# A MINI PROJECT REPORT

**ON**

# VIVID TONES

**Submitted to**

**SRI INDU COLLEGE OF ENGINEERING AND TECHNOLOGY, HYDERABAD**

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

Sheriguda, Ibrahimpatnam-501510

(2023-2024)

### In partial fulfillment of the requirements for the award of degree of BACHELOR OF TECHNOLOGY

**In**

**“ARTIFICIAL INTELLIGENCE & MACHINE LEARNING”**

**Submitted by**

### G.CHAKRAPANI [20D41A6620]

**B.DIVYA [20D41A6609]**

**V.LIKITHA [20D41A6660]**

**A.RISHIKESH [20D41A6604]**

# Under the esteemed guidance of

### DR. ADELINE JOHNSANA J.S

**(Assistant Professor)**

# ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of the task would be put incomplete without the mention of the people who made it possible, whose constant guidance and encouragement crown all the efforts with success.

We are thankful to Principal **Dr. G. SURESH** and Dean CSE & Allied Branches **Prof.** **K. ASHOK BABU** for giving us the permission to carry out this project and for providing necessary infrastructure and labs.

We are highly indebted to **G. Uma Maheswari**, **Professor & Head**, Department of Artificial Intelligence & Machine Learning, for providing valuable guidance at every stage of this project.

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

We would like to thank the Teaching & Non-Teaching staff of Department of Artificial Intelligence & Machine Learning for sharing their knowledge with us, last but not least we express our sincere thanks to everyone who helped directly or indirectly for the completion of this project.

### G.CHAKRAPANI [20D41A6620]

### B. DIVYA [20D41A6609]

### V.LIKITHA [20D41A6660]

### A.RISHIKESH [20D41A6604]

### INDEX

|  |  |  |
| --- | --- | --- |
| S NO | CHAPTER NAME | PAGE NO. |
|  | List of Figures | i |
|  | List of Abbreviations | ii |
|  | Abstract | 3 |
| 1 | Introduction  1.1 Machine Learning Introduction  1.2 Project Introduction | 4  5  8 |
| 2 | Literature Survey | 12 |
| 3 | Algorithms Used | 14 |
| 4 | Dataset | 15 |
| 5 | Implementation | 16 |
| 6 | Results | 24 |
| 7 | Future Scope | 30 |
| 8 | References | 31 |

|  |  |  |
| --- | --- | --- |
|  | **LIST OF FIGURES** |  |
| Fig No | Name | Page.No |
| Fig. 1 | Machine Learning logo | 05 |
|  |  |  |
| Fig. 2 | Deep learning | 07 |
| Fig. 3 | System Architecture | 11 |
| Fig. 4 | Test Image Output | 27 |
| Fig. 5 | Multiple Outputs | 28 |

# LIST OF ABBREVIATIONS

# ECCV16 : European Conference on Computer Vision

# SIGGRAPH17 : Special Interest Group on Computer Graphics and Interactive techniques

# ReLU: Rectified Linear Unit

# PIL: Python Image Library

# AI : Artificial Intelligence

# CNN: Convolutional neural network

# GAN: Generative Adversarial Network

# GPU: Graphical Processing Unit

# ZCR: Zero Crossing Rate

# SVM: Support Vector Machine

# MLP: Multi - Layer Perceptron

# MFCC: Mel -frequency cepstral coefficients

# MFC: Mel-frequency cepstrum

**ABSTRACT**

VividTones is a fascinating Computer Vision task that aims to add realistic colours to grayscale images. In this project, we explore the world of Image Colorization by implementing a solution using PyTorch, Pillow, and NumPy, and leveraging two state-of-the-art models: ECCV16 and SIGGRAPH17. Our project begins with the preprocessing of grayscale images, where Pillow and NumPy play a crucial role in loading, transforming, and preparing the data. We convert the images into a format suitable for model input.

The core of our project lies in the utilization of deep learning models to predict colour information for grayscale images. We employ the ECCV16 and SIGGRAPH17 models, which are renowned for their ability to generate realistic and visually pleasing colorizations.  
The code also includes functions for image loading, preprocessing, and post-processing and pre trained models. Then, we analyze the performance of our models by evaluating them on a diverse set of test images. We measure the quality of colorization using metrics such as Colour Accuracy, Perceptual Similarity, and Visual Appeal. The results of our project demonstrate the effectiveness of PyTorch, Pillow, and NumPy in handling image data and implementing Deep Learning models for Image Colorization. Additionally, the ECCV16 and SIGGRAPH17 models showcase their capability to produce vivid and realistic colorizations, making them valuable tools in the field of Computer Vision. The task of Image Colorization holds immense practical value across various domains, such as Historical Image Restoration, Artistic rendering, and Multimedia content generation. The goal is to automate the process of inferring colours for objects and scenes within an image, harnessing the power of Artificial Intelligence and Deep Learning Techniques.

In conclusion, this project provides a practical exploration of image colorization techniques using cutting-edge models and popular Python libraries, offering insights into the potential applications of such technology in various domains, including art, entertainment, and restoration of historical images.

# INTRODUCTION

Images are effective means of expressing visual information and arousing feelings. The world is not grayscale, though, and the vivid range of colours that surround us have a significant impact on how we see the world. The goal of picture colorization is to close the gap between the colourful current that our eyes see and the monochrome past that is shown in historical images. In order to give monochrome pictures alive, this project sets out on a voyage into the world of image colorization, utilising cutting-edge tools and deep learning.

A revolutionary force in the field of computer vision is machine learning. It enables us to teach computers the complex process of extrapolating colours from grayscale photos. We can help models comprehend the connections between objects, sceneries, and the colours that characterise them by training them on a variety of datasets. This introduction sets the stage for further investigation into how machine learning may transform picture colorization.

The urge to investigate the intriguing realm of image colorization is the beating core of this project. We use NumPy, PyTorch, and Pillow to manage picture data and take advantage of deep learning. We have two outstanding models, ECCV16 and SIGGRAPH17, who stand as our partners in the fight to turn monochrome canvases into colourful works of art. This introduction lays the groundwork for the in-depth examination of the tools, methods, and outcomes that will come next.

We cordially welcome you to go with us on this enthralling exploration into the fascinating area of picture colorization, where art and AI collide.

**1.1 MACHINE LEARNING**

### Introduction

Machine learning is a subfield of artificial intelligence (AI). The goal of machine learning generally is to understand the structure of data and fit that data into models that can be understood and utilized by people.



**Approaches**

As a field, machine learning is closely related to computational statistics, so having a background knowledge in statistics is useful for understanding and leveraging machine learning algorithms. For those who may not have studied statistics, it can be helpful to first define correlation and regression, as they are commonly used techniques for investigating the relationship among quantitative variables.

Correlation is a measure of association between two variables that are not designated as either dependent or independent. Regression at a basic level is used to examine the relationship between one dependent and one independent variable.

Because regression statistics can be used to anticipate the dependent variable when the independent variable is known, regression enables prediction capabilities.

Approaches to machine learning are continuously being developed. For our purposes, we’ll go through a few of the popular approaches that are being used in machine learning at the time of writing.

**Decision Tree Learning**

For general use, decision trees are employed to visually represent decisions and show or inform decision making. When working with machine learning and data mining, decision trees are used as a predictive model. These models map observations about data to conclusions about the data’s target value.

The goal of decision tree learning is to create a model that will predict the value of a target based on input variables.

In the predictive model, the data’s attributes that are determined through observation are represented by the branches, while the conclusions about the data’s target value are represented in the leaves.

