Transforming Grayscale images to RGB through a convolutional autoencoder

Carlos Iván Fonseca Mondragón | A01771689

ABSTRACT This report presents the development and implementation of a convolutional autoencoder designed for image colorization using grayscale and color landscape images. The methodology includes image preprocessing, dataset standardization, and an unsupervised reconstruction-based learning approach. The model is trained using pixel-level Mean Absolute Error (MAE), with performance monitored across training, validation, and test subsets. Results show that the autoencoder learns low-level spatial structure and produces harsh color approximations, but struggles to infer accurate colors due to the limitations of reconstruction loss, latent space compression, and the absence of perceptual objectives. Despite these constraints, the model successfully demonstrates the basic mechanics of convolutional autoencoders, highlighting their capabilities and the architectural improvements needed for realistic colorization.

1. INTRODUCTION

The field of computer vision has increasingly focused on techniques that can restore or enhance visual information. Among these techniques, **autoencoders** have proven to be some of the most powerful tools for reconstructing complex image patterns. In this context, colorizing grayscale images represents a challenging task, as it requires the model to deduct color distributions from limited information.

This paper presents a first implementation and evaluation of a convolutional autoencoder designed to transform grayscale images into color images, trained using TensorFlow and Keras, using a publicly available on *Kaggle*. The model was developed from scratch, following a convolutional architecture that encodes grayscale input features into a latent space and reconstructs them in RGB format. The final model achieved a validation loss of 0.1367, although this metric alone does not necessarily reflect the perceptual quality of the generated images, which depends on both structural accuracy and visual realism.

Prior to training, the dataset goes through a size transformation in order to speed up processing. Through experimentation and parameter fine-tuning, the model demonstrates the ability to generalize colorization patterns of bright environments, with potential to further improve with more training time and architectural changes.

2. DESCRIBING THE DATASET

The "*Landscape Color and Grayscale Images*" dataset, uploaded by user Bhabuk Ghimire on Kaggle.com, contains a total of 14,258 images in .jpeg format, each with a resolution of 150×150 pixels, with a combined size of 203.47 MB.

The dataset is evenly divided into two folders: "color" and "gray", each containing 7,129 images. Every image pair shares the same numeric label (from 0 to 7,128), which uniquely identifies corresponding color and grayscale versions of the same landscape.

None of the elements provided in the dataset are corrupted or are of a different resolution than the specified above.

3. DATA EXTRACTION

3.1 Preparing the dataset

A TensorFlow-based routine was developed to load and preprocess all images from the dataset. The two main folders from the original dataset were placed in a shared folder, resulting in two directories: (gray/) and (color/). Each image was read using the tf.io.read_file() method and decoded via tf.image.decode_jpeg(), automatically assigning one channel for grayscale inputs and three channels for color inputs.

4. DATA TRANSFORMATIONS

4.1 Image rescaling

After loading, all images were rescaled to a uniform resolution of 64×64 pixels using tf.image.resize_with_pad(), ensuring dimension consistency across the dataset. The chosen resize method was nearest neighbor, which preserves local contrast and edge definition, rather than performing interpolation-based smoothing.

Both integer (uint8) and float32 tensor versions were generated for each image:
The integer version represents the raw pixel data (0–255).
The float version corresponds to the normalized representation (0-1), used for model training.

Each processed image was stored in new folders (gray_processed/ and color_processed/) to maintain a clean division between raw and transformed data.

This resulted in a total size reduction average of approximately 80% for both gray and color images, going from the initial 203.47 MB, down to 41.3 MB, which dramatically increased the model's training speed.

Finally, a subset of 28 image pairs was manually moved from the procs into new directories (gray_test/ and color_test/) to serve as unseen samples for manual evaluation of the model's performance.

4.2 Train, test, and validation splits

After rescaling, the dataset was divided into training, validation, and test subsets using the train_test_split() function from scikit-learn. Although implemented in a separate script, this operation is conceptually part of the Load phase of the ETL process.

A fixed randomization seed of 42 was used to ensure reproducibility. First, the dataset

was split into a temporary subset (80%) and a test subset (20%). Then, the temporary subset was further divided into training (64%) and validation (16%) sets, resulting in a final distribution of approximately **64%** for training, **16%** for validation, and **20%** for testing.

