Advanced Deep Learning - Final Project

Raz Diamond(ID. 322281528)[1], Ron Elyashar(ID. 209350644)[2],
Liav Ermias(ID. 211684956)[3], and Roeei Itshayek(ID. 212210751)[4]

[1,2,3,4]Colman

Submitted as final project report for Basics Of Deep Learning
course, Colman, 2025

1 Introduction

1.1 Data

In this project, we worked with the `birds-garden-uk` dataset, which contains thousands of bird images.

We divided it into training, validation, and test sets with a corresponding 70% 10% 20% split. We used the pictures with backgrounds and needed to create pairs of grayscale and colored images.

The input images were then resized to 128×128 pixels with a single channel (grayscale), and the output images were 128×128 pixels with three channels (RGB).

Figure 1: Dataset Example

1.1.1 Data Augmentation

For our data augmentation needs we used the RandomZoom and RandomRotation class in Keras which adds support for augmenting images.

The layers are configured with a low augmentation factor, this is done in order to reduce potional noise learning that is possible through the data augmentation.

1.2 Problem

The goal of this project is to develop a deep learning model that can automatically colorize grayscale images of birds. The challenge lies in accurately predicting the color distribution in the image while maintaining the structural integrity of the original grayscale input. We explore two main approaches: a naive method using traditional neural networks and an advanced method based on Conditional Generative Adversarial Networks (cGANs).

2 Solution - Part 1: Naive Method

2.1 General Approach

The naive method involves training a neural network to directly map grayscale images to their corresponding colored versions. We experimented with three different architectures: a Fully Connected Neural Network (FCNN), a Convolutional Neural Network (CNN), and a U-Net.

2.2 Design

All the models in the part were naive meaning that they did everything by their own, recieving a grayscale image and output a colorized one. This means that the input and output sizes were fixed at $128 \times 128 \times 1$ (grayscale), and the output size was $128 \times 128 \times 3$ (RGB) between all the models and the only differentiating part was their inner architecture.

2.3 First Experiment: Fully Connected Neural Network (FCNN)

2.3.1 Model Architecture

In this experiment we worked with a Fully Connected Neural Network.

An FCNN consists of fully connected layers, which are simple and easy to implement. However, they are problematic for image data because they ignore the spatial structure of the image, as such the input grayscale image is flattened into a 1D vector, passed through several dense layers, and then reshaped back into a 2D image.

Our FCNN Architecture:

- Input layer: a flattening layer that receives 128*128*1 shapes and converts them into 16384 neurons
- Hidden Layers: 3 Dense Layers with 512, 1024, 2048 neurons accordingly activated by the relu function Each followed by BatchNormalization and Dropout
- Output Layer: A dense layer with 128*128*3 neurons activated by the Sigmoid function.

2.3.2 Optimization Method

Adam optimizer was used with default parameters.

2.3.3 Loss Function

Mean Squared Error (MSE) was used to measure the pixel-wise difference between the predicted and actual colored images

2.3.4 Hyper Parameters

- Learning Rate: 0.001 – A standard choice for Adam optimizer.
- Batch Size: 32 – A common batch size that balances memory usage and training stability.
- Epochs: 200 – Enough to observe convergence with EarlyStopping based on validation loss to stop Over Fitting when it starts.

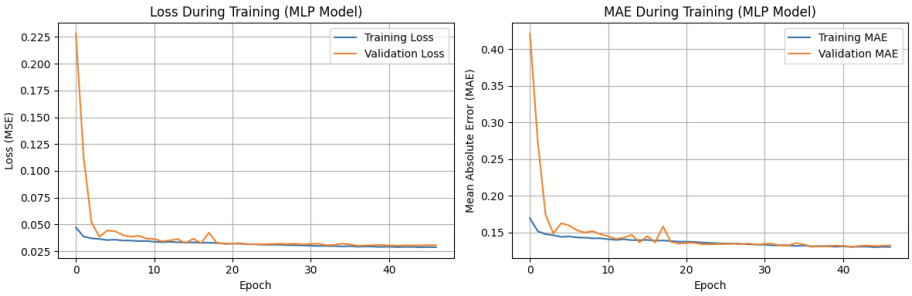

Figure 2: Loss And MAE over epoch

2.3.5 Results

This experiment serves as a baseline to demonstrate the limitations of fully connected networks for image-to-image tasks. FCNNs are computationally expensive for high-resolution images and struggle to capture spatial relationships.

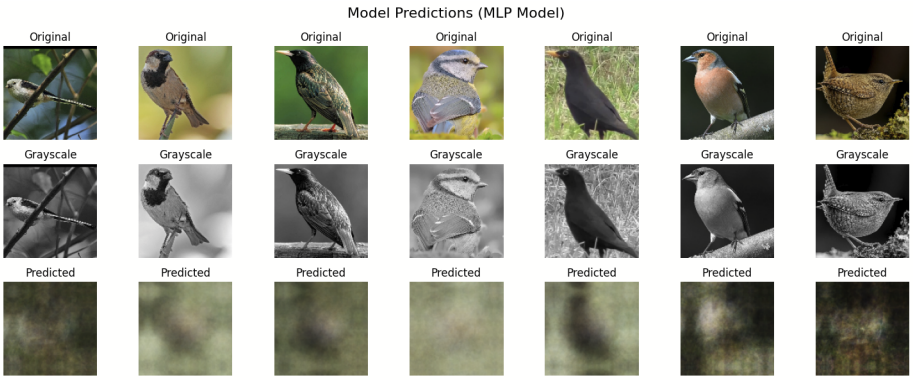

Figure 3: Final Results

2.4 Second Experiment: Convolutional Neural Network (CNN)

2.4.1 Model Architecture

In this experiment we used a naive CNN.

The CNN uses convolutional layers to capture spatial features in the image. Convolutions are more efficient than fully connected layers for image data because they preserve spatial relationships and thus can solve the problem faced in the last model where even though the colors were similar the original shape of the bird was lost completely.

Our CNNs Architecture:

- Input layer: A simple layer containing 128*128*1
- Hidden Layers: 6 Convolution layers each followed with a batch normalization layer, activated by the relu function split into the following sections:
  - Encoding layers: 2 Convolution layers, 64 and 128 accordingly with a 3x3 mask followed by a pooling layers which reduces spatial dimensions
  - Bottleneck layers: a 128 Convolution layer with a 3x3 mask
  - Decoding layers: A Upsampling Convolution layer followed by two Convolution layers all masked with a 3x3 mask.
- Output Layer: A Convolution layer which outputs 128*128*3 neurons activated by the Sigmoid function.

2.4.2 Optimization Method

Adam optimizer was used with default parameters.

2.4.3 Loss Function

Mean Squared Error (MSE) was used.

2.4.4 Hyper Parameters

- Learning Rate: 0.001 – Same as FCNN for consistency.
- Batch Size: 32 – Same as FCNN.
- Epochs: 200 – With early stopping based on validation loss, Same as FCNN.

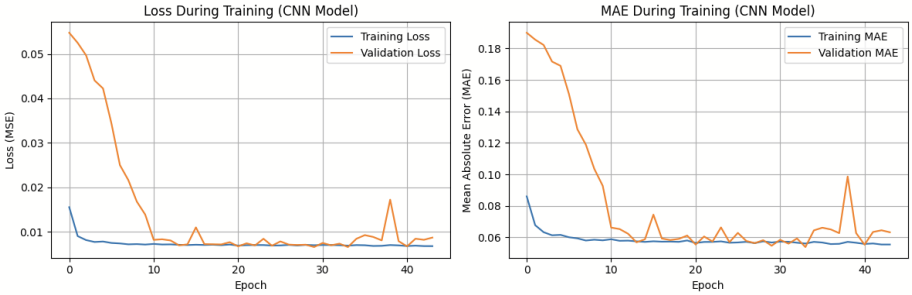

Figure 4: Loss And MAE over epoch

2.4.5 Results

This experiment shows that a simple CNN can outperform the FCNN as CNNs can learn local patterns (e.g., edges, textures) and spatial hierarchies and perserve patterns between pixels.

