Landmark Recognition

Bachelor of Technology

in

Computer Science & Engineering

Submitted by

Aditya Navin Nair
Ashwin MS

Federal Institute of Science And Technology (FISAT)®
Angamaly, Ernakulam

Affiliated to

APJ Abdul Kalam Technological University
CET Campus, Thiruvananthapuram

FEDERAL INSTITUTE OF SCIENCE AND TECHNOLOGY
(FISAT)
Mookkannoor (P.O), Angamaly-683577

CERTIFICATE

This is to certify that the report entitled “Landmark Recognition” is a bonafide record of the mini project submitted by Ashwin M S(FIT19CS032), in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology (B. Tech) in Computer Science & Engineering during the academic year 2021-22.

		Dr. Jyothish K John
Staff in Charge	Project Guide	Head of the Department

Abstract

This project aims to create an application that recognizes various landmarks as well as its architecture from an image by implementing machine learning. This will allow users to easily recognize tourist attractions and attain information about these places. The challenge faced in this project is mainly the similarities between many of the structures in their architecture. Also we'll able to identify the landmark with a snippet of it's image. In the initial stage, we intend to classify landmarks with not many similarities and later on make classes that maybe similar in many ways.

Contribution by Author

Training the model required acquiring a good dataset. Most datasets were not specific to the needs of the project. A dataset was created using images scraped from across the internet. The main contribution was towards fine tuning the AlexNet layers. The future of the project is in creating a CNN completely catