Project Report : CS 7643

Albert Chen
albertchen@gatech.edu

Kamal Feracho
kferacho3@gatech.edu

Han Mai Nguyen
hnguyen402@gatech.edu

Taichang Zhou
tzhou318@gatech.edu

Georgia Institute of Technology

Abstract

Modifying an image can be done in various ways using a plethora of image editing, computer vision, and machine learning related techniques. Grayscale photos tend to be less vocal than colored photos, in which details from photos dating years ago have been lost due to lack of colorization. In this project we explored methods that can hallucinate a potential "colored" version of the gray scale image with intentions of matching the ground truth images. We selected a few models from existing literature and trained them with a standard dataset and loss function to compare their performances. One is a CNN based on Resnet-18 and the other is a C-GAN. Our experiments show the GAN model performs better.

1. Introduction/Background/Motivation

(5 points) What did you try to do? What problem did you try to solve? Articulate your objectives using absolutely no jargon. In this project, we want to achieve high quality colorizing of gray-scale images with deep learning networks.

Through extensive experiments with the COCO, we plan to optimize a recently established model using image colorization with an optimized conditional GANs to boost downstream model performance in hopes of getting better results. The Generator is created using a U-Net model. The code makes the U-Net from the middle part of the figure below, as it curves down in the U shape, and adds down-sampling and up-sampling modules to the left and right of that middle module (respectively). This is done for each iteration until it reaches the input module and output module. The blue rectangles represents the ordering the modules are built in. The layers depicted in the image is minimal as our code will create more.

(5 points) How is it done today, and what are the limits of current practice? The state of the art image colorization use deep learning methods such as feed forward convolutional neural networks and generative adversarial networks. The limits of these models are their need for vast and varying amounts of data, compute, and tuning. Different methods may perform differently given different contexts in the images, which is why training with a balanced dataset is important.

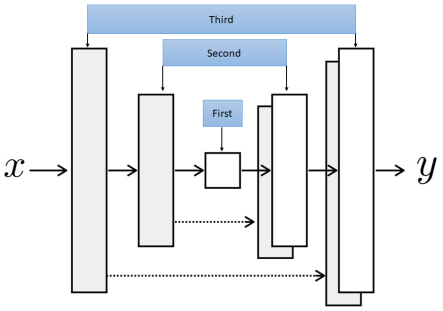

Figure 1. Architecture of our Model

The primary difference between colorization algorithms in the past and now derives from the ways data is obtained and treated for modeling the gray-scale-to-color methods, which resembles that of a non-parametric methodical approach. Non-parametric and parametric methods best generalizes the type of approach used to colorize images, in which our optimized model along with other concurrent projects adopt in their approach to the problem. Non-parametric methods define one or more color reference image(s) to be used as the source data. Following, the color is transferred onto the input image from analogous religious provided from the reference image.

Parametric methods learn prediction functions from large datasets of color images all during training time, which implies the problem is either regression onto a continuous color space or classification that uses quantized values for the RGB space. Current models use significantly more data than previous models due to larger models and more data to train with. This, along with optimized loss

functions are paired to create a final and continuous output. Other projects have developed similar systems using various CNN models but with different architectures and data sets used to train their models.

There is a lack in depth on the answer to what it is that each of the various experiments that involves color restoration needs to do to improve. We will try optimizing the loss functions by tweaking the parameters as we see fit. We intend to create two or more models that run off of the same dataset and a similar loss function with a finalized comparative study to see if our methods improved the results and further discussions.

(5 points) Who cares? If you are successful, what difference will it make?

Currently there are many methods that do gray scale image colorizing very well, we seek to run a comparative study on several models with controls such as a standard dataset, length of training, and loss function. Understanding how these models perform under these limitations can reveal properties of these models that may be helpful in future investigations. The ultimate goal of this problem is to achieve image results and outputs that are fully accurate in the colorization of images. Achieving this accuracy will allow us to pass in grayscale images from years ago that were never converted in hopes of getting an accurate colorization to "restore" the image to match the coloring of the modern cameras. Even with a slight improve the results, we will be able to provide potential avenues for future projects to explore and expand upon to help achieve the ultimatum.

(5 points) What data did you use? Provide details about your data, specifically choose the most important aspects of your data mentioned here. You have to choose all of them, just the most relevant.

In this project, we used the Common Objects in Context (COCO) dataset. It is a object detection, segmentation, and captioning dataset. The reason we choose COCO includes its diversity in objects and scenes that have a variety of colors. We will not be using the segmentation, caption, or category labels that come with the dataset; we will only be using the color images. The dataset has 330K images, but our experiments will only be using a subset of the COCO dataset from "fastai" that consists of 21837 images. This was done to make the data manageable.

2. Approach

(10 points) What did you do exactly? How did you solve the problem? Why did you think it would be successful? Is anything new in your approach?

As stated before, our goal is to optimize current models by generalizing two loss functions between a CNN based model and a C-GANs based model (C-GANs = Conditional GANs model). We decided on this approach because we thought we would be able to demonstrate the effectiveness of a GAN in producing more realistic outputs compared to a plain CNN. We will focus our efforts first on optimizing the GANs model implemented in the paper referenced below. We can improve GAN by turning our attention in balancing the loss between the generator and the discriminator. Unfortunately, the solution seems elusive. We can maintain a static ratio between the number of gradient descent iterations on the discriminator and the generator. Most typical GANs have a generator and discriminator model which both work together in learning the problem.

The generator is responsible for taking in a 1-channel grayscale image to produce a 2-channel image, denoted 1* and 2* respectively. The discriminator comes in and takes in 1* and 2*, along with some ground truth images from another 3-channel that uses the LAB color space and concatenates them together with the input image (grayscale) and determines if for some new 3-channel image whether 3* is real or fake. The 3-channel LAB images are trained and learned by the discriminator so it recognizes which images are real. The "condition" in our case would be that of the grayscaled images which the generator and discriminator both see as the condition for the models in the GANs in which we expect this condition to be considered. The loss function for our Conditional GANs model will be adopted from on of the experiments referenced below:

x as the grayscale image z as the input noise for the generator y as the 2-channel output we want from the generator (it can also represent the 2 color channels of a real image) G is the generator model D is the discriminator

$$\mathcal{L}_{cGAN}(G,D) = \mathbb{E}_{x,y}[\log D(x,y)] + \mathbb{E}_{x,z}[\log(1 - D(x, G(x,z))]$$

Figure 2. First Formula

We can also include an additional L1 loss function, which by itself is capable of learning how to color in the images in a conservative color consisting mostly of "brown" and "grays" as it uses the average of colors given a worse case scenario. The L2 loss (or mean squared error) is not efficient either as it does the same as L1 but yields more grayish results in the average which is the opposite of our intentions. Combining both functions yields:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}[\|y - G(x,z)\|_1]$$

Figure 3. Second Formula

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G,D) + \lambda \mathcal{L}_{L1}(G)$$

Figure 4. Final Combined Formula

First adjustment we made was normalizing the parameters (image values) to a value between -1 and 1 and have Tanh as the last layer of the generator output. This is due to the fact that when generating the images, we normalized them making all images fall in range of [-1, 1] so we want to ensure the outputs follow the same. ReLu can be used but Tanh guarantees non-negative values.

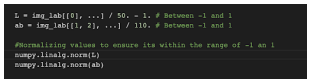

Figure 5. First Implementation

```
criterion = nn.L1Loss().Tanh()
```

```
#Flipped labels Real = Fake to Fake = real
#L, ab = data['L'].to(device), data['ab'].to(device)
ab, L = data['ab'].to(device), data['L'].to(device)
```

We avoided the use of Sparse Gradients such as ReLU and MaxPooling since they cause conflicts with the stability of the GANs. This was achieved by ensuring none were used in the U-Net model and instead were replaced by something appropriate for GANs models. We used methods of LeakyReLU to replace ReLU in both our Generator and Discriminator. For downsampling, we ensured that only Average Pooling and Conv2d + stride was used. For upsampling, we made sure to use ConvTranspose2d + stride.

We also updated the labels to either make them more soft or noisy by changing the values of "Real =1 Fake =0" to ranges 0.7-1.2 and 0.9-0.3 respectively [2]. For the discrim-

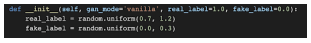

Figure 6. Second Implementation

inator, noise was added by including an if statement for randomness that would alternate the labels Finally, Gaussian

```
#random noise added for label swaps
ran = random.uniform(1)
if ran < 0.5:
  Ls, abs_ = data['L'], data['ab']
else:
  abs_, Ls = data['ab'], data['L']
print(Ls.shape, abs_.shape)
print(len(train_dl), len(val_dl))
```

Figure 7. Third Implementation

noise was added to the discriminator as per [3].

```
  #Added Gausian Noise to generator
  if norm: layers += [nn.BatchNorm2d(nf)]
  layers = GaussianNoise(layers)
  #LeakyReLU and Batch Norm check
  if act: layers += [nn.LeakyReLU(0.2, True)]
  return nn.Sequential(GaussianNoise(*layers))
```

Figure 8. Final Implementation

(5 points) What problems did you anticipate? What problems did you encounter? Did the very first thing you tried work? One of the problems biggest problems we believe we will encounter will include loading and properly training the optimized data set and also potentially yielding underwhelming results and trying to figure out how we can improve upon them. Our initial strategy involved parameter tuning, which was not as successful as we'd hoped. We then applied to optimized GANs model from the paper into our main notebook we decided to adopt. Applying the modified GANs yielded significant results and changes indicating that the loss function is what we need to focus our time into analyzing and improving to get the desired results. Another problem comes from the architectures used within the google colab in which it requires lots of storage and high GPU processing in order to train the data in a timely matter, which otherwise takes quite some time. During our training process for the DISCRIMINATOR model, our colab notebook often crashed or ran out of GPU, etc. which led us to reduce the epoch by 0.75 (from 100 epochs to 25). The most successful runs were saved where we have a notebook that made it to 42 epochs, one that completed 25/25 epochs (but yielded errors when training the GENERATOR), and a final run that was successfully completed. The final run couldn't be trained with the same set of images as the failed runs as there was a bug that called for restarting the entire model and removing the saved trained weights. We can see when comparing the results between the three that some were significantly better than others, which has to do with the tuning and randomized value switches. We could place print statements to indicate what specifically happened during each change to replicate those exact runs in hopes of getting the same good results.

Important: Mention any code repositories (with citations) or other sources that you used, and specifically what changes you made to them for your project.

3. Experiments and Results

(10 points) How did you measure success? What experiments were used? What were the results, both quantitative and qualitative? Did you succeed? Did you fail? Why? Justify your reasons with arguments supported by evidence and data.

To measure our success, we devised a comparative study between the two models we optimized, the original models, and the ground truth values. Based off the paper in "Colorizing black white images with U-Net and conditional GAN — A Tutorial", we optimized the GANs model found within our adopted model by implementing their version of the loss function (a GANs loss function resembling C-GANs modeling) along with appropriate parameter tuning. After implementing all of the necessary parameter tunings, we will then train and run the newly "optimized" model in hopes of it making slight differences in the out puts. We will conclude by utilizing.

To measure our results, we a printed out a set of images (ones being trained through the model) during each epoch, in which the first set represents the gray scaled images, the second set represents the currently modified images, and the third set represents the ground truth set of images. We will post our last iterations of running our optimized generator below. In the notebook within the zip file, there is a photo of every iteration to show how images trained gradually get closer and closer in resemblance.

We understood our intentions of implementing all of the optimizations, which are each detailed within the Approach section. There were many other things we wanted to implement but were running into bugs and simply didn't have enough time. For example, we wanted to perform the Genetic Algorithm which is a type of learning algorithm that uses the idea that crossing over the weights of two good neural networks will result in a better neural network.

