Image Colorization using GANs

ABSTRACT

Our project is concerned with implementing a generative adversarial network that takes black and white images as input and the output obtained is a corresponding colored image. We uses a special type of GAN called Conditional Generative Adversarial Network which takes a specific type of input rather than taking vectors from the random probability distribution. The GAN that we have implemented takes images (with one channel) of fixed size as input (32 X 32 pixels) and the output obtained has the same size as the former. The images that can be efficiently coloured do not belong to a particular class of objects but an array of things to choose from such as animals, vehicles, scenarios, etc.

CONTENTS

CHAPTER 1: INTRODUCTION

1.1 GENERAL

Colour adds a sense of visual appeal when one looks at mundane things. Colours also have the capability to drive emotions of a human being. For example, consider the job of interior designers. They are majorly concerned with deciding upon those color combinations that gives soothing effect to residents and deciding in what patterns these colours are to be used to enhance aesthetics of the place. Thus, at first glance the problem statement of colouring black and white images may look trivial but if suppose black and white image of a house has been given, that GANs can be used to color those photos in such a way that color composition obtained may perfectly match with the structure of the house.

1.2 OBJECTIVE

The main objective of this project is to color fixed-size black and white images of any object in general using generative adversarial networks. Solution to this problem statement can be used for automating applications which require determining or enhancing aesthetic look of something. For instance, UI developers who need to determine the color combinations that appeal to users.

CHAPTER 2: LITERATURE SURVEY

2.1 GENERAL

Though a lot of resources are available on GANs, but since paper published by Ian Goodfellow, inventor of the GAN, was a benchmark paper, we have referred to it to understand basic working of GAN. Architecture of GAN is the most crucial part to decide upon, as performance of a neural network is directly dependent upon the architecture. Our GAN architecture is similar to that described in a research paper on "Image Colourization Using GANs" authored by Kamyar Nazeri, Eric Ng, and Mehran Ebrahimi.

Hyperparameters were needed to be tuned to improve performance. Different online blogs were referred to obtain tips about training GAN more efficiently. Some online tutorials were also referred to understand implementation details of our project.

2.2 LITERATURE REVIEW

Sr. No.	Name of Paper	Name of Author	Year of Publication	Content referred
1.	Generative adversarial nets	Ian Goodfellow, and et. al.	2014	Concept and internal working of GANs
2.	Image Colorization using GANs	Kamyar Nazeri, Eric Ng and Mehran Ebrahim	2018	Structure and architecture of Conditional GANs for image colorization

Table 1: Literature Review

CHAPTER 3: GENERATIVE ADVERSARIAL NETWORKS

3.1 INTRODUCTION

GAN is a generative algorithm introduced by Ian Goodfellow and other researchers in 2014. GAN tries to mimic the distribution of input dataset and accordingly generate new samples from the learnt distribution. GAN is analogous to a game between two players, where one player tries to produce fake notes and the other player tries to distinguish between real and fake notes. GAN is said to be trained well enough when it one player generates fake notes that are good enough to fool the other player to believe that generated notes are real.

3.2 ARCHITECTURE

Two smaller networks, namely generator and discriminator constitute generative adversarial network. The task of generator as mentioned earlier is to produce an output that is indistinguishable from real data. The task of the discriminator, on the other hand is to classify whether a sample came from real data or is fake i.e generated by generator. Architecture of both generator and discriminator is generally a multilayer perceptron model.

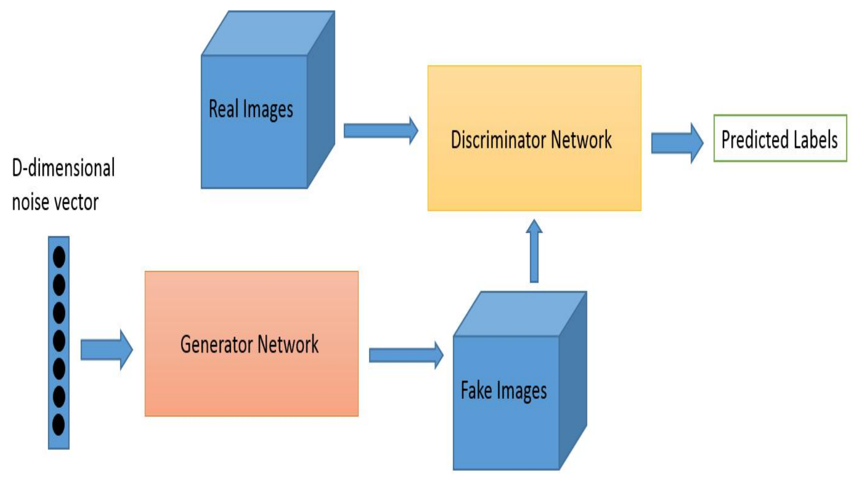


Fig. 1 GAN architecture

3.3 WORKING

Generator tries to fool discriminator by generating data instances close to input dataset. Thus **G** tries to maximize the probability of **D** making mistake. Discriminator tries to identify counterfeit data instances produces by generator. This framework corresponds to a minimax two player game(One for maximizing the probabilities for the real images and another for minimizing the probability of fake images).

3.4 TRAINING

A generator alone will just create random noise. Conceptually, the discriminator in GAN provides guidance to the generator on what data instances to create. GAN builds a discriminator to learn what contributes as real images, and it provides feedback to the generator to help it create more realistic data instances.

3.4.1 TRAINING DISCRIMINATOR

The discriminator looks at real images (training samples) and generated images separately. It distinguishes whether the input image to the discriminator is real or generated. The output *D(X)* is the probability that the input *x* is real, i.e. *P(class of input = real data instance).* We train the discriminator just like a deep network classifier. If the input is real, we want *D(x)=1.* If it is generated, it should be zero. Through this process, the discriminator identifies features that contribute to real data instances.

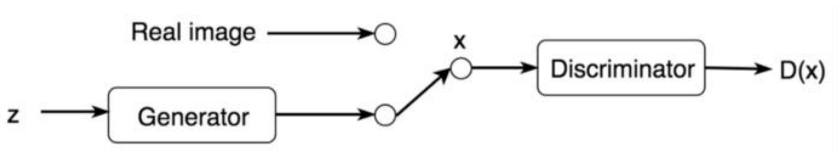


Fig.2 Training Discriminator

The discriminator outputs a value ***D(x)*** indicating the chance that *x* is a real data instance. Our objective is to maximize the chance to recognize real data instance as real and generated data instance as fake. i.e. the maximum likelihood of the observed data. Cross

$$\max_D V(D) = \underbrace{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})]}_{\text{recognize real images better}} + \underbrace{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}_{\text{recognize generated images better}}$$

entropy is used for cost option.

3.4.2 TRAINING GENERATOR

We want the generator to create images with *D(x) = 1*. So we can train the generator by back propagating this target value all the way back to the generator, i.e. we train the generator to create data instances that are inclined towards what the discriminator thinks it is real.

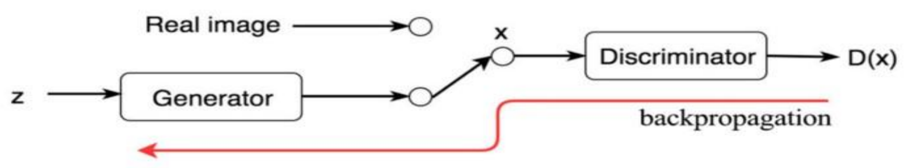


Fig. 3 Training Generator

On the generator side, its objective function wants the model to generate images with the highest possible value of *D(x)* to fool the

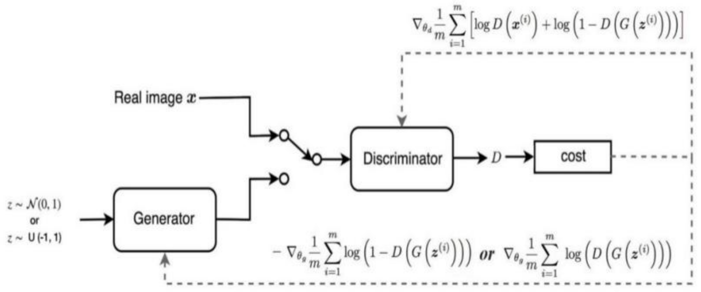
$$\min_G V(G) = \underline{\mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]}$$

discriminator.

3.4.3 SIMULTANEOUS TRAINING OF GENERATOR AND DISCRIMINATOR

Once both objective functions are defined, they are learned jointly by the alternating gradient descent. We fix the generator model's parameters and perform a single iteration of gradient descent on the discriminator using the real and the generated images. Then we switch sides. Fix the discriminator and train the generator for another single iteration. We train both networks in alternating steps until the generator produces good quality images.

