^{1\times W\times H} \longrightarrow y \in [-1, 1]^{2\times W\times H}$ such that

$$y \simeq G(x) \tag{9}$$

Following the notations of Section 4.1, we denote the output of the function G as I_{ab}^{Gen} where:

$$I_{ab}^{Gen} = G(I_L) \tag{10}$$

The Generator must be designed to warrant output $I_{ab}^{Gen} \in [-1, 1]$. To this end, we use the tangent hyperbolic activation function (Tanh) in the final layer of the Generator network. Eventually, the fully-color fake image produced by the Generator can be derived as:

$$\boxed{I_{Lab}^{Gen} = I_L \bigoplus I_{ab}^{Gen}} \tag{11}$$

One last thing that is also worth noting is about noise and randomness. Conventional conditional GANs have acknowledged the significance of adding random Gaussian noise z into the Generator in addition to the condition x, i.e., the grayscale image. This can also be demonstrated by examining their typical loss function in Equation 3. However, Isola *et al.* [11] chose a different way to introduce randomness to their final models after conducting several experiments. Specifically, they provide noise only in the form of dropout, applied on the first three layers of the decoder of their Generator during both training and testing. For our Generator model, we follow the same strategy with [11] and feed the network only with grayscale images.

Pre-trained ResNet Generator with skip connections. Transfer learning is commonly used to overcome the challenge of limited training data. A widely used approach leverages the ImageNet dataset [18], consisting of natural images, for pre-training. Motivated by the favorable outcomes of transfer learning in numerous contexts, we modify the Generator network from what was originally proposed by Isola *et al.* [11] and incorporate transfer learning into it. To be more precise, we utilize a pre-trained ResNet-18 [14] model that was first trained on ImageNet for the task of Image Classification with RGB images. By doing so, we anticipate that the pre-trained model has already learned rich representations from vast amounts of data, thereby providing a robust initialization that can assist our model in rapidly converging to a satisfactory solution for our target task.

To this end, we consider from ResNet-18 only the layers that can aid our task, specifically, all the layers excluding the fully-connected ones. We employ this pre-trained model as the encoder stage and integrate it into a U-Net-style architecture along with the corresponding decoder of upsampling operations and with skip connections that can facilitate the flow of information between them.

Another alteration that we made in this case pertains to the dimensions of the input images. Considering that we are using a pre-trained model on RGB images, it is necessary to use 3-dimensional inputs to our modified Generator as well. For this reason, the L^* channel is replicated three times and stacked along the channel dimension in the following manner:

$$I_{3L}^{norm} = I_L^{norm} \bigoplus I_L^{norm} \bigoplus I_L^{norm}, \tag{12}$$

where $I_L^{norm} \in [-1, 1]^{1\times W\times H}$ is the normalized grayscale image (ref to Section 4.1) and $I_{3L}^{norm} \in [-1, 1]^{3\times W\times H}$ is the updated grayscale input to the modified Generator.

Patch Discriminator [11]. The Discriminator network that we use is identical to the one suggested by Isola *et al.* in [11] and operates on the patch level of images. That means that it tries to classify if each $N \times N$ patch in an image is real or fake, where the patch size N is set to 70. The Discriminator runs convolutionally across the image, averaging all responses to provide the ultimate output of D. The task of the Discriminator can be then defined as follows:

Given a color input image $x \in [-1, 1]^{3\times W\times H}$ and a true label $y \in \{0, 1\}$ of whether the image is fake or real, respectively, we want to learn a mapping function $D : x \in [-1, 1]^{3\times W\times H} \longrightarrow y \in \{0, 1\}^{1\times W_p\times H_p}$ such that

$$y = D(x), \tag{13}$$

where $W_p \times H_p$ represents the number of $N \times N$ patches in the image.

4.3 Objective function

The main learning objective for a conditional GAN is for the Generator to generate realistic color images that the Discriminator will not be able to differentiate from the real ones in an adversarial setting. In this study, the loss function used for training the Generator and Discriminator networks is the same as the one proposed by Isola *et al.* in [11].