This division guarantees that the model is evaluated on unseen data, ensuring a fair and reliable analysis of its generalization performance.

5. MODEL IMPLEMENTATION

5.1 Input Data

After preprocessing and splitting the data into training, validation, and testing, the model uses grayscale landscape images as inputs and their corresponding color versions as target outputs. Each image has a fixed resolution of 64×64 pixels. Unlike a regression or classification model, this implementation learns spatial and chromatic relationships from pixel intensities. The input tensors have a shape of (64, 64, 1), while output tensors have a shape of (64, 64, 3). Values of each pixel were normalized to facilitate training stability.

5.2 Model Architecture and Assumptions

This autoencoder was implemented through the use of TensorFlow and Keras. As it stands, this architecture is capable of learning latent space representations from unlabeled image data. It consists of two main components:

- **Encoder**: The encoder is in charge of compressing the initial grayscale image into a low-dimensional representation.
    - The encoder uses Conv2D layers with 32, 64, 128 and 256 filters, each followed by a use of BatchNormalization() and a ReLU activation.
    - Downsampling is applied through convolutional strides of 2.
    - Obtained feature maps are flattened and passed through a dense layer of 128 neurons, defining the **latent dimension (LATENT_DIM = 128)**.
    - Finally, a Dropout layer with a 0.5 rate is applied to avoid overfitting.

- **Decoder**: The decoder reconstructs the color version of the image through the latent representation.
    - It starts with a dense layer, reshaped into (8, 8, 256), followed by multiple Conv2DTranspose layers, that gradually upsample the image back to the original size (64×64), while having 3 layers corresponding to the RGB spectrum, the result is a (64, 64, 3) image.
    - Each layer mirrors the encoded structure, using ReLU activations and BatchNormalization() for stability.
    - The final output employs a sigmoid activation to normalize color values.

The encoder and decoder were combined into a sequential model and compiled utilizing the Adam optimizer and Mean

Absolute Error (MAE) as the loss function, given its utility for image processing tasks.

Training was set to 135 epochs, with a batch size of 128, using the following callbacks:

- ModelCheckpoint (saving the best model based on validation loss)
- ReduceLROnPlateau (reducing learning rate when progress stalls)
- EarlyStopping (stopping training after 10 patience epochs with no improvement).

5.3 Model Objective and the MAE formula

The model aims to learn the mapping between grayscale and color representations by reducing the pixel-wise reconstruction error. This can be represented through the Mean Absolute Error formula:

$$L = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$$

Figure 1: Mean Absolute Error formula

Where $\hat{y}_i$ represents the reconstructed color image output delivered by the decoder. Reducing this loss value makes the model reproduce chromatic tones and gradients, learning colorization patterns while being unsupervised.

6. MODEL EVALUATION

The performance of the autoencoder is evaluated primarily through the training and validation loss, both measured using Mean Absolute Error (MAE). MAE quantifies the average absolute difference between the predicted color pixel values and their ground-truth counterparts. Lower MAE values indicate more accurate color reconstruction, as the model produces color outputs closer to the original images.

In addition to the observed losses during training, this implementation also reports an MAE value after running a query.

7. MODEL RESULTS

Following model training and evaluation, the autoencoder demonstrated stable reconstruction performance, measured through Mean Absolute Error (MAE). Although training was set to 135 epochs, the EarlyStopping mechanism halted the process after epoch 37 after the validation loss stopped improving, ensuring that the model maintained generalization and did not drift into overfitting.

At the final epoch (Epoch 37), the results were:

- Training loss (MAE): 0.1291
- Validation loss (MAE): 0.1367
- Learning rate: 5×10^{-4}

For comparison, during the first epoch, the model began with:

- Training loss (MAE): 0.2010
- Validation loss (MAE): 0.2171
- Learning rate: 1×10^{-3}

This reduction in error across the first 37 epochs indicates that the autoencoder successfully optimized its reconstruction capabilities, learning a compressed latent representation capable of partially restoring color information.

The decrease from an initial MAE of around 0.20 to around 0.13 shows that the model improves upon its ability to approximate the ground truth image. The gap between the training (0.1291) and validation (0.1367) losses remains small, implying that the model is not suffering from overfitting. However, its capacity is limited, as the training error stops improving early and settles quickly.

