INTERNATIONAL JOURNAL OF SCIENTIFIC RESEARCH IN ENGINEERING AND MANAGEMENT (IJSREM)
VOLUME: 04 ISSUE: 02 | FEB -2020 ISSN: 2582-3930

Grayscale Image and Video Colorization using Generative Adversial Networks

C. Balamurugan[1], R. Chandra Hassan[2], D.S. Jaya Surya[3], J. Jerome Marshall[4]

[1]Assistant Professor, Adhiyamaan College of Engineering
[2]Student, Adhiyamaan College of Engineering
[3]Student, Adhiyamaan College of Engineering
[4]Student, Adhiyamaan College of Engineering

Abstract - In this project, we will demonstrate an approach that uses deep learning techniques to color grayscale images and videos. Using GAN (Generative Advertising Network), a neural network architecture designed for image processing, we are able to differentiate the content and style of different images and combine them into one image. And it can be very effective. Another important thing about Nogan training is that you can pre-empt the Discriminator on the pictures you produce after the first Generative Adversial training, after which the same training repeated in same manner. Thus, we got to a point it produced colorful results with the same model. But it comes with a bigger cost right now - the output sample from the Generator has become unstable and we need to experiment with the render factor to get optimum results. But the current outputs are still better and then what we could achieve with previous model. We did about five of these repetitive cycles before we got the diminishing returns. The final classification-based model we build produces colorful images and videos that are similar to reality. This project really helps in improving computer vision.

1. INTRODUCTION

Supervised learning is central to much of the research in deep learning. But, the need to create models that can learn from less data is growing rapidly. To put this in an understanding manner, Semi-Supervised learning uses the combination of both labeled data and unlabeled data which is then used to train the model. This type of training makes use of both the data (labeled and unlabeled) from a similar domain. The aim is to combine these data sources to train deep convolutional neural networks (DCNN) to determine the performance and we present the GAN model to color images and videos with a labeled training set. In fact, the model uses approximately 1.3% of the original ImageNet training labels, i.e., 1000 (thousand) labeled examples. We use some of the methods described in the Improved Techniques Paper for GANs trained from OpenCV. We use Nogan. We propose a new but a similar type of Generative Adversial training to address some of the key issues in the already existing system. This provides the advantages of the Generative Adversial training method. As a result, a less amount of time is spent on GAN training. Instead, the generator and the critic is trained more often with more faster and reliable methods. An important point to note here is that the "training" methods usually get the results we were expecting for and they can use GANs to render images closer to reality. During the shortest period of Generative Adversial training, the Generator does not receive the full range of real-time characterization capabilities that would normally take days to re-sized GAN training, and virtually no error and other mis colorization which were caused in the previous iteration.

2. METHODOLOGY

The methodology we proposed is inspired in [1] and [2]. As mentioned earlier, the goal is to color the black and white images. To achieve this, our main proposition is to use a generative advertising network consisting of two neural networks, the generator and the discriminator. The general training workflow can be seen in Figure 2.1. In the following subsections we describe each step-in detail on how each model is responsible for colorizing the image and video in each aspect and the principle behind them.

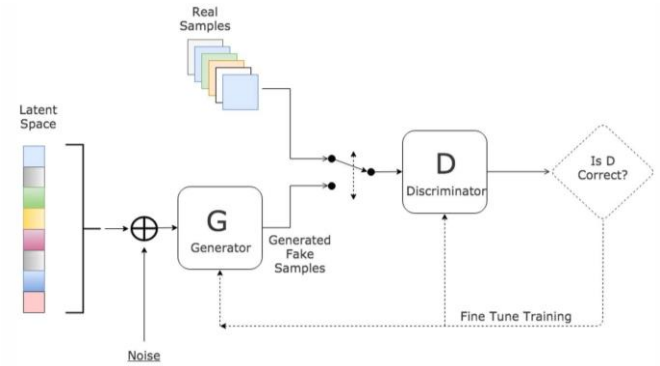

Fig. 2.1 GAN Architecture

A. Generator

The Generator part of the Generative Adversial Network get to learn from the critic to create duplicate images by using the loss function of the Discriminator. The

Discriminator learns to truly classify the samples it receives from the original dataset and from the generator.

Generator training requires more rigorous integration than discrimination training between generator and discriminator. Theb Generator learns from:

- Random input
- Generator network, which converts image noise into an image representation
- A Discriminator and critic, which classifies the sample images from the dataset and generator
- Generator gradient descent, penalizes the Generator for not passing the Discriminator classification.

