CHAPTER 1
INTRODUCTION

1.1 About Project: -

In image colorization, our goal is to produce a colored image given a grayscale input image. This problem is challenging because it is grayscale image may correspond to many plausible colored images. As a result, traditional models often relied on significant

Deep neural networks have shown remarkable success in automatic image colorization -- going from grayscale to color with no additional human input. This success may in part be due to their ability to capture and use semantic information (i.e. what the image actually is) in colorization. In recent years, CNNs have emerged as the de facto standard for solving image classification problems, achieving error rates lower than 4% in the ImageNet challenge. CNNs owe much of their success to their ability to learn and discern colors, patterns, and shapes within images and associate them with object classes.

Given a grayscale photograph as input, this paper attacks the problem of hallucinating a plausible color version of the photograph. This problem is clearly underconstrained, so previous approaches have either relied on significant user interaction or resulted in desaturated colorizations. We propose a fully automatic approach that produces vibrant and realistic colorizations. We embrace the underlying uncertainty of the problem by posing it as a classification task and use class-rebalancing at training time to increase the diversity of colors in the result.

At first glance, hallucinating their colors seems daunting, since so much of the information (two out of the three dimensions) has been lost. Looking more closely, however, one notices that in many cases, the semantics of the scene and its surface texture provide ample cues for many regions in each image: the grass is typically green, the sky is typically blue.

1.2 Project Objectives :-

Changing the grayscale image into a color image using deep learning model. We present a convolutional-neural-network-based system that faithfully colorizes black and white photographic images without direct human assistance. We explore various network architectures, objectives, color spaces, and problem formulations.

CHAPTER 2

SOFTWARE & HARDWARE

2.1 Software:-

Software: - Software used for Coding, Hosting and Database are given below:

For Developing:

2.1.1 Anaconda Navigator

2.1.2 Python 3.72

2.1.3 OpenCV == 3.0

2.1.4 Django == 2.2.6

2.1.5 Numpy == 1.13.3

2.1.6 Pandas == 0.21.1

2.2 Hardware:-

Hardware: -

1.	Processor	:	2.8 GHZ or above
2.	RAM	:	8GB or above
3.	HDD	:	512GB or above

CHAPTER 3
PROBLEM DESCRIPTION

3.1 Problem Statement :-

Image colorization is the process of taking an **input grayscale (black and white) image** and then producing an **output colorized image** that represents the semantic color and tones of the input (for example, an ocean on a clear sunny day must be plausibly "blue" — it can't be colored "hot pink" by the model).

3.1.1 Previous methods for image colorization either:

3.1.1.1 Relied on significant human interaction and annotation

3.1.1.2 Produced desaturated colorization

Previous approaches to black and white image colorization relied on *manual human annotation* and often produced desaturated results that were not "believable" as true colorizations. Evaluating synthesized images is notoriously difficult [4]. Since our ultimate goal is to make results that are compelling to a human observer, we introduce a novel way of evaluating colorization results, directly testing their perceptual realism.

This test demonstrates that in many cases, our algorithm is producing nearly photorealistic results. We also show that our system's colorizations are realistic enough to be useful for downstream tasks.

CHAPTER-4
LITERATURE AND SURVEY

4.1 Scope:-

The scope of the application is limited to desktop application right now. The application is targeted towards providing free web app service for the user.

4.2 Benefits:-

There are lots of benefits in Automatic Image Colorization some of these are:

4.2.1 Colorizing black and white Image.

4.2.2 Colorizing night vision images in Military applications.

4.2.3 Colorizing images to help in decorations.

4.2.4 Colorizing Ultrasonic images.

CHAPTER-5
SOFTWARE REQUIREMENT SPECIFICATION

5.1 Functional Requirement:-

5.1.1 Image Name Panel.

5.1.2 Selecting Image Panel and Upload Button.

5.2 Non-Functional Requirement:

A non-functional requirement is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors. They are contrasted with functional requirements that define specific behavior or functions. A careful specification and adherence of non-functional requirements such as performance, security, privacy & availability are crucial to the success or failure of any software system. The correct specification and adherence of non-functional requirements similarly plays at least an equal, if not a greater role in the success of mobile applications.

5.2.1 INTERFACE REQUIREMENT

How will the system interface work with its environment, users and other systems.

e.g. user-friendliness, simple and interactive.

5.2.2 PERFORMANCE REQUIREMENT

Time/space bounds, such as workloads, response time, throughput and available storage space.

5.2.3 SECURITY

Permissible access to data and operations. e.g user login.

CHAPTER-6
SOFTWARE DESIGN

6.1 Use Case Diagram:-

Use case diagram is a behavioral UML diagram type and frequently used to analyze various systems. They enable you to visualize the different types of roles in a system and how those roles interact with the system. This use case diagram tutorial will cover the following topics and help you create use case better.

6.2 Importance of Use Case Diagrams:-

As mentioned before use case diagrams are used to gather a usage requirement of a system. Depending on your requirement you can use that data in different ways. Below are few ways to use them.

6.2.1 **To identify functions and how roles interact with them** – The primary purpose of use case diagrams.

6.2.2 **For a high-level view of the system** – Especially useful when presenting to managers or stakeholders. You can highlight the roles that interact with the system and the functionality provided by the system without going deep into inner workings of the system.

6.2.3 **To identify internal and external factors** – This might sound simple but in large

6.2.4

6.3 Use Case Diagram objects

6.3.1 Use case diagrams consist of 4 objects.

6.3.2 Actor

6.3.3 Use case

6.3.4 System

6.3.5 Package

6.3.1 Actor

Actor in a use case diagram is any entity that performs a role in one given system. This could be a person, organization or an external system and usually drawn like skeleton shown below

6.3.2 Use Case

A use case represents a function or an action within the system. It’s drawn as an oval and named with the function.

6.3.3 System

The system is used to define the scope of the use case and drawn as a rectangle. This an optional element but useful when you’re visualizing large systems. For example, you can create all the use cases and then use the system object to define the scope covered by your project. Or you can even use it to show the different areas covered in different releases.

6.3.4 Package

The package is another optional element that is extremely useful in complex diagrams. Similar to class diagrams, packages are used to group together use cases. They are drawn like the image shown below.

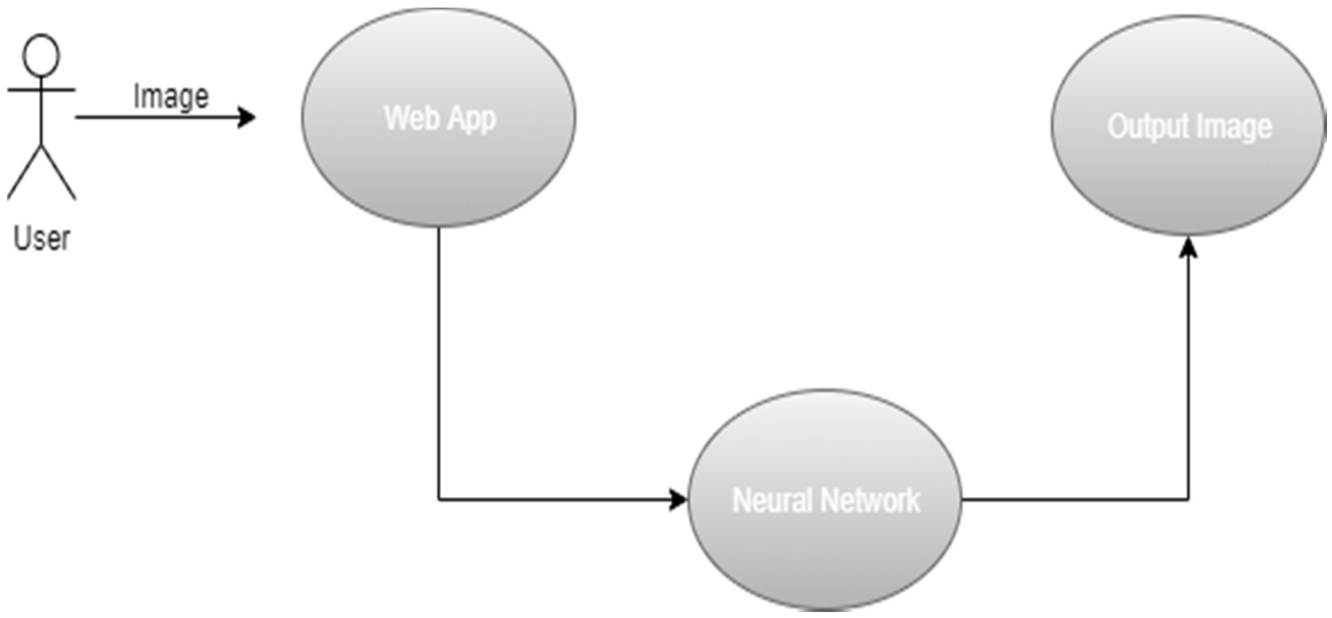


Fig. 6.1 Use Case Diagram

CHAPTER-7
DEPLOYMENT

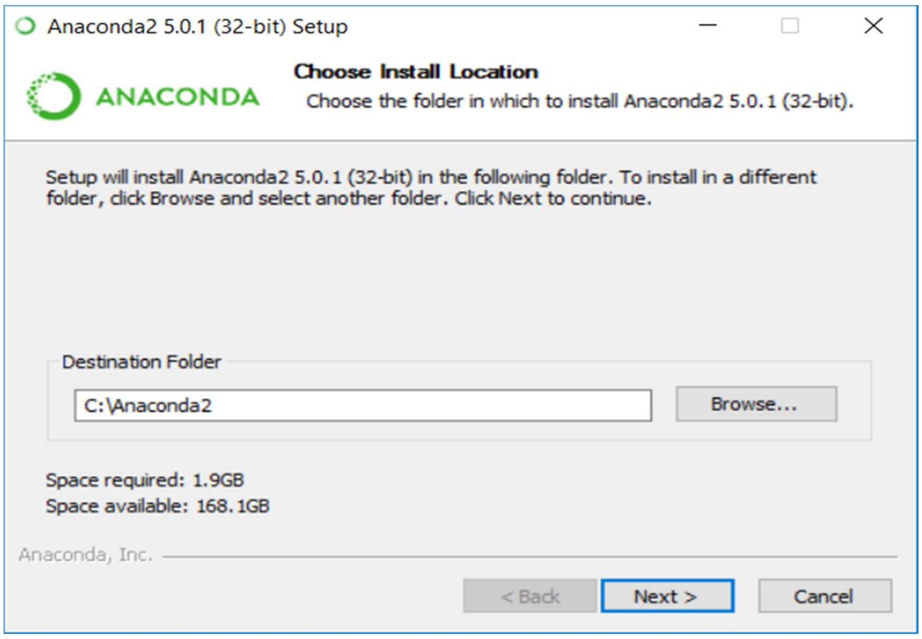
7.1 ANACONDA

Anaconda Cloud is a package management service that makes it easy to find, access, store and share public notebooks, environments, and conda and PyPI packages.

