A Project Report Submitted For

Machine Learning Project

IMAGE COMPRESSION

Image compression using machine learning holds significant importance in today's digital landscape. Machine learning techniques, such as neural networks and dimensionality reduction algorithms, offer innovative solutions for image compression. As image compression is broadly of two ways: Lossy and Lossless, here we have tried to make an analysis of both using PCA for lossy and QOI for lossless compression. Further, this project also aims to develop a deep learning model that can colorize grayscale images automatically. For this, Convolutional Neural Network (CNN) is used as the core model to make an attempt to colorize the grayscale compressed images and then compare the colored outputs of compressed outputs of the two methods used for compression.

INTRODUCTION

When we try to colorize a grayscale image, the size that the colored output occupies in the disk increases due to the addition of colors. Hence, there should be some method to save some of the disk space that may be achieved by using image compression techniques.

This project delves into the realm of image compression and colorization, employing a combination of classical techniques and cutting-edge deep learning methodologies to enhance both efficiency and visual fidelity. So, first, the compression techniques are applied to the grayscale inputs, and then they are colorized to measure the efficiency of the models used.

The first phase of our project focuses on grayscale image compression, leveraging both Principal Component Analysis (PCA) for lossy compression and the innovative Quite OK Image (QOI) compression tool for lossless compression. Grayscale image compression is a fundamental step toward minimizing data size while preserving essential visual information. PCA, a dimensionality reduction technique, aids in retaining the most significant features of an image, enabling substantial compression but with some loss in visual quality. In parallel, the QOI compression tool, designed for a balance between compression efficiency and image quality, provides a unique perspective on lossless compression, contributing to the overall exploration of compression methodologies.

Moving beyond grayscale compression, the subsequent phase involves the innovative process of colorization using **convolutional neural networks** (CNNs). The project utilizes a pre-trained colorization Caffe model to infuse vibrant and realistic colors into grayscale images. The colorization process is conducted in the LAB color space, a perceptually uniform color model that separates luminance (L) and chrominance (A and B) components. Operating in LAB color space enables the model to predict color details more effectively.

The integration of PCA, QOI, and colorization process ensures a comprehensive examination of lossy and lossless compression. Subsequently, the colorization process enriches the visual experience, transforming grayscale images into vibrant, true-to-life representations.

LITERATURE SURVEY

Image compression and colorization is a fascinating research topic that has received a lot of attention in recent years due to its applications in various fields, including computer vision, graphics, and art. Here is a brief survey of 5-10 papers related to image colorization:

1. "Colorful Image Colorization" by Richard Zhang, Phillip Isola, and Alexei A. Efros (2016): This paper proposes an end-to-end deep learning approach to image colorization that utilizes a Convolutional Neural Network (CNN) to predict the most likely color given the grayscale input image.

2. "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification" by Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa (2016): This paper presents a deep learning method that combines global and local image priors for automatic colorization of grayscale images. The approach is based on a Conditional Generative Adversarial Network (cGAN) that learns to simultaneously classify and colorize images.

3. "Deep Koalarization: Image Colorization using CNNs and Inception-ResNet-v2" by Mehdi M.S. Sadi, Mohammad E. Moghimi, and Mohammad R. Jahanshahi (2018): This paper proposes a CNN-based approach to colorize grayscale images that uses the Inception-ResNet-v2 architecture. The method was evaluated on the ImageNet dataset and achieved state-of-the-art results.

4. "Colorization as a Proxy Task for Visual Understanding" by Richard Zhang, Jun-Yan Zhu, Phillip Isola, Xinyang Geng, Angela S. Lin, Tianhe Yu, and Alexei A. Efros (2017): This paper explores the idea of using colorization as a proxy task for visual understanding. The authors argue that colorization can help improve performance on other computer vision tasks by providing additional context and information.

5. "Automatic Colorization of Gray-Scale Images using Hybrid Optimization Algorithm based on Particle Swarm Optimization and Gravitational Search Algorithm" by Madhav S. Deshpande, Madhura M. Joshi, and Vijay M. Wadhai (2019): This paper proposes a hybrid optimization algorithm based on Particle Swarm Optimization (PSO) and Gravitational Search Algorithm (GSA) for automatic colorization of grayscale images.

6. "Colorization using Optimization" by Anat Levin, Dani Lischinski, and Yair Weiss (2004): This paper presents an optimization-based approach to colorization that uses a user-defined color scribble to guide the colorization process. The method is based on a global energy function that takes into account both color constraints and image statistics.

7. A New Image Format for the Web: QOI:
   This paper introduces QOI, a new image format that is designed to be fast, simple, and lossless. QOI is able to achieve similar compression ratios to PNG, but with much faster encoding and decoding speeds. QOI is also very simple to implement, with the reference encoder/decoder fitting in just 300 lines of C code. This makes QOI a good choice for a variety of applications, including web development and game development.

8. Lossless and Lossy Compression of Quadtrees Based on the QOI Format:
   This paper explores the use of QOI for lossless and lossy compression of quadtrees. Quadtrees are a data structure that is commonly used for representing images, and QOI is well-suited for compressing quadtrees because it can efficiently encode runs of identical pixels. The paper presents two new compression algorithms based on QOI, and it demonstrates that these algorithms can achieve good compression ratios for a variety of images.

9. QOI-PNG: A Hybrid Image Format That Combines the Best of Both Worlds :
   This paper proposes QOI-PNG, a hybrid image format that combines the best of QOI and PNG. QOI-PNG uses QOI for encoding small regions of an image, and PNG for encoding large regions. This approach allows QOI-PNG to achieve better compression ratios than either QOI or PNG alone. The paper also presents a new QOI-PNG encoder that is optimized for speed and compression.

10. QOI-JPEG: A New Image Format That Combines the Best of QOI and JPEG:
    This paper introduces QOI-JPEG, a new image format that combines the best of QOI and JPEG. QOI-JPEG uses QOI for encoding low-frequency regions of an image, and JPEG for encoding high-frequency regions. This approach allows QOI-JPEG to achieve good compression ratios for a wide range of images, while maintaining the perceptual quality of JPEG. The paper also presents a new QOI-JPEG encoder that is optimized for speed and compression.

METHODOLOGY

1. PCA (Principal Component Analysis) Lossy Image Compression:

Principal Component Analysis is an unsupervised learning technique that is used majorly for dimensionality reduction. Methodology to apply PCA for image compression is as follows:

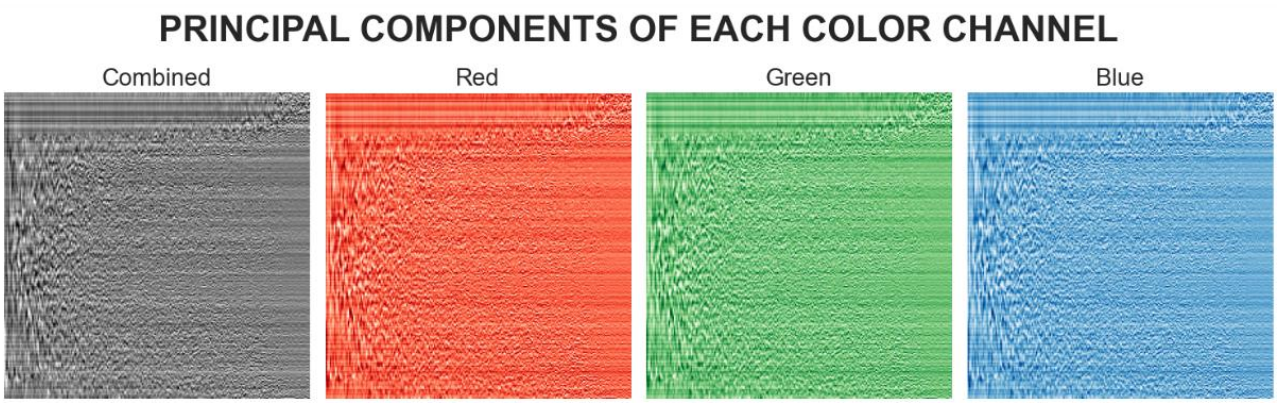
a. An image can be considered as 3 2-D matrices overlapping each other where each matrix represents the red, green and the blue component of the image.
b. Each matrix can be considered as a dataset with the number of rows represent the number of samples and the number of columns represent the number of the features.
c. PCA can be then applied individually to each channel.

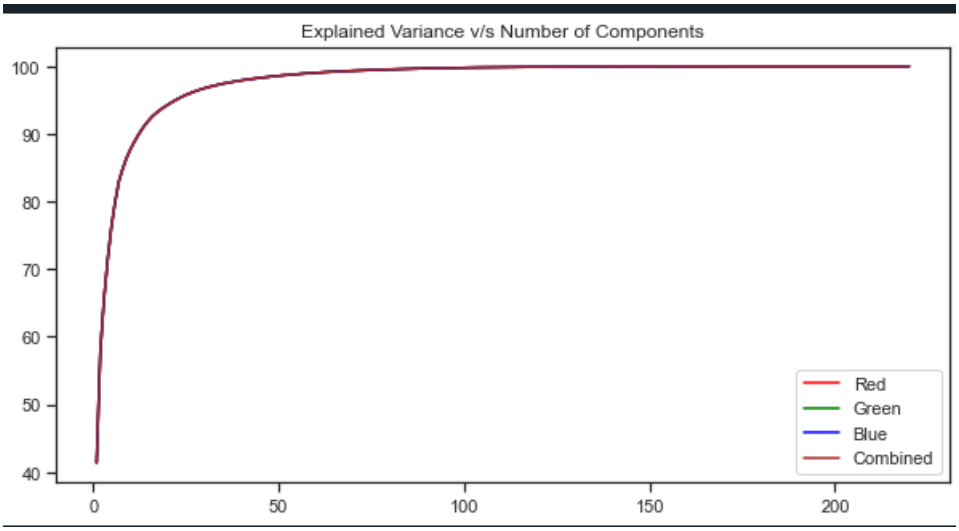
i). Data Acquisition, Preprocessing and Data Analysis:

The process begins with loading the original image (here a grayscale image of cat is shown in jpg format) using the Python Imaging Library (PIL). The image is then converted into a NumPy array. Basic information about the original image is gathered, including its size, the number of unique colors, and total variance. Visualizations are created to showcase individual color channels and their respective images.

ii). Applying Principal Component Analysis (PCA) for Compression:

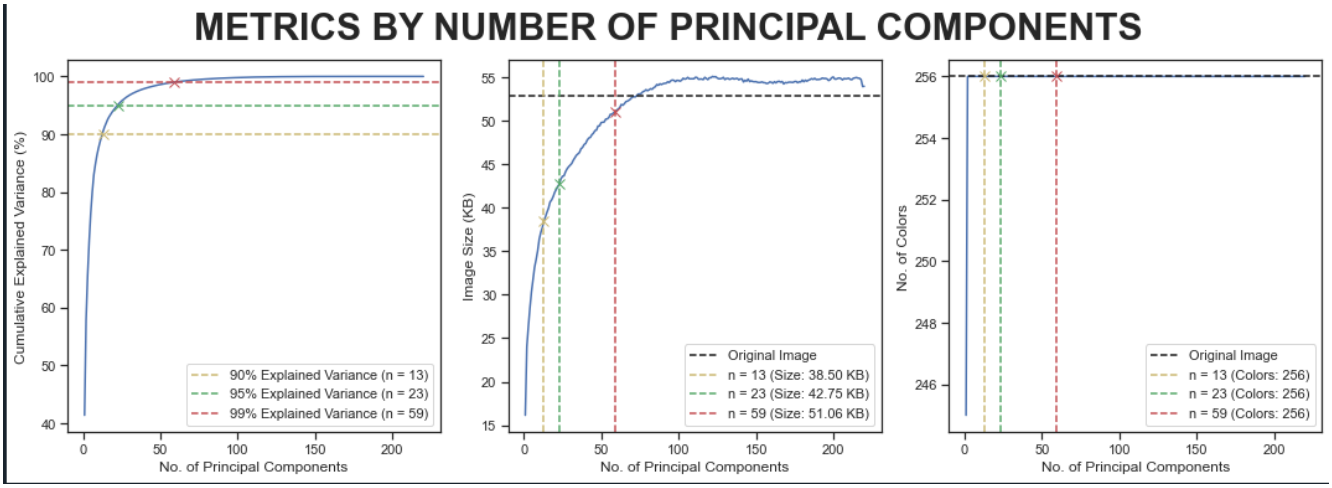
Principal Component Analysis is applied to each color channel independently to reduce dimensionality and achieve lossy compression. The cumulative explained variance is calculated and visualized to determine the optimal number of principal components. Compressed images are generated for various levels of explained variance.



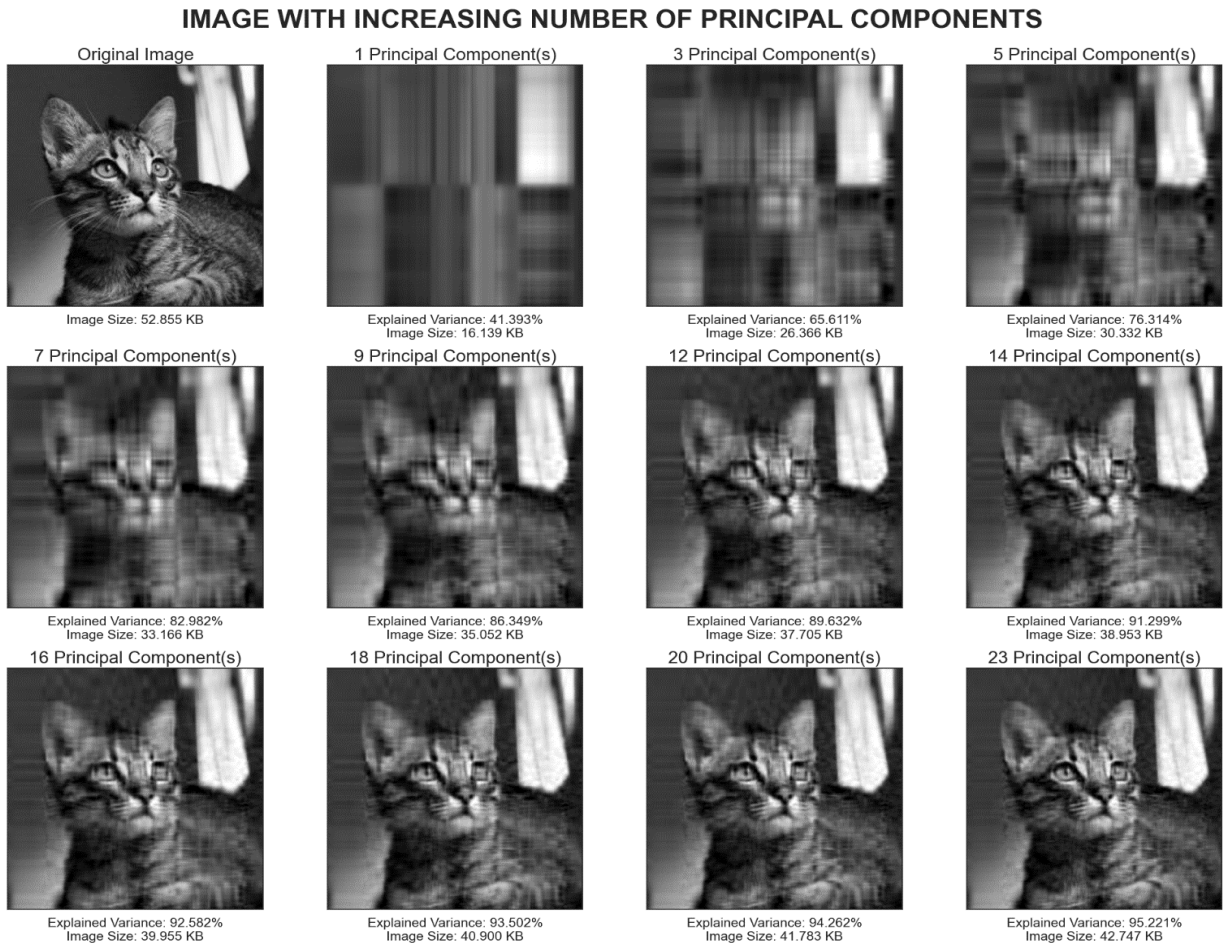


iii). <u>Analysis of Compression Metrics:</u>

Metrics such as explained variance, image size, and the number of colors are analyzed for each level of compression.
Plots are generated to visualize the relationship between the number of principal components and these metrics.



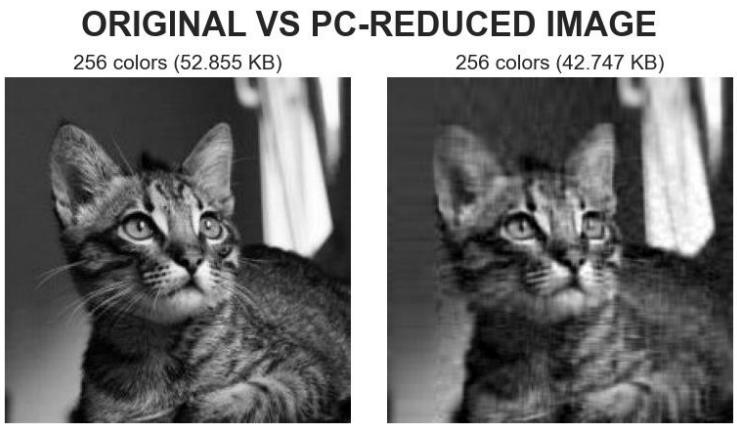
iv). Plotting Compressed (Reconstructed) images for various components:



These images are plotted with their respective sizes and explained variances.
From the plots, 33 Principal Components are the ideal choice because at this value reasonably high variance is covered and the color intensity is almost preserved as the original image (but there is some loss).

v). Comparison with the original image:

The original and compressed images are compared, and relevant metrics are presented. The impact of compression on image quality, size reduction, and color intensity is displayed. A side-by-side comparison is presented to showcase the original and optimally compressed images.




```
Reduction: 19.125% from original image size
Explained Variance: 95.221%
```

vi). Algorithm:

INPUT: Grayscale images of dimensions 220 x 220 pixels (standard test image).
OUTPUT: Compressed images resulting from PCA along with the original one for side-by-side comparison.
IMAGE_COMPRESSION_PCA_ALGORITHM (sample grayscale_image):
Step 1: Load the original image
original_image = load_image('sample.jpg')

Step 2: Explore the original image
image_info = explore_image(original_image)

Step 3: Apply Principal Component Analysis (PCA) to each color channel for lossy compression
compressed_images = []
for color_channel in original_image.color_channels():
    pca_result = apply_pca(color_channel)
    compressed_images.append(pca_result)

Step 4: Analyze compression metrics
compression_metrics = analyze_metrics(compressed_images)

Step 5: Visualize the compression metrics
visualize_metrics(compression_metrics)

Step 6: Compare original and compressed images
comparison_results = compare_images(original_image, compressed_images)

Step 7: Visualize the optimal compression results
visualize_optimal_compression(original_image, optimal_compression)

2. QOI (Quite OK Image format) Lossless Image Compression:

Quite OK Image (QOI) compression tool is a compression algorithm designed to balance compression efficiency and image quality. It is lossless and it can encode a huge image at a fast clip. It compresses about as well as PNG and was about as fast as an image can get to encode and decode. It is a great option for simple and fast encoding while also being on par with state-of-the-art formats. The methodology to apply qoi for lossless image compression is as follows:

i). Image Loading:

First, the sample image is loaded (continuing with the same grayscale sample image taken while using PCA) from the specified file path using a suitable image processing library.

ii). Compression Formats Definition:

The image compression formats to be used are defined, including 'png', 'jpeg', and 'qoi' (QOI compression).

iii). Image Encoding and Format Comparison:

The sample image is encoded using different compression formats and then we have compared the file sizes and visual quality of the encoded images using a specified function (e.g., compare_formats).

iv). File Size Calculation:

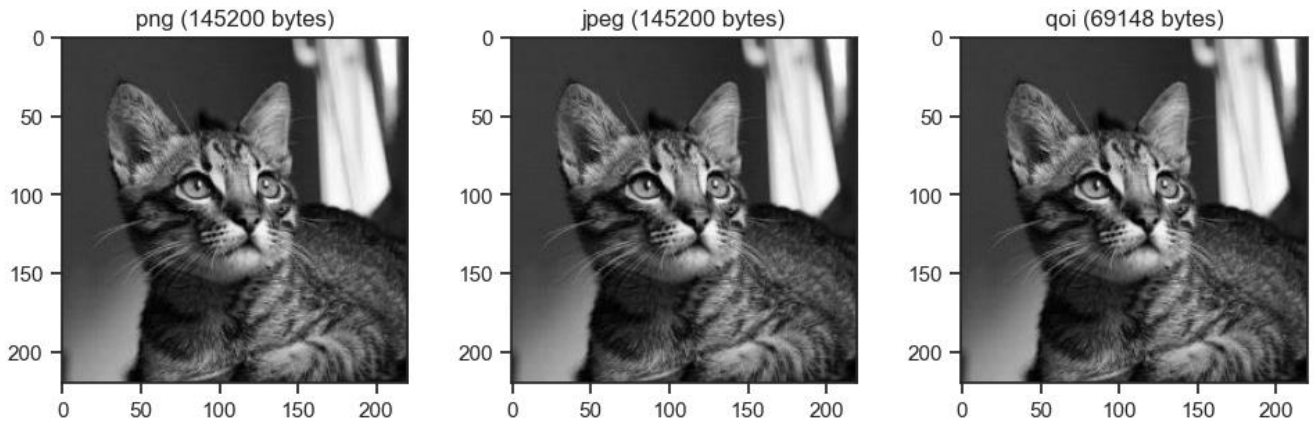
The file sizes are calculated for each compression format to quantify the impact of compression on storage requirements.

v). Visualization of Decoded Images:

Visualization of the decoded images is done for each compression format to assess the visual quality and fidelity of the compression.

vi). Conclusion and Analysis:

Conclusions based on the comparison of different compression formats are drawn by analyzing the trade-offs between file size and visual quality for each format.



vii). Algorithm:

INPUT: Grayscale images of dimensions 220 x 220 pixels (standard test image).
OUTPUT: Compressed images and file sizes resulting from QOI along with other formats like png and jpeg for comparison.
IMAGE_COMPRESSION_QOI_ALGORITHM (sample grayscale_image):
Step 1: Load the original image
original_image = load_image('sample.jpg')

Step 2: Define image compression formats
compression_formats = ['png', 'jpeg', 'qoi']

Step 3: Encode and compare image formats
encoded_images = {}
for format in compression_formats:
    if format == 'qoi':
        encoded_image = qoi.encode(original_image)
    else:
        encoded_image = encode_image(original_image, format)
    encoded_images[format] = encoded_image

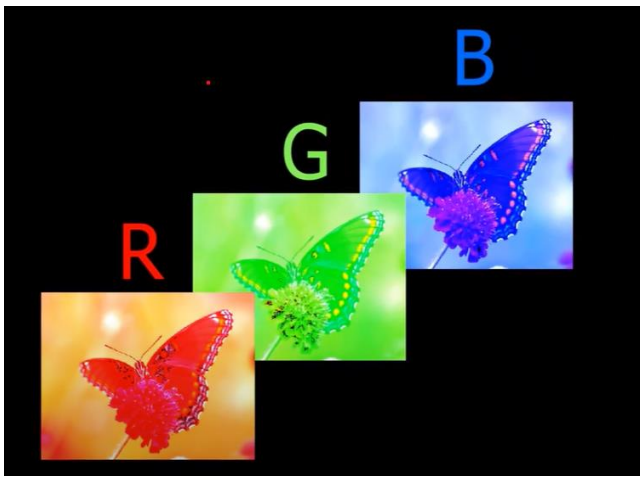
Step 4: Calculate file sizes for each format
file_sizes = calculate_file_sizes(encoded_images)

Step 5: Visualize the decoded images for comparison
visualize_decoded_images(original_image, encoded_images, file_sizes)

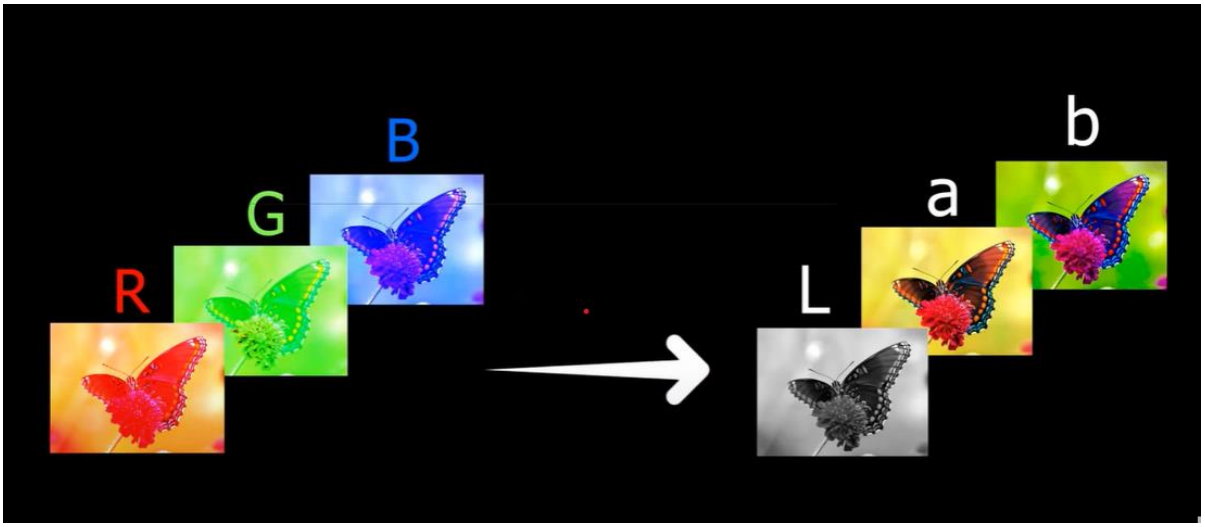
3. CNN (Convolutional Neural Networks) Image Colorization:

Image Colorization is a concept that takes a grayscale image as input, and by using Convolutional Neural Networks developed with the help of deep learning, converts it into RGB colored output image. Here, we are supplying the outputs from the compression techniques applied on the original ones as inputs to the colorization model. The methodology is explained as follows:

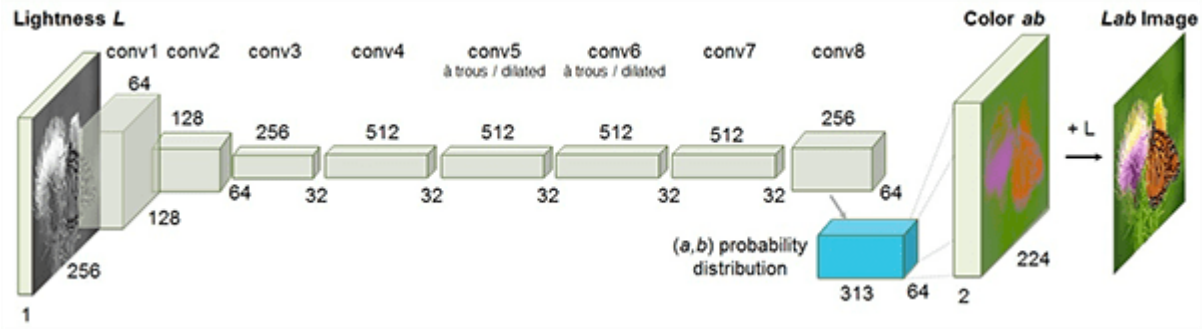
1. The whole trained imagenet data is converted from RGB format to the LAB color space. The color of every image is made up of the combination of Red-Green-Blue colors. For example: a sample image is taken to explain the whole procedure.



