IE402
Optimization Project Report

Image Restoration Using Convolutional AutoEncoders

Group - 5

Assigned By - Proff. Nabin Kumar Sahu

Group Members

- **202201129** - Tasmay Patel
- **202201207** - Swayam Hingu
- **202201253** - Smit Fefar
- **202201504** - Kishan Pansuriya
- **202201525** - Heer Shah

Contents

1 Abstract

In this study, we focus on improving image restoration—bringing high-quality images back from noisy, blurry, or compressed versions. We use convolutional autoencoders (CAEs), a type of neural network designed for this purpose, which include layers that shrink and then expand images to remove noise and restore details.

To make our model effective, we use the Adam optimizer to minimize a custom loss function. This function measures the difference between the color values of the restored image and the original, pushing the model to produce images with accurate colors and fine details. Thanks to Adam's adaptive learning rate, our model learns quickly and reliably, even when images are significantly degraded.

We test our model on standard image restoration datasets, covering both synthetic and real-world scenarios, and find that it often outperforms traditional methods and recent deep-learning approaches in terms of accuracy, visual quality, and efficiency. The combination of downsampling and upsampling layers, along with our color-based loss function and the Adam optimizer, allows the CAE to work well across various types of degraded images.

Our findings show promise for real-world applications like photography, video processing, and medical imaging, where high-quality restoration is essential. This work highlights the power of deep learning and advanced optimization in improving image processing, moving us closer to smarter, more adaptable visual enhancement tools.

2 Introduction

2.1 Objective

Image restoration aims to involves **recovering a high-quality image from a degraded or distorted version.** This is a significant challenge in computer vision, particularly in applications such as digital photography, video processing, and medical imaging. One common example is converting **grayscale images into color images**, where the goal is to infer the missing color information based on luminance data.

Autoencoders are a learning architecture comprising of two networks: **an encoder and a decoder.** The encoder, through a series of Convolutional layers and downsampling, learns a compressed representation of the input data, while the decoder reconstructs the data from these representations. A well-trained decoder is able to regenerate data that is identical or as close as possible to the original input data. Autoencoders are widely used for anomaly detection, image denoising, and colorization. Here, we are **focusing on colorizing landscape images using autoencoders.**

In this report, we focus on developing an optimal solution to a widely recognized problem in computer vision—image restoration. This problem can be described as follows: **"Given a Grayscale image, reconstruct a Color image by removing unwanted distortions and restoring missing details."** Our approach utilizes Convolutional Autoencoders (CAEs), a deep learning model specifically designed to capture intricate spatial features in images, making it well-suited for image restoration tasks.

We have implemented a Python-based CAE model that takes in Grayscale images as input, processes them through multiple layers of encoding and decoding, and outputs a reconstructed image with minimized distortions. The code accepts a set of Grayscale images and trains the CAE to learn optimal feature representations, allowing it to generate clearer, more accurate reconstructions.

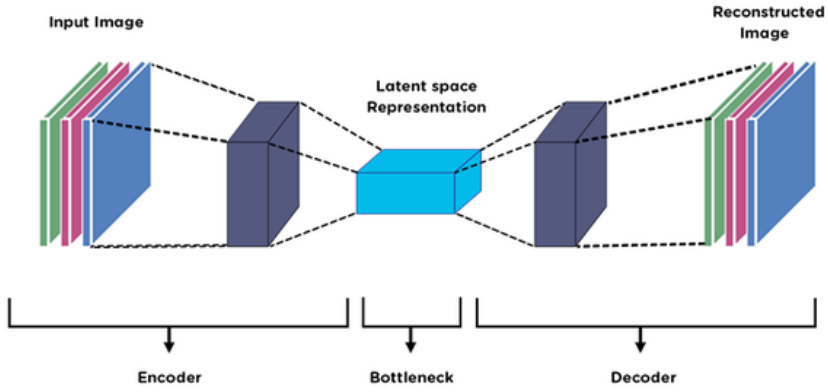

Figure 1: Autoencoders Overview

2.2 Problem Statement

1. **Data Representation:**

   - We have a dataset of paired images, where each pair consists of a greyscale image x_i and its corresponding color image y_i. The set of greyscale images is denoted as $X = \{x_1, x_2, \ldots, x_N\}$ and the set of color images as $Y = \{y_1, y_2, \ldots, y_N\}$.
   - The objective is to find a mapping function f such that $f(x_i) \approx y_i$ for all i.

2. **Mathematical Representation:**

   - The mapping can be mathematically expressed as:

     $$\hat{y}_i = f(x_i; \theta)$$

     where $\hat{y}_i$ is the predicted color image generated by the model given the parameters θ.

3. **Reconstruction Error:**

   - To evaluate the quality of the reconstructed images, we introduce a loss function that quantifies the difference between the predicted images $\hat{y}_i$ and the true images y_i. A common choice for this loss function is the **Mean Absolute Error (MAE):**

     $$L(\theta) = \frac{1}{N} \sum_{i=1}^{N} \|y_i - \hat{y}_i\|_1$$

   - **The goal is to minimize this loss function, effectively improving the model's ability to predict accurate color representations from grayscale images.**

2.3 Motivation For Using Autoencoders

Autoencoders offer a promising solution for image restoration tasks for several reasons:

1. **Feature Learning:** Autoencoders automatically learn feature representations from the data, enabling them to capture essential characteristics necessary for reconstructing images.

2. **Dimensionality Reduction:** By compressing the input data into a lower-dimensional latent space, autoencoders can effectively reduce noise and focus on the most critical aspects of the image.

3. **Flexibility:** The architecture of autoencoders can be easily adapted to accommodate different types of data and specific requirements of the image restoration task.

4. **End-to-End Learning:** Autoencoders facilitate an end-to-end learning approach, allowing the model to optimize directly for the desired output without the need for extensive feature engineering.

2.4 Types Of Images

2.4.1 GrayScale Images

A grayscale image is represented as a **two-dimensional matrix I of pixel intensities,** where each element I_{ij} corresponds to the brightness of the pixel at position (i, j). These intensity values range between 0 and 255 (in an 8-bit format) or between 0 and 1 (when normalized).

$$I \in \mathbb{R}^{m \times n}$$

where m and n are the dimensions of the image.

Figure 2: GrayScale Image

2.4.2 Color Images

A color image is an image that contains information about the colors of the objects or scenes it represents. In a digital color image, each pixel is represented by a combination of **red, green, and blue (RGB)** values that determine the intensity of each color channel. Other color models, such as the cyan, magenta, yellow, and key (CMYK) model used in printing, may also be used for color images.

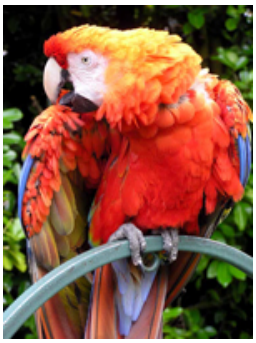

Figure 3: RGB Color Image

2.5 Challenge Faced In Grayscale To Color Mapping

- **In a greyscale image, only the intensity of light is represented, not the specific hues or saturation.** For instance, the intensity of a grey pixel could correspond to many possible colors. A tree trunk, a building, and even a human face could all have similar greyscale values but vastly different color values.

- **This means that the model has to "guess" the color**, often relying on contextual clues and patterns to make an educated choice, but it may not always guess correctly.

2.6 Color Spaces

2.6.1 The RGB Space

In the RGB color space, each color is represented as a combination of red, green, and blue values, typically ranging from 0 to 255. For example, pure red would be represented as (255,0,0), pure green as (0,255,0), and pure blue as (0,0,255). Other 15 colors are represented as a combination of these values, such as yellow as (255,255,0) and purple as (128,0,128). When all three colors are added together at full intensity (255,255,255), it creates pure white, while adding no colors (0,0,0) creates pure black.

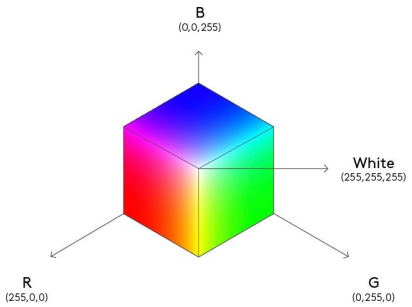

Figure 4: RGB Space

2.6.2 LAB Color Space

The LAB color space is a device-independent color model that separates color information into three components: lightness (L), green-red (a), and blue-yellow (b). This makes it well-suited for color correction, image analysis, and other applications where precise color information is important. In the LAB color space, lightness (L) ranges from 0 (pure black) to 100 (pure white), while the green-red (a) and blue-yellow (b) axes can range from -128 to 127. Positive values along the a-axis represent green tones, while negative values represent 16 red tones. Similarly, positive values along the b-axis represent yellow tones, while negative values represent blue tones.