The reason that genetic algorithms are so effective is because there is no direct optimization algorithm, allowing for the possibility to have extremely varied results. Additionally, they often come up with very interesting solutions that often give valuable insight into the problem. We failed to implement this because either A: our weights would fail to save going into the next step or we would get the the algorithm and have many bugs which caused us to scrap the algorithm altogether.

The following figures will represent the results of the final recorded iteration for each of our failed attempts and our successful attempt. The results will be shown for the final iteration of the Discriminator model and the Generator model (if applicable). The first two failed runs only made it through training the Discriminator models.

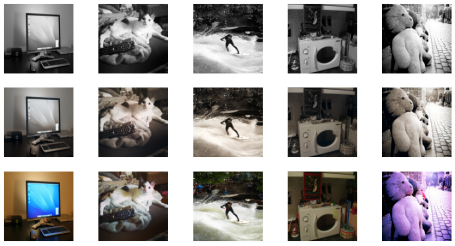

Figure 9. Failed Discriminator 1a

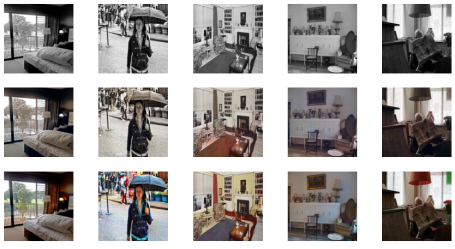

Figure 10. Failed Discriminator 1b

The first failed run was intended to last for a total of 100 epochs, in which the last recorded epoch was 42/100. The actual notebook was running for approximately 8.5 hours and only actually made it to epoch 56 but it was not saved or recorded.

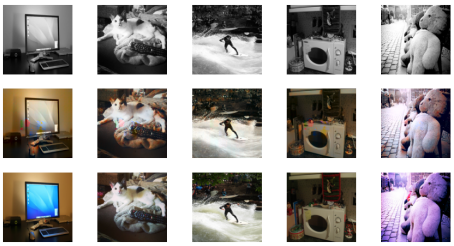

Figure 11. Failed Discriminator 2a

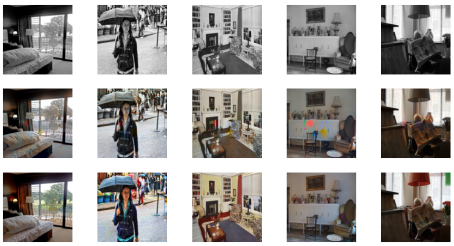

Figure 12. Failed Discriminator 2b

The second failed attempt was only ran for 25 epochs for the Discriminator as opposed to 100, in which the results were better than that of the previously ran model. This indicates (as stated before) that the parameter hypertuning and optimization we performed on the model works but varies (in which we need print statements to find what values truly optimize the model).

Figure 13. Original Discriminator

Above, we can see the original results and will compare them to that of ours from the models we optimized.

The Discriminator in the final run had the most promising results despite using different images from the first two failed models. On top of it, we reduced the epochs by 0.75 indicating that the result would be even greater had we have the time to allow more training for this portion of the model.

The final output of our Generator demonstrates that the optimizations of our models worked sufficiently. We believe that if we were to train the Discriminator for a longer

Figure 14. Original Generator

Figure 15. Successful Discriminator 1a

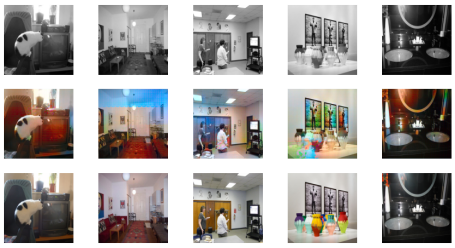

Figure 16. Successful Discriminator 1b

Figure 17. Successful Generator 1a

Figure 18. Successful Generator 1b

time that the results would have been even better than the original results from the adopted model.