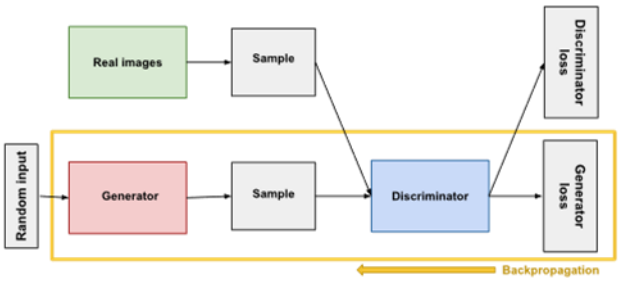

Fig. 2.2 Generator

B. Discriminator

Discrimination in GAN is used as a classifier. It tries to separate the real data from the data generated by the generator. It can use any network structure that matches the data type it classifies. Discrimination training data comes from two sources:

- Real data is the representation of real-world entities. Discrimination uses these cases as the true part of the training process.
- Generator generates duplicate data. Discrimination uses these cases as the false part of the training process.

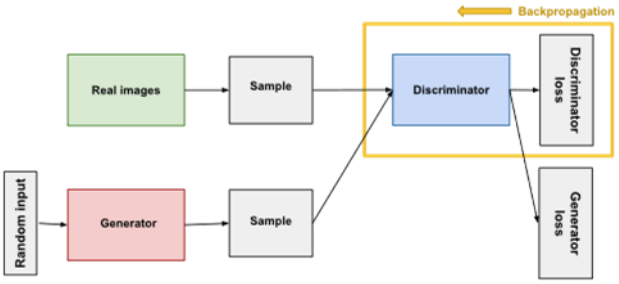

Fig. 2.3 Discriminator

C. Back Propagation

This is the summary of Back Propagation Neural Net Training. It is a method of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous era (i.e. repetition). Proper tuning of weights ensures lower error rates, making the model more reliable by increasing its generalizability.

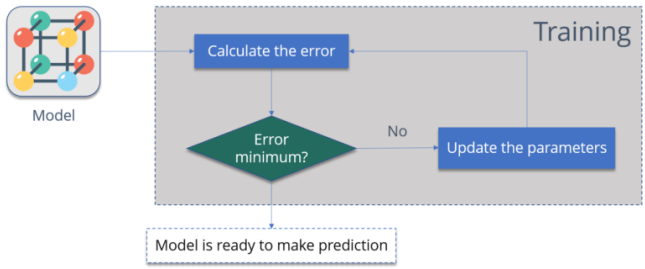

Fig. 2.4 Back Propagation

Working of Back-Propagation. We perform a delta calculation step at each unit, re-propagate the loss, and determine what damage each node causes.

D. K-Nearest Neighbor

The KNN algorithm groups the nodes that are close to similar things. In other words, things like this are close to one another.

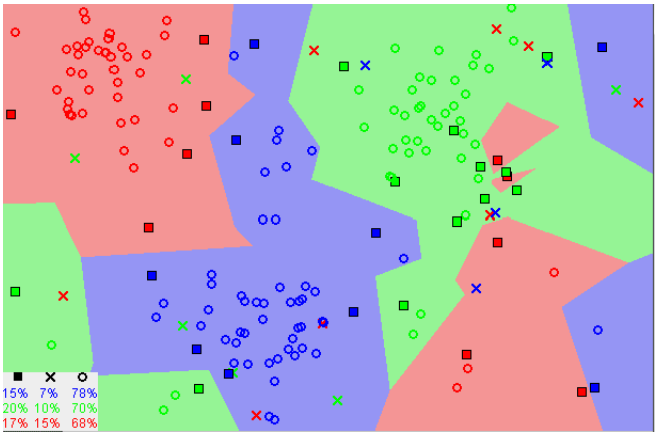

Fig. 2.5 KNN

Note in the picture above, that the same data points are close to each other. The KNN algorithm adheres to this assumption to be true enough for the algorithm to be useful. KNN incorporates the concept of similarity (sometimes called distance or proximity) into a specific statistic calculating the distance between points on a graph.

E. Dataset Construction

Beginning with preprocessed images and the codebook, the next step is to build a data set for training. Image pre-processing and color reduction work-flow. The simplest dataset consists of one attribute, the L value of the pixel and the target for each attribute, and the U and V values for that pixel. In this work we want to consider the pixels near the pixels we want to represent. This is because, in general, the images have some degree of similarity in color, and the information of the surrounding pixels helps to estimate the color of the target pixel. Additionally, adjacent pixels have

a structure that helps to classify pixels. Therefore, for each pixel we consider the neighborhood of the pixels, from which we call the patch. These patches are odd shaped squares where the pixels are in the middle of tingling. Then, our dataset has many features depending on the number of pixels of a patch. After getting the patch, we need to get to the target for the properties of that line. As mentioned earlier, in our case the target is the U, V portion of the central pixel of the patch, but with a decrease in the potential values. The U and V component of the central patch is given as input to the trained SOM and an indicator representing the cluster of the U, V component is obtained. This indicator is used as a target in our dataset.

F. Neural Networks

We propose to use the General Advisory Network (GAN) neural network. The purpose is to classify each patch in the KNN cluster under the central patch. Therefore, the neural network is classified in a way that the weights are adjusted whenever there is a change in the pretrained weights of each individual neurons in both the generator and the discriminator networks. Fig. 2.6. Neural network model. NNs are trained in the batch process using a dataset already built with the back-propagation algorithm

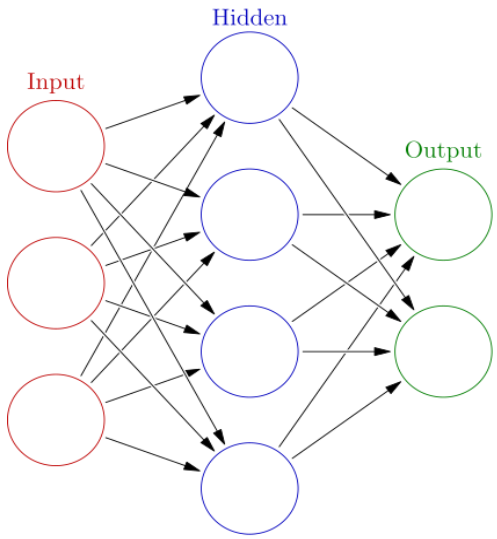

Fig. 2.6 Neural Network

G. Generator Loss

The NoGAN learning phase involves losses of two parts: one is a basic perceptual loss (or feature loss) based on the VGG16 - which makes the generator model to bias and reflect the given grayscale image. And the second one comes from the score of the critic. Interesting Perceptual Loss is not enough to give good results. It promotes brown, green, blue, and other cheating testing, basically, what neural networks really do. The important point to note here is that Generative Adversial Networks are essentially learning loss for you - a big step closer to the ideal we are shooting at machine learning. In fact, you usually get optimum results when the machine learns on what you already did by hard coding. That's exactly it here.

The main features of using this methodology comes from a motive where it is usually used for all kinds of image editing and it can be done very well. What you see in the results are the characterization model, but it's part of the pipeline, and we're developing the same approach.

3. UML DIAGRAMS

UML Diagram is a diagram based on UML (Unified Modeling Language), about the system to better understand, change, organize or provide document information, with the purpose of representing the system visually with its major actors, characters, actions, artifacts or classes.

In this case, different aspects and features of the system are used to communicate. However, this is only a high-level view of the system and does not contain all the details necessary to run the project.

A. Activity Diagram

Activity diagrams are very important UML diagrams for business model processing. In software development, it is commonly used to describe the flow of different activities. These have workflow that are vertical and horizontal. They describe the relationship between the workflow and various activities.

