Simpsons Image Colorization using cGAN and PatchGAN

Alexandra Hotti Jacob Malmberg Marcus Nystad Öhman

Abstract

Colorizing gray scale images is a problem which can be solved via hand or machine. In recent years, Conditional Generative Adversarial Networks (cGANs) have been used to automatically solve this problem. In this paper, a cGAN is contrasted against an extended version called PatchGAN, a variation of a Generative Adversarial Network, on a novel dataset containing Simpsons images. Multiple networks were trained in RGB space. The results indicate that the PatchGAN network both converges and colorizes faster than the regular GANs.

1. Introduction

Automatic image colorization is a broad topic with multiple possible fields of application. Ranging from colorization of old black and white pictures and movies, image augmentation and colorization of medical images for better visualisation.

In this study several cGANs were used to colorize gray-scaled images from 17 seasons of The Simpsons. Different architectures were explored to find a suitable structure for the given dataset. In particular several different U-Net based generators with and without skip connections in the RGB color space were tried. Two different extensions of cGANs using PatchGAN and skip connections as well as a U-net generator were implemented.

The PatchGAN architecture outperformed the normal cGAN network. Skip connections appears to help the network to preserve structural information during the up-sampling in the generator. The results are good, but not as good as hand colorized images.

2. Problem statement and motivation

This report aims to successfully colorize gray-scale images from The Simpsons TV-series using cGANs and PatchGAN. Finding a good architecture that colorize gray-scale images well could be of use in other applications of colorization, such as those mentioned in section 1.

3. Related Work

3.1. Semi-Automatic Colorization

Hint-based colorization are methods which are used for colorization of gray scale images and requires human input or supervision. One such popular method is a scribble-based method by Levin et al. (2004). The method colorizes based on human scribbles on top of gray-scale images, were the method colors a given area with the scribbles color-tone via convex optimization. The method assumes that neighboring pixels in space and time that have similar intensities and should thereby have similar coloring.

Another semi-automatic technique for colorization of grayscale images was proposed by Ashikhmin et al. (2002). The method involves coloring gray images by transferring color from a colored reference image. It does so by inspecting the luminance values of pixels within a vicinity of each pixel in the target image and then transferring the color pixels with matching neighborhoods in the reference image. This method performs well given that the input images distinctly different colored region produces distinct luminance clusters or contains dissimilar textures.

3.2. Automated Colorization

Regardless of the qualitative colorized images created by the semi-automatic algorithms the downside of these methods is that they require an immense amount of human input or supervision. Instead there are automatized colorization image processing methods which are based on features such as color histograms (Hafner et al., 1995) and HoG (Dalal & Triggs, 2005).

However, as deep learning methods have gained popularity in recent years feature based methods have to a large extent been replaced. Efros et al. (2016) proposed a Convolutional Neural Network for colorizing gray scale images. This method effectively produces vibrant and realistic colorization.

3.3. Generative Adversarial Networks used in colorization

Generative Adversarial Networks were introduced by Goodfellow et al. (2014) to produce images form noise. Since

then, several extensions of the algorithm has been proposed, including cGANs (Mirza & Osindero, 2014) and Pix2Pix (Isola et al., 2017). In cGANs the network is given some meaningful information to condition on when generating its' images, as opposed to noise as in the original method. Lastly, Pix2Pix is a version of cGAN where the network utilizes images as the condition. Thus, it can be used for image-image translation problems such as colorization.

4. Theory

4.1. Generative Adversarial Networks (GANs)

Generative adversarial networks (Goodfellow et al., 2014) were suggested as a new framework for estimating generative models via an adversarial process. A GAN is comprised of two models, a generative models (G) that tries to learn the underlying data distribution and a discriminative model (D) which estimates the probability that a given sample comes from the real training data rather than the model distribution. These two models are trained simultaneously competing against each other in a mini-max game in order to optimize the value function V(G, D):

$$\min_G \max_D V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[log(1-D(G(\mathbf{z})))] \quad (1)$$

4.2. Conditional Generative Adversarial Networks (cGANs)

The conditional Generative adversarial network is an extension of the regular GAN and was first proposed by Mirza and Osindero (2014). A cGAN is a GAN that is conditioned on some added information such as class label or even data that comes from different modalities. This may direct the generation process to make training easier than without conditioning on extra information. The value function in the mini-max game for a cGAN can bee seen in equation 4.2 below.

$$\min_G \max_D V(G,D) = \mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}[log(D(\mathbf{x}|\mathbf{y}))] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[log(1-D(G(\mathbf{z}|\mathbf{y})))]. \quad (2)$$

4.3. PatchGAN

4.3.1. AVERAGING PATCHGAN

PatchGAN is an architecture where the discriminator focuses on local image patches in the case of image colorization presented by Isola et al. (2017). This architecture leads to a discriminator that only penalizes structures at the scale of patches. Instead of classifying whether or not the whole image is real or fake, the discriminator in PatchGAN attempts to classify each NxN patch as either fake or real. The outputs for each patch is then averaged to yield the final output of the discriminator, resulting in a single valued scalar between 0-1 indicating whether the image is real or not.

4.3.2. SOFTMIN/SOFTMAX PATCHGAN

Introduced here is a novelty feature which in this study is added to the original PatchGAN presented in the paper by Isola et al. (2017). Instead of computing the average over all the patches each patch is weighted according to their proximity to the lowest patch likelihood and then summed to receive a final softmin or softmax likelihood for the image being real or fake (Versteegen et al., 2016). Thus, variations in the image can be taken into account if some parts of the image has low probability. Sofmin and softmax is used as smoothed differentiable version of the min and max objective functions. The softmin approach can be found in equation 3, where $\boldsymbol{\ell}$ is a vector of patch-region likelihoods, w is the weight-matrix, $\boldsymbol{\alpha}$ is set to -1 to compute softmin and 1 to compute softmax.

$$\mathbf{w} = \frac{\exp(\alpha * \log \boldsymbol{\ell})}{\sum_i \exp(\alpha * \log \boldsymbol{\ell_i})}$$
$$S = \sum_j (\log \boldsymbol{\ell_j} * \mathbf{w_j}) \quad (3)$$

4.4. Skip connections

Skip connections was first introduced by Srivastava et al. (2015) in a method called Highway networks . The technique is used as an architectural feature in CNNs where feature maps from different layers are stacked upon each other such that information from earlier layers can be incorporated into the learning phase of layers further down in the architecture. Due to the down-sampling in CNNs information from the earliest layers become abstract further down the network, leading to loss of previous information. By using a skip connection architecture this information is preserved into later layers (Orhan & Pitkow, 2018).

5. Data

The data consisted of 36 000 images split into a trainingset of 28 000 images, a validationset of 7 000 images and a testset of 1 000 images. The images were taken from season 10-27 of The Simpsons. From these episodes, one image was taken every 15 seconds, excluding the first and last minute to avoid capturing the intro and outro multiple times. The images were down-sampled to a resolution of 256x256 and transformed to gray scale using the imagemagick and ffmpeg software suites. This resolution was chosen since it enables large batches. An example picture and its' gray scale version is displayed in Figure 1.

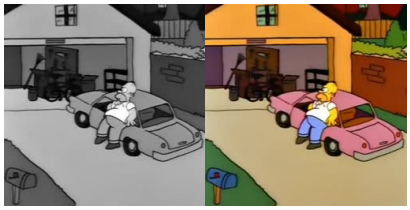

Figure 1. Example datapoint

6. Method

Both Conditional GANs and PatchGAN architectures are used and their architectures are in section 6.2 and section 6.3 respectively. Parameters common to both architectures are described in section 6.1. A high-level summary of the hyper parameters for the different networks are shown in Table 1. The training is described in section 6.4 and the setup for the experiments is outlined in section 6.5.

6.1. Common parameters

All models operate in the RGB color space. The generators used follow the U-Net architecture originally presented in (Ronneberger et al., 2015). They consist of an encoder and a decoder. The encoder encodes the high-dimensional input image into a low-dimensional representation. The decoder then decodes this representation into the final high-dimensional output image. This architecture was selected as it was successfully used by Nazeri et al. (2018). According to Isola et al. (2017), skip-connections improve the quality of the final output and was therefore used between mirrored layers in the encoder and decoder. In the networks were skip-connections were not used, network V1 & V3, they were omitted to see if a difference could be noted.

While GANs are notoriously hard to train (Salimans et al., 2016), some techniques have been proven to stabilize the training (Radford et al., 2015). First, batch normalization was used in both the generator and discriminator. Also, instead of using pooling layers, strided convolutions were used in the discriminator. Lastly, in the generator, ReLU was used as the activation function in all layers except the output layer, where Tanh was utilised. In the discriminator, LeakyReLU with slope 0.2 was used.

The Adam optimization algorithm was implemented with the momentum term β_1 set to 0.5 as suggested in (Salimans et al., 2016). When SGD was used, momentum was set to 0.9. To effectively train the discriminator, a method introduced by Salimans et al. (2016) named 'one-sided label smoothing' was used. Which is a technique were instead of using 1 as the label for real data discriminator input, a value between [0.8, 1.0] is randomly sampled. The learning rate was the same in both in the generator and discriminator for all networks.

In order to create images close to ground truth, a L_1 regularization term was added to the generator loss. The weight of this regularization term, λ, was set to 0.392. This technique was successfully used in (Isola et al., 2017).

6.2. Conditional GANs

Three conditional GANs were trained. The first, V1, has a structure similar to the one used in (Nazeri et al., 2018), except that it does not have skip connections. The filter size in this generator is consistently 3x3, irrespective of layer. It has 10 layers in the generator. The discriminator has 8 filters, where the first 5 filters are 3x3, the sixth is 5x5, the seventh is 1x1, and the eighth is 4x4.

The second network, V2, has the same amount of layers as network V1 but uses skip connections. The generator architecture is displayed in Figure 3 and the discriminator architecture is displayed in Figure 2. The filter size is not constant but instead has a size corresponding to the size of the input image to the layer. The first layer has a filter size of 6x6 which tapers off to 2x2 in the encoder part of

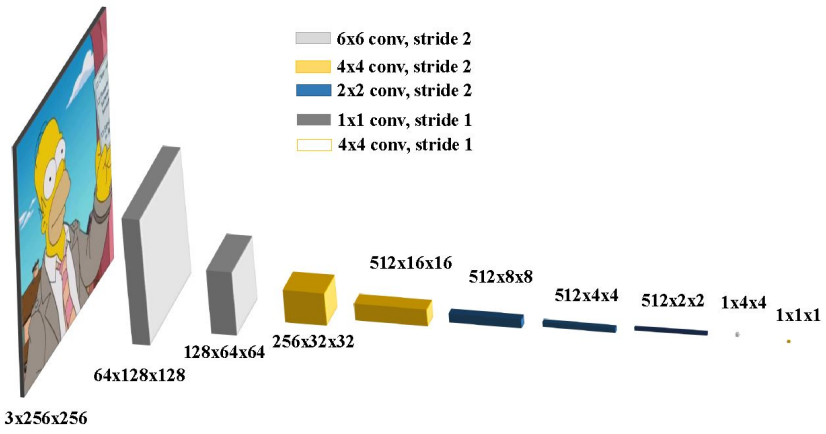

Figure 2. Discriminator architecture for Network 2.

Table 1. Network hyper parameters

Version	Skip conn.	PatchGAN	Filter size	Optimizer G	Optimizer D	Learning rate
V1	No	No	4x4	SGD	ADAM	2e-4
V2	Yes	No	Pyramid (6x6)	ADAM	SGD w/ decay	4e-4, sched.
V3	No	No	Pyramid (8x8)	ADAM	ADAM	2e-4
Avg-PatchGAN	Yes	Yes	Pyramid (6x6)	ADAM	SGD w/ decay	4e-4 sched.
Min-PatchGAN	Yes	Yes	Pyramid (6x6)	ADAM	SGD w/ decay	4e-4 sched.

the generator. In the decoder part of the network the filter sizes go from 2x2 to 6x6 in the last layer. The discriminator similarly has a 6x6 filter size in the first layer and tapers off to 1x1 in the last layer. This innovative architecture where the filter size corresponds to the size of the input image to the layer is referred to as "pyramid". The reasoning behind it is that larger images should use larger filters since the features, such as Homer's head, are large. As the image size is decreased, the size of the features are decreased and the filter size should thus be decreased. L2 regularization in the discriminator was used to decrease overfitting, with the penalty set to 1e-4.

The third network, V3, has a pyramid architecture but with different filter sizes compared to network 2. In the generator, the filters are 8x8 in the input layer and taper off similarly to network 2 to a size of 2x2 in the encoding part and in the decoder part the filters grow from 2x2 to 8x8. Also, Adam optimization was used in both the generator and the discriminator, similarly to the architecture used in (Radford et al., 2015).

6.3. PatchGANs

Two different versions of PatchGAN networks were constructed. They share a discriminator architecture, which outputs 1x32x32 'patches'. This discriminator architecture is displayed in Figure 4. The size of the output patches was chosen so that each patch in the output corresponds to a 8x8 patch in the input image. The generator for the PatchGAN networks are identical to the the one used in the previous section and is displayed in Figure 3. The learning rate for these networks was set to 4e-4 and learning rate scheduling was used to decay the learning rate by 0.5 every 5th epoch.

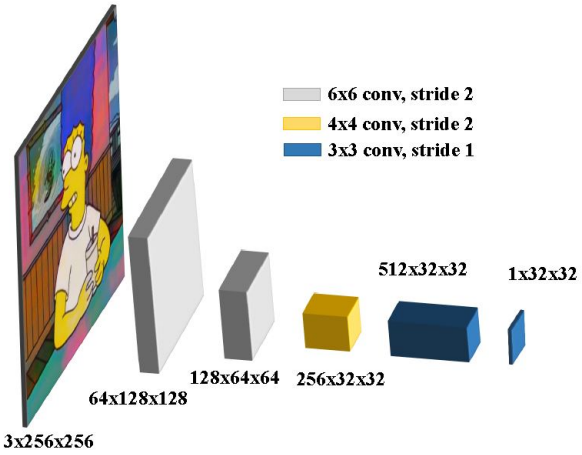

Figure 4. Discriminator for the PatchGAN architecture.

The first network is Avg-PatchGAN. In this network, to

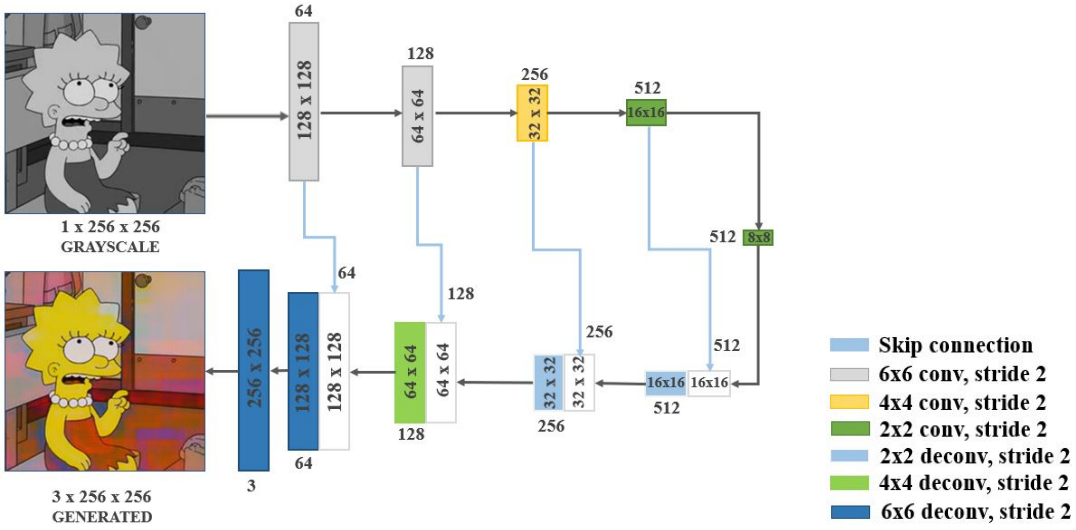

Figure 3. Generator with skip connections.

receive a final likelihood score for an image the mean of the patches is computed, as in (Isola et al., 2017), and as described in section 4.3.1. This network is used as a baseline implementation.

The second network, Min-PatchGAN, has the same generator and discriminator as Avg-PatchGAN but the final likelihood is computed differently. From the first epoch up to a certain point when the image quality plateaus, the likelihood will be computed using the mean of the patches, just as in Avg-PatchGAN. After this point and until the last epoch, the likelihood will be computed using the method outlined in section 4.3.2. The reasoning behind the switch is that the mean processing outlined in section 4.3.1 should achieve a likelihood score which is approximately correct, and then the min based approach should fine tune the likelihood score by focusing on local patches with low likelihood. To find the point when the switch from mean to the method in section 4.3.2 should be made, an experiment was conducted. This experiment is outlined in the section 6.5. The implementation was written from scratch.

6.4. Training

The data input to the discriminator was scaled to [0,1]. The batch size was set to 56 for the trainingset and 14 for the validationset, in order to be able to fit all images in a batch into the GPU memory. Since the trainingset contained 28 000 images and the validationset contained 7 000 images, one epoch was 500 update steps.

Training was done using PyTorch 1.1 on one GCP VM instance using one NVIDIA K80. The K80 was selected due to its' cost effectiveness. In total, approximately $425 was spent during training.

6.5. Experiment setup

6.5.1. Experiment 1

The first experiment compared network 1, 2, and 3. The networks were trained for 10 epochs after which the best network was selected. The selection criterion was image quality. The experiment indicated whether or not pyramid filters and skip connections were meaningful extensions of a cGAN.

6.5.2. Experiment 2

The second experiment involved training the best network from experiment 1 for 60 epochs. The results of this experiment allowed evaluation of convergence and if the picture quality improved after training for many epochs.

6.5.3. Experiment 3

The third experiment aimed at evaluating Avg-PatchGAN for 60 epochs. This indicated when the image quality had stabilized and thus when the change from Avg-PatchGAN to Min-PatchGAN should occur. The experiment also allowed comparisons between Network 2 and Avg-PatchGAN in terms of image quality and convergence.

6.5.4. Experiment 4

The fourth experiment was conducted to evaluate the performance of the Min-PatchGAN. Convergence and image quality was compared to the original Avg-PatchGAN.

7. Experiments

7.1. Results

7.1.1. Experiment 1

The results from Experiment 1 can be found in Figure 5. As can be seen in the figure, qualitatively Network 2 produced superior images. Therefore, this network was selected to be trained for 60 epochs in Experiment 2.

Figure 5. Experiment 1 - Comparison of generated images from Network 1, 2 and 3 at epoch 10. From the upper left to the lower right: ground truth and generated images from Network 1, 2, and 3.

7.1.2. Experiment 2

The images generated in Experiment 2 can be found in Figure 7. The figure contains generated images from Network 2 at epoch 10, 40, 60. The cost graph for Network 2 is found in Figure 6.

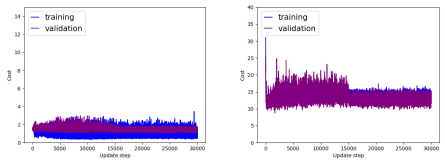

Figure 6. Experiment 2 - Network 2. Discriminator and generator cost graph

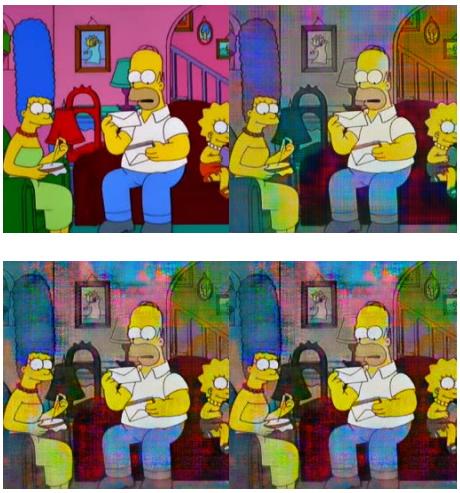

Figure 7. Experiment 2 - The ground truth image is at the upper left corner and next to it is a generated image at epoch 10. At the next row from left to right colorized images from epoch 40 and 60 can be found.

7.1.3. Experiment 3

The results for experiment 3 can be found in Figure 9 and 10. The figures display generated images from Network 2 and the Avg-PatchGAN from left to right at epoch 1, 5, 15 and 60.

Since the generated images from Figure 10 and the plot of its costs in the second row in Figure 8 appeared to have stabilized at epoch 5 this was used as the point when the switch from Avg-PatchGAN to Min-PatchGAN was made in Experiment 4, as described in the last paragraph in section 6.3.

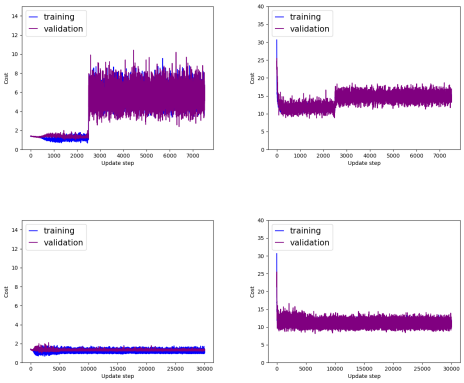

Figure 8. Experiment 4. First row: Discriminator and generator cost graph for Min-PatchGAN. Second row: Discriminator and generator cost graph for Avg-PatchGAN.

7.1.4. Experiment 4

The images generated in Experiment 4 can be found in Figure 11. Cost graphs for Avg-PatchGAN and Min-PatchGAN are found in Figure 8.

7.2. Analysis

7.2.1. Experiment 1

Through Experiment 1 the optimal architecture for Network 2 was found. In line with the results of Isola et al. (Isola et al., 2017) it was found that using skip connections de-

Figure 9. Experiment 2 - Generated images from Network 2 from left to right at epoch 1, 5, 15 and 60.

Figure 10. Experiment 3 - Generated images from Average-PatchGan from left to right at epoch 1, 5, 15 and 60.

creased the amount of blurring in the generated images. Another potential contributing factor, not used in the U-net generator or discriminator architecture by either (Nazeri et al., 2018) or (Isola et al., 2017) , was using relatively large filter kernel sizes that gradually decreased in size as the intermediary results of the generator and discriminator in Network 2 decreased in size.

7.2.2. Experiment 2

The best performing network, Network 2, found in Experiment 1, was run for 60 epochs. However, as shown in Figure 9, despite that some generated images change in appearance throughout the entire training process, qualitatively the colorization does not necessarily improve throughout the entire process for all images in the validationset.

7.2.3. Experiment 3

In Experiment 3 the convergence rate of Avg-PatchGAN and Network 2 was compared. This experiment uncovered that Avg-PatchGAN qualitatively converged faster than Network 2. This is exemplified via the figures Figure 9 and 10. In fact the generated image in Figure 10 at epoch 5 is similar to the ones generated at epoch 60. While the generated images in Figure 9 look distinctly dissimilar.

Since the results of Avg-PatchGAN appeared to have stabilized at epoch 5 it was used as a switching point as described in the last paragraph in section 6.3.

7.2.4. Experiment 4

In Experiment 4, switching from taking the mean of the patches outputted by the discriminator to using a smooth differentiable version of the min and max function as described in section 4.3.1 was done. The approach qualitatively produced better results. Images generated at epoch 10 and 15 are similar to each other, therefore convergence appear to have been reached. Looking at the cost graphs in Figure 8, the curve flatlines which indicates convergence. Further, there is a clear jump in the cost graph for both the discriminator and generator when the switch to Min-PatchGAN is made.

8. Discussion

The networks using PatchGAN converged faster than regular conditional GANs. Fast convergence indicate that training could be feasible using CPUs.

In regular conditional GANs, a good baseline appears to be having the pyramid structure as well as skip connec-

Figure 11. Experiment 4 - Comparison between the two PatchGAN networks. The first row contains the ground truth images, the second row contains generated images from the Avg-PatchGAN method and the last row displays colorized images from the Min-PatchGAN method.

tions. The best results however were yielded by a PatchGAN architecture, with the min max approach described in section 4.3.2. Given the successful application of this approach, future studies could investigate what type of other approaches could be used to calculate the final likelihood score from the patches outputted by the PatchGAN discriminator.

PatchGAN's superior results may be due to the fact that using patches helps the network to better focus on local differences instead of overall differences leading to better image colorization (Isola et al., 2017). Since PatchGAN only penalizes on the scale of patches it appears to more often ascribe single objects uniform colors as they often are a part of a single or a few number of patches.

By adapting to the specific dataset, better results could be achieved. Instead of using the entire RGB space to colorize the images, an investigation and application of the color range used by the original artists may improve the color fidelity in the generated images. Furthermore, as the trainingset consisted of images from 17 seasons done by different animators over a large period of time, one network may not be able to produce excellent results. Thus, training different networks for a smaller number of seasons may lessen variations and produce better results.

The results are good, but cannot replace colorization made by a human, thus further improvements are appropriate. The learning rates used in this study may be too low, leading the model to a local optima. Starting from a different position may also alleviate this problem.

9. Conclusion

In this paper, conditional GANs and PatchGANs were successfully used to colorize cartoon images. Using pyramid structured filters and skip-connections, the image quality of the generated images were significantly improved. PatchGANs produced better colorization and converged faster than regular conditional GANs. The best results were achieved with Min-PatchGAN where the final likelihood score for each image was computed using a weighted approach rather than computing the final likelihood score using the mean function as in the original paper.

References

Dalal, N. and Triggs, B. Histograms of oriented gradients for human detection. In *international Conference on computer vision & Pattern Recognition (CVPR'05)*, volume 1, pp. 886–893. IEEE Computer Society, 2005.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. C., and Bengio, Y. Generative adversarial nets. In *NIPS*, pp. 2672–2680, 2014.

Hafner, J., Sawhney, H. S., Equitz, W., Flickner, M., and Niblack, W. Efficient color histogram indexing for quadratic form distance functions. *IEEE transactions on pattern analysis and machine intelligence*, 17(7):729–736, 1995.

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

Levin, A., Lischinski, D., and Weiss, Y. Colorization using optimization. In *ACM transactions on graphics (tog)*, volume 23, pp. 689–694. ACM, 2004.

Mirza, M. and Osindero, S. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

Nazeri, K., Ng, E., and Ebrahimi, M. Image colorization with generative adversarial networks. *arXiv preprint arXiv:1803.05400*, 2018.

Orhan, E. and Pitkow, X. Skip connections eliminate singularities. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=HkwBEMWCZ.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., and Chen, X. Improved techniques for training gans. In *Advances in neural information processing systems*, pp. 2234–2242, 2016.

Srivastava, R. K., Greff, K., and Schmidhuber, J. Training very deep networks. *CoRR*, abs/1507.06228, 2015.

Versteegen, R., Gimel'farb, G., and Riddle, P. Texture modelling with nested high-order markov–gibbs random fields. *Computer Vision and Image Understanding*, 143: 120–134, 2016.

Welsh, T., Ashikhmin, M., and Mueller, K. Transferring color to greyscale images. In *ACM transactions on graphics (TOG)*, volume 21, pp. 277–280. ACM, 2002.

Zhang, R., Isola, P., and Efros, A. A. Colorful image colorization. In *European conference on computer vision*, pp. 649–666. Springer, 2016.

A. Acquired deep learning skills

A.1. Alexandra Hotti

1. **Comprehending cGANs, GANs and PatchGANs.** Prior to starting this project, I had never heard about GANs. Therefore, initially I spent a lot of time researching what a GAN is, how it works and what is the difference between a cGAN and a GAN. This information was acquired through scientific papers, articles and YouTube videos. By doing so I learned that a cGAN is conditioned on some meaningful information while a regular GAN is not. Thus, the pro of choosing a cGAN is that it is easier to train compared to a GAN. However, such meaningful information is not available in all tasks.

   Later on, in the project in order optimize our results I started researching PatchGAN. The pro of using this method instead of the regual cGAN approach is that the discriminator classifies NxN patches for each image instead of computing a single classification for an entire image. A con with this method is that it is relatively new and thus information about it was a bit scarce. Therefore, information about it was mainly acquired from the original scientific paper by Isola et al. (2017).

   Evidence: On the project github repository our implementations of the cGan and PatchGAN networks can be found. Without an understanding of these methods writing the code from scratch would not have been feasible.

2. **Training a GAN.** Practically training a GAN on image data is a tedious task which requires a lot of time and computational power. During this project we implemented our deep learning architecture in the software package called PyTorch. By doing so I learned both how to write a GAN architecture in PyTorch as well as how to write code adapted for GCP. Therefore, we could significantly speed up our training process such that over a week of training on a CPU took half a day.

   However, this could still be considered time consuming. Therefore, systematically tuning all the different hyperparameters of both the generator and discriminator becomes infeasible. Thus, a con with GANs is that finding an optimal solution requires a lot of time and in our case money. Despite implementing our first network on April 14th and continuously working on the project we did not have enough time to systematically try a lot of different hyperparameters.

   Evidence: Again evidence for our GPU implementation in PyTorch can be found on the project GitHub repository. Also, the fact that we spent a lot of time training our networks is shown on our Google Cloud Accounts where we spent around $425 on the project. Another fact that supports that we continuously worked on our project during this period is that we utilised the help sessions almost every week.

3. **Optimizing a GAN.** Lastly, I have researched and implemented a lot of different performance improvement suggestions for GANs found in scientific papers and articles. For instance, I have learned the importance of selecting correct filter sizes to get images which are not blurry and to use regularization techniques such as weight decay (L2 regularization) and L1 regularization to avoid overfitting.

