A MINI PROJECT REPORT ON

**AN EFFICIENT METHOD OF COLOUR CONVERSION OF VIVID TONES**

Submitted in partial fulfillment of the requirements for the award of a degree of

**BACHELOR OF TECHNOLOGY**

**in**

###### ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

**BY**

GAJJI CHAKRAPANI (20D41A6620)

BETHI DIVYA (20D41A6609)

VUPPALA LIKITHA (20D41A6660)

AVULA RISHIKESH SHIVADHAR REDDY (20D41A6604)

Under the esteemed guidance of

**DR.ADELINE JOHNSANA J.S**

(Associate Professor)



DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

**SRI INDU COLLEGE OF ENGINEERING AND TECHNOLOGY**

(An Autonomous Institution under UGC, accredited by NBA, Affiliated to JNTUH) Sheriguda, Ibrahimpatnam (2023-2024)

SRI INDU COLLEGE OF ENGINEERING & TECHNOLOGY

**(An Autonomous Institution under UGC, Accredited by NBA, Affiliated to JNTUH**)

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING**



**CERTIFICATE**

Certified that the Mini Project entitled “**VIVID TONES**”is a bonafide work carried out by **GAJJI CHAKRAPANI (20D41A6620), BETHI DIVYA (20D41A6609), VUPPALA LIKITHA (20D41A6660), AVULA RISHIKESH SHIVADHAR REDDY (20D41A6604)** in partial fulfillment for the award of **Bachelor of Technology** in **Artificial Intelligence and Machine Learning** of SICET, Hyderabad for the academic year 2023- 2024. The Project has been approved as it satisfies academic requirements concerning the work prescribed for **IV YEAR, I-SEMESTER** of **B. TECH** course.

**Dr. Adeline Johnsana J.S Mrs. G. Uma Maheswari**

**INTERNAL GUIDE** **HOD - AIML**

**EXTERNAL EXAMINER**

**ACKNOWLEDGMENT**

With great pleasure, we want to take this opportunity to express our heartfelt gratitude to all the people who helped in making this project a success. We thank the almighty for giving us the courage & perseverance in completing the project.

We are thankful to the principal Prof. **Dr. G. Suresh**, for permitting us to carry out this project and for providing the necessary infrastructure and labs.

We are highly indebted to, **Mrs. G. Uma Maheswari**, Head of the Department of Artificial Intelligence and Machine Learning, Project Guide, for providing valuable guidance at every stage of this project.

We are grateful to our internal project guide, **Dr. Adeline Johnsana J.S. (Associate Professor)** for her constant motivation and guidance given by her during the execution of this project work.

We want to thank the Teaching and non-teaching staff of the Department of Artificial Intelligence and Machine Learning for sharing their knowledge with us.

Last but not least we express our sincere thanks to everyone who helped directly or indirectly with the completion of this project.

GAJJI CHAKRAPANI - 20D41A6620

BETHI DIVYA - 20D41A6609

VUPPALA LIKITHA - 20D41A6660

AVULA RISHIKESH SHIVADHAR REDDY - 20D41A6604

**TABLE OF CONTENTS**

**PAGE NO:**

**ABSTRACT**

**LIST OF FIGURES**

**1. INTRODUCTION………………………………………………..….**1

* 1. INTRODUCTION………………………………………………....2
  2. RESEARCH AND SIGNIFICANCE……………………………..3

**2. LITERATURE AND SURVEY…………………………….……**16

2.1 LITERATURE AND SURVEY – 1 ……………………….....16

2.2 LITERATURE AND SURVEY - 2………………………….…………..17

2.3 LITERATURE AND SURVEY – 3 ……………………………………….

**3. AIM AND SCOPE**…………………..………….20

3.1 AIM OF THE PROJECT……………………………………………….32

3.2 SCOPE …..……………….34

3.3 OBJECTIVES ……………………………………………………………….…36

3.4 ADVANTAGES ………………………………………………

3.5 CROWD COUNTING …………………………………………………..

3.5.1 DENSITY CROWD COUNTING ……………………………….

3.5.2 REGRESSION BASED COUNTING ……………………………

3.5.3 DENSITY MAP BASED COUNTING ……………………………..

3.6 ABLATION STUDIES……………………………..

3.6.1 DENSITY MAPS………………………..

3.6.2 KERNEL STUDIES ………………………………………….

3.6.3 REGULARIZATION…………………………………………..

3.6.4 SELF-ATTENTION MODULE ……………………

3.6.5 LOCAL COUNTING PERFORMANCE………………………

3.6.6 VISUALIZATION …………………………………….

**4**. **METHODOLOGY** …………….………………….…39

4.1 EXISTING SYSTEM ………………………………….…………41

4.2 SYSTEM FUNCTIONALITIES………………………………….…………41

4.3 HARDWARE SPECIFICATION …………………..

4.4 SOFTWARE SPECIFICATION………………………

4.4.1 BACKWND TECHNOLOGIES…………………….

4.4.2 FRONTEND TECHNOLOGIES ………………………………

4.5 MODULES …………………….

4.5.1 DATASET ……………………………………………

4.5.2 LOAD WEIGHTS………………………

4.5.3 PREDICTION ………………………………..

4.6 DESCRIPTION OF DIAGRAM………………….

4.7 ARCHITECTURAL DESIGN …………………………………

4.8 COUNTING BY DETECTION ………………………..

4.8.1 MONOLITHIC DETECTION ……………………….

4.8.2 PART BASED DETECTION ………………….

4.8.3 SHAPE MATCHING ……………………….

**5. RESULTS AND DISCUSSION …………………….……………**42

**6. CONCLUSIO AND FUTUREWORK …………………………………….**

**REFERENCES …………………………**

**APPENDIX ……………………….**

1. **SOURCE CODE ………………………….**
2. **JOURNAL PAPER …………………………**
3. **PLAGARISM REPORT ………………………………**

**ABSTRACT**

Object detection (OD) in crowded scenes is a challenging task since objects are densely distributed and partially overlapped. In this paper, we propose a novel OD method by fully exploring the information provided by the image and its estimated density map. Our proposed OD method consists of two main stages. Initial object locations are firstly computed based on object spatial distribution information obtained from the estimated density maps. Inspired by the human visual attention mechanism, a saliency map which offers object boundaries is then employed to accurately estimate the bounding boxes with the support of the estimated initial object locations. We propose the task to estimate the density map of objects from single image with unknown perspective map. We follow the recent progress in object counting through density map estimation. Object density map estimation is usually suffered from scale variance of objects caused by unknown perspective. In addition, the background and irrelevant objects in the image lead to artifacts in the resulting density maps, which build up the error when heat maps are needed by aggregating density maps.

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIG NO.** | **NAME OF THE DIAGRAM** | **PAGE NO.** |
| 4.1 | BLOCK DIAGRAM | 16 |
| 4.2 | ACTIVITY DIAGRAM | 39 |
| 5.1 | INPUT DIAGRAM | 41 |
| 5.2 | GRAYSCALE IMAGE OF GIVEN INPUT |  |
| 5.3 | DENSITY CALCULATING IMAGE |  |
| 5.4 | DENSITY OF THE IMAGE |  |

# 1 . INTRODUCTION

* 1. **OVERVIEW**

Dense object counting tasks, including crowd counting, vehicle counting, and general object counting, aim to estimate the number of objects in the image. Counting tasks have practical usage for understanding crowded scenes. For example, crowd counting can be used to prevent accidents caused by overcrowding and estimate the crowd flows in station. And vehicle counting can be used for traffic management on roads or in parking lots. General object counting is useful for the management of goods in the supermarket, farms and factories.

Although counting tasks are important and useful, the real usage is still limited since dense object counting is challenging. One of the main challenges is scale variation. Since the scale of people varies dramatically in images and across different images, it is difficult to extract features for density regression. Another challenge is the occlusions among people since only a small part of each person may be visible in crowd images. Complex backgrounds may also hurt the counting performance, and the domain gap between scenes in datasets and the real world scenes also limits the usage of counting algorithms.

Current state-of-the-art methods use crowd density maps to achieve superior counting performance [1, 2, 3]. Density maps are an intermediate representation, where the sum over any region in the density map indicates the number of people in the region. First, the density maps are generated from the dot annotations, where each dot indicates a person’s location. Second, given the input image, algorithms are designed to predict the density map which is then summed to obtain the count.

In practice, the method for generating the density maps is crucial for crowd counting. Improperly generated density maps may dramatically hurt the counting performance – the choice of the kernel bandwidth or kernel shape used to generate the density map is often dataset dependent, and such choices often do not work across different datasets.

In the era of deep learning, we may consider current density maps as a hand- crafted intermediate representation, which is used as a target for training deep networks to count. From the standpoint of end-to-end training, these hand- designed intermediate representations may not be optimal for the particular network architecture and particular dataset.

* 1. **RESEARCH AND OVERVIEW**

In this, we call these two steps density map generation and density map estimation, respectively. Most works focus on density map estimation and ignore density map generation. Many different deep networks have been proposed to improve density map estimation, e.g., using different kernel sizes or image pyramids to handle scale variations, or using context or prior information to handle occlusions. Although density map estimation is well-studied, the generation of density maps is often overlooked and uses handcrafted designs without adequate investigation and analysis. The simplest approach to obtain a density map is to convolve the annotation dot map with a Gaussian with fixed width, i.e., place a Gaussian on each dot. Other works scale the Gaussian bandwidth according to the scene perspective, or adaptively use the local congestion level (or distance to nearest neighbors) uses human-shaped kernels, composed of two Gaussians, but is less popular since the body of the person is often occluded in crowd images.

Current state-of-the-art methods use crowd density maps to achieve superior counting performance . Density maps are an intermediate representation, where the sum over any region in the density map indicates the number of people in the region. First, the density maps are generated from the dot annotations, where each dot indicates a person’s location. Second, given the input image, algorithms are designed to predict the density map , which is then summed to obtain the count.

In practice, the method for generating the density maps is crucial for crowd counting. Improperly generated density maps may dramatically hurt the counting performance – the choice of the kernel bandwidth or kernel shape used to generate the density map is often dataset dependent, and such choices often do not work across different datasets. In the era of deep learning, we may consider current density maps as a hand-crafted intermediate representation, which is used as a target for training deep networks to count.

From the standpoint of end-to-end training, these hand- designed intermediate representations may not be optimal for the particular network architecture and particular dataset.

In this paper, we take the first step towards learnable density map representations, which are jointly trained with the density map estimator (counter). In particular, we first generate a unique normalized kernel for each object (e.g. a person or a vehicle) given an image as the input. The scale and shape of kernels are automatically learned during the joint optimization with a counter.

**2. LITERATURE SURVEY**

**2.1 LITERATURE SURVEY – 1**

|  |  |
| --- | --- |
| Title | Benchmark Data and Method for Real-Time People Counting in  Cluttered Scenes Using Depth Sensors |
| Authors | Shijie Sun , Naveed Akhtar , Huansheng Song, Chaoyang Zhang,  Jianxin Li , and Ajmal Mian |
| Published Year | 2019 |
| Efficiency | * Proved its efficiency in large volumes of extremely imbalanced data. * Construct a comprehensive feature vector. * Outperforms from previous schemes in terms of recall, precision, and time complexity. |
| Drawbacks | * Poor Application Performance. * Not based on real-time datasets. * Cannot exploit the new feature space. |
| Description | Description Vision-based automatic counting of people has widespread applications in intelligent transportation systems, security, and logistics. However, there is currently no large-scale public dataset for benchmarking approaches on this problem. This paper fills this gap by introducing the first real-world RGB-D people counting dataset (PCDS) containing over 4500 videos recorded at the entrance doors of buses in normal and cluttered conditions. It also proposes an efficient method for counting people in real-world cluttered scenes related to public transportations using depth videos. The proposed method computes a point cloud from the depth video frame and re-projects it onto the ground plane to normalize the depth information. The resulting depth image is analyzed for identifying potential human heads. The human head proposals are meticulously refined using a 3D human model. The proposals in each frame of the continuous video stream are tracked to trace their trajectories. The trajectories are again refined to ascertain reliable counting. People are eventually counted by accumulating the head trajectories leaving the scene. To enable effective head and trajectory identification, we also propose two different compound features. A thorough evaluation on PCDS demonstrates that our technique is able to count people in cluttered scenes with high accuracy at 45 fps on a 1.7- GHz processor, and hence it can be deployed for effective real-time people counting for intelligent transportation systems.  We concatenate the above mentioned four geometric features into a vector in R13. Notice that, although we do consider varied areas of IH in the above mentioned features, the compound feature only accounts for the information that is local to individual rectangles. In the real-world scenarios, the relative locations of the rectangles (that we suspect to contain human heads) can provide useful information about a bounded object being a human head or not. Therefore, we further define NRDF to account for this additional information. For each rectangle in FH , we compute NRDF as a vector in the 3D- space that is directed towards the center of the rectangle from the center of its nearest rectangle in our current set of head proposals. This feature is further illustrated in Fig. 6. The resulting NRDF  ∈ R3 is concatenated with the above mentioned feature vector to finally arrive at our compound feature vector in R16.      Background removal is a major task in many surveillance related problems. For our approach, reliable background subtraction is necessary for the success of subsequent processing of video frames. Therefore, we separately analyze the performance of our method for this task. We use the popular Gaussian mixture-based background segmentation method (MOG) and the K-nearest neighbors (KNN) based method to benchmark our technique. We note that other approaches for background subtraction also exist, however the selected baseline methods are chosen for their well- established effectiveness for the depth videos.To analyze the performance of our method for human head identification, we first manually labeled 12,148 rectangle proposals in height images for people entering the buses as ‘heads’ and ‘non-heads’. These proposals were generated automatically by the method in Sec.  We can argue that the employed classifiers are able to identify human heads in the proposals successfully. We note that the classification performance depicted by is better for the people exiting buses than for the people entering buses. The reason behind this phenomenon is that while  providing the ground truth we only    labeled those proposal rectangles as ‘heads’ that bounded complete human heads. For the case of people entering the buses, many half- heads appeared in the frames due to queuing of people on bus doors. On scrutiny, we found that most of those heads resulted in false positive identifications in our experiment. However, this is not problematic for the overall approach because the final results rely more strongly on tracking of heads on multiple frames, and the half- heads eventually transform into complete heads in the subsequent video frames. We also provide the details of precision, recall and the f1-scores for our head identification experiment. |

The motivation behind the provided code is to enable automatic image colorization using deep learning techniques. Image colorization is the process of adding realistic and visually pleasing colors to grayscale images. The primary goals and motivations for this project can be summarized in simple paragraphs:

Grayscale images, particularly old or historical photographs, lack the vibrant colors that can make them more engaging and relatable. The motivation is to revitalize these images by adding color, thereby preserving visual content and making it more appealing to viewers.

Traditional manual colorization is a time-consuming and labor-intensive task. The code's motivation is to automate this process, making it more efficient and accessible to a broader audience, including photographers, artists, and historians.

Deep learning models, such as ECCV16 and SIGGRAPH17, have shown remarkable capabilities in image colorization. The motivation is to harness the power of these models to produce accurate and realistic colorizations, even for complex images.

The code's goal is not only to add color but also to enhance the overall quality of images. By leveraging sophisticated models and postprocessing techniques, the motivation is to create colorized images that are visually pleasing and maintain the integrity of the original content.

Image colorization can be a valuable tool for educational purposes, allowing students and researchers to explore historical imagery more engagingly. Additionally, it provides a creative outlet for artists and designers to experiment with color and style.

By offering multiple models (ECCV16 and SIGGRAPH17), the code enables users to compare and contrast different colorization techniques, potentially leading to insights into the strengths and weaknesses of each approach.

The ability to automate image colorization has practical applications, such as photo restoration, film colorization, and enhancing user-generated content for social media and marketing.

* 1. **EXISTING SYSTEM**

The existing systems for image colorization can be divided into manual methods and automated methods:

**Manual Methods**:

* **Traditional Hand Coloring**: Historically, black and white photographs were manually colorized using paints or dyes. This labor-intensive process was performed by skilled artists.
* **Digital Image Editing Software**: Modern graphic design software, like Adobe Photoshop, provides tools for manual colorization. Artists can use brushes and layers to add color to grayscale images.
* **Colorization Services**: Some companies and professionals offer manual colorization services for individuals or organizations looking to colorize old or historical photos. This often involves skilled artists who work on each image.
* **Automated Methods:**
* **Deep Learning-Based Colorization**: Automated image colorization has seen significant advancements due to deep learning techniques.
* **Convolutional Neural Networks (CNNs):** CNN-based models are trained on large datasets of color images to learn the mapping from grayscale to color.
* **Generative Adversarial Networks (GANs):** Conditional GANs have been used to generate realistic colorizations. These models learn to produce visually convincing color images.
* **Pretrained Models**: Many automated colorization tools leverage pre-trained deep learning models to perform colorization. Users can input grayscale images, and the models add color based on their learned knowledge.
* **Online Colorization Tools**: Several online tools and software applications offer automated image colorization, making it accessible to a wide range of users. These tools often use deep learning models behind the scenes.
* **Commercial Products and Services:**

Some companies offer commercial software and services for image colorization. These tools are designed for various purposes, including photo restoration, film colorization, and enhancing digital media content.

* **Academic Research Tools:**

Researchers in the field of computer vision and image processing have developed open-source tools and libraries for image colorization. These tools often provide access to state-of-the-art models and techniques for experimentation and research.

