Project Deliverable D1.3

Team: Chulpan Valiullina, Anastasia Pichugina, Saveliy Khlebnov

October 27, 2024

Repository

Link to the repository

Project Topic

Black and White Photo Colorization and Restoration

What has been done so far

- Integrated the model with the website using Streamlit, making necessary changes in multiple files.
- Added GPU support and enabled fallback to CPU for training on T4.
- Refactored data loaders to allow batch loading to avoid memory errors.
- Updated custom dataset to include resizing and modified the `getitem` method for proper batch loading.
- Implemented loss return for training and validation for plotting.
- Created files for `train_dataset` and `val_dataset` to reduce repeated loading times.
- Added some code to build and save plots in a dedicated folder.
- Tried different types of architectures, defined 2 models for further improvement.
- Note: For training, we used small images (256 by 256 pixels) to reduce training time.

Results

A working version is on GitHub. Below are some plots and images:

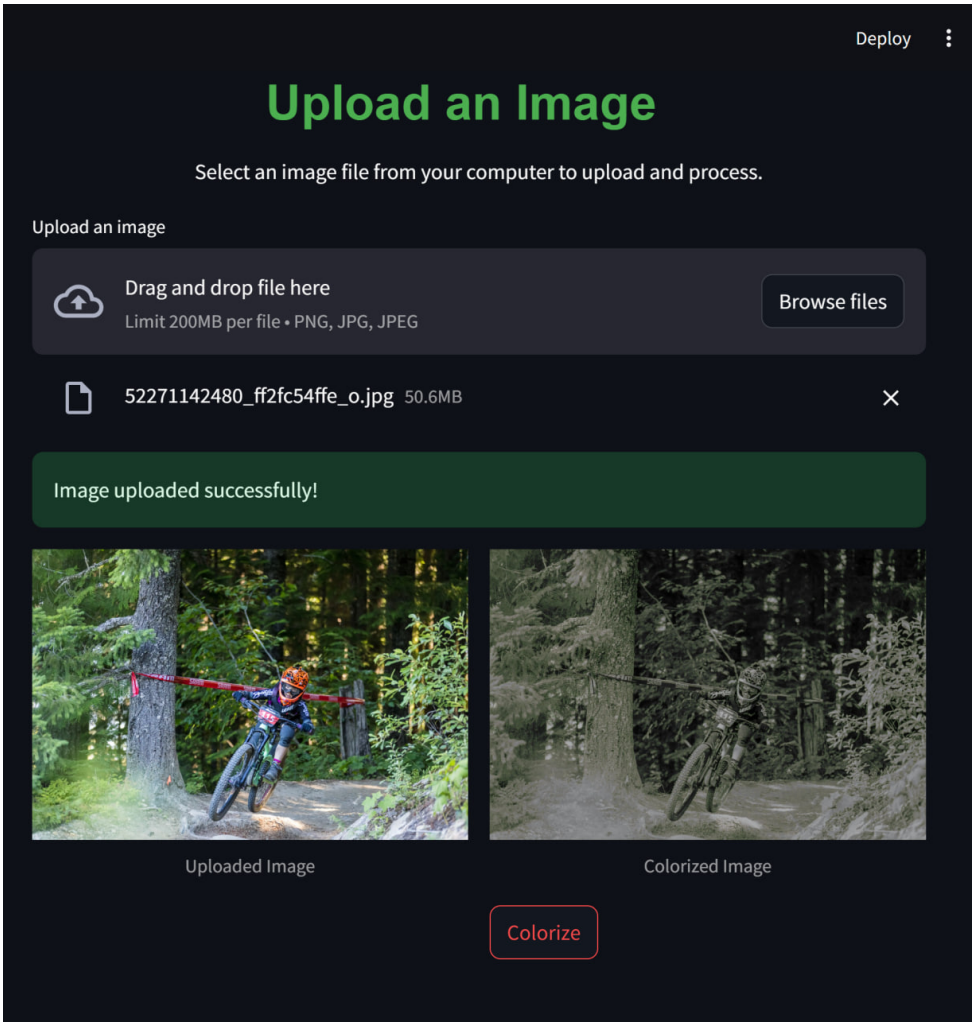

Figure 1: Example of usage

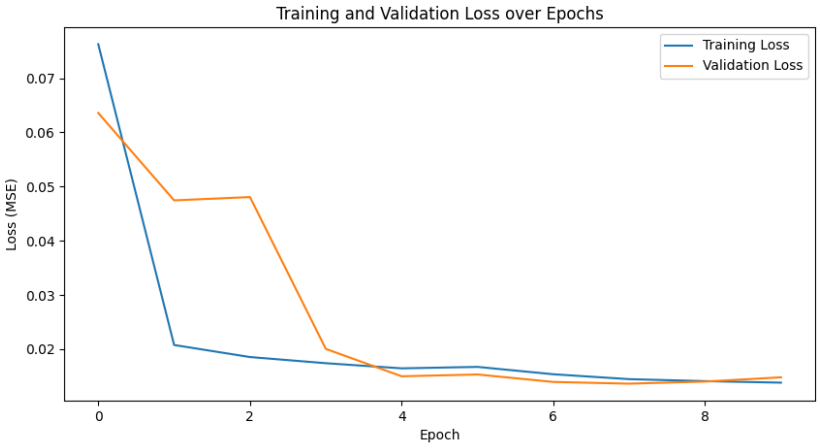

Figure 2: Training and validation loss

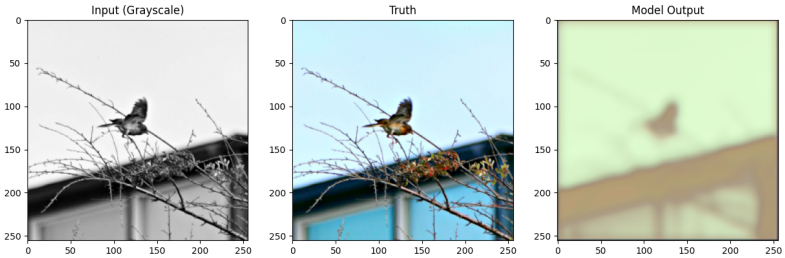

Figure 3: Colorized image after 0 epochs

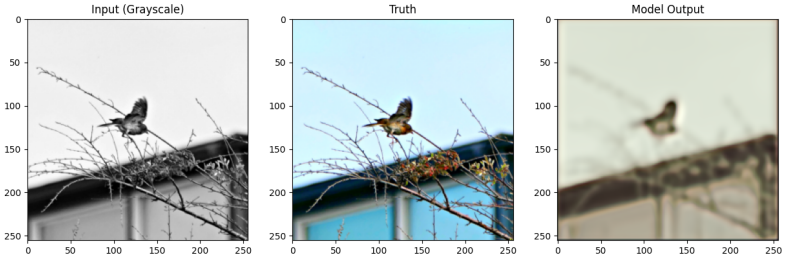

Figure 4: Colorized image after 3 epochs

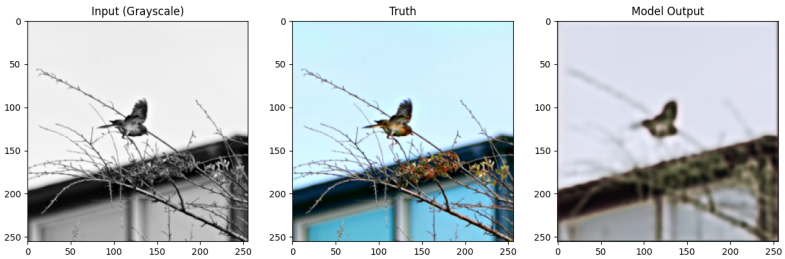

Figure 5: Colorized image after 6 epochs

Figure 6: Colorized image after 9 epochs

10 epochs in total.

Work Distribution

- Chulpan - connected model to the website via Streamlit, wrote report, added example to the README.
- Saveliy - Improved model, loaded data to machine, and optimized memory consumption.
- Anastasia - model improvements and training, added GPU support, refactored data loaders, introduced image resizing, wrote report, and built plots.

Plan for the Next Week

We plan to improve the model by either simplifying the data by lowering image quality or increasing model complexity. Below is the current model architecture:

Python Code Snippet

```
from data.transforms import transform