Fig. 4 Training GAN


$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right]$
Real image $\boldsymbol{x}$
Discriminator
D
cost
$z \sim \mathcal{N}(0,1)$ or $z \sim$ U (-1, 1)
Generator
$-\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$ *or* $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)$

CHAPTER 4: GAN for Image Colorization

In GANs, the input of the generator network consists of noise vector which are randomly generated. But in the task of Image colourisation problem, such GANs will not work as our GAN is not designed to generate images from random vectors but instead add colours to already existing photos having only one channel (Black and white). So the basic task for our GAN is to add three channels (RGB) with relevant intensities of each color channel. Hence, to address this problem, we use a special flavor of GAN called Conditional GANs which accepts grayscale images (with one intensity channel) as input (i.e G(0 z |x), mathematically). The discriminator input is also changed to be compatible with the conditional GANs. Our final cost functions are as follows with above modifications.

$$\min_{\theta_G} J^{(G)}(\theta_D, \theta_G) = \min_{\theta_G} -\mathbb{E}_z \left[\log(D(G(\mathbf{0}_z|x)))\right] + \lambda \|G(\mathbf{0}_z|x) - y\|_1$$

$$\max_{\theta_D} J^{(D)}(\theta_D, \theta_G) = \max_{\theta_D} \left(\mathbb{E}_y \left[\log(D(y|x))\right] + \mathbb{E}_z \left[\log(1 - D(G(\mathbf{0}_z|x)|x))\right]\right)$$

The discriminator takes in input as a pair of grayscale image and generated image, and grayscale image and original image. It then judges for fake or real pair.

4. 1 METHOD

The problem we are trying to solve comes under the category of Image to Image translation with mapping from high dimension input to high dimension output. It is actually regression on pixel level with a condition of output having structure similar to the input. Hence the network needs to have very high similarity of spatial dimensions of input and output and also provide information regarding the color to each pixel in the original grayscale image.

4.2 GAN ARCHITECTURE

The network for this model is based on "fully connected networks". We use layers of convolutions in Generator. But instead of pooling layers, downsampling of the image is done till it becomes vector of size 2x2 pixels. Then upsampling is done to expand the compressed part and making it to the size of the input sample (i.e 32 x 32 pixels). This strategy is motivated by special types of deep networks called Encoder-Decoder networks containing encoding and decoding networks for contracting and then expanding and hence reconstructing the input. This strategy helps in training the network without consuming large amount of memory.

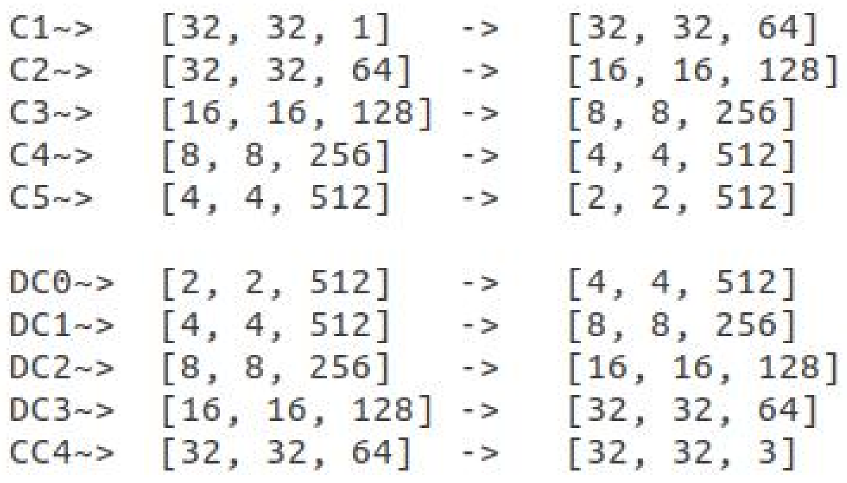
The generator takes an input X as a grayscale image having dimensions (32 x 32 x 1). It is initially downsampled with kernel of size 1 and stride

1. After this layer, it is subsequently compressed to image to size (2 x 2) with kernel of size 2 and strides 2. This is done four times after the initial layer, making the matrix of dimensions (2 x 2 x 512).