Specifically, the Discriminator network tries to classify the real images as real (1 for real) and the corresponding fake images produced by the Generator as fake (0 for fake). Equation 14 below shows the loss component used to train the Discriminator to correctly identify a real L^* a^* b^* image:

$$\mathcal{L}_D^{real} = \mathcal{L}_{BCE}\Big(\{1\}^{1\times H_p\times W_p}, D(I_{Lab}^{real})\Big) \tag{14}$$

Equation 15 below shows the loss component used to train the Discriminator to correctly identify a fake generated L^* a^* b^* image from the Generator:

$$\mathcal{L}_D^{fake} = \mathcal{L}_{BCE}\Big(\{0\}^{1\times H_p\times W_p}, D(I_{Lab}^{fake})\Big) \tag{15}$$

The combined final loss used to train the Discriminator is presented in Equation 16:

$$\boxed{\mathcal{L}_D = \frac{\mathcal{L}_D^{real} + \mathcal{L}_D^{fake}}{2}} \tag{16}$$

On the other hand, the Generator tries to generate fake images that will be classified as real by the Discriminator. This can be formulated by a loss component as shown in Equation 17:

$$\mathcal{L}_G^{BCE} = \mathcal{L}_{BCE}\Big(\{1\}^{1\times H_p\times W_p}, D(I_{Lab}^{fake})\Big) \tag{17}$$

Following Isola *et al.* in [11], we also incorporate the L1 loss function that measures the difference between a real and a fake generated image as follows:

$$\mathcal{L}_G^{L1} = L_1\Big(D(I_{ab}^{real}), D(I_{ab}^{fake})\Big) \tag{18}$$

Thus, the total loss for the Generator is given by:

$$\boxed{\mathcal{L}_G = \mathcal{L}_G^{BCE} + \lambda\, \mathcal{L}_G^{L1}} \tag{19}$$

5 Experimental setup

5.1 Datasets

Training. For training our CGANs models, we use the Microsoft Common Objects in Context dataset, widely known as the MS COCO dataset [19]. From the entire training set of COCO consisting of $83K$ color images we use 8,000 random samples (~9.6%) and apply some further transformations to these input images. Specifically, we resize the variable-sized images to a fixed size of 256×256 ($W = H = 256$) in order to fit the input of our networks. In addition, we augment our training set by applying random horizontal flip with probability 0.5 as proposed by Isola *et al.* in [11].

Test. For testing the models, we use a subset from the test set of COCO consisting of 2,000 images (~5% of the total set). We apply the same transformations as in the training set but without augmentation.

5.2 Training details

Following the setup suggested by Isola *et al.* [11], we set $W = H = 256$ and resize all training and test images to 256×256. For training the Generator and Discriminator networks, we use two independent Adam optimizers, with a learning rate of 0.0002 and momentum parameters $\beta_1 = 0.5$, $\beta_2 = 0.999$, since these networks are trained separately in consecutive steps in an epoch.

The U-Net Generator and Patch Discriminator networks are trained from scratch. Weights are initialized from a Gaussian distribution with mean 0 and standard deviation 0.02. In turn, the weights of the decoder of ResNet-18 Generator are initialized from the same Gaussian distribution, while the weights of the corresponding encoder are already pre-trained on ImageNet. Finally, in Equation x, we set $\lambda = 100$.

Using this protocol, the models are trained on a subset of the MS COCO dataset with a batch size of 16 for up to 300 epochs, which is a total of $150K$ iterations. All experiments

are conducted on a single NVIDIA A100 GPU in a runtime of ~4 hours.

6 Results

Here we show both versions of the cGAN in action. In Figure 7 we present how the two models colorize some samples taken from the test set. In the first column we can see the original image, in the second column the grayscale version, in the third column the output of the UNet based generator and in the last column the output of the ResNet based generator.

