COMPUTER VISION PROJECT 3

Image Colourization

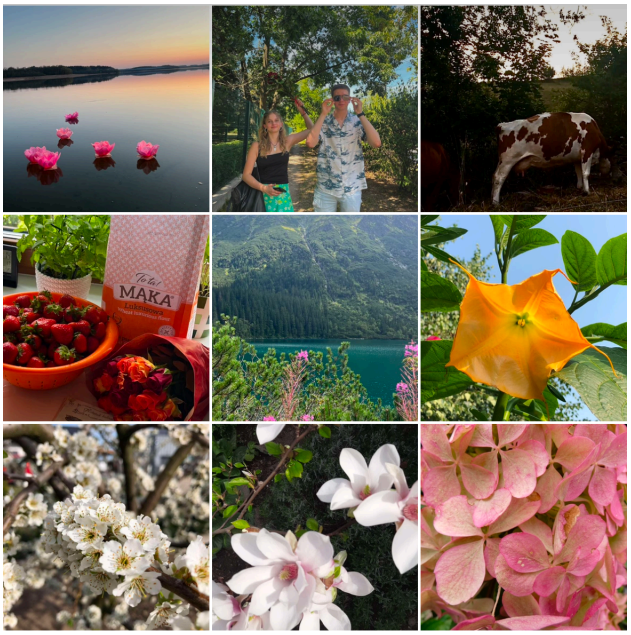
Joanna Szczublińska & Wiktoria Szarzyńska

COLOVERS

DATASET

This dataset, available on Kaggle, contains images of flowers from 5 different categories: **daisy, dandelion, rose, sunflower, and tulip**. The dataset consists of over 4,000 labeled images of these flowers, making it a great resource for training image classification models, especially in the field of plant recognition. The images are high-quality and vary in size and resolution, providing a diverse set of examples to work with.

In addition to this Kaggle dataset, we have supplemented it with our own collection of over **500 personal photos of plants**, further enriching the dataset for a more robust and varied training experience. This custom set adds even more diversity, with different lighting conditions, angles, and environments, enhancing the overall dataset's quality and size for machine learning tasks.



Source: Kaggle (by alxmamaev); format: JPEG, image size: varies (most images around 200x200 pixels), image resolution: varies (ranging from small to large); total number of images: 4 242 images + 530 our own.

PROBLEM

We want to **transform grayscale (black-and-white) images into their corresponding colorized versions** by predicting the missing color information. This process involves generating realistic and contextually appropriate colors for each pixel in the image based on its content. The goal is to create vibrant, visually appealing images that are as close as possible to natural color photographs.

BIBLIOGRAPHY

1. **Dataset**: Flowers Recognition.
2. **TensorFlow**. Intro to auto encoders.
3. Building an Image Colorization Neural Network - Part 4: Implementation. **Medium**
4. Deep Learning Techniques for Image Colorization: A step by step guide. **Medium**.
5. Aksenyuk: aksenyuk/image-inpainting.
6. Youtube and internal resources to it: http://richzhang.github.io/colorization/.
7. Image Colorization basic implementation with CNN.

GITHUB

https://github.com/Wikusia-s/Image-Colourization/settings.

LIBRARIES

1. CV2 library.
2. Numpy library.
3. Scikit-sklearn library.
4. Matplotlib library.
5. Tensorflow library.

MODEL

Firstly, **we chose Autoencoder.** Autoencoder is a type of neural network used for unsupervised learning, typically consisting of an encoder and a decoder. **Encoder** compresses the input image into a lower-dimensional representation (latent space).

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 200, 200, 1)	0
conv2d (Conv2D)	(None, 200, 200, 64)	640
max_pooling2d (MaxPooling2D)	(None, 100, 100, 64)	0
conv2d_1 (Conv2D)	(None, 100, 100, 128)	73,856
max_pooling2d_1 (MaxPooling2D)	(None, 50, 50, 128)	0
conv2d_2 (Conv2D)	(None, 50, 50, 256)	295,168
max_pooling2d_2 (MaxPooling2D)	(None, 25, 25, 256)	0
conv2d_transpose (Conv2DTranspose)	(None, 25, 25, 256)	590,080
up_sampling2d (UpSampling2D)	(None, 50, 50, 256)	0
conv2d_transpose_1 (Conv2DTranspose)	(None, 50, 50, 128)	295,040
up_sampling2d_1 (UpSampling2D)	(None, 100, 100, 128)	0
conv2d_transpose_2 (Conv2DTranspose)	(None, 100, 100, 64)	73,792
up_sampling2d_2 (UpSampling2D)	(None, 200, 200, 64)	0
conv2d_3 (Conv2D)	(None, 200, 200, 2)	1,154

