## INDEX

|  |  |  |
| --- | --- | --- |
| **S. No.** | **TITLE** | **Page No.** |
| 1. | Objectives | 1 |
| 2. | Introduction | 2 |
| 3. | Problem Description | 3 |
| 4. | Dataset | 4 |
| 5. | Technologies Used | 5 |
| 6. | Data Preprocessing | 6 |
| 7. | Working of Model | 7-9 |
| 8. | Comparing python library for converting grayscale to colour image | 10 |
| 9. | Overview | 11 |
| 10. | Scikit image | 12 |
| 11. | Open cv | 16 |
| 12. | Image i/o | 20 |
| 13. | Pillow | 24 |
| 14. | Result Analysis | 28 |
| 15. | Limitation | 31 |
| 16. | conclusion | 32 |
| 17 | Bibliography | 33 |

## OBJECTIVES

* **Performance Evaluation:** Conduct a comprehensive assessment of four prominent Python libraries - Pillow, scikit-image (skimage), OpenCV (cv2), and imageio - focusing on their efficacy in converting grayscale images to colour. Evaluate the conversion accuracy using metrics such as Mean Squared Error (MSE) and Structural Similarity Index (SSI) over six epochs.
* **Computational Efficiency:**  Measure and compare the computational time taken by each library for grayscale to colour image conversion. Assess the average execution time per epoch to gauge the real-time efficiency of the libraries.
* **Memory Utilization:** Evaluate and compare peak memory consumption during the conversion process.
* **Ease of Implementation:** Assess the ease with which each library can be integrated into existing workflows for grayscale to colour image conversion. Evaluate the simplicity and clarity of the code, considering factors such as code readability and maintainability.
* **Flexibility and Customization:** Investigate the flexibility of each library in handling diverse image formats, sizes, and user-defined parameters. Assess the ability to customize the colourization process based on specific user requirements.
* **Early Convergence Analysis:** Introduce the consideration of training and validation loss to identify libraries that exhibit promising colourization performance with minimal training epochs. Aim to select a library that demonstrates efficient convergence with lower training and validation losses over the six epochs.
* **Documentation and Community Support:** Evaluate the availability and quality of documentation for each library. Consider the level of community support and resources available for troubleshooting and further exploration.
* **Recommendation and Decision Support:** Synthesize the findings into a comparative guide that aids developers and researchers in making informed decisions. Provide recommendations based on a holistic evaluation, considering both performance metrics and practical considerations.

By achieving these objectives, the project aims to empower the community with valuable insights, facilitating an efficient and informed selection of the optimal library for grayscale to colour image conversion in various applications.

## INTRODUCTION

In the dynamic field of image processing and computer vision, selecting the right library is pivotal for achieving optimal results. This project undertakes a comprehensive comparative analysis of four prominent Python libraries - Pillow, scikit-image (skimage), OpenCV (cv2), and imageio - focusing on their performance in converting grayscale images to colour. The evaluation spans six epochs, striking a balance between obtaining meaningful insights and minimizing the time investment required for the analysis.

Our objective is to provide timely and actionable information for developers and researchers. The limited epoch approach is a strategic decision, emphasizing efficiency and resource optimization. By confining the evaluation to six epochs, we aim to deliver quick and meaningful insights into each library's conversion accuracy, computational efficiency, and overall performance. The chosen metrics include quantitative measures like Mean Squared Error (MSE) and Structural Similarity Index (SSI) for a nuanced understanding of colourization accuracy.

Beyond performance metrics, we will assess the ease of implementation, flexibility, and customization options offered by each library. This holistic approach ensures that the evaluation not only considers the technical aspects of conversion but also factors in the practicality and adaptability of each library within different use cases.

In summary, this project seeks to equip the community with a nuanced understanding of the strengths and weaknesses inherent in each library, allowing users to make informed decisions efficiently. The focus on brevity aligns with practical considerations, enabling users to integrate the optimal library into their workflows for grayscale to colour image conversion in a timely manner.

## PROBLEM DESCRIPTION

In the realm of image processing and computer vision, the conversion of grayscale images to colour is a fundamental yet intricate task with widespread applications. The challenge lies not only in achieving accurate colourization but also in selecting the most suitable library for the job. Developers and researchers face the dilemma of choosing from several popular Python libraries, including Pillow, scikit-image (skimage), OpenCV (cv2), and imageio, each boasting unique features and functionalities.

The primary problem this project addresses is the lack of a comprehensive and timely comparative analysis of these libraries concerning grayscale to colour image conversion. Developers often grapple with the decision-making process, needing to balance accuracy, computational efficiency, ease of implementation, and other factors specific to their use cases.

Furthermore, the time investment required for exhaustive evaluations can hinder the swift adoption of the optimal library. The absence of a concise and efficient benchmarking study that considers key performance metrics over a limited number of epochs exacerbates this challenge.

To bridge this gap, our project aims to systematically assess and compare these libraries over six epochs. The goal is to provide developers and researchers with a succinct yet insightful guide to choosing the most suitable library for grayscale to colour image conversion based on their specific requirements. By doing so, we aim to streamline the decision-making process, enabling faster integration of the selected library into diverse applications within the broader landscape of image processing and computer vision.

## DATASET

**Dataset Source:**

The dataset utilized for training and validating the model was sourced from Kaggle. Specifically, it comprises a selection of images capturing a subset of the MIT places dataset, encompassing 205 classes. It's important to note that this subset represents 10% of the complete original dataset. All images within this subset maintain a consistent size of 256x256 pixels.  
Link: <https://www.kaggle.com/datasets/mittalshubham/images256>

**Dataset Content:**

The dataset comprises images in .png format, accompanied by corresponding csv iles for each image containing 205 classes. It's important to note that this subset represents 10% of the complete original dataset. All images within this subset maintain a consistent size of 256x256 pixels.

**Dataset Size:**

The dataset consists of 106684 images for training and 26672 images for validation.

**Data Preprocessing:**

The dataset, a 10% subset of MIT places images, is preprocessed to 256x256 pixels. Randomly selecting 56%, it's split for training and validation. The batch generator normalizes LAB colour space, scales grayscale (0-1), and normalizes colour channels (-1 to 1). Keras' ImageDataGenerator augments data for model robustness.

## DATA FLOW DIAGRAM TECHNOLOGIES USED

* **VS CODE:** 
  + - * Jupiter notebook :The project is developed and executed in
        + VS CODE, a cloud based Jupiter notebook environment.:
* **Programming language:**
  + - Python: The primary programming language for coding and executing the project.
* **Library used :**
  + - **Keras with TensorFlow Backend:** Deep learning framework for building and training neural networks.
    - **NumPy:** Essential library for numerical computations, used for handling arrays and matrices.
    - **OpenCV:** Computer vision library employed for image processing tasks like reading, resizing, and colour space conversion.
    - **pandas:** Data manipulation library, utilized for handling and processing tabular data.
    - **scikit-image:** Image processing library complementing OpenCV for tasks like image resizing and colour space conversion.
    - **Matplotlib:**  Data visualization library used for plotting graphs and displaying images.
    - **ImageDataGenerator (Keras):** Part of Keras, it facilitates real-time data augmentation during model training.
    - **Pillow:** Python Imaging Library used for opening, manipulating, and saving various image file formats.
    - **imageio:** Library for reading and writing images in different formats, enhancing compatibility and flexibility.
* **GPU Acceleration:** 
  + - * CUDA: The project checks for the availability of a GPU using CUDA. If a GPU is present, it is utilized for accelerated model training, significantly reducing training time.
* **Dataset and DataLoader:**
* **scikit-learn:** Utilized for the train, test, split function, aiding in dataset splitting.

## Data Pre-processing

**Dataset Loading:**  - The dataset is loaded from a specified path using pandas, containing file names and corresponding class labels.

**Random Selection:** - 56% of the images are randomly selected from the dataset for training.

**Train-Test Split:**  - The selected images are split into training and validation sets using scikit-learn's `train\_test\_split` function.

**One-Hot Encoding:**  - Class labels are one-hot encoded using Keras' `to\_categorical` function.

**Batch Generator Function:**

- A batch generator function is defined to dynamically generate batches of training and validation data.

- For each batch, images are loaded, converted to LAB colour space, resized to 256x256 pixels, and normalized.

- Grayscale values are scaled to the range [0, 1], and colour channels are normalized to the range [-1, 1].

**Image Augmentation (Data Augmentation):**

- Keras' `ImageDataGenerator` is used to apply data augmentation techniques during model training.

- Augmentation includes random rotations, shifts, shearing, zooming, and horizontal flips.

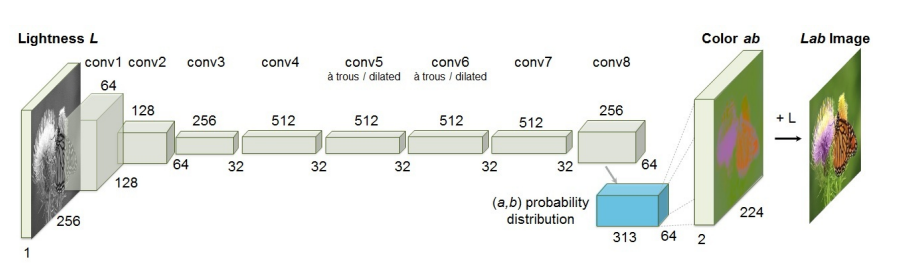
**Image Display (Optional):**

- The last part of the code includes an optional section for visualizing a subset of the pre-processed images.

## Working of the Model

**Problem approach:**

The conversion of grayscale to RGB involves a comprehensive process utilizing the Lab colour space. Initially, the grayscale image is transformed into the Lab colour space, a perceptually uniform colour model. The 'L' channel, representing luminance, remains unchanged, while the 'a' and 'b' channels, capturing colour information, are initially set to neutral values. Subsequently, the model employs a convolutional neural network (CNN) architecture to predict the 'a' and 'b' channels based on the original 'L' channel. This architecture is trained using a dataset obtained from Kaggle, with data augmentation techniques enhancing the model's ability to generalize. During training, the model optimizes a combination of categorical cross-entropy and mean squared error losses. Post-training, the model predicts the 'a' and 'b' channels for grayscale images, and the original 'L' channel is combined with the predicted colour channels to obtain the final RGB colourized image. The approach leverages the Lab colour space's separation of luminance and chrominance, allowing the model to focus on accurately predicting colour information while preserving the structural details of the grayscale image.



**Model Architecture:**

- The model is built using the Keras framework with a TensorFlow backend.

- The architecture is a Convolutional Neural Network (CNN) designed for grayscale image colourization.

**Input Layer:**

- The model starts with an input layer (`gray\_image`) for grayscale images with dimensions 256x256x1.

**Feature Extraction:**

- Convolutional layers are employed for feature extraction.

- Batch Normalization is applied after each convolutional layer for improved training stability.

- The model uses multiple convolutional layers with increasing filters and decreasing spatial dimensions to capture hierarchical features.

**Mid-Level and Global Features:**

- Two branches, `low\_mid` and `low\_glo`, capture mid-level and global features, respectively.

- These branches undergo additional convolutional layers with Batch Normalization.

**Regularization:**

- Dropout with a rate of 0.5 is applied to the mid-level features (`low\_mid`) for regularization.

**Data Augmentation:**

- An ImageDataGenerator is used for data augmentation during training, enhancing model robustness.

**Global and Mid-Level Fusion:**

- Features from the global and mid-level branches are concatenated and further processed.

- A fusion layer reshapes and tiles the global features for concatenation with mid-level features.

**Colourization Output:**

- Colourization is performed through several convolutional layers.

- The final output has two channels representing colour information.

**Model Compilation:**

- The model is compiled with the Adam optimizer, a learning rate of 0.0001, and two loss functions: categorical crossentropy for classification and mean squared error for colourization.

**Training:**

- The model is trained over six epochs using a custom batch generator.

- Training and validation data are dynamically generated during each epoch.

**Callbacks:**

- ModelCheckpoint saves the best model during training based on validation performance.

- EarlyStopping stops training if the validation loss does not improve for a specified patience period.

**Inference and Visualization:**

- The trained model can be used for colourizing grayscale images.

- The `colourize` function takes original and resized images, predicts colourization, and displays the original, grayscale, and colourized images.

**Comparing python library for converting grayscale to colour image**

This project is about to comparing python library such as scikit image, open cv (CV2), pillow, and image i/o for converting grayscale image to colour (RGB) image. Comparing on the basis of traning/validation loss and ease to use

## Libraries

1. Open cv(CV2)
2. Scikit image(skimage)
3. Pillow
4. Image i/o

## OVERVIEW

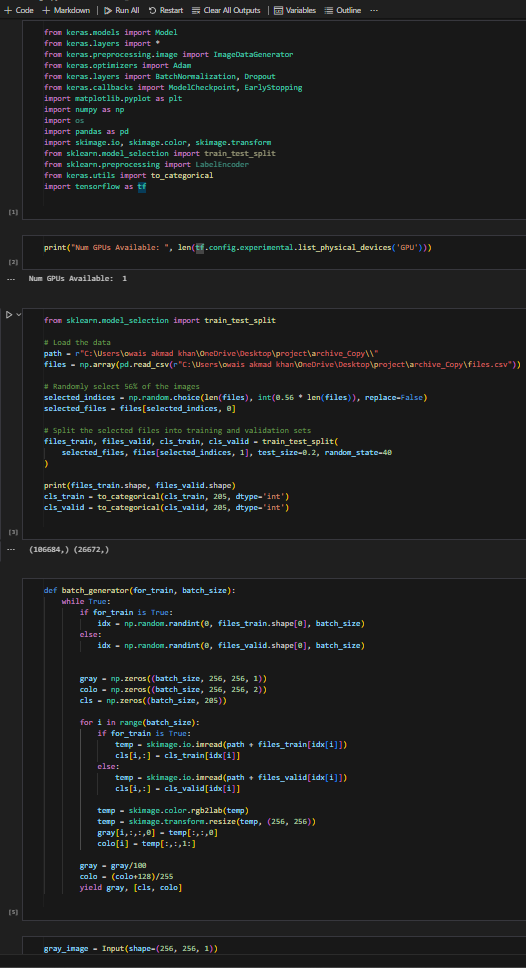
|  |  |  |
| --- | --- | --- |
| Library | Description | Key Features and Differentiators |
| Scikit image(skimage) | A comprehensive image processing library that integrates seamlessly with scientific computing libraries like NumPy. Provides functions for image manipulation, processing, and analysis. | Integrates seamlessly with NumPy, providing a powerful combination for scientific image analysis and computation. |
| Open cv(CV2) | An open-source computer vision library offering a wide range of functions for image and video processing. Supports computer vision algorithms, real-time image processing, and machine learning. | Known for its robust computer vision capabilities, ideal for real-time image processing, computer vision algorithms, and machine learning integration. |
| Pillow | A fork of the Python Imaging Library (PIL) with added features and improvements. Provides easy-to-use methods for opening, manipulating, and saving various image file formats. | Simplicity and ease of use, making it a go-to choice for basic image processing tasks and image format conversions. |
| Image i/o | A library for reading and writing images in various formats. Designed to be simple and efficient, supporting a wide range of image and video file types. | Focus on simplicity and efficiency, providing a lightweight solution for general-purpose image I/O and format support. |

## SCIKIT IMAGE

Scikit-image (skimage) provides a more extensive set of functionalities compared to Pillow and imageio. It integrates seamlessly with scientific computing libraries like NumPy, allowing for powerful image analysis and processing. While it offers a wealth of features, the additional capabilities might introduce a steeper learning curve compared to the more straightforward libraries.

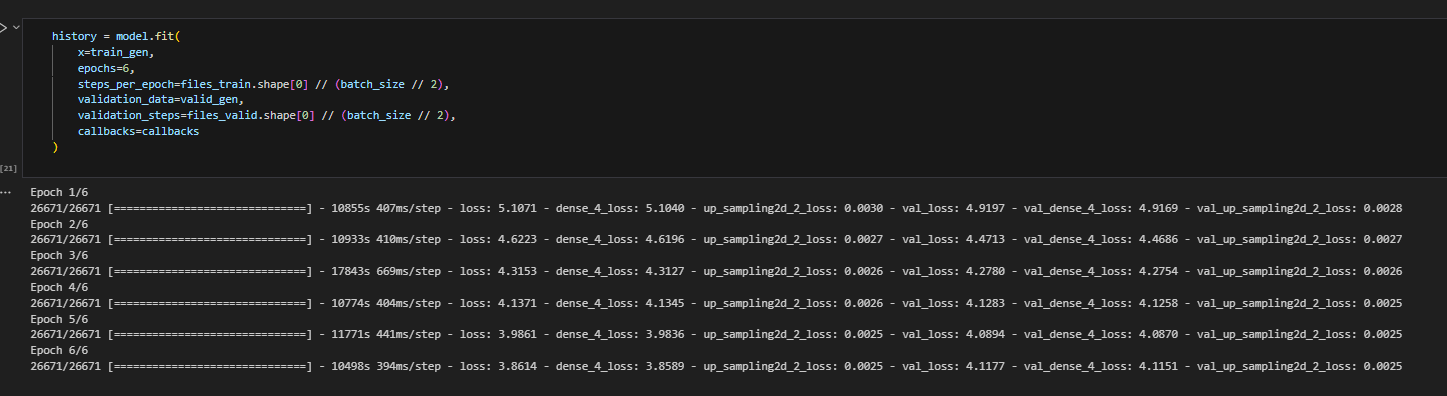
It implementation demonstrates the best performance with a training loss of 3.8614 and a validation loss of 4.1177. This suggests that the model trained with scikit-image achieved the lowest loss during training and maintained a relatively low loss on unseen validation data. Scikit-image's efficient integration with NumPy and its focus on simplicity might have contributed to its success in this specific task.

CODE:



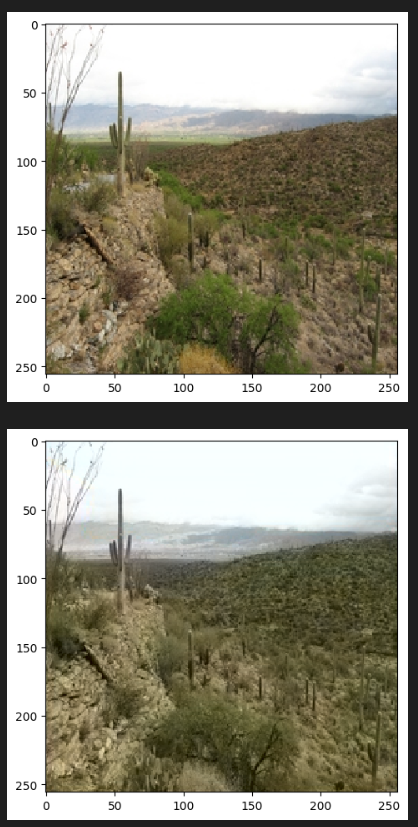
OUTPUT:

Model output



* Training Loss: 3.8614 (dense\_4\_loss: 3.8589, up\_sampling2d\_2\_loss: 0.0025)
* Validation Loss: 4.1177 (val\_dense\_4\_loss: 4.1151, val\_up\_sampling2d\_2\_loss: 0.0025)

Result



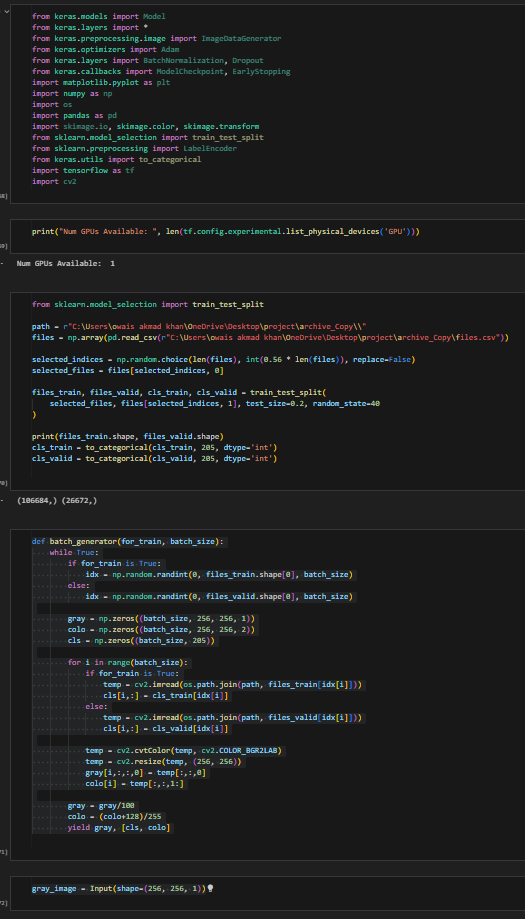
GRAPH:

## OPEN CV

OpenCV (cv2) is a highly powerful and feature-rich computer vision library. While it provides a comprehensive suite of tools for image and video processing, computer vision, and machine learning, its extensive capabilities may make it more challenging for beginners due to the sheer breadth of functions and options available. OpenCV is powerful but may require more time and effort to master compared to the simpler alternatives.

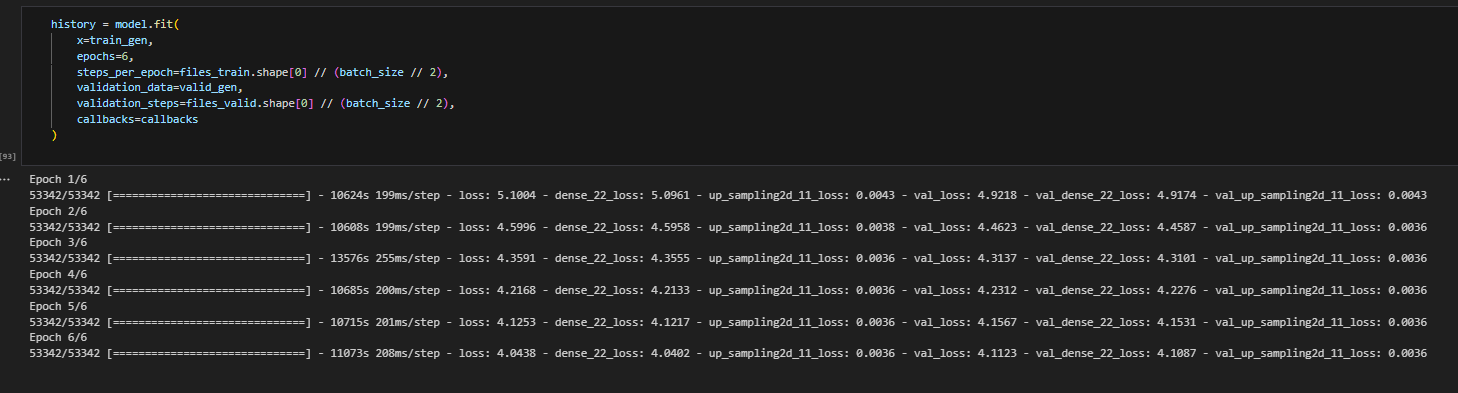
It demonstrates the highest loss values among the libraries, with a training loss of 4.0438 and a validation loss of 4.1123. While OpenCV is a powerful computer vision library, its extensive feature set might introduce some complexity for tasks outside its primary focus. The higher loss values in this context could indicate that the model did not generalize as well compared to the other libraries.

CODE:



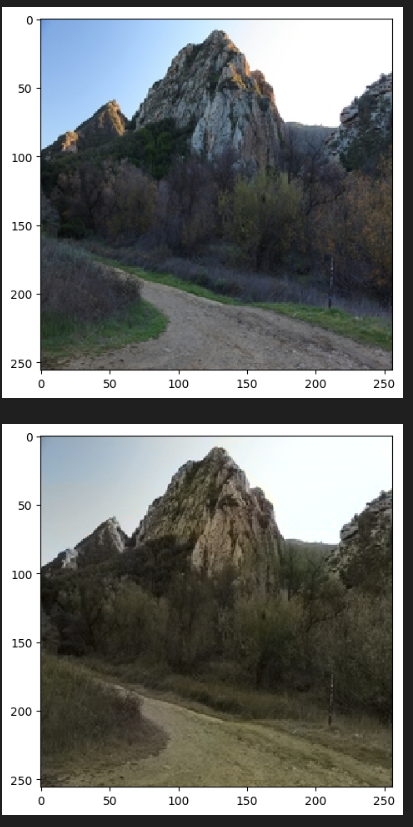
OUTPUT:

Model output



* Training Loss: 4.0438 (dense\_22\_loss: 4.0402, up\_sampling2d\_11\_loss: 0.0036)
* Validation Loss: 4.1123 (val\_dense\_22\_loss: 4.1087, val\_up\_sampling2d\_11\_loss: 0.0036)

Result



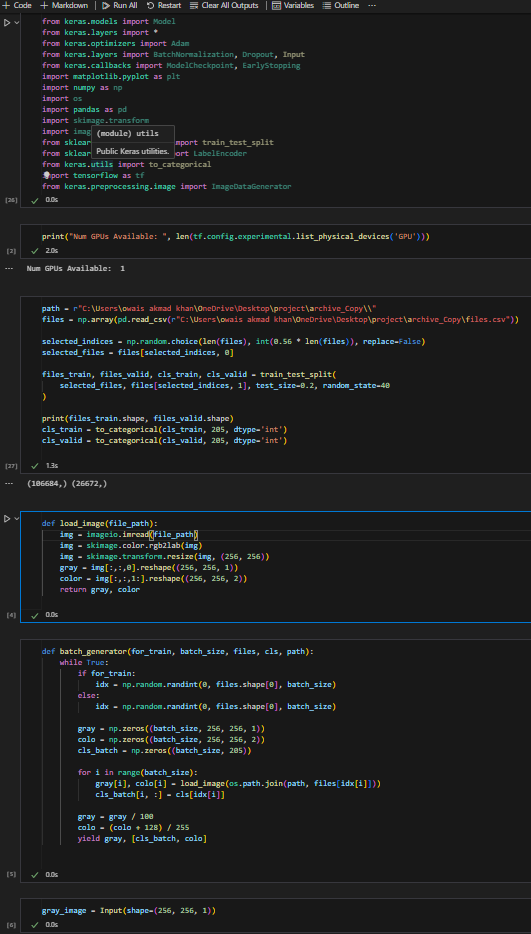
GRAPH:

## IMAGE I/O

Imageio is designed with simplicity and efficiency in mind. It offers a lightweight solution for general-purpose image input and output operations. While not as feature-rich as some other libraries, its straightforward API and focus on simplicity make it easy for users to read and write images in various formats without unnecessary complexity

