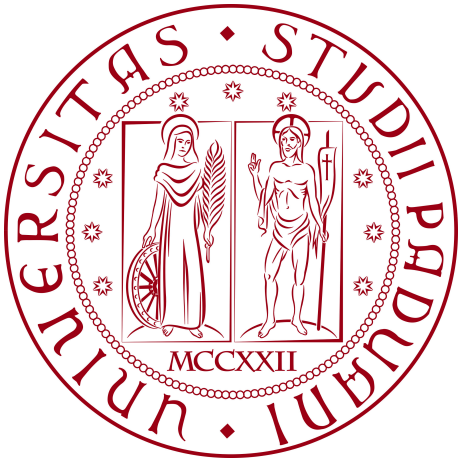
University of Padua

Department of Mathematics "Tullio Levi-Civita"

Bachelor's Degree in Computer Science



Artificial Colorist: A Fine-Tuning Pipeline for Enhancing Color Recognition in Vision-Language Models

Bachelor's Thesis

Supervisor
Prof. Lamberto Ballan
Co-supervisor
Dr. Elena Izzo

Undergraduate
Lorenzo Pasqualotto
Student ID: 2008651

Academic Year 2023-2024

The English translation of this document was assisted by ChatGPT.

© Lorenzo Pasqualotto, November 2024. All rights reserved. Bachelor's Thesis: "*Artificial Colorist: A Fine-Tuning Pipeline for Enhancing Color Recognition in Vision-Language Models*", University of Padua, Department of Mathematics "Tullio Levi-Civita".

Acknowledgements

Alla mia famiglia,
ai miei amici.

Padua, November 2024 *Lorenzo Pasqualotto*

This document describes the work carried out during the internal internship, lasting 300 hours, by the undergraduate student Lorenzo Pasqualotto. The work was carried out under the supervision of Professor Lamberto Ballan, as the proposer, and Professor Nicolò Navarin, internal tutor for the degree program, with the directive contribution of Dr. Elena Izzo, who provided the guidelines for the project's development.

Abstract

The recent development and growing adoption of vision-language models (VLMs) have led to significant advancements in the field of computer vision. Numerous state-of-the-art models achieve human-comparable performance on traditional datasets; however, they still exhibit limited sensitivity to specific image attributes, such as spatial relationships between objects and chromatic characteristics, occasionally making gross errors on tasks that are intuitive for humans. This work proposes a pipeline to enhance the capability of recognizing chromatic attributes in one of the most renowned and widely used models in computer vision: CLIP (Contrastive Language–Image Pre-training). The proposed methodology involves generating a synthetic dataset composed of chromatic variants of segmented objects, derived from the images and annotations of the MSCOCO dataset. The adopted fine-tuning algorithm is based on a contrastive learning approach.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context

In recent years, research in the field of computer vision has achieved remarkable performance advancements thanks to the creation of datasets that are qualitatively and quantitatively superior to those of the past. State-of-the-art examples include: MSCOCO (Microsoft, 2014), ImageNet (Stanford University, 2009), Flickr30k (University of Illinois Urbana-Champaign, 2015), and MNIST (Yann LeCun, 1998). Parallel to this evolution in data, visual-language models (VLMs) have been developed, capable of connecting textual and visual information through the combined use of computer vision algorithms and Large Language Models (LLMs). These models produce a shared embedding space with results significantly superior to previous algorithms. State-of-the-art models include: CLIP (OpenAI, 2021), VilBert (Facebook, 2019), Blip (Salesforce, 2022), and VisualGPT (Microsoft, 2021).

These models achieve human-level performance in traditional evaluation benchmarks containing a wide variety of compositional richness and visual content diversity. However, when tested on specific benchmarks designed to evaluate a particular attribute (e.g., color, spatial relationship...) for specific tasks, performance drops drastically, despite the examples being easily comprehensible by humans [22] [21].

The CLIP model, which is the focus of this project, leverages a shared embedding space for textual and visual data, providing a metric to evaluate the similarity between an image and a textual description: the CLIP-score [6].

Figure 1.1: Example of incorrect CLIP-score assignment

By defining specific tests, the use of this metric can help analyze how well the model understands specific aspects of images. Let us evaluate its ability to recognize the color of an object by calculating the CLIP-score between an image and its description containing chromatic information, comparing it with the results obtained when the original image is paired with altered descriptions where the color attribute is modified. We observe that the model tends not to assign the highest CLIP-score to the correct image-text pair, as illustrated in 1.1, where the provided example shows that the highest score (highlighted in green) surpasses the score of the correct description (highlighted in red), and 1.2, which presents some model performance on images from MSCOCO.

METRICA	DESCRIZIONE	VALORE
Accuratezza	Quante volte il modello assegna il CLIP-Score più elevato alla descrizione contenente la variante cromatica corretta	43.6 %
Differenza Media	Media cumulativa delle medie delle differenze tra CLIP-Score	0.0176
Deviazione Standard Media della Differenza	Media cumulativa delle deviazioni standard calcolate sulle differenze tra CLIP-Score	0.0110

Figure 1.2: Current performance of CLIP-Vit-Base-32 on 742 images from MSCOCO

The mean difference is calculated as the cumulative average, over all considered examples, of the differences between the CLIP-scores of the correct pair and those of the incorrect pairs for each individual example. The standard deviation, on the other hand, is determined as the cumulative average of the standard deviations of the differences between the CLIP-score of the correct pair and the scores of the incorrect pairs.

The observed values for these metrics, particularly low, highlight how the color attribute is not adequately considered as a discriminative feature by the current state of the model. Indeed, a variation of this attribute in the description does not lead to a substantial difference in the assigned CLIP-score, resulting in excessively low accuracy in recognizing the correct color.
Based on these observations, the following section will define the approach chosen to refine the model's performance in recognizing chromatic attributes, aiming for improvement over the previously mentioned metrics.

1.2 Identified Pipeline

The identified pipeline involves generating a synthetic dataset of chromatic variants from any dataset containing **images**, **textual descriptions**, and **segmentation masks**. The images and synthetic descriptions generated through a specific algorithm are then used during training for fine-tuning the model, employing a **contrastive approach**.

The following subsections summarize the completed steps, providing an explanatory overview, with various points elaborated upon in subsequent chapters of the document.

1.2.1 Generating a Synthetic Dataset Through Pre-Processing Techniques

To adequately fine-tune training for this specific task, a synthetic dataset of chromatic variants was created using pre-processing techniques based on the MSCOCO dataset. As summarized in 1.3, for each MSCOCO image whose caption contains at least one color attribute referring to an object, the color and associated entity are extracted, proceeding to create N (number of studied colors) alternative descriptions where the color has been modified. Each caption is then paired with a synthetic image, derived from the original image, where the segmentation mask of the referenced object is used to produce an appropriate chromatic variant through a recoloring algorithm, details of which are presented in the dedicated chapter. As analyzed in the aforementioned chapter, the segmentation masks used do not correspond to those available in the MSCOCO dataset; instead, a combination of two models, Grounding Dino and SAM, is employed to extract precise masks better suited for recoloring.

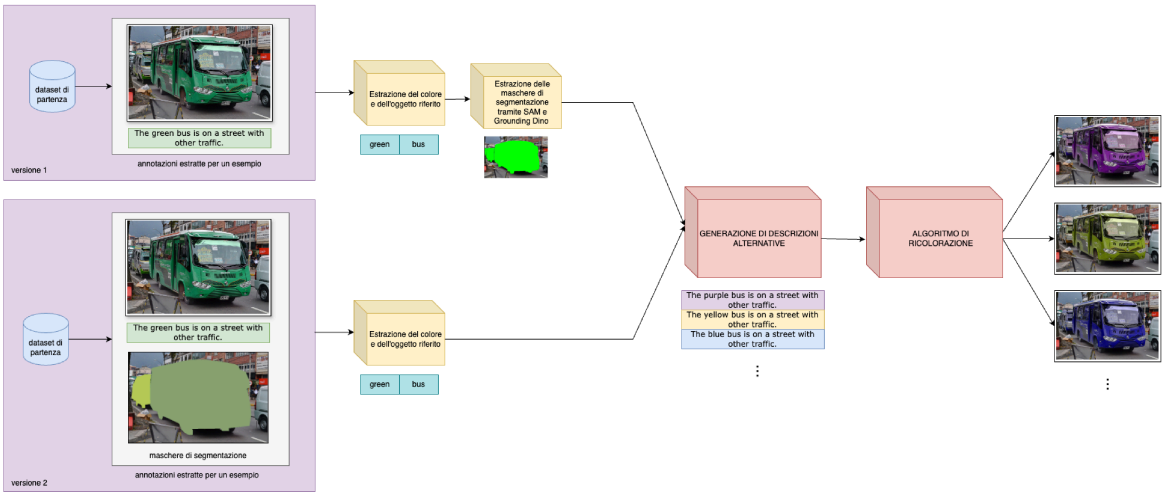


Figure 1.3: Summary of the pipeline for creating the synthetic dataset of chromatic variants

1.2.2 Fine-tuning

The fine-tuning algorithm utilizes the synthetic dataset, applying a supervised training technique called *Contrastive Learning* [8]. This method optimizes the model by training it to distinguish between similar and

dissimilar example pairs. In this case, in the model's embedding space, image-description pairs where the description references a color different from that of the object in the image are separated, while pairs where the described color matches the one in the visual data are brought closer together. The mathematical techniques used are briefly introduced below and elaborated upon in the relevant chapter.

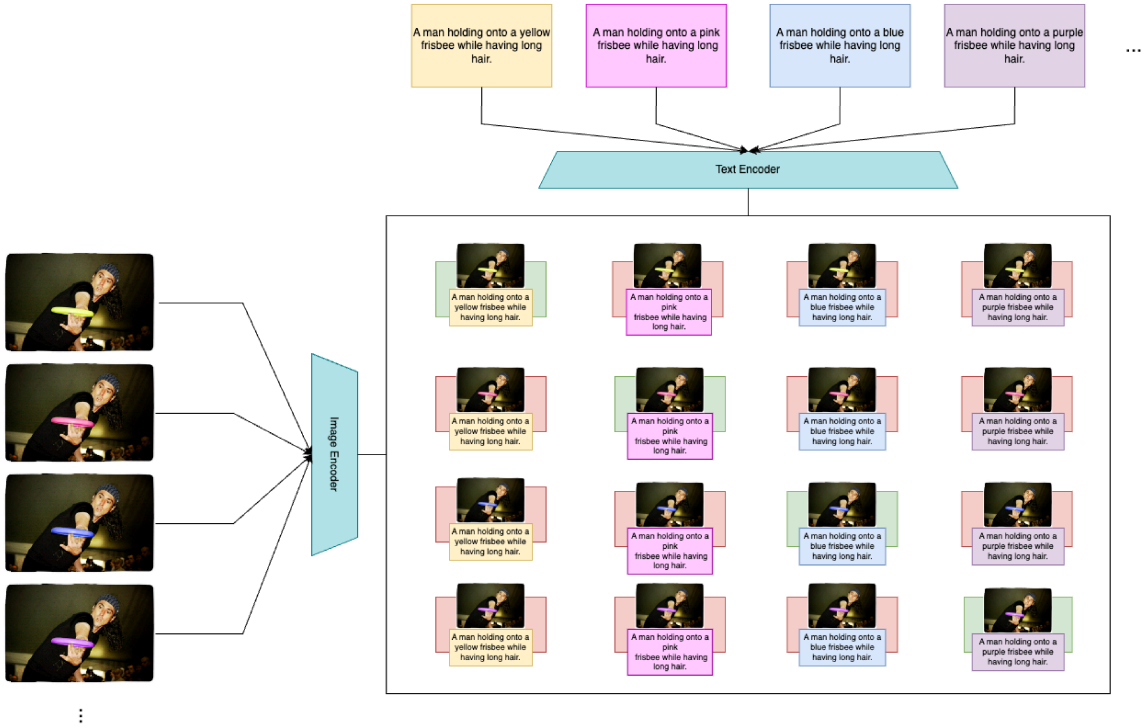


Figure 1.4: Visualization of the contrastive approach used to bring image-description pairs closer or farther apart. The diagonal elements, highlighted in green, are those whose distance is to be minimized.