Qualitative evaluation provides the best picture of performance in colorization models. The following figure compares:

- Input grayscale image
- Ground truth image
- Model output with its corresponding MAE.

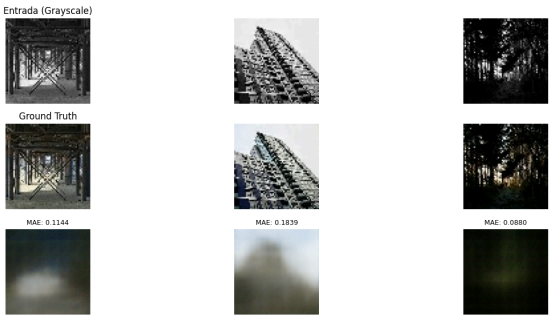

Figure 2: Evaluation of three unseen images in various lighting conditions.

The current model successfully reconstructs basic shapes and scene layout, captures tonal regions (such as a "dark forest") and produces color intensities slightly consistent with the illumination.

The current issues with this first model are:

- Fine textures and high-frequency detail are lost
- Colors often appear washed out, monochromatic, or incorrect.
- Buildings, trees, and structural details lack clear definition.
- Complex scenes yield higher MAE values.

In the examples shown, MAE values ranged from 0.08 to 0.18, but even low-MAE results lack color diversity. This, however, is expected because MAE measures pixel-level deviation, not actual realism.

Overall, the model produces silhouette-level approximations rather than actual color reconstructions. It mainly captures illumination (dark vs. light regions), not the actual color tones of the original scene.

A line plot was generated to visualize MAE across train and validation sets. Where validation values appear to show no progress past around epoch 20, training MAE keeps decreasing, albeit at a very slow pace.

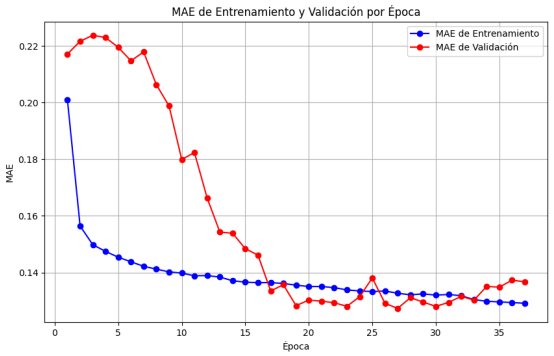

Figure 3: Train vs. Validation MAE line plot.

Plotting the loss curves confirms that early stopping was appropriate, as continued training would not reduce error further. This suggests that the model rapidly

reaches the limits of what it can learn under its architectural constraints.

8. CONCLUSION

This paper explored a convolutional autoencoder approach, designed to perform grayscale to color image reconstruction. The model successfully learned basic structural and illumination patterns, and training converged early at epoch 37 due to the stabilization of validation loss. MAE values improved from 0.2010 to 0.1291 on the training set and from 0.2171 to 0.1367 on the validation set, indicating that the network captured low-level spatial regularities and general brightness distributions across scenes.

However, results show that the model is unable to recover meaningful of semantically accurate color information. The reconstructions primarily depict blurred silhouettes, demonstrating that the autoencoder is limited by its unsupervised reconstruction objective, pixel-level MAE loss, and relatively small latent space. This means that the model can infer where light and shading occurs, but not what colors or shapes to exhibit.

These limitations point towards the possibility of future improvements through perceptual feature losses (e.g., VGG-based), a larger latent dimension, skip connections (as in the case of U-Net) or a variational autoencoder approach. Despite its restricted performance, this implementation provides a functional baseline and highlights the challenges of colorizing natural images without supervised learning.

9. REFERENCES

a. Johnson, J., Alahi, A., & Fei-Fei, L. (2016). *Perceptual losses for real-time style transfer and super-resolution*. In European Conference on Computer Vision (ECCV).
b. https://arxiv.org/abs/1603.08155
c. Lyashenko, V., & Jha, A. (2025, April 25). *Cross-validation in machine learning: How to do it right*. Neptune AI. https://neptune.ai/blog/cross-validation-in-machine-learning-how-to-do-it-right
d. Zhang, R., Isola, P., & Efros, A. A. (2016). *Colorful Image Colorization*. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR). https://richzhang.github.io/colorization/