PM DL
Project Presentation

Colourizing Black and White
Images using GANs

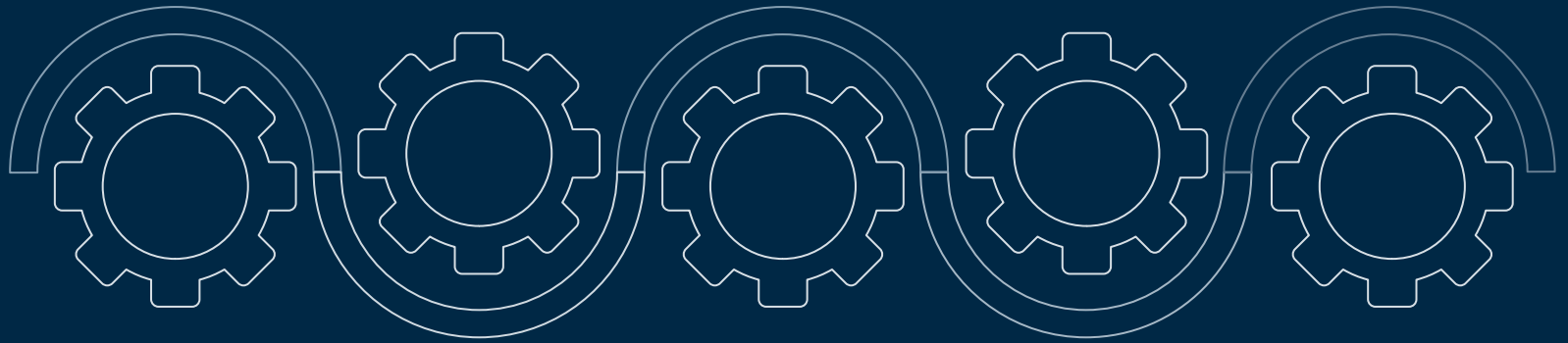
Our team

Parth Kalkar

Hasan Khadra

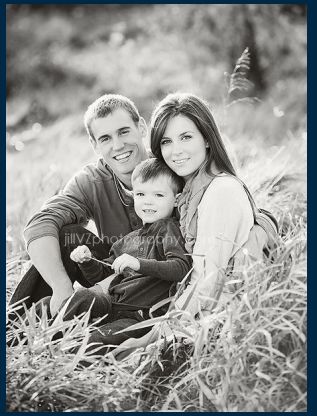
Doro Umarov

Project Methodology



The Problem

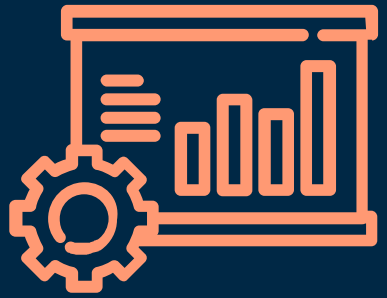
Memories play an important role in our lives. All of the memories are captured either by pictures or videos. Everybody wants to relive the moments that took place in the past, but many of those memories are in black-white.



Goals

- **Try different Deep learning architectures, algorithms, and models**
- **Build a fully functioning AI-based web app.**

The Colorization problem



An image has 32*32 pixels and a pixel has 256^3 different possible colors. That adds up to 17179869184. That is the number of different colored images one can create based on a single gray image. So how can we color images efficiently?

How to solve it?

During the last few years, many different solutions have been proposed to colorize images using deep learning.

- The colorful Image Colorization paper approached the problem as a classification task.

- Another paper approached the problem as a regression task (with some more tweaks!).

Each approach has pros and cons, but we will use a different strategy in this project.

Our Strategy

We followed the methods explained in Image-to-Image Translation with Conditional Adversarial Networks (pix2pix).

In this approach, two losses are used: an L1 loss, which makes it a regression task, and an adversarial (GAN) loss, which helps to solve the problem in an unsupervised manner by assigning the outputs a number indicating how "real" they look.

We first implemented what the authors did in the paper and then tried to introduce a whole new generator model with some tweaks in the training strategy.