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text. Uses include: data cleaning and transformation, numerical simulation, statistical modeling, data visualization, machine learning, and much more. For installation- Install Anaconda. Anaconda conveniently installs Python, the Jupyter Notebook, and other commonly used packages for scientific computing and data science.

Use the following for installation steps:

1. Download Anaconda. Install the version of Anaconda which you downloaded, following the instructions on the download page.
2. We have installed Jupyter Notebook. To run the notebook: Jupyter notebook.



Anaconda2 5.0.1 (32-bit) Setup

Choose Install Location

Choose the folder in which to install Anaconda2 5.0.1 (32-bit).

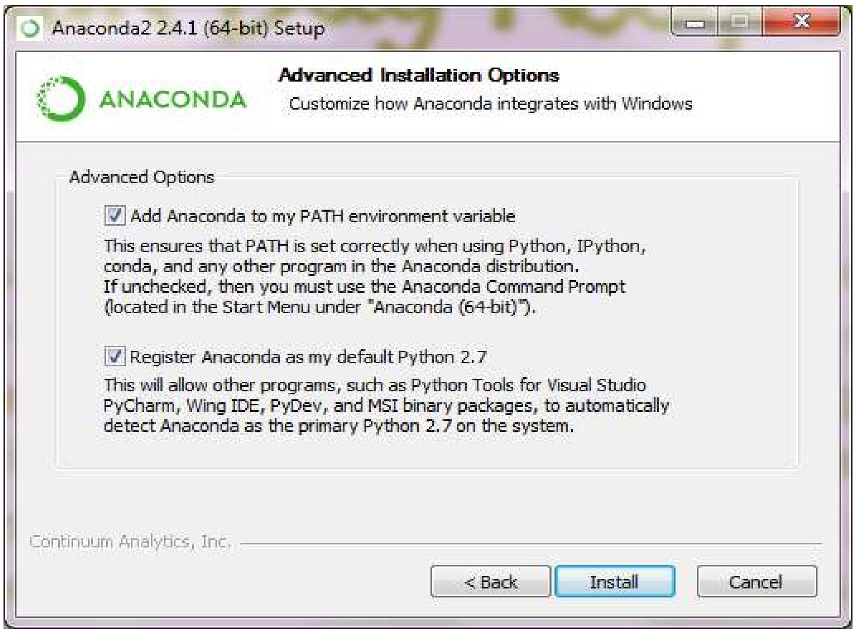
Setup will install Anaconda2 5.0.1 (32-bit) in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder: C:\Anaconda2

Space required: 1.9GB
Space available: 168.1GB

Anaconda, Inc.

Fig 7.1 Anaconda Location



Anaconda2 2.4.1 (64-bit) Setup

ANACONDA

Advanced Installation Options

Customize how Anaconda integrates with Windows

Advanced Options

[✓] Add Anaconda to my PATH environment variable

This ensures that PATH is set correctly when using Python, IPython, conda, and any other program in the Anaconda distribution. If unchecked, then you must use the Anaconda Command Prompt (located in the Start Menu under "Anaconda (64-bit)").

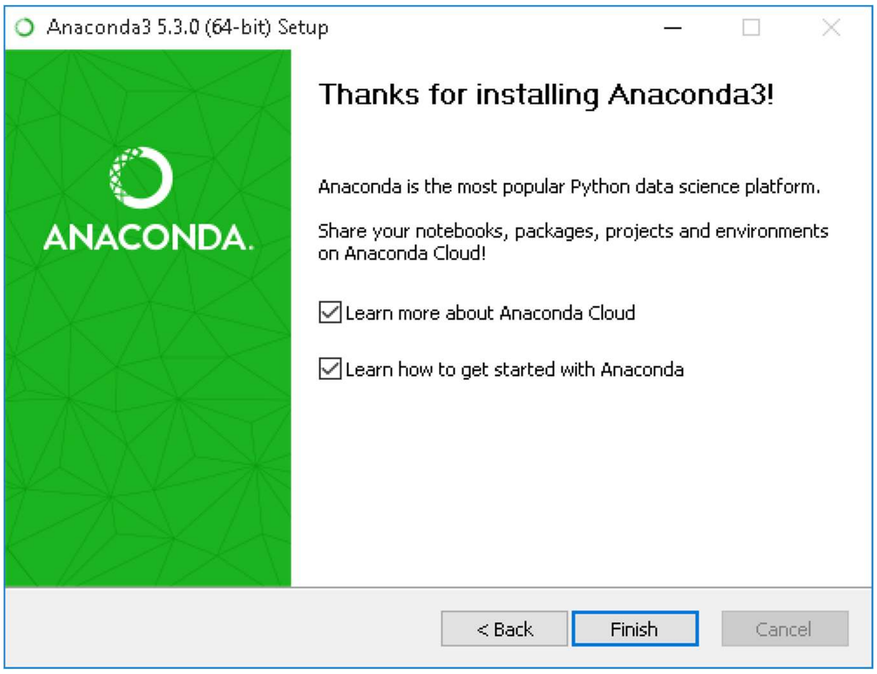
[✓] Register Anaconda as my default Python 2.7

This will allow other programs, such as Python Tools for Visual Studio PyCharm, Wing IDE, PyDev, and MSI binary packages, to automatically detect Anaconda as the primary Python 2.7 on the system.

Continuum Analytics, Inc.

< Back | Install | Cancel

Fig 7.2 Anaconda Path



Anaconda3 5.3.0 (64-bit) Setup

ANACONDA.

Thanks for installing Anaconda3!

Anaconda is the most popular Python data science platform.

Share your notebooks, packages, projects and environments on Anaconda Cloud!

[✓] Learn more about Anaconda Cloud

[✓] Learn how to get started with Anaconda

< Back | Finish | Cancel

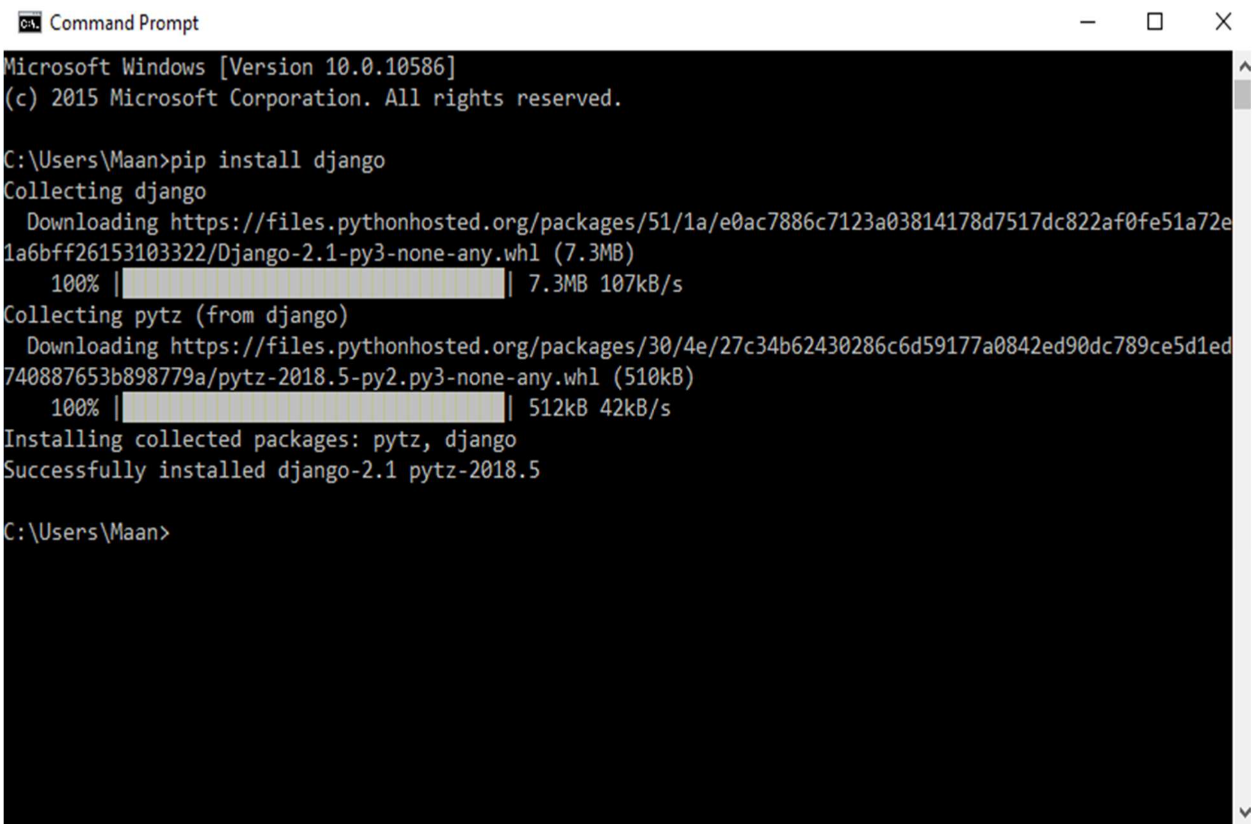
Fig 7.3 Anaconda Final Step

7.2 Django

With Django, you can take Web applications from concept to launch in a matter of hours. Django takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It’s free and open source.

Django is a Python web framework, thus requiring Python to be installed on your machine. And as we have already installed the Anaconda Navigator that time we have to install the python.