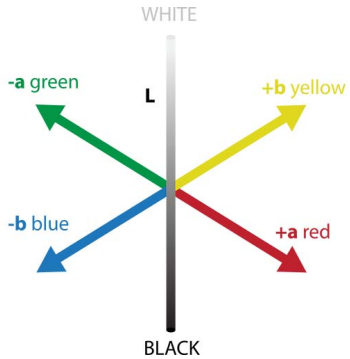

Figure 5: LAB Space

3 Mathematical Theory

3.1 Image Representation As Matrices

3.1.1 GrayScale Image Representation

A grayscale image can be represented as a 2D matrix G, where each element g_{ij} represents the intensity of the pixel at position (i, j). For an image of size $H \times W$ (Height $\times$ Width), the grayscale image matrix G is defined as:

$$G = \begin{bmatrix} g_{11} & g_{12} & \cdots & g_{1W} \\ g_{21} & g_{22} & \cdots & g_{2W} \\ \vdots & \vdots & \ddots & \vdots \\ g_{H1} & g_{H2} & \cdots & g_{HW} \end{bmatrix}$$

where $g_{ij} \in [0, 1]$ for normalized intensity values.

3.1.2 Color Image Representation

An RGB color image is represented as a 3D tensor C of size $H \times W \times 3$, where each matrix in the tensor corresponds to one color channel (Red, Green, or Blue):

$$C = \begin{bmatrix} \begin{bmatrix} c_{111} & c_{112} & \cdots & c_{11W} \\ c_{121} & c_{122} & \cdots & c_{12W} \\ \vdots & \vdots & \ddots & \vdots \\ c_{1H1} & c_{1H2} & \cdots & c_{1HW} \end{bmatrix} \\ \begin{bmatrix} c_{211} & c_{212} & \cdots & c_{21W} \\ c_{221} & c_{222} & \cdots & c_{22W} \\ \vdots & \vdots & \ddots & \vdots \\ c_{2H1} & c_{2H2} & \cdots & c_{2HW} \end{bmatrix} \\ \begin{bmatrix} c_{311} & c_{312} & \cdots & c_{31W} \\ c_{321} & c_{322} & \cdots & c_{32W} \\ \vdots & \vdots & \ddots & \vdots \\ c_{3H1} & c_{3H2} & \cdots & c_{3HW} \end{bmatrix} \end{bmatrix}$$

where $c_{ijk} \in [0, 1]$ for normalized intensity values of the color channels.

3.2 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique used to reduce the dimensionality of a dataset while preserving as much of its variability (or information) as possible. By identifying the main directions (or axes) along which the data varies the most, PCA projects the data onto these axes, called principal components.

3.2.1 Theory

PCA aims to find a lower-dimensional representation of data while maximizing variance. The key steps involved in PCA are:

- Standardization of data to ensure that each feature contributes equally to the analysis.
- Calculation of the covariance matrix to understand the relationships between features.
- Extraction of eigenvalues and eigenvectors from the covariance matrix to identify the principal components.
- Selection of the top k principal components based on the eigenvalues.
- Projection of the original data onto the new lower-dimensional space.

The principal components are orthogonal to each other, which means they are uncorrelated and capture the maximum variance of the data.

3.3 Autoencoder

An autoencoder is a type of artificial neural network used primarily for unsupervised learning tasks, such as data compression, noise reduction, and dimensionality reduction. It consists of two main components:

- **Encoder:** Compresses the input data into a lower-dimensional representation.
- **Decoder:** Reconstructs the original data from this compressed representation.

Autoencoders are trained by minimizing the difference between the input and the reconstructed output, encouraging the model to learn efficient, compact representations of the data. They are widely used in applications like image and audio processing, anomaly detection, and generative modeling.We are using convolution autoencoder for our project.