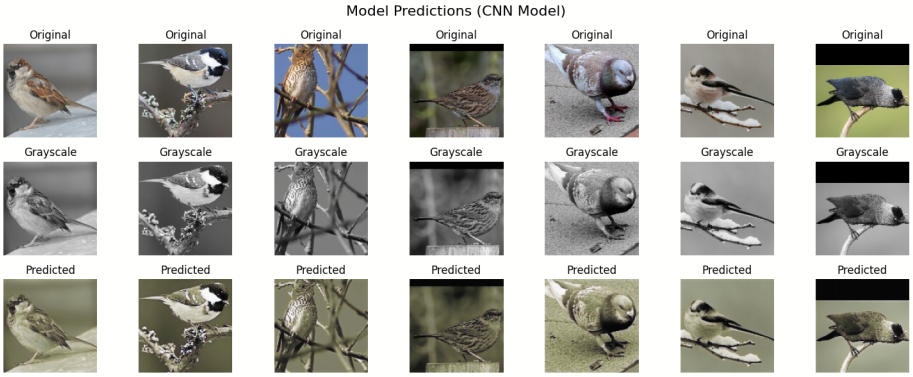

Figure 5: Final Results

2.5 Third Experiment: U-Net

2.5.1 Model Architecture

In this experiment we will use a U-Net model, similar to a CNN the U-Net is Convolution based encoder-decoder network with skip connections.

The encoder downsamples the image to extract high-level features, while the decoder upsamples to reconstruct the image.

Skip connections help preserve fine-grained details by passing information directly from the encoder to the decoder. U-Net is particularly effective for image-to-image tasks because it combines global and local information.

Our U-Net Architecture:

- Input layer: A simple layer containing 128*128*1
- Hidden Layers: 6 Convolution layers activated by the relu function split into the following:
  - Encoder layers: 3 Convolution layers, 64, 128, 256 accordingly with a 3x3 mask followed by a batchNormalization and a pooling convolution layer
  - Decoding layers: 2 Upsampling Convolution layers (128, 64) followed by a skip connection connecting the first and second Encoder layers to the output of the decoder layers, these are followed by a pooling convolution layer
- Output Layer: A Convolution layer which outputs 128*128*3 neurons activated by the Sigmoid function.

2.5.2 Optimization Method

Adam optimizer was used with default parameters.

2.5.3 Loss Function

Mean Squared Error (MSE) was used.

2.5.4 Hyper Parameters

- Learning Rate: 0.001 – Same as FCNN for consistency.
- Batch Size: 32 – Same as FCNN.
- Epochs: 200 – With early stopping based on validation loss, Same as FCNN.

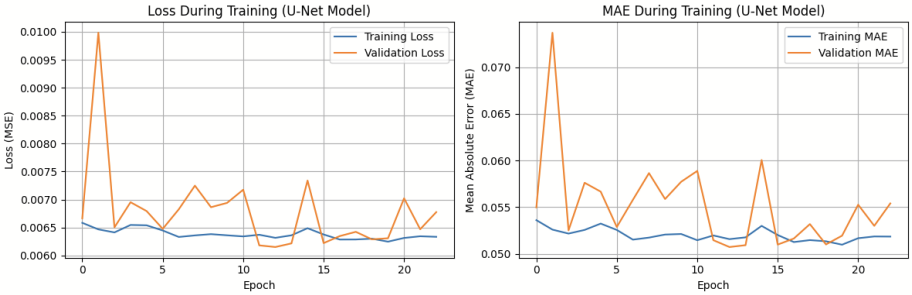

Figure 6: Loss And MAE over epoch

2.5.5 Results

This experiment shows that U-Net is more complex than FCNN and CNN but produces better results due to its ability to capture both high-level and low-level features. Thus the results are shown in the next section.

2.6 Best Model Results and Metrics

The U-Net model performed the best among the three naive methods, achieving the lowest MSE on the validation set. The results were visually superior, with more accurate color predictions and better preservation of image details.

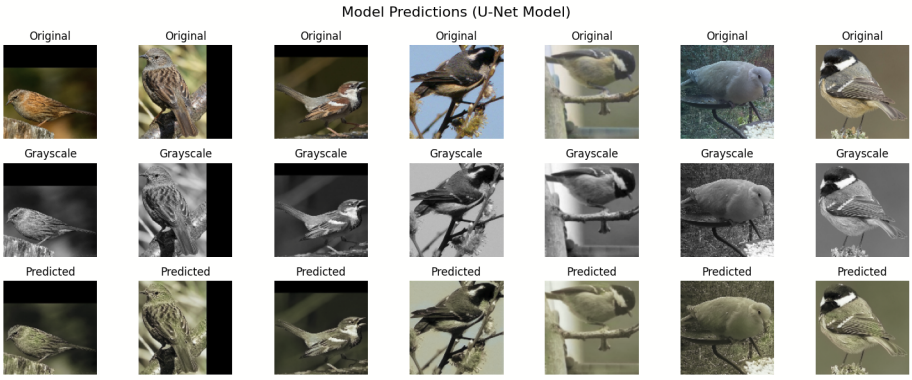

Figure 7: Final Results

3 Solution - Part 2: Advanced Method (cGAN)

3.1 General Approach

The advanced method uses Conditional Generative Adversarial Networks (cGANs), which are designed for image-to-image translation tasks. cGANs consist of two networks: a generator that produces colored images from grayscale inputs, and a discriminator that distinguishes between real and generated images. The adversarial training process encourages the generator to produce more realistic images.