* **Mobile Apps:**

There are mobile applications that provide on-the-go image colorization. Users can take photos with their smartphones and apply automatic colorization effects.

* **Historical and Archival Applications:**

In archival and historical preservation contexts, specialized software is used to restore and colorize historical photographs and documents, ensuring their long-term preservation and accessibility.

The existing systems vary in terms of complexity, accuracy, and the level of user involvement. Manual methods offer a high degree of control but require artistic skill and significant effort. Automated methods, especially those based on deep learning, have made image colorization more accessible and efficient. They are valuable tools for various applications, from art and entertainment to historical preservation. The choice of system depends on the specific needs and goals of users, whether it's professional image restoration or creative expression**.**

**1.3 LITERATURE SURVEY**

VividTones is a captivating field within computer vision that has seen remarkable progress in recent years. This section provides an overview of key research findings and methodologies in the realm of VividTones.

**Image Colorization Methods:** Various techniques have been employed to tackle the challenge of image colorization. Early methods relied on manual colorization or interpolation techniques. With the advent of deep learning, Convolutional Neural Networks (CNNs) gained prominence. Models like DeOldify and Colorful Image showcased the potential of deep learning in this domain.

**Deep Learning Advances:** The utilization of deep learning models, such as Generative Adversarial Networks (GANs) and Autoencoders, has revolutionized image colorization. Zhang et al. introduced a GAN-based approach in "Colorful Image Colorization demonstrating impressive results by predicting pixel-wise color distributions.

**State-of-the-Art Models:** Notably, the ECCV16 model and SIGGRAPH17 model have emerged as leading solutions for image colorization. These models leverage vast datasets and intricate architectures to produce realistic colorizations. The ECCV16 model, in particular, employs a classification network for colorization, while the SIGGRAPH17 model utilizes a deep learning framework.

**Evaluation Metrics:** Assessing the quality of colorization is crucial. Metrics such as Color Accuracy, Perceptual Similarity, and Visual Appeal have been widely used to evaluate colorization results. These metrics provide insights into how well-colorized images align with human perception.

**Challenges and Future Directions:** Despite significant advancements, challenges persist in handling complex scenes, fine details, and rare color patterns. Future research directions may involve leveraging attention mechanisms, domain adaptation, and semi-supervised learning to address these challenges.

In summary, the literature survey underscores the evolution of image colorization, from manual techniques to deep learning-driven solutions. The ECCV16 and SIGGRAPH17 models represent milestones in this field, and ongoing research continues to enhance the quality and applicability of image colorization techniques.

**1.4 CHALLENGES IN THE EXISTING SYSTEM**

Existing systems for image colorization, both manual and automated, face several challenges and limitations:

**Realism and Accuracy**:

Over-saturation and Unnatural Colors: Automated colorization models may produce oversaturated or unrealistic colors, especially when they lack sufficient context or semantic understanding.

**Lack of Contextual Understanding**:

Existing systems may struggle with understanding the semantics of images, leading to colorizations that do not align with real-world expectations (e.g., blue bananas).

**Complex Scenes:**

Automated colorization can struggle with images that contain intricate details, fine textures, or complex scenes. These challenges can result in color "bleeding" or misalignment.

**Lack of User Control:**

Some automated tools may not provide users with fine-grained control over the colorization process, limiting their ability to guide the results.

**Historical and Rare Photos:**

Automated models rely on large datasets of color images, which may not include examples of historical or rare scenes. This can affect the accuracy of colorization for such images.

**Ethical Considerations:**

There are ethical considerations surrounding the use of automated colorization for historical or archival photographs. Adding color to historical images can be controversial if not done with care and respect for the original context.

**Computational Resources:**

Deep learning-based colorization models can be computationally intensive and may require powerful hardware, which can be a limitation for some users.

**Skill and Training:**

Manual colorization methods, such as digital painting, require artists to develop skills and expertise, which can be a barrier for many users.

**Perceptual Quality:**

Evaluating the perceptual quality of colorizations remains a challenge. It can be subjective and challenging to quantify.

**Hardware and Software Compatibility:**

Some colorization tools may not be compatible with certain hardware or software environments, limiting their usability for some users.

**Preserving Original Context**:

There's a challenge in preserving the original historical context of grayscale images when colorizing them. Adding color can change the perception of the historical scene.

Advancements in AI and deep learning are continually addressing some of these challenges. However, it's important to recognize that no system, whether manual or automated, is perfect, and users should consider these limitations when using image colorization tools. Additionally, addressing ethical and historical preservation concerns is crucial in the field of image colorization.

* 1. **PROPOSED SYSTEM**

A proposed system for image colorization would aim to enhance and expand upon the existing system you provided, addressing some of its limitations and potentially offering new features and capabilities.

Below is, the outline of a possible proposed system for image colorization:

Proposed System for Image Colorization:

1. Improved model selection

2. Custom model training

3. Real-time colorization

4. Multi-model colorization

**Improved Model Selection:**

In the context of image colorization, improved model selection refers to the process of selecting or designing a more advanced and effective colorization model compared to existing approaches. This may involve research and experimentation to identify models that produce more realistic and visually appealing colorizations. Some considerations for improved model selection might include:

Deep Learning Architectures

Transfer Learning

Hybrid Models

Objective Evaluation

**Custom Model Training:**

Custom model training involves the process of training a colorization model tailored to the specific dataset or problem domain.

**Real-Time Colorization:**

Real-time colorization aims to make the colorization process efficient enough to perform in real time, meaning that colorization occurs as quickly as the user interacts with the system. Achieving real-time colorization involves optimizing the model and the inference process.

* 1. **OBJECTIVES**

The objectives of the provided code for image colorization can be summarized as follows:

* **Automated Image Colorization:** The primary objective is to automate the process of adding color to grayscale images. This automation simplifies and expedites the colorization process, making it accessible to a wider audience.
* **Utilize Deep Learning Models**: The code leverages deep learning models, specifically ECCV16 and SIGGRAPH17, for image colorization. These models have been trained on extensive datasets to effectively predict color information for grayscale images.
* **Preservation of Visual Content**: An important objective is to preserve and enhance the visual content of grayscale images. By adding color, the code aims to revitalize old or historical photographs, making them more visually appealing and engaging.
* **Efficiency and Accessibility**: Automation not only saves time but also makes image colorization accessible to individuals who may not have artistic skills. The code's user-friendly approach allows a broader range of users to utilize colorization techniques.
* **Comparison of Colorization Models**: By providing two different colorization models (ECCV16 and SIGGRAPH17), the code allows users to compare and contrast the results generated by these models. This objective facilitates experimentation and learning about the strengths and weaknesses of each model.
* **Educational and Creative Use**: The code serves educational purposes, allowing students, researchers, and history enthusiasts to explore and experiment with colorization. It also provides a creative outlet for artists and designers to apply color to their grayscale images.
* **Real-World Applications**: The code can be applied to real-world scenarios, such as photo restoration, film colorization, and enhancing digital media content for social media and marketing.
* **Post-Processing for Quality**: Another objective is the post-processing of colorized images to enhance their quality. This includes applying techniques to improve the overall appearance of the colorized images.
* **Providing Multiple Outputs**: The code generates colorized images using both ECCV16 and SIGGRAPH17 models, offering users multiple options for their colorization needs.

In summary, the code's primary objective is to provide an automated and efficient solution for image colorization, leveraging deep learning models and post-processing techniques to enhance the quality and visual appeal of colorized images. It caters to a wide range of users, from those interested in image restoration to creative artists and historians.

* 1. **METHODOLOGY**

The methodology implemented in the provided code for image colorization is based on a deep-learning approach. The key steps involved in this methodology can be summarized as follows:

* **Data Preparation:**

The methodology begins with data preparation. The code leverages pre-trained deep learning models, ECCV16 and SIGGRAPH17, which have been trained on large datasets of color images. These datasets are used to teach the models the mapping between grayscale and color images.

* **Grayscale Image Input**:

The user provides a grayscale image as input to the system. Grayscale images typically consist of a single channel representing the intensity or luminance information.

* **Model Selection:**

The methodology allows users to choose between two different colorization models, ECCV16 and SIGGRAPH17. These models have been designed to predict the color information that corresponds to the grayscale input.

* **Deep Learning Colorization**:

The selected model is applied to the grayscale image. The model's deep learning architecture processes the grayscale image and generates a colorized version. This process involves convolutional layers, activation functions, and normalization techniques to predict the color information.

* **Post-Processing:**

To enhance the quality and visual appeal of the colorized image, post-processing techniques are applied. This step can include resizing, adjusting color balance, and other image enhancement operations.

* **Multiple Outputs:**

The methodology offers users the option to generate colorized images using both ECCV16 and SIGGRAPH17 models. This allows for a comparison of results and the selection of the preferred colorization.

* **Output Presentation:**

The colorized image is presented to the user for evaluation and use. The methodology aims to ensure that the colorization output is realistic and visually pleasing.

* **Real-World Applications:**

The colorized image can be applied to various real-world scenarios, such as photo restoration, art creation, and historical image preservation. Users can utilize the colorized images for different purposes based on their objectives.

In summary, the methodology combines deep learning techniques with user-friendly functionalities to automate the process of image colorization. It provides users with options to choose from different models and post-processing operations to achieve high-quality colorized results, serving both creative and practical applications.

**1.8 HARDWARE AND SOFTWARE REQUIREMENTS**

The hardware and software requirements for running the provided code for image colorization depend on whether you want to use the code with GPU acceleration or on CPU. Below are the requirements for both scenarios:

**Hardware Requirements**:

For GPU Acceleration:

* **GPU**: A dedicated NVIDIA GPU is recommended for faster model inference. A GPU with CUDA support is highly beneficial. While the code is written to run on a CPU as well, using a GPU can significantly speed up the colorization process.

For CPU (No GPU) Use:

* **CPU**: A multi-core CPU is suitable for running the code, but note that it may be slower compared to GPU acceleration. A modern CPU with multiple cores is preferred for better performance.

**Software Requirements:**

* **Python**: The code is written in Python. You need to have Python installed on your system. It's recommended to use Python 3. x.
* **PyTorch:** The code relies on the PyTorch deep learning framework. You should install PyTorch, and you can do this using pip

pip install torch

* **PyTorch Models (Torchvision):** The PyTorch models are part of the Torchvision library. Ensure you have it installed:

pip install torchvision

* **NumPy**: NumPy is used for numerical operations. Install it if not already present:

pip install numpy

* **PIL** (Python Imaging Library): PIL is used for image loading and manipulation. You can install it with:

pip install pillow

* **Matplotlib**: Matplotlib is used for image visualization. You can install it with:

pip install matplotlib

* **scikit-image:** The code uses scikit-image for certain image processing tasks. Install it with:

pip install scikit-image

* **Jupyter Notebook (** **Optional):** If you intend to run the code in a Jupyter Notebook environment, make sure Jupyter is installed.
* **Access to the Internet:** The code uses pre-trained models, so you'll need internet access to download these models from the provided URLs.

These are the general hardware and software requirements to run the code successfully. The specific versions and configurations may vary based on your system setup, but the above components are the essential prerequisites for image colorization using the provided code.

**1.9 ORGANISATION OF THE PROJECT**

The code for image colorization demonstrates a clear and structured organization, following common practices in machine learning and deep learning projects. The code is divided into distinct sections, each serving a specific purpose. At the outset, the code begins by importing the necessary libraries and frameworks. These libraries include PyTorch for deep learning, PIL for image processing, NumPy for numerical operations, and others. This section sets up the foundational components required for the subsequent image colorization process.

Following the import section, the code defines key classes that form the core of the project. The BaseColor class contains methods for normalizing and unnormalizing color channels in the LAB color space, the pre-processing and post-processing of image data. Additionally, the code introduces two colorization models, ECCVGenerator and SIGGRAPHGenerator. These models specify the architecture for colorizing grayscale images and inherit functionalities from the BaseColor class. The code proceeds to provide functions for loading pre-trained models. The eccv16 and siggraph17 functions allow users to create instances of the colorization models. They also offer the option to load pre-trained weights, enabling the utilization of state-of-the-art models for image colorization. Image processing is a fundamental component of the project, and the code includes functions for this purpose. Notably, the loading function loads\_images, ensuring they have at least three color channels. The resize\_img function resizes images while maintaining the aspect ratio, and the preprocess\_img function extracts L channels for input to the colorization models. The heart of the code lies in the image colorization and visualization section. Here, the colorization process is demonstrated step by step. Images are loaded, pre-processed, and passed through the chosen colorization model, resulting in colorized images. The code thoughtfully provides the option to use either the ECCV16 or SIGGRAPH17 architecture, allowing flexibility for users' preferences. To ensure users can visualize the results, the code uses Matplotlib for image display. The grayscale input and the colorized output are all presented for visual assessment.

Lastly, the code considers the use of a GPU for model inference, enhancing the efficiency of the colorization process, especially when dealing with complex images. In summary, the code's organization is designed to provide clarity and maintainability.

**2. PROPOSED SYSTEM**

**2.1 ARCHITECTURE / DATAFLOW DIAGRAM**

**2.2 DESCRIPTION OF ALGORITHMS:**

**ECCV 2016 Colorization Algorithm:** The ECCV 2016 colorization algorithm is designed to add color information to grayscale images. It leverages a deep neural network architecture to predict color channels (a and b) based on the input grayscale image (L channel). The algorithm is divided into several layers, each responsible for extracting and processing features from the input image. The ECCV 2016 model begins with a series of convolutional layers, followed by activation functions (ReLU) to capture essential image features. These initial layers help in understanding the grayscale input. Batch normalization is applied to normalize the network's activations. To incorporate multi-scale information, the model uses several convolutional layers with varying kernel sizes and strides. This enables the network to capture details at different levels of granularity within the image.

The ECCV16 model, short for "ECCV Generator 2016," is a deep neural network architecture designed for the task of image colorization. This model is based on a convolutional neural network (CNN) and is capable of adding color to grayscale images. It's an important component of the code for colorization**.**

**The ECCV16 model consists of several key components:**

**1. Convolutional Layers:** The model starts with a series of convolutional layers. These layers are responsible for learning various features from the input grayscale images. Each convolutional layer applies a set of learnable filters to the input data, extracting low-level to high-level features.

**2. ReLU Activation:** After each convolutional layer, a rectified linear unit (ReLU) activation function is applied. ReLU introduces non-linearity into the model and helps in capturing complex patterns and structures in the images.

**3. Normalization:** The model employs batch normalization after some of the convolutional layers. Batch normalization is a technique that normalizes the activations of a neural network, making training more stable and accelerating convergence.

**4. Dilated Convolutions:** The ECCV16 model includes dilated convolutions. These are convolutional layers with gaps in between the filter values. Dilated convolutions have a larger receptive field, allowing the model to capture information from a wider context. This is particularly useful for image colorization as it considers spatial relationships.

**5. SoftMax Layer:** At the end of the model, there is a SoftMax layer. SoftMax is used to convert the model's output into a probability distribution over different color values. This step is crucial for mapping the grayscale input to color information.

**6. Up-sampling:** The model also incorporates an up-sampling layer, which increases the spatial resolution of the output. This is essential to match the resolution of the color channels with the input grayscale image.

**7. Normalization Functions:** Throughout the model, there are normalization functions for L, AB, and color channels, which play a role in preparing and recovering the color information.

The ECCV16 model is capable of colorizing images efficiently and is known for its ability to produce visually pleasing colorizations. It's trained on a large dataset of images and has learned to understand the relationships between grayscale and color information. In the code, the ECCV16 model is loaded and applied to grayscale images to produce colorized outputs. Its architecture is a fundamental component of the colorization process and showcases the advancements in deep learning for image processing, particularly in the context of computer vision and computer graphics**.**

**SIGGRAPH 2017 Colorization Algorithm:**

The SIGGRAPH 2017 colorization algorithm is another approach to adding color to grayscale images. It utilizes a deep neural network with a similar objective but introduces skip connections for improved performance and better handling of details. Like ECCV 2016, the SIGGRAPH 2017 model starts with a series of convolutional layers. However, it incorporates skip connections at various layers to merge low-level and high-level features. This enables the network to preserve fine details during colorization. The model produces two types of outputs. The classification output predicts color categories, while the regression output estimates the color within each category. This dual-output approach enhances colorization accuracy.

The SIGGRAPH17 model, short for "SIGGRAPH Generator 2017," is a deep neural network architecture designed for the task of image colorization. It is one of the key models included in the code for adding color to grayscale images. The SIGGRAPH17 model is an evolution of the ECCV16 model, incorporating additional improvements. However, there are notable differences and enhancements:

1. Additional Convolutional Layers: The SIGGRAPH17 model extends the depth of the network by adding more convolutional layers. These additional layers allow the model to capture increasingly complex and high-level features in the input images.

2. Leaky ReLU Activation: Instead of the standard ReLU activation, the SIGGRAPH17 model employs leaky ReLU activation functions. Leaky ReLU introduces a small negative slope for negative inputs, which can help prevent dead neurons during training.

3. Dilated Convolutions: Similar to ECCV16, the SIGGRAPH17 model includes dilated convolutions. These layers have a larger receptive field, enabling the model to capture information from a broader context. This is beneficial for understanding the spatial relationships within images.

4. Multi-Scale Processing: The SIGGRAPH17 model is designed for multi-scale processing. It has multiple convolutional layers of different scales, allowing it to analyze images at various resolutions. This multi-scale approach is valuable for capturing both fine and coarse details.

5. Classification and Regression Outputs: The model has two key outputs:

Classification Output: This output is used for semantic segmentation and classifying different regions of the image.

Regression Output: This output is responsible for predicting color information. It provides two channels representing the 'a' and 'b' values in the LAB color space.

6. State-of-the-Art Weights: The SIGGRAPH17 model is pre-trained with state-of-the-art weights, making it capable of producing high-quality colorizations. It has learned to understand the relationships between grayscale and color information from a large dataset of images.

In the code, the SIGGRAPH17 model is loaded and applied to grayscale images to produce colorized outputs. Its architecture represents advancements in deep learning for image colorization and is particularly effective in generating visually pleasing colorizations. The multi-scale processing, leaky ReLU activations, and additional convolutional layers contribute to its superior performance, making it a valuable tool for a wide range of applications in computer vision and image processing.

**3. IMPLEMENTATION & TESTING:**

# Connecting to Google Drive

from google.colab import drive

drive.mount('/content/drive')

# Impoting required modules

import torch

from torch import nn

import torch.nn as nn

from PIL import Image

import numpy as np

from skimage import color

import torch

import torch.nn.functional as F

from IPython import embed

import matplotlib.pyplot as plt

class BaseColor(nn.Module):

    def \_\_init\_\_(self):

        super(BaseColor, self).\_\_init\_\_()

        self.l\_cent = 50.

        self.l\_norm = 100.

        self.ab\_norm = 110.

    def normalize\_l(self, in\_l):

        return (in\_l-self.l\_cent)/self.l\_norm

    def unnormalize\_l(self, in\_l):

        return in\_l\*self.l\_norm + self.l\_cent

    def normalize\_ab(self, in\_ab):

        return in\_ab/self.ab\_norm

    def unnormalize\_ab(self, in\_ab):

        return in\_ab\*self.ab\_norm

class ECCVGenerator(BaseColor):

    def \_\_init\_\_(self, norm\_layer=nn.BatchNorm2d):

        super(ECCVGenerator, self).\_\_init\_\_()

        model1=[nn.Conv2d(1, 64, kernel\_size=3, stride=1, padding=1, bias=True),]

        model1+=[nn.ReLU(True),]

        model1+=[nn.Conv2d(64, 64, kernel\_size=3, stride=2, padding=1, bias=True),]

        model1+=[nn.ReLU(True),]

        model1+=[norm\_layer(64),]

        model2=[nn.Conv2d(64, 128, kernel\_size=3, stride=1, padding=1, bias=True),]

        model2+=[nn.ReLU(True),]

        model2+=[nn.Conv2d(128, 128, kernel\_size=3, stride=2, padding=1, bias=True),]

        model2+=[nn.ReLU(True),]

        model2+=[norm\_layer(128),]

        model3=[nn.Conv2d(128, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model3+=[nn.ReLU(True),]

        model3+=[nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model3+=[nn.ReLU(True),]

        model3+=[nn.Conv2d(256, 256, kernel\_size=3, stride=2, padding=1, bias=True),]

        model3+=[nn.ReLU(True),]

        model3+=[norm\_layer(256),]

        model4=[nn.Conv2d(256, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model4+=[nn.ReLU(True),]

        model4+=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model4+=[nn.ReLU(True),]

        model4+=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model4+=[nn.ReLU(True),]

        model4+=[norm\_layer(512),]

        model5=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model5+=[nn.ReLU(True),]

        model5+=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model5+=[nn.ReLU(True),]

        model5+=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model5+=[nn.ReLU(True),]

        model5+=[norm\_layer(512),]

        model6=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model6+=[nn.ReLU(True),]

        model6+=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model6+=[nn.ReLU(True),]

        model6+=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model6+=[nn.ReLU(True),]

        model6+=[norm\_layer(512),]

        model7=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model7+=[nn.ReLU(True),]

        model7+=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model7+=[nn.ReLU(True),]

        model7+=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model7+=[nn.ReLU(True),]

        model7+=[norm\_layer(512),]

        model8=[nn.ConvTranspose2d(512, 256, kernel\_size=4, stride=2, padding=1, bias=True),]

        model8+=[nn.ReLU(True),]

        model8+=[nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model8+=[nn.ReLU(True),]

        model8+=[nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model8+=[nn.ReLU(True),]

        model8+=[nn.Conv2d(256, 313, kernel\_size=1, stride=1, padding=0, bias=True),]

        self.model1 = nn.Sequential(\*model1)

        self.model2 = nn.Sequential(\*model2)

        self.model3 = nn.Sequential(\*model3)

        self.model4 = nn.Sequential(\*model4)

        self.model5 = nn.Sequential(\*model5)

        self.model6 = nn.Sequential(\*model6)

        self.model7 = nn.Sequential(\*model7)

        self.model8 = nn.Sequential(\*model8)

        self.softmax = nn.Softmax(dim=1)

        self.model\_out = nn.Conv2d(313, 2, kernel\_size=1, padding=0, dilation=1, stride=1, bias=False)

        self.upsample4 = nn.Upsample(scale\_factor=4, mode='bilinear')

    def forward(self, input\_l):

        conv1\_2 = self.model1(self.normalize\_l(input\_l))

        conv2\_2 = self.model2(conv1\_2)

        conv3\_3 = self.model3(conv2\_2)

        conv4\_3 = self.model4(conv3\_3)

        conv5\_3 = self.model5(conv4\_3)

        conv6\_3 = self.model6(conv5\_3)

        conv7\_3 = self.model7(conv6\_3)

        conv8\_3 = self.model8(conv7\_3)

        out\_reg = self.model\_out(self.softmax(conv8\_3))

        return self.unnormalize\_ab(self.upsample4(out\_reg))

def eccv16(pretrained=True):

    model = ECCVGenerator()

    if(pretrained):

        import torch.utils.model\_zoo as model\_zoo

        model.load\_state\_dict(model\_zoo.load\_url('https://colorizers.s3.us-east-2.amazonaws.com/colorization\_release\_v2-9b330a0b.pth',map\_location='cpu',check\_hash=True))

    return model

class SIGGRAPHGenerator(BaseColor):

    def \_\_init\_\_(self, norm\_layer=nn.BatchNorm2d, classes=529):

        super(SIGGRAPHGenerator, self).\_\_init\_\_()

        model1=[nn.Conv2d(4, 64, kernel\_size=3, stride=1, padding=1, bias=True),]

        model1+=[nn.ReLU(True),]

        model1+=[nn.Conv2d(64, 64, kernel\_size=3, stride=1, padding=1, bias=True),]

        model1+=[nn.ReLU(True),]

        model1+=[norm\_layer(64),]

        # Conv2

        model2=[nn.Conv2d(64, 128, kernel\_size=3, stride=1, padding=1, bias=True),]

        model2+=[nn.ReLU(True),]

        model2+=[nn.Conv2d(128, 128, kernel\_size=3, stride=1, padding=1, bias=True),]

        model2+=[nn.ReLU(True),]

        model2+=[norm\_layer(128),]

        # Conv3

        model3=[nn.Conv2d(128, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model3+=[nn.ReLU(True),]

        model3+=[nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model3+=[nn.ReLU(True),]

        model3+=[nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model3+=[nn.ReLU(True),]

        model3+=[norm\_layer(256),]

        # Conv4

        model4=[nn.Conv2d(256, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model4+=[nn.ReLU(True),]

        model4+=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model4+=[nn.ReLU(True),]

        model4+=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model4+=[nn.ReLU(True),]

        model4+=[norm\_layer(512),]

        # Conv5

        model5=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model5+=[nn.ReLU(True),]

        model5+=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model5+=[nn.ReLU(True),]

        model5+=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model5+=[nn.ReLU(True),]

        model5+=[norm\_layer(512),]

        # Conv6

        model6=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model6+=[nn.ReLU(True),]

        model6+=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model6+=[nn.ReLU(True),]

        model6+=[nn.Conv2d(512, 512, kernel\_size=3, dilation=2, stride=1, padding=2, bias=True),]

        model6+=[nn.ReLU(True),]

        model6+=[norm\_layer(512),]

        # Conv7

        model7=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model7+=[nn.ReLU(True),]

        model7+=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model7+=[nn.ReLU(True),]

        model7+=[nn.Conv2d(512, 512, kernel\_size=3, stride=1, padding=1, bias=True),]

        model7+=[nn.ReLU(True),]

        model7+=[norm\_layer(512),]

        # Conv7

        model8up=[nn.ConvTranspose2d(512, 256, kernel\_size=4, stride=2, padding=1, bias=True)]

        model3short8=[nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model8=[nn.ReLU(True),]

        model8+=[nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model8+=[nn.ReLU(True),]

        model8+=[nn.Conv2d(256, 256, kernel\_size=3, stride=1, padding=1, bias=True),]

        model8+=[nn.ReLU(True),]

        model8+=[norm\_layer(256),]