In 1.4, it is shown how, starting from the images and descriptions produced by the synthetic dataset generation algorithm, a matrix of image-text pairs can be constructed.
Each piece of textual and visual data is initially processed through its respective encoder, which converts the information into a shared embedding space where operations to bring closer and separate can be performed.
The similarity of embeddings for pairs where the visual chromatic attribute corresponds to the textual one, represented on the diagonal of the matrix in the figure, is thus reinforced, while that of the remaining pairs is weakened.

1.3 Document Organization

After a brief description of the technologies used, the text will address the studied techniques for recoloring objects using pre-processing techniques, which generate a synthetic dataset, whose details will be analyzed in a subsequent chapter. Following this, I will present the details of the fine-tuning algorithm, exploring the theoretical foundations of this technique and concluding with an analysis of the model's performance using the identified pipeline.

The project code is available here.

Chapter 2

Technologies

The following technologies were adopted for the development of the project:

2.1 Libraries

Conda 24.9.2

Conda is an open-source environment and package management library, designed to easily manage development environments with multiple and complex dependencies. It is particularly valued for its ability to create isolated virtual environments, each with specific versions of Python and the required packages, ensuring efficient and reproducible dependency management.
In this project, Conda was employed to enable GPU usage during the model fine-tuning process, making computational processes significantly faster compared to execution on a CPU. This allowed for an optimized and parallelized fine-tuning process, reducing training time and increasing the overall efficiency of the workflow.

OpenCV 4.10.0

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning library developed to facilitate image and video analysis and manipulation. Originally developed by Intel and later maintained by the OpenCV community, it is now a standard for image processing and visual analysis, widely used in both academia and industry.

In this project, it is used for loading and manipulating images.

PyTorch 2.5.1

PyTorch is an open-source machine learning library developed by Facebook AI Research (FAIR) for numerical computation and building deep learning models. It is based on Python and also supports C++ for high-performance scenarios, making it ideal for developing complex models and optimizing neural networks. PyTorch offers flexibility and ease of use thanks to its modular structure and its ability to perform computations on GPUs,

optimizing runtime and facilitating the handling of large datasets and large-scale models.

In the project, it is used for fine-tuning, leveraging its functionalities for managing datasets and the algorithms necessary for model training.

LoRA (Peft 0.13.2)

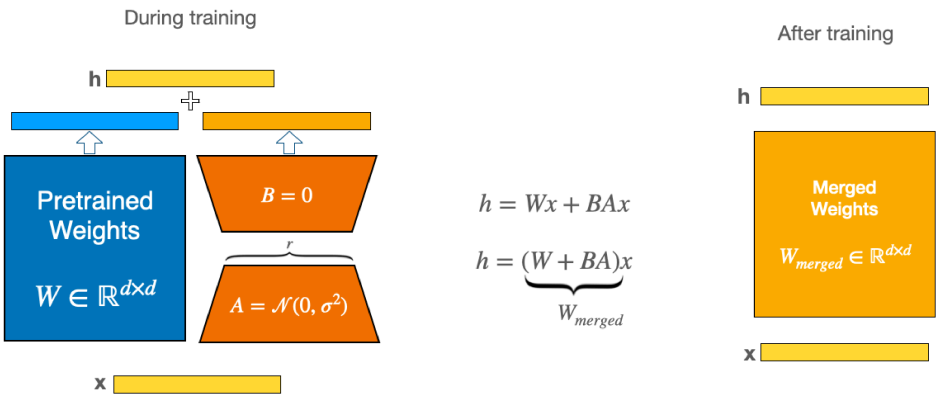


Figure 2.1: Functionality of LoRA

LoRA (Low-Rank Adaptation) is an optimized training technique for deep learning models, designed to reduce computational costs and memory requirements during the training of large models. It is based on an innovative approach that reduces the degrees of freedom of the model's parameters through low-rank matrices, allowing only a small subset of parameters to be adapted, thus making optimization more efficient [12]. LoRA is particularly advantageous for fine-tuning linguistic and visual models, enabling training on new, specific data without entirely modifying the structure of the original model.

For the mentioned advantages, LoRA is used during the training phase of the project.

SpaCy 3.8.2

SpaCy is an advanced open-source library for Natural Language Processing (NLP), designed to offer an optimized and scalable architecture for linguistic analysis.

In the project, it is used for extracting colors and associated entities from captions in the dataset, as described in detail in the relevant section.

2.2 Models

CLIP Vit-Base-Patch32

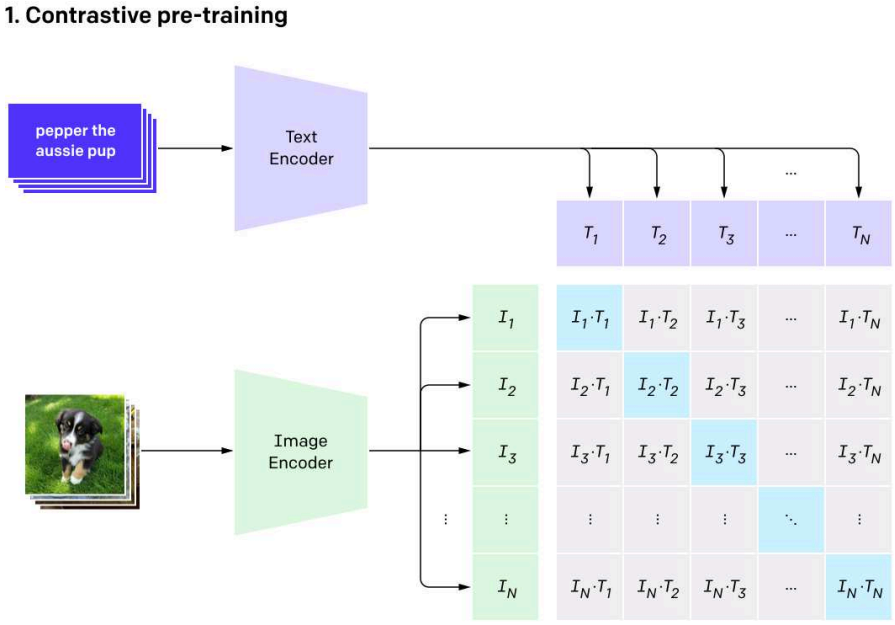


Figure 2.2: Overview of CLIP training

CLIP (Contrastive Language-Image Pretraining) is a vision-language model developed by OpenAI to bridge the gap between computer vision and natural language understanding. Designed to interpret and connect images and text, CLIP learns to recognize visual concepts through direct associations between linguistic descriptions and visual representations. Thanks to the contrastive approach, visualized in 2.5, CLIP can respond to textual queries by identifying relevant images or semantically describe visual content, demonstrating exceptional generalization capabilities across a wide range of visual and linguistic tasks. There are various versions of the model:

- ViT-B/32: based on the Vision Transformer (ViT) with a patch size of 32x32 pixels.
- ViT-B/16: similar to the previous, but with a reduced patch size of 16x16 pixels.
- ViT-L/14: a larger version of the ViT model with a patch size of 14x14 pixels.
- ViT-L/14@336px: a variant of ViT-L/14 trained on images with a resolution of 336x336 pixels, further improving performance on high-resolution images.

To balance computational costs, which are high for ViT-L models, and performance, the ViT-B/32 version was used in this project.

Grounding Dino 1.0

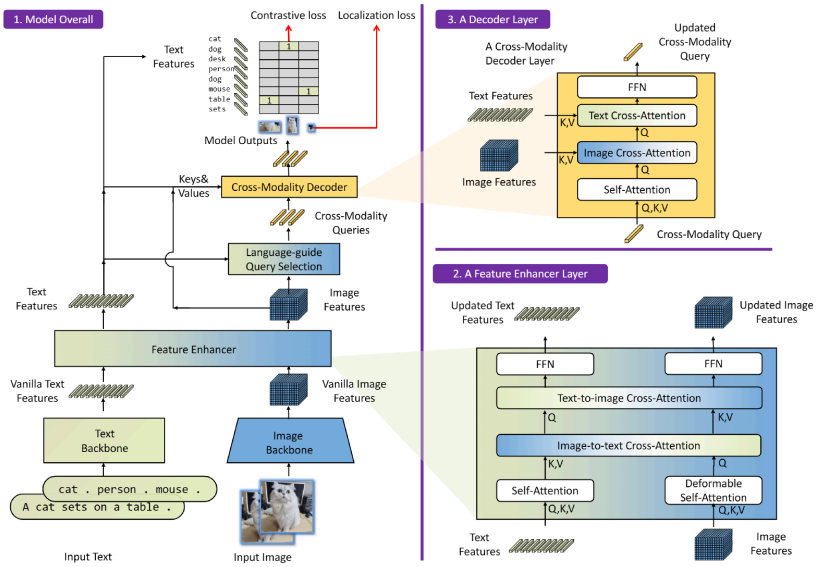


Figure 2.3: Overview of Grounding-Dino functionality

Grounding DINO is an advanced model for computer vision and natural language understanding, specifically designed for visual grounding, i.e., identifying objects in images based on textual descriptions. It is an end-to-end system capable of associating text with specific portions of an image, accurately recognizing the visual entities described.