3.2 First Experiment: Simple cGAN

In this experiment we will create a simple cGAN, with a naive CNN based generator and discriminator.

3.2.1 Model Architecture

- Generator: A simple CNN that takes a grayscale image as input and outputs a colored image.
  - Input Layer: A simple 128x128x1 layer
  - Hidden Layers: 5 Convolution Layers Activated by the LeakyReLU Function Each followed by a batch normalization method.
    - Encoding layers: 2 Convolution Layers with a 4x4 mask Activated by the LeakyReLU Function Each followed by a batch normalization method.
    - Bottleneck layers: a 256 Convolution layer with a 4x4 mask
    - Decoding layers: 2 Transpose Convolution Layers with a 4x4 mask Activated by the LeakyReLU Function Each followed by a batch normalization method.
  - Output Layer: A Single Transpose Convolution Layer activated by the tanh activation function outputting a 128x128x3 image
- Discriminator: A standard CNN classifier that checks wherether images as real or fake.
  - Input Layer: A simple 128x128x3 layer
  - Hidden Layers: 2 Convolution Layers Activated by the LeakyReLU Function with a 4x4 mask
  - Output Layer: A single neuron layer activated by the sigmoid function

3.2.2 Hyper Parameters

- Learning Rate: 0.0002 – A common choice for GANs to ensure stable training.
- Batch Size: 32 – Same as naive methods.
- Epochs: 400 – GANs typically require more epochs to converge.

3.2.3 Optimization Method

Adam optimizer was used with $\beta_1 = 0.5, \beta_2 = 0.999$ to stabilize training.

3.2.4 Loss Function

Catigorical Cross Entropy (CCE) was used.

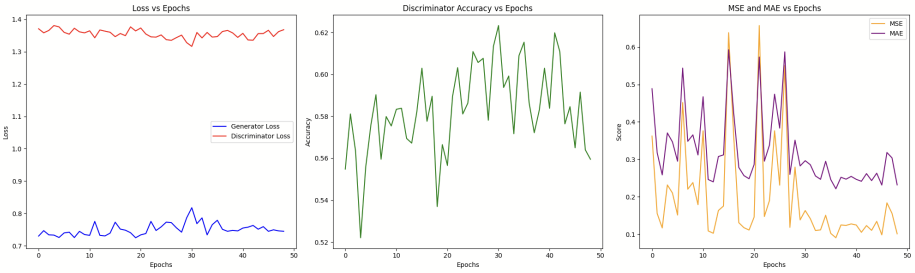

Figure 8: Loss And MAE over epoch

3.2.5 Results

This experiment shows that a basic cGAN can outperform the naive methods.

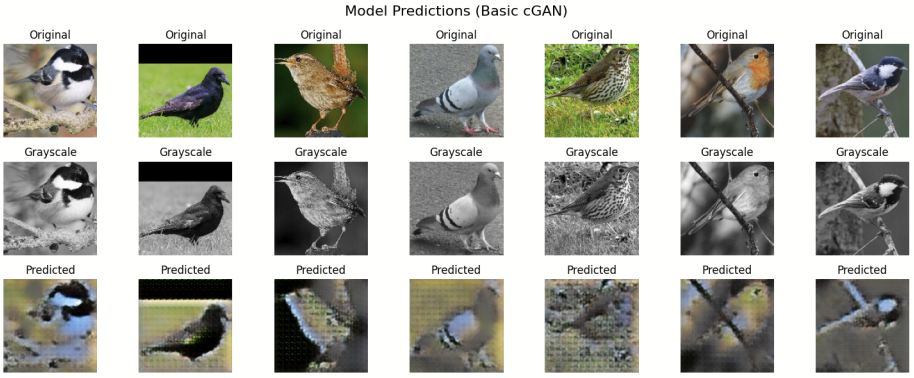

Figure 9: Final Results

3.3 Second Experiment: cGAN with Complex Loss functions

In this experiment we will create a More Complex cGAN, as Catigorical Cross Entropy is a common loss used in GAN models, we have chosen to use a more complex loss solution more fitted to cGAN.