Solution

- Conditional GAN with an extra loss function L1
- Discriminator train on some real images taking **L** into consideration.
- The generator model takes a grayscale (1-channel image) and produces a 2-channel image, a channel for ***a** and another for ***b**.
- The discriminator takes these two produced channels, concatenates them with the input grayscale image, and decides whether this new 3-channel image is fake or real.
- The condition in our GAN model is the grayscale image that both the generator and discriminator see.

L*a*b Scheme

We call each of these components a **channel**

- **L** - Lightness of each pixel
- ***a** - how much is a pixel green-red
- ***b** - how much is a pixel yellow-blue

RGB Scheme

- **R** - How much of red is there in a pixel
- **G** - How much of green is there in a pixel
- **B** - How much of blue is there in a pixel

Our dataset – Common Objects in Context (COCO)

01

COCO Dataset

was created to advance image recognition.

02

Dataset Contains

328,000 images of everyday objects and humans.

03

For training

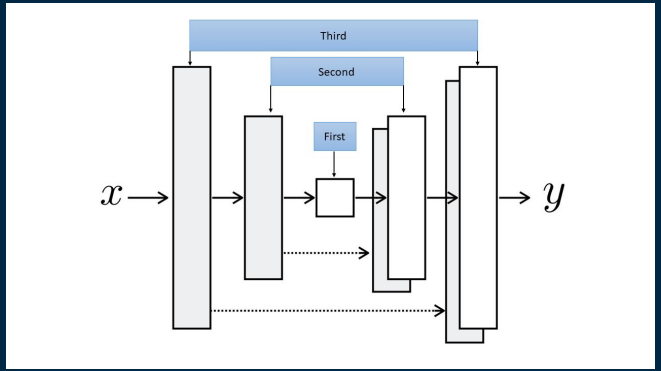
we use only 8,000 Images

Generator: U-NET

The U-Net is from the middle part (down in the U shape) and adds down-sampling and up-sampling modules to the left and right of that middle module (respectively) at every iteration until it reaches the input module and output module.

The U-Net we built has more layers than depicted in this image, but it suffices to give you the idea.

Also, notice in the code that we are going eight layers down, so if we start with a 256 by 256 image in the middle of the U-Net, we will get a 1 by 1 ($256 / 2^8$) image, and then it gets up-sampled to produce a 256 by 256 image (with two channels).



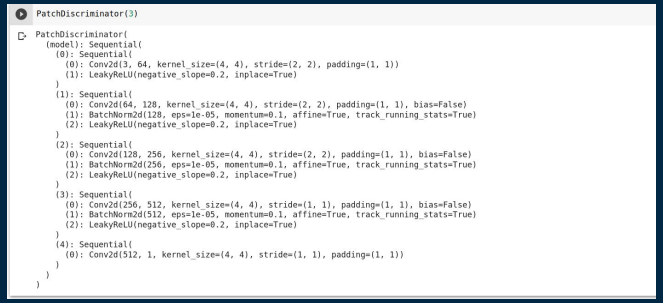
Discriminator - 1

We use a "Patch" Discriminator!

The architecture of our discriminator is rather straightforward and it is made by stacking blocks of:

Conv-BatchNorm-LeackyReLU

to decide whether the input image is fake or real.



Discriminator – 2

In a vanilla discriminator, the model outputs one number (a scaler), which represents how much the model thinks the input (which is the whole image) is real (or fake).

In a patch discriminator, the model outputs one number for every patch of, say, 70 by 70 pixels of the input image, and each of them decides whether it is fake or not separately.

Using such a model for the task of colorization seems reasonable because the local changes that the model needs to make are important, and maybe deciding on the whole image as in vanilla discriminator cannot take care of the subtleties of this task.