3.4 Why Choose Autoencoders Over PCA ?

- **Non-linearity:** Autoencoders can learn complex, non-linear transformations due to their use of neural networks, while PCA is limited to linear transformations.
- **Higher-dimensional representation:** Autoencoders can capture more intricate patterns in data and work well with higher-dimensional datasets compared to PCA, which can struggle with complex structures.
- **Noise reduction and denoising:** Autoencoders can be specifically adapted for denoising tasks, whereas PCA does not explicitly support this functionality.

3.5 Autoencoder Processing

3.5.1 Forward Propagation In The Encoder

The encoder takes the grayscale image G as input and applies a series of convolutional operations to extract meaningful features from the image. Each convolutional layer in the encoder uses filters (or kernels) K that scan across the input, capturing spatial hierarchies and patterns at various levels.
The convolution operation is expressed mathematically as:

$$F = G * K$$

where F represents the feature map resulting from convolving the input G with the kernel K.
In forward propagation, each layer applies an activation function to the feature map to introduce non-linearity, which helps the model capture complex patterns. The output of each layer then serves as the input for the subsequent layer, gradually building a higher-dimensional representation of the grayscale image. By stacking multiple convolutional layers, the encoder compresses the information, capturing essential features that are critical for accurate colorization in the later stages.

3.5.2 Downsampling

- **Strided Convolutions:** Strided convolutions reduce the spatial dimensions of an input feature map while simultaneously learning features. This is accomplished by applying a convolution operation with a stride greater than 1.

  Mathematical Formulation: Given an input feature map F of size $H \times W$ and a convolution kernel K of size $k \times k$, the output feature map F' after applying a strided convolution with stride s can be defined as:

  $$F'_{ij} = \sum_{m=0}^{k-1} \sum_{n=0}^{k-1} F_{(si+m)(sj+n)} K_{mn}$$

  where F' will have dimensions approximately $\left\lfloor \frac{H}{s} \right\rfloor \times \left\lfloor \frac{W}{s} \right\rfloor$.

- **Max Pooling:** Max pooling reduces the spatial dimensions of the input feature map by taking the maximum value within a defined pooling window, retaining the most salient features while discarding irrelevant information.

  Mathematical Formulation: Given an input feature map F of size $H \times W$ and a pooling size of $p \times p$, the output feature map F' after applying max pooling with stride s can be defined as:

  $$F'_{ij} = \max_{(m,n) \in \text{pooling region}} F_{(si+m)(sj+n)}$$

  The output dimensions will be approximately $\left\lfloor \frac{H}{s} \right\rfloor \times \left\lfloor \frac{W}{s} \right\rfloor$.

3.5.3 Activation Functions

Activation functions introduce non-linearity into the model, enabling the autoencoder to learn complex mappings from inputs to outputs. Common activation functions include ReLU, sigmoid, and tanh.

ReLU Activation Function: The ReLU activation function is defined as:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

Leaky ReLU Activation Function: The Leaky ReLU function addresses the vanishing gradient problem by allowing a small, non-zero gradient when the input is negative:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases}$$

where α is a small constant (e.g., $\alpha = 0.01$). This allows the model to maintain some gradient for negative inputs, mitigating the vanishing gradient problem.

3.5.4 Upsampling

In the decoder, transposed convolutions are applied to upsample the feature map F' back to the original image dimensions. This process effectively reverses the downsampling that occurred in the encoder, allowing the model to reconstruct the final output image.
Mathematically, this upsampling operation can be expressed as:

$$\hat{G} = F' \uparrow K^T$$

where K^T represents the transposed convolution kernel.
The output of the decoder is then given by:

$$\hat{G}_{ij} = \sum_{m=-k}^{k} \sum_{n=-k}^{k} F'_{(2i+m)(2j+n)} k_{mn}$$

In simpler words, upsampling works by taking the compressed feature map F' and increasing its size to match the dimensions of the original grayscale image. This is done by applying a special kind of convolution that spreads out the values in the feature map, effectively "filling in" the gaps to create a larger image. The goal of this process is to ensure that the final output image $\hat{G}$ retains the important features learned during encoding while being resized back to the original dimensions. By using upsampling, the decoder can accurately reconstruct the image, leading to better colorization results.

3.5.5 Skip Connections

To further enhance the reconstruction quality, skip connections are employed within the autoencoder. These connections concatenate corresponding encoder and decoder features to preserve spatial information that may be lost during downsampling. For the skip connection between encoder output E and decoder input D, we define:

$$C_{\text{skip}} = [D, E]$$

where $[D, E]$ denotes concatenation along the channel dimension. By integrating features from both the encoder and decoder, the model can maintain fine-grained details and improve overall image quality.