We also went further into the topic and briefly experiment with the Zhang's model and were able to implement some of his insight and develop a brief approach to image colorization using Convolutional Neural Networks [4]. We use a number of convolutional layers to extract features from the datasets, then continue to apply deconvolutional layers with the purpose increasing the spacial resolution of our features.

The first half of model will be ResNet-18, which is an image classification network consists 18 layers and residual connections. First layer of the network will be modified to grayscale input, and afetr the 6th set of layer, it will be cut off. The seccond half of the net consists the deconvolutional layers. Below is the architecture followed by the image results.

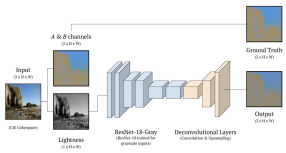

Figure 19. CNN and ResNet-18 Implementation Architecture

Figure 20. CNN and ResNet-18 Implementation Results

Important: This section should be rigorous and thorough. Present detailed information about decision you made, why you made them, and any evidence/experimentation to back them up. This is especially true if you leveraged existing architectures, pretrained models, and code (i.e. do not just show results of fine-tuning a pre-trained model without any analysis, claims/evidence, and conclusions, as that tends to not make a strong project).

4. Structure

A generalized structure of how a conditional GANs model works. The figure below provides an accurate representation of how our model works as we use the DISCRIMINATOR first to pre-train our model. Then we pass in the results denoted 'z' into our GENERATOR model with the same weights we used for our discriminator.

5. Conclusion

In this paper, we have explored colorization of grayscaled images by implementing and optimizing a GANs model that uses a pretrained U-Net as its Generator,

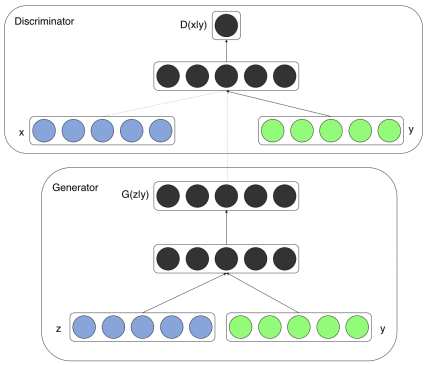

Figure 21. Structure of Conditional GANs Models

and an implemented Discriminator. We faced many challenges such as GPU and computationally expensive methods that either crashed our training process or were hard to debug. After multiple long and failed attempts, we managed to have one successful attempt at the cost of the training time. We were still able to yield promising results which indicates that with proper training time, our optimized model could potentially be better than that of the adopted model.

Overfitting was not an issue at all as we significantly reduced the training time which in conjunction reduced our chances of the model memorizing the noise. Otherwise it would've fit too closely to the training set and the model will be unable to generalize well to new data. If a model cannot generalize well to new data, then it will not be able to perform the classification or prediction tasks that it was intended for.

We hoped to have been able to run the models we optimized as we intended (fully pre-trained weights, correct number of epochs, etc.) but was burdened by the models computational expense and our lack of powerful GPUs. With that, we made the necessary adjustments to get a finalized and working model, which performed very well given its constraints when compared to the adopted model.

Briefly discuss potential future work that the research community could focus on to make improvements in the direction of your project's topic.

If anyone continuing any research on this project can pick up where we left off by successfully implementing all of the additional GANs related optimizations and with a more powerful GPU and processor to better train the discriminator.

(5 points) Appropriate use of figures / tables / visualizations. Are the ideas presented with appropriate illustration? Are the results presented clearly; are the important differences illustrated?

(5 points) Overall clarity. Is the manuscript self-contained? Can a peer who has also taken Deep Learning understand all of the points addressed above? Is sufficient detail provided?

(5 points) Finally, points will be distributed based on your understanding of how your project relates to Deep Learning. Here are some questions to think about:

What was the structure of your problem? How did the structure of your model reflect the structure of your problem?