As shown in the article [2] the best performance a cGAN Received is while the generator was using a loss function called 'Mean Absolute Error' in combination with 'Catigorical Cross Entropy', supposedly this causes the generator to improve its performance imancly due to the different aspects of image generation that each loss function addresses.

As BCE is concerned with the classification of pixels, MAE measures the average absolute difference between the generated image and the target image

In essence, the combination of MAE and BCE provides a more comprehensive loss function that guides the cGAN to generate images that are both semantically accurate and visually realistic.

3.3.1 Model Architecture

- Generator: Unchanged
- Discriminator: Unchanged

3.3.2 Hyper Parameters

- Learning Rate: 0.0002 – Unchanged.
- Batch Size: 32 – Unchanged.
- Epochs: 400 – Unchanged.

3.3.3 Optimization Method

Unchanged

3.3.4 Loss Function

The Discrimintator used Categorical Cross Entropy (CCE) and the Generator Used a combination of 'Mean Absolute Error' and 'Catigorical Cross Entropy'

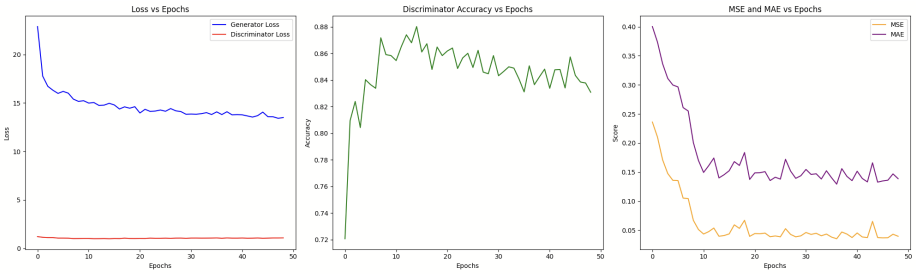

Figure 10: Loss And MAE over epoch

3.3.5 Results

This experiment shows that a cGAN with a more complex loss system can outperform the naive cGAN method.

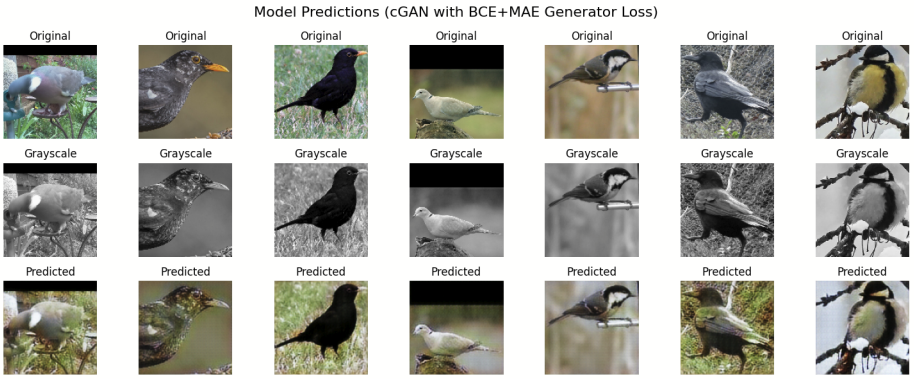

Figure 11: Final Results

3.4 Third Experiment: cGAN with U-Net Generator and PatchGAN Discriminator

In this experiment we will use the State of the art model shown in the following article. [2]

It uses a combination of U-Net based CNN as its generator and a PatchGAN based CNN as its Discriminator.

PatchGAN is supposedly more effective than a standard discriminator because it focuses on local image patches, which helps the generator produce sharper and more detailed images.

U-Net is supposedly a better generator than a standard CNN because it is better at preserving the fine details in the output image, making it a natural choice for the generator in a cGAN.