Press Strat button and then type cmd then:



```
Microsoft Windows [Version 10.0.10586]
(c) 2015 Microsoft Corporation. All rights reserved.

C:\Users\Maan>pip install django
Collecting django
  Downloading https://files.pythonhosted.org/packages/51/1a/e0ac7886c7123a03814178d7517dc822af0fe51a72e
1a6bff26153103322/Django-2.1-py3-none-any.whl (7.3MB)
    100% |████████████████████████████████| 7.3MB 107kB/s
Collecting pytz (from django)
  Downloading https://files.pythonhosted.org/packages/30/4e/27c34b62430286c6d59177a0842ed90dc789ce5d1ed
740887653b898779a/pytz-2018.5-py2.py3-none-any.whl (510kB)
    100% |████████████████████████████████| 512kB 42kB/s
Installing collected packages: pytz, django
Successfully installed django-2.1 pytz-2018.5

C:\Users\Maan>
```

Fig 7.4 Django Setup

CHAPTER-8
OUTPUT SCREEN

In resource Module of shipyard there are many entities like:-

1. Name
2. Choose image
3. Upload

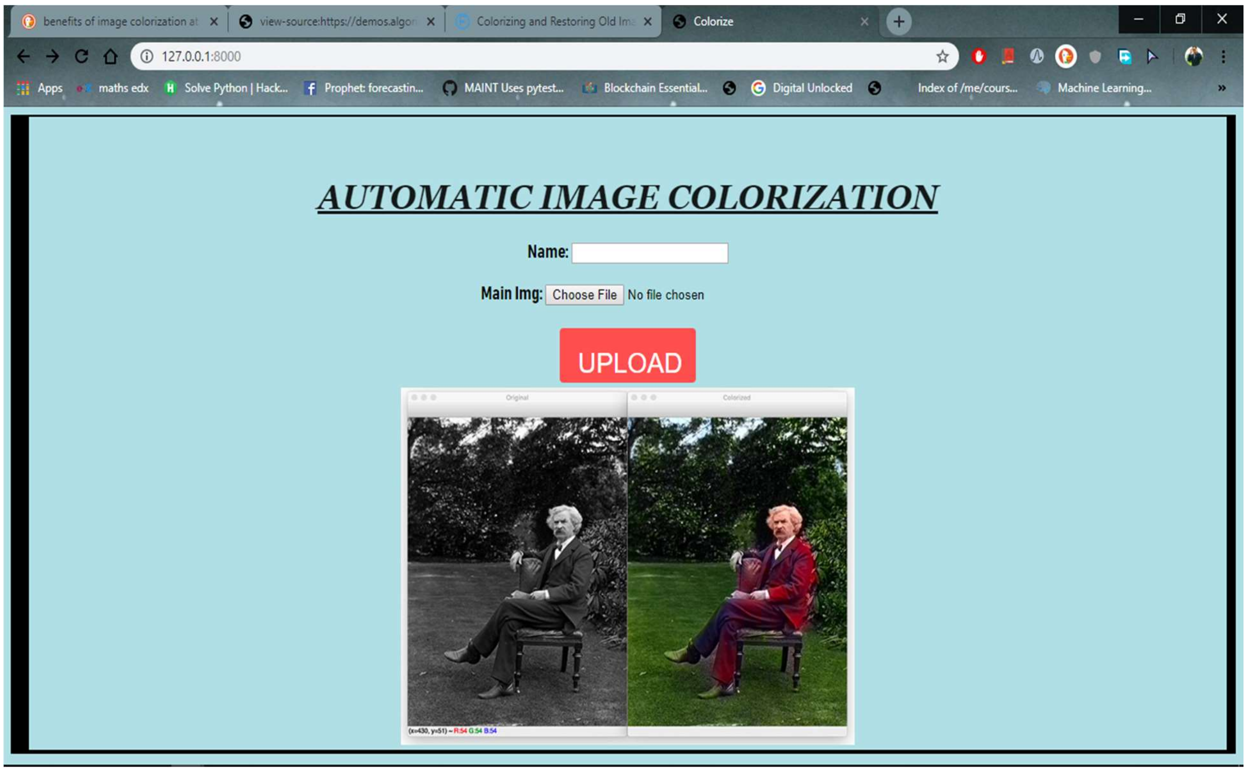
Module:

Name– User can enter the name for the new colorize image of his/her own choice.

Choose Image:-User can choose any grayscale image from the directory for the colorization process.

Upload: - While pressing the upload button the colorization process will starts and output will be shown.

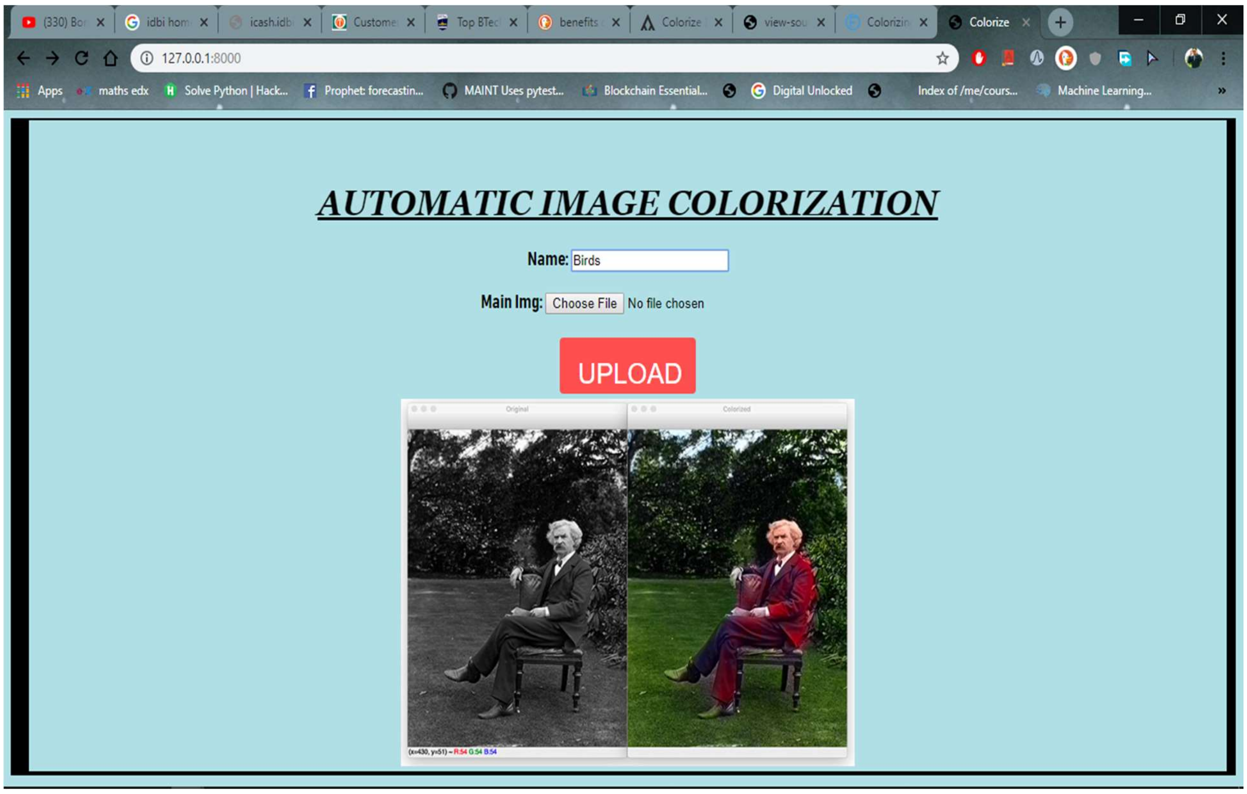
8.1 Interface of Our Project



Name:

Main Img: No file chosen

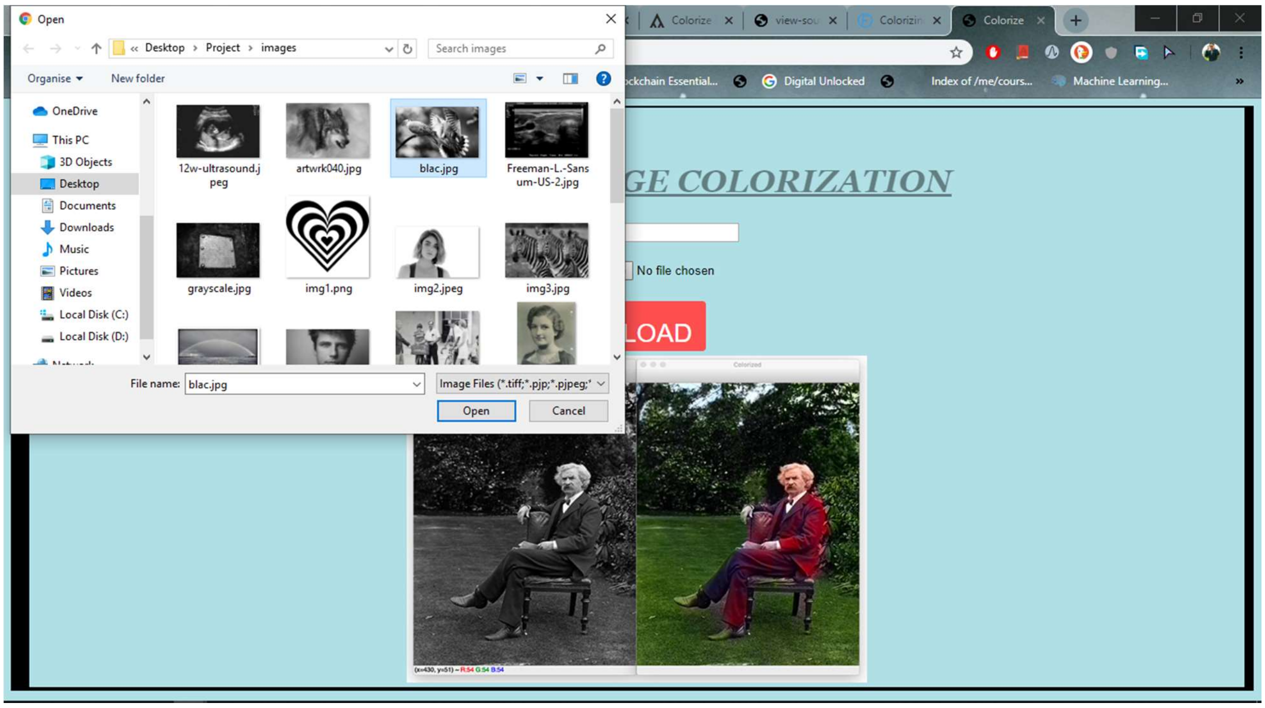
Fig 8.1 Home Page



Name: Birds

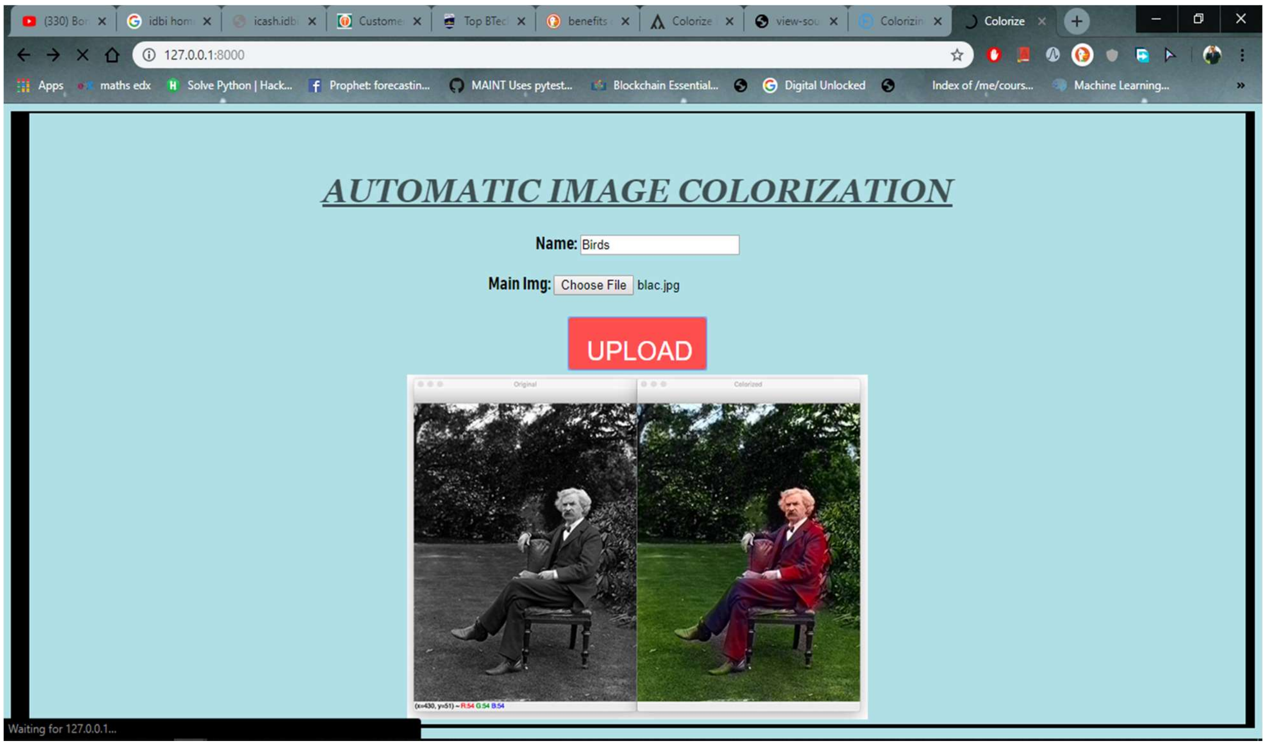
Main Img: No file chosen

Fig 8.2 Add Own Name for New colorize image (Example 1)



File name: blac.jpg

Fig. 8.3 Selection of Image (Example 1)



AUTOMATIC IMAGE COLORIZATION

Name: Birds

Main Img: blac.jpg

Fig. 8.4 Uploading (Example 1)

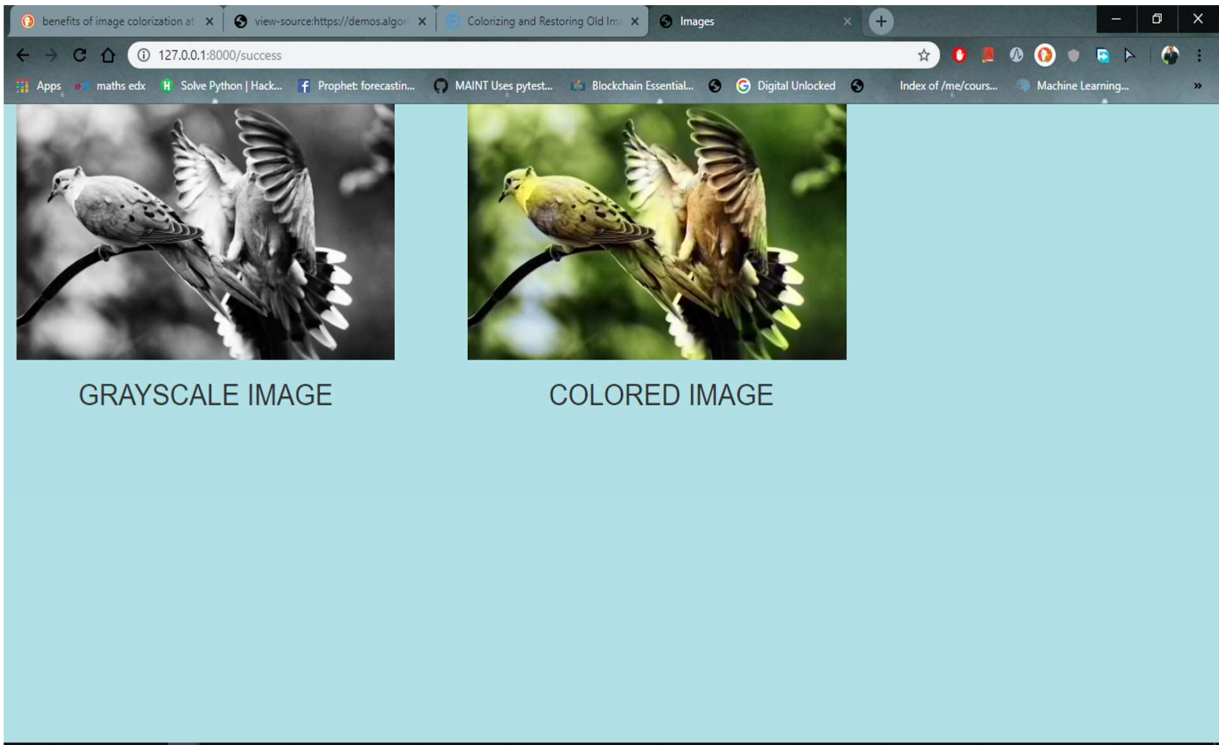


Fig. 8.5 Final output (Colorize Image) (Example 1)

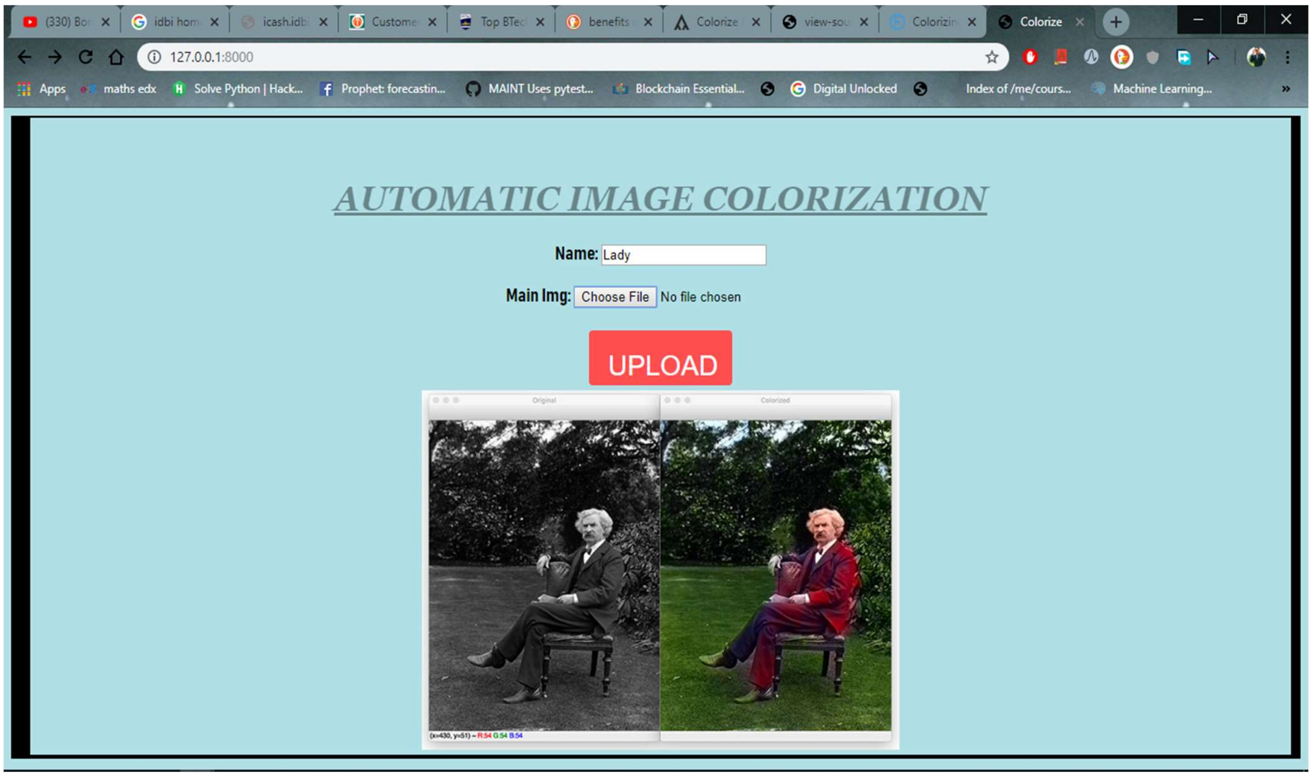

127.0.0.1:8000
AUTOMATIC IMAGE COLORIZATION
Name: Lady
Main Img: Choose File No file chosen
UPLOAD
Original
Colorized

