Image Colorization for Monochrome CCTV Footage Using Deep Neural Networks

Student: Mehher Ghevandiani

BS in Data Science

Supervisor: Anna Tshngryan

Machine Learning Specialist

Yerevan 2025

Abstract

Surveillance systems often rely on monochrome footage, limiting the extraction of crucial visual details. This paper explores the application of deep learning techniques to colorize grayscale CCTV footage, enhancing its interpretability. Different models were trained, with the final U-net architecture showing promising results. Key challenges included dataset limitations and computational constraints. The results demonstrate the potential of deep learning to improve the utility of CCTV footage in real-world applications.

The Problem

Surveillance systems are widely used for security, traffic monitoring, and public safety, often operating around the clock. However, a significant limitation of many CCTV systems, particularly those operating in low-light or nighttime conditions, is their reliance on infrared (IR) imaging or monochrome sensors. These cameras produce grayscale footage that lacks color information, making it difficult to identify critical visual details such as clothing colors, vehicle paint, or distinguishing environmental features. This limitation can hinder law enforcement, reduce situational awareness, and impair decision-making in forensic investigations.

Colorized images can provide richer visual context, enhance human interpretation, and support automated systems such as object tracking or facial recognition. While manual colorization is time-consuming and subjective, ML deep learning models are capable of solving this problem. The aim of this project is to train an ML model that will be able to colorize images captured by a CCTV camera in low light environments.

Related Work

In the field of image colorization, several open-source projects have significantly contributed to advancements in deep learning-based methods. Zhang et al. developed a pioneering approach using a convolutional neural network (CNN) model for automatic image colorization, available through their GitHub repository. Their method leverages classification-based loss functions and incorporates a user interaction mode to refine results, setting a benchmark for research in automatic colorization. This model is known for balancing realism and user control, and has been widely cited and implemented in both academic and practical applications.

Similarly, Antic's *DeOldify* presents a high-quality, GAN-based solution for colorizing and restoring old images and videos. Built on top of fast.ai and PyTorch, DeOldify offers superior output through the use of advanced models like NoGAN training and perceptual loss. Its ease of use and visually compelling results have made it popular among hobbyists and professionals alike. In contrast, Berkay's implementation serves as a more lightweight and educational version, showcasing a U-Net-based architecture for colorization. While not as robust as

DeOldify, it provides a clear and approachable framework for understanding the mechanics of colorization networks. These works collectively illustrate a spectrum of methodologies, from research-grade models to accessible tools, informing and inspiring the direction of my own project.

The Solution

Over the course of this project, three different models were trained (CNN, U-net iteration 5 (the main model), U-net iteration 6, respectively), each more complex than the one before it. Each model underwent many iterations during the development, with the goal of improving the results. Although the last and especially the first model didn't provide good and presentable results, they contributed to the learning and the research process. The code can be found in the Github repository of this project.

The Data and Preprocessing

The Datasets

Three distinct datasets were utilized in this research project, each selected to address specific challenges associated with training a deep learning model for image colorization. One of the key datasets was the **"Best Artworks of All Time"** collection, comprising over 16,000 digitized images of paintings by celebrated artists. Originally assembled for an artist classification task, this dataset was repurposed for the early stages of model training due to its wide variety of colors, textures, and compositional styles. Despite some inherent limitations — such as the presence of grayscale works or stylized color schemes — this dataset played a crucial role in helping the model learn a broad and creative representation of color relationships.

When the scope of the project was decided, some models were trained on the **"Human Detection Dataset"**. This dataset contains over 900 images captured from CCTV footage and was originally intended for binary classification tasks, specifically to determine the presence or absence of a person in the frame. Despite its limited size and original purpose, the dataset was chosen because it was CCTV footage.

Another dataset that was used in this project was the **"Image colorization dataset"** which was specifically designed for the image colorization task. This dataset offers a larger volume of training samples with more balanced and realistic color distributions. This dataset contains about 6000 images of general objects or animals, with usually vibrant colors, which improved the accuracy of the models.

The combination of the "Human Detection Dataset" and the "Image colorization dataset" was used for the training of the models.

The Data Preprocessing

For this project, all image data were programmatically aggregated into a single dataset using Python. Each image was resized to a uniform resolution of **256 x 256 pixels** to reduce the computational load. As part of the preprocessing pipeline, the RGB pixel values of the images were converted to the CIELAB (LAB) color space (see appendix 3), which is widely regarded as a perceptually uniform and device-independent color representation system.
When working with the CIELAB color system, there is a need for normalization. The model has to train on values [0, 1] for the L channel and [-1, 1] for the A and B channels. However, when converting an image from the BGR to LAB using the `cvtColor` method of the `opencv` library, we get [0, 255] values for the L, A and B channels. Hence, there is a need for normalization when feeding the channels to the model (find out more about the normalization of the channels on the Github repository).

The Main Model: U-net iteration 5

After a lot of refurbishing, minor changes and iterations, this model worked well, compared to all of the other models that were trained previously. The hyperparameters were tuned by trial and error, after seeing the results of the colorizations and the loss functions' graph s.
The final values of the hyperparameters were:

- **`Number of epochs: 50`**
- **`Learning rate: 0.0001`**
- **`Batch size: 32`**

The encoder path consists of four convolutional blocks, with each block comprising two convolutional layers with ReLU activation and **"`same`"** padding, followed by batch normalization. These layers progressively increase the number of filters from 64 to 512, enabling the model to capture low-level textures in the early layers and more abstract visual features in the deeper layers. After each block, a max pooling layer halves the spatial dimensions, effectively downsampling the feature maps.

At the center of the network is the bottleneck, where two convolutional layers with 1024 filters are applied. This stage encodes the most compressed representation of the input image and captures high-level semantic content that is critical for accurate color prediction.

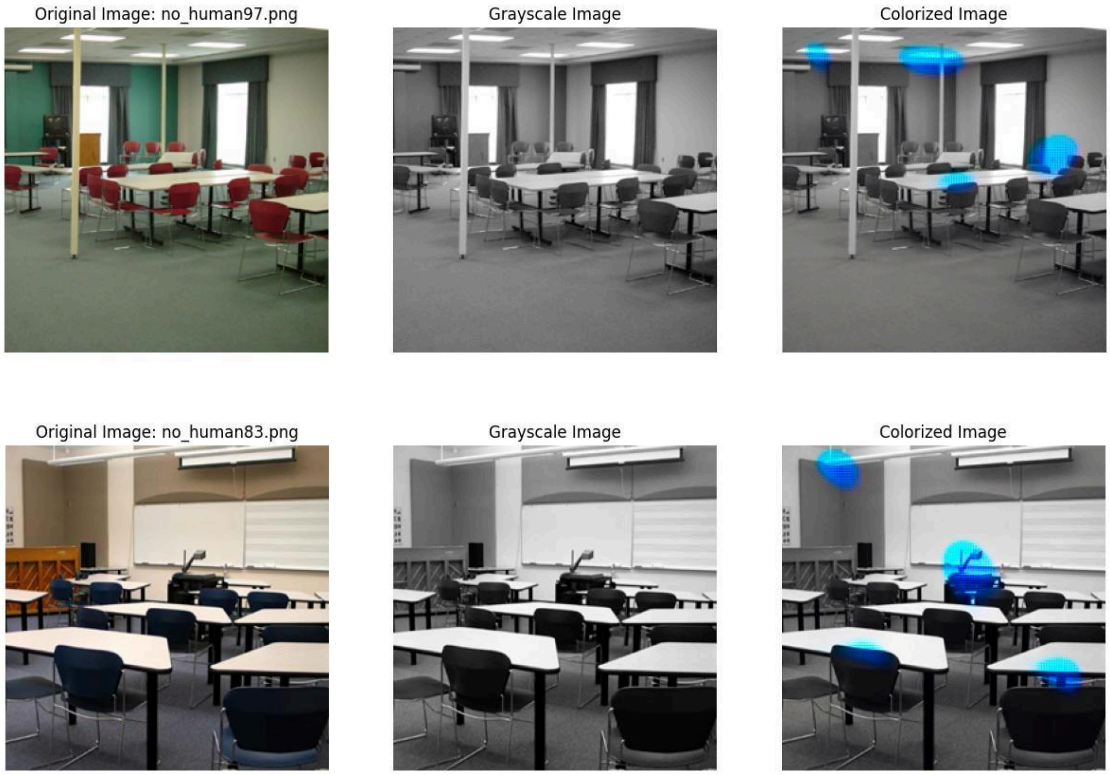
The decoder then upscales the image back to its original resolution. Upsampling is performed using transposed convolutional layers with a stride `(2, 2)`. After each upsampling step, the upsampled feature maps are concatenated with the corresponding feature maps from the encoder via skip connections. This enables the model to combine detailed spatial information from earlier layers with high-level semantic understanding from deeper layers. Each decoder block mirrors the encoder structure, using two convolutional layers followed by batch normalization.

An interesting empirical observation emerged during the very first iterations of this model. Initially, this model was trained solely on the "CCTV Human Detection Dataset". The results

were suboptimal: the majority of the generated outputs remained largely desaturated, with minimal or incorrect chromatic restoration. Most images retained a predominantly gray appearance and failed to reflect the original color distributions present in the original image.

However, one consistent and notable exception was observed across multiple samples — regions containing grass were consistently colorized correctly, often showing accurate green hues. This pattern suggested that the model was indeed capable of learning meaningful color associations from grayscale cues but was limited by the scope and diversity of the training data. The uniform success in colorizing grass, a semantically and texturally consistent object, highlighted the dataset's insufficient variability and coverage of different scene types and objects.

See examples below.



Examples of images that don't contain grass

Examples of image that contain grass

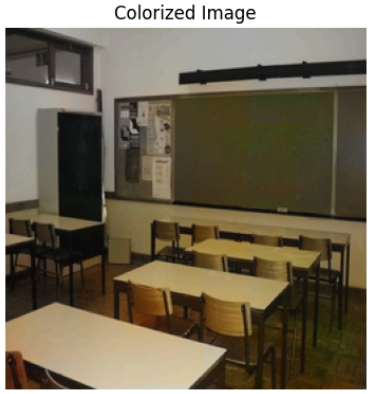
This led to the hypothesis that the primary bottleneck was not the model architecture or the implementation, but rather the quality and domain suitability of the training dataset. To test this, the model was retrained using the combination of the “CCTV Human Detection Dataset” and the “Image Colorization Dataset”, which is specifically curated for learning mappings between luminance and chrominance across a broad range of scenes and textures. As a result of this dataset integration, the model demonstrated notably improved performance, producing outputs with more natural and varied colorizations across different objects and backgrounds.

After training the same model on the combination of “Image colorization dataset” and the “CCTV Human Detection Dataset”, the model gave these results:

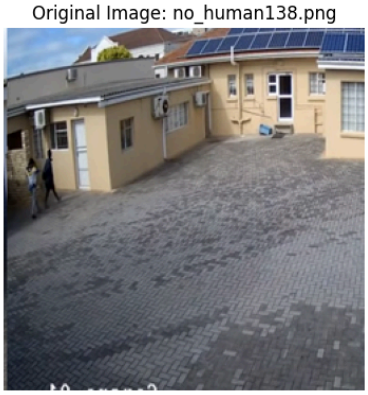
Original Image: no_human197.png

Grayscale Image

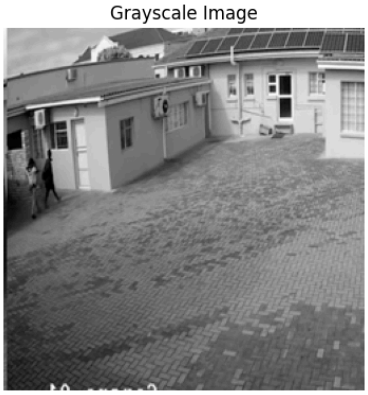
Colorized Image



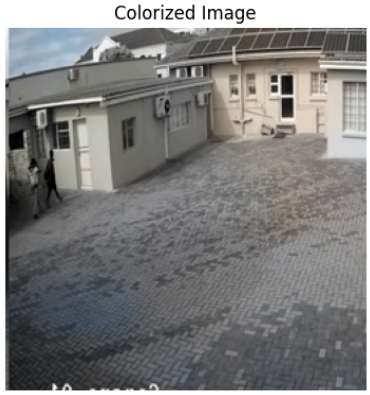
Original Image: no_human138.png



Grayscale Image



Colorized Image

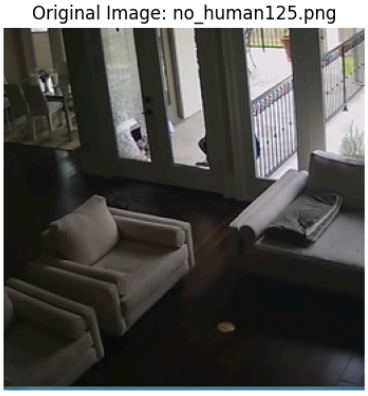


Original Image: no_human156.png

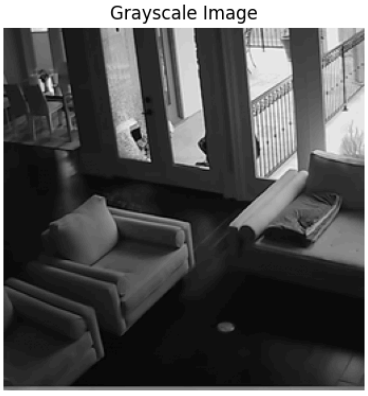
Grayscale Image

Colorized Image

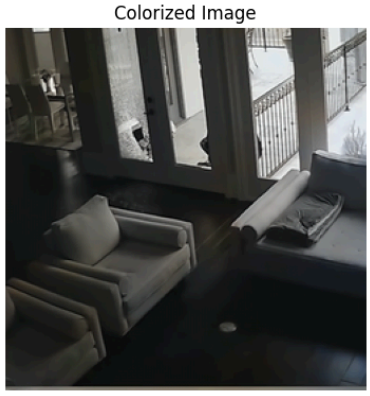
Original Image: no_human125.png



Grayscale Image



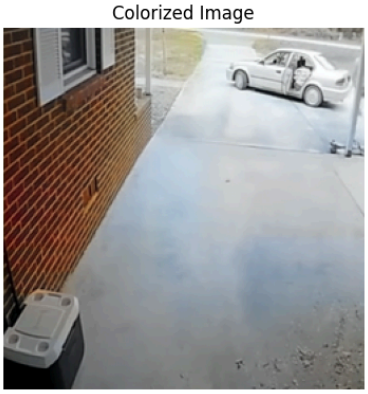
Colorized Image



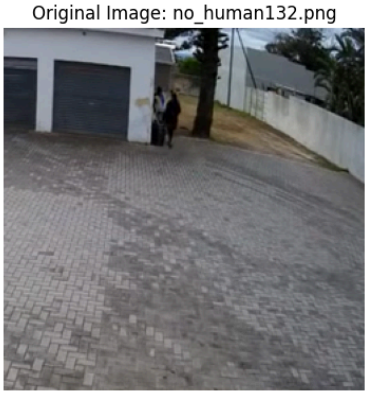
Original Image: no_human165.png

Grayscale Image

Colorized Image



Original Image: no_human132.png



Grayscale Image

Colorized Image

Original Image: 15.jpg

Grayscale Image

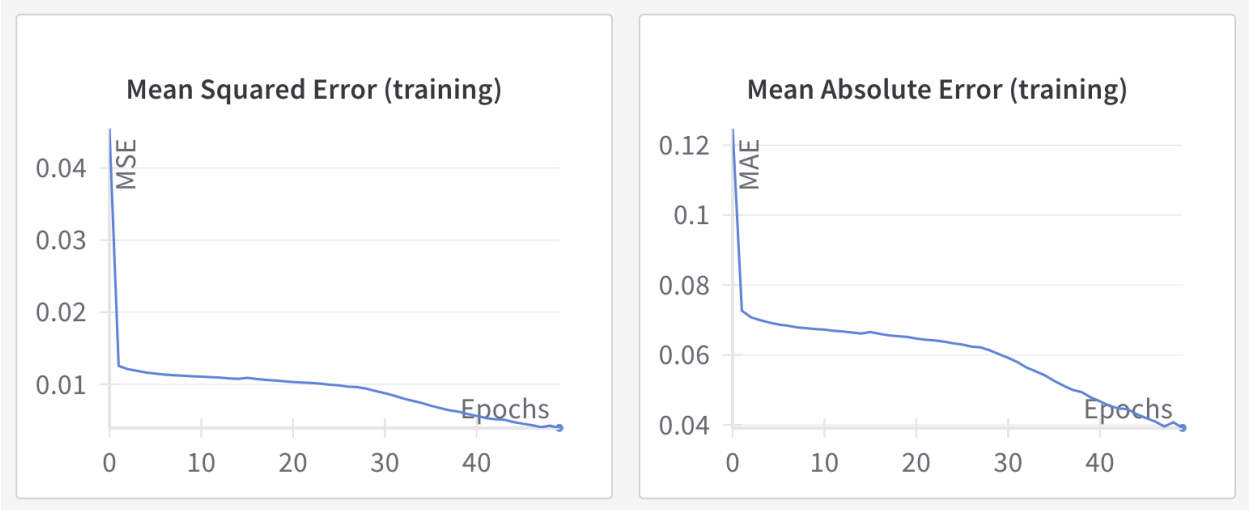
Colorized Image

Original Image: 202.jpg

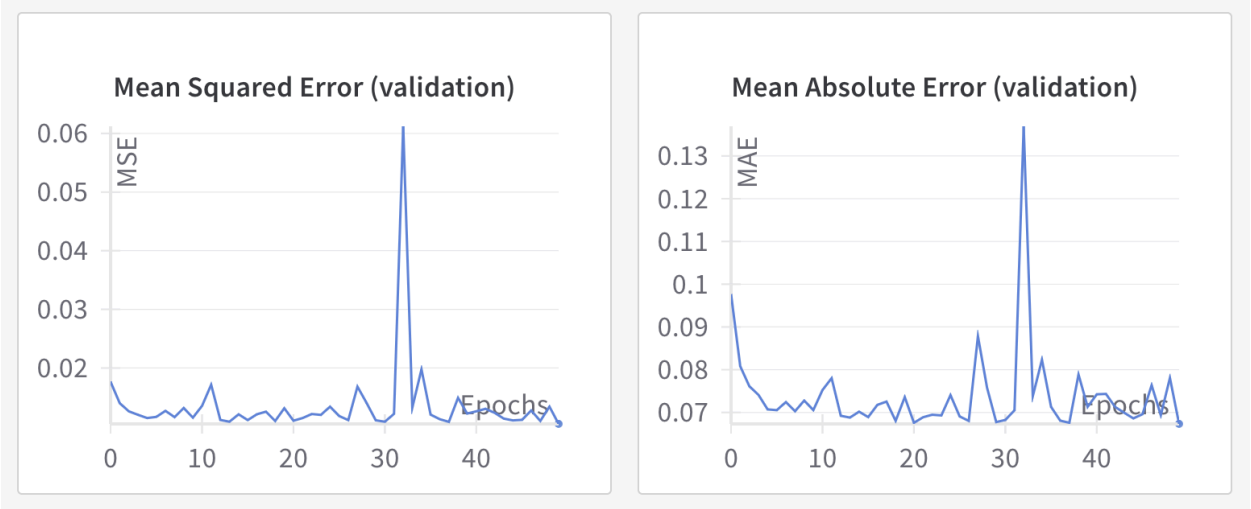
Grayscale Image

Colorized Image

Apart from the visual results, some metrics were used to evaluate the model. Using the Weights and Biases platform and the library integration (`wandb`), the mean squared and absolute error of the validation and the training dataset was tracked.



The MSE and the MAE of the training



The MSE and the MAE of the validation

Other models (CNN, U-net)

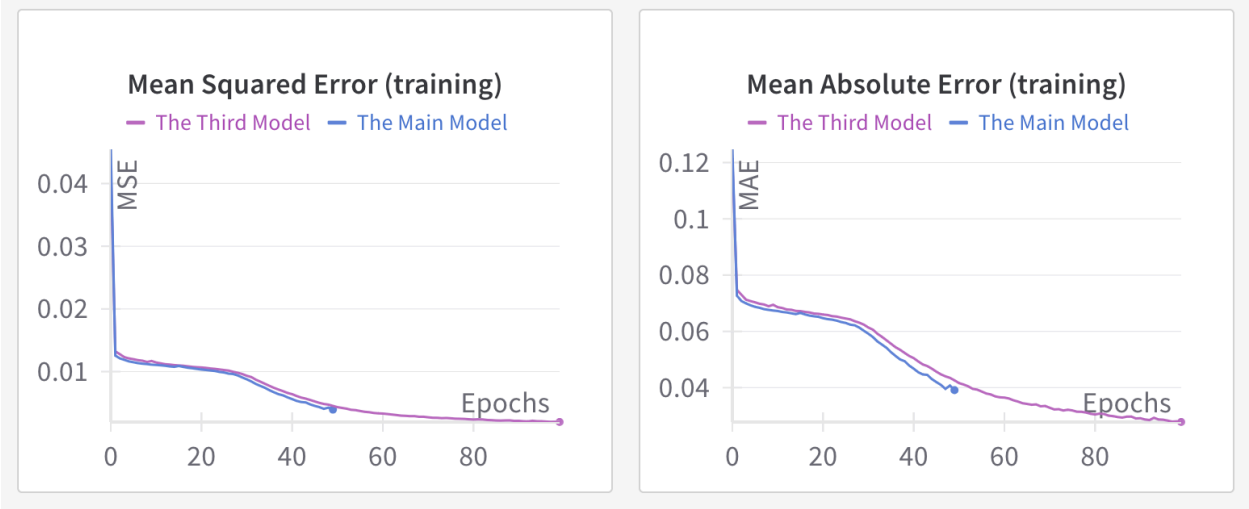
Apart from the main U-net model that was trained in the scope of this project, two other versions were also trained and researched.