3.4.1 Model Architecture

- Generator: A U-Net based CNN that takes a grayscale image as input and outputs a colored image.
  - Input Layer: A simple 128x128x1 layer
  - Hidden Layers: 5 Convolution Layers Activated by the LeakyReLU Function Each followed by a batch normalization method In addition to 2 Skip Connections connecting the encoding and decoding layer output.
    - Encoding layers: 2 Convolution Layers with a 4x4 mask Activated by the LeakyReLU Function Each followed by a batch normalization method.
    - Bottleneck layers: a 256 Convolution layer with a 4x4 mask
    - Decoding layers: 2 Transpose Convolution Layers with a 4x4 mask Activated by the LeakyReLU Function Each followed by a batch normalization method.
    - Skip Connections: 2 Skip connections were added after each Decoding layer connecting the encoded features and the decoded attributes in order to try and pass through more data from the original signal.
  - Output Layer: A Single Transpose Convolution Layer activated by the tanh activation function outputting a 128x128x3 image
- Discriminator: A PatchGAN, which classifies overlapping patches of the image instead of the entire image. This encourages the generator to produce locally realistic images.
  - Input Layer: A simple 128x128x3 layer
  - Hidden Layers: 5 Convolution Layers Activated by the LeakyReLU Function with a 4x4 mask
  - Output Layer: A Convolution layer which classifies each mask as fake or real

3.4.2 Hyper Parameters

- Learning Rate: 0.0002 – Unchanged.
- Batch Size: 32 – Unchanged.
- Epochs: 400 – Unchanged.

3.4.3 Optimization Method

Unchanged

3.4.4 Loss Function

Unchanged at CCE and MAE&CCE

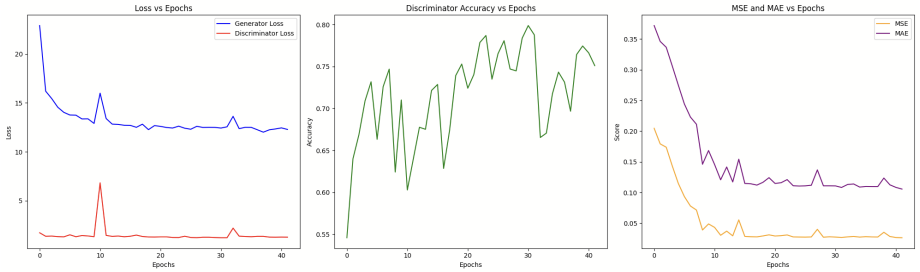

Figure 12: Loss And MAE over epoch

3.4.5 Results

This experiment shows that a cGAN with a more complex loss system can outperform the naive cGAN method.

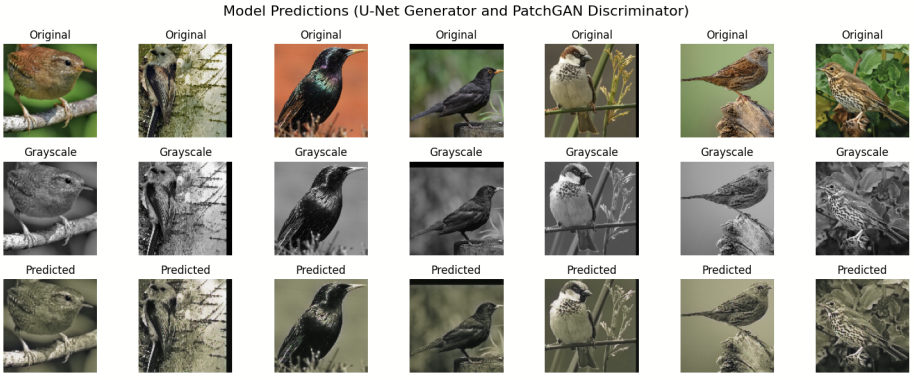

Figure 13: Final Results

4 Conclusion

4.1 Best Model Results and Metrics

Supprisingly the best-performing model overall was the basic cGAN with CCE and MAE&CCE loss. It achieved the lowest loss on the validation set and produced the most visually appealing results.

The U-Net architecture in the naive method also performed well, but the cGAN approach was superior in terms of image quality and color accuracy.

5 Code

5.1 Notebook Part 1

https://colab.research.google.com/drive/1tsFcOgRydmz39jILi01OOOYc0dqK1gfw?usp=sharing

5.2 Notebook Part 1: Train

https://colab.research.google.com/drive/1uo53rHGtg1xbefcv1gLsO-dWiTX90TBj?usp=sharing

5.3 Notebook Part 2

https://colab.research.google.com/drive/1HUJUY9YvVQKETTJ15odrARgyacliYpfl?usp=sharing

5.4 Notebook Part 2: Train

https://colab.research.google.com/drive/1WfIW1RzE31Da0yKr0cYtFnKV_jaWskeV?usp=sharing

References

[1] RandAugment: Practical automated data augmentation with a reduced search space

https://arxiv.org/pdf/1909.13719

[2] Image-to-Image Translation with Conditional Adversarial Networks

https://arxiv.org/pdf/1611.07004