        # Conv9

        model9up=[nn.ConvTranspose2d(256, 128, kernel\_size=4, stride=2, padding=1, bias=True),]

        model2short9=[nn.Conv2d(128, 128, kernel\_size=3, stride=1, padding=1, bias=True),]

        model9=[nn.ReLU(True),]

        model9+=[nn.Conv2d(128, 128, kernel\_size=3, stride=1, padding=1, bias=True),]

        model9+=[nn.ReLU(True),]

        model9+=[norm\_layer(128),]

        # Conv10

        model10up=[nn.ConvTranspose2d(128, 128, kernel\_size=4, stride=2, padding=1, bias=True),]

        model1short10=[nn.Conv2d(64, 128, kernel\_size=3, stride=1, padding=1, bias=True),]

        model10=[nn.ReLU(True),]

        model10+=[nn.Conv2d(128, 128, kernel\_size=3, dilation=1, stride=1, padding=1, bias=True),]

        model10+=[nn.LeakyReLU(negative\_slope=.2),]

        # classification output

        model\_class=[nn.Conv2d(256, classes, kernel\_size=1, padding=0, dilation=1, stride=1, bias=True),]

        # regression output

        model\_out=[nn.Conv2d(128, 2, kernel\_size=1, padding=0, dilation=1, stride=1, bias=True),]

        model\_out+=[nn.Tanh()]

        self.model1 = nn.Sequential(\*model1)

        self.model2 = nn.Sequential(\*model2)

        self.model3 = nn.Sequential(\*model3)

        self.model4 = nn.Sequential(\*model4)

        self.model5 = nn.Sequential(\*model5)

        self.model6 = nn.Sequential(\*model6)

        self.model7 = nn.Sequential(\*model7)

        self.model8up = nn.Sequential(\*model8up)

        self.model8 = nn.Sequential(\*model8)

        self.model9up = nn.Sequential(\*model9up)

        self.model9 = nn.Sequential(\*model9)

        self.model10up = nn.Sequential(\*model10up)

        self.model10 = nn.Sequential(\*model10)

        self.model3short8 = nn.Sequential(\*model3short8)

        self.model2short9 = nn.Sequential(\*model2short9)

        self.model1short10 = nn.Sequential(\*model1short10)

        self.model\_class = nn.Sequential(\*model\_class)

        self.model\_out = nn.Sequential(\*model\_out)

        self.upsample4 = nn.Sequential(\*[nn.Upsample(scale\_factor=4, mode='bilinear'),])

        self.softmax = nn.Sequential(\*[nn.Softmax(dim=1),])

    def forward(self, input\_A, input\_B=None, mask\_B=None):

        if(input\_B is None):

            input\_B = torch.cat((input\_A\*0, input\_A\*0), dim=1)

        if(mask\_B is None):

            mask\_B = input\_A\*0

        conv1\_2 = self.model1(torch.cat((self.normalize\_l(input\_A),self.normalize\_ab(input\_B),mask\_B),dim=1))

        conv2\_2 = self.model2(conv1\_2[:,:,::2,::2])

        conv3\_3 = self.model3(conv2\_2[:,:,::2,::2])

        conv4\_3 = self.model4(conv3\_3[:,:,::2,::2])

        conv5\_3 = self.model5(conv4\_3)

        conv6\_3 = self.model6(conv5\_3)

        conv7\_3 = self.model7(conv6\_3)

        conv8\_up = self.model8up(conv7\_3) + self.model3short8(conv3\_3)

        conv8\_3 = self.model8(conv8\_up)

        conv9\_up = self.model9up(conv8\_3) + self.model2short9(conv2\_2)

        conv9\_3 = self.model9(conv9\_up)

        conv10\_up = self.model10up(conv9\_3) + self.model1short10(conv1\_2)

        conv10\_2 = self.model10(conv10\_up)

        out\_reg = self.model\_out(conv10\_2)

        conv9\_up = self.model9up(conv8\_3) + self.model2short9(conv2\_2)

        conv9\_3 = self.model9(conv9\_up)

        conv10\_up = self.model10up(conv9\_3) + self.model1short10(conv1\_2)

        conv10\_2 = self.model10(conv10\_up)

        out\_reg = self.model\_out(conv10\_2)

        return self.unnormalize\_ab(out\_reg)

def siggraph17(pretrained=True):

    model = SIGGRAPHGenerator()

    if(pretrained):

        import torch.utils.model\_zoo as model\_zoo

        model.load\_state\_dict(model\_zoo.load\_url('https://colorizers.s3.us-east-2.amazonaws.com/siggraph17-df00044c.pth',map\_location='cpu',check\_hash=True))

    return model

def load\_img(img\_path):

    out\_np = np.asarray(Image.open(img\_path))

    if(out\_np.ndim==2):

        out\_np = np.tile(out\_np[:,:,None],3)

    return out\_np

def resize\_img(img, HW=(256,256), resample=3):

    return np.asarray(Image.fromarray(img).resize((HW[1],HW[0]), resample=resample))

def preprocess\_img(img\_rgb\_orig, HW=(256,256), resample=3):

    # return original size L and resized L as torch Tensors

    img\_rgb\_rs = resize\_img(img\_rgb\_orig, HW=HW, resample=resample)

    img\_lab\_orig = color.rgb2lab(img\_rgb\_orig)

    img\_lab\_rs = color.rgb2lab(img\_rgb\_rs)

    img\_l\_orig = img\_lab\_orig[:,:,0]

    img\_l\_rs = img\_lab\_rs[:,:,0]

    tens\_orig\_l = torch.Tensor(img\_l\_orig)[None,None,:,:]

    tens\_rs\_l = torch.Tensor(img\_l\_rs)[None,None,:,:]

    return (tens\_orig\_l, tens\_rs\_l)

def postprocess\_tens(tens\_orig\_l, out\_ab, mode='bilinear'):

    # tens\_orig\_l   1 x 1 x H\_orig x W\_orig

    # out\_ab        1 x 2 x H x W

    HW\_orig = tens\_orig\_l.shape[2:]

    HW = out\_ab.shape[2:]

    # call the resize function if needed

    if(HW\_orig[0]!=HW[0] or HW\_orig[1]!=HW[1]):

        out\_ab\_orig = F.interpolate(out\_ab, size=HW\_orig, mode='bilinear')

    else:

        out\_ab\_orig = out\_ab

    out\_lab\_orig = torch.cat((tens\_orig\_l, out\_ab\_orig), dim=1)

    return color.lab2rgb(out\_lab\_orig.data.cpu().numpy()[0,...].transpose((1,2,0)))

# Specify the path to the image

img\_path = '/content/drive/MyDrive/Mini-Project/Image Colourization/Images/place3.jpg'

use\_gpu = True  # or False if you don't want to use GPU

save\_prefix = 'saved'

img = load\_img(img\_path)

# load colorizer

colorizer\_eccv16 = eccv16(pretrained=True).eval()

colorizer\_siggraph17 = siggraph17(pretrained=True).eval()

if(use\_gpu):

    colorizer\_eccv16.cuda()

    colorizer\_siggraph17.cuda()

# default size to process images is 256x256

# grab L channel in both original ("orig") and resized ("rs") resolutions

img = load\_img(img\_path)

(tens\_l\_orig, tens\_l\_rs) = preprocess\_img(img, HW=(256,256))

if(use\_gpu):

    tens\_l\_rs = tens\_l\_rs.cuda()

# colorizer outputs 256x256 ab map

# resize and concatenate to the original L channel

img\_bw = postprocess\_tens(tens\_l\_orig, torch.cat((0\*tens\_l\_orig,0\*tens\_l\_orig),dim=1))

out\_img\_eccv16 = postprocess\_tens(tens\_l\_orig, colorizer\_eccv16(tens\_l\_rs).cpu())

out\_img\_siggraph17 = postprocess\_tens(tens\_l\_orig, colorizer\_siggraph17(tens\_l\_rs).cpu())

plt.imsave('%s\_eccv16.png'%save\_prefix, out\_img\_eccv16)

plt.imsave('%s\_siggraph17.png'%save\_prefix, out\_img\_siggraph17)

plt.figure(figsize=(12,8))

plt.subplot(2,2,1)

plt.imshow(img)

plt.title('Original')

plt.axis('off')

plt.subplot(2,2,2)

plt.imshow(img\_bw)

plt.title('Input')

plt.axis('off')

plt.subplot(2,2,3)

plt.imshow(out\_img\_eccv16)

plt.title('Output (ECCV 16)')

plt.axis('off')

plt.subplot(2,2,4)

plt.imshow(out\_img\_siggraph17)

plt.title('Output (SIGGRAPH 17)')

plt.axis('off')

plt.show()

plt.title('Output (SIGGRAPH 17)')

plt.axis('off')

plt.show()

**3.1 DATASETS DESCRIPTION**

The dataset consists of the test images that are passed to the models to process them and modify them to the required format for coloring them.

Images containing various types of objects and scenarios are present to use for the project.

**Images**

00000001.jpg

00000002.jpg

00000003.jpg

00000004,jpg

00000005.jpg

00000006.jpg

00000007.jpg

00000008.jpg

00000009.jpg

00000010.jpg

00000011.jpg

00000012.jpg

00000013.jpg

00000014.jpg

00000015.jpg

00000016.jpg

00000017.jpg

00000018.jpg

00000019.jpg

00000020.jpg

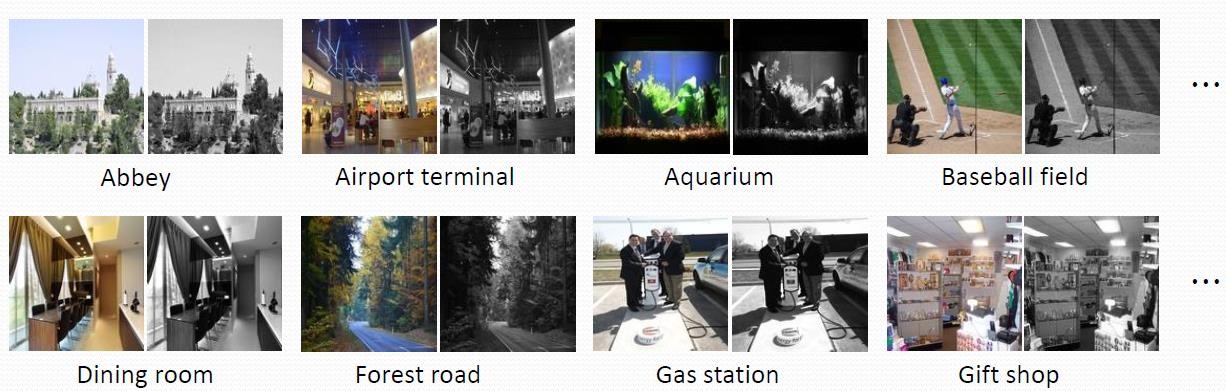
00000021.jpg

00000022.jpg

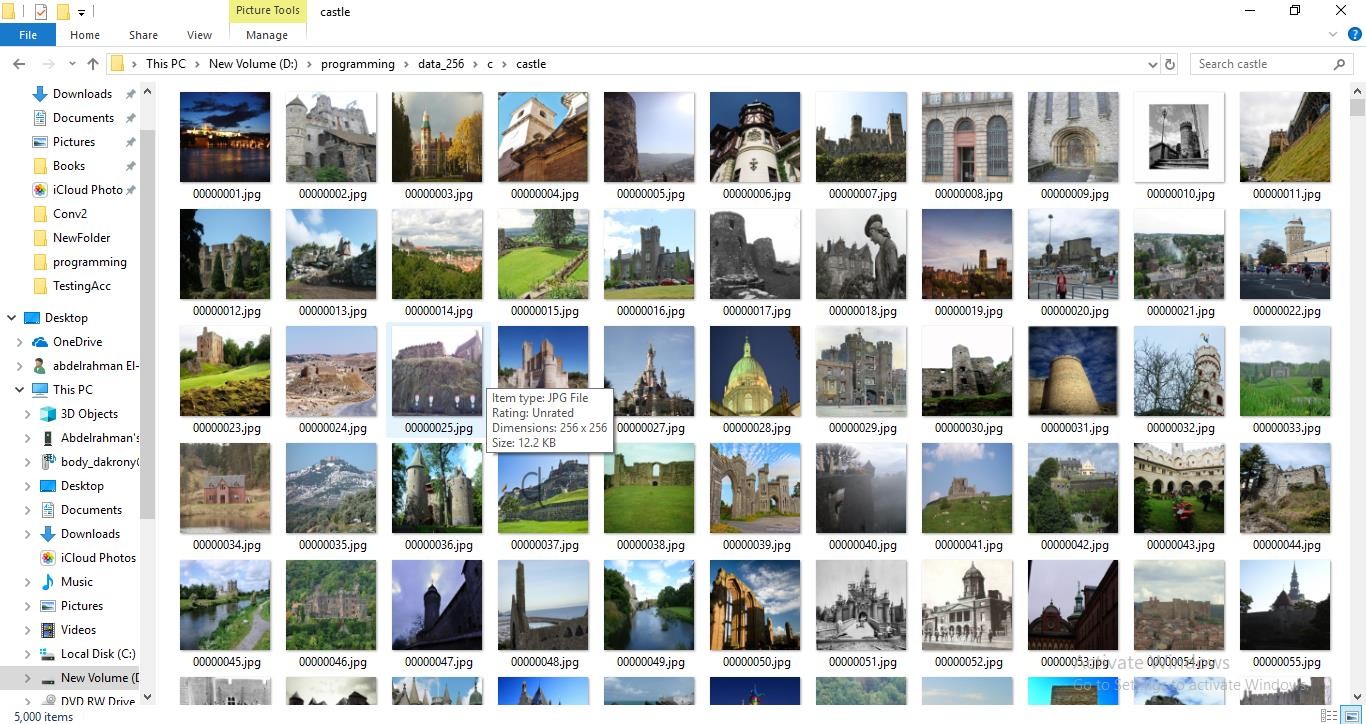
00000023.jpg

00000024.jpg

00000025.jpg



In the dataset, there are different images such as the image of roads, sea, gardens, towers, etc. But in our model from millions of datasets (Images), we have used only 15 thousand pictures



**3.2 DESCRIPTION OF TECHNOLOGIES/TOOLS USED:**

The image colorization project combines various tools and technologies to achieve its goal. Here's a comprehensive list of both the tools and technologies used in the project:

**Tools:**

1. **Python**: Python serves as the primary programming language for implementing the project, orchestrating various operations, and interfacing with libraries and frameworks.

2. **PyTorch**: PyTorch is a leading deep learning framework used for building, training, and deploying neural network models. It is instrumental in implementing image colorization.

3. **Torchvision**: A part of the PyTorch ecosystem, Torchvision provides access to computer vision utilities and pre-trained models. It simplifies the integration of pre-trained colorization models.

4. **NumPy**: NumPy is a fundamental library for numerical operations, aiding in handling and processing image data and performing mathematical and array operations.

5. **Pillow (PIL):** Pillow, also known as the Python Imaging Library (PIL), is an image processing library used for loading, resizing, and performing basic image operations.

6. **Matplotlib**: Matplotlib is a data visualization library employed for displaying and visualizing colorized images, enabling users to assess results visually.

7. **scikit-image**: Scikit-image is an image processing library that offers tools for image transformation and manipulation, enhancing image processing capabilities.

8. **Jupyter Notebook** (Optional): Jupyter Notebook is an interactive development environment that facilitates step-by-step execution and code visualization, aiding experimentation and code understanding.

9. **Google Colab (**Optional): Google Colab, a cloud-based Python environment with GPU support, is used to run the code in the cloud. It is particularly useful for remote execution and collaboration.

Technologies:

1. **Deep Learning:** Deep learning is the foundational technology for image colorization. It involves the use of deep neural networks to predict color information for grayscale images.

2**. Convolutional Neural Networks** (CNNs): CNNs are a specific type of deep neural network well-suited for image-related tasks. They are used for feature extraction and image colorization.

3. **Computer Vision: Computer** vision technologies are applied to handle image-related operations such as converting color spaces, image segmentation, and channel separation.

4. **Pre-trained Models**: Pre-trained models are deep neural networks that have been trained on large datasets and are essential for image colorization without the need for extensive training.

5. **Image Processing**: Image processing techniques are employed for tasks like image resizing, transformations, and the separation of image channels.

6. **GPU Acceleration: Graphics Processing Units** (GPUs) are utilized to accelerate complex computations involved in deep learning, making the colorization process faster and more efficient.

**3.3 MODULES DESCRIPTION**

Let's dive into the key modules and their details.

**Connecting to Google Drive**:

This section connects the Google Colab environment to Google Drive. It's useful for accessing data stored on Google Drive. The drive.mount method is called with the path to mount Google Drive.

**Importing Required Modules**:

This part imports essential Python libraries and deep learning frameworks required for the project.

Notable modules include torch and torch.nn for deep learning, PIL for image processing, numpy for numerical operations, and skimage.color for color space conversion.

**BaseColor Class**:

The BaseColor class defines methods for normalizing and unnormalizing color channels (L and AB channels) in the LAB color space.

These methods are used to preprocess and post-process the L channel in the colorization process.

**ECCVGenerator Class:**

The ECCVGenerator class is a deep neural network model for colorization. It inherits from the BaseColor class.

The model consists of multiple convolutional layers organized into sequential blocks, employing rectified linear unit (ReLU) activations.

Batch normalization is used to improve training stability.

The model architecture resembles an encoder-decoder structure, and it includes dilated convolutions for a wider receptive field.

The forward method defines the forward pass of the model, taking an input L channel and returning colorized AB channels.

**ECCV16 Function**:

The eccv16 function creates an instance of the ECCVGenerator model. It can load pre-trained weights if specified.

**SIGGRAPHGenerator Class:**

The SIGGRAPHGenerator class is another colorization model, similar to ECCVGenerator.

It includes additional convolutional layers and a classification output layer for a broader set of colors.

The forward method processes input images to generate colorized images.

**SIGGRAPH17 Function**:

The siggraph17 function creates an instance of the SIGGRAPHGenerator model and loads pre-trained weights if specified.

**Load Image Function**:

The load\_img function loads an image from a given file path and ensures that it has at least three color channels (RGB).

**Resize Image Function:**

The resize\_img function resizes an image to a specified size while maintaining the aspect ratio.

**Preprocess Image Function**:

The preprocess\_img function takes an RGB image, converts it to the LAB color space, and extracts the L channel.

It returns the original size L channel and a resized L channel as torch Tensors.

**Postprocess Tens Function:**

The postprocess\_tens function combines the original L channel with the colorized AB channels to create a full LAB image.

It then converts the LAB image back to RGB for visualization.

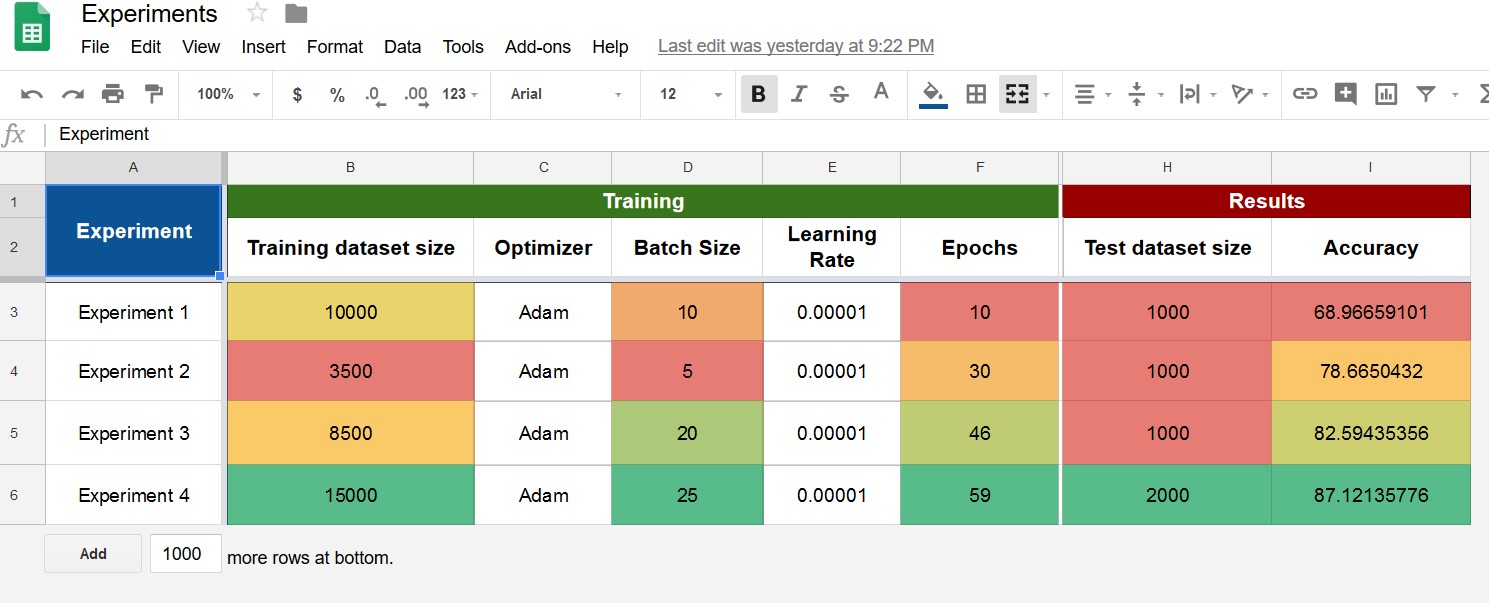
**Image Colorization and Visualization**:

The code demonstrates the colorization process by loading an image, pre-processing it, running it through the colorization models, and visualizing the original, input, and colorized images.

These modules collectively enable image colorization using deep learning models, with an option to use either the ECCV16 or SIGGRAPH17 architecture. The code supports loading pre-trained weights for these models, making the colorization process accessible and efficient**.**

**4. RESULTS AND DISCUSSIONS**

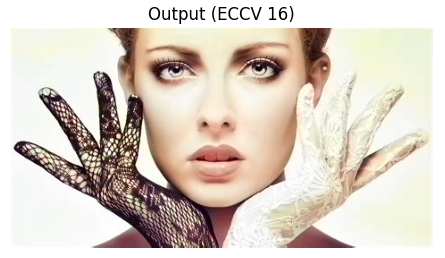
1. The Below chart shows the Model Training Parameter



2. This is the Grayscale Image which we are taking as the Input Image



3. After Colorizing the image with ECCV 16 we got the following image as Output

****

4. After Colorizing the image with SIGGRAPH 17, we get the following image as output.



**4.1 RESULTS OF ALGORITHM1:** ECCV 16



**4.2 RESULTS OF ALGORITHM2 :** SIGGRAPH 17



**5. CONCLUSION &** **FUTURE WORK**

In Conclusion, image colorization has become more accessible and convenient through various applications and tools. Historical images and media are converted into richly colorized images using this approach. This is the best approach to converting grayscale or black-and-white media to colorized images film colorization from old black-and-white films to richly enhanced and colorized films, and photo restoration from old black-and-white photos to richly enhanced and colorized photos.

The **Future scope** for image colorization holds potential in several areas such as, Advancements in deep learning can lead to more realistic and high-fidelity colorizations, improving visual quality, Real-time or interactive colorization tools that allow users to guide and customize the colorization process will likely become more prevalent. Continued research on preserving the context and cultural significance of colorized images, especially in historical and archival applications. Integration with AI-powered design and creative tools for artists and content creators. Application of colorization to medical imaging for improved diagnostics and visualization. Integration of colorization techniques into AR applications for real-time scene enhancement. Extending colorization techniques to other modalities, such as video and audio. Colorization for accessibility, assisting individuals with visual impairments to interpret images. Colorization for enhancing and analyzing visual evidence in forensic investigations. As technology evolves, image colorization will continue to find diverse applications across various domains, driven by advances in AI and deep learning. The future scope of image colorization is multifaceted, encompassing historical preservation, artistic expression, accessibility, and a wide range of practical applications across industries. As technology and AI continue to advance, image colorization will play an increasingly significant role in how we interact with and derive value from visual content.

**REFERENCES**

1. Zhang, Richard, et al. "Colorful Image Colorization." In Proceedings of the European Conference on Computer Vision (ECCV), 2016. Link
2. Iizuka, Satoshi, Edgar Simo-Serra, and Hiroshi Ishikawa. "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification." ACM Transactions on Graphics (TOG), 2016. Link

1. Zhang, Richard, Phillip Isola, and Alexei A. Efros. "Colorful Image Colorization." In European Conference on Computer Vision (ECCV), 2016. Link
2. Iizuka, Satoshi, Edgar Simo-Serra, and Hiroshi Ishikawa. "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification." ACM Transactions on Graphics (TOG), 2016. Link
3. Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Learning Representations for Automatic Colorization." At European Conference on Computer

Vision (ECCV), 2016. Link

1. Zhang, Richard, et al. "Real-Time User-Guided Image Colorization with Learned

Deep Priors." ACM Transactions on Graphics (TOG), 2017. Link

1. He, Mingming, et al. "Deep Exemplar-Based Colorization." ACM Transactions on

Graphics (TOG), 2018. Link

1. Larsson, Gustav, et al. "Pix2pix: Image-to-Image Translation with Conditional Adversarial Networks." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017. Link
2. Wang, Zhiqin, Jianlong Fu, and Hanqing Lu. "Deep Colorization with Improved Structural Consistency." In Proceedings of the European Conference on Computer

Vision (ECCV), 2018. Link

1. Isola, Phillip, et al. "Image-to-Image Translation with Conditional Adversarial Networks." In Proceedings of the IEEE Conference on Computer Vision and Pattern

Recognition (CVPR), 2017. Link