The expansion stages consists of upsampling of the matrix with kernel size 2 and strides 2 except the last layer. Concatenation of i and n-i layers are done to preserve the structural integrity of the image. In the first and second expansive layers, dropout of scale 0.5 is done to introduce noise for robust training of Generator. Batch normalization is done for better training. In our model, we used LeakyReLU with slope of 0.2 as it has shown better performance than ReLU activation function. In the last layer convolution with kernel size 1 and strides 1 is done to construct image of dimension (32 x 32 x 3). "tanh" activation function is used as it has shown to have better performance than linear activation functions. It gives output in the form of matrix containing values from -1 to 1.

We train the model to minimize the cosine distance between predicted and the original image.

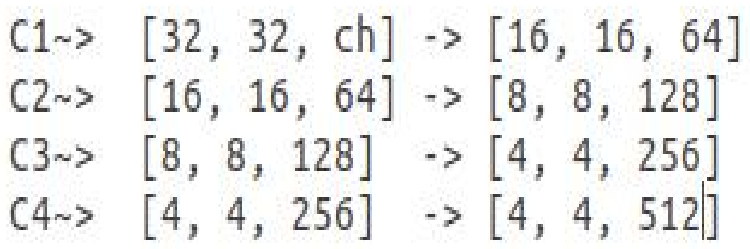
For discriminator, we first concatenate the grayscale image and the predicted or the grayscale image and the ground truth image on the channel axis (axis=3), hence it forms a coloured image. We perform downsampling of the matrix successively using convolutional layer with filter size of (2 x 2) and strides equal to 2. Each layer has Leaky ReLU activation function with slope 0.2 and Batch Normalization is performed at every layer. The last layer is flattened followed by a hidden layer containing 128 units, which are connected to output layer containing 1 unit. The activation function which is used in the last layer is "sigmoid", which gives the probability of the input image to belong to predicted one or the ground truth.



```
C1~>   [32, 32, 1]    ->   [32, 32, 64]
C2~>   [32, 32, 64]   ->   [16, 16, 128]
C3~>   [16, 16, 128]  ->   [8, 8, 256]
C4~>   [8, 8, 256]    ->   [4, 4, 512]
C5~>   [4, 4, 512]    ->   [2, 2, 512]

DC0~>  [2, 2, 512]    ->   [4, 4, 512]
DC1~>  [4, 4, 512]    ->   [8, 8, 256]
DC2~>  [8, 8, 256]    ->   [16, 16, 128]
DC3~>  [16, 16, 128]  ->   [32, 32, 64]
CC4~>  [32, 32, 64]   ->   [32, 32, 3]
```

Fig. 5   Generator Architecture Plan



```
C1~>  [32, 32, ch] -> [16, 16, 64]
C2~>  [16, 16, 64] -> [8, 8, 128]
C3~>  [8, 8, 128]  -> [4, 4, 256]
C4~>  [4, 4, 256]  -> [4, 4, 512]
```