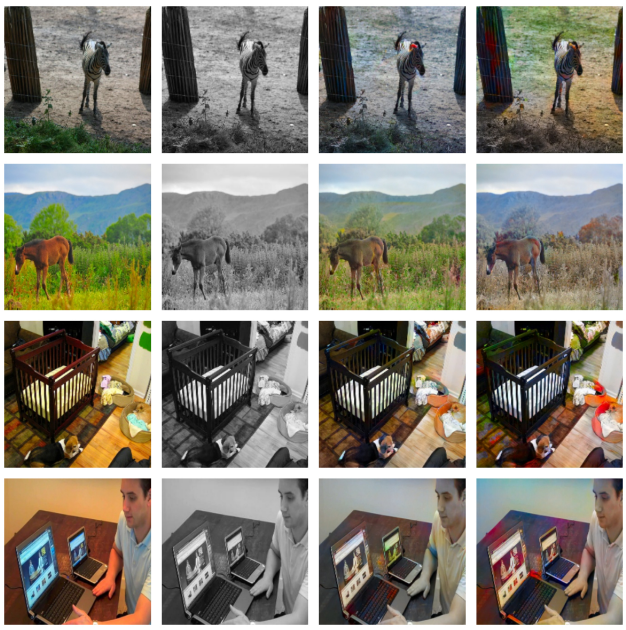

Figure 7. Comparison of the generated images. From left to right: Original images, grayscale versions, images generated by the UNet based cGAN and finally images generated by the ResNet based cGAN.

To gain an insight on how the training evolved over time, we collected the loss metrics over all the epochs. In Figure 8 we show the BCELoss of the discriminator (scaled by 0.1) and the generator networks of the ResNet based GAN over epochs, as well as the L1 loss of the generator. We present the same metrics for the UNet based GAN in Figure 9. On the test set, the L1 loss of the ResNet GAN is 0.0945, while for the UNet GAN it is 0.0917.

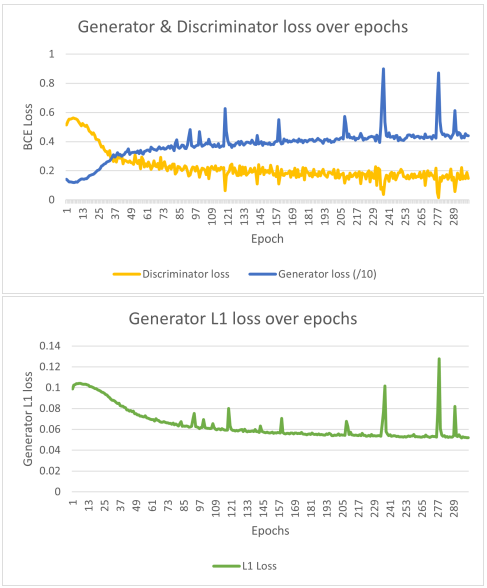

Figure 8. Top: BCELoss of the generator (scaled by 0.1) and discriminator networks of the ResNet based GAN over epochs. Bottom: L1 loss of the generator over epochs.

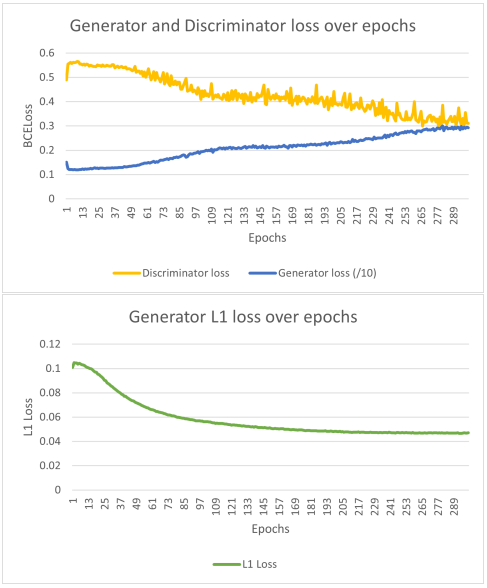

Figure 9. Top: BCELoss of the generator (scaled by 0.1) and the discriminator networks of the UNet based GAN over epochs. Bottom: L1 loss of the generator over epochs.

7 Discussion

The first thing an observer notices from the generated images as displayed in Figure 7 is that both models create some artifacts in their output images. Additionally, the two models choose a different palette of images, and even though the

generated image is not always very close to the original one, in certain cases they look realistic enough.

From the visualizations of the BCE losses over time in Figures 8 and 9 we see the adversarial training process in action: the loss of the discriminator grows or shrinks inversely to the loss of the generator, with the two networks eventually reaching an equilibrium and the losses converging. The plots of the L1 losses over the epochs suggests that the generator gets better at estimating the ground truth in the first 100 epochs, and after that progress slows down significantly.

We expected the ResNet based cGAN to perform better than the UNet based cGAN, since the pre-training on the ImageNet dataset should theoretically give the former an edge. However, the test set L1 losses suggest the opposite. We can attribute this to the much smaller number of parameters of the generator networks: the UNet one has 54 million parameters while the ResNet one only has 15 million. Additionally, the ResNet network was trained on the task of image classification, so while it was able to capture certain high level features in images, perhaps not enough of them translate well to the task of image colorization.

Despite the above, we can still argue that the pre-training helped the ResNet version of the cGAN. We base this belief on the observation that even with so much fewer parameters, it is able to match the performance of the UNet version and produce images of similar quality. Additionally, by comparing the progression of the BCE losses between the two, we see that for ResNet they converge faster, which is an indication towards the idea that the initialized weights are near enough to a good solution already.

8 Conclusion

In this project we implemented the ideas presented in the paper by Isola et al. [11], applied the proposed framework to the task of image colorization and introduced a small modification to it. Due to the smaller scope of our undertaking, our results are not at the level of the paper that inspired it, but still confirm the idea that conditional GANs can be successfully used for the purpose of generating color to a grayscale image.

Acknowledgments

We thank the Center for Information Technology of the University of Groningen for their support and for providing access to the Hábrók high performance computing cluster.

References

[1] T. Welsh, M. Ashikhmin, and K. Mueller, "Transferring color to greyscale images," *ACM Trans. Graph.*, vol. 21, p. 277–280, jul 2002.

[2] A. Levin, D. Lischinski, and Y. Weiss, "Colorization using optimization," *ACM Transactions on Graphics*, vol. 23, 06 2004.

[3] G. Charpiat, M. Hofmann, and B. Schölkopf, "Automatic image colorization via multimodal predictions," vol. 2008, 09 2008.

[4] Y. Morimoto, Y. Taguchi, and T. Naemura, "Automatic colorization of grayscale images using multiple images on the web," 08 2009.

[5] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be color! joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification," *ACM Trans. Graph.*, vol. 35, jul 2016.

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

[7] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning representations for automatic colorization," 2017.

[8] Y. Xiao, A. Jiang, C. Liu, and M. Wang, "Single image colorization via modified cyclegan," in *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 3247–3251, 2019.

[9] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[10] A. Deshpande, J. Lu, M.-C. Yeh, M. J. Chong, and D. Forsyth, "Learning diverse image colorization," 2017.

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

[12] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

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

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

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

[16] Y. Dan, Y. Zhao, X. Li, S. Li, M. Hu, and J. Hu, "Generative adversarial networks (gan) based efficient sampling of chemical composition space for inverse design of inorganic materials," *npj Computational Materials*, vol. 6, no. 1, p. 84, 2020.

[17] C. Ongun and A. Temizel, "Paired 3d model generation with conditional generative adversarial networks," 08 2018.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009.

[19] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pp. 740–755, Springer, 2014.

[20] "Colorizing black white images with u-net and conditional gan — a tutorial." https://towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8.

A Network architectures

We adopt our basic network architectures from those in [11] and the modified architectures partially from [20].

Let *Ck* denote a Convolution-BatchNorm-ReLU layer with *k* filters. *CDk* denotes a Convolution-BatchNorm-Dropout-ReLU layer with a dropout rate of 50%. All convolutions are 4×4 spatial kernels applied with stride 2 and padding 1. Convolutions in the encoder, and in the discriminator, downsample by a factor of 2, whereas in the decoder they upsample by a factor of 2.