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchvision.transforms import Normalize

class PictureColorizer(torch.nn.Module):
    def __init__(self):
        super(PictureColorizer, self).__init__()
        self.norm = Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])

        self.conv1 = nn.Conv2d(2, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1, stride
            =2)
        self.batchnorm1 = nn.BatchNorm2d(64)

        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1, stride
            =2)
        self.conv4 = nn.Conv2d(128, 128, kernel_size=3, padding=1)
        self.batchnorm2 = nn.BatchNorm2d(128)

```

```
        self.conv5 = nn.Conv2d(128, 256, kernel_size=3, padding=1)
        self.batchnorm3 = nn.BatchNorm2d(256)

        # Upsampling layers to increase the spatial dimensions
        self.upsample1 = nn.Upsample(scale_factor=2, mode='nearest')
        self.conv6 = nn.Conv2d(256, 128, kernel_size=3, padding=1)
        self.batchnorm4 = nn.BatchNorm2d(128)

        self.upsample2 = nn.Upsample(scale_factor=2, mode='nearest')
        self.conv7 = nn.Conv2d(128, 64, kernel_size=3, padding=1)
        self.batchnorm5 = nn.BatchNorm2d(64)

        self.conv8 = nn.Conv2d(64, 32, kernel_size=3, padding=1)
        self.conv9 = nn.Conv2d(32, 3, kernel_size=3, padding='same')
        self.batchnorm6 = nn.BatchNorm2d(2)


    def forward(self, x):
        # Initial convolution layers with ReLU activations
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.batchnorm1(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.batchnorm2(x)
        x = F.relu(self.conv5(x))
        x = self.batchnorm3(x)
        # Upsampling and convolution layers with ReLU activations
        x = self.upsample1(x)
        x = F.relu(self.conv6(x))
        x = self.batchnorm4(x)
        x = self.upsample2(x)
        x = F.relu(self.conv7(x))
        x = self.batchnorm5(x)
        x = F.relu(self.conv8(x))
        x = F.relu(self.conv9(x))
        x = self.norm(x)
        x = x - torch.min(x)
        x = x/torch.max(x)
        return x

def predict(model, gray):
    prepared = transform(gray)
    img = model(prepared)
    return img
```