Fig. 8.6 Add new name of the output image (Example 2)

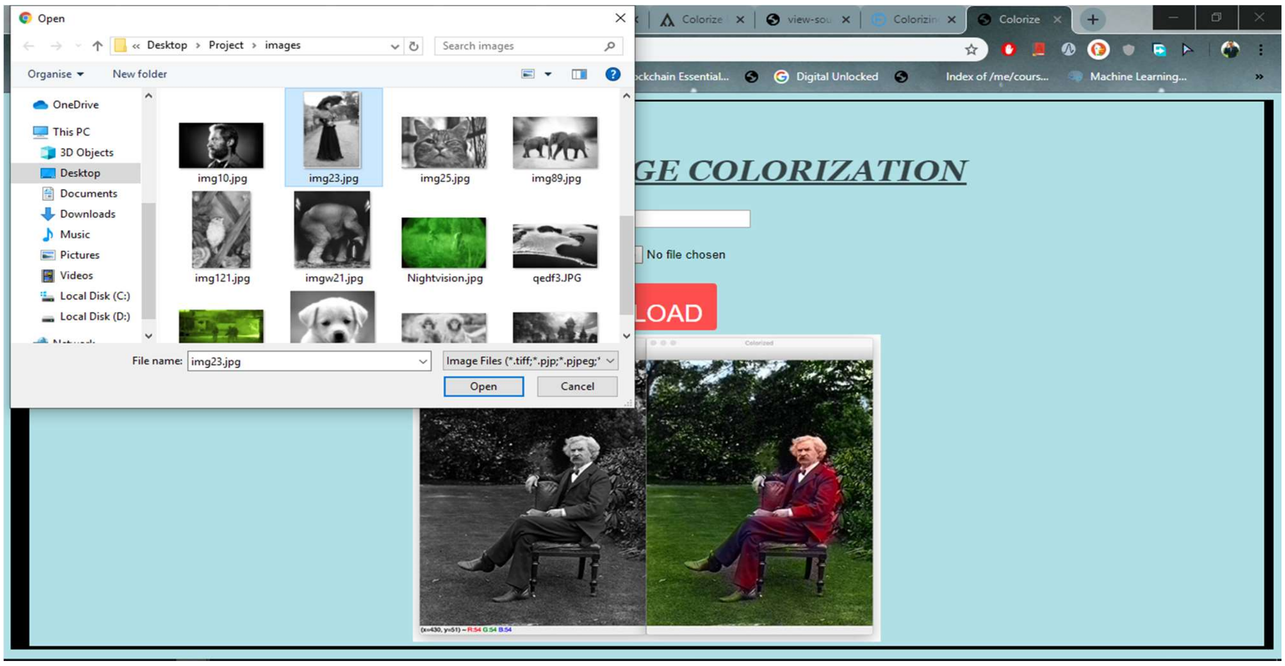


Fig. 8.7 Select Image for the Colorization (Example 2)

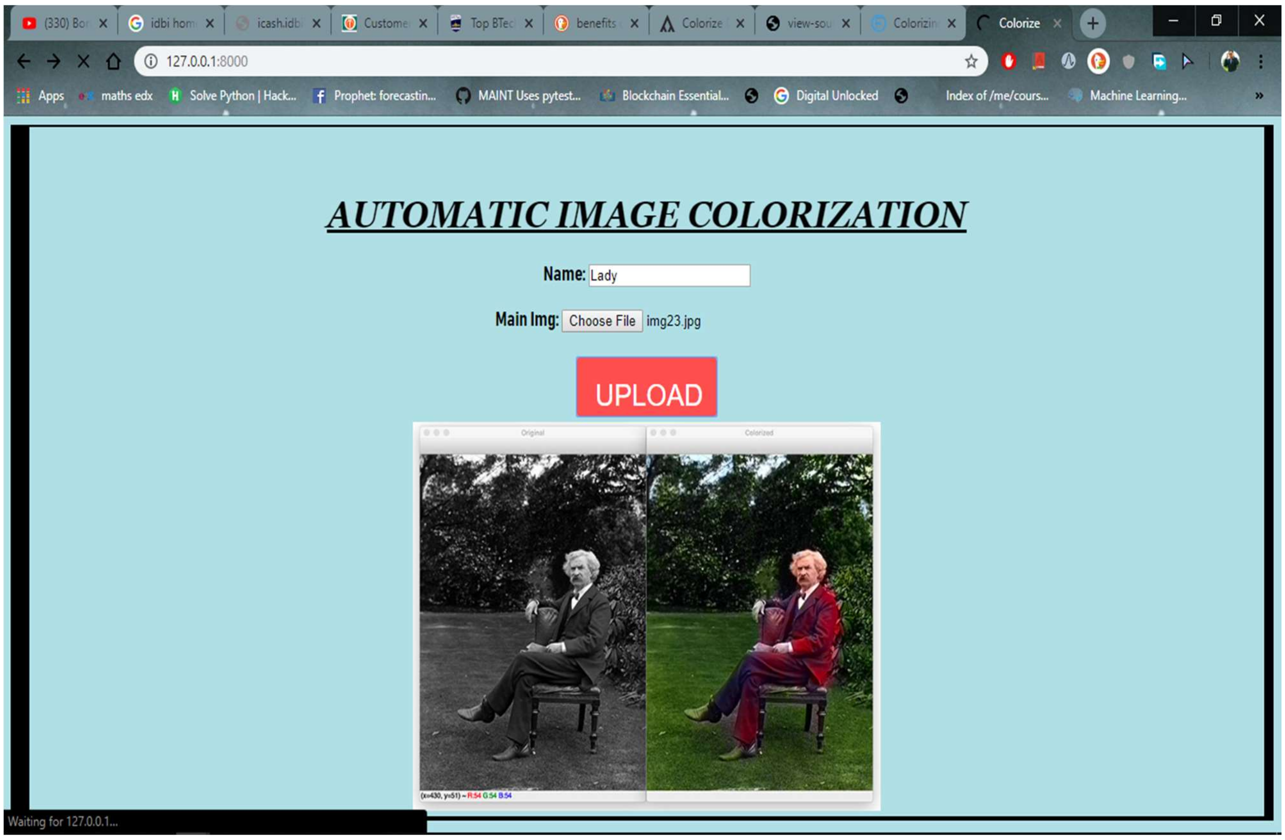


Fig 8.8 Uploading the image (Example 2)

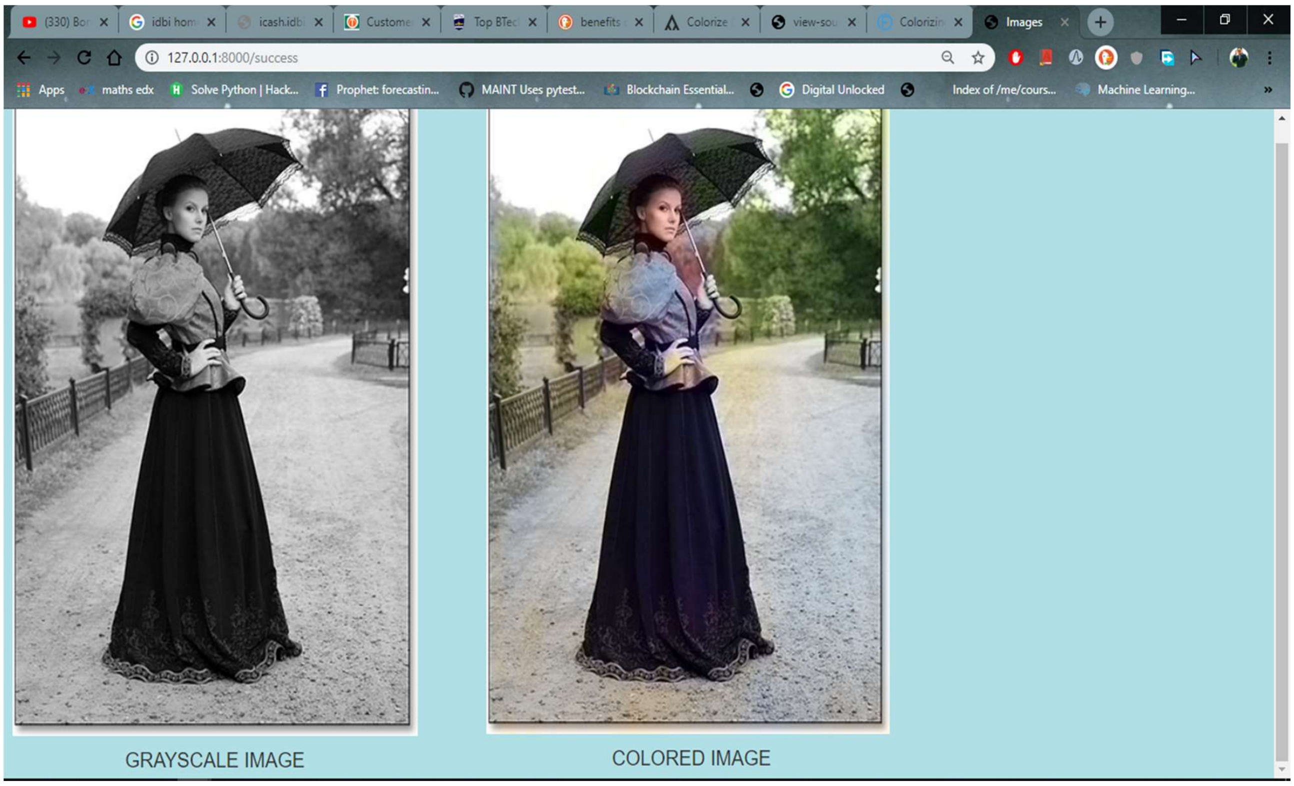


Fig. 8.9 Final Output (Example 2)