Training

1. Initialize the weights of the model with a mean of 0.0 and a standard deviation of 0.02.
2. Define two loss functions and the optimizers of the generator and discriminator.
3. Call the module's forward method and store the outputs in the fake_color variable of the class.
4. First train the discriminator by using the backward_D method by feeding the fake images produced by the generator and label them as fake.
5. Then feed a batch of real images from training sets to the discriminator and label them as real.
6. Add up the two losses for fake and real, take the average, and then call the backward on the final loss. Now, train the generator.
7. In the backward_G method, we feed the discriminator the Fake image and try to fool it by assigning real labels to them and calculating the adversarial loss

Evaluation

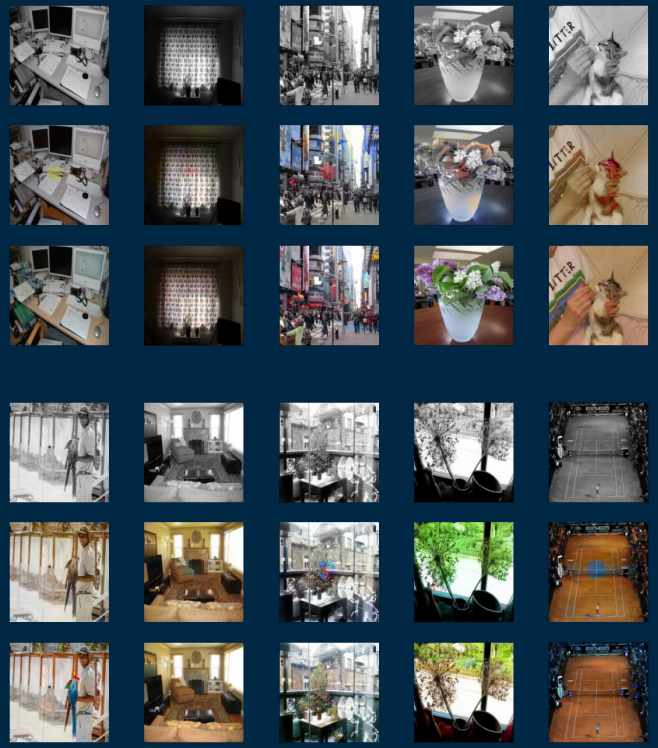
1. As mentioned earlier, we use L1 loss as well and compute the distance between the predicted two channels and the target two channels and multiply this loss by a coefficient (which is 100 in our case) to balance the two losses and then add this to the adversarial loss.

2. Then we call the backward method of the loss.

Results - 1

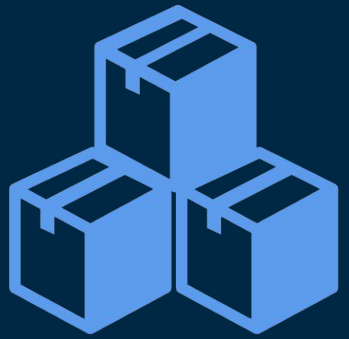
Epoch	loss_D_fake	Loss_D	loss_D_real	loss_G_GAN	loss_G_L1	loss_G
5	0.52271	0.55515	0.58760	1.24612	10.92885	2.17497
10	0.51748	0.54679	0.57610	1.26481	10.57909	11.84390
15	0.50693	0.53339	0.55985	1.26905	9.99937	11.26843
20	0.50832	0.52814	0.54796	1.30233	9.38216	10.68449
25	0.52603	0.54133	0.55662	1.30400	8.75683	10.06083
30	0.52737	0.53602	0.54468	1.29681	8.17807	9.47488
35	0.55535	0.55738	0.55940	1.31022	7.70705	9.01727
40	0.53682	0.54192	0.54703	1.31163	7.39645	8.70808

Result – 2



Observations

- Model has basic understanding of common objects
- The output is far from appealing
- Cannot decide colors of rare objects
- Displays some color spillovers
- Small dataset cannot give out good results

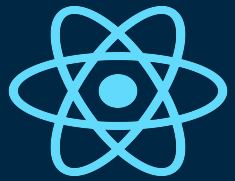


Technologies and Tools

Front-end

Back-end

Model



Web Application

- Grey images to colored
- Front-end requests backend by sending grey image in POST request
- Backend calls model to process image
- Save image to folder
- Return saved image to front

Links

Github

Demo Folder

Demo Time

THANKS FOR YOUR ATTENTION