   However, what most improved the performance of our initial network was not one of the many common optimization advices tried, instead going from a regular cGAN to a PatchGAN network was what improved our results the most. Therefore, solely focusing on the most popular techniques and architectures might not always be optimal for the task at hand.

   Evidence: The optimization techniques and the architectures used are presented mainly in the Method section 6 in this paper. Also, the exact implementations can be found in the generators, discriminators and main training files of our networks on GitHub. In addition, the fact that changing the filters significantly improved the results is shown in section 7.1.1. Lastly, The superior results of the PatchGAN implementation compared to Network 2 is apparent for instance in Figure 9 and Figure 10.

A.2. Jacob Malmberg

1. **An understanding of GAN, cGAN, and PatchGAN**. How the generator & discriminator works together, how the loss is calculated, the difference between a CGAN & GAN, what a PatchGAN is. Compared to a regular CNN, a GAN is harder to train, but it may also produce superior results. This skill was acquired by reading tutorials, guides and research papers.
   Evidence: Together with my teammates I implemented a cGAN and a PatchGAN. This would not be possible without an understanding of the theory behind it.

2. **The practicalities of implementing and training a deep learning architecture**. How to use PyTorch and how to effectively use GCP. PyTorch is an excellent tool to implement GAN in. Since we utilized the cloud/GCP for training, we could train these models in 10 hours instead of in 10 days.
   Evidence: We wrote the implementation in PyTorch. Further, we wrote bash shell scripts to automatically turn off the VM after training so we would not have to wake up in the middle of the night to turn it off.

3. **That an off the shelf solution may not work**

and that there may not be a universal architecture/solution that fits bets. During this project, we tried multiple GAN architectures that had been successfully used to colorize other data sets. For us, these architectures did not produce good enough results. Instead, we had to rely of custom solutions like the Min-PatchGAN and Pyramid shaped filters. To me, this underscores the difficulty of training a GAN, how it is important to try different things, and how time & money are important factors.
Evidence. The non-standard things we did such as Min-PatchGAN and pyramid shaped filters produced the best results. We started writing code on the 15th of April, and started research well before that date.

A.3. Marcus Nystad Öhman

1. **An understanding of GA, cGAN and PatchGAN.** When we first started this project I had never even heard about GANs before. But by reading about it in scientific papers, articles, tutorials and looking at videos on youtube I gained an understanding about what a GAN is and how it works, what the difference is between a GAN and a cGan, how the loss is calculated, what defines a PatchGAN, the adversarial training that occurs in a GAN. A cGAN is condition on some meaningful information and are therefore easier to train. In our case the cGAN is conditioned on the gray scale images. Though such information is not always available.

   Evidence: I implemented together with my project members a cGAN and a PatchGAN that can be found on our project GitHub repository. Without the understanding of this theory and methods that would not have been possible.

2. **Practicalities of training and implementing a deep learning architecture.** I learned how to use the software package PyTorch and how to write a GAN architecture using this very package. PyTorch proved to be a very good package to implement a GAN with. I also learned how to effectively use Google Cloud Platform, GCP, and write code adapted for GCP. By using the GCP for training we were able to train our models within the timeframe of hours instead of days when using GPUs instead of CPUs.

   Evidence: The implementation was written using PyTorch and was adapted to work on GCP which can be seen in our project GitHub repository. We also spent around 425 dollars on the project. Furthermore we also wrote and used bash shell scripts to be able to automatically turn of the VM when we had finished.

3. **Optimizing a GAN and finding a good architecture (not easy).** During the project I have implemented a lot of different improvements suggestions that have been suggested in articles and papers for GANs. For example I learned the importance of the size of the filters to not end up with blurry images, as well as the importance of using different regularization techniques such as weight decay (L2 regularization) and L1 regularisation to avoid overfitting We tried a lot of different GAN architectures during this project that had been successful when colorizing images on other data sets. But we found that the biggest improvement we saw was when switching from a regular cGAN architecture to a PatchGAN architecture. This tells me that focusing only on the most common optimization techniques might not be the best option, and that training a GAN is very difficult, time consuming, costly and also that the architecture that works best depends on the dataset and task at hand.

   Evidence: The optimizations and architectures we used can be found mainly in the Method section of our project report. The implementation in code can be found on our project GitHub repository. In our case the implementations of non-standard architectures like our Min-PatchGAN and pyramid shaped filters resulted in the best results. We started writing the code on the 15th of April and started reading and doing research well before this.