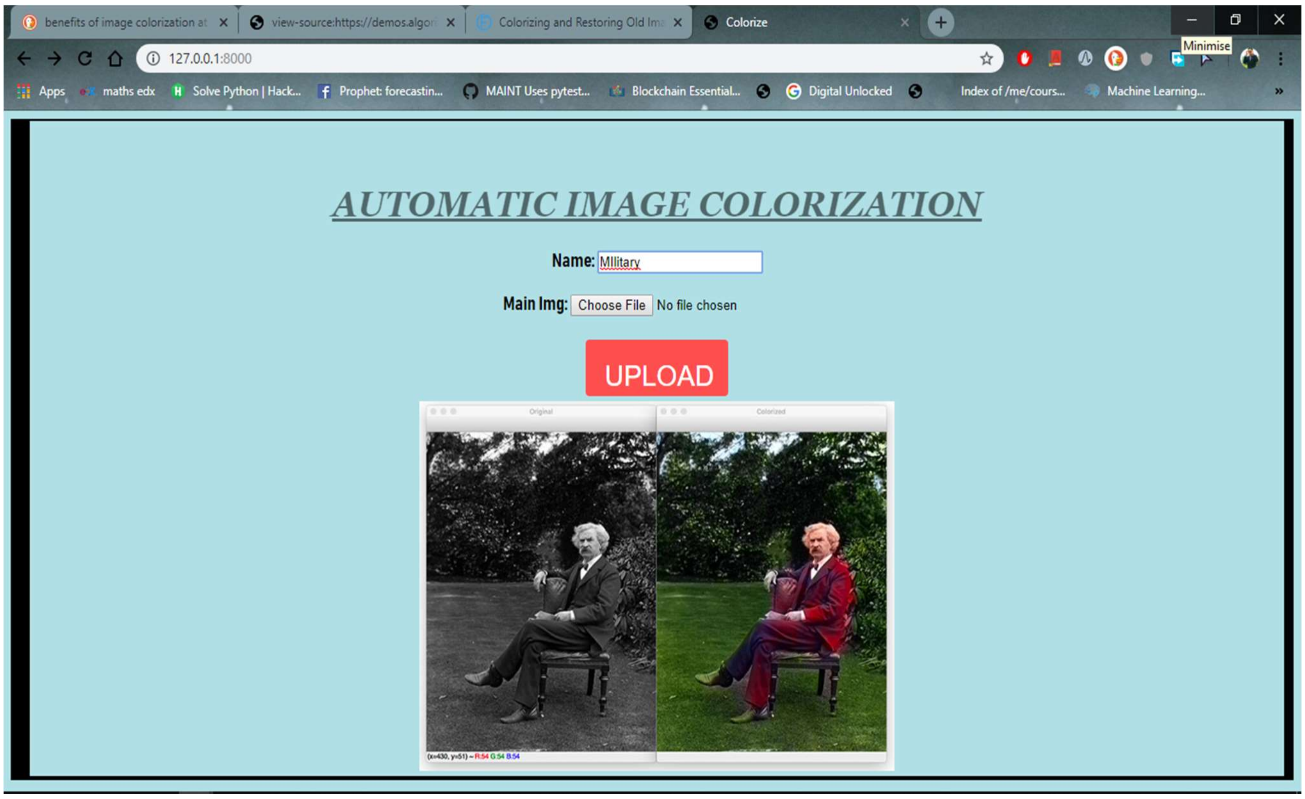

127.0.0.1:8000
AUTOMATIC IMAGE COLORIZATION
Name: Military
Main Img: Choose File No file chosen
UPLOAD

Fig 8.10 Add new name for output image (Example 3)

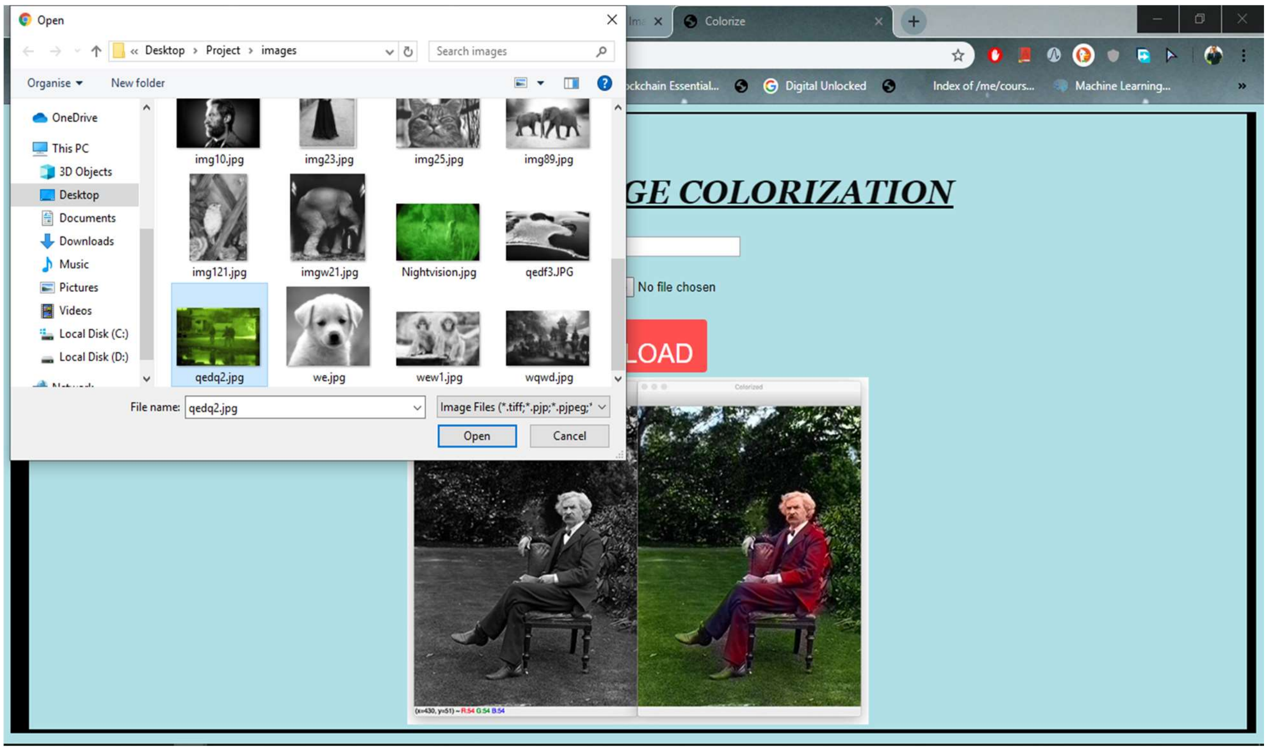


Fig. 8.11 Select the image (Example 3)

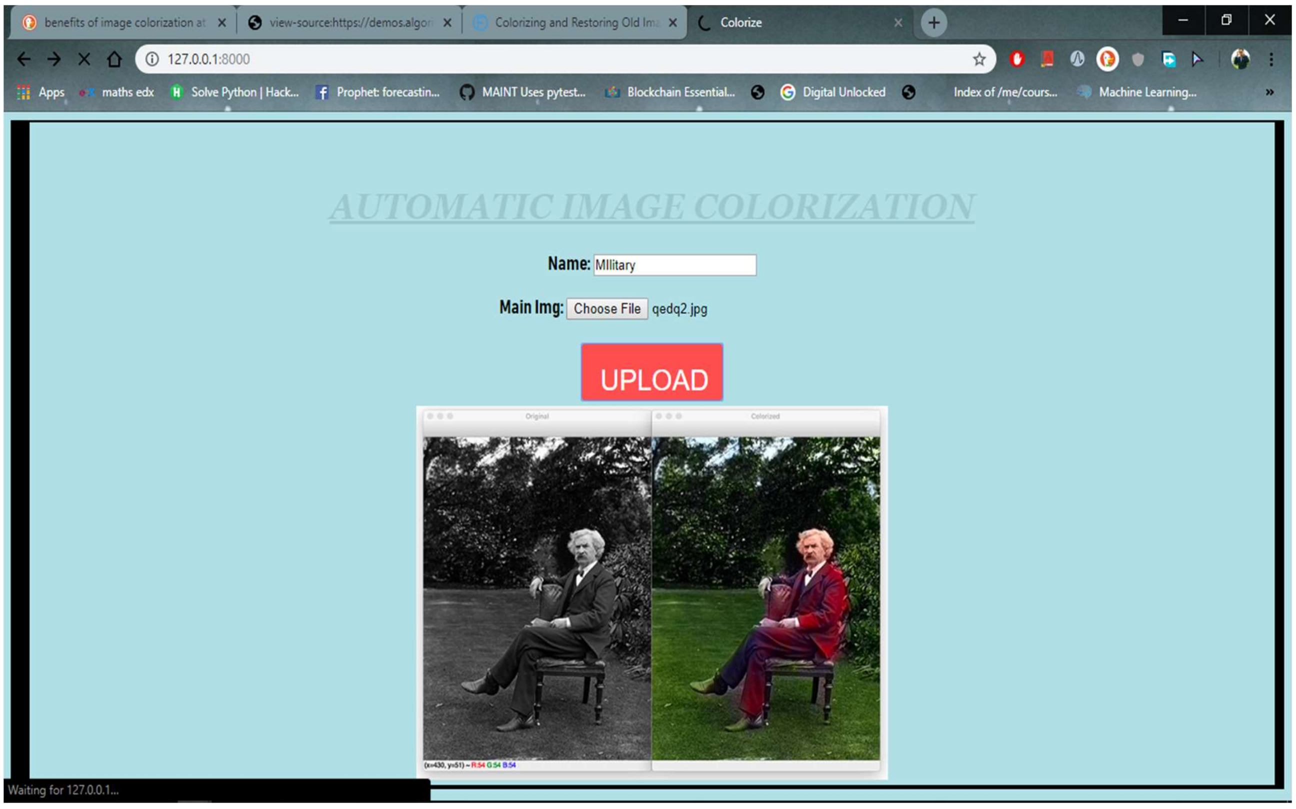


Fig. 8.12 Uploading Image (Example 3)