Fig. 6 Discriminator Architecture Plan

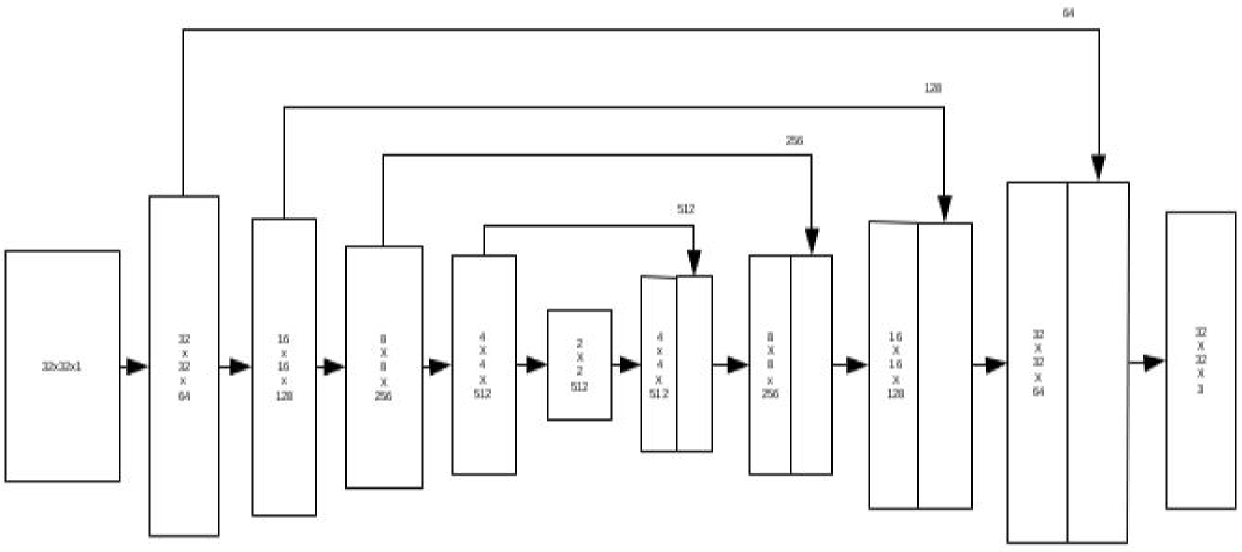


Fig.7 Generator Network visualization

4.3 SOME SUCCESSFUL OUTCOMES OF OUR MODEL

-The top image shows the single channeled image (Equivalent to GrayScale image)
-The middle image shows the image generated by our GAN
-The bottom image shows the ground truth.