3.5.6 Loss Function

To evaluate the autoencoder's performance in colorizing images, we define the loss function L as the mean absolute error (MAE):

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

where N is the number of training samples, C_i is the ground truth color image, and $\hat{C}_i$ is the predicted color image. This loss function quantifies the discrepancy between the predicted and actual images. By minimizing L, the autoencoder improves its colorization accuracy, utilizing upsampling and skip connections to enhance the final output quality.

4 Adam Optimization

4.1 Need For Optimizer

The task of colorizing an image involves training the model to map grayscale images (input) to their corresponding color images (output). The neural network learns this mapping by adjusting its internal weights based on how well its predictions match the actual color values of the images. **To guide this learning process, we need an optimization algorithm that updates the weights efficiently, helping the model improve its predictions over time.** That's where Adam comes in.

4.2 What Is Adam Optimization?

Adam (Adaptive Moment Estimator) is a stochastic gradient descent (SGD) optimization algorithm used to train deep neural networks. It was introduced by Kingma and Ba in 2014 as a modification to the RMSProp algorithm. It requires less memory and is efficient. Intuitively, it is a combination of the **'gradient descent with momentum' algorithm and the 'RMSP' algorithm.**

4.3 How Adam Optimizer Works ?

4.3.1 Forward Pass (Predictor)

1. The neural network (autoencoder) takes a grayscale image as input, processes it through its layers, and outputs a predicted color image.

2. The output is typically in the RGB color space, where the model predicts the 'R', 'G', and 'B' color channels (representing color information) while the input grayscale image provides the 'L' (lightness) channel.

4.3.2 The Loss Function

The loss function (**Mean absolute Error**) calculates the difference between the **predicted color channels ('R', 'G', 'B') and the true color channels from the original color image.** This loss represents how far off the model's predictions are from the actual colors.

$$\mathbf{L} = \frac{1}{N}\sum_{i=1}^{N}\Big(|\mathbf{R}_i - \hat{\mathbf{R}}_i| + |\mathbf{G}_i - \hat{\mathbf{G}}_i| + |\mathbf{B}_i - \hat{\mathbf{B}}_i|\Big) \tag{1}$$

where N is the number of pixels, $\mathbf{R}_i$, $\mathbf{G}_i$, and $\mathbf{B}_i$ are the true color channel values for pixel i, and $\hat{\mathbf{R}}_i$, $\hat{\mathbf{G}}_i$, and $\hat{\mathbf{B}}_i$ are the predicted color channel values for pixel i.

4.3.3 Backward Propagation (Gradient Calculation)

During backpropagation, the **gradients (partial derivatives) of the loss function concerning each weight in the neural network are computed.** These gradients indicate how much each weight contributed to the error, giving the model a direction to update them to reduce the error.

4.3.4 Two Main Algorithms

The Adam optimizer adjusts the weights of the model using the gradients calculated during backpropagation. Two key components of Adam are:

1. **Momentum:** Accelerates the gradient descent algorithm by using the exponentially weighted average of the gradients, which helps the algorithm converge faster.

$$\mathbf{m_t} = \beta_1 \mathbf{m_{t-1}} + (1 - \beta_1) \left[\frac{\partial L}{\partial w_t} \right] \tag{2}$$

   where:

   - $\mathbf{m_t}$: Aggregate of gradients at time t (current, initially $\mathbf{m_t} = 0$)
   - $\mathbf{m_{t-1}}$: Aggregate of gradients at time $t - 1$ (previous)
   - $\frac{\partial L}{\partial w_t}$: Derivative of the loss function concerning the weights at time t
   - β_1: Moving average parameter (typically a constant around 0.9)

2. **RMSprop:** An adaptive learning algorithm that improves on AdaGrad by using the exponential moving average of squared gradients instead of their cumulative sum.

$$\mathbf{v_t} = \beta_2 \mathbf{v_{t-1}} + (1 - \beta_2) \left[\frac{\partial L}{\partial w_t} \right]^2 \tag{3}$$

   where:

   - $\mathbf{v_t}$: Sum of the square of past gradients (initially $\mathbf{v_t} = 0$)
   - $\mathbf{v_{t-1}}$: Previous sum of squared gradients
   - $\frac{\partial L}{\partial w_t}$: Derivative of the loss function concerning the weights at time t
   - β_2: Moving average parameter (typically a constant around 0.9)

4.4 Bias Correction in Adam

In the Adam optimizer, both m_t (the first moment estimate) and v_t (the second moment estimate) are initialized as zero, which leads to an initial bias toward zero. This bias is especially significant in the early steps of optimization, where β_1 and β_2 are typically close to 1. To address this, Adam applies **bias-corrected versions of** m_t **and** v_t to provide unbiased estimates of the gradient and its variance, ensuring more accurate parameter updates.

4.4.1 Bias-Corrected First Moment

To counteract the bias in m_t, we introduce a bias-corrected first moment, defined as:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{4}$$

This correction scales m_t so that its expected value $\mathbb{E}[\hat{m}_t]$ aligns with the true gradient g, yielding an unbiased estimate of the gradient's mean.

4.4.2 Bias-Corrected Second Moment

Similarly, the second moment v_t is corrected to remove its initial bias:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{5}$$

This correction adjusts v_t so that $\mathbb{E}[\hat{v}_t]$ approximates the square of the gradient, g^2, providing an unbiased estimate of the gradient's variance.
By applying these bias-corrected moments, Adam improves the reliability of the adaptive learning rate, particularly in the early stages of training when biases would otherwise impact convergence.

4.5 Derivation of Bias Correction Factors

In this section, we derive the bias correction factors for the first and second moments m_t and v_t in the Adam optimizer by examining their expected values.

4.5.1 Derivation of Bias for the First Moment m_t

The first moment m_t is computed recursively as an exponentially weighted average of past gradients. Expanding m_t recursively, we get:

$$m_t = (1 - \beta_1)g_t + \beta_1(1 - \beta_1)g_{t-1} + \beta_1^2(1 - \beta_1)g_{t-2} + \dots$$

Taking the expectation, assuming g_t is an unbiased estimate of the true gradient g, we find:

$$\mathbb{E}[m_t] = g \cdot (1 - \beta_1) \sum_{i=0}^{t-1} \beta_1^i$$

This summation simplifies to:

$$\mathbb{E}[m_t] = g \cdot (1 - \beta_1) \frac{1 - \beta_1^t}{1 - \beta_1} = g \cdot (1 - \beta_1^t)$$

This shows that m_t is biased by the factor $1 - \beta_1^t$. To obtain an unbiased estimate, we divide by this factor, yielding the bias-corrected first moment:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

This correction ensures that $\hat{m}_t$ is an unbiased estimate of the gradient g, as $\mathbb{E}[\hat{m}_t] = g$.

4.5.2 Derivation of Bias for the Second Moment v_t

Similarly, the second moment v_t is computed as an exponentially weighted average of past squared gradients. Expanding v_t, we get:

$$v_t = (1 - \beta_2) \sum_{i=0}^{t-1} \beta_2^i g_{t-i}^2$$

Taking the expectation, assuming that g_t^2 is an unbiased estimate of the true squared gradient g^2, we find:

$$\mathbb{E}[v_t] = g^2 \cdot (1 - \beta_2) \sum_{i=0}^{t-1} \beta_2^i$$

Simplifying this, we obtain:

$$\mathbb{E}[v_t] = g^2 \cdot (1 - \beta_2^t)$$

This shows that v_t is biased by $1 - \beta_2^t$. To correct this, we divide by $1 - \beta_2^t$, yielding the bias-corrected second moment:

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

This correction ensures that $\hat{v}_t$ is an unbiased estimate of g^2, as $\mathbb{E}[\hat{v}_t] = g^2$.
By applying these bias corrections, the Adam optimizer obtains accurate estimates for both the gradient mean and variance, enhancing its performance, particularly in the early stages of training.

4.6 Weight Update

Finally, Adam updates **each weight based on the adjusted first and second moments:**

$$\theta_{t+1} = \theta_t - \frac{\alpha \, \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{6}$$

where:

- α is the learning rate,
- ϵ is a small constant to prevent division by zero (usually 10^{-8}).

This adaptive update means weights associated with large gradients get smaller updates, while weights with smaller gradients get larger updates.

4.7 Conclusion

This process of the forward pass, loss calculation, gradient calculation, and weight update using Adam repeats for multiple iterations as the model trains. Over time, the model's predictions become more accurate, as the weights converge towards values that minimize the loss.

Benefits of using Adam on non-convex optimization problems:

- Straightforward to implement.
- Computationally efficient.
- Little memory requirements.
- Invariant to diagonal rescaling of the gradients.
- Well suited for large problems in terms of data and parameters.
- Appropriate for problems with very noisy or sparse gradients.

5 Implementation

5.1 Downsampling

Algorithm 1 Downsampling with Strided Convolution

Require: Input feature map F of size $H \times W$, convolution kernel K of size $k \times k$, and stride $s > 1$
Ensure: Downsampled output feature map F' with reduced spatial dimensions

Initialize output feature map F' of size $\lfloor H/s \rfloor \times \lfloor W/s \rfloor$

for each output position (i, j) in F' **do**
  Set $F'_{i,j} \leftarrow 0$
  for each element (m, n) in kernel K **do**
    Compute input position $(x, y) \leftarrow (s \times i + m, s \times j + n)$
    if $x < H$ and $y < W$ **then**
      $F'_{i,j} \leftarrow F'_{i,j} + F_{x,y} \cdot K_{m,n}$
    end if
  end for
end for

Return: Downsampled feature map F'

5.2 Upsampling

Algorithm 2 Upsampling with Transposed Convolution

Require: Input feature map F' of size $H \times W$, convolution kernel K of size $k \times k$,and stride $s > 1$
Ensure: Upsampled output feature map $\hat{G}$ with expanded spatial dimensions

Initialize output feature map $\hat{G}$ of size $(H \times s) \times (W \times s)$ with zeros

for each input position (i, j) in F' **do**
  for each element (m, n) in kernel K **do**
    Compute output position $(x, y) \leftarrow (s \times i + m, s \times j + n)$
    if $x < H \times s$ and $y < W \times s$ **then**
      $\hat{G}_{x,y} \leftarrow \hat{G}_{x,y} + F'_{i,j} \cdot K_{m,n}$
    end if
  end for
end for

Return: Upsampled feature map $\hat{G}$

5.3 Adam Optimzation

Algorithm 3 Adam Optimization Algorithm

Require: Initial weights θ_0, learning rate α, decay rates β_1, β_2, small constant ϵ, number of iterations T

Ensure: Optimized weights θ_T

Initialize first moment vector $m_0 = 0$, second moment vector $v_0 = 0$, and time step $t = 0$

for $t = 1$ to T **do**

Compute gradient $\frac{\partial L}{\partial \theta_t}$ ▷ Derivative of L with respect to current weights θ_t

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot \frac{\partial L}{\partial \theta_t}$ ▷ Update first moment estimate (Momentum)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot \left(\frac{\partial L}{\partial \theta_t}\right)^2$ ▷ Update second moment estimate (RMSprop)

$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$ ▷ Correct bias in first moment

$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$ ▷ Correct bias in second moment

$\theta_{t+1} \leftarrow \theta_t - \frac{\alpha \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ ▷ Update weights

end for

Return: Optimized weights θ_T

5.4 Code Implementation

We developed a convolutional neural network (CNN) model in Python using TensorFlow, Keras, NumPy, and OpenCV. Trained on a large dataset, the model effectively converts grayscale images to color. TensorFlow and Keras power the model training, while NumPy and OpenCV support data handling and image processing. This project highlights automatic image colorization using CNNs and efficient Python libraries.

Here is the link to the project code

https://www.kaggle.com/code/swayamhingu/group-5-optimization-grayscale-to-color-image?scriptVersionId=205750044

5.5 Flow Chart

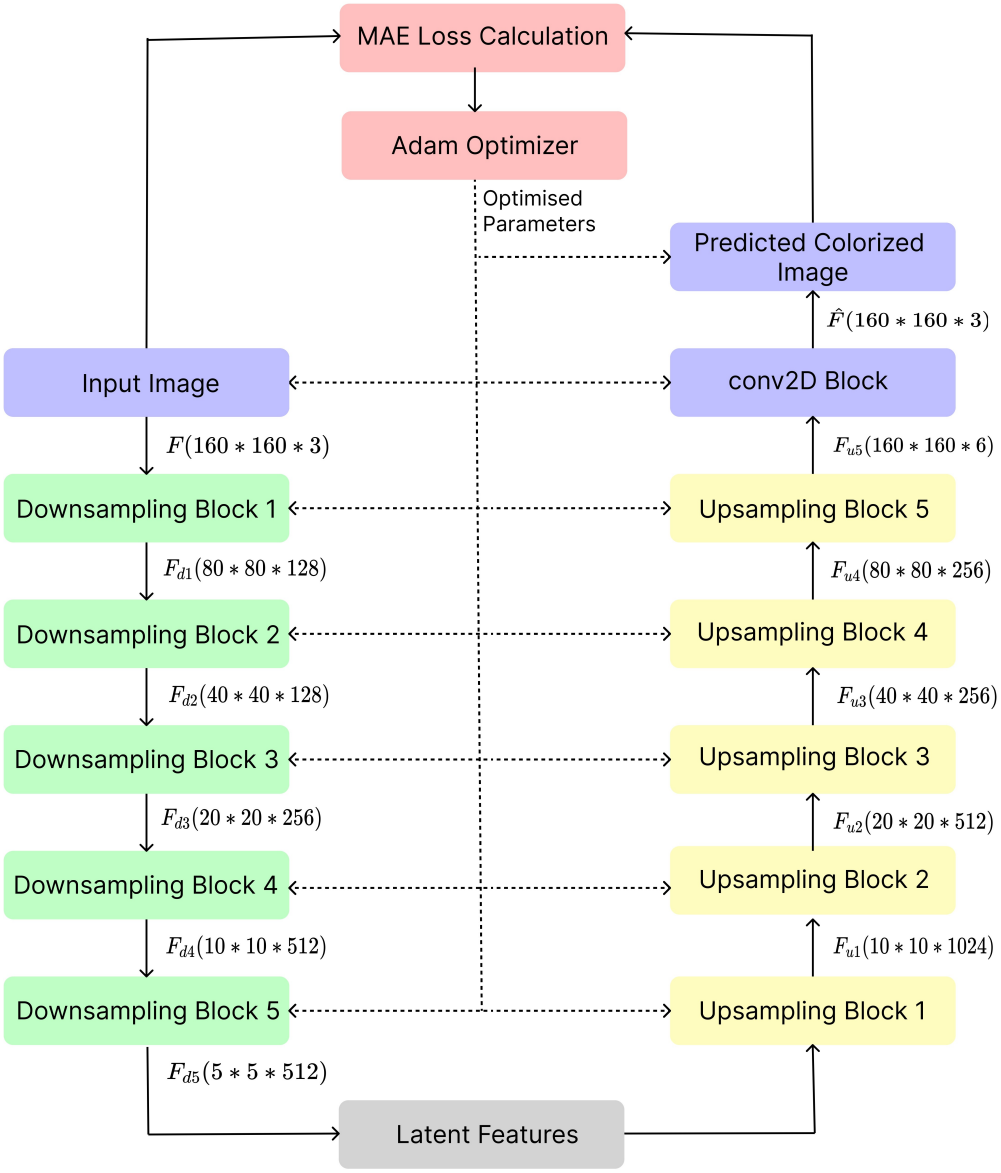

Figure 6: Flow Chart of Each Iteration

6 Results

Color Image

Grayscale Image

Predicted Image

Color Image

Grayscale Image

Predicted Image

Color Image

Grayscale Image

Predicted Image

Color Image

Grayscale Image

Predicted Image

Color Image

Grayscale Image

Predicted Image

Accuracy of Model - 48.74%
Loss - 5%

7 References

7.1 Introduction

- ANITS, "Types Of Images and Color Spaces" click here
- YouTube, "Simple Explanation of Autoencoders," click here

7.2 Mathematical Theory

- GeeksforGeeks, "Principal Component Analysis" click here
- GeeksforGeeks, "PCA vs Autoencoder" click here
- Linkedin, "Autoencoder" click here

7.3 Adam Optimization

- GeeksforGeeks, "Adam Optimizer - GeeksforGeeks," click here
- Kingma, D. P., & Ba, J. (2014). "Adam: A Method for Stochastic Optimization." *International Conference on Learning Representations (ICLR)*. click here
- Medium, "Derivation Of Bias Correction Factors," click here