CSCI 1430 Final Project Report: Colorization: Using Deep Learning to Colorize DigiFace Images

Colorizers: John Chung, Pradyut Sekhsaria, Aayush Setty, Mingchao Zhang.
TA name: Emily Wang. Brown University

Abstract

Given a black and white image of a synthetic face from DigiFace-1M, we recreate a plausible colored version of it. To carry out this task, we create three models: a CNN, a combined model using ResNet-50, and a GAN. We evaluate each model using mean squared error (MSE) and a custom threshold loss.

1. Introduction

Traditionally, colorizing historical black and white images is done manually and demands significant time and research. A single image can take up to a month to colorize as faces alone require 20 layers of pink, blue, and green shades.[1] Given the massive amount of historical black and white photographs, the colorizing community could benefit from deep learning methods as they can automate and accelerate the colorization process.

For our data, we used DigiFace-1M, a dataset of over one million images of synthetic faces. Although facial recognition models are typically trained on images of real faces, we used DigiFace because of ethical issues and data bias. Most face datasets are obtained by aggregating images of people online without their consent. Furthermore, these datasets typically contain images of celebrity faces, which are taken with strong lighting and makeup, and lack a balanced racial distribution. However, DigiFace images are generated from head scans of people who have given their consent and have a balanced racial distribution. Using this data, we trained three models: a CNN, a combined model using ResNet-50, and a GAN.

2. Related Work

We referred to lecture slides and past assignments, published papers and online blogs, and Tensorflow documentation to design our models. Specifically, we referenced Zhang, Isola, Efros's paper[2] on colorization with CNNs to guide the design of our CNN models, their loss functions, and how to measure accuracy. Additionally, Luke Melas' [3] blog post helped us understand how to incorporate the LAB color space and ResNet-50 into our CNN model. When designing and implementing our GAN, we referred to Tensorflow's tutorial on GANs[4] to develop a training and testing pipeline and Moein Shariatnia's[5] blog post for our generator loss function.

While there exists significant research on colorization, we did not see work specifically for DigiFace images, let alone real faces. Therefore, our project builds on previous research to develop models that specialize in colorizing images of synthetic faces.

3. Method

3.1. Pre-processing

DigiFace is split into two parts. The first part contains 720K images with 10K identities (72 images per identity), and the second part contains 500K images with 100K identities (5 images per identity). We downloaded the second split because we wanted our models to train on as many identities as possible. To avoid bias, we randomly sampled one of the five images for each identity. Therefore, we reduced our dataset to 100K images, where each image represents a unique identity.

The color values of the original images were stored as RGBA, so we first converted them to RGB, then converted the resulting RGB images to LAB. We then randomly shuffled the images, and performed a 80-10-10 split to create

[1] Wallner, E. (2022, August 27). How to colorize black and white photos with just 100 lines of Neural Network Code. Medium. https://emilwallner.medium.com/colorize-b-w-photos-with-a-100-line-neural-network-53d9b4449f8d

[2] Zhang, R., Isola, P., and Efros, A. A. (n.d.). Colorful Image Colorization. https://doi.org/10.48550/arXiv.1603.08511

[3] Luke Melas. Image colorization with convolutional neural networks. (n.d.). https://lukemelas.github.io/image-colorization.html

[4] Deep convolutional generative Adversarial Networ: Tensorflow Core. TensorFlow. (n.d.-a). https://www.tensorflow.org/tutorials/generative/dcgan

[5] Shariatnia, M. (2020, November 18). Colorizing Black and white images with U-Net and conditional Gan–A tutorial. Medium. https://towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8

training, validation, and test sets. Finally, we stored each set in its own pickle file so that we could efficiently share the data with our team.

3.2. Accuracy

We used two metrics to measure accuracy. The first was a simple L2 loss that measured the MSE between each pixel in the a* and b* channels of the predicted image and the ground truth image. Our second metric was a threshold loss that measured the percentage of the pixels in the predicted a* and b* channels that were within a certain range/threshold of their corresponding ground truth values. We used this metric because colorization is inherently subjective, and images whose pixel values are not exactly the same as their ground truth counterparts (but close enough) may still appear "real" to a human. In our loss function, we used a threshold of 10, which is about 4 percent of the range of possible values (-128 to 127) for the a* and b* channels.