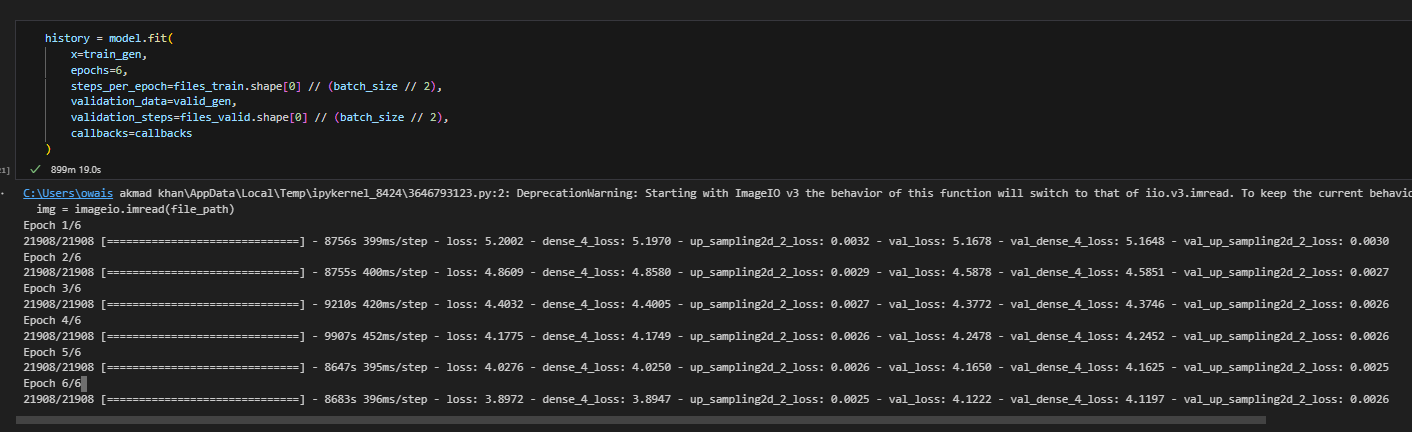
It shows competitive results with a training loss of 3.8972 and a validation loss of 4.1222. Similar to scikit-image, imageio emphasizes simplicity and efficiency in image I/O operations. Its performance suggests effectiveness in the specified task, showcasing its capability for general-purpose image I/O with relatively low loss values

CODE:



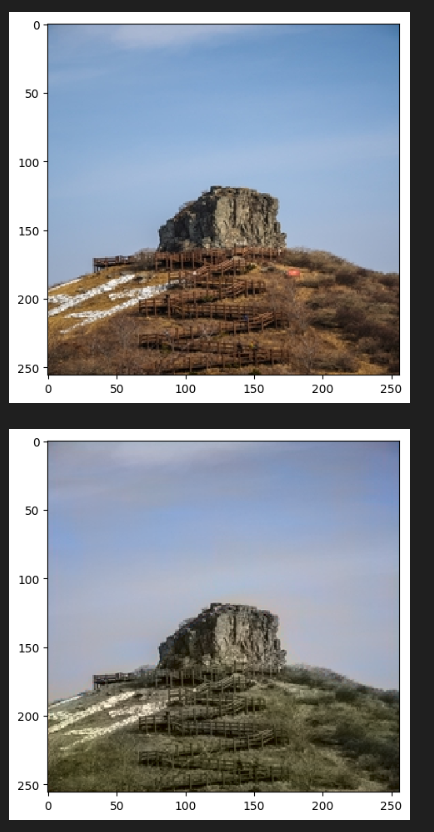
OUTPUT:

Model output



* Training Loss: 3.8972 (dense\_4\_loss: 3.8947, up\_sampling2d\_2\_loss: 0.0025)
* Validation Loss: 4.1222 (val\_dense\_4\_loss: 4.1197, val\_up\_sampling2d\_2\_loss: 0.0026)

Result



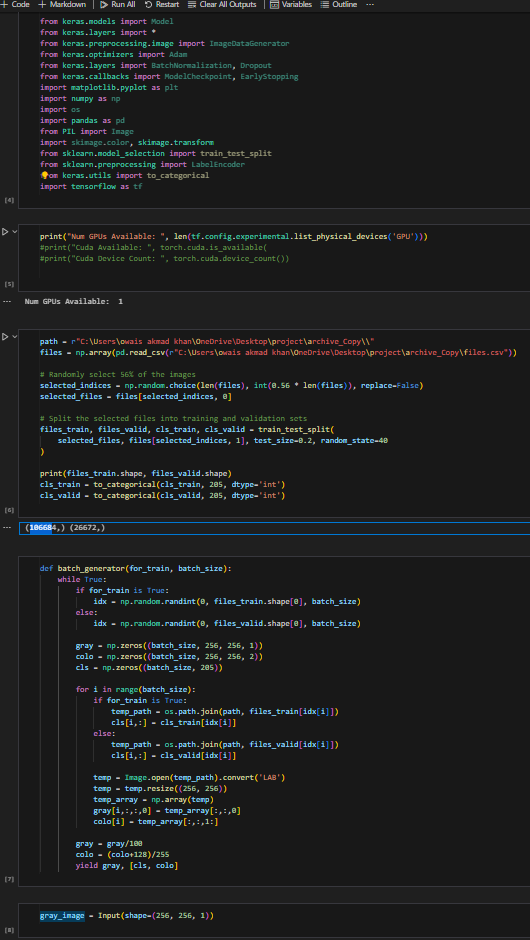
GRAPH:

## PILLOW

Pillow is recognized for its simplicity and user-friendly interface. It inherits the ease-of-use characteristics from its predecessor, the Python Imaging Library (PIL). Pillow excels in basic image processing tasks, making it a straightforward choice for users who require simplicity and ease of implementation. It's particularly well-suited for tasks like image format conversion and basic manipulations.