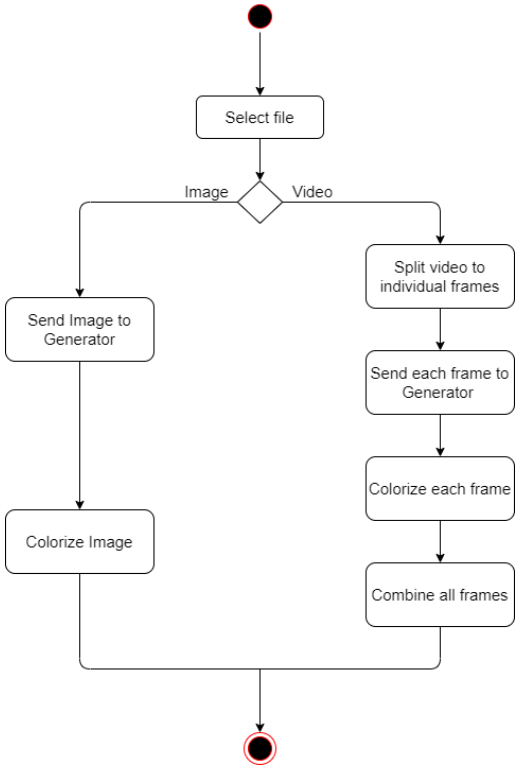

Fig. 3.1 Activity Diagram

B. Use Case Diagram

The use case diagram is a dynamic or behavior diagram in UML. Use Case Diagrams model the functionality of the system using actors and use cases. Use cases are a set of actions, services and functions that the system needs to perform. In our case, this "system" is a website developed or maintained. "Actors" are individuals or organizations that operate under defined roles in the system.

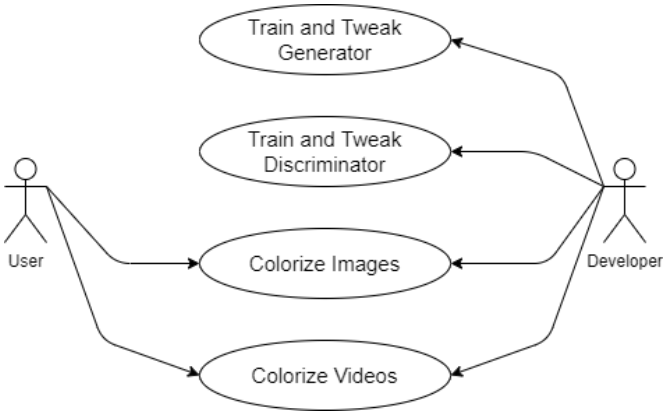

Fig. 3.2 Use Case Diagram

C. Architecture Diagram - Training

- Training the Generator

The generator is provided with some noise as input and the output is obtained relative to the current weight of the neuron. In our case, the grayscale image input from the dataset and the generator tries to colour the grayscale image. Beforehand, we train the Generator in the traditional way with feature loss. Next, create images from it and train discriminator to identify those derivatives, real colour images as the basic binary classification. At the end, train the critic and the generator, both together in the Generative Adversial Network setting. And for the next part: all the useful GAN training time here only happens in a very short burst. There is a point where the critic has trained the generator to an extent it can learn no more. Skip to this phase, the quality of the image colouring scheme will change to the best colour accuracy we can get in this point, or the way it's bad. After this infliction point, there is no point in training the generator as it will have little to no improvement.

- Training the Discriminator

Discrimination is a type of classifier. It is given two inputs, the output of the generator and its corresponding pattern image (colour) from the dataset. Discrimination classifies input as a fake or real image. It is trained using the K-nearest neighbour algorithm until the discriminant images are correctly classified. The difficult part is to find this inflation point. So far, we've been saving the entire group of checkpoints (repeating every 0.1% of the data) and searching for a point in the execution where the images are like the original coloured images before going completely bonkers with the Orange Skin (this always happens first). In addition, the generator rendering begins to become unstable and more unstable at this phase, and this is not particularly good for video. Unfortunately, nothing definite has happened for us yet. For one, this is happening in the midst of a decrease in training loss — not when it is flattened, but rather on the surface.

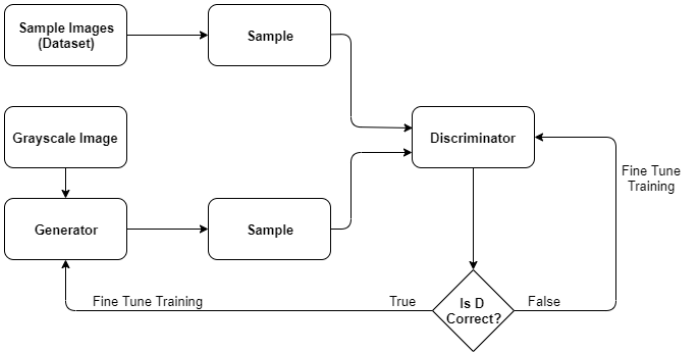

Fig. 3.3 Training Architecture

D. Architecture Diagram - Execution

- Colorize Image

The trained generator is given any image and it produces the coloured image.

- Colorize Video

The video is split into frames. Each frame is then processed individually through the generator and the output frames are combined to form a video.