3.3. CNN

We began by creating a simple CNN based on an encoder-decoder architecture. Our model first applies a series of convolutional layers to extract features from the L channel, then a series of deconvolutional layers to upsample the resulting features to form the a* and b* channels. Initially, our model underperformed because its last layer was a convolutional layer. However, adding several fully-connected dense layers significantly boosted performance. Additionally, we added a batch normalization layer betweeen every two convolutional layers to combat overfitting. Figure 1 shows the architecture of our final CNN model.

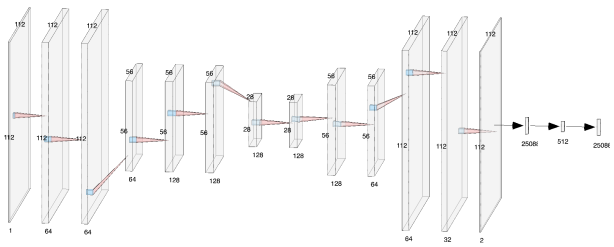

Figure 1. Architecture of naive CNN.

3.4. ResNet-50 and CNN

We then replaced the encoder of our naive CNN with ResNet-50 for better feature extraction. In this model, we passed the L channel as an input, with ResNet-50 returning 2048 4x4 features, which our decoder uses to reconstruct the a* and b* channels.

Unfortunately, this model significantly overfits to the training data. To combat overfitting, we first tried regularization by experimenting with dropout layers at various locations within the architecture. After this attempt failed, we then tried standardization. Specifically, we re-processed the data but divided the a* and b* channels by 128 to restrict the range between -1 and 1. Additionally, we updated the last layer to use the tanh activation function because it returns a value between -1 and 1 for any input. However, this resulted in worse performance. Therefore, we continued with our initial model whose architecture is shown in Figure 2.

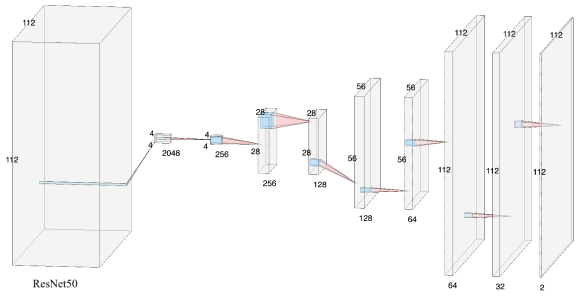

Figure 2. Architecture of model using ResNet-50 as encoder (GAN generator).

3.5. GAN

The output images of both the naive CNN and ResNet-50 CNN were quite muted and lacked vibrant colors. We speculated that these models were conservative with their color predictions because of their loss functions (L2 loss), which penalize large deviations. To combat this effect, we implemented a GAN, which has two components: a generator and a discriminator.

3.5.1 Generator

Given the L channel, the generator outputs its predictions for the a* and b* channels. For our generator, we used our ResNet-50 CNN model described in Section 3.4. The only difference was the loss function; our generator loss function has two components:

$$\mathcal{L}_{generator} = \text{Cross-Entropy} + \lambda L_2$$

The Cross-Entropy function takes the output of the discriminator on the generator's images (the probability that the generated images are real) and computes the cross-entropy of that probability with a probability of 1. Our loss function then adds the resulting cross-entropy to the L_2 loss of the generator's predicted images (a* and b* channels) with the real images. Initially, we used the L_1 loss. However, we found that this was not sufficiently penalizing large fluctuations from the real values, and the model was taking extremely long to converge.

λ is used to combat the masking effect that the L_2 loss has on the cross-entropy loss. Initially, we simply added the L_2 loss to the cross-entropy. However, we noticed that the results

of our GAN were extremely similar to our previous model. We realized that while the cross-entropy loss was between 0 and 1, the L_2 loss was typically above 50. Therefore, the effect of the cross-entropy loss was hidden by the large magnitude of the L_2 loss. Setting $\lambda = 0.1$, resulted in better performance.

3.5.2 Discriminator

The discriminator attempts to distinguish fake images produced by the generator from real images. For our discriminator, we implemented a CNN that takes in all three channels of an image and outputs the probability that it is real. Figure 3 shows the architecture of our discriminator.