SAM Vit-h 4b8939

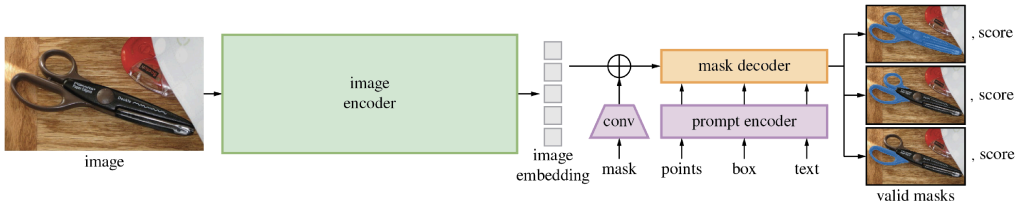


Figure 2.4: Overview of SAM functionality

SAM, or Segment Anything Model, is an advanced model for image segmentation, designed to identify and segment objects or specific regions within an image efficiently and flexibly. Created by Meta AI, SAM is distinguished by its ability to generalize across a wide range of scenes and apply to different contexts without requiring specific customizations. It is structured to be used with any type of image, regardless of the scene or domain, and has been trained on a large dataset to improve its ability to identify objects of different shapes and sizes, even in complex scenarios.

In the project, SAM is used alongside Grounding Dino to extract segmentation masks of objects to be recolored.

2.3 Dataset

Flickr 8k

The Flickr 8k dataset is a dataset created for research in computer vision and natural language processing. It was developed by the research group at the University of Illinois. The dataset includes 8,000 images collected from Flickr, a photo-sharing platform, and each image is accompanied by five textual descriptions (captions) created by human annotators.

The dataset images cover a wide range of everyday scenes and depict people, objects, and natural environments. The descriptions were designed to be concise and semantic, aiming to accurately capture the visual content of the images.

A distinctive aspect of the dataset is the annotation evaluation system, ensuring quality. The descriptions provided by human annotators were evaluated by expert evaluators based on their coherence and relevance to the visual content of the images. The evaluations include three independent judgments, with scores ranging from 1 to 4, based on the following criteria:

- 1: The caption does not describe the image at all.
- 2: The caption describes only marginal aspects of the image but does not provide a complete description.
- 3: The caption describes the image almost completely, but with minor errors or omissions.
- 4: The caption describes the image accurately and completely.

These expert judgments ensure that the dataset is annotated with a high level of precision, making it ideal for evaluating machine learning algorithms.

MSCOCO

Figure 2.5: Examples of MSCOCO data

The MSCOCO (Microsoft Common Objects in Context) dataset [11], developed in 2014, was created to address complex computer vision problems by providing a wide variety of annotated data. With its 330,000 images of realistic scenes and over 1.5 million object instances, MSCOCO represents one of the most detailed datasets for object recognition and segmentation in realistic contexts. This dataset provides precise annotations for 80 categories of common objects, enabling in-depth analysis of tasks such as *Object Detection*, *Instance Segmentation*, *Semantic Segmentation*, *Panoptic Segmentation*, *Keypoint Detection*, and *Image Captioning*.
Specifically, MSCOCO offers:

- **Segmentation Annotations**: each object is represented with polygon masks, facilitating the training of neural networks for object detection with defined contours.
- **Human Pose Data**: keypoint annotations on images containing people, particularly useful for *Human Pose Estimation*.
- **Image Captions**: each annotated image includes five textual descriptions created by human annotators, essential for *Image Captioning* models and for studying the relationship between vision and language.

The completeness and variety of annotations have made MSCOCO a benchmark for training and evaluating state-of-the-art models in numerous studies. It has also significantly contributed to improving the performance of *Object Detection*, *Instance Segmentation*, and *Image Captioning* models, making it a standard for evaluating new computer vision algorithms.

Chapter 3

Object Recoloring Using Pre-Processing Techniques

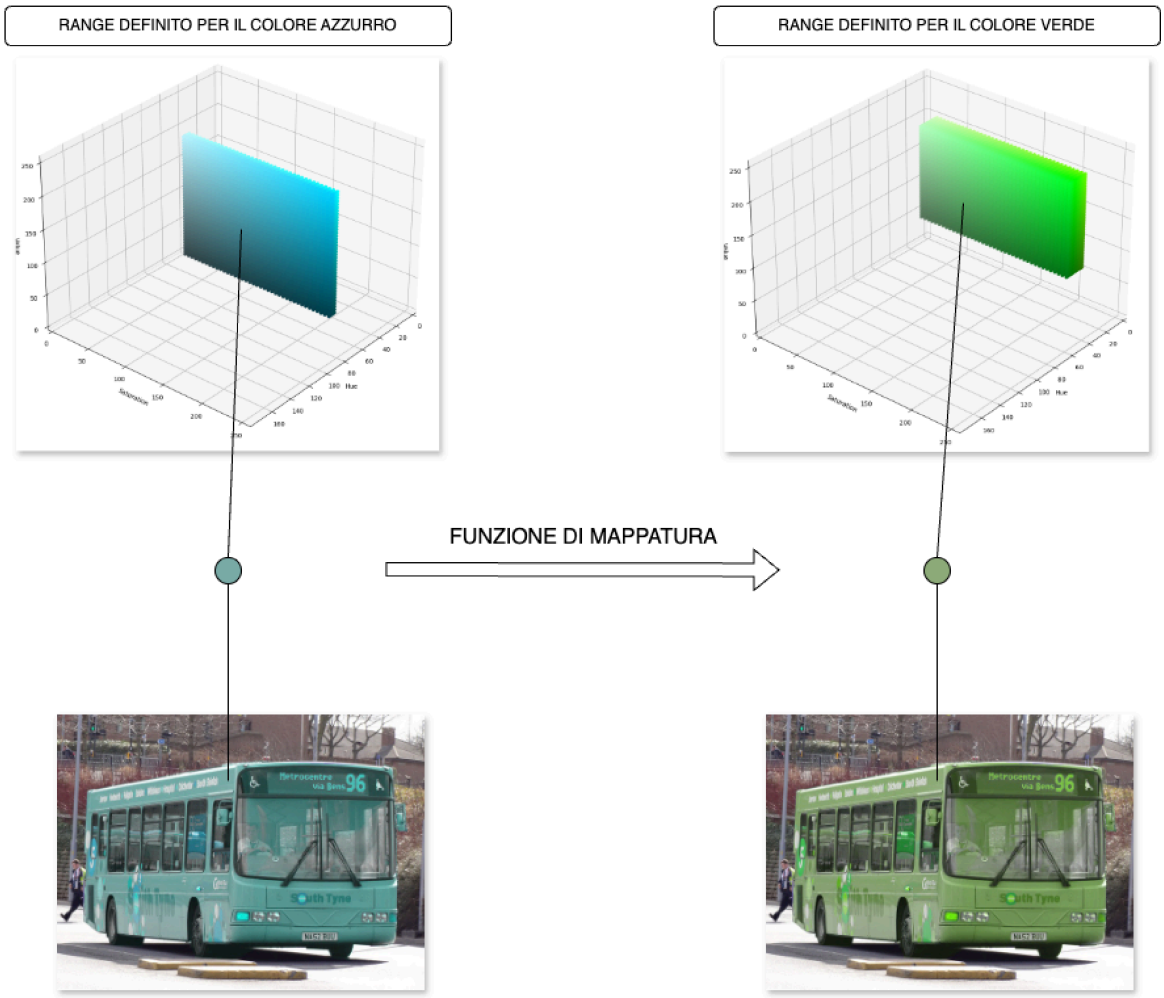


Figure 3.1: Visualization of the recoloring mechanism, focusing on the transformation of a single pixel in the HSV colorspace.

Recoloring objects is a popular task in image editing. The literature includes numerous approaches to tackle this challenge, some based on color distribution in the image [5] [1], others on machine learning techniques [9], and yet others on complex mathematical-optical methods [4].
While these solutions provide state-of-the-art results, they are unsuitable for this project's requirements due to high computational costs, lack of direct manipulation control over recoloring, or absence of the necessary code. Hence, I propose an alternative, simple, and effective solution using pre-processing techniques. This method, as illustrated in 3.1, is based on the **definition of specific ranges** within a color space for a studied set of colors and on a **mapping function**.
In the following sections, I will provide details on the analyzed color spaces and the actual recoloring function, illustrating the rationale behind the choices made and the challenges encountered during development.

3.1 Introduction to Color Theory

Before proceeding further, I provide a brief focus on *color theory*, necessary to better understand the mechanisms underlying the manipulation of this attribute for digital images.

Color theory is a field that studies how colors are perceived, combined, and represented, rooted in both physics and human visual perception. The three main attributes that describe a color are **hue**, **saturation**, and **light intensity** (or brightness).

Hue (or Chromaticity)

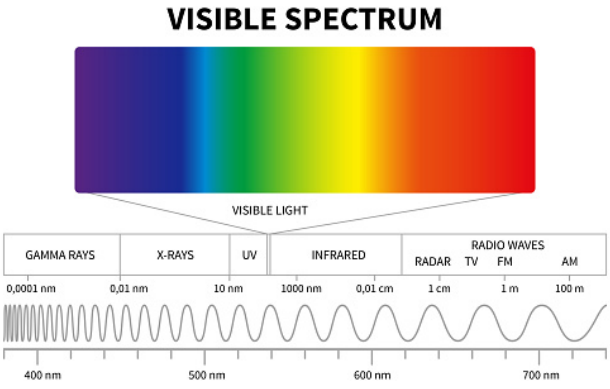


Figure 3.2: Visualization of the visible color spectrum.

Hue refers to the perceptual quality that distinguishes colors, such as red, green, and blue. It is determined by the predominant wavelength of light reflected or emitted by an object. Each wavelength corresponds to a specific color in the visible spectrum: red has longer wavelengths, while blue and violet have shorter wavelengths (3.2).
In real-world contexts, the color in a scene arises from complex interactions of reflection, refraction, and absorption of light by objects [13] [10]. When light strikes a surface, it alters the object's color based on the angle

and quality of reflected light, without significantly affecting intensity and saturation. Changing a color's hue simulates this physical effect by replacing the dominant wavelength of the color without altering other aspects. Practically, this changes the perceived "shade," while the surface maintains its reflective and textural properties, making the manipulation visually realistic.

Saturation

Figure 3.3: Visualization of Saturation.

Saturation indicates the intensity of the color, representing the distance from gray. A fully saturated color is vivid and intense, while a desaturated one leans toward gray. Physically, saturation depends on the complexity of the reflected light spectrum: monochromatic light (a single wavelength) appears extremely saturated, while a combination of wavelengths (like white light mixed with a tint) reduces saturation.

Light Intensity (or Brightness)

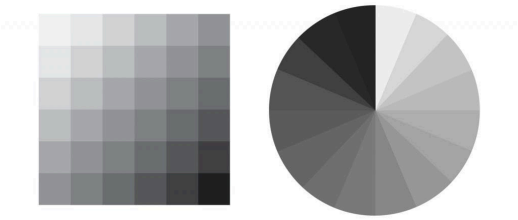


Figure 3.4: Visualization of Brightness.

