INTRODUCTION TO PYTHON

Image Colorization

A REPORT

Submitted by

Group-A3

Anushka Singh (CB.EN.U4AIE22006)

Lakshmi KN (CB.EN.U4AIE22023)

Surya Santhosh Kumar (CB.EN.U4AIE22049)

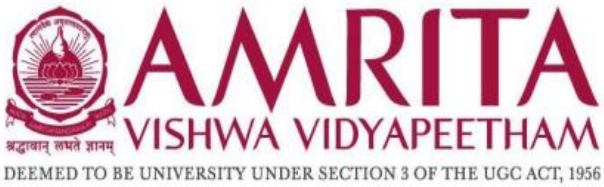
ABHIRAMARAJU RALLABANDI (CB.EN.U4AIE22066)

In partial fulfilment for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

CSE(AI)



Centre for Excellence in Computational Engineering and Networking (CEN)

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE

AMRITA VISWA VIDHYAPEETHAM

COIMBATORE – 641 112 (INDIA)

DECEMBER– 2023

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

BONAFIDE CERTIFICATE

This is to certify that the thesis entitled "Image Colorization" Submitted by "**Anushka Singh (CB.EN.U4AIE22006), Lakshmi KN (CB.EN.U4AIE22023), Surya Santhosh Kumar (CB.EN.U4AIE22049), ABHIRAMARAJU RALLABANDI (CB.EN.U4AIE22066)"** for the award of the Degree of Bachelor of Technology in the "CSE(AI) " is a bonafide record of the work carried out by her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

Ms. Sreelakshmi K

Project Guide

Dr. K.P.Soman

Professor and Head CEN

Submitted for the university examination held on

____date__

AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE
AMRITA VISHWA VIDYAPEETHAM
COIMBATORE - 641 112

DECLARATION

I, **Anushka Singh (CB.EN.U4AIE22006), Lakshmi KN (CB.EN.U4AIE22023), Surya Santhosh Kumar (CB.EN.U4AIE22049), ABHIRAMARAJU RALLABANDI (CB.EN.U4AIE22066)** hereby declare that this thesis entitled "Image Colorization", is the record of the original work done by me under the guidance of Ms. Sreelakshmi K, Assistant Professor, Centre for Computational Engineering and Networking, Amrita School of Artificial Intelligence, Coimbatore. To the best of my knowledge this work has not formed the basis for the award of any degree/diploma/ associate ship/fellowship/or a similar award to any candidate in any University.

Place: Coimbatore

Date: **Signature of the Student**

Acknowledgement

We would like to express our special thanks of gratitude to our teacher (**MS. SREELAKSHMI K** ma'am), who gave us the golden opportunity to do this wonderful project on the topic **Image Colorization**, which also helped us in doing a lot of Research and we came to know about so many new things. We are thankful for the opportunity given. We would also like to thank our group members, as without their cooperation, we would not have been able to complete the project within the prescribed time.

Abstract

Image colorization poses a significant challenge due to its inherently ill-posed nature, often characterized by color-multimodality, where various plausible colorized versions can exist for a single grayscale image. Traditional manual colorization methods, carried out by skilled artists, required expertise and extensive effort for each image. Semi-modern approaches involve user intervention, relying on manual annotations to guide the algorithm. In contrast, our proposed method is fully automatic, eliminating the need for user involvement. Upon inputting a grayscale image, the algorithm generates a colorized version without further intervention. The choice between automatic and semi-automatic methods depends on the context; automatic approaches excel in scenarios requiring swift, intervention-free output, such as colorizing films with numerous frames. Conversely, semi-automatic methods offer maximum control over colors and areas, beneficial for precise colorization when guided by color anchors. A recent development is the emergence of Hybrid image colorization, combining the precision of semi-automatic methods with the efficiency of automatic processing, all accomplished in an automated manner. This abstract provides an overview of the challenges, methodologies, and contextual considerations in the domain of image colorization.