The First model: Convolutional Neural Networks

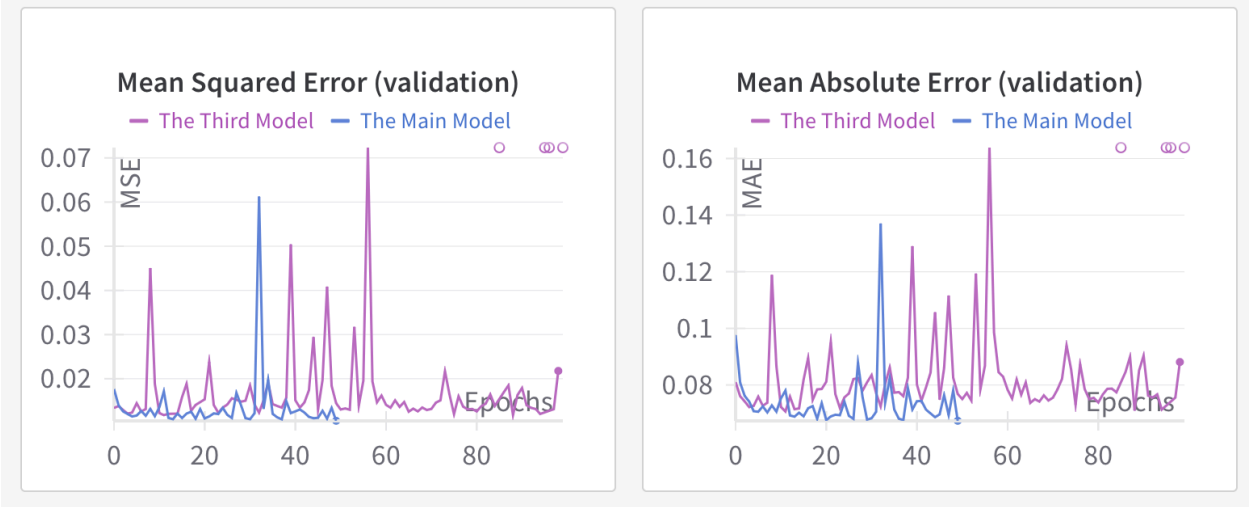
This was the first model that was trained and researched for the capstone project. For research purposes this model was trained on the “Best Artworks of All Time” dataset. After extensive experimentation and multiple iterations, the initial model failed to produce meaningful image colorizations. Consequently, a more advanced architecture was pursued, leading to the development of a U-Net-based model, which ultimately formed the foundation of the final approach.The code of the model can be found in the GutHub repository.

The Third Model: U-net iteration 6

After getting the decent results of the iteration 5 model, some experimentations with a new version with the U-net architecture were done. The key difference between iteration 5 and this one is that in this model, on each layer of the encoder part, an L2 regularizer was applied, also experimentation with dropout layers was conducted. This model did not perform well, and because of time constraints, the work on this model was stopped. This model was trained with 100 epochs, with checkpoints that saved the best version (the one with the lowest loss, in this case, the lowest MAE).
Below you can see the comparison of the MAE and the MSE of the main and the third model.



The MSE and MAE of the training



The MSE and MAE of the validation

As we can see the training loss value of the third model is lower than the value of the second model but also we see great fluctuations in the graph of the validation loss function, and overall higher values compared to the main model.

Challenges Faced During This Project

During the course of this project, several challenges emerged that were both significant and intellectually engaging to address. One of the primary difficulties encountered was securing adequate computational resources. Initially, while training relatively simple models and conducting preliminary research, the computational capabilities of a personal laptop were sufficient. However, as the project progressed to the training of more complex architectures such as U-Net, the limitations of local hardware became apparent. To overcome this, cloud-based solutions were utilized—specifically, paid access to Google Colab's enhanced

computing resources. This reliance on cloud infrastructure not only facilitated the continuation of the project but also heightened the importance of code optimization. Consequently, efforts were made to streamline the codebase, resulting in a cleaner, more efficient implementation.

Another substantial challenge involved the acquisition of suitable training data. Given that the objective of the project was to colorize CCTV footage, sourcing a comprehensive dataset that featured vibrant colors and diverse, recognizable objects proved to be particularly difficult. Most publicly available CCTV datasets are limited in variety and quality, often consisting of images captured from steep downward angles, with much of the frame occupied by ground surfaces and only partial views of objects. These limitations impeded the model's ability to generalize effectively from the available data. To mitigate this issue, a secondary dataset (the "Image colorization dataset") was incorporated into the training process, thereby enhancing the model's exposure to a broader range of visual elements and improving overall performance.

Conclusion

This project focused on the application of deep learning techniques for the colorization of grayscale CCTV footage. Throughout the development process, multiple models were researched, implemented, and evaluated, ultimately leading to the adoption of a U-Net architecture, which yielded the most promising results in terms of performance and visual quality. A significant challenge encountered was the acquisition of suitable datasets—specifically, footage that not only resembled real-world CCTV environments but also contained sufficient variety in objects and scenes to enable effective model training and generalization.