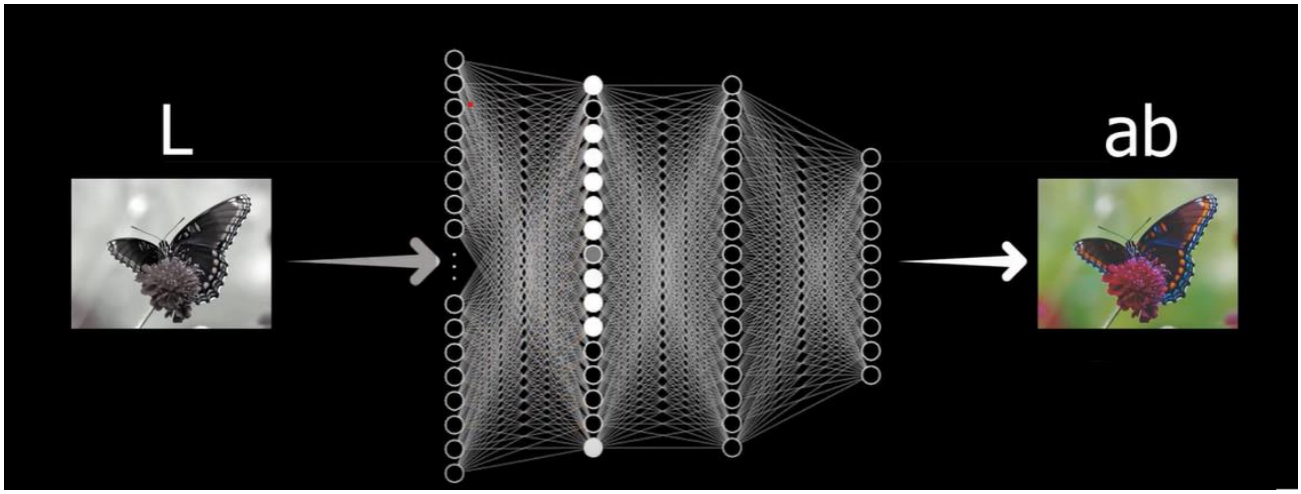
By default, OpenCV reads color of an image in BGR format, so by using cv2.cvtColor(image, cv2.Color_BGR2LAB), we convert the image into LAB color space where L channel refers to Lightness or Intensity, A channel refers to Green-Red and B refers to Blue-Yellow color combinations.



2. The L channel is supplied as input to the deep neural networks and the network is trained to predict the AB channels. The points are loaded that have 313 kernels (0-312) and 8 convolution matrices are developed.

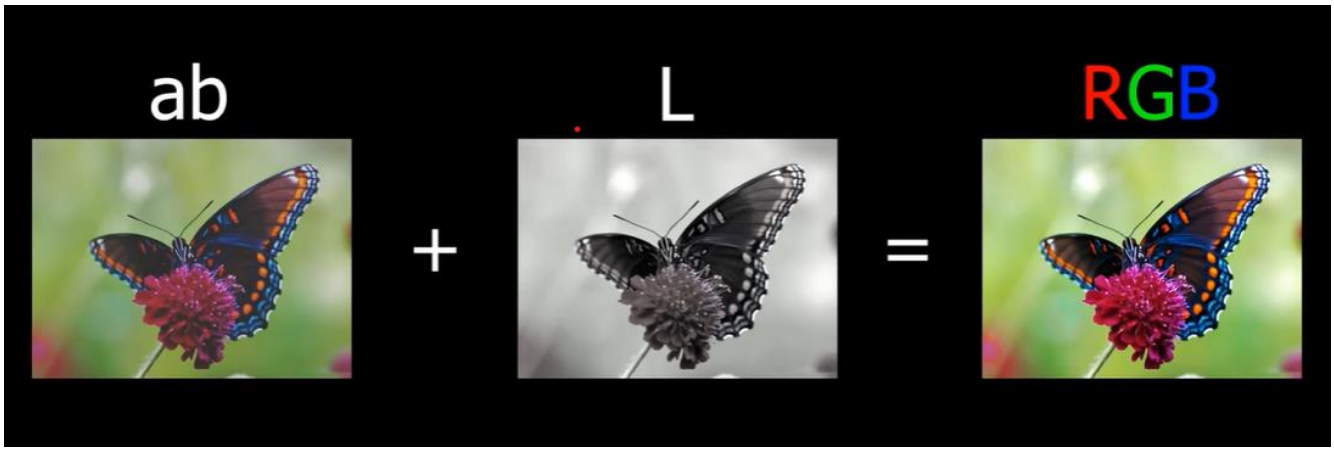


So, with the help of a large image data supplied to the networks, the whole model gets trained to be able to predict AB colors.
The intensity of the L channel plays an important role to provide AB colors based on the level of lightness of a particular color.



3. Now, the predicted AB channels are combined with the L channel to produce the LAB color space.

4. Finally, as we know that cv2 recognizes and works with images in BGR format, the LAB color space is again converted back to BGR by using cv2.cvtColor(image, cv2.COLOR_LAB2BGR).

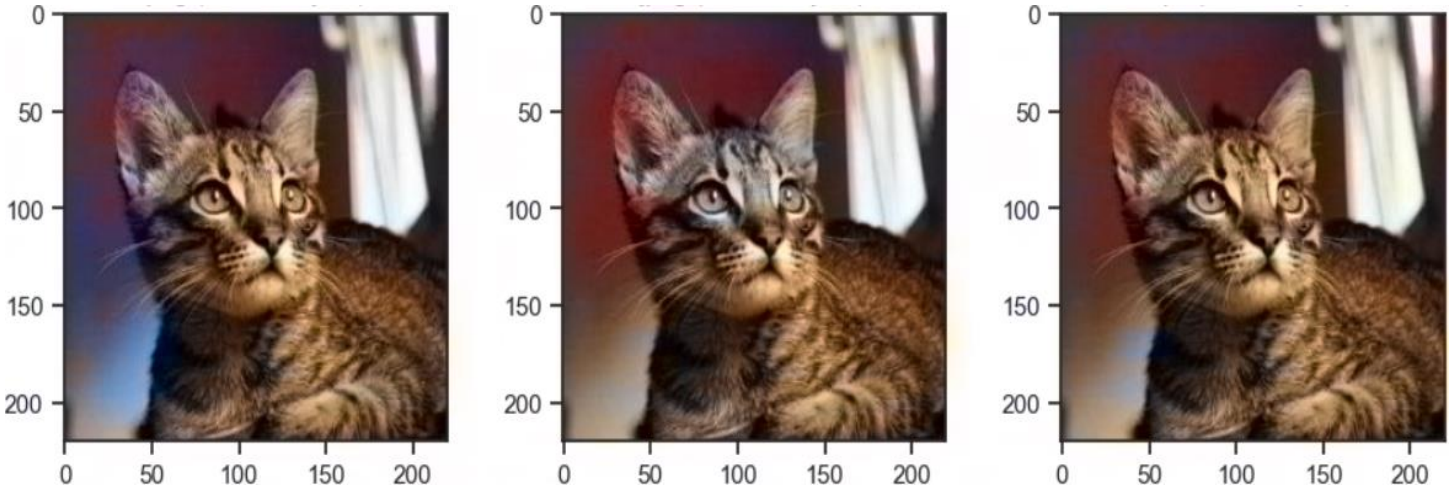


Hence, at the end of the whole procedure, we get the desired output depicted above.

Now according to the compressed outputs of lossy and lossless compression, below are the colorized outputs achieved after applying the colorization models.

(i) Colorized output of image compressed using PCA:

(ii) Colorized output of image compressed using QOI:

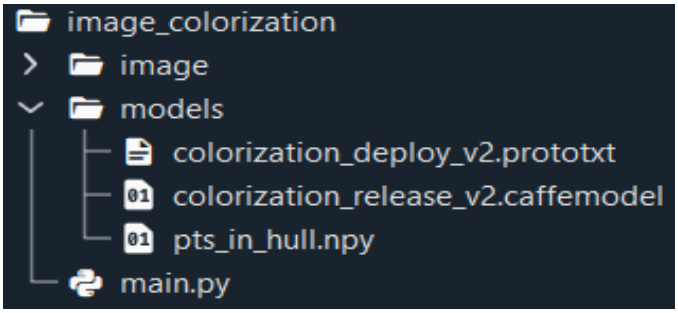


DATASET USED:

To conduct our experiment on a wide variety of images, we compiled our database from widely known imagenet datasets and colorization models available on the internet.

- There is a .prototxt file, a text file which is a prototype machine learning model created for the use with Caffe.
- Caffe (Convolutional Architecture for Fast Feature Embedding) is a deep learning framework that allows users to create image classification and image segmentation models. Initially, the models are created and saved as plain text .prototxt files. After the model is trained and refined using Caffe, the program saves the trained model as a CAFFEMODEL file.
  Hence, .prototxt files are used to create .caffemodel files.
- We also used .npy file to collect the points for creating the convolutional matrices for the color prediction. This file is accessed using inbuilt NumPy library.

Here is a snippet of the dataset used for colorization procedure:



ALGORITHM:

INPUT: All the compressed output images (grayscale version) are supplied to neural networks.
OUTPUT: Colored version of the grayscale image data predicted by the project model.
COLORIZATION_ALGORITHM (compressed grayscale_image**):**
STEP1: import all the required standard libraries. Here, import NumPy for numerical operations on the array related data in Python and import OpenCV for image storage, manipulation and retrieval.
STEP2: Declare 3 variables prototxt, model and points and assign the model and points files to them.
STEP3: Read the data from the model files and design a deep neural network using cv2.dnn.readNetFromCaffe(files) function. Also, load the points from .npy file.
STEP4: Get the convolution 8 matrix information to apply it to the input layer to obtain AB channel after the normalization.
STEP5: Declare a variable image and store the path of sample image from the image folder on which the prepared model is to be tested and scale it down by 255.0.
STEP6: IF (image in RGB format) THEN:
Convert the image from BGR to LAB using cv2.cvtColor(image, cv2.COLOR_BGR2LAB).
ELSEIF(image in grayscale format) THEN:
Directly take the L channel form the black and white sample image.
ELSE:
Print("Error! Image cannot be detected.")
END IF
STEP7: Supply the L channel as input to the neural network using setInput(cv2.dnn.blobFromImage(L)) function to predict AB channels.
STEP8: Resize the AB channels and concatenate it with L using np.concatenate(L,AB).
STEP9: Store this image in other variable say colorized and convert it back to RGB format using cv2.cvtColor(image, cv2.COLOR_LAB2BGR). Scale up the image: (255*colorized).astype("uint8").
STEP10: Final output is displayed using:
cv2.imshow(image)
cv2.imshow(colorized)
cv2.waitKey(0)
STEP11: To further test for more test data images, repeat steps 5 to 9 and check the working of the designed model by displayed result.

FLOW OF THE ALGORITHM:

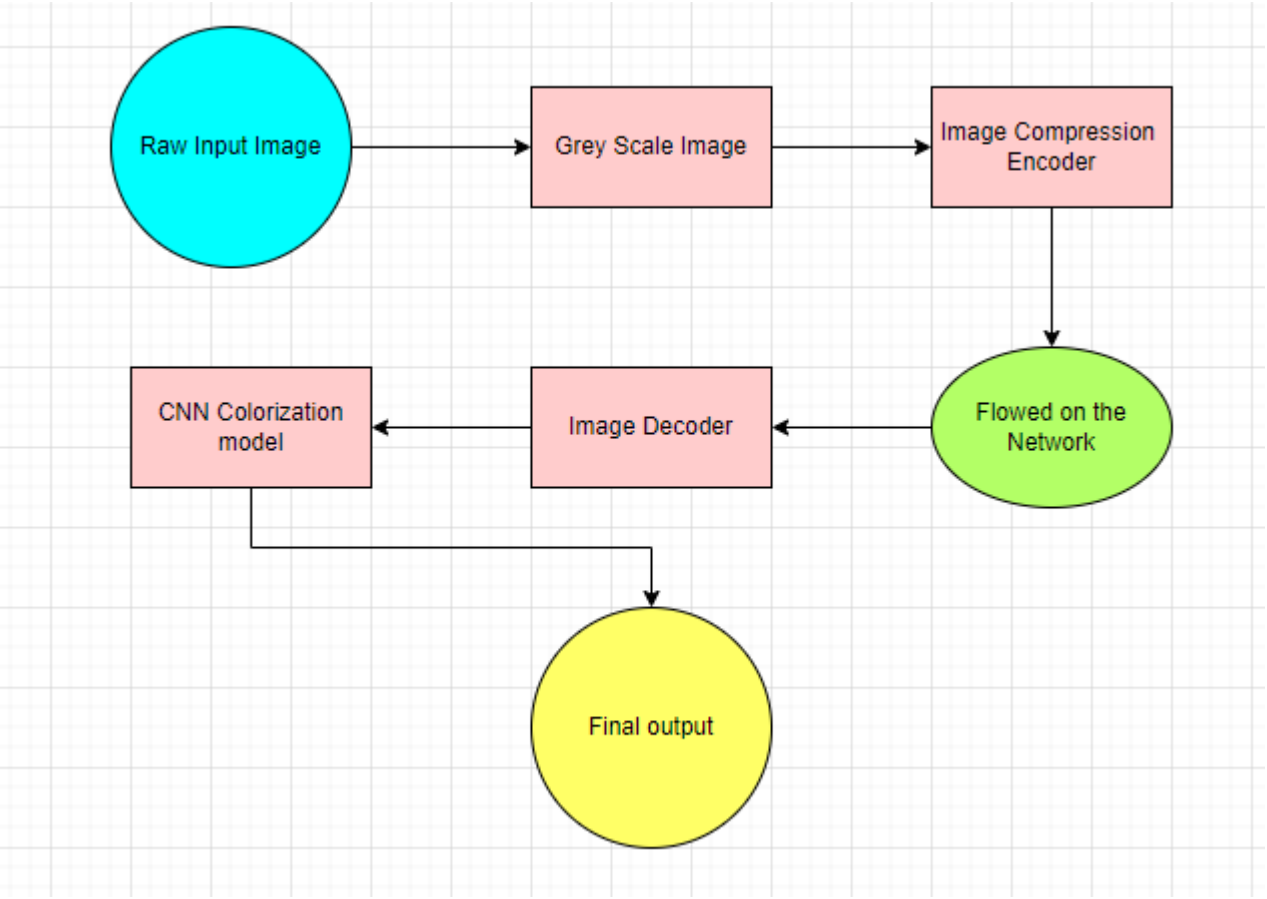
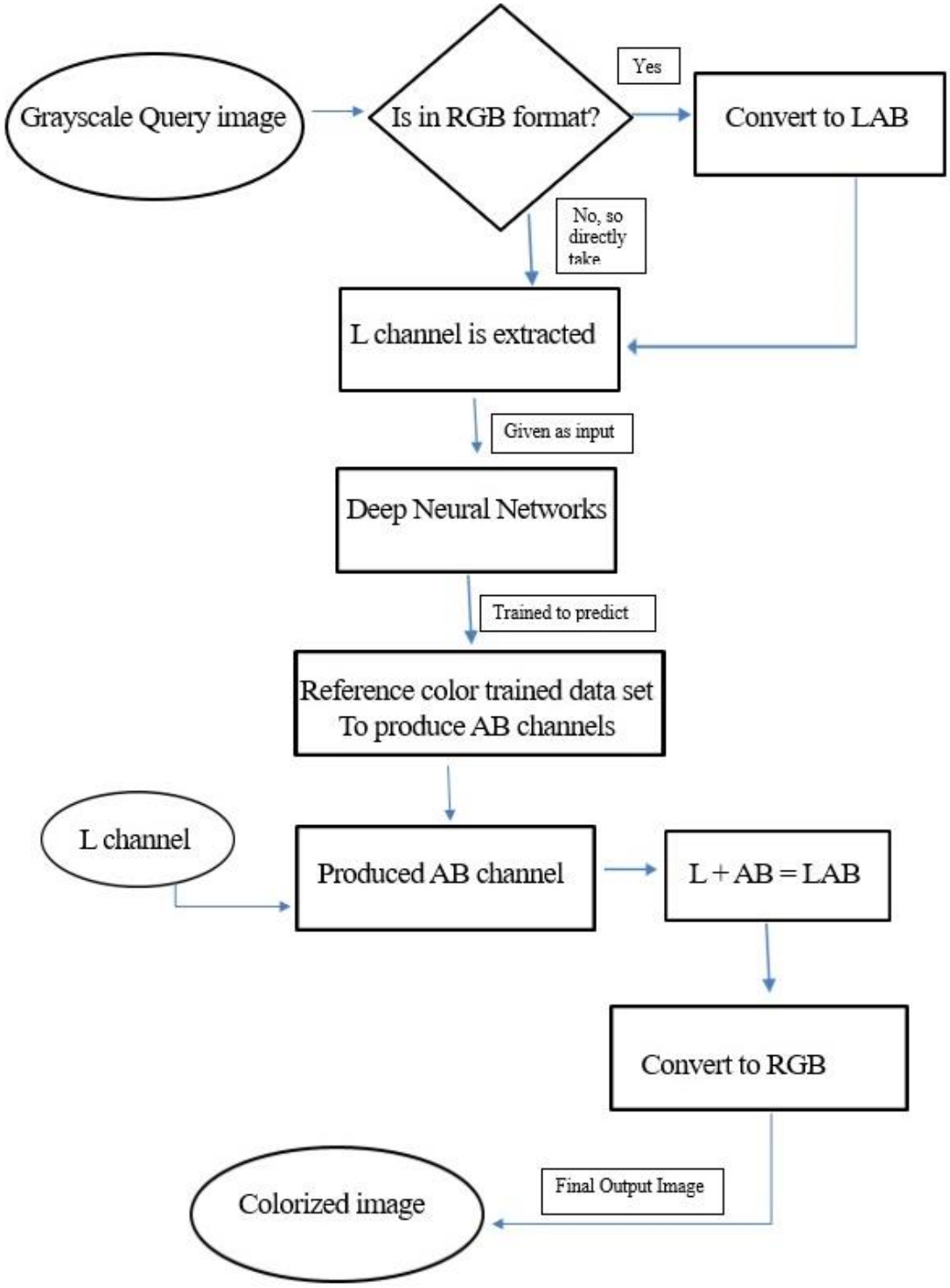


Image describing the whole flow of Our Compression Algorithm.

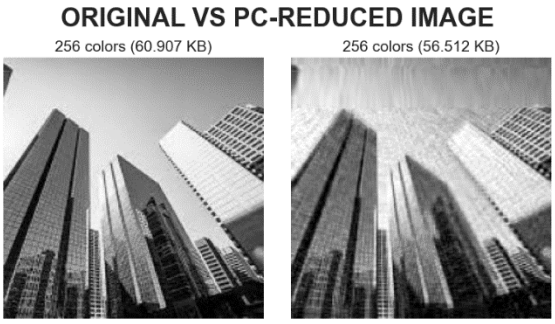
FLOW CHART OF THE COLORIZATION MODEL:

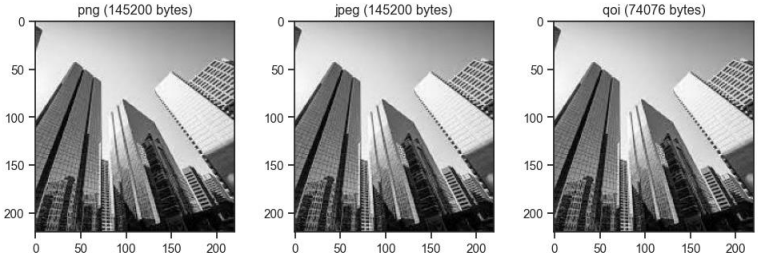


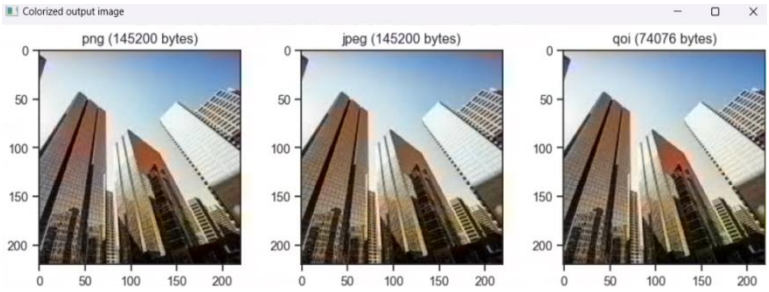
RESULTS

The outputs of the qualitative evaluation of the proposed method are shown below. Observations can also be made to check the efficiency of compression techniques and to which extent, the resulting colorized images resemble the ground truth color images.
GitHub Repository link:
https://github.com/amisha01014/ML_Project_DecodeCompressColorize

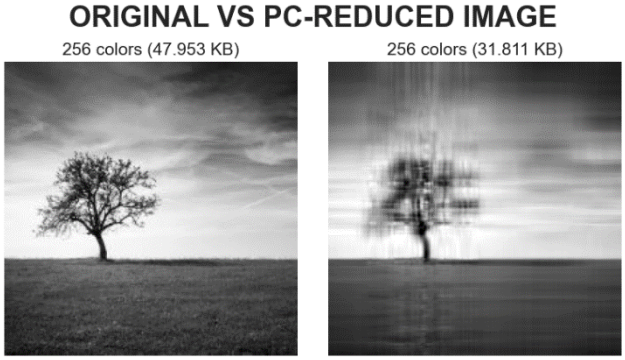
Sample image 1:

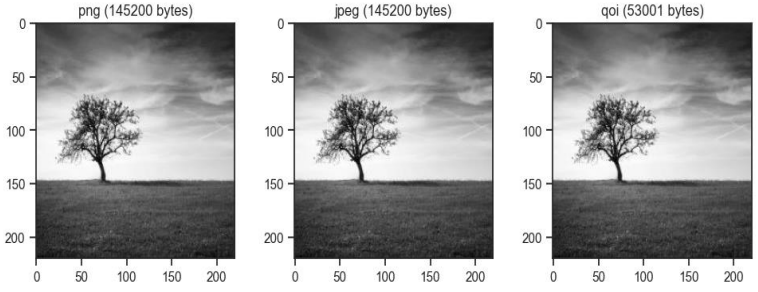


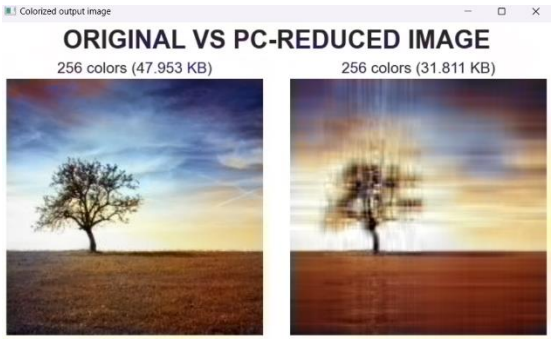


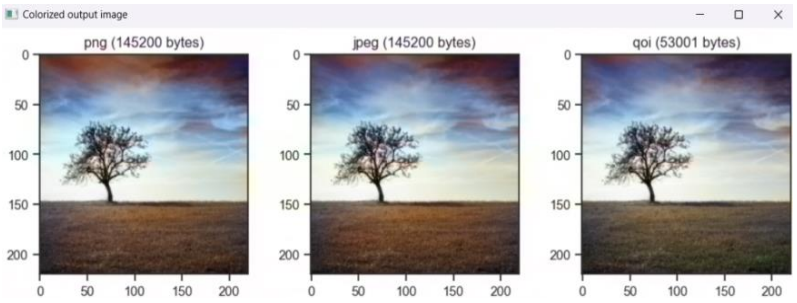


Sample image 2:

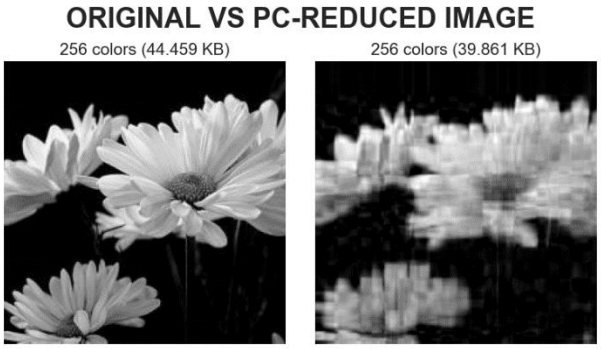


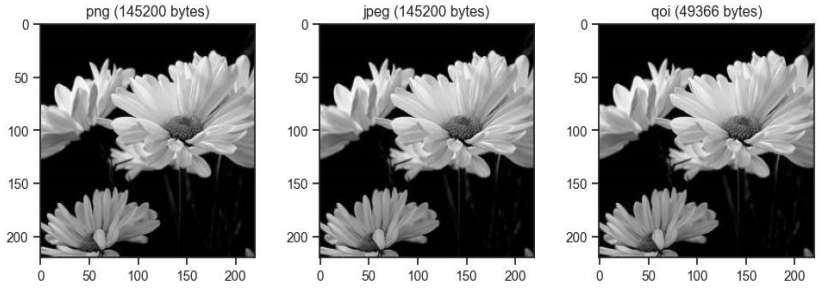


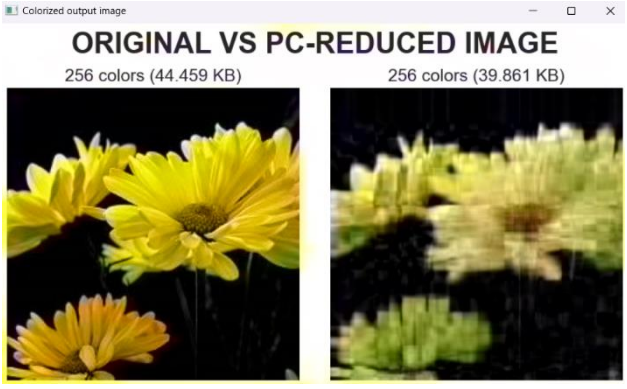


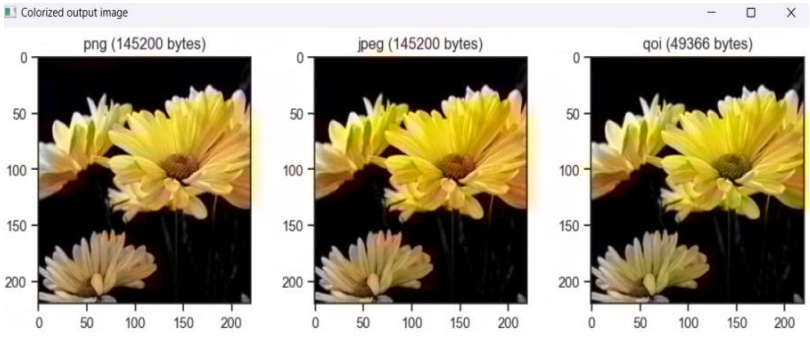


Sample image 3:









FUTURE SCOPE

QOI (Quite OK Image Format) is a promising new image compression format that offers a good balance of compression ratio, speed, and simplicity compared to more complex formats like PNG. Recent research has explored adopting QOI in gaming pipelines and optimizing QOI encoding for GPUs. However, QOI has room for improvement and further development as an ecosystem. Potential areas of advancement include support for additional image types beyond RGB, more advanced compression techniques like context adaptive coding and progressive decoding, wider integration into software, and building out a comprehensive QOI library and toolset.

The image colorization techniques are in great use. Particularly, it was designed keeping in mind its application as a DeOldify model. DeOldify is a Deep Learning (DL) based project for coloring and restoring old images and videos. It helps us add color to old black and white photos, adding life to them. Though in our model we are using CNN, this is a special type of GAN called a self-attention GAN, where GAN stands for Generative Adversarial Neural Network.
Hence, the Deoldify model lets us recolor old images and videos of family members or even cities.
The present color detection project takes the path of an image as an input and looks for the composition of three different colors red, green and blue in the given image. In this, various steps are implemented using OpenCV platform. The main positive point of this method is its color differentiation of a mono color.
However, nowadays diverse and complex images require superior colorization technology. Deep learning models are more and more commonly used in image colorization to perform automatic or semi-automatic colorization and achieve good performance.

In the future scope, the detection of the edge detection techniques has different other applications like facial detection, color conversion for grayscale image etc. that can also be implemented. So, the version of this project can be upskilled by adding the technology of converting the original color as per the requirement by adjusting intensity, saturation and color codes.
As in the existing system complete accuracy is not guaranted, so future work can be done to address this issue and increase the quality of the luminance provided to the image color produced.
Driven by novel technology in the future, Image colorization will develop in the direction of unsupervised learning to achieve high-quality reconstructed images. If possible, combining with new technologies, using diversified colorization and multiple data sets may achieve a better colorization effect and broader application scenarios.

REFERENCES

[1] Deep learning for image colorization: Current and future prospects - ScienceDirect
[2] Automatic Image and Video Colorization using Deep Learning | IEEE Conference Publication | IEEE Xplore
[3] Image Colorization Using CNN With OpenCV (learnopencv.com)
[4] Deep Learning based image colorization with OpenCV – CV-Tricks.com
[5] Deep Learning Project - Colorize Black and White Images with Python - TechVidvan
[6] Quite Ok image format | Anticryptography