Light intensity is the perceived amount of light emitted or reflected by an object, referring to how light or dark the color appears. Brightness depends on the total amount of luminous energy and directly influences how "bright" or "dark" a shade appears, regardless of saturation and hue. Physically, it is measured in terms of luminous flux or intensity and can be altered by the amount of light a surface reflects.
Manipulating brightness in images, by increasing or decreasing luminosity, can simulate changes in light exposure. When digitally altering the color of an image, the context and original lighting must be considered to avoid producing unnatural effects, as the manipulated color might not align with shadows, reflections, or other environmental properties [13]. Manipulating light intensity can create a flat effect, losing exposure matching with the rest of the image.

3.2 Color Spaces

A color space consists of a **color model** combined with an appropriate **mapping function**, enabling the model's effective use. A color model is an abstract mathematical system describing how colors are represented using numerical combinations, typically expressed through three or four values called **color components**. Since the color model is a theoretical representation, it is complemented by specific rules tailored to its use context, transforming it into a color space. This allows colors to be represented as coordinates in a space. Each color space has advantages and disadvantages for image processing, influencing the outcome of computer vision operations [3]. In this section, I delve into three widely used color spaces: **RGB**, **CIELAB**, and **HSV**.

3.2.1 Types of Color Spaces

Color spaces can be categorized based on their representation of light intensity and their alignment with human perception. The main distinctions, relevant to the desired recoloring task, are between **linear** and **non-linear** spaces and between **perceptually uniform** and **perceptually non-uniform** spaces.

Linear Color Spaces

Linear color spaces represent colors such that the overall intensity is proportional to the sum of chromatic components. An example is the RGB (Red, Green, Blue) space, widely used in digital devices. Linear spaces are ideal for simple mathematical calculations, as they do not require complex conversions, but they are not suitable for handling complex manipulations due to their misalignment with human perception.

Non-Linear Color Spaces

Non-linear color spaces, such as sRGB and HSV, apply non-linear compression of intensity levels to represent colors in a way that better reflects human visual perception, enhancing visual quality for the observer. However, this representation may require conversion to linear spaces for advanced numerical operations.

Perceptually Uniform Color Spaces

Perceptually uniform spaces are designed so that Euclidean distances between colors correspond to differences perceived by the human eye. An example is the CIELAB space, widely used for assessing visual quality and color difference. These spaces accurately represent human perception of color variations but require complex transformations, making them computationally expensive.

Perceptually Non-Uniform Color Spaces

Perceptually non-uniform spaces, such as HSV and HSL, offer a color representation that separates components like hue, saturation, and value, making chromatic manipulation more intuitive for graphic applications and color selection. However, as they are not uniform, the distances between colors do not accurately reflect human perception, and variations in saturation or brightness may not correspond to proportional changes in perceived color.

3.2.2 Analysis of RGB, CIELAB, and HSV

I now analyze three specific color spaces: RGB, CIELAB, and HSV, representing each of the above categories.

RGB

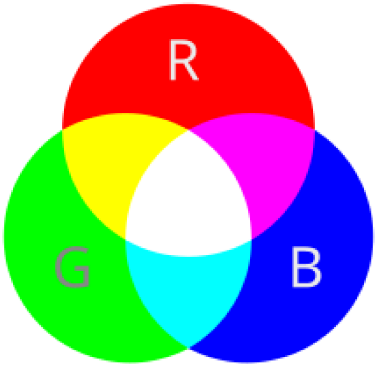


Figure 3.5: Visualization of the RGB Color Space.

RGB is a linear, additive color space where colors are represented as combinations of intensities of three primary colors: red, green, and blue. As a linear color space, it is widely used in electronic devices such as monitors, scanners, and cameras.

RGB is simple and intuitive: each color is represented by a linear combination of its three components, making image manipulation operations more straightforward.

A drawback of the RGB space is its inability to reflect human color perception, as the human eye perceives chromatic variations non-linearly. Consequently, RGB is not ideal for applications requiring visual correspondence.

CIELAB

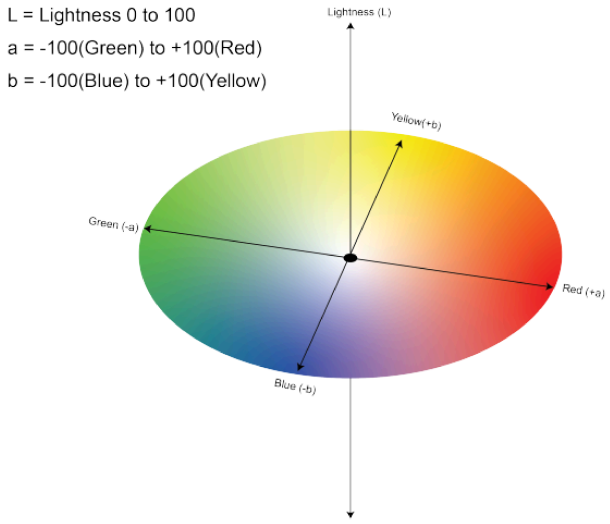


Figure 3.6: Visualization of the CIELAB Color Space.

The CIELAB model, or LAB, is a perceptually uniform color space that represents colors using three components: L^* (lightness), a^* (green-red), and b^* (blue-yellow). It is designed to ensure that Euclidean distances between colors reflect perceived differences.

CIELAB provides an accurate representation of visual perception of color, making it suitable for color correction, visual quality assessment, and printing. Moreover, being device-independent, it ensures greater color consistency across monitors and printers.

Using CIELAB, however, requires complex transformations, which can increase computation time and demand higher processing power.

HSV

HSV (Hue, Saturation, Value) is a non-linear and perceptually non-uniform color space representing colors using three components: hue, saturation, and value. The *hue* channel is designed to intuitively represent color tones, facilitating identification and selection of specific hues. The *saturation* and *value* components, however, do not represent human perception with the same accuracy, as variations in these components may not correspond proportionally to perceived changes.

HSV simplifies color manipulation by allowing the isolation and individual modification of its three main components, as studied earlier: hue, saturation, and brightness.

A limitation of HSV is that it is not fully perceptually uniform, so distances between colors do not accurately reflect perceived differences.

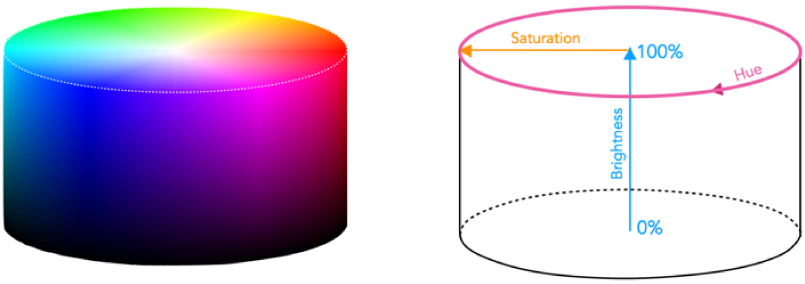


Figure 3.7: Visualization of the HSV Color Space.

Conclusions

After analyzing their general characteristics, it is essential to select a color space based on the specific requirements of our project. The key factors for discrimination are:

- The simplicity of **range definition**, i.e., the boundaries separating one color from another.
- The ease and effectiveness of implementing the **recoloring function**.

In the RGB space, color separation is simple. However, its linear structure, misaligned with human perception, hinders intuitive implementation of the desired function, leading to its exclusion.
Conversely, the CIELAB space appears most suitable for this type of manipulation. Unfortunately, color separation is complex, and major Python libraries provide limited support, increasing data processing difficulty.
The HSV space's structure, where chromatic hue is separated from other brightness-related information, makes it well-suited for this task. Manipulations and boundary definitions are much simpler and more intuitive than in the other analyzed spaces. For these reasons, **HSV is the color space used in this project**.

3.2.3 Range Definition

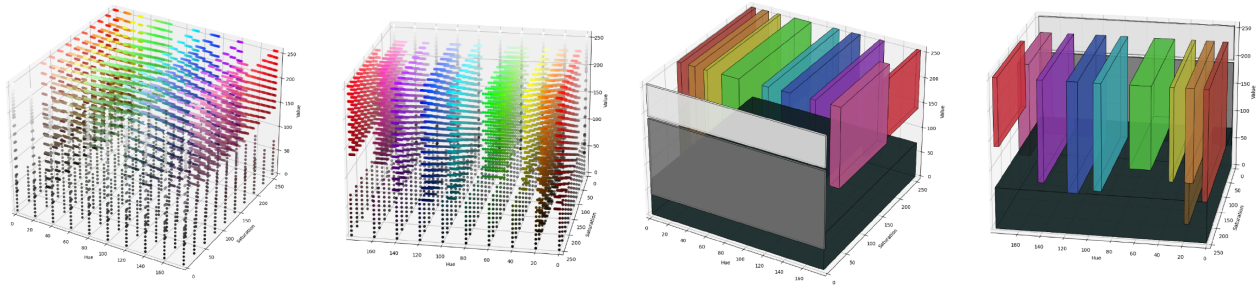


Figure 3.8: Visualization of defined ranges for colors: red, green, blue, cyan, yellow, pink, purple, orange, brown, white, gray, black.

The first concrete step in the project's development was defining boundaries within the chosen color space (HSV) to determine which values represent which colors.
The boundaries were defined as follows:

```
color_limits = {
    'red': [np.array([0,70,70]), np.array([4.4,255,255])],
    'red2': [np.array([175,110,140]), np.array([179,255,255])],
    'green': [np.array([47,70,117.3]), np.array([65,255,255])],
    'blue': [np.array([108,70,70]), np.array([117,255,255])],
    'cyan': [np.array([88.5,70,76]), np.array([95,255,255])],
    'orange': [np.array([12,70,100]), np.array([19,255,255])],
    'purple': [np.array([137,70,84]), np.array([142,255,255])],
    'pink': [np.array([157,70,100]), np.array([167,200,255])],
    'brown': [np.array([15,40,30]), np.array([19,255,190])],
    'yellow': [np.array([28.5,70,100]), np.array([32,255,255])],
    'black': [np.array([0,0,0]), np.array([179,255,70])],
    'white': [np.array([0,0,195]), np.array([179,3,255])],
    'gray': [np.array([0,0,40]), np.array([179,3,190])]
}
```

The defined values, visualized in 3.8, correspond to the library used for image manipulation: OpenCV. The more these values were refined during development, the greater the observed improvements in the recoloring algorithm's performance, which is presented below.

3.3 Recoloring Function

The recoloring function is based on a technique called *Linear Scaling*, defined and analyzed below.

3.3.1 Linear Scaling for Range Conversion

Linear scaling is a method commonly used to map a value x from an original range $[a, b]$ to a destination range $[c, d]$. The transformation formula assumes that the relative variation of the value in the first range is proportional to the relative variation of the value in the second range.
To map a value $x \in [a, b]$ to a corresponding value $x' \in [c, d]$, we use the following *linear scaling* formula:

$$x' = c + (x - a) \cdot \frac{d - c}{b - a}$$

Where:

- x is the initial value in the range $[a, b]$.
- a and b represent the minimum and maximum values of the source range, respectively.
- c and d represent the minimum and maximum values of the target range, respectively.
- x' is the scaled value, i.e., the result in the range $[c, d]$.