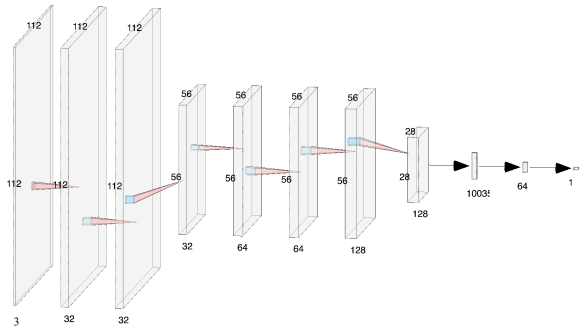

Figure 3. Architecture of GAN discriminator.

Initially, the input of our discriminator was just the a* and b* channels. However, we updated our model to intake the entire image because the discriminator needs all three channels to best assess the probability an image is real.

Using cross-entropy loss, our discriminator loss first compares the discriminator's probabilities on real images to a probability of 1, then compares its probabilities on the generator's (fake) images to a probability of 0. Our discriminator loss then adds the resulting cross-entropy losses.

3.5.3 Training

We trained our GAN for 50 epochs, which entailed alternating training between the generator and discriminator. Initially, we trained both the generator and the discriminator on the same batch. However, we learned that this made training extremely erratic as our partially trained models were affecting each other. Therefore, we pivoted to training the generator and discriminator individually, swapping the two at every epoch.

4. Results

Figure 4 shows a comparison of a random sample of true images from the test set and the images predicted by each of our models. For the most part, it seems that all three models decently colorize. The ResNet-50 CNN seems to perform slightly better than the naive CNN. For example, the output of the image in the third column for the ResNet-50 CNN better captures the person's skin tone and includes blue and green shades in the background. As hypothesized, our GAN seems to have the best performance. The GAN's output images in the first and third columns have the most vibrant colors compared to those produced by the other models. However, like the other models, the GAN is still conservative with its color predictions as some images appear muted and contain gray areas. Interestingly, all of our models incorrectly predict the person's hair in the image in the first column to be blonde. We speculate that our models poorly distinguish hair from a person's face, which causes them to color the two features a similar shade. For future work, we would investigate how our models would perform if we provided them hints to the colors of some features.

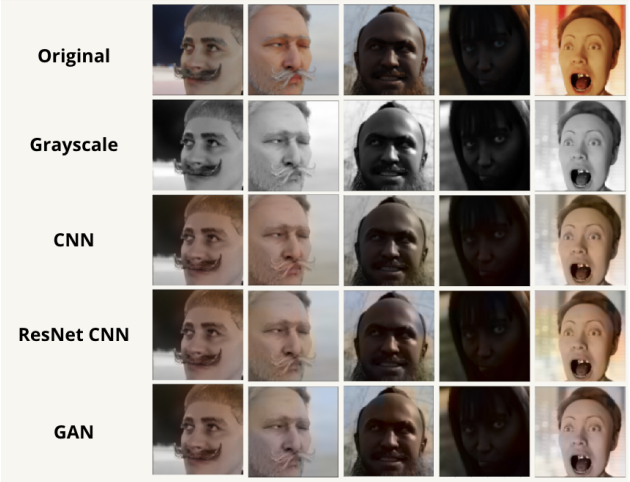

Figure 4. Comparison of true images and images predicted by our models.

4.1. Technical Discussion

Although we had a dataset of 100K images, we were only able to train each model on 20K images for 50 epochs because of limits on computation and time imposed by Google Colab. We suspect that training our models on the entire dataset for more epochs would lead to significant improvements in accuracy.

Additionally, the grayish color values of our models' predictions may be a consequence of our loss function. Since MSE heavily penalizes large deviations, our models may have been encouraged to choose average values. Figure 5, which shows the MSE and threshold accuracy curves for each of our models, provides further evidence that our evaluation metrics are imperfect. We see that all three models have similar values for MSE and threshold accuracy after 50 epochs, yet they predicted different color values across

images.

Furthermore, we see that the way each model's accuracy changed at each epoch was different. While both the naive CNN and ResNet-50 CNN show smooth curves, the GAN follows an uneven training path since it trained two models simultaneously. For future work, we consider making training the discriminator more sparse by training it once every few epochs the generator is trained.

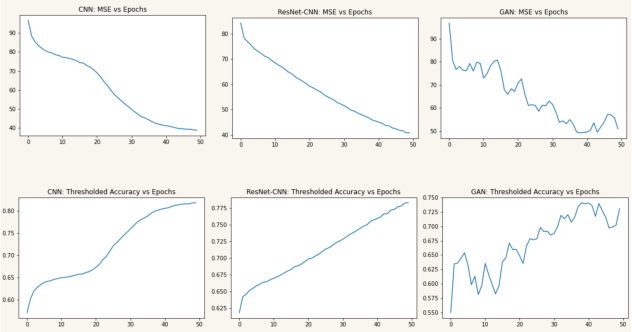

Figure 5. MSE and threshold accuracy curves. *Top*: MSE; *Bottom*: Threshold accuracy; *Left to right*: CNN, ResNet-50 CNN, GAN

Lastly, while the GAN took significantly longer to train than both the naive CNN and ResNet-50 CNN (twice as long per epoch), its results were only marginally better. Therefore, with constraints on time and resources, using the simpler CNN models may be better in practice.

4.2. Socially-responsible Computing Discussion via Proposal Swap

Unfortunately, our proposal did not get critiqued due to a Gradescope error. Instead, we came up with our own SRC concerns and rebutted them.

One of the big critiques on the setup of our project is the presence of possible racial bias in our dataset and how that relates to the colorization results. A fair number of existing large-scale datasets of face images is largely biased in the context of our task. Some datasets have an uneven distribution of people of different colors, races, or ages. Some other datasets may contain images with bad lighting and noises that come from the background. Having these potential dataset issues in mind, we were able to find a perfect dataset, DigiFace-1M, that has very diverse images in terms of skin colors and races, and since images are synthetic and generated by another program, every image captures facial features clearly and has little background noise. As a result, our model does equally well for faces of different colors, shades or ages.

Nevertheless, there could still be some subtle problem with this setup. Our models generate great results for test images, but these test images also come from DigiFace-1M, which has little background noise. In the real world, the colorization task often is applied to real face images that can have a lot more noise and the faces in them may not be perfectly centered. There might be some more factors we need to keep in mind when designing the models if the testing dataset is different.

5. Conclusion

Our models can adequately colorize black and white images of synthetic faces. With more tuning and training, our models could potentially extrapolate their results to images of real faces. Such a model could be used to colorize historical photographs, which could reveal details within them that were previously unseen. Furthermore, people relate better to colored images because black and white photographs feel old and distant. Through colorization, our project could help people connect with the past and preserve the stories and lessons behind historical images.

References

[1] Wallner, E. (2022, August 27). How to colorize black and white photos with just 100 lines of Neural Network Code. Medium. https://emilwallner.medium.com/colorize-b-w-photos-with-a-100-line-neural-network-53d9b4449f8d

[2] Zhang, R., Isola, P., and Efros, A. A. (n.d.). Colorful Image Colorization. https://doi.org/10.48550/arXiv.1603.08511

[3] Luke Melas. Image colorization with convolutional neural networks. (n.d.). https://lukemelas.github.io/image-colorization.html

[4] Deep convolutional generative Adversarial Network: Tensorflow Core. TensorFlow. (n.d.-a). https://www.tensorflow.org/tutorials/generative/dcgan

[5] Shariatnia, M. (2020, November 18). Colorizing Black: white images with U-Net and conditional Gan–A tutorial. Medium. https://towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8

Appendix

Team contributions

John I performed pre-processing as described in Section 3.1. Additionally, I helped design and implement the architectures and training and testing pipelines for both the ResNet-50 CNN and GAN with Pradyut.

Pradyut Adapted the code from HW5 to work for our project (including the system to save checkpoints etc.). Found relevant papers/blogs that we could use to design

our models. Along with John, designed and coded the entire ResNet-50 and GAN models, and wrote all the code to train and test our models. Collected and plotted the data for our various charts.

Aayush Helped adapt HW5 skeleton code to fit our project. Helped build, train, and fine tune the CNN architecture on google colab. Researched and wrote the custom thresholded accuracy metric function.

Mingchao Involved in the choosing the loss function and the threshold loss function. Worked on the CNN model, researched on how different types of layers affect the model performance and tuned the architecture layers based on results in the validation and the testing phases to get the best accuracies. Researched on mounting large-size training data in Google Colab and successfully integrated our project with colab.