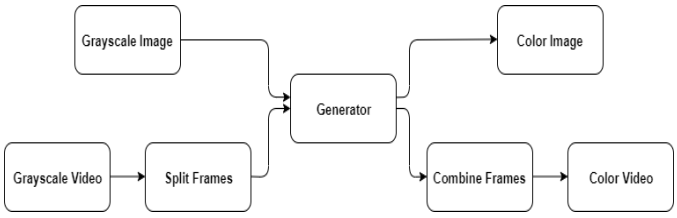

Fig. 3.4 Execution Architecture

E. Modules

There are four modules implemented in this project. First one is the Dataset Collection where all the training images are collected and stored in the required format. The second module is the Training the Generator where the generator is trained with Back Propagation and the Generator Loss function. The third module is training the discriminator. The discriminator is trained as a classifier which classifies the image from the Generator as a fake or real image. The fourth and final module is the colorization module where the images are colorized by the Generator.

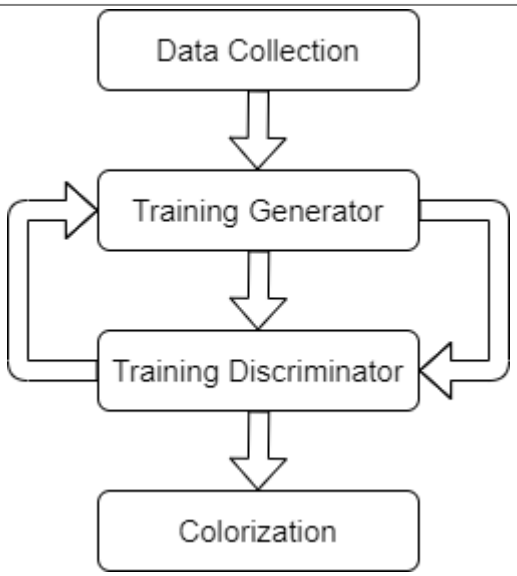

Fig. 3.5 Modules

4. USE CASE

In this section we present our proposal implementation and utilization. We use Python language with Pytorch and FastAi library for NN classification implementation and generator and discriminant implementation. We made all the codes available. Obviously, this system can be used on any given image of any class. This is because the model is trained irrespective of the image specified. Fig. 3.6. Color estimation work-flow. We used the images from ImageNet as the source of our dataset.

A. Dataset

The dataset mainly consists of images from the ImageNet database. The dataset consists of two parts, the original color images and their respective grayscale counter parts.

The original imagers are encoded in jpeg format and are of the resolution 256 x 256. The dataset contains images from different sources like fields, desert, landscapes, mountain ranges, hill, etc. The images were randomly sampled into pixels for training purposes. During the training phase 2500 – 3000 pixels from each individual images were processed to form different patches.

B. Render Factor

Rendering at high resolution (high Render_Factor) will optimize the process of choosing the colors when coloring the image. This happens because there is too much reliable image pixels and data to access and work with the model and the "right" decision is more likely to be consistent. Closely related is the application of resnet11 instead of resnet343 as the background of the generator - objects are more stable and accurate.

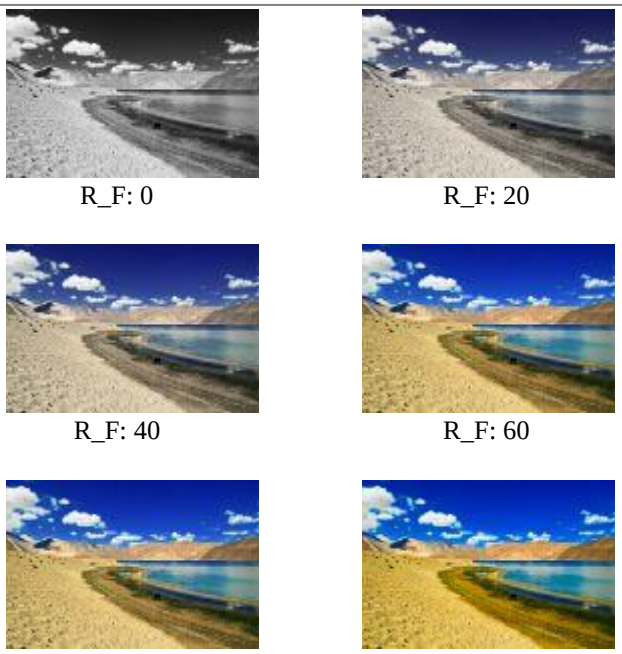

Fig. 3.6 Render Factors

The Render Factor (R_F) determines the colorizing factor of the Generator. The greater the R_F the accurate the coloring becomes. It is implied in the above given example.

C. Results

The results show that the generator and discriminator are learning some features and patterns on how to color based on the subtle hints in grayscale images. And the result paves the way to good decisive and optimum results and this means that you have no model tracked colorizing decisions because they are not arbitrary.

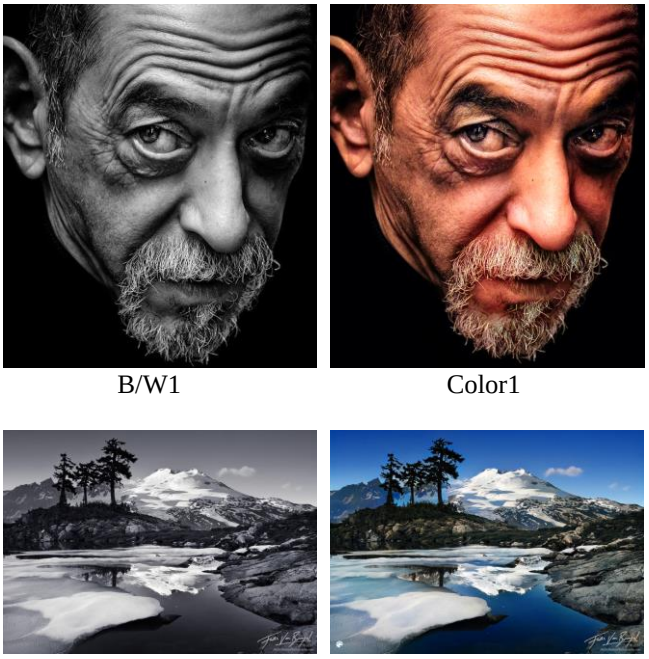

Fig. 3.7 Results

5. CONCLUSION AND FUTURE WORK

In this project we implemented a methodology based on Generative Adversial Networks to colorize grayscale images. The overall process involves the necessary steps to train each of the neural network in the GAN network, which represents the generator and the discriminator respectively. This project is inspired from two other existing systems: GAN networks to achieve color accuracy and high accuracy using vector quantization. We have completed this project and have tested it across multiple sources of images. In the perspective of the end-user, we are getting good results, the rendered images are of higher quality and the error rate is almost closer to null. Our model achieves a color accuracy of 90%. Since prediction is highly dependent on a set of training images, images similar to a training set may produce better results than different ones.

So this is the essence of the project - we are proposing to make old images and movies look better with Generative Adversial Networks, and most importantly, the project is useful. We are try to make it as user-friendly as possible. We document the code correctly. And we are considering the possibility of porting this project to a mobile platform. We are in the direction of offer mobile support.

REFERENCES

[1] Image Colorization with Neural Networks: - Mat´ıas Richart, Student member, IEEE & Jorge Visca, Senior member, IEEE. https://ieeexplore.ieee.org/document/8278079
[2] Image Restoration for Halftone Pattern Printed Pictures - - Adrian Ciobanu, Tudor Barbu, Mihaela Luca. https://ieeexplore.ieee.org/document/7823203
[3] J. Ironi, R. Cohen-Or, and D.S Lischinski, “Colorization by example.” vol. 15, no. 5, pp. 1120–1129, 2006. in Rendering Techniques, 2005, pp. 201–210. [5]
[4] L. Yatziv and G. Sapiro, “Fast image and video colorization using chrominance blending,” IEEE Transactions on Image Processing,
[5] R. Cheng, P. Yang, and C. Sheng, “Deep colorization,” , pp. 415–423. in Proceedings of the IEEE International Conference in Computer Vision, 2017
[6] G. Larsson, M. Maire, and G. Shakhnarovich, “Learning representations for automatic colorization,” arXiv preprint arXiv:1603.06668, 2016.
[7] J. Zhang, J. Isola, and A. A. Efrose, “Colorful image colorization,” arZiv preprint arXiv:1603.06541, 2015.
[8] J. Hwange and Z. Zhou, “Image colorization with deep convolutional neural networks,” Stanfort University, Tech. Rep., 2018. [Online]. Available: http://cs231n.stanford.edu/reprts2018/219 Report.
[9] J. Yoo and S.-Y. Oh, “A coloring method of gray-level image using neural network,” vol. 2, 1997, pp. 1203–1206. in Proceedings of the 1998 International Conference on Neural Information Processing