Total params: 1,329,730 (5.07 MB)

Trainable params: 1,329,730 (5.07 MB)

Non-trainable params: 0 (0.00 B)

The encoder's goal is to capture the [1] most important features of the input image and represent them in a compressed form (latent space). This is achieved by progressively reducing the spatial dimensions of the input image while increasing the depth

[1] autoencoder.summary()

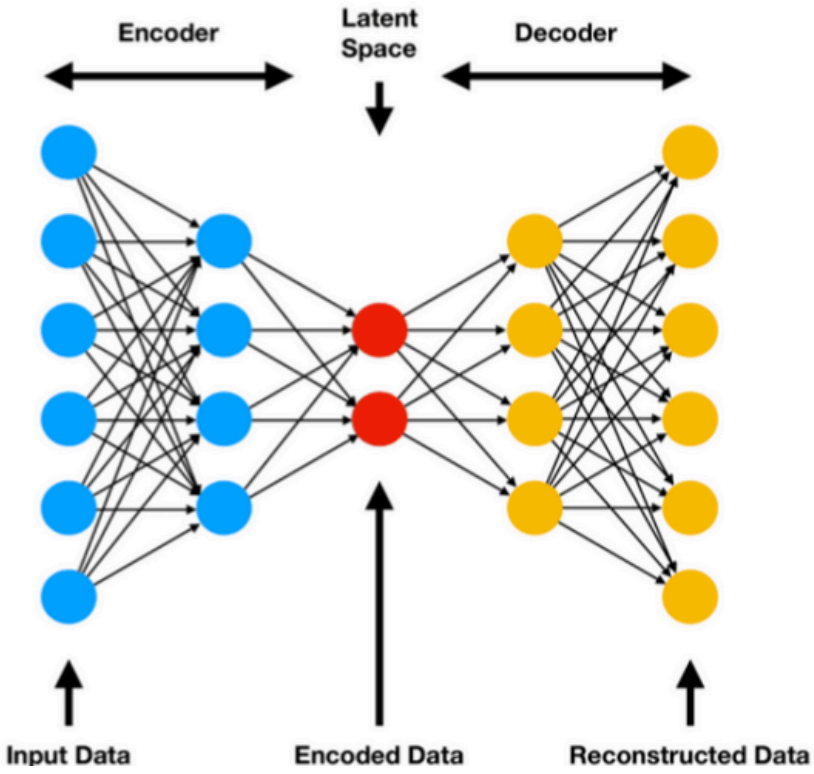
(number of channels).

- **Input Layer:** The input to the encoder is typically the grayscale image (in the case of colorization, this will be the grayscale version of the original image).
- **Convolutional Layer:** These layers apply convolution operations to extract spatial features, such as edges and textures.
- **Latent Space**: The latent representation or bottleneck layer is the final compressed version of the input image. It represents the image in a reduced form, usually as a vector or a very low-dimensional feature map.

Decoder reconstructs the image from the compressed representation, often adding features like colorization.

- **Latent Space Input:** The decoder receives the compressed representation from the encoder as its input.
- **Convolutional Layers:** applies convolution layers to refine the image features. These layers work similarly to the encoder's convolution layers but in reverse.
- **Output Layer.**

[2]



[2] GeeksForGeeks image

INITIAL TRAINING

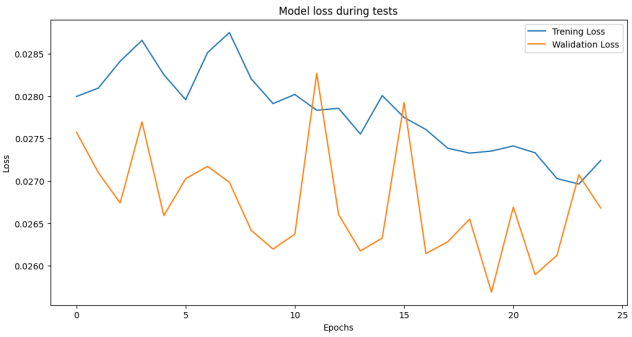
ro run the training please install all the requirements and run the jupyter file; in the file there is initial training (before any parameters checked) and final training, which after all is saved

Metrics:

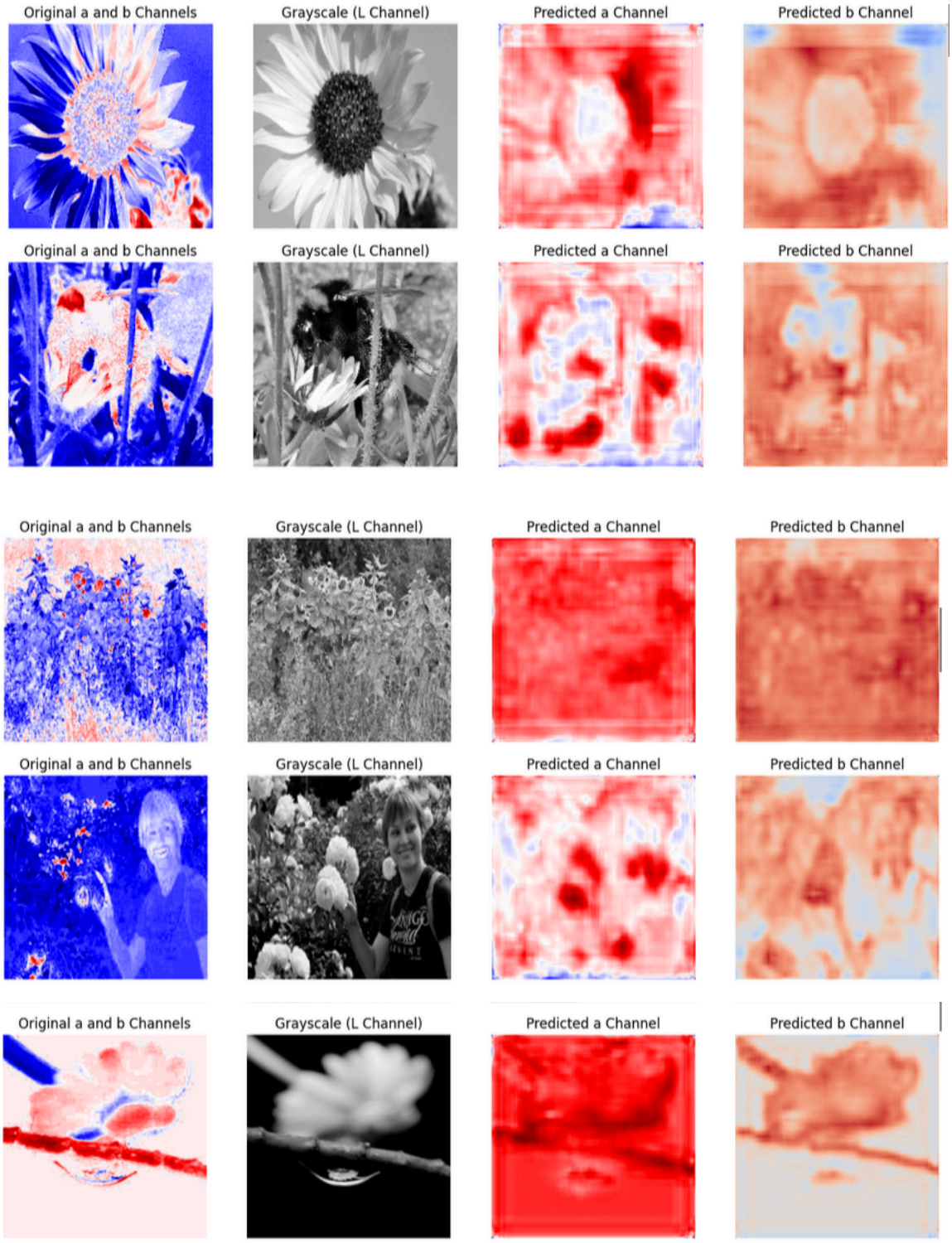
- ➔ **PSNR (Peak Signal-to-Noise ratio)** measures how "similar" the predicted image is to the original image, in decibel (dB) units. A higher PSNR value indicates better quality, as it means the predicted image is closer to the original.
  - ◆ **PSNR > 30 dB**: High quality (the predicted image is very similar to the original).
  - ◆ **PSNR 20-30 dB**: Decent quality (the predicted image has noticeable differences but is still acceptable).
  - ◆ **PSNR < 20 dB**: Low quality (the model struggles with predicting accurate colors or details).