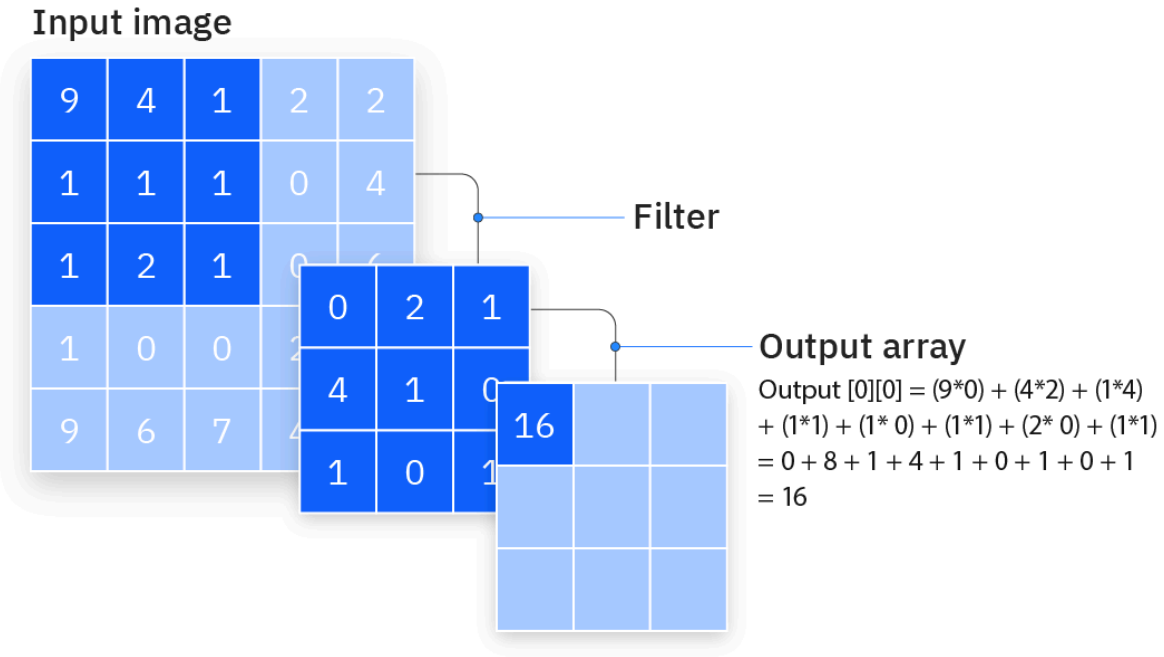
Despite entering the project with no prior experience in computer vision, the capstone provided a valuable opportunity for in-depth exploration and skill development in this domain. It facilitated practical application of theoretical knowledge acquired throughout the university curriculum and fostered a deeper understanding of neural networks, data preprocessing, and model optimization. The successful execution of this project not only met the initial learning objectives but also served as a demonstration of the competencies gained during the course of the degree program.

Appendix

1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a class of deep learning models particularly well-suited for analyzing visual imagery. Inspired by the visual cortex of animals, CNNs have become the dominant approach for image classification, object detection, segmentation, and related computer vision tasks.

A typical CNN consists of multiple layers, including **convolutional layers**, **pooling layers**, and **fully connected layers**. The **convolutional layers** apply a set of learnable filters (or kernels) to the input image to extract local patterns such as edges, textures, and shapes. Each filter slides across the image, computing a dot product between the filter weights and the receptive field of the image, producing a feature map that highlights specific spatial features.

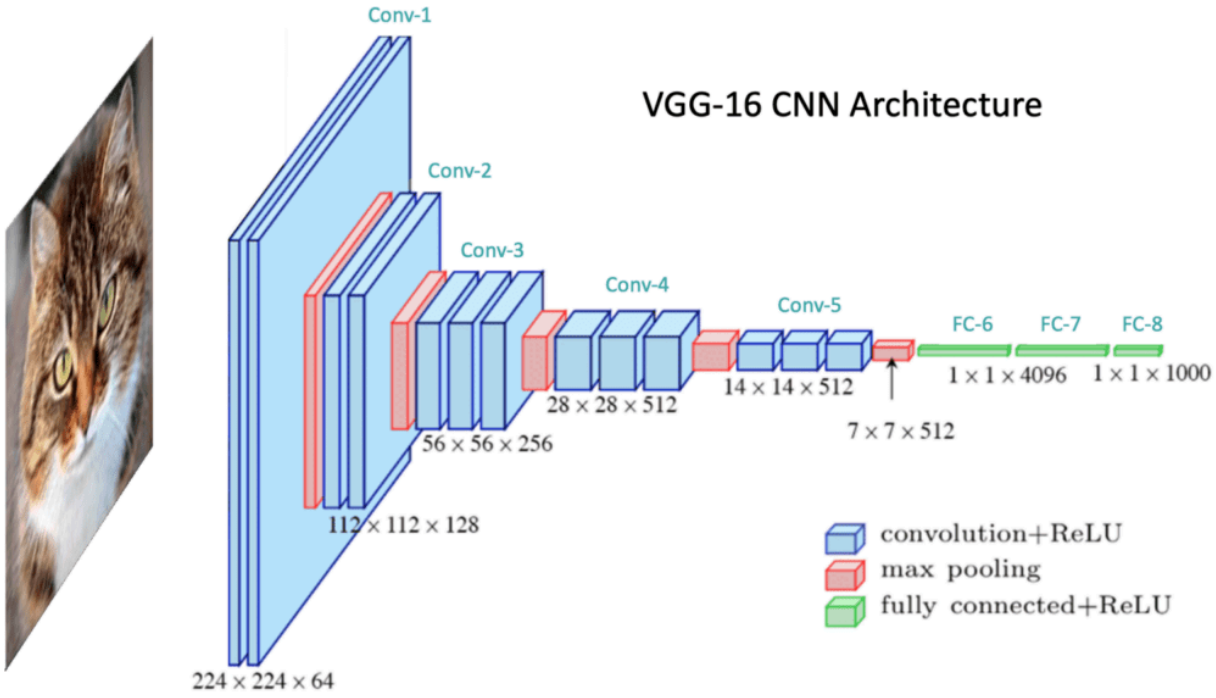


An example of a convolution operation

Pooling layers, often using max or average pooling, are interspersed between convolutional layers to reduce the spatial dimensions of the feature maps. This downsampling operation decreases computational load and introduces a degree of translational invariance. As the network deepens, it captures increasingly abstract and complex features.

Finally, **fully connected layers** are typically used at the end of the network to combine features extracted from previous layers and produce a final prediction, the AB channels in our case. In the case of segmentation or detection, the fully connected layers may be replaced or augmented with other structures to maintain spatial resolution in the output.

CNNs are trained using large datasets and supervised learning techniques, where a loss function (e.g., cross-entropy loss) guides the optimization of model weights through backpropagation and stochastic gradient descent. Their architectural flexibility and ability to learn hierarchical feature representations make CNNs a foundational tool in modern computer vision.

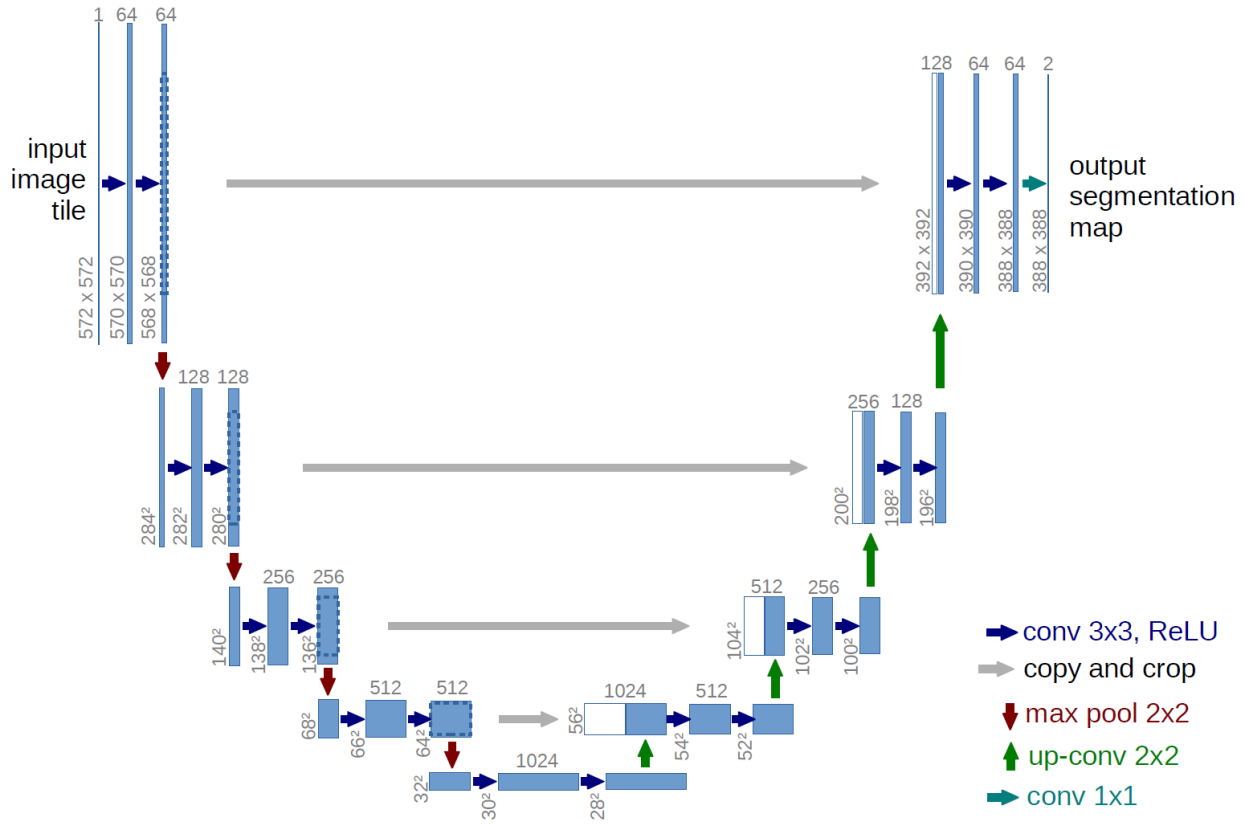


An example of a CNN implementation

2. U-net

U-Net is a type of convolutional neural network (CNN) architecture specifically designed for semantic segmentation tasks, particularly in the field of biomedical image analysis. It was first introduced by Ronneberger et al. in 2015 and has since become a foundational model for tasks requiring pixel-level classification.

The architecture of U-Net is characterized by a symmetric encoder–decoder structure. The **encoder** (contracting path) is composed of successive convolutional layers followed by pooling operations, which progressively reduce the spatial dimensions of the input while increasing the depth of feature maps. This allows the model to capture high-level contextual features. In contrast, the **decoder** (expanding path) uses upsampling operations (e.g., transposed convolutions) to gradually restore the original resolution of the input image.



An example of a U-net implementation

A key innovation of U-Net is the introduction of **skip connections** between corresponding layers in the encoder and decoder paths. These connections concatenate feature maps from the encoder to the decoder at the same hierarchical level, allowing the network to preserve fine-grained spatial information that might otherwise be lost during downsampling. This mechanism enhances the precision of the segmentation boundaries, which is critical in domains like medical imaging.

U-Net models are trained end-to-end using annotated datasets, with a loss function typically based on pixel-wise classification error, such as cross-entropy loss or Dice loss. Their ability to work effectively with relatively small amounts of data and to produce high-resolution output has made them widely adopted in both academic and applied research.

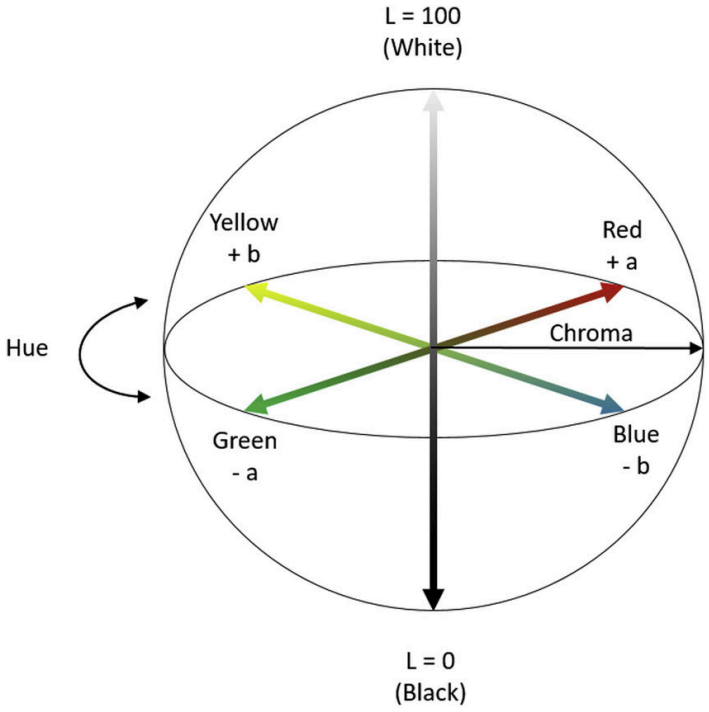
3. Hyperparameters

Epochs is the number of complete passes through the entire training dataset. During each epoch, the model processes all training examples once, adjusting its weights based on the calculated loss. Multiple epochs (in the case of this project usually 50) are often required to achieve convergence, as the model incrementally improves its predictions over successive iterations. However, an excessively high number of epochs can lead to **overfitting**, where the model learns to perform well on training data but generalizes poorly to unseen inputs.

Batch size defines the number of training samples processed before the model's weights are updated. Training with the entire dataset at once (known as full-batch gradient descent) is computationally expensive and often impractical for large datasets. Instead, models are typically trained using **mini-batches**. A smaller batch size results in noisier updates but can lead to better generalization. Larger batches provide more stable updates but require more memory and may lead to convergence at sharp minima, potentially reducing model robustness.

The **learning rate** controls the size of the steps the model takes while updating its weights in response to the loss gradient. A higher learning rate allows for faster convergence but risks overshooting the optimal solution or causing instability in training. Conversely, a very low learning rate may lead to slow convergence or getting stuck in suboptimal local minima. Proper tuning of the learning rate is essential for ensuring efficient and stable learning.

4. The LAB color system



The LAB color system

The **CIELAB color space**—commonly referred to as **LAB**—is a color representation model developed by the International Commission on Illumination (CIE) designed to approximate human vision. Unlike the more familiar RGB color space, which encodes color based on device-dependent red, green, and blue components, LAB is a **device-independent** model that separates color into three distinct channels: **L** for lightness, and **A** and **B** for the color-opponent dimensions. Specifically, the A channel represents the green–red axis, while the B channel represents the blue–yellow axis.

One of the key advantages of the LAB color space is its **perceptual uniformity**—a given numerical change in values corresponds more closely to a consistent change in perceived color. This makes it especially useful in tasks where the goal is to analyze or manipulate images based on human visual perception.

In the context of **image colorization**, LAB is especially useful because it naturally separates the **luminance (L)** component from **chrominance (A and B)**. This decomposition allows a neural network to learn only the chrominance components, while preserving the original luminance structure of the input image. Moreover, working with already separated channels is computationally lighter than having to deal with the RGB color system.

References

Academic papers and resources

IBM. (n.d.). *Convolutional neural networks*. IBM. Retrieved May 8, 2025, from https://www.ibm.com/think/topics/convolutional-neural-networks

ResearchGate. (n.d.). *The CIELAB color space diagram*. Retrieved May 8, 2025, from https://www.researchgate.net/figure/The-CIELAB-color-space-diagram-The-CIELAB-or-CIE-L-a-b-color-system-represents_fig1_338303610

Shariatnia, M. (2020, November 18). *Colorizing black & white images with U-Net and conditional GAN – A Tutorial*. Towards Data Science. Retrieved May 8, 2025, from https://towardsdatascience.com/colorizing-black-white-images-with-u-net-and-conditional-gan-a-tutorial-81b2df111cd8/

Datasets

Werner, C. (2021). *Human detection dataset* [Data set]. Kaggle. https://www.kaggle.com/datasets/constantinwerner/human-detection-dataset

Ikarus777. (2019). *Best artworks of all time* [Data set]. Kaggle. https://www.kaggle.com/datasets/ikarus777/best-artworks-of-all-time

Aayush9753. (2022). *Image colorization dataset* [Data set]. Kaggle. https://www.kaggle.com/datasets/aayush9753/image-colorization-dataset

Codes and inspirations

Zhang, R. (n.d.). *Colorization* [Source code]. GitHub. Retrieved May 8, 2025, from https://github.com/richzhang/colorization/tree/master

Antic, J. (n.d.). *DeOldify* [Source code]. GitHub. Retrieved May 8, 2025, from https://github.com/jantic/DeOldify?tab=readme-ov-file#easiest-approach

Berkay, M. (n.d.). *Image colorization* [Source code]. GitHub. Retrieved May 8, 2025, from https://github.com/mberkay0/image-colorization?tab=readme-ov-file