Table of Contents

List of Figures

1. Introduction

Image colorization poses a considerable challenge as an ill-posed problem [3], characterized by color-multimodality, wherein multiple plausible colorized versions may exist for a given grayscale image. Historically, manual image colorization was undertaken by skilled artists relying on their expertise and mental visualization of specific scenes. This labor-intensive process demanded significant dedication for colorizing a single image. Semi-modern approaches often necessitate user intervention, involving manual annotation of different image areas with colors to guide the algorithm in the colorization process [7]. In contrast, our approach is entirely automatic [8] , eliminating the need for user involvement. The idea is illustrated in Fig.1. Upon inputting a grayscale image, the user is presented with the colorized version without further intervention. While the superiority of one method over the other is context-dependent, automatic approaches are advantageous in situations demanding swift, intervention-free output, such as colorizing films with numerous frames. Conversely, semi-automatic methods offer maximum control over colors and areas, beneficial when precise colorization is paramount, facilitated by color anchors. In recent years, a third category has emerged—Hybrid image colorization [5]—blending the effectiveness of semi-automatic precision with the rapidity of automatic processing, all achieved in an automated manner.

Figure 1 Image Colorization

In this project, we have restated the image colorization problem as an image classification problem and employ Support Vector Regressions for our backbone operation.

2. Dataset

The dataset selection focused on images showcasing Yellowstone National Park, sourced from Flickr using the tags "yellowstone" and "landscape." This strategy ensured a diverse collection encompassing various elements, including animals, humans, and buildings. To maintain uniformity, all images in the dataset underwent standardization, fixing their width at a constant 500 pixels. Notably, pre-existing grayscale photos were systematically excluded from consideration.

Figure 2 A glimpse into the Dataset

3. Feature Extraction

3.1 Segmentation

Image segmentation refers to the splitting up of an image into various areas of interest called segments [9][10] (Fig.3) . It is usually done with the help of an image segmentation algorithm which on its own can be a semi-automatic [7] and also an automatic approach [3][5][8]. Segmentation on its own is a vast area under Machine Learning [9]. For our task we use a SLIC(Simple Linear Iterative Clustering) Algorithm for segmentation.

For the classification of local features, we use segment the images into sections utilizing the SLIC superpixel algorithm. Our preference for the SLIC algorithm over other segmentation methods stems from its efficacy in generating uniform, compact segments. The implementation in scikit-image was utilized for this purpose. The results of SLIC clustering on one of our test images are illustrated in Fig.4.

Figure 3 Segmentation by SAM (Segment Anything Model) by Meta AI [10]

Figure 4 SLIC Segmentation demo (n_segments=350, compactness=25)

- **n_segments**: This parameter defines the approximate number of segments (or superpixels) that the image will be divided into. Higher values will result in a larger number of smaller segments, while lower values will produce fewer larger segments.
- **compactness**: The compactness parameter balances color proximity and spatial proximity. Higher values make the segments more square and more regularly shaped, while lower values may result in more irregularly shaped segments.

Experimenting with different values for ‘n_segments’ and ‘compactness’ is recommended. The SLIC algorithm is typically designed to work with color images, and it may not provide optimal results when applied directly to grayscale images. This is because SLIC takes advantage of both spatial and color information to create superpixels, and if you use it on a grayscale image, it lacks the color information. While using the SLIC algorithm on a grayscale image, it is necessary to set an additional argument channel-axis=None while making the function call (There appears to be some errors in python while using the SLIC implementation in the skimage library. It was resolved when channel-axis=None was set.).

3.2 Subsquare Data

Once the segments of an image is obtained, the next task is to find the centroid of each of those segments. Firstly, the mask of the segment over the original image is created. The segment mask image (image mask is a matrix of 0’s and 1’s) will have 1’s for all pixels that comes within the segment and 0’s for all other pixels of the image. Next find the mean of all rows to get x-coordinate of centroid and mean of all columns to get y-coordinate of centroid. After finding the centroid, we construct a 10x10 square around the centroid, this is called a Subsquare. We convert the Subsquare (which is in RGB scale initially while training) into YUV scale. The feature extraction procedure involves 3 steps to be done with the Y, U and V channels. First take the Fast Fourier Transform(FFT) of the Y Subsquare, which gives us 100 complex numbers (100 values because Subsquare is 10x10 size. The resulting numbers are split as 100 real + 100 imaginary numbers giving a 1x200 vector). Secondly and Thirdly, take the average U and V values of the Subsquare giving 2 additional values U_{avg} and V_{avg} respectively. For one Segment, we will be left with a 1x200 vector (FFT), a 1x1 value (U_{avg}) and another 1x1 value (V_{avg}). We perform this feature extraction for all segments of a image. For example, if an image has 180 segments, there will be 180 Subsquares which gives a 180x200 matrix of feature vectors, 180x1 vector of U values and another 180x1vector of V values.

```
def generateSubsquareFeatureData(im):
    segments = slic(im, n_segments=n_segments, compactness=compactness, sigma=1)

    square_size = 10
    fv=[]
    U=[]
    V=[]

    ...
    ...
    ...

    return fv, U, V
```

Figure 5 Generating subsquare feature data: code snippet

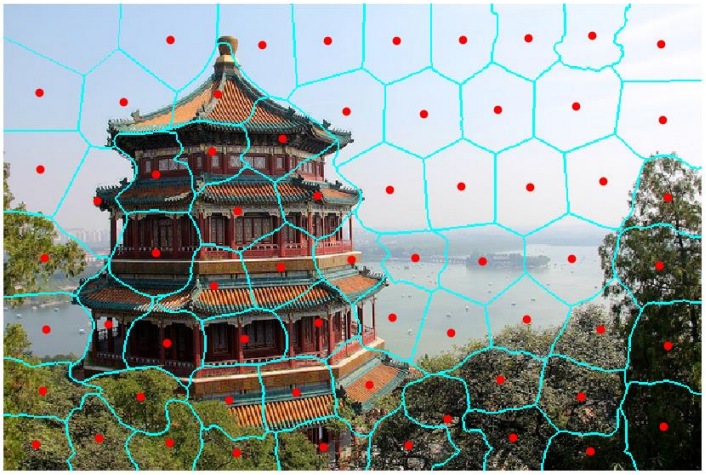


Figure 6 Subsquare Centroids

There are cases when the Subsquare does not fit inside the segment. In which case we might have to do one of few things. Increase the size of each segment so that the subsquare can fit in (by decreasing n_segments of SLIC), or reduce the size of the subsquare for all the segments, maybe take a 5x5 instead of 10x10 (less preferred), or, the final thing to do would be to completely ignore extracting features for that particular Subsquare (We have chosen to do this, it is the simplest way). Fig.6 Shows the Segments, Subquares and Centroids of a given image.

3.3 Preparing the Training Data

If there are 10 images, and each image has 180 segments. Assuming we are using a 10x10 sliding window for FFT feature extraction [5] we will be left with a 180x200 matrix for each image. For preparing the training data we implement another function as shown in Fig.7.

```
def resize_and_generate_feature_data(files):
    ...
    ...
    ...
    return Y, features, U, V
```

Figure 7 Preparing training data: code snippet

If the training was done only on one image, then the **X_train** would just be the 180x200 matrix just obtained. However, since there are 10 images, a 10x180x200 matrix will be obtained. In reality, some images may have more or less segments, so the actual size of data we obtain might vary. (eg: $\mathbf{F_{180x200}}$, $\mathbf{F_{230x200}}$,$\mathbf{F_{165x200}}$……$\mathbf{F_{220x200}}$ list of 10 matrices for each image). Finally, all the matrices are stacked vertically using vstack function.

4. Model Training

Training the SVR model with our extracted data is a straightforward process. The feature vectors are stored in the variable **X_train**, while **U_train** and **V_train** hold the corresponding U and V values for each feature vector. The training procedure is concise, consisting of merely six lines of code. Firstly both SVRs are trained by providing **X_train** as the first input, it serves as the first input for both models. Subsequently, the U and V values from **U_train** and **V_train** are supplied to the respective SVR models. To streamline future use, the trained model can be saved in the directory using the **joblib** module, employing the **joblib.dump()** method. This enables seamless utilization of the model for subsequent colorization tasks without the need for re-training. Fig.8 Shows The difference the choice o kernel has over the ability to find hyperplane separating the training data.

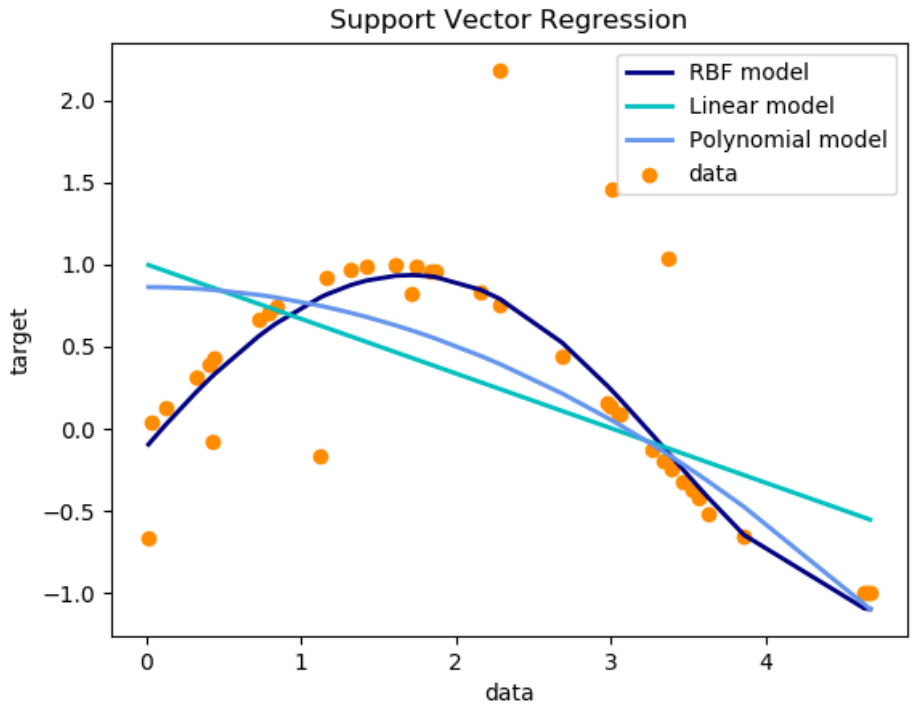


Figure 8 Support Vector Regression plot on some random data to showcase the difference between different kernels

The selection of SVR (Support Vector Regression) as our Machine Learning Model stems from its design tailored for regression tasks. Unlike SVM (Support Vector Machines), which predicts categorical labels, SVR focuses on forecasting continuous output. The outcome of SVR is a real number that represents the predicted value. Post-training, the two SVR models obtained are denoted as **U_SVR** and **V_SVR**. Both models are capable of predicting U and V values, respectively, when provided with a feature vector input.

5. Hyperparameter Tuning

The SVR Model has additional parameters called Hyperparameters. These are called C and Epsilon. These parameters can be seen in the equations of the SVM formulation given in the image in Fig.9.

By varying the C and Epsilon values one can control the training of the model. Choosing an optimal C and Epsilon value is very crucial for the effectiveness and performance of the model.

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*)$$

$$\text{subject to} \quad \begin{cases} y_i - \langle w, x_i\rangle - b \le \varepsilon + \xi_i \\ \langle w, x_i\rangle + b - y_i \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases}$$

$$|\xi|_\varepsilon := \begin{cases} 0 & \text{if } |\xi| \le \varepsilon \\ |\xi| - \varepsilon & \text{otherwise.} \end{cases}$$

Figure 9 Support Vector Regression Formulation

The constant C > 0 determines the trade-ff between the flatness of ***f*** and the amount up to which deviations larger than Epsilon are tolerated. One additional parameter is the kernel for the SVR, this is a special function that maps the input feature vectors to a higher dimensional space and draws a classifier in that higher dimensional space. There are many kernels experimented like ***poly***, ***linear***, ***rbf*** etc.. and we found that the RBF kernel (Radial Basis Function) works best for the the model.

To assess the model's effectiveness, various error formulation methods are at our disposal. In this project, Mean Squared Error (MSE) has been chosen as the metric for evaluating the model's performance. A lower MSE value signifies closer proximity to the ground truth, indicating superior colorization results.

6. Colorization

The testing code for image colorization utilizes two trained Support Vector Regression (SVR) models, **u_svr** and **v_svr**, to predict the U and V values of superpixel segments in a given grayscale image. The algorithm employs the same segmentation function defined earlier that uses SLIC (Simple Linear Iterative Clustering) algorithm to generate superpixels and extracts features by applying Fourier Transform to the Y channel within each segment. For each of the segments the same procedure is performed, and the extracted features for each segment is passed to both the SVR's that outputs the predicted U and V average values for that segment. The predicted U and V values are then utilized to modify the U and V channel of the Y (grayscale) input image, achieving colorization. The Mean Squared Error (MSE) between the predicted and true U and V values is calculated and shown to evaluate the accuracy of the predictions. Finally, the YUV image is converted to RGB image. See Fig.10 for function implementation

```
def colorize_image(image_path, u_svr, v_svr, n_segments=500, compactness=20, sigma=1):

    # Read the image
    ...
    ...

    # Perform Feature Extraction and Prediction
    ...
    ...


    count = 0
    for label in np.unique(segments):
        mask = label == segments

        yuv_image[mask, 1] = predicted_U[count]  # U channel
        yuv_image[mask, 2] = predicted_V[count]  # V channel
        count += 1

    colorized_image_rgb = cv2.cvtColor(yuv_image, cv2.COLOR_YUV2RGB)

    return colorized_image_rgb
```

Figure 10 Colorize Image Function: code snippet

The code iterates through a set of test images, displaying the original, grayscale, and colorized versions for visual assessment. This process aids in evaluating the effectiveness of the trained SVR models in accurately colorizing grayscale images.

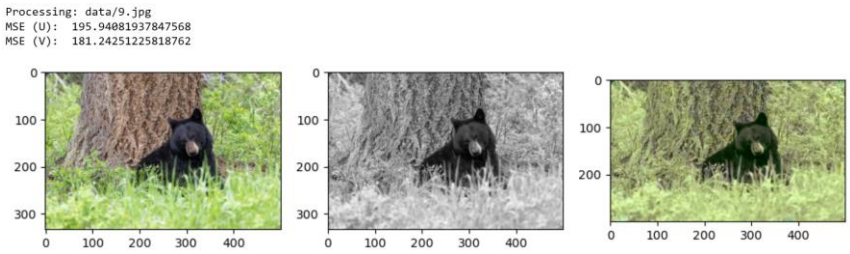


Figure 11 Colorization of an input Grayscale image

7. Results

Presented here are the outcomes yielded by our model. It is evident that the model performs admirably in numerous instances, exhibiting minimal errors. Nevertheless, discernible instances exist wherein the model encounters challenges in achieving flawless colorization of images. Currently, the model serves as a color transfer mechanism, being trained on a singular reference image and subsequently applying that color scheme to our input test image through feature matching. It is noteworthy that the model also demonstrates efficacy when trained on a corpus of images, enabling it to predict the coloration of various test images [6].

Figure 12 Image Colorization

The results highlight areas for enhancement. Notably, improving segmentation involves adopting global image segmentation to identify object classes and then performing local segmentation. This dual approach enhances the model's understanding with a consideration of both global and local features. Another refinement is transitioning to advanced feature extraction methods [11], such as Haar-DWT or histogram features [4]. These adjustments aim to elevate the model's proficiency in image colorization. All these results leave a lot of room for improvement, and this project paper is just a small stepping stone towards in the right direction towards our goal of Image colorization.

Github link: github.com/MonospaceSurya/image_colorization/tree/master

Figure 13 Image Colorization using SVR - Our Results a) Trained Reference Image, b)Groundtruth Image, c) Grayscale Input, d) Colorized Output

Testing No.	C	Epsilon	MSE(U)	MSE(V)	Total Error
1	91	7	230.24	183.65	413.65
2	91	7	54.34	108.71	163.05
3	91	5	124.91	264.06	388.97
4	91	7	145.22	164.21	309.43
5	91	7	343.51	531.32	874.83
6	11	7	19.40	75.47	94.87

8. Conclusion

In the vibrant journey of exploring image colorization, we have delved into the intricate world of algorithms and methodologies. From the historical hands of manual colorization by skilled artists to the automatic prowess of our chosen approach, we've witnessed the evolution of a captivating process. Our automatic approach, requiring no user intervention, emerges as a beacon of efficiency in transforming grayscale images into vivid, lifelike compositions. As we conclude this chapter of our project, it's crucial to acknowledge the dynamic nature of colorization techniques. One method may not universally overshadow the other; instead, their effectiveness depends on the context and requirements. Whether it's the swift pace of automatic colorization for film frames or the meticulous control afforded by semi-automatic methods, the art of colorization unfolds in diverse ways.

This project not only unravelled the complexities of image colorization but also underscored the importance of adaptability in choosing the right method for the right scenario. The canvas of possibilities is vast, and our exploration is but a stroke in the grand masterpiece of computer vision.

9. Future Work

- Refined Segmentation Techniques:

  The accuracy of image colorization is tied to the quality of segmentation. Future endeavors should aim to explore and develop more sophisticated segmentation techniques.

- Enhanced Feature Extraction Methods:

  The richness of colorization lies in the features extracted from grayscale images. Future work should delve into refining and expanding feature extraction methods. By uncovering more intricate details and patterns, we can enhance the model's ability to faithfully replicate the colors of the real world, ensuring a more lifelike and visually appealing output.

- Evolving Kernels for Enhanced Performance:

  The heart of Support Vector Regression (SVR) lies in its kernel function. A dedicated exploration into refining and diversifying kernel options can open new avenues for improved colorization results.

10. References

[1] Development of Colorization of Grayscale Images Using CNN-SVM. *Abdallah Abualola, Teddy Surya Gunawan, Mira Kartiwi, Eliathamby Ambikairajah, Mohamed Hadi Habaebi*

[2] STEFANN: Scene Text Editor using Font Adaptive Neural Network. *Prasun Roy, Saumik Bhattacharya, Subhankar Ghosh, Umapada Pal*

[3] Automatic Colorization of Grayscale Images. *Austin Sousa, Rasoul Kabirzadeh, Patrick Blaes. Department of Electrical Engineering, Stanford University*

[4] Support Vector Machine (SVM) for Colorization of Grayscale Image. *Shymaa Akram Alrubaie, Israa Mohammed Hassoon. Al-Mustansiriyah University, College of Science, Mathematics Science, Baghdad-Iraq*

[5] From Grayscale to Color: Digital Image Colorization using Machine Learning. *Cris Zanoci, Jim Andress*

[6] Automatic Image Colorization. *Harrison Ho and Varun Ramesh. Stanford University*

[7] Image colorization based on self-adaptive mutation particle swarm optimization and support vector machine. *Chen Ying, Gao Lelian, Liu Guoqing, Chen Hengshi, Department of Computer Science and Information Engineering, Shanghai Institute of Technology, Shanghai, China*

[8] Image Colorization using GAN, www.youtube.com/watch?v=ndfVyeA0Wts

[9] Segment Anything Model (SAM) - Open Sourced! Segment Anything (SAM) was released open sourced by facebook research. www.youtube.com/watch?v=KP0LGE5Qrlw

[10] Segment Anything by Meta AI, www.github.com/facebookresearch/segment-anything

[11] The Remarkable Story Behind The Most Important Algorithm Of All Time , *Fast Fourier Transform Veritasium* www.youtube.com/watch?v=nmgFG7PUHfo