- ➔ **SSIM (Structural Similarity Index)** measures the structural similarity between the predicted image and the original image. It ranges from 0 (completely different images) to 1 (identical images). It focuses on the luminance, texture, and spatial arrangement of pixel values.
  - ◆ **SSIM > 0.8**: High structural similarity (good result).
  - ◆ **SSIM 0.5-0.8**: Decent quality (the model preserves the overall structure, but fine details might be **blurred** or not perfectly recreated).
  - ◆ **SSIM < 0.5**: Low structural similarity (the model has difficulty recreating the spatial layout of the image).

At the beginning, we used only two (mentioned above) metrics. Number of epochs equals 50, batch size initially is 32. Before any experiments with parameters we achieved test loss equals 0.0264, PSNR equals 16.91, SSIM equal 0.5692.



Results Autoencoder



IMPROVEMENTS

Optimizers:

These metrics are really important, because they can indicate which optimizer is the best one to use in this training procedure. We want to compare three optimizers:

- Adam optimizer,
- RMSprop optimizer,
- SGD optimizer.

Comparison of Optimizers:

Optimizer	Validation Loss	PSNR	SSIM
Adam	0.0266655	17.025	0.580011
RMSprop	0.0446986	14.6512	0.569433
SGD	0.0279864	16.7696	0.5706

Adam achieved the best overall performance with the lowest validation loss, the highest PSNR, and a high SSIM, indicating effective colorization quality and image similarity.

RMSprop performed the weakest among the three, with the highest validation loss, the lowest PSNR , and a relatively low SSIM), suggesting less effective training convergence.

SGD offered a balanced performance, with a slightly higher validation loss than Adam but better than RMSprop. Its PSNR and SSIM values were competitive, indicating reasonable colorization performance.

In conclusion, **Adam** stands out as the optimal choice for this task, balancing effective convergence and high-quality predictions.

Loss functions:

- **Mean Squared Error (MSE):** one of the most commonly used loss functions in regression tasks. It calculates the average of the squared differences between the predicted and true pixel values.
- **Mean Absolute Error (MAE):** computes the average of the absolute differences between predicted and true pixel values. Unlike MSE, MAE treats all errors linearly, regardless of their size, making it less sensitive to outliers.
- **Structural Similarity Index (SSIM) Loss:** SSIM is a perceptual loss function designed to measure the similarity between two images. It focuses on

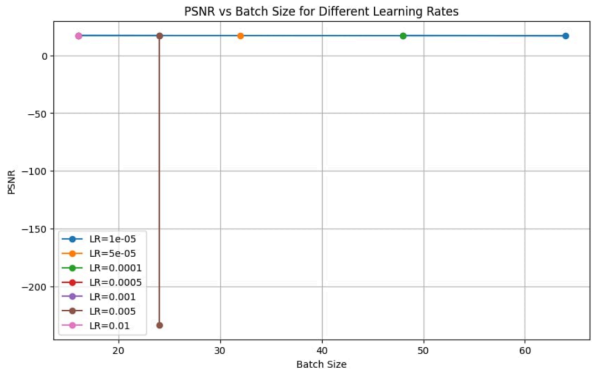
preserving structural and perceptual details, as it considers luminance, contrast, and structure. The SSIM loss is calculated as 1-SSIM1 - SSIM-SSIM, where higher SSIM values indicate better perceptual similarity. This loss is particularly useful in image reconstruction tasks, as it aligns better with human perception compared to pixel-wise metrics like MSE or MAE.