Our problem, colorizing gray images, require the model to understand the content in the image and detect the objects in the image, i.e. the boundary of contents should be detected and grass should have different color to the sky. Our model in C-GAN uses generator and discriminator, where generator can embed the image by convolution layers and up-sampling for object detection and generate two channels of the fake graph. For the discriminator, it tries to find the difference between the generated LAB images and the real ones by Conv-BatchNorm-LeackyReLU sections. For the CNN model, we are going to use the convolutions to understand the image and use up-sampling to generate graphs.

What parts of your model had learned parameters (e.g., convolution layers) and what parts did not (e.g., post-processing classifier probabilities into decisions)?

In the C-GNN model, it will learn parameters in the convolution layers in both generator and discriminator, for generator, it will also learn parameters in the up-sampling part. For other parts like batch normalization, activation, and L1 loss, there is no parameters learned. In the CNN model, it will learn parameters in the convolution and up-sampling layers, but no parameters learned in the batch normalization and LeakyReLu.

What representations of input and output did the neural network expect? How was the data pre/post-processed? What was the loss function?

For the C-GNN model, the input for the generator is an one-channel gray scaled image, and it will output a 2-channel image which are A and B channels in the LAB. Those output 2 channels will be composed with the original gray image to form a 3-channel LAB image as the input of the discriminator. The discriminator will output a label to show that image is real or not. For the CNN model, it takes the gray image as input and will output a 3-channel image. Did the model overfit? How well did the approach generalize?

What Deep Learning framework did you use?

Both the Resnet-18 CNN and C-GAN implementations used pytorch.

What existing code or models did you start with and what did those starting points provide? We used existing implementations of the Resnet-18 CNN[2] and C-GAN colorizing models[4].

[3] [5] [6] [4] [1] [7] [2]

References

[1] Gregory Shakhnarovich Gustav Larsson, Michael Maire. Learning representations for automatic colorization. pages 1–29, 2017. https://arxiv.org/pdf/1603.06668.pdf. 4

[2] Luke Melas-Kyriazi. Image colorization with convolutional neural networks. 2018. https://lukemelas.github.io/image-colorization.html. 4

[3] Adrian Rosebrock. Black and white image colorization with opencv and deep learning, 2019. https://pyimagesearch.com/2019/02/25/black-and-white-image-colorization-with-opencv-and-deep-learning/. 4

[4] Ivan Miguel Pires Petre Lameski Sonja Gievska Sandra Treneska, Eftim Zdravevski. Gan-based image colorization for self-supervised visual feature learning. *MDPI*, pages 1--17, 2022. https://doi.org/10.3390/s22041599. 4

[5] Moein Shariatina. Colorizing black white images with u-net and conditional gan | a tutorial, 2020. Supplied as additional material https://towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8. 4

[6] Victor Sim. Using genetic algorithms to optimize gans, 2020. Supplied as additional material https://towardsdatascience.com/using-genetic-algorithms-to-optimize-gans-c64e5e02ead4. 4

[7] Wojciech Zaremba Vicki Cheung Alec Radford Xi Chen Tim Salimans, Ian Goodfellow. Improved techniques for training gans. 2016. https://arxiv.org/abs/1606.03498. 4

6. Repositories Used

1.https://github.com/moein-shariatnia/Deep-Learning/tree/main

2. https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix

3. https://github.com/soumith/ganhacks

7. Work Division

Student Name	Contributed Aspects	Details
Albert Chen	Implementation and Experiment Setup	Created the data sourcing logic to standardize the training and validation data for comparing models. Implemented the convolutional neural network model with deconvolutional layers.
Kamal Feracho	Optimized GANs and U-Net Models	Trained COCO dataset with optimized GANs Discriminator and Generator. Analyzed results and compared our model to adopted model.
Han Mai Nguyen	Implementation and Experiment Setup	Contributed to implement the convolutional neural network model with deconvolutional layers. Zhang's model investigation and experimentation
Taichang Zhou	Train and Analysis	Trained the GNN model and tried several parameters, and analyzed the results. Compare the results from different models.

Table 1. Contributions of team members.