A.1 U-Net Generator architecture [11]

The encoder-decoder architecture of U-Net consists of:

Encoder:
C64-C128-C256-C512-C512-C512-C512-C512

Decoder:
CD512-CD1024-CD1024-C1024-C1024-C512-C256-C128

A convolution is applied to the output of the final decoder layer to map it to the desired number of output channels, which is usually 3, but in colorization it is 2 (a^* and b^* color channels). The output of the convolution is then passed through a Tanh activation function. The first *C*64 layer in the encoder is an exception, where BatchNorm is not applied. Leaky ReLUs with a slope of 0.2 are used in all encoder layers, while simple ReLUs without slope are applied in the decoder. Also, the skip connections concatenate activations from layer i to layer $n - i$.

Figures 10 and 11 present the output shape of each layer of the encoder and decoder networks of the U-Net Generator, respectively, along with the trainable parameters, given as input a grayscale image of size $1 \times 256 \times 256$.

A.2 Pre-trained ResNet-18 Generator architecture [20]

The encoder-decoder architecture of ResNet-18 consists of:

Encoder:
C64-C64-C64-C128-C128-C256-C256-C512-C512

Decoder:
C256-C512-C256-C128-C128

A convolution is applied to the output of the final decoder layer to map it to the desired number of output channels, which is again 2 (a^* and b^* color channels). The output of the convolution is then passed through a Tanh activation function. ReLUs without slope are used in all layers of both the encoder and decoder networks. Also, the skip connections concatenate activations from layer i to layer $n - i$.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─Conv2d: 1-1                            [-1, 64, 128, 128]        1,024
├─Sequential: 1-2                        [-1, 128, 64, 64]         --
|    └─LeakyReLU: 2-1                    [-1, 64, 128, 128]        --
|    └─Conv2d: 2-2                       [-1, 128, 64, 64]         131,072
|    └─BatchNorm2d: 2-3                  [-1, 128, 64, 64]         256
├─Sequential: 1-3                        [-1, 256, 32, 32]         --
|    └─LeakyReLU: 2-4                    [-1, 128, 64, 64]         --
|    └─Conv2d: 2-5                       [-1, 256, 32, 32]         524,288
|    └─BatchNorm2d: 2-6                  [-1, 256, 32, 32]         512
├─Sequential: 1-4                        [-1, 512, 16, 16]         --
|    └─LeakyReLU: 2-7                    [-1, 256, 32, 32]         --
|    └─Conv2d: 2-8                       [-1, 512, 16, 16]         2,097,152
|    └─BatchNorm2d: 2-9                  [-1, 512, 16, 16]         1,024
├─Sequential: 1-5                        [-1, 512, 8, 8]           --
|    └─LeakyReLU: 2-10                   [-1, 512, 16, 16]         --
|    └─Conv2d: 2-11                      [-1, 512, 8, 8]           4,194,304
|    └─BatchNorm2d: 2-12                 [-1, 512, 8, 8]           1,024
├─Sequential: 1-6                        [-1, 512, 4, 4]           --
|    └─LeakyReLU: 2-13                   [-1, 512, 8, 8]           --
|    └─Conv2d: 2-14                      [-1, 512, 4, 4]           4,194,304
