Image Colorization Of a Photo Album by Ara Güler Using GAN

1st Ahmet Kaan Memioğlu
Engineering Department
Istanbul Kultur University
Istanbul, Turkey
1900005528@stu.iku.edu.tr

2nd Emrecan Üzüm
Engineering Department
Istanbul Kultur University
Istanbul, Turkey
1900005485@stu.iku.edu.tr

3rd Şükrü Erim Sinal
Engineering Department
Istanbul Kultur University
Istanbul, Turkey
1900003587@stu.iku.edu.tr

I. Introduction

Objective of this project is colorizing images created by Turkish Photographer Ara Guler in way that they do not look photorealistic but looking visually appealing. Therefore instead of making more closer to originality, we decided to make conspicious colored images by using G.A.N convolutional neural network.

II. Defining Model

This is the part where we teach the model how to behave when it receives an image. In summary, when a photo arrives, algorithm grayscale it and makes a prediction(colorizing) on it, then sends this prediction to the discriminator model and this time it compares the original version of the image with the version we have produced and performs a learning based on it.

A. Generator

We used PyTorch implementation of a Generator network for generating images. The Generator network has an encoder-decoder architecture. The encoder part consisting of a series of GenBlock modules that downsample the input image, and the decoder part consisting of a series of GenBlock modules that upsample the encoded image. The GenBlock module consists of several operations, including a Conv2d or ConvTranspose2d operation, a BatchNorm2d operation, a Dropout operation, and a ReLU or LeakyReLU activation. The Generator network takes an input image of size (batch size, 1, H, W) and applies the series of GenBlock modules to generate an output image of size (batch size, 2, H, W). The output image has 2 channels because it represents a grayscale image with an additional alpha channel. The final layer in the decoder part is a ConvTranspose2d operation followed by a Tanh activation, which scales the output image to the range (-1, 1).

B. Discriminator

We used PyTorch implementation of a Discriminator network for a Generative Adversarial Network (GAN). The Discriminator network takes an input image and attempts to classify it as either real or fake. The Discriminator network consists of a series of DiscBlock modules, each of which applies a Conv2d operation, a BatchNorm2d operation, and a LeakyReLU activation. The Discriminator network takes an input image of size (batch size, input channels, H, W) and applies the series of DiscBlock modules to generate an output scalar of size (batch size, 1). The output scalar represents the probability that the input image is real. If the output is close to 1, the input is classified as real, and if the output is close to 0, the input is classified as fake. The final layer in the Discriminator network is a Conv2d operation followed by a sigmoid activation, which scales the output to the range (0, 1).

C. Biases within the Model

Our Algorithms generated a visually appealing model.But within the model we have encountered some biases in specific cases which are also concurrent the Paper we have looked up to (Richard Zhang, Phillip Isola, Alexei A. Efros):
1.Tends to Overshoot colors
2.De-saturates pinkish colors.
3.Confuses red with blue time to time on bright colors with pictures that have great contrast scale.

To overcome these biases we had to implement a technique called class rebalancing.

Which essentially creates classes within the dataset and gives color ranges associated to that object.

Unfortunately this task was not achieveable from our deadline standpoint however instead we tried to tackle the problem that arose by just picking our dataset in a way that will not have a need for rebalancing while we were coloring Ara Güler's photos.

III. Data Scraping

A. Libraries

İn coding side, we used following necessary Python libraries for Neural Network, Image coloring, and train sampling, Dataset preparation etc.

- Pytorch for creating deep neural network,Activation functions,Image transformations
- PIL (Python Imaging Library) for image manipulation

Identify applicable funding agency here. If none, delete this.

- fastai for loading custom-made dataset to use in training process training
- numpy and Matplotlib for mathematical operations and visualising image channels, losses etc.
- glob for I/O operation functions

B. Scraping

The images that are going to be colored are the work of famous Turkish Photographer Ara Guler in which are taken in Istanbul around 60's. What are to be used on training, Related places,objects, animals (etc.) that is found in photos commonly. Totally in 9 category colored photos have been collected for training. "Horse, Mosque, Balat (place in Istanbul), Car, Children, Steam Train and Seaport". All of the related photos have been downloaded from internet sources via custom python function. After that, all photos that are unrelated/ futile to use in training removed one by one. At first there was 8000 photos in total. After clearing and scraping, we reached to approximately 4800 photos in total, of which are going to be used in training section.

IV. Training

A. Hyperparameters

For the hyperparameters we used in out machine learning model:

The learning rate (2e-4) determines the step size at which the optimizer makes updates to the model's parameters during training.
The number of epochs (950) is the number of times the model will cycle through the training data.
The lambda (100) hyperparameter is a coefficient for the L1 loss term in the model's loss function.
The betas hyperparameter is a tuple of coefficients used by the optimizer when computing running averages of the model's parameters.
The lossOfDiscriminator and lossOfGenerator lists are probably used to store the loss values of the discriminator and generator, respectively, after each epoch of training.

B. Input Paths

For the results and input path variables we are getting the file paths from our input and output data and checkpoint files for a machine learning model which is located in our google drive.
The input folder is the location of the data that the model will use as input. The output folder is the location where the model will save its outputs. The checkpoint paths are the locations where the model will save its trained parameters, also known as checkpoints.
The loadModel variable is a boolean value that determines whether the model should load saved checkpoints. If loadModel is True, the model will try to load saved checkpoints from the checkpoint paths.
If loadModel is False, the model will start training from scratch, without loading any saved checkpoints.

C. Checkpoints

Two functions that are related to saving and loading checkpoints in a machine learning model (SaveCheckpoint, LoadCheckpoint).

The SaveCheckpoint function takes in a model, optimizer, epoch number, and filename as input, and saves the model's state dictionary (which contains the trained parameters), optimizer's state dictionary, epoch number, and loss values for the discriminator and generator to a file with the given filename.

The LoadCheckpoint function takes in a checkpoint file, a model, an optimizer, and a learning rate as input, and loads the saved state dictionaries for the model and optimizer, as well as the epoch number and loss values for the discriminator and generator, from the checkpoint file. It also updates the learning rate of the optimizer with the given value.

D. Training Function Arguments

Our our main train function contains input arguements as follows:
-discModel: the discriminator model
-genModel: the generator model
-loader: a data loader that provides a stream of training data
-optimizerForDiscriminator: an optimizer for the discriminator model
-optimizerForGenerator: an optimizer for the generator model
-L1Loss: a loss function for computing the L1 loss
-BCELoss: a loss function for computing the binary cross entropy loss
-generatorScaler: an object for scaling the loss of the generator during training
-discriminatorScaler: an object for scaling the loss of the discriminator during training

E. Training Function Dynamic

And the Trainfunction we have implemented works within a loop it iterates over the data in the data loader for each iteration:
1.Computes the output of the generator model (YGenerated) given the input (L)
2.Computes the output of the discriminator model (discReal) given the true data (L and ab) and computes the real loss (discRealLoss) using the BCELoss function
3.Computes the output of the discriminator model (discGenerated) given the generated data (L and YGenerated) and computes the generated loss (discGeneratedLoss) using the BCELoss function
4.Computes the average of the real and generated losses as the discriminator loss (discriminatorLoss) 5.Backpropagates the discriminator loss and updates the discriminator's parameters using the optimizerForDiscriminator
6.Computes the output of the discriminator model (discGenerated) given the generated data (L and YGenerated) and

computes the generated loss (genGeneratedLoss) using the BCELoss function
7.Computes the L1 loss (L1) between the generated output and the true data using the L1Loss function, and scales it by the hyperparameter LAMBDA
8.Computes the sum of the generated loss and L1 loss as the generator loss (generatorLoss)
9.Backpropagates the generator loss and updates the generator's parameters using the optimizerForGenerator

F. *How it works*

This function also uses torch.cuda.amp.autocast() and torch.cuda.amp.GradScaler to enable automatic mixed precision training.
This can help speed up training by using lower-precision floating point numbers to represent the model's parameters and activations during training, while still maintaining the model's accuracy.

Our model creation and training loop also appears to be a combination of a discriminator and a generator, and both are trained in each iteration of the loop. The loop has several conditions:

TRAIN: a boolean variable that determines whether the loop should continue running. If it is True, the loop will continue running. If it is False, the loop will terminate.

epoch equal or less than EPOCHS: another condition that determines whether the loop should continue running. The loop will continue running as long as the epoch number (which starts at 1 and is incremented by 1 at the end of each iteration) is less than or equal to the total number of epochs specified by the EPOCHS hyperparameter.

Inside the loop, the following actions are performed:

If the visualizeWhileTraining flag is True, the ShowSamples function is called to visualize the model's output on the validation data, and the VisualizeLoss function is called to visualize the loss values of the generator and discriminator.

If the saveModel flag is True, the model's parameters and other relevant information are saved to checkpoint files using the SaveCheckpoint function.

The model is trained on the training data using the TrainFunction function. The epoch number is incremented by 1. This loop will continue running until one of the conditions for the loop is no longer met (e.g. TRAIN becomes False or epoch greater than EPOCHS).

V. Results

Although much more training and visuals were used on the Pretrained Model than our model, we achieved a break-even success. It can be said that we owe the realism of these visuals, for which we make predictions, to the model we produced. Because we created this model with images and objects that were completely targeted towards our goal (to colorize Ara Güler's album). This specially selected model showed us its ability to predict the objects it was previously trained on while coloring Ara Güler's album. Thus, we now have a model that we can develop, a colored album.

VI. Future Works

A. *Survey*

After making color predictions on the images, we compared the images we trusted the most (train, fisherman, harbor) with the images we colored with the pretrained model and asked our users. Our question in the survey we started with about thirty people was which one was more "preferable". As a result of the survey, we observed that the pretrained model filled the image with colors better, but it did not capture the feeling of realism as much as we did. Because we created our data set only on the objects in Ara Güler's photographs. In the survey, we achieved about eighty percent (24/30) success in realism.

B. *Train the Pretrained Model*

Another important thing we will do in the future of the project is to add the visuals from the model we created on the pretrained model.

References

[1] "Gan training — machine learning — google developers," Google. [Online]. Available: https://developers.google.com/machine-learning/gan/training. [Accessed: 06-Jan-2023].
[2] Richard Zhang - Research scientist, Adobe Research. [Online]. Available: https://richzhang.github.io/colorization/resources/imagenet_comparison.html. [Accessed: 06-Jan-2023].
[3] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," arXiv.org, 05-Oct-2016. [Online]. Available: https://arxiv.org/abs/1603.08511v5. [Accessed: 06-Jan-2023].