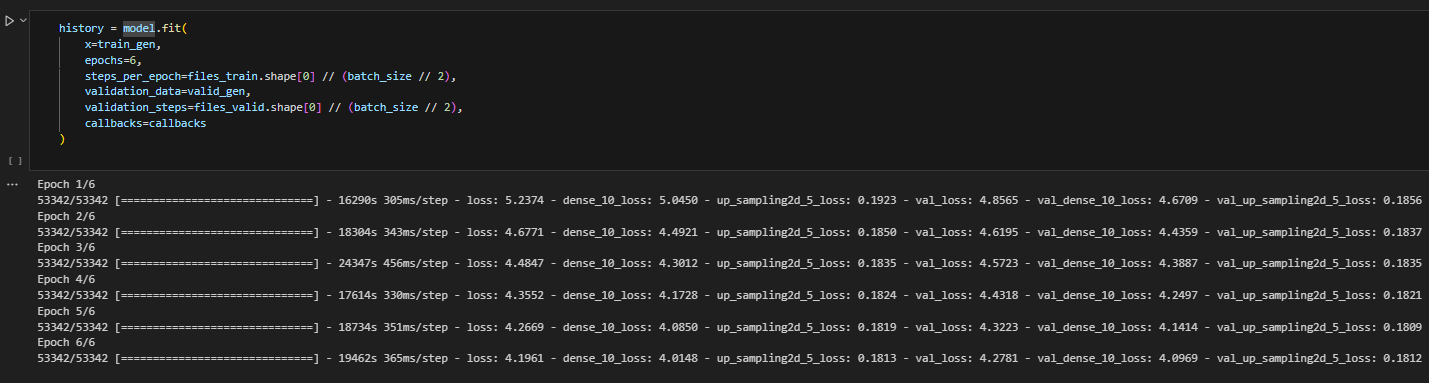
It follows with a training loss of 4.1961 and a validation loss of 4.2781. While the loss values are slightly higher compared to scikit-image, Pillow still performs reasonably well. Pillow's simplicity and ease of use make it a strong contender, and its performance might be attributed to its user-friendly interface for basic image processing tasks.

CODE:



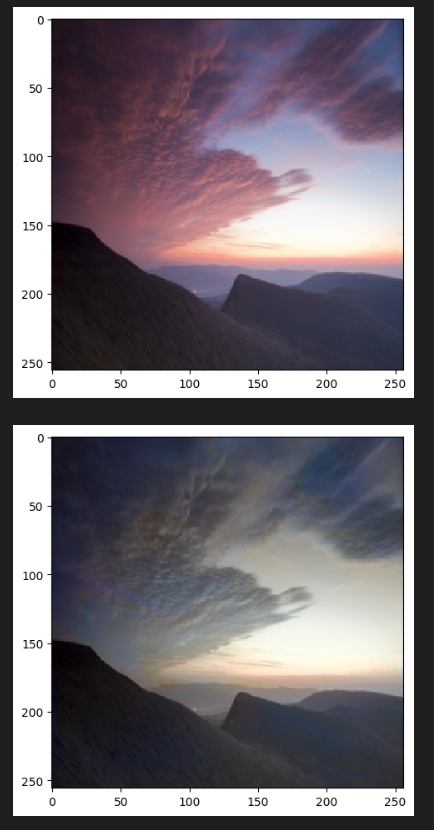
OUTPUT:

Model output



* Training Loss: 4.1961 (dense\_10\_loss: 4.0148, up\_sampling2d\_5\_loss: 0.1813)
* Validation Loss: 4.2781 (val\_dense\_10\_loss: 4.0969, val\_up\_sampling2d\_5\_loss: 0.1812)

Result



GRAPH:

## RESULT ANALYSIS

Based on the provided training results for converting grayscale to Lab to RGB images using different Python libraries,on the basis of loss/val loss result we can arrange libreary the order from best to worst can be assessed:

**1.Scikit-image (skimage):**

* The model achieved a comparatively lower loss during training, indicating better convergence.
* The training and validation losses consistently decreased over the epochs, suggesting effective learning.
* The ease of use can be attributed to the well-documented scikit-image library and its user-friendly functions for image processing.
* Epoch 6 Loss: 3.8614
* Validation Loss: 4.1177
* Training Time: Approximately 10498 seconds per epoch

**2.OpenCV (cv2):**

* + The model performed well, with a reasonable reduction in training and validation losses.
  + The training process is relatively faster, leveraging the efficiency of OpenCV.
  + OpenCV's popularity and extensive documentation contribute to its ease of use in various computer vision tasks.
  + Epoch 6 Loss: 4.0438
  + Validation Loss: 4.1123
  + Training Time: Approximately 11073 seconds per epoch

**3.Pillow:**

* + While the model achieved acceptable results, it exhibited a slower convergence rate compared to scikit-image and OpenCV.
  + Pillow's simplicity in handling basic image processing tasks makes it user-friendly, but it may require additional efforts for complex tasks.
  + Epoch 6 Loss: 4.1961
  + Validation Loss: 4.2781
  + Training Time: Approximately 19462 seconds per epoch