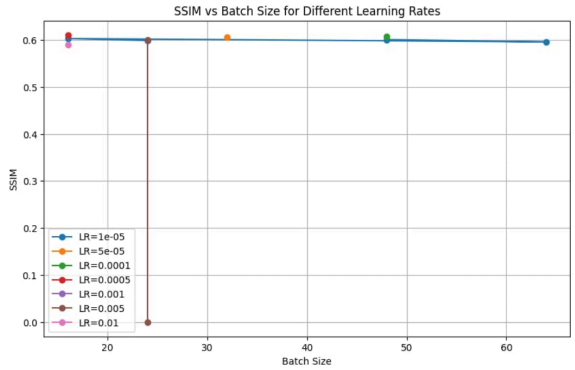
Comparison of Optimizers:

Optimizer	Validation Loss	PSNR	SSIM
Mean Squared Error (MSE)	0.0269622	17.0818	0.578695
Mean Absolute Error (MAE)	0.112162	17.2573	0.581707
Structural Similarity Index (SSIM) Loss	0.401162	17.0711	0.598838

The SSIM Loss function produces the highest perceptual quality in terms of both PSNR and SSIM, making it the most effective for tasks where maintaining the structural integrity and visual similarity of the image is crucial. However, MSE and MAE also offer reasonable performance, with MSE being better at minimizing pixel-wise errors but slightly worse in terms of perceptual quality.

Hyperparameters





We had to check different learning rate (lr) values, because the learning rate controls how quickly the model adjusts the weights during training. A value that's too high can lead to overshooting the optimal solution, while a value that's too low can slow down convergence.

Batch size determines the number of training examples used in one forward and backward pass of the model. A smaller batch size generally leads to noisier updates, which might help the model escape local minima but could also make the training unstable. A larger batch size provides more stable gradients and can speed up training, but requires more memory and might lead to poor generalization.

Epochs represent the number of times the entire dataset is passed through the model during training. The ideal number of epochs depends on the complexity of the model and the dataset. Too few epochs can result in underfitting, while too many can cause overfitting.

Cross validation

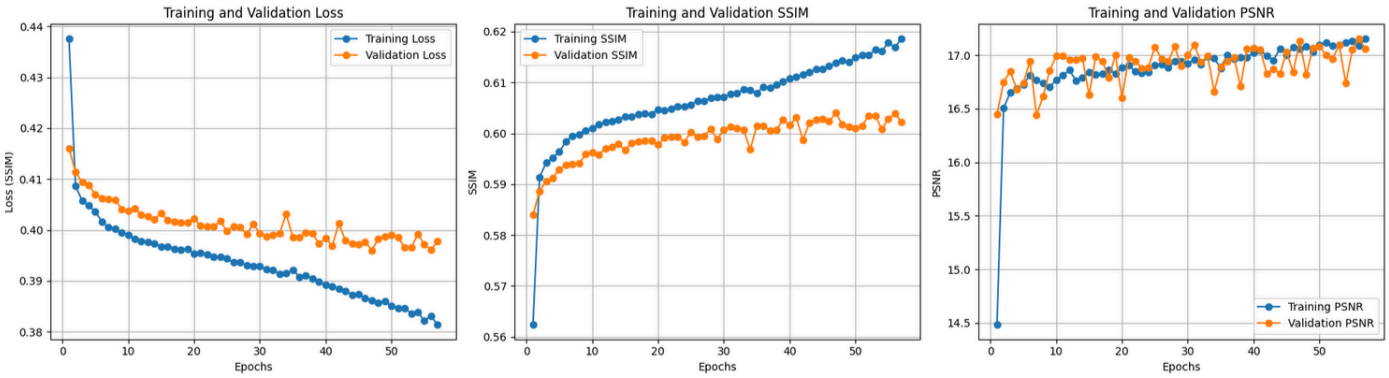
With cross validation:average validation loss was 0.0195, average PSNR was 17.7466, average SSIM was 0.5350. The overall performance is not better than previously.