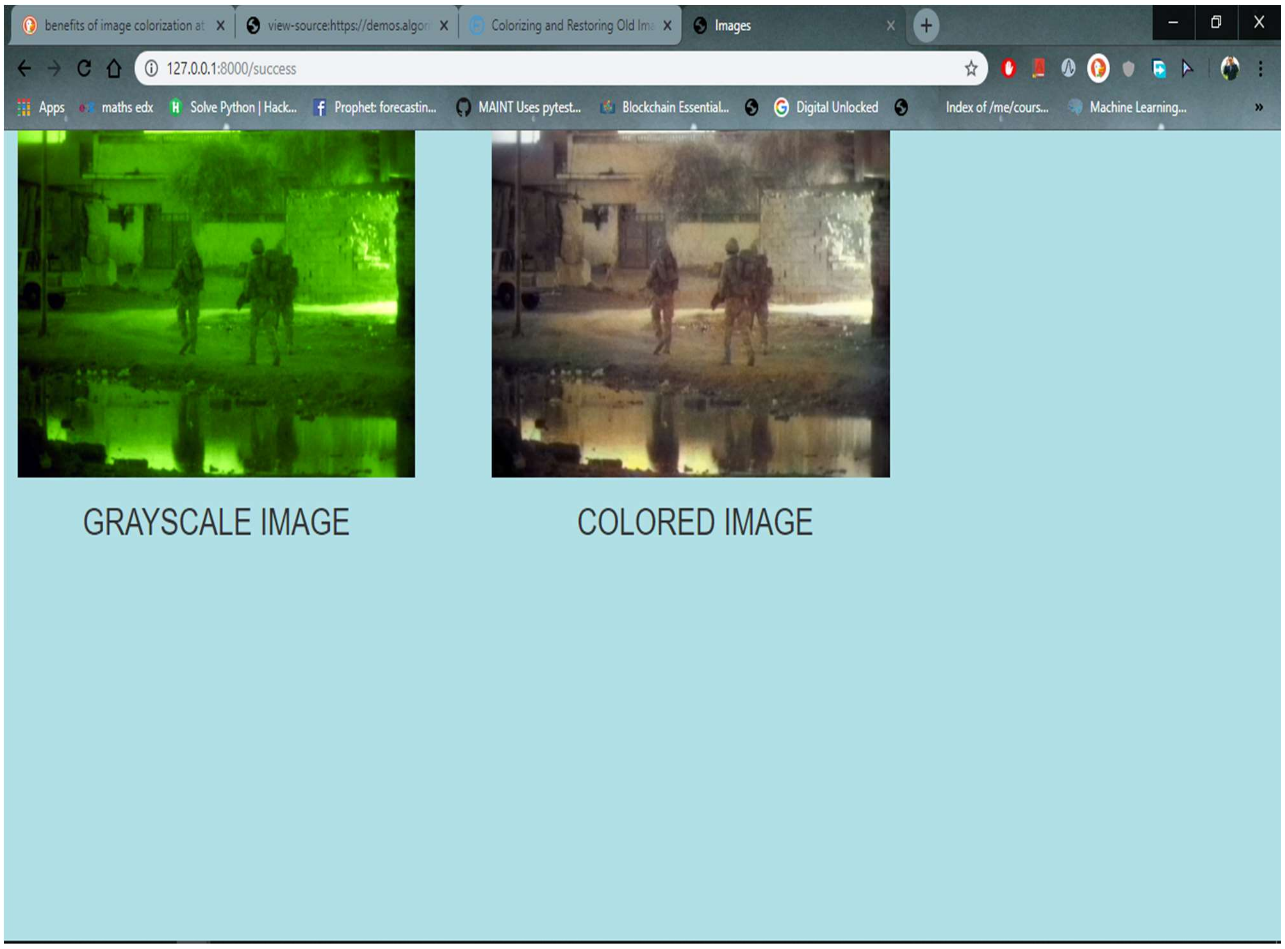


Fig 8.13 Output Image (Example 3)

CHAPTER-9
METHODOLOGY

Image colorization is the process of taking an **input grayscale (black and white) image** and then producing an **output colorized image** that represents the semantic colors and tones of the input (for example, an ocean on a clear sunny day must be plausibly "blue" — it can't be colored "hot pink" by the model)

Previous methods for image colorization either:

1. Relied on significant human interaction and annotation
2. Produced desaturated colorization

The novel approach we used relies on deep learning. We will utilize a Convolutional Neural Network capable of colorizing black and white images with results that can even fool humans.

9.1 Project Module:

- The phase one phase two and the phase three is done by Abinash Sinha.
- The two and third phase is done by Vikram Singh Rathore.
- The last phase is done by Aman Kumar Singh

9.1.1 Phase-I: Preprocessing

To train the network we started with the ImageNet dataset and converted all images from the RGB color space to the **Lab color space.**

Similar to the RGB color space, the Lab color space has *three channels*. But *unlike* the RGB color space, Lab encodes color information differently:

- The ***L* channel** encodes lightness intensity only
- The ***a* channel** encodes green-red.
- And the ***b* channel** encodes blue-yellow

9.1.2 Phase-II: Neural Network

We train a CNN to map from a grayscale input to a distribution over quantized color value outputs using the architecture shown in figure 91.2.1. In the following, we focus on the design of the objective function, and our technique for inferring point estimates of color from the predicted color distribution.

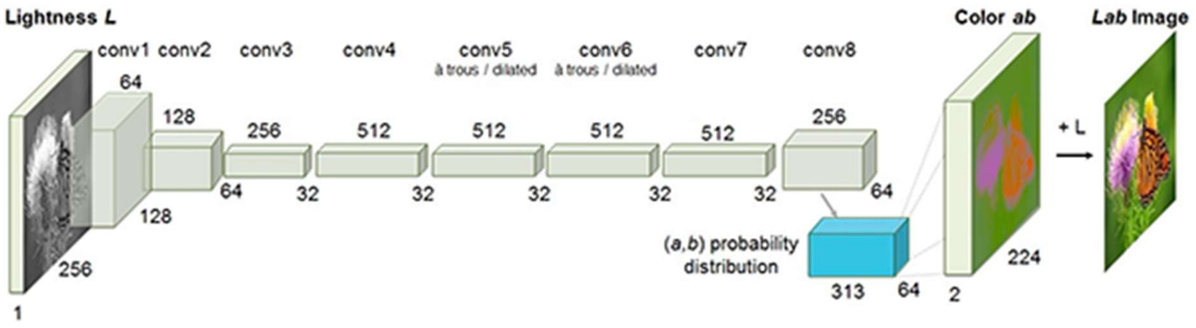
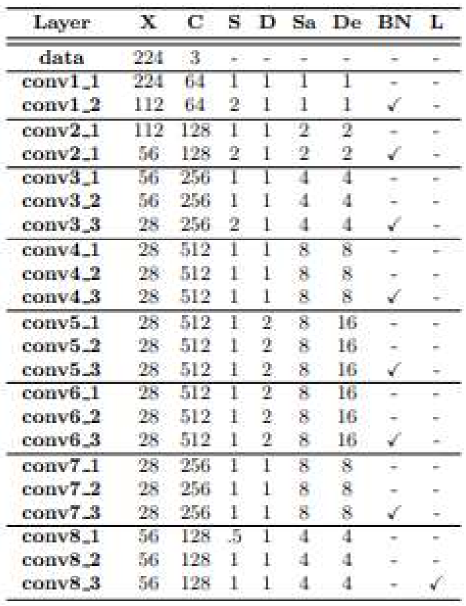


Fig 9.1 Convolutional Neural Network

Architecture of our neural network is given in figure 9.1.2.2. Our network architecture. X spatial resolution of output, C number of channels of output; S computation stride, values greater than 1 indicate downsampling following convolution, values less than 1 indicate upsampling preceding convolution; D kernel dilation; Sa accumulated stride across all preceding layers (product over all strides in previous layers); De effective dilation of the layer with respect to the input (layer dilation times accumulated stride); BN whether BatchNorm layer was used after layer; L whether a 1x1 conv and cross-entropy loss layer was imposed



Layer	X	C	S	D	Sa	De	BN	L
data	224	3	-	-	-	-	-	-
conv1_1	224	64	1	1	1	1	-	-
conv1_2	112	64	2	1	1	1	✓	-
conv2_1	112	128	1	1	2	2	-	-
conv2_1	56	128	2	1	2	2	✓	-
conv3_1	56	256	1	1	4	4	-	-
conv3_2	56	256	1	1	4	4	-	-
conv3_3	28	256	2	1	4	4	✓	-
conv4_1	28	512	1	1	8	8	-	-
conv4_2	28	512	1	1	8	8	-	-
conv4_3	28	512	1	1	8	8	✓	-
conv5_1	28	512	1	2	8	16	-	-
conv5_2	28	512	1	2	8	16	-	-
conv5_3	28	512	1	2	8	16	✓	-
conv6_1	28	512	1	2	8	16	-	-
conv6_2	28	512	1	2	8	16	-	-
conv6_3	28	512	1	2	8	16	✓	-
conv7_1	28	256	1	1	8	8	-	-
conv7_2	28	256	1	1	8	8	-	-
conv7_3	28	256	1	1	8	8	✓	-
conv8_1	56	128	.5	1	4	4	-	-
conv8_2	56	128	1	1	4	4	-	-
conv8_3	56	128	1	1	4	4	-	✓

Fig 9.2 Network Layers

Note that Image Colorization has several advantages in this setting: the test images are from the SUN dataset, which we did not train on and the 23 images were hand-selected from 1344 by the authors, and is not necessarily representative of algorithm performance. We were unable to obtain the 1344 test set results through correspondence with the authors. Additionally, we compare the methods on several important dimensions in algorithm pipeline, learning, dataset, and run-time. Our method is faster, straightforward to train and understand, has fewer hand-tuned parameters and components, and has been demonstrated on a broader and more diverse set of test images than Image Colorization.