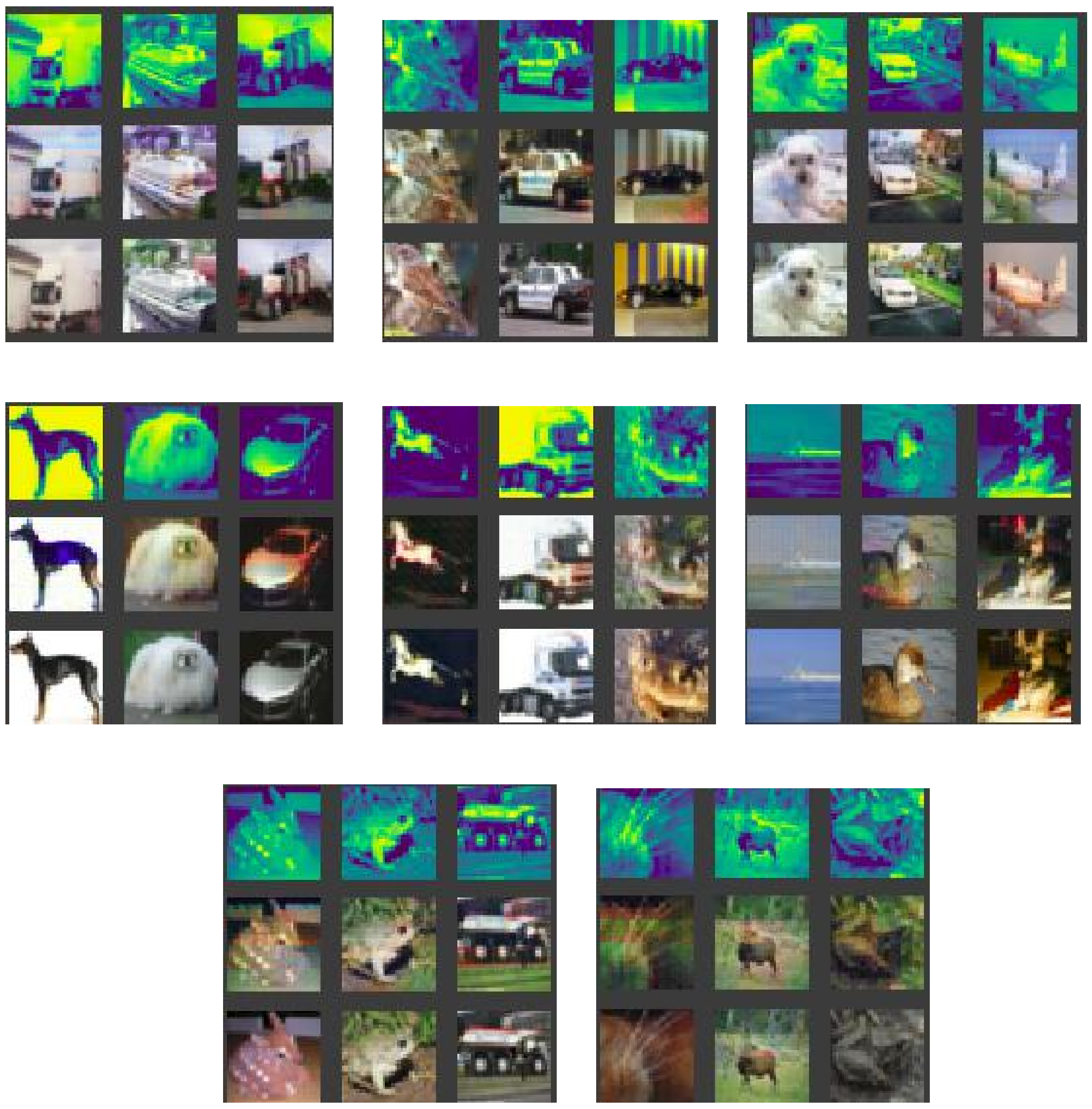


Fig.8 Some results of the model

4. 3 TRAINING STRATEGIES

For our model, we have used Adam's optimizer. The learning rate for the optimizer is kept to 0.0001. We have made use of open source python libraries Tensorflow and Keras for our model implementation. We have trained the model in free Google Colab GPU. The size of our batch is 50 images.

4.3.1 BATCH NORMALIZATION

The GAN is believed to be hard to train because of a phenomenon called 'mode collapse'. In the model collapse phase, the generator succeeds in fooling the discriminator into believing the same generated output to be true. Hence, the generator generates similar outputs everytime and hence the generation lack variety. This phenomenon is an unwanted state in the training phase of the GANs. To avoid the above phenomenon, we use batch normalization which is proven to reduce the probability of mode collapse. However, batch normalization is avoided in the first layer of the discriminator and generator and the last layer of the generator as suggested by[1].

4.3.2 ALL CONVOLUTIONAL NET

Instead of using spatial pooling, strided convolutions are used. Hence, rather than depending upon fixed downsampling and upsampling, strided convolutions allows model to learn its own upsampling and downsampling. This technique has shown to upgrade the performance of training and helps the network to learn the important invariances with convolution layers only.

4.3.3 LEAKY RELU ACTIVATION FUNCTION

LeakyReLU activation function gives better performance then normal ReLU, hence we have used it where ever activation function is used.

4.3.4 MODE COLLAPSE AVOIDANCE STRATEGY

In our case, during some part of the training, we found that the generator generated images using same pattern of colors, along with color grids. This was the outcome of ‘mode collapse’ as explained earlier. Other than batch normalization, we used a novel approach devised by our friend Kush Jajal, in which we train the generator to avoid using the same colors. This is sort of reverse training for the generator as we intuitively make the generator avoid making the mistake of using the same colors every time.

CHAPTER 5: SUMMARY AND CONCLUSION

5.1 SUMMARY

Our project involved use of conditional GAN for colourisation of black and white images. While implementing our project, we realised that architecture of a neural network as well as careful selection of hyper parameters act as a bottleneck to any deep learning project's success. We realized that even minor changes in such aspects of GAN can massively influence performance of GAN or any neural network in general.

5.2 CONCLUSION

In this project, we were successful in automatically color the grayscale images using Generative adversarial networks, to a visual degree which is acceptable. The images of CIPHER 10 with synthetic colors by GAN looked reasonably well and similar to the original images. There were some incidences where the model misunderstood the sea water for grass during the training process, but with further training, it was successful in coloring green color for grasses. We observed that the model faced unusual problem with red color, which it learnt after many epochs as compared to other colors.

REFERENCES

1. Kamyar Nazeri, Eric Ng and Mehran Ebrahim*, Image Colorization using Generative Adversarial Networks*, arXiv:1803.05400 [cs.CV]

2. Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Advances in neural information processing systems, pages 2672–2680, 2014.

APPENDIX - A LIST OF USEFUL WEBSITE

1. https://medium.com/deep-dimension/gans-a-modern-perspective-83ed64b42f5c
2. https://medium.com/@jonathan_hui/gan-whats-generative-adversarial-networks-and-its-application-f39ed278ef09
3. https://www.oreilly.com/learning/generative-adversarial-networks-for-beginners
4. https://becominghuman.ai/gans-the-art-of-creating-fakes-ab245a5a4aa1
5. https://hackernoon.com/how-do-gans-intuitively-work-2dda07f247a1
6. https://medium.com/nurture-ai/keeping-up-with-the-gans-66e89343b46
7. https://towardsdatascience.com/generative-adversarial-networks-using-tensorflow-c8f4518406df

LIST OF FIGURES

LIST OF TABLES