Summary

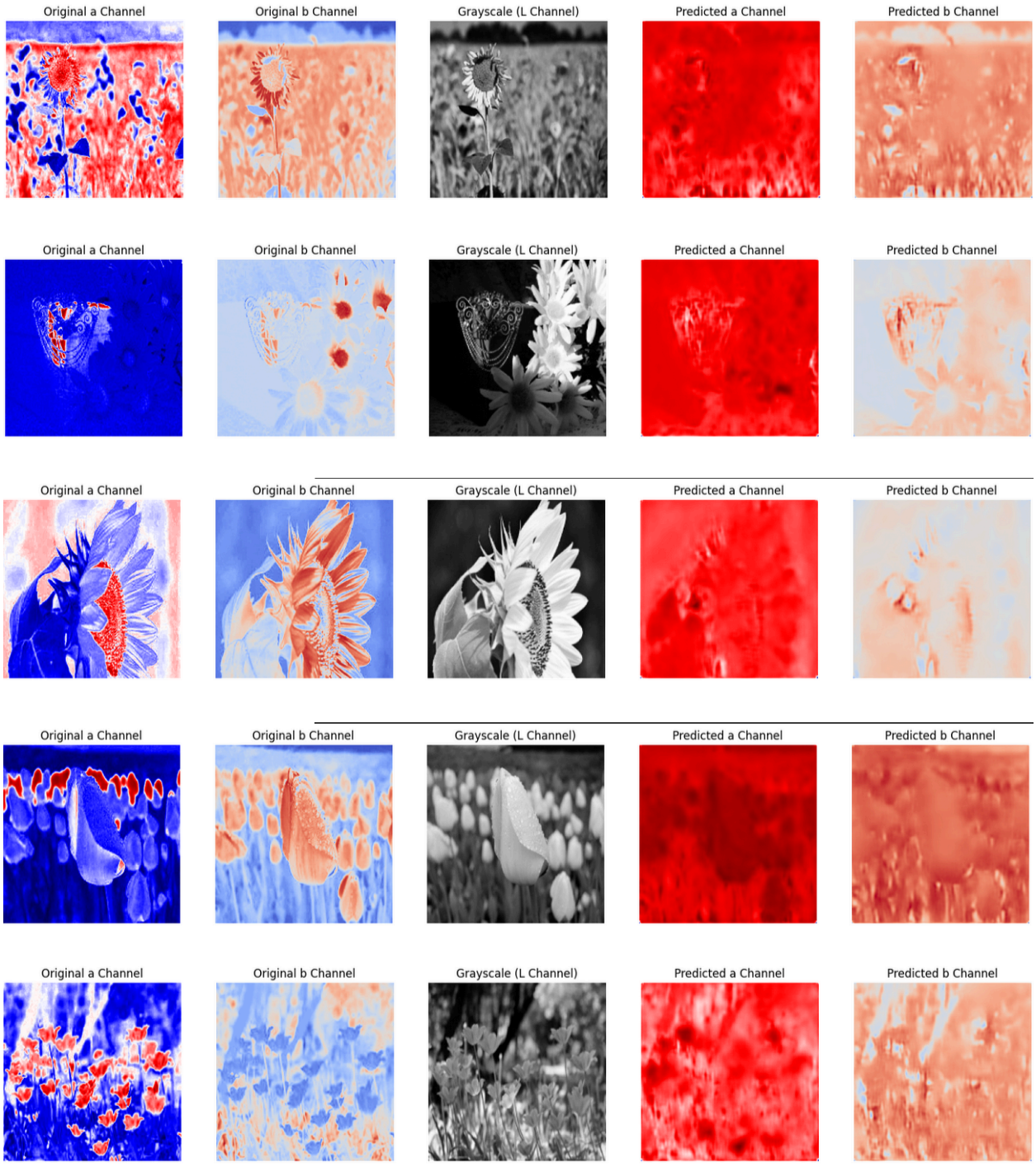
We should use Adam as optimizer, SSIM as loss function and we can easily see that cross validation is not needed. We also should increase the number of epochs.

FINAL TRAINING

Results Loss Function



Results Predictions



RUNTIME ENVIRONMENT

Requirements.txt file in **Github**.

TRAINING AND INFERENCE TIME

About **40** minutes of final training using colab.

COMPLETED ITEMS

ITEMS	POINTS
Choosing a problem: image colourization.	**1** point
Choosing a model from scratch: auto encoder.	**1** point
Add our own part of the dataset (> 500 images).	**1** point
Hyperparameter tuning or estimation.	**1** point
Testing a few (3) optimizers.	**1** point
Testing various (3) loss functions.	**1** point
Cross - validation.	**1** point