|    └─BatchNorm2d: 2-15                 [-1, 512, 4, 4]           1,024
├─Sequential: 1-7                        [-1, 512, 2, 2]           --
|    └─LeakyReLU: 2-16                   [-1, 512, 4, 4]           --
|    └─Conv2d: 2-17                      [-1, 512, 2, 2]           4,194,304
|    └─BatchNorm2d: 2-18                 [-1, 512, 2, 2]           1,024
├─Conv2d: 1-8                            [-1, 512, 1, 1]           4,194,816
==========================================================================
Total params: 19,536,128
Trainable params: 19,536,128
```

Figure 10. Summary of the encoder of the U-Net Generator given an input grayscale image of size $1 \times 256 \times 256$.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─Sequential: 1-1                        [-1, 512, 2, 2]           --
|    └─ReLU: 2-1                         [-1, 512, 1, 1]           --
|    └─ConvTranspose2d: 2-2              [-1, 512, 2, 2]           4,194,304
|    └─BatchNorm2d: 2-3                  [-1, 512, 2, 2]           1,024
|    └─Dropout: 2-4                      [-1, 512, 2, 2]           --
├─Sequential: 1-2                        [-1, 512, 4, 4]           --
|    └─ReLU: 2-5                         [-1, 1024, 2, 2]          --
|    └─ConvTranspose2d: 2-6              [-1, 512, 4, 4]           8,388,608
|    └─BatchNorm2d: 2-7                  [-1, 512, 4, 4]           1,024
|    └─Dropout: 2-8                      [-1, 512, 4, 4]           --
├─Sequential: 1-3                        [-1, 512, 8, 8]           --
|    └─ReLU: 2-9                         [-1, 1024, 4, 4]          --
|    └─ConvTranspose2d: 2-10             [-1, 512, 8, 8]           8,388,608
|    └─BatchNorm2d: 2-11                 [-1, 512, 8, 8]           1,024
|    └─Dropout: 2-12                     [-1, 512, 8, 8]           --
├─Sequential: 1-4                        [-1, 512, 16, 16]         --
|    └─ReLU: 2-13                        [-1, 1024, 8, 8]          --
|    └─ConvTranspose2d: 2-14             [-1, 512, 16, 16]         8,388,608
|    └─BatchNorm2d: 2-15                 [-1, 512, 16, 16]         1,024
├─Sequential: 1-5                        [-1, 256, 32, 32]         --
|    └─ReLU: 2-16                        [-1, 1024, 16, 16]        --
|    └─ConvTranspose2d: 2-17             [-1, 256, 32, 32]         4,194,304
|    └─BatchNorm2d: 2-18                 [-1, 256, 32, 32]         512
├─Sequential: 1-6                        [-1, 128, 64, 64]         --
|    └─ReLU: 2-19                        [-1, 512, 32, 32]         --
|    └─ConvTranspose2d: 2-20             [-1, 128, 64, 64]         1,048,576
|    └─BatchNorm2d: 2-21                 [-1, 128, 64, 64]         256
├─Sequential: 1-7                        [-1, 64, 128, 128]        --
|    └─ReLU: 2-22                        [-1, 256, 64, 64]         --
|    └─ConvTranspose2d: 2-23             [-1, 64, 128, 128]        262,144
|    └─BatchNorm2d: 2-24                 [-1, 64, 128, 128]        128
├─ConvTranspose2d: 1-8                   [-1, 2, 256, 256]         4,098
==========================================================================
Total params: 34,874,242
Trainable params: 34,874,242
```

