Final Technical Report

Team: Chulpan Valiullina, Anastasia Pichugina, Saveliy Khlebnov

November 24, 2024

Repository

Link to the repository

Project Topic

Black and White Photo Colorization and Restoration

Artifacts

Our generated dataset with 2000 pictures in total
Models' weights: CNN and GAN
Scripts (for data loading) and config files (for path extraction and variables setting)
Notebooks with demo: GAN and CNN (available on github too, but "Sorry, this is too big to display.")

What we tried to do during the project

Before the Research Phase we aimed to implement a project that would be helpful and useful. We wanted a robust and simple model that deals with gray or old photos efficiently and fast. We decided not to use Diffusion models since they are too heavy. The simplest way was to download a pretrained model and finetune it. However, we wanted our own models because of 2 reasons:
1) Upgrade of our Deep Learning skills writing almost everything from scratch
2) Challenge yourself to achieve acceptable results via relatively simple architectures.

Workflow

- **Research**. We were looking for possible approaches, available applicable methods, and study papers. During the Research Phase we found out that Colorization isn't a rare practice. There are multiple repositories with implementations. But the 1st part of them used Diffusion models and the 2nd part achieved perfect results with the CIFAR dataset. So, either the underlying implementation was too large, either authors used finetuning, or images resolution was no more than 100 pixels.

- **Definition of goals**. No diffusion and high resolution became the main tasks.

- **Workload distribution**. We divided the project into 2 parts: DL including overall structure, logs processing, and data preparation (Anastasia and Savelyi) and Frontend (Chulpan).

- **Backbone of project**. To initiate a workflow we decided to start with the simplest realization. Our 1st model:

```
class PictureColorizer(torch.nn.Module):
    def init(self):
        super(PictureColorizer, self).init()
        self.conv1 = torch.nn.Conv2d(2, 4, 3, padding=1)
        self.conv6 = torch.nn.Conv2d(4, 3, 3, padding=1)
        self.norm = Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    def forward(self, x):
        x = self.conv1(x)
        x = torch.nn.functional.relu(x)
        x = self.conv6(x)
        x = torch.nn.functional.relu(x)
        x = self.norm(x)
        x = x - torch.min(x)
        x = x/torch.max(x)
        return x
```

Pictures were loading on the fly and weren't being saved anywhere (we used access by url), which took a lot of time. Train and validation functions consisted of iteration over DataLoaders:

```
optimizer.zero_grad()
outputs = model(gray)
loss = loss_fn(outputs, colored)
loss.backward()
optimizer.step()
# and several prints.
```

Our 1st result (original from the left and colorized from the right):

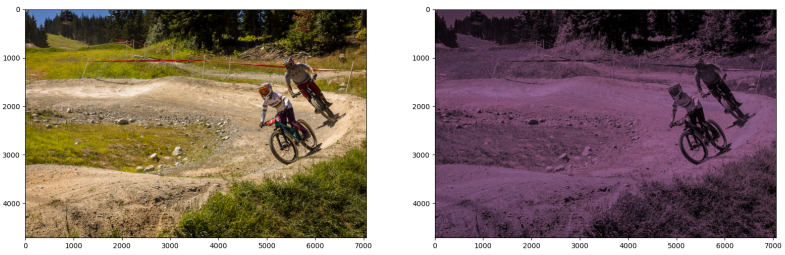

Figure 1: First model's result

- **Process**. In further improvements we were adding new methods, architectures, parameters, and helper functions. We chose to decrease image resolution to 128 by 128 pixels and check if models are able to deal with this level of complexity.

We implemented .pkl files with train and validation datasets to get rid of duty to send requests hundreds of times during picture loading. Then we abandoned this approach because of incompatibilities with Kaggle and generated the dataset yourself using our script. We uploaded it to Kaggle Datasets to access it without loading. We added losses and images saving for plots, and lots of other features.

- **Close to finish**. We chose 2 models: CNN and GAN. CNN was chosen because it's easy to understand: it uses Conv2D blocks, normalization and upsample. It was interesting to bound yourself with convolutions and build something. GAN structure was chosen because it is powerful and simple. One of the main challenges became the training process. Both models took about 10 minutes per epoch or more, and 50 and 25 epochs for CNN and GAN correspondingly. We had to retrain them several times: addition of new plots, methods, increase of resolution, typos, or GPU server disconnection.

- **Finalization**. Right now we finalized our project and several steps are left (regarding the repository structure, README files and small adjustments). Also we would like to improve our CNN model.

Main Results

CNN

Loss after epoch 45 CNN (we forgot to update numbers, it's not 30, it's 45)