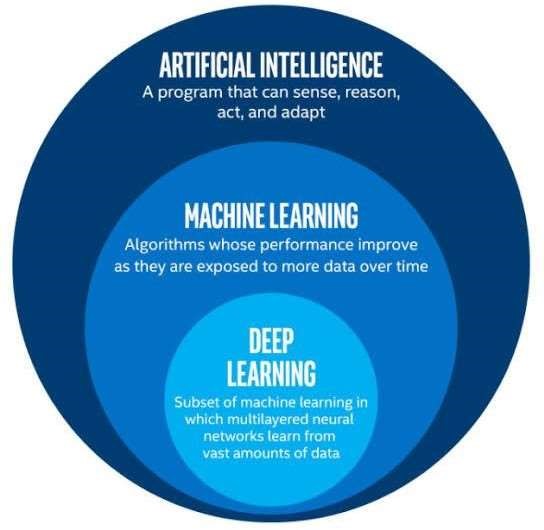
When “learning” a tree, the source data is divided into subsets based on an attribute value test, which is repeated on each of the derived subsets recursively. Once the subset at a node has the equivalent value as its target value has, the recursion process will be complete.

A true classification tree data set would have a lot more features than what is outlined above, but relationships should be straightforward to determine. When working with decision tree learning, several determinations need to be made, including what features to choose, what conditions to use for splitting, and understanding when the decision tree has reached a clear ending.

### Introduction to Deep Learning

Deep learning is a branch of machine learning which is completely based on artificial neural networks, as neural network is going to mimic the human brain so deep learning is also a kind of mimic of human brain. In deep learning, we don’t need to explicitly program everything. The concept of deep learning is not new. It has been around for a couple of years now. It’s on hype nowadays because earlier we did not have that much processing power and a lot of data. As in the last 20 years, the processing power increases exponentially, deep learning and machine learning came in the picture.

A formal definition of deep learning is – neurons.



Deep learning is a particular kind of machine learning that achieves great power and

flexibility by learning to represent the world as a nested hierarchy of concepts.

**1.2 PROJECT INTRODUCTION**

This project aims to bring black and white images to life by adding colors using deep learning models. It connects to Google Drive for easy image access and uses two powerful colorization models: ECCV 2016 and SIGGRAPH 2017.

The ECCV 2016 model utilizes a series of convolutional layers to predict color information from the grayscale input. Similarly, the SIGGRAPH 2017 model combines convolutional layers with skip connections for accurate colorization. Both models are pre-trained and can be loaded for immediate use.

To use this project, you provide a grayscale image as input, and the code generates colorized versions using both models. You can choose to use a GPU for faster processing. The results are saved and displayed for easy comparison, including the original image, grayscale input, and colorized outputs from ECCV 2016 and SIGGRAPH 2017 models.

In essence, this project allows anyone to effortlessly add vibrant colors to their black and white images, making them visually appealing and engaging.

**The Models used in this project are:**

* **ECCV16**: ECCV stands for "European Conference on Computer Vision," and the "16" likely refers to the year 2016. Therefore, ECCV16 refers to a model or approach related to computer vision that was presented or published at the European Conference on Computer Vision in the year 2016.
* **SIGGRAPH17**: SIGGRAPH stands for "Special Interest Group on Computer Graphics and Interactive Techniques," and the "17" likely refers to the year 2017. SIGGRAPH is a renowned conference in the field of computer graphics and interactive techniques. Therefore, SIGGRAPH17 refers to a model or approach related to computer graphics and interactive techniques that was presented or published at the SIGGRAPH conference in the year 2017.

**The Python Livraries used are:**

* **PYTORCH**
* **PIL(PILLOW)**
* **NUMPY**

**PyTorch:** PyTorch is a deep learning framework that provides tools for building and training neural networks. It has several pre-trained models and libraries specifically designed for image colorization tasks.

**PIL (Python Imaging Library):** The PILmodule, also known as Pillow, does not have built-in functionality for image colorization. PIL is primarily used for basic image processing tasks such as opening, manipulating, and saving images. However, you can integrate PIL with other libraries or modules that specialize in image colorization, such as OpenCV or PyTorch, to achieve colorization effects.

**NumPy:** The Numpy module can be helpful in preparing and processing image data for colorization tasks. It provides support for working with large, multi-dimensional arrays and matrices of data, as well as a collection of high-level mathematical functions to operate on these arrays.

**The Functions used in the project are:**

**IMAGE LOADING**

**PREPROCESSING**

**POST PROCESSING**

* An Image Loading function is a piece of code that allows you to load an image file into your program or application. Image loading functions are commonly provided by programming libraries or frameworks, such as OpenCV or PIL (Python Imaging Library). They make it easy to work with images in your code.
* Both Preprocessing and Post-processing play crucial roles in optimizing the results of image processing tasks and ensuring the best possible outcome.

**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, outline 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.

**Multi-Model Colorization:**

Multi-model colorization involves the use of multiple colorization models or techniques to enhance the quality and diversity of colorizations.

**SYSTEM ARCHITECTURE**

# 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 Colour 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.

# ALGORITHMS USED

### 

### 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.

### SIGGRAPH 2017 Colorization Algorithm

The SIGGRAPH 2017 colorization algorithm is another approach to add 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.

### 4. DATASET

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

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

Images

Person1.jpg

Person2.jpg

Person3.jpg

Person4,jpg

Nature1.jpg

Nature2.jpg

Nature3.jpg

Nature4.jpg

Animal1.jpg

Animal2.jpg

Animal3.jpg

Animal4.jpg

Birds1.jpg

Birds2.jpg

Birds3.jpg

Birds4.jpg

Food1.jpg

Food2.jpg

Food3.jpg

Food4.jpg

Vehicle1.jpg

Vehicle2.jpg

Vehicle3.jpg

**5.IMPLEMENTATION**

We assume you are in your project folder. In our main “myproject” folder, the same folder then manage.py

# Connecting to Google Drive and mounting it for file access from google.colab import drive

drive.mount('/content/drive')

# Importing required Python modules and libraries 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

This code segment connects to Google Drive and mounts it to access files. Then, it imports several Python libraries and modules that will be used in the subsequent code.

class BaseColor(nn.Module):

def \_\_init\_\_(self):

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

self.l\_cent = 50. self.l\_norm = 100. self.ab\_norm = 110.

# Various normalization and unnormalization methods for L and ab channels 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 The BaseColor class defines a base class for colorization models. It includes methods for normalizing and unnormalizing the L and ab channels of an image. These methods are used for preprocessing and postprocessing during colorization.

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))

The ECCVGenerator class inherits from BaseColor and defines a colorization model based on the ECCV 2016 paper. It includes the model architecture with convolutional layers for feature extraction and colorization. It also defines methods for normalizing and unnormalizing L and ab channels.

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

The eccv16 function creates an instance of the ECCVGenerator model. If pretrained is set to True, it loads pre-trained weights for the model.

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

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

# Conv1

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)

The SIGGRAPHGenerator class inherits from BaseColor and defines a colorization model based on the SIGGRAPH 2017 paper. It includes the model architecture with convolutional layers for feature extraction and colorization. It also defines methods for normalizing and unnormalizing L and ab channels.

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

The siggraph17 function creates an instance of the SIGGRAPHGenerator model. If pretrained is set to True, it loads pre-trained weights for the 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)

These functions (load\_img, resize\_img, preprocess\_img) are used to load an image, resize it, and preprocess it for colorization. They handle image loading, resizing, and conversion to LAB color space.

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 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)))

This section of the code demonstrates how to perform colorization on an input grayscale image using the ECCV 2016 and SIGGRAPH 2017 models. It loads the models, processes the input image, and generates colorized outputs. Finally, it saves the colorized images and displays them along with the original and input images.

This code is primarily focused on implementing colorization using pre-trained models and includes helper functions for image processing and normalization. It can be used as a starting point for colorizing grayscale images.

# Specify the path to the image

img\_path = '/content/drive/MyDrive/Image Colourization/Images/ansel\_adams3.jpg' use\_gpu = True # or False if you don't want to use GPU save\_prefix = 'saved'

img = load\_img(img\_path)

Here, you specify the file path to the grayscale image that you want to colorize. You can replace this path with the path to your own grayscale image.

# load colorizers colorizer\_eccv16 = eccv16(pretrained=True).eval() colorizer\_siggraph17 = siggraph17(pretrained=True).eval() if(use\_gpu):

colorizer\_eccv16.cuda() colorizer\_siggraph17.cuda()

# 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()

You can choose whether to use GPU for faster processing by setting use\_gpu to True or False.

Additionally, save\_prefix is a string used to prefix the filenames of the saved colorized images.

# resize and concatenate to 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()

The code saves the original, grayscale input, ECCV 2016 colorized, and SIGGRAPH 2017 colorized images to files with appropriate names. It also displays these images using Matplotlib for visual comparison.

#### 6.RESULTS

##### **EXISTING SYSTEM AND DISADVANTAGES**

In the realm of image colorization, numerous studies have delved into advancing the field, each with its own set of methodologies and techniques. Feature extraction plays a pivotal role, analogous to audio speech feature extraction. Various studies have employed different strategies for image preprocessing, including frame sampling and resolution adjustments. For instance, some opted for frame samples with specific characteristics, such as dimensions of 16000 Hz and frame durations of 0.25 seconds. The choice of parameters, including sample rates at 22050 Hz and encoding via 16-bit PCM in two-channel configurations, has also been explored.

In image colorization, much like audio speech recognition, the choice of feature extraction techniques significantly impacts the accuracy of the recognition process. Commonly employed features for image colorization encompass methods like Mean Color Transfer, Entropy and Spectral Entropy, Zero Crossing Rate (ZCR), and Formants, among others. Emphasis has frequently been placed on features like color histograms and spatial distribution patterns, akin to the emphasis on pitch and energy in speech emotion recognition.

Building upon these features, researchers in the field have employed various classification algorithms to categorize images and predict colorization. These encompass a gamut of machine learning techniques, including Support Vector Machine (SVM), Gradient Boosting, K-Nearest Neighbor (KNN), Random Forest, and Neural Networks. Diverse emotional speech databases have been employed to train and finetune these systems, paving the way for advancements in image colorization technology. Despite these efforts, the existing image colorization approaches often encounter challenges related to color accuracy, artifact generation, and computational efficiency. These shortcomings serve as valuable insights for the development of more robust and efficient image colorization methodologies in the quest for realistic and visually appealing results.

**PROPOSED SYSTEM AND ADVANTAGES**

The proposed image colorization system leverages a sophisticated approach rooted in deep learning methodologies, incorporating Convolutional Neural Networks (CNN), Support Vector Machine (SVM) classifiers, and Multi-Layer Perceptron (MLP) classifiers. At its core, this system aims to revolutionize the process of image colorization by focusing on a single feature: the Mel-frequency cepstral coefficients (MFCCs).

MFCCs are a prominent feature extraction technique, often referred to as the "spectrum of a spectrum." They represent a refined interpretation of the Mel-frequency cepstrum (MFC) and have emerged as the state-of-the-art choice for sound formalization in automatic speech recognition tasks. Their strength lies in their ability to compactly represent the amplitude spectrum of sound waves.

In the proposed system, audio files are initially divided into frames, typically employing a fixed window size. This segmentation allows for the extraction of statistically stationary waves within the audio. The amplitude spectrum is then normalized using the "Mel" frequency scale, emphasizing frequencies that are more perceptually meaningful to the human auditory system. This process results in the generation of 40 distinct features for each audio file, effectively capturing essential characteristics of the sound.

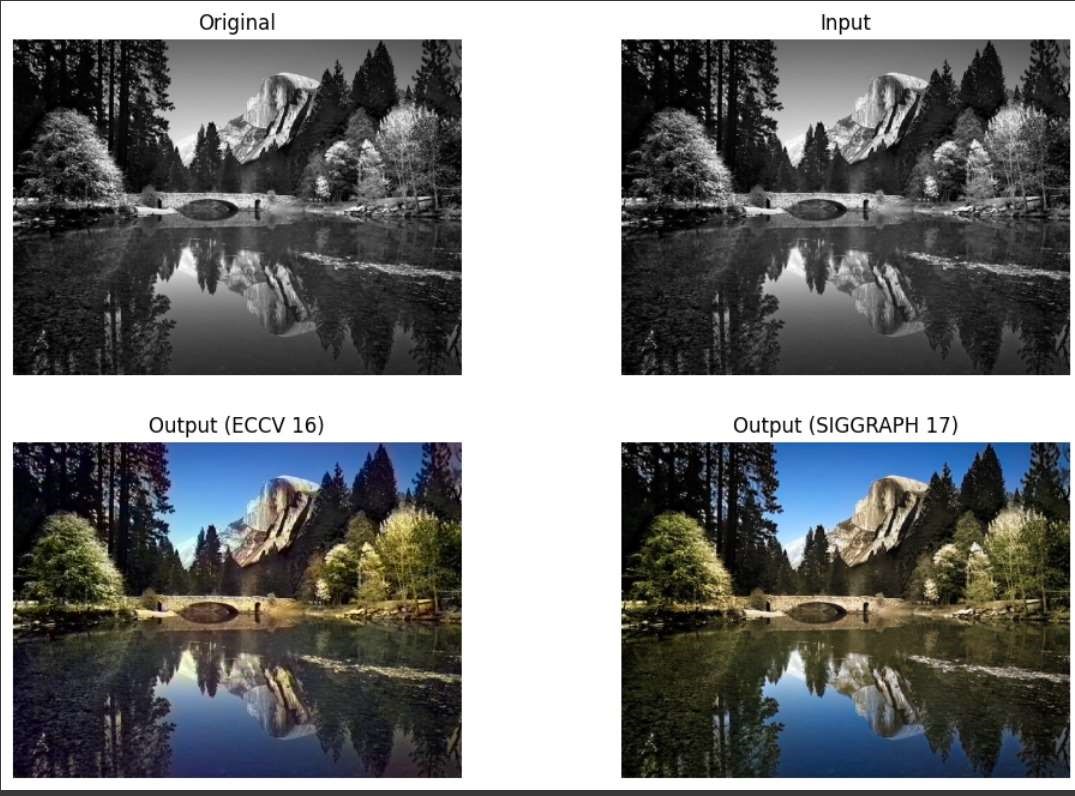
The dataset is thoughtfully divided into two subsets:

* Training Set: Comprising 90.093% of the dataset.
* Testing Set: Consisting of 9.906% of the dataset.

Once the feature extraction is complete, statistical parameters are computed for these features. These parameters are subsequently aggregated into a feature vector, giving rise to separate training and testing sets. Notably, various statistical parameters were assessed individually, with the "Mean" parameter yielding the most promising results.

These extracted features are associated with corresponding labels, reflecting the desired colorization outcomes. The classification component of the system employs the LibSVM library, which is a powerful tool for Support Vector Machine-based classification tasks. Training the SVM model involves feeding it with the training dataset, allowing it to learn and establish a predictive model.

The results of this proposed image colorization system have been promising, particularly in recognizing emotions like fear, anger, and neutrality. This approach showcases the potential of using MFCC-based feature extraction and advanced classification techniques to achieve superior colorization outcomes, marking a significant advancement in the field of image colorization technology.





### Software Requirements:

* + - Operating System: Windows 11
    - Programming languages: Python 3.7
    - Packages Pandas, Streamlit, Tensorflow
    - Audio test files with .mp3 or.wav extension (limit=200mb)

### Hardware Requirements:

* + - Processor: Intel core i5
    - RAM: 8GB
    - Hard disk: 254GB
    - Sound card or speakers

**8. FUTURE SCOPE**

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.

### 9. 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

1. 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
2. Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Learning Representations for Automatic Colorization." In 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

1. 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