The formula's logic can be explained as follows: 1. Subtract a from x to determine the relative position of x within the source range. 2. Multiply this difference by the ratio $\frac{d-c}{b-a}$, representing the scaling factor between the range widths. 3. Finally, add c to translate the result into the target range.

Application Example

Suppose we want to transform a value $x = 5$, originally in the range $[0, 10]$, to a new range $[20, 30]$. Applying the formula:

$$x' = 20 + (5 - 0) \cdot \frac{30 - 20}{10 - 0} = 20 + 5 \cdot \frac{10}{10} = 20 + 5 = 25$$

Thus, the value 5 in the range $[0, 10]$ maps to 25 in the range $[20, 30]$.

Properties of Linear Scaling

Linear scaling preserves the proportions of the original values, ensuring that every point within the source range maintains its relative position in the target range. This property is particularly useful for the pre-processing steps necessary for our project.

3.3.2 Performance Analysis and Challenges

The following subsections detail the results and challenges encountered while applying linear scaling to the various color channels.

3.3.2.1 Differences in Channel-Based Approaches

Hue

Linear scaling produces realistic results when applied to the hue channel. Colors differing primarily in hue—such as red, orange, green, yellow, cyan, blue, pink, and purple—are easily manipulated. Starting from and targeting any of these colors yields satisfactory chromatic variations (3.9).

Two critical attributes are the range width and the relative distance, which respectively preserve luminous information and ensure perceived color differentiation. Colors like orange-red, pink-purple, and cyan-blue are very close in the hue channel. Maximizing the distance between defined boundaries is therefore necessary.

The difference between orange and brown, however, is not expressed in terms of hue but of light intensity. The resulting challenges are analyzed in subsequent sections.

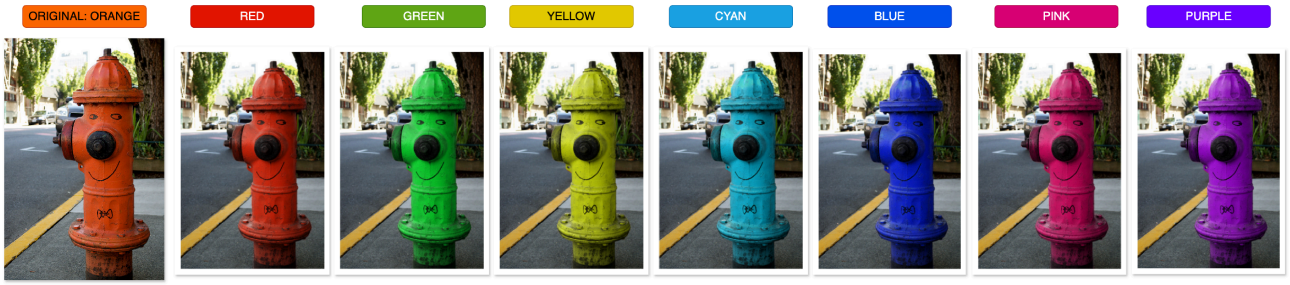


Figure 3.9: Chromatic variants achieved by manipulating the hue channel. In this case, the red and orange variants are very similar. MSCOCO masks are used.

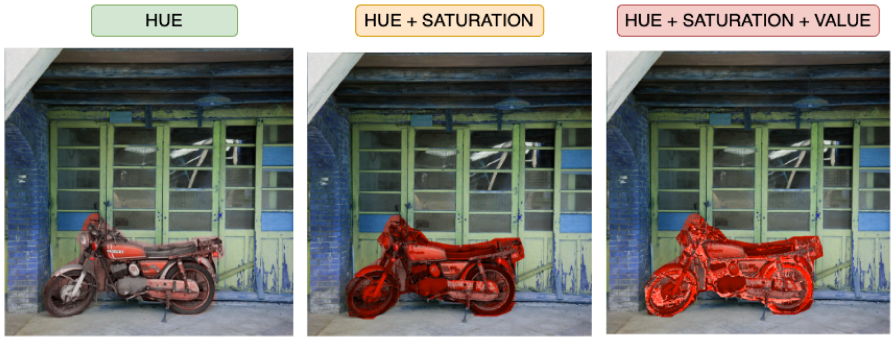


Figure 3.10: Differences in channel manipulations, MSCOCO masks.

Saturation

When manipulated via linear scaling, the saturation channel produces extremely unrealistic results (3.10). However, it is unnecessary to alter this channel for colors that differ in hue. The only colors for which saturation alteration makes sense are gray, white, and black. Unlike chromatic colors, which correspond to specific wavelengths of the visible spectrum, these achromatic colors do not derive from a single predominant wavelength. Instead, they result from a uniform or balanced combination of all wavelengths in the visible spectrum.
To achieve a color devoid of saturation, it is sufficient to scale the pixels by a parameter *alpha*, i.e., perform a simple multiplication rather than the scaling applied to the hue channel.

Figure 3.11: Gray variants achieved through saturation channel manipulation; MSCOCO masks are used.

Value

As previously analyzed, altering light intensity produces unrealistic effects, especially when using linear scaling. Like saturation, we use a parameter to multiply the pixel value instead of applying scaling.

The colors to differentiate using this channel are brown-orange and white-black. In the former case, a minimal variation (low multiplicative parameter) suffices to separate the two colors, while in the latter, the distance should be maximized (high multiplicative parameter). As seen in 3.12, variations for white and black are highly unrealistic.

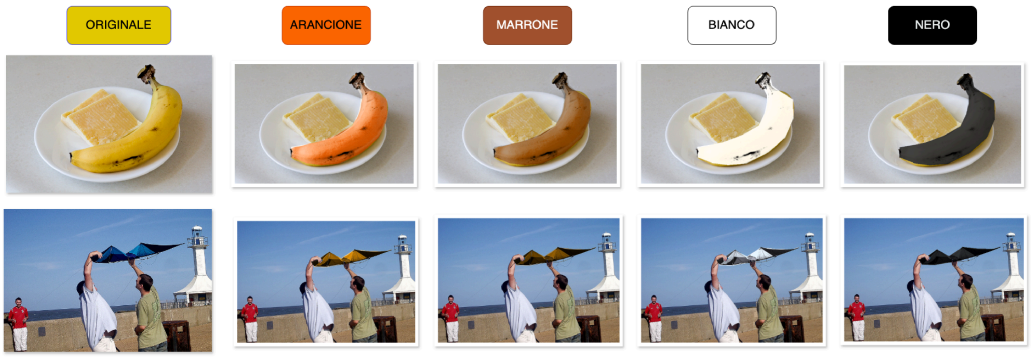


Figure 3.12: Manipulations of the value channel; MSCOCO masks are used.

3.3.2.2 Other Challenges

Lack of Chromaticity in Base Colors

So far, we have analyzed the generation of chromatic variants from saturated colors (red, orange, green, yellow, blue, cyan, pink, purple). If the original color of the object to recolor belongs to brown, gray, white, or black—less saturated colors—the recoloring process is compromised. Applying linear scaling to saturation does not add information about the interaction between color and light or the surface's properties, failing to simulate these factors realistically. The result is a representation that is too intense, insufficiently intense, or unnatural relative to the original image context (3.13). For this reason, some colors are avoided in dataset construction (details in this section).

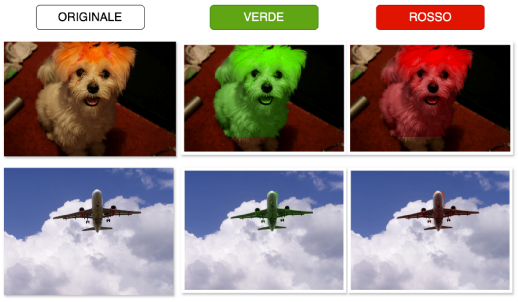


Figure 3.13: Chromatic variations from low-saturation base colors; MSCOCO masks are used.

Recoloring Area

When referring to an object by its color, it is understood that the object is not entirely composed of that shade; instead, the dominant color is referenced. During image manipulation, even with the object of interest's mask, care must be taken to recolor only the parts defining the object's original color.

Figure 3.14: Example of an object with multiple colors. Here, the carrot is defined as orange, but the carrot's green leaves are also part of its mask. A chromatic variant of this image should modify only the area corresponding to the object's identifying color.

An additional segmentation based on color is thus desired, while adhering to pre-processing techniques. Below, I present the methods explored for resolving this task.

Resolution Attempts

Closest Range Calculation

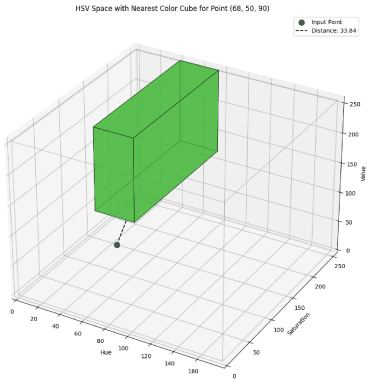


Figure 3.15: Calculation of the closest color to a given value in the HSV space.

I developed a simple function that calculates the closest color from a point in the HSV space. It is used to generate a mask by selecting all pixels closest to a specific color.

Flood Fill

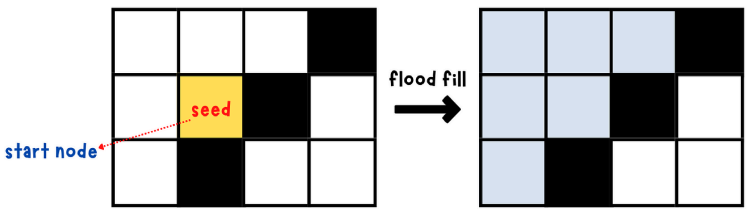


Figure 3.16: Visualization of the Flood Fill algorithm.

The Flood Fill algorithm is a method used to fill a connected area of pixels sharing similar properties (in this case, color) within an image. The algorithm starts from a specified pixel and "expands" the area by checking adjacent pixels and adding those that meet a condition (tolerated color distance) to the fill. The starting pixel

is randomly chosen among those in the object's mask; if the closest color to it is the one of interest, it is selected; otherwise, the next pixel is evaluated.

Edge Detection

Figure 3.17: Visualization of the Edge Detection algorithm.

Edge detection is an image processing technique used to identify the contours or edges of objects in an image. Edges represent significant variations in color or brightness intensity and are essential for recognizing shapes and segmenting objects. The idea here is to use the areas of the produced segmentations, calculating the mean value in the HSV space. If the closest color to this mean is the one of interest, the area is selected. Unfortunately, segmentations produced for complex images are unsatisfactory.

Results

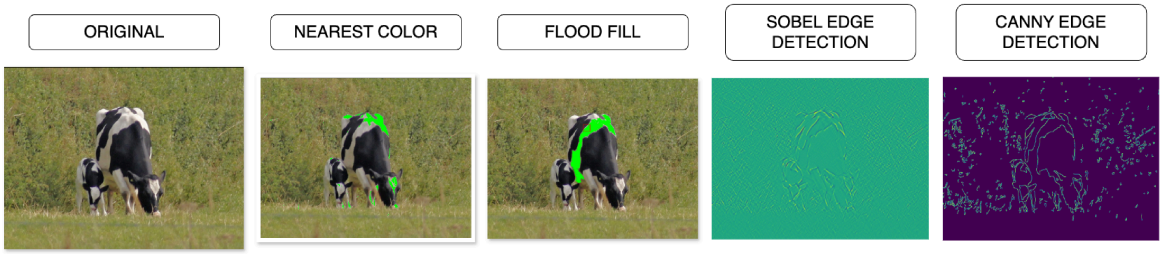


Figure 3.18: Attempts at color-based segmentation; in this case, I aim to select the white parts of the animal in the image.

The analyzed methods, in addition to being computationally expensive (about 3 minutes of computation per image), fail to achieve satisfactory results due to the complexity of natural scenes (4.18). Reflections and shadows caused by light and objects in the scene produce significant variations in the pixel values associated with the area, and the resulting segmentation is often imprecise. Given that the model's training objective focuses on the color attribute, it is essential to ensure that generated images accurately present the desired color. For this reason, an approach prioritizing the risk of chromatic overlap over insufficient segmentation is adopted. Consequently, these methods are not employed in the process.

Chapter 4

Synthetic Dataset Generation for Chromatic Variants

After developing the recoloring function, I applied it to a starting dataset (MSCOCO) to generate the proposed synthetic dataset. In this chapter, I will first describe the initial analysis of the data provided by MSCOCO, how they are extracted, and how they are used. Then, I will analyze the produced dataset of chromatic variants, which is subsequently used for fine-tuning the CLIP model.

4.1 Analysis and Extraction of Starting Data

This section focuses on analyzing the starting dataset and the methods for extracting relevant data. Before applying the recoloring function, a detailed study of the nature of the examples provided by MSCOCO was necessary to identify potential development challenges. Following this, I will discuss the techniques, implemented through simple Python functions, for extracting the relevant data. The motivations for extracting this information and the selection criteria for the dataset examples, based on these annotations, will also be analyzed.

4.1.1 Data Study

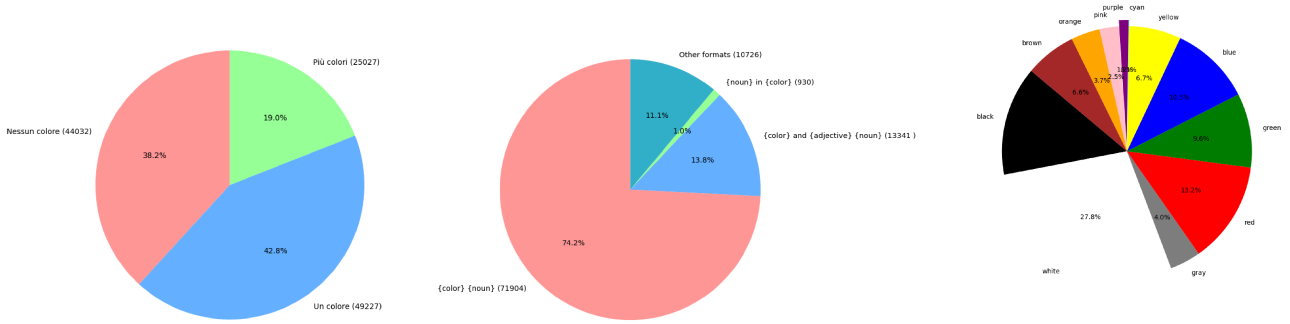
In addition to images, the following annotations are essential for project development: **Textual Descriptions**, **Entities**, and **Segmentation Masks**. From the descriptions, I extract information about colors and the objects to which these colors refer; detected entities verify the presence of segmentations; and the masks serve as inputs for the recoloring algorithm.

Textual Descriptions

Textual descriptions are necessary for extracting examples containing chromatic information. It is essential to avoid using external *Image Captioning* models to generate descriptions with color references. Since we aim to

improve recognition of this attribute, relying on the current performance of such models would introduce bias. Examples are selected from those whose descriptions contain at least one color reference, which, as shown in 4.1 a), represents the majority of MSCOCO data.

From textual descriptions, the key information extracted is: **Color** and **Referenced Object**.



(a) Study of color references per image

(b) Analysis of sentence formats containing colors

(c) Study of types of mentioned colors

Figure 4.1: Analysis of MSCOCO descriptions

Entities

The detected object classes (or entities/categories) are crucial for verifying whether the dataset contains segmentation mask information for objects possessing a chromatic attribute mentioned in the description. Additionally, studying the number of instances of an object in an image is particularly important.

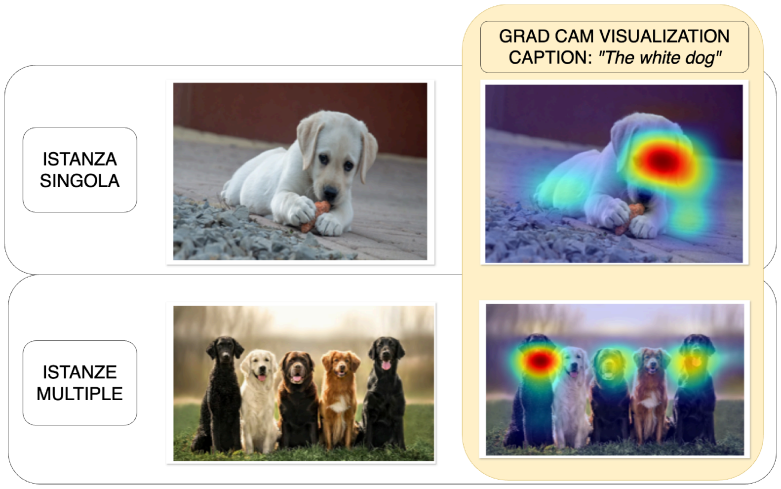


Figure 4.2: Study of object instances; the images on the right show the result of the *GradCam Visualization* algorithm. Colored areas (green, yellow, red) correspond to the pixels the model focuses on for a specific text. The redder the color, the more the model interprets the area as relevant to the provided description. A simple implementation of this algorithm is available at this link.

Suppose we extract from a description the color "White" referring to a "Dog." Looking at the images in 4.2, which use the *GradCam Visualization* algorithm [18] to generate a *Heat Map* of the model's activation zones associated with a specific description, we observe the following:

- If the object's instance is singular, we can confidently select the correct mask, ensuring that the model will focus on it.

- For multiple instances, selecting the correct mask to identify the referenced dog requires annotations specifying the color of an object within a mask (not provided in MSCOCO), relying on an external *Visual Question Answering* model (introducing strong bias), or using pre-processing algorithms.

Even if we possess the necessary information to identify the correct mask, additional challenges remain. Observing the *GradCam Visualization* for an image with multiple instances shows how the model can activate incorrectly for other instances, confirming the model's poor performance on this specific task. This could cause confusion during training: what should the model learn to separate in the contrastive phase?
For these reasons, only examples with single object instances are selected. This choice is supported by a study of their frequency, shown in 4.3.

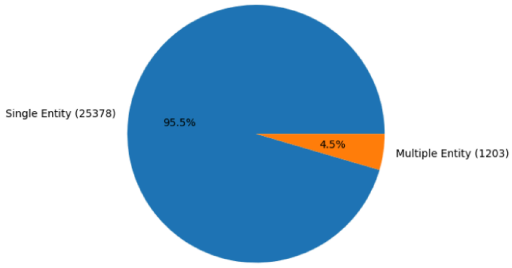


Figure 4.3: Analysis of the number of object instances with color references in an MSCOCO subset

Segmentation Masks

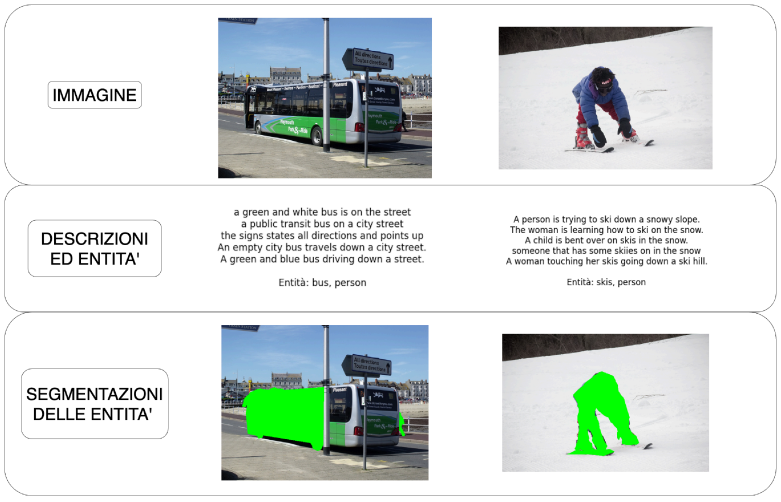


Figure 4.4: Examples of images and annotations in MSCOCO

Each image includes segmentation masks for every instance of detected entities.
During the development of the recoloring algorithm and analysis of the generated dataset, several issues arose due to the imprecision of segmentation masks. In 4.4, for instance, the mask for the bus entity is highly inaccurate. Using Grounding Dino and SAM, as analyzed in the dedicated section, resulted in notable improvements.

4.1.2 Extraction Methods

Extraction of Colors and Referenced Objects

To extract colors and referenced objects from textual descriptions of images provided in the MSCOCO dataset, I used the spaCy library. It offers a function capable of identifying whether a word in a sentence is a verb, noun, adjective, etc.
As shown in 4.1 b), most sentences containing color references follow the **"noun color"** format (Format 1), such as:

> A man is driving a *green bus*.
> Three office workers wearing a *black tie*.
> ...

The remaining distribution includes formats like **"color and adjective"** (Format 2) and **"noun in color"** (Format 3). Other formats are not uniform, making it challenging to define algorithms for extraction. Examples of the "color and adjective" format include:

> A *blue and bright lake*.
> A *black and heavy stone*.
> ...

While examples of the "noun in color" format include:

> The *lady in red*.
> ...

Extraction of the color and the referenced entity is straightforward:

- First, define the set of colors to work with.
- Then:
  - For Format 1: Iterate through the words until a color from the defined set is found. Use spaCy to verify if the next word is a noun. If true, check whether the noun is a detected entity in MSCOCO annotations; if not, continue scanning. If true, store the identified color, its position in the sentence, and the noun, then continue scanning.
  - For Format 2: Follow the same logic but verify that the word following the color is "and," then use spaCy to check that the subsequent word is an adjective and the word after that is a noun. Store the color, noun, and position of the color in the sentence, then continue scanning.
  - For Format 3: Check for a color followed by the word "in" and then a noun, following the same logic as the previous formats.

Key notes:

- The algorithm continues scanning after finding one color-noun pair, allowing the generation of multiple synthetic images for a single MSCOCO entry if multiple pairs are detected in the captions. Since each example has five captions, only unique pairs are stored.
- Objects described by two colors (e.g., "a black and white dog") are excluded. Proper recoloring in such cases would require additional segmentation based on the object's colored areas, as analyzed in the previous chapter.

Enhancement of Segmentation Masks

As previously mentioned, the precision of segmentation masks is crucial for the quality of the synthetic dataset generated through recoloring. The combined use of Grounding Dino and SAM [2], guided by the entities extracted from dataset descriptions, significantly improves this aspect (see 4.5).
Note that verifying the number of instances of the extracted entity is a prerequisite to enhancing the segmentation masks.

Figure 4.5: Comparison of segmentation masks

Standard Format for Recoloring

To make the pipeline independent of the dataset used and therefore available for future research, I defined the following standard format for preparing custom data.
The format requires defining an **ID** for the image, storing **Captions** and **Entities**. For each entity, the corresponding **segmentation masks**, separated by instance, must also be included:

{

```
    "image_id": 12345,
    "captions": [
        "A group of people riding bicycles."
    ],
    "entities": {
        "person": [
            [[x1, y1, x2, y2, ...], [...]], # First instance of 'person'
            [[x3, y3, x4, y4, ...], [...]] # Second instance of 'person'
        ],
        "bicycle": [
            [[x1, y1, x2, y2, ...]] # Segmentation mask for 'bicycle'
        ]
    }
}
```

4.2 Synthetic Dataset

Data Used

Based on the previously analyzed challenges, the synthetic dataset was generated using the following filters applied to the original dataset:

- **Colors used:** Red, Green, Yellow, Blue, Cyan, Orange, Pink, Purple. *Addressed Issue:* Naturalness preservation.
- **Single instances:** Only objects uniquely present in the image are selected. *Addressed Issue:* Referential uniqueness.
- **Avoid objects described with two colors:** If an object is described with two colors in at least one of the five captions, it is excluded. *Addressed Issue:* Area to recolor.

To avoid excessive computational costs and potential overfitting, **500 starting images** from the MSCOCO training set were used, generating **3,500 recolored images** for a total of **4,000 image-description pairs**.

Format

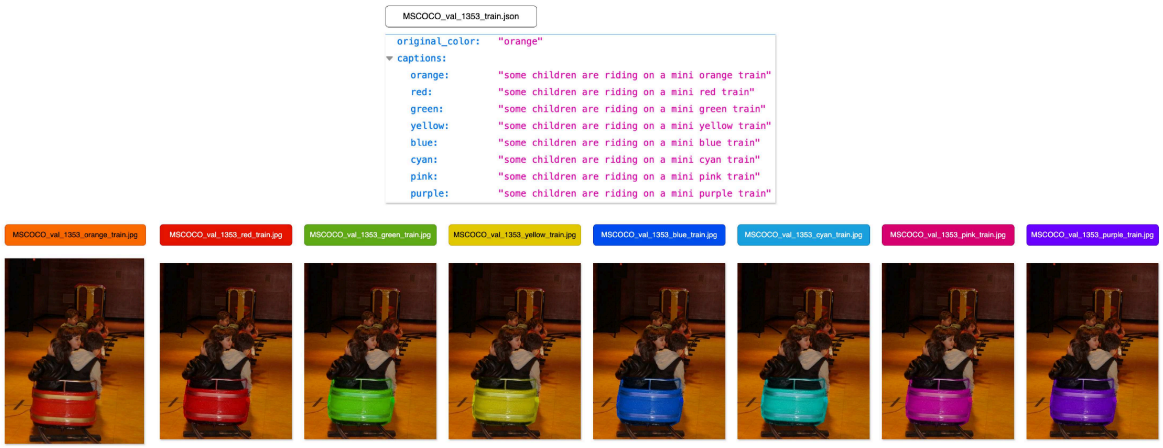


Figure 4.6: Synthetic dataset storage structure

Two folders are generated: one containing images and the other containing descriptions. Each image is saved as 'Dataset_Val/Train_id_color_entity.jpg'; metadata files are saved as 'Dataset_Val/Train_id_entity.json'. Each JSON file contains all modified descriptions related to an object in an image, referencing all generated chromatic variants.

Recoloring Examples

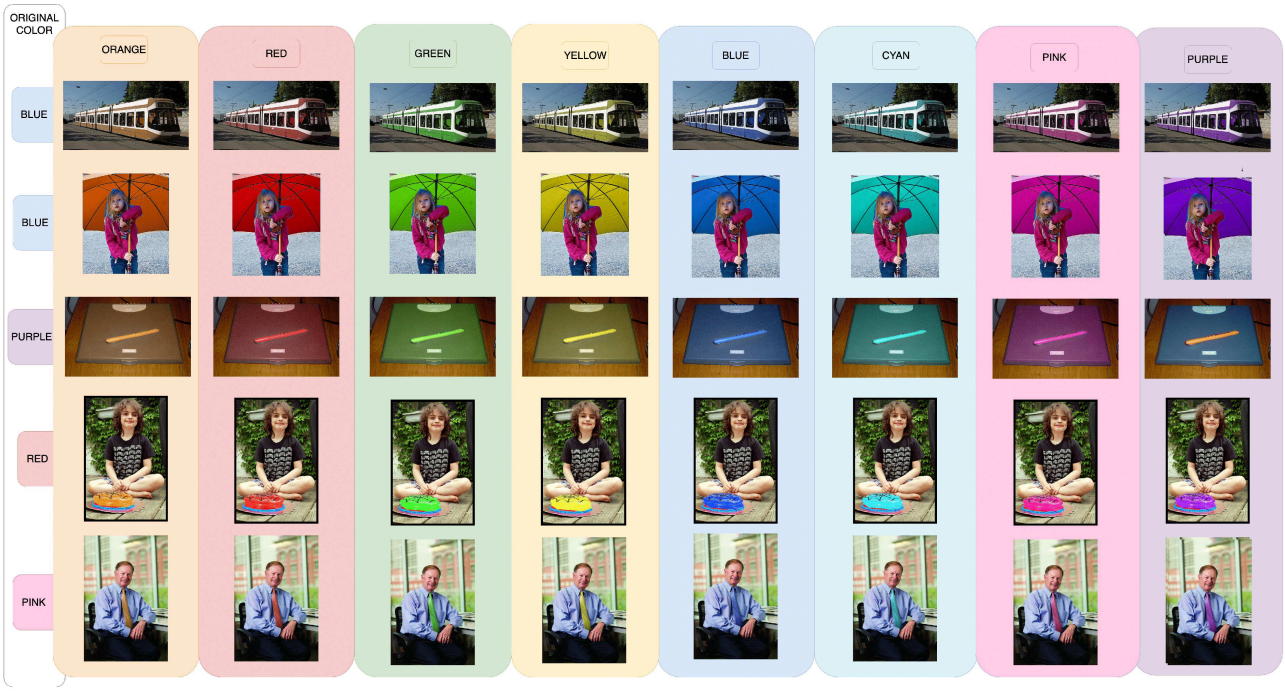


Figure 4.7: Examples of recoloring

Chapter 5

Fine-tuning

Fine-tuning is a crucial technique in machine learning that allows adapting a pre-trained model to a specific task. The model, initially trained on generic datasets, is refined using a more focused dataset to optimize its performance on a specific problem.

In this chapter, after a theoretical introduction, I will provide a detailed description of the implementation process for fine-tuning the CLIP ViT-Base-Patch32 model, analyzing the dataset construction, technical specifications, and the evaluation methods used to measure performance.

5.1 Theoretical Background

Fine-tuning uses pre-trained models as a starting point, leveraging the knowledge acquired during the initial training to reduce computational costs and improve convergence speed. Typically, the early layers of the model are frozen to preserve the general representations learned, while the subsequent layers are adapted to the new task by optimizing their parameters on a specific dataset.

Contrastive Learning

The contrastive learning paradigm, used in fine-tuning the CLIP model, assumes that similar data pairs should be close in a latent representation space, while dissimilar pairs should be separated. The pairs are categorized as follows:

- **Positive pairs:** formed by an image and a textual description sharing a common attribute, such as color.
- **Negative pairs:** consisting of an image and a description that do not share the common attribute.

In this work, positive pairs are those where the object's color in the image matches the color described in the text. Negative pairs are generated by associating images with descriptions containing colors different from the one represented.

Loss Function

```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

Figure 5.1: Pseudocode of the CLIP model training core, presented in the related paper [16].

The *Cross Entropy Loss* was used for fine-tuning, consistent with the original implementation of the CLIP model (Figure 5.1). This function measures the similarity between latent representations of images and texts. Initially, two losses are computed:

$$\mathcal{L}_i = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{\exp(\text{logits}[i,i])}{\sum_{j=1}^{n}\exp(\text{logits}[i,j])}\right),$$

$$\mathcal{L}_t = \frac{1}{n}\sum_{i=1}^{n} -\log\left(\frac{\exp(\text{logits}[i,i])}{\sum_{j=1}^{n}\exp(\text{logits}[j,i])}\right),$$

The final loss is the mean of the two:

$$\mathcal{L} = \frac{\mathcal{L}_i + \mathcal{L}_t}{2}.$$

Although mathematically similar, they differ in their perspective:

- $\mathcal{L}_i$ operates along the **rows** of the logits matrix ($axis = 0$), treating each image as a reference and calculating the conditional probability with respect to all texts.
- $\mathcal{L}_t$ operates along the **columns** of the logits matrix ($axis = 1$), treating each text as a reference and calculating the conditional probability with respect to all images.

The probabilities differ because they reflect two distinct tasks:

- **Image → Text Alignment:** How well the model associates each image with the correct text among all textual alternatives.
- **Text → Image Alignment:** How well the model associates each text with the correct image among all visual alternatives.

For example, an image might be more ambiguous than the texts (and therefore harder to associate with a specific text), while a detailed text might strongly correlate with a unique image. This results in different conditional probabilities for the two cases. However, while this dual approach made sense for CLIP's general-purpose training, it is redundant in our specific context. As shown in Figure 5.2, both the images and texts in a batch for our model vary only in the color attribute. Therefore, only one computation of the *Cross Entropy Loss*, corresponding to $\mathcal{L}_i$, is performed.

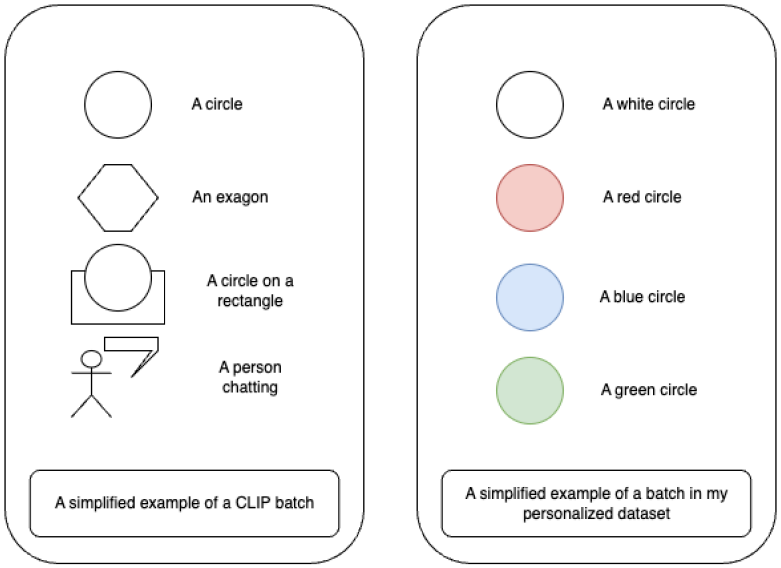


Figure 5.2: Simplified example explaining the difference between CLIP batches and those created for this project.

5.2 Fine-tuning Implementation

This section describes the implementation details of the fine-tuning process, including dataset construction and adopted parameters.

5.2.1 Dataset Construction

The dataset used for fine-tuning was built from 500 examples from the MSCOCO *Train* dataset, selected using the filters described in the relevant section. For each selected image:

- The original image and its associated description were retained.
- Seven recolorations were generated, each with a modified description reflecting the object's new color.

In total, the dataset includes:

$$500 \text{ (original examples)} \ + 500 \times 7 \text{ (color variants)} \ = 4000 \text{ examples (image-description pairs).}$$

The dataset was split into a **training set** and a **validation set** as follows:

- **Training set:** 3200 examples (80%), consisting of 400 original images and their color variants.
- **Validation set:** 800 examples (20%), consisting of 100 original images and their color variants.

Validation Set

For testing, only the 100 original images from the validation set are used. The remaining 700 variants are excluded to avoid overlap with images in the training set, ensuring that the **Validation Set without recolorations** is used for evaluation.

Complementary Dataset

To evaluate the model's ability to generalize color recognition, an additional **complementary dataset** was created. This dataset contains images from MSCOCO whose descriptions include colors not seen during training: *black, gray, white, brown*. It was used exclusively for testing model performance.

5.2.2 Training Parameters

Fine-tuning was conducted with the following parameters:

- **LoRa Rank:** 8
- **Optimizer:** AdamW
- **Temperature:** 0.07

LoRA Application

Low-Rank Adaptation (LoRA) was employed to update the weights of the 12th (final) layer of both the visual and textual encoders of CLIP ViT-Base-Patch32. Projections Q (*Query*), K (*Key*), V (*Value*), and Out (*Output*) within the self-attention mechanism were refined, reducing trainable parameters while maintaining model stability. LoRA's low-rank decomposition minimizes computational and memory requirements, making it a resource-efficient choice for fine-tuning [7].
The decision to modify only the final layer was aimed at avoiding overfitting and preserving the model's generality.

5.3 Performance Evaluation

5.3.1 Training Progression

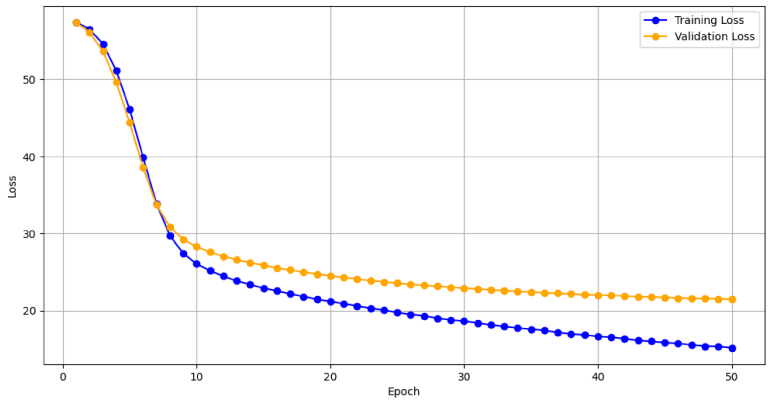


Figure 5.3: Training loss progression.

The training loss progression, shown in Figure 5.4, indicates slight **overfitting** due to the limited dataset size. The relatively high loss values, despite showing a positive trend, reflect the constraints of available computational resources.

5.3.2 Task-Specific Analysis

To evaluate the model's performance on color recognition, several metrics were used to provide comprehensive insights into its ability to correctly identify the color attribute. Testing was conducted on both the Validation Set and the complementary dataset.

5.3.2.1 Metrics

Accuracy

Accuracy measures the percentage of examples where the description containing the correct color attribute receives the highest similarity score. It is calculated as:

$$\text{Accuracy} = \left(\frac{\text{Number of correct predictions}}{\text{Total number of examples}} \right) \times 100\%.$$

Average Difference

This metric assesses the mean difference between the similarity score of the positive caption and those of negative captions. It quantifies how well the model separates positive and negative pairs.

Standard Deviation of Differences

This metric measures the variability in differences between positive and negative captions. High variability is expected due to the differences in how colors (e.g., red vs. orange) are perceived. The metric is calculated as:

$$\sigma_j = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\Delta_i^{(j)} - \mu_j\right)^2}.$$

5.3.2.2 Results

Table 5.1: Performance comparison: fine-tuned model vs. CLIP baseline.

Model	Accuracy (%)	Avg. Difference	Std. Dev.
Baseline CLIP	63.00	0.0199	0.0090
Fine-tuned (10 epochs)	67.00	0.1336	0.0518
Fine-tuned (50 epochs)	**76.00**	**0.2051**	**0.0734**

The results demonstrate significant improvements in all metrics, particularly with longer training (50 epochs).

5.3.3 Correlation with Human Judgments

The model's performance was further evaluated against human annotations using the Flickr8k dataset, correlating similarity scores with expert-provided ratings. The fine-tuned model showed reduced generality with increased epochs.

5.3.3.1 Metrics of Correlation

Kendall Tau, Spearman Rho, and Pearson R were used to compare model similarity scores against human evaluations, highlighting the balance between specificity and generality.

5.3.3.2 Results and Future Directions

While fine-tuning improved the task-specific performance of color recognition, future work will focus on balancing specificity and generalization by leveraging a more diverse and extensive dataset. A hybrid dataset including non-color-specific examples could enhance the model's robustness and mitigate overfitting.

Bibliografy

Articoli

[1] Mahmoud Afifi et al. “Image Recoloring Based on Object Color Distributions”. In: (2019). Ed. by Paolo Cignoni and Eder Miguel. ISSN: 1017-4656. DOI: 10.2312/egs.20191008.

[2] Liu et al. “Grounding dino: Marrying dino with grounded pre-training for open-set object detection”. In: *arXiv preprint arXiv:2303.05499* (2023).

[3] Coloma Ballester et al. “Influence of Color Spaces for Deep Learning Image Colorization”. In: *ArXiv* abs/2204.02850 (2022). URL: https://api.semanticscholar.org/CorpusID:247996442.

[4] Shida Beigpour and Joost van de Weijer. “Object recoloring based on intrinsic image estimation”. In: (2011), pp. 327–334. DOI: 10.1109/ICCV.2011.6126259.

[5] Meng-Yao Cui et al. “Towards natural object-based image recoloring”. In: *Computational Visual Media* 8 (June 2022), pp. 317–328. DOI: 10.1007/s41095-021-0245-5.

[6] Jack Hessel, Ari Holtzman, and Maxwell Forbes. “CLIPScore: A Reference-free Evaluation Metric for Image Captioning”. In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing* (2021). DOI: https://doi.org/10.48550/arXiv.2104.08718.

[7] Edward J. Hu et al. “LoRA: Low-Rank Adaptation of Large Language Models”. In: (2021). DOI: https://doi.org/10.48550/arXiv.2106.09685.

[8] Le-Khac et al. “Contrastive Representation Learning: A Framework and Review”. In: *IEEE Access* 8 (2020). DOI: 10.1109/ACCESS.2020.3031549.

[9] Siavash Khodadadeh et al. “Automatic Object Recoloring Using Adversarial Learning”. In: (2021), pp. 1487–1495. DOI: 10.1109/WACV48630.2021.00153.

[10] Ichiro Kuriki. “Effect of material perception on mode of color appearance”. In: *Journal of Vision* 15.8 (2015), pp. 4–4. DOI: 10.1167/15.8.4. URL: https://doi.org/10.1167/15.8.4.

[11] Lin and TY et al. “Microsoft COCO: Common Objects in Context”. In: *Computer Vision – ECCV 2014* (2014), pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48.

[13] Andrea Avendano Martinez, Jake Zueina, and Jaclyn Pytlarz. “The Impact of Background Luminance on the Perception of Chromatic Lightness”. In: *SMPTE 2023 Media Technology Summit* (2023). URL: https://api.semanticscholar.org/CorpusID:267295561.

[15] Luca Parolari, Elena Izzo, and Lamberto Ballan. “Harlequin: Color-driven Generation of Synthetic Data for Referring Expression Comprehension”. In: *Proc. of International Conference on Pattern Recognition (ICPR)* (2024), pp. 1–3. URL: https://hdl.handle.net/11577/3523768.

[16] Alec Radford et al. “Learning Transferable Visual Models From Natural Language Supervision”. In: *CoRR* abs/2103.00020 (2021). URL: https://arxiv.org/abs/2103.00020.

[17] Sara Sarto, Manuele Barraco, and Marcella Cornia. “Positive-Augmented Contrastive Learning for Image and Video Captioning Evaluation”. In: (2023). DOI: https://doi.org/10.48550/arXiv.2303.12112.

[18] Ramprasaath R. Selvaraju et al. “Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization”. In: *CoRR* abs/1610.02391 (2016). URL: http://arxiv.org/abs/1610.02391.

[19] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* (2017). URL: http://arxiv.org/abs/1706.03762.

[21] Mert Yuksekgonul and Federico Bianchi. “When and why vision-language models behave like bags-of-words, and what to do about it?” In: *International Conference on Learning Representations* (2022). DOI: https://doi.org/10.48550/arXiv.2210.01936.

[22] Yunan Zeng et al. “Investigating Compositional Challenges in Vision-Language Models for Visual Grounding”. In: (2024). URL: https://openaccess.thecvf.com//content/CVPR2024/papers/Zeng_Investigating_Compositional_Challenges_in_Vision-Language_Models_for_Visual_Grounding_CVPR_2024_paper.pdf.

Webliography

[12] *LORA*. URL: https://huggingface.co/docs/peft/main/en/conceptual_guides/lora.

[14] *MSCOCO*. URL: https://cocodataset.org/#home.

[20] *VIMP group*. URL: http://vimp.math.unipd.it/index.html#.