Figure 11. Summary of the decoder of the U-Net Generator given the output of the encoder of size $512 \times 1 \times 1$.

Figures 12 and 13 present the output shape of each layer of the encoder and decoder networks of the pre-trained ResNet-18 Generator, respectively, along with the trainable parameters, given as input a grayscale image of size $3 \times 256 \times 256$.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─ResNet: 1                              []                        --
|    └─Conv2d: 2-1                       [-1, 64, 128, 128]        9,408
|    └─BatchNorm2d: 2-2                  [-1, 64, 128, 128]        128
|    └─ReLU: 2-3                         [-1, 64, 128, 128]        --
|    └─MaxPool2d: 2-4                    [-1, 64, 64, 64]          --
|    └─Sequential: 2-5                   [-1, 64, 64, 64]          --
|    |    └─BasicBlock: 3-1              [-1, 64, 64, 64]          73,984
|    |    └─BasicBlock: 3-2              [-1, 64, 64, 64]          73,984
|    └─Sequential: 2-6                   [-1, 128, 32, 32]         --
|    |    └─BasicBlock: 3-3              [-1, 128, 32, 32]         230,144
|    |    └─BasicBlock: 3-4              [-1, 128, 32, 32]         295,424
|    └─Sequential: 2-7                   [-1, 256, 16, 16]         --
|    |    └─BasicBlock: 3-5              [-1, 256, 16, 16]         919,040
|    |    └─BasicBlock: 3-6              [-1, 256, 16, 16]         1,180,672
|    └─Sequential: 2-8                   [-1, 512, 8, 8]           --
|    |    └─BasicBlock: 3-7              [-1, 512, 8, 8]           3,673,088
|    |    └─BasicBlock: 3-8              [-1, 512, 8, 8]           4,720,640
==========================================================================
Total params: 11,176,512
Trainable params: 11,176,512
```

Figure 12. Summary of the encoder of the ResNet-18 Generator given an input grayscale image of size $3 \times 256 \times 256$.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─Sequential: 1-1                        [-1, 256, 16, 16]         --
|    └─ReLU: 2-1                         [-1, 512, 8, 8]           --
|    └─ConvTranspose2d: 2-2              [-1, 256, 16, 16]         2,097,152
|    └─BatchNorm2d: 2-3                  [-1, 256, 16, 16]         512
├─Sequential: 1-2                        [-1, 128, 32, 32]         --
|    └─ReLU: 2-4                         [-1, 512, 16, 16]         --
|    └─ConvTranspose2d: 2-5              [-1, 128, 32, 32]         1,048,576
|    └─BatchNorm2d: 2-6                  [-1, 128, 32, 32]         256
├─Sequential: 1-3                        [-1, 64, 64, 64]          --
|    └─ReLU: 2-7                         [-1, 256, 32, 32]         --
|    └─ConvTranspose2d: 2-8              [-1, 64, 64, 64]          262,144
|    └─BatchNorm2d: 2-9                  [-1, 64, 64, 64]          128
├─Sequential: 1-4                        [-1, 64, 128, 128]        --
|    └─ReLU: 2-10                        [-1, 128, 64, 64]         --
|    └─ConvTranspose2d: 2-11             [-1, 64, 128, 128]        131,072
|    └─BatchNorm2d: 2-12                 [-1, 64, 128, 128]        128
├─ConvTranspose2d: 1-5                   [-1, 2, 256, 256]         4,098
==========================================================================
Total params: 3,544,066
Trainable params: 3,544,066
```

Figure 13. Summary of the decoder of the ResNet-18 Generator given the output of the encoder of size $512 \times 8 \times 8$.

A.3 Discriminator architecture

The 70×70 Patch Discriminator architecture is:
C64-C128-C256-C512

After the final $C512$ layer, a convolution is applied to map to a 1-dimensional output, followed by a Sigmoid activation function. As an exception to the provided notation, BatchNorm is not applied to the first $C64$ layer. Also, all ReLUs in the Discriminator are leaky with slope 0.2.

Figure 14 illustrates the architecture of the Discriminator network. Figure 15 presents the output shape of each layer of the Discriminator along with the trainable parameters, given as input a color image of size $3 \times 256 \times 256$ and patch size $N = 70$.

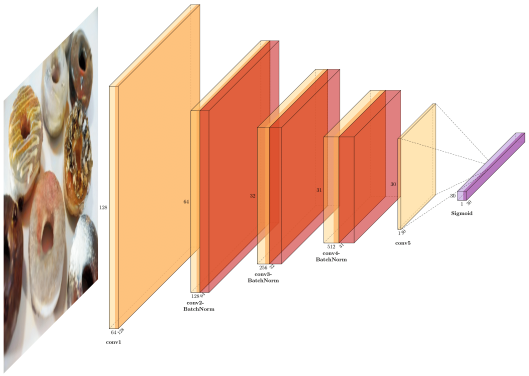

Figure 14. Discriminator network architecture.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
├─Conv2d: 1-1                            [-1, 64, 128, 128]        3,136
├─Conv2d: 1-2                            [-1, 128, 64, 64]         131,072
├─BatchNorm2d: 1-3                       [-1, 128, 64, 64]         256
├─Conv2d: 1-4                            [-1, 256, 32, 32]         524,288
├─BatchNorm2d: 1-5                       [-1, 256, 32, 32]         512
├─Conv2d: 1-6                            [-1, 512, 31, 31]         2,097,152
├─BatchNorm2d: 1-7                       [-1, 512, 31, 31]         1,024
├─Conv2d: 1-8                            [-1, 1, 30, 30]           8,193
==========================================================================
Total params: 2,765,633
Trainable params: 2,765,633
```

Figure 15. Summary of the Discriminator given an input color image of size $3 \times 256 \times 256$ and patch size $N = 70$.