Figure 2: CNN train and validation loss

Results with best.pth

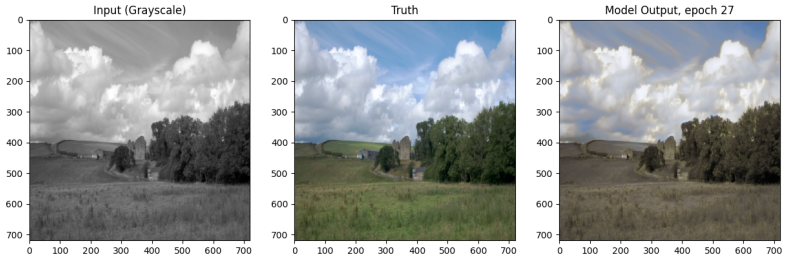

Figure 3: CNN example on one of the best epochs

GAN
Loss during 25 epochs

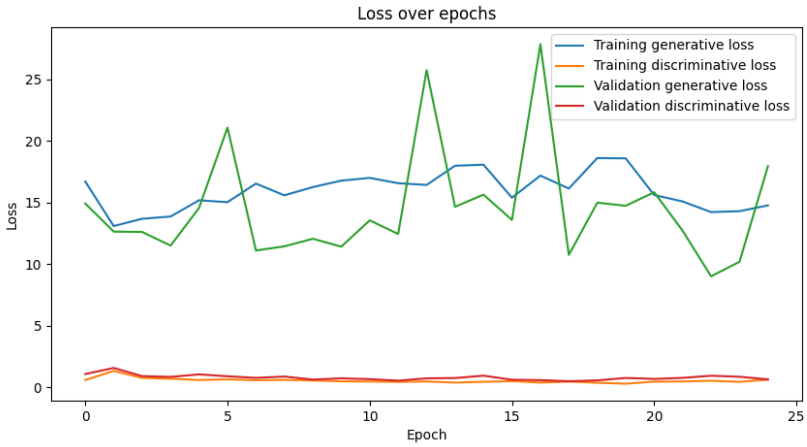

Figure 4: GAN train and validation loss

Results with best.pth

Figure 5: Best GAN model result

Combination of CNN, GAN, and color enhancement (on images 2048 by 2048 pixels)

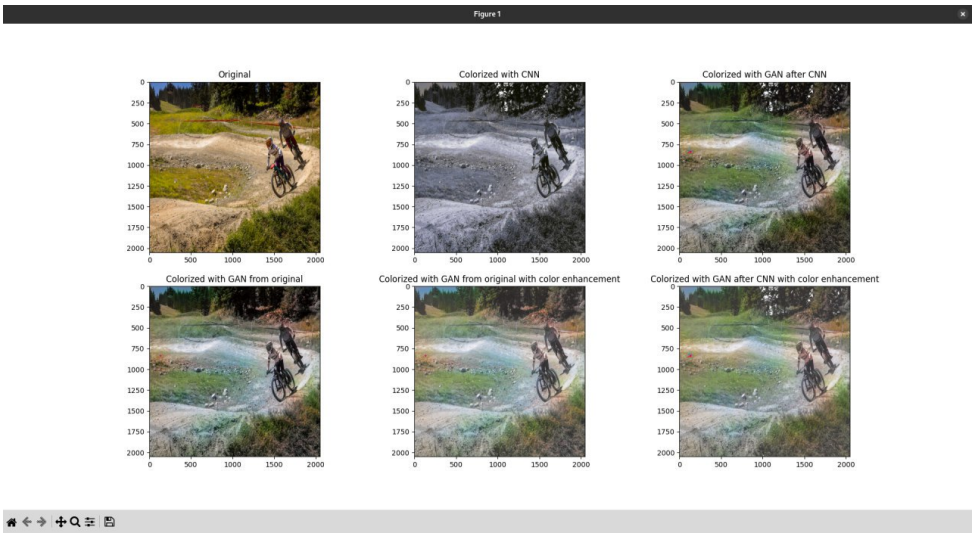

Figure 6: CNN, GAN, and color enhancement results

Timeline of the project

Topic/Task	Name	Date
Choose topic, write first report, create repo	Chulpan, Anastasia, Saveliy	20.09
Images upload to web page	Chulpan	4.10
Structure for ML, base functions	Savelyi	4.10 - 6.10
Train, garbage collector, normalization	Savelyi	28.10 - 25.10
Connected streamlit	Chulpan	26.10
Dataset resize, plot function, models adjusting, train changes, better folders structure, best weights, new model selection	Anastasia	18.10 - 7.11
Long training, weights after 42, 80, 66 epochs on cnn, update readme, brightness function addition	Chulpan, Anastasia	8.11 - 19.11
GAN Demo file (.ipynb) for independent training combining	Anastasia	15.11
Files structure, model, train, data, updates. Mistakes fixing, GAN evaluation, addition of new loss collection. Git-lfs, GAN model, GAN weights, requirements, configuration files, main.py updates	Anastasia	14.11-22.11
CNN Demo file (.ipynb) for independent training combining	Anastasia	19.11 - 21.11
Test.py with GAN, CNN, and increasing saturation	Saveliy	23.11
The CNN model training continues	Anastasia	23.11 - ...
Main.py fixing bags, changed model download (from cloud)	Chulpan	24.11
Report contribution	Anastasia, Chulpan	24.11

Individual contribution

Chulpan

- Readme
- Web representation with streamlit, model connection to frontend, model upload from cloud to web page
- Model train

Savelyi

- Initial repository structure
- Base structure of CNN
- Initial preprocessing for CNN
- First training loop
- Compilation of results of CNN and GAN (test.py)

Anastasia

- Overall final repository structure
- Model choosing (research and implementation), training (the whole training loop structure), and testing
- Data preprocessing: images extracting, composing, and loading for usage
- Util functions (model/data/functions.py, transforms.py, Dataset.py)

References

- Paper with code GAN architecture: Link
- Github with GAN model: Link
- CNN model: Link