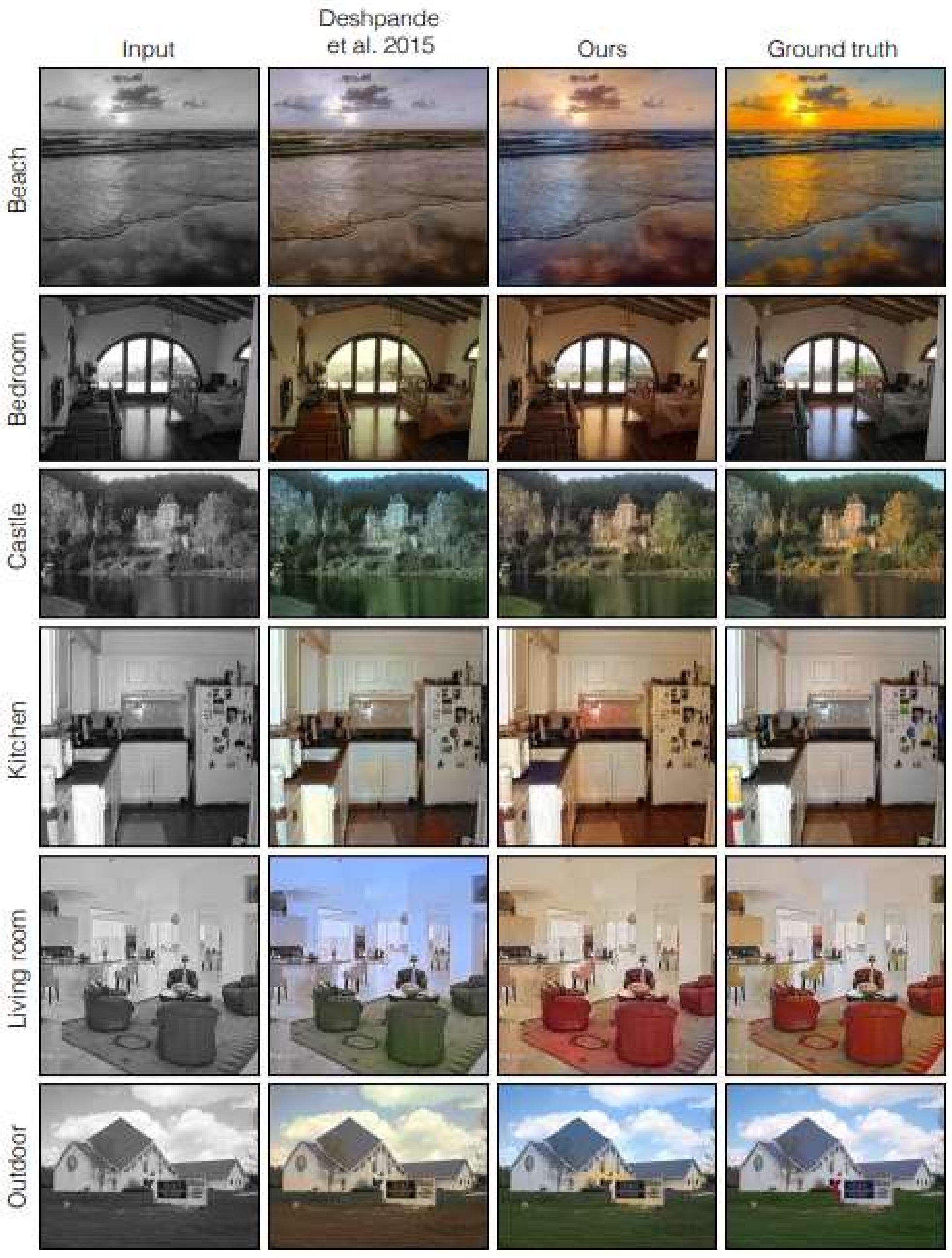


Fig 9.3 Test Images

9.1.1.3 Phase III: Web App

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django was initially developed between 2003 and 2005 by a web team who were responsible for creating and maintaining newspaper websites. After creating a number of sites, the team began to factor out and reuse lots of common code and design patterns. This common code evolved into a generic web development framework, which was open-sourced as the "Django" project in July 2005.

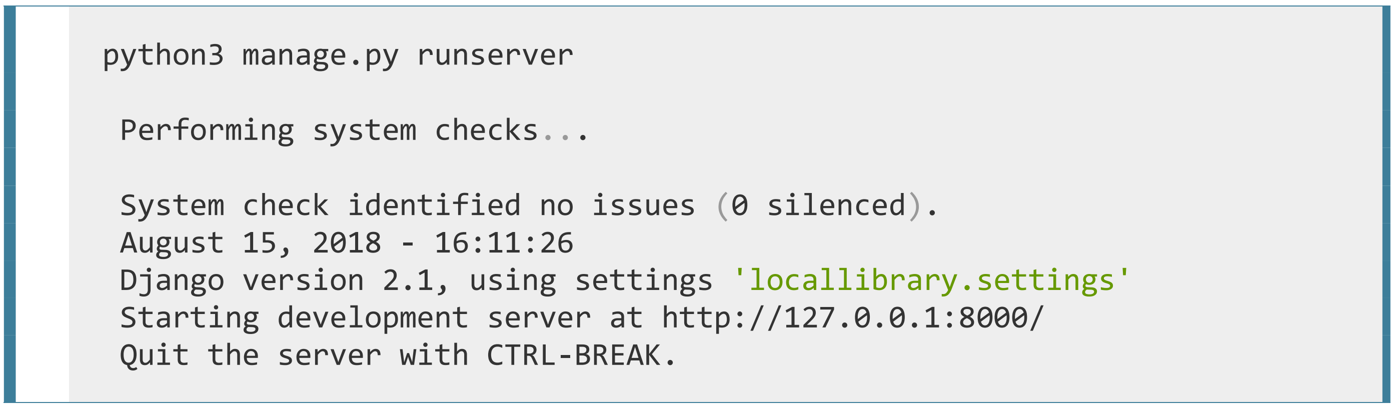
Our Web App the following directory.

```
├───colorize
│   └───__pycache__
├───imagecolorize
│   ├───migrations
│   │   └───__pycache__
│   ├───model
│   ├───static
│   ├───templates
│   └───__pycache__
└───media
    └───images
```

It has the following files:

- **_init__.py** is an empty file that instructs Python to treat this directory as a Python package.
- **settings.py** contains all the website settings. This is where we register any applications we create, the location of our static files, database configuration details, etc.
- **urls.py** defines the site url-to-view mappings. While this could contain *all* the url mapping code, it is more common to delegate some of the mapping to particular applications, as you'll see later.
- **wsgi.py** is used to help your Django application communicate with the web server. You can treat this as boilerplate.

The **manage.py** script is used to create applications, work with databases, and start the development web server. During development you can test the website by first serving it using the *development web server*, and then viewing it on your local web browser. Run the *development web server* by calling the runserver command (in the same directory as **manage.py**)



```
python3 manage.py runserver

 Performing system checks...

 System check identified no issues (0 silenced).
 August 15, 2018 - 16:11:26
 Django version 2.1, using settings 'locallibrary.settings'
 Starting development server at http://127.0.0.1:8000/
 Quit the server with CTRL-BREAK.
```

Fig 9.4 Django Run Server

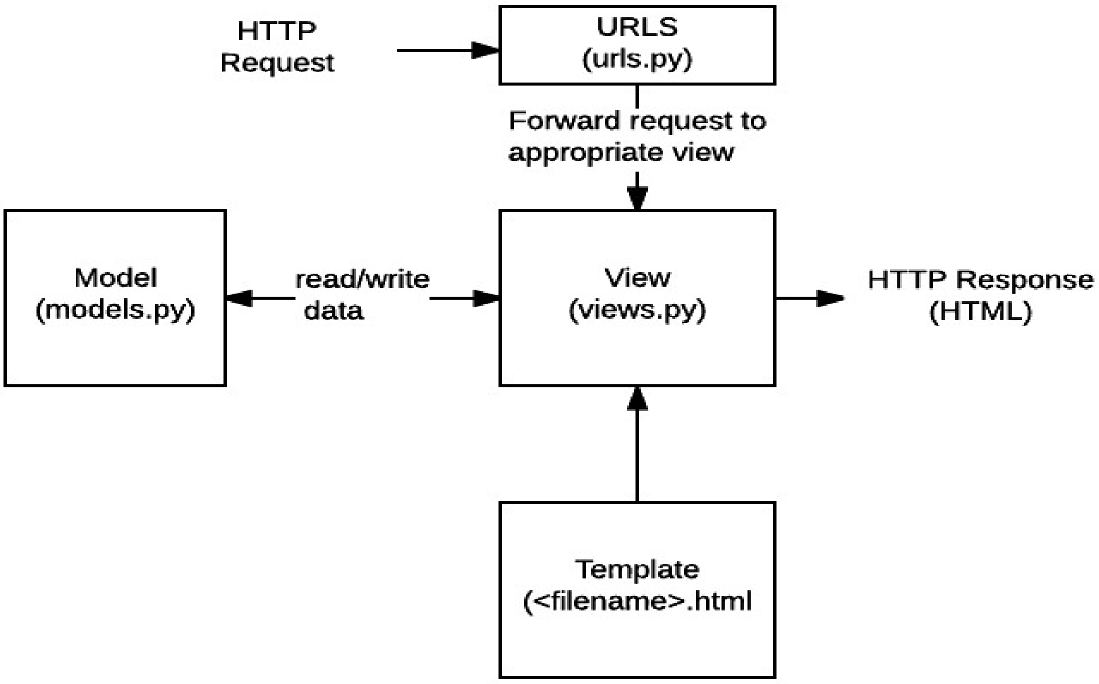


Fig 9.5 Server Architecture

REFERENCE

YouTube:-

Corey Schafer

https://www.youtube.com/watch?v=UmljXZIypDc

Website:-

https://www.w3schools.com/css/default.asp

https://www.w3schools.com/bootstrap/default.asp

https://www.djangoproject.com

CONCLUSION

Overall, we gained a newfound appreciation for the challenge of producing realistic colorizations and enjoyed experimenting with the approaches of previous papers.

The Lab color space seems like the best way to work with grayscale and color images.

The series of convolutional neural networks is a good simple model to colorize photos, and adding a parallel classifier to learn more about these grayscale photos to help in the colorization process makes intuitive sense.

Still, we find that we need to iterate further on these approaches to realize better colorings than the desaturated nature of our test results.