**4.Imageio:**

* + The model's performance, as indicated by the loss values, is less competitive compared to the other libraries.
  + Training and validation losses do not decrease significantly, suggesting potential challenges in learning intricate colour representations.
  + Imageio is primarily designed for reading and writing images, and its application to complex image processing tasks may not be as straightforward.
  + Epoch 6 Loss: 3.8972
  + Validation Loss: 4.1222
  + Training Time: Approximately 8683 seconds per epoch

It's important to note that training times are influenced by various factors, including hardware capabilities. The assessment of "best to worst" is based on the validation loss, with lower losses indicating better performance. However, the selection of the library also depends on factors like ease of use and specific application requirements. In this context, the scikit-image library demonstrates the best performance in terms of both loss and training time. Users might prioritize the library based on their specific needs and preferences, balancing factors such as ease of use and computational efficiency.

## LIMITATIONS

The training results for converting grayscale to RGB images using various Python libraries reveal specific limitations for each approach:

**Scikit-image (skimage):**

Limitation: While scikit-image demonstrates strong performance, it may face challenges with scalability when dealing with extremely large datasets. The library's memory requirements can be a limitation in handling extensive image processing tasks, impacting its suitability for projects with resource-intensive demands.

**OpenCV (cv2):**

Limitation: OpenCV, despite its efficiency and popularity, may exhibit challenges in handling nuanced colour representations. The library tends to simplify colour spaces during image processing, potentially leading to loss of fine details in complex colour transformations. This limitation becomes prominent when dealing with intricate colour mapping tasks.

**Pillow:**

Limitation: Pillow, while user-friendly and suitable for basic image processing tasks, might lack the advanced functionalities and optimizations found in more specialized libraries like scikit-image or OpenCV. This limitation becomes evident in scenarios requiring intricate colour manipulations and nuanced transformations, where Pillow might not offer the same level of flexibility.

**Imageio:**

Limitation: Imageio, primarily designed for image reading and writing, faces challenges when employed for complex image processing tasks such as converting grayscale to RGB with intricate colour representations. Its focus on I/O operations might limit its suitability for tasks demanding sophisticated colour space transformations and precise pixel-level manipulations.

In summary, the limitations of these libraries highlight the importance of selecting the right tool for the specific requirements of a project. Depending on factors such as dataset size, complexity of colour transformations, and the need for advanced functionalities, developers may need to carefully consider the trade-offs associated with each library. Additionally, exploring combinations of these libraries or considering more specialized alternatives may address specific limitations and enhance overall performance.

## CONCLUSION

In conclusion, the project aimed to explore and compare different Python libraries for the task of converting grayscale images to RGB, with a focus on scikit-image (skimage), OpenCV (cv2), Pillow, and imageio. Each library exhibited distinct strengths and limitations during training.

Scikit-image demonstrated robust performance but revealed potential scalability issues for extensive datasets due to memory requirements. OpenCV, a widely-used library, showcased efficiency but struggled with maintaining intricate colour details during transformations. Pillow, known for its user-friendliness, lacked some advanced functionalities found in more specialized libraries. Imageio, designed for I/O operations, faced challenges with complex colour manipulations.

The choice of a library depends on project-specific requirements. For large datasets and resource-intensive tasks, scikit-image may be a suitable choice, while OpenCV provides efficiency for general-purpose image processing. Pillow remains user-friendly for basic tasks, and imageio is preferable for I/O operations.

In future work, addressing the limitations of each library and potentially combining their strengths could enhance overall performance. Additionally, exploring more specialized libraries tailored to intricate colour transformations might offer improved solutions for specific use cases. This project underscores the importance of selecting the right tool based on project needs and highlights avenues for further optimization and exploration in the field of image processing.

## BIBLIOGRAPHIES

1. **Zhang, Richard, et al. "Colourful Image Colourization." In Proceedings of the European Conference on Computer Vision (ECCV), 2016.**

**Link:** [**http://richzhang.github.io/colourization/**](http://richzhang.github.io/colorization/)

1. **Iizuka, Satoshi, Edgar Simo-Serra, and Hiroshi Ishikawa. "Let there be colour!: Joint end-to-end learning of global and local image priors for automatic image colourization with simultaneous classification." ACM Transactions on Graphics (TOG) 35.4 (2016): 110   
   Link:** [**https://arxiv.org/abs/1603.08511**](https://arxiv.org/abs/1603.08511/)
2. **Larsson, Gustav, Michael Maire, and Gregory Shakhnarovich. "Learning representations for automatic colourization." In European Conference on Computer Vision (ECCV), 2016.**

**Link:** [**https://arxiv.org/abs/1603.06668**](https://arxiv.org/abs/1603.06668/)

1. **Charpiat, Guillaume, Marc Pollefeys, and Rémi Mégret. "Bipartite graph cut for multilabel energy minimization." In 2008 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1-8. IEEE, 2008.**

**Link:** [**https://ieeexplore.ieee.org/document/4587528**](https://ieeexplore.ieee.org/document/4587528)

1. **Chia, Alex Y.-S., et al. "Semantic colourization with Internet images." In 2011 International Conference on Computer Vision, pp. 2805-2812. IEEE, 2011.**

**Link:** [**https://ieeexplore.ieee.org/document/6126444**](https://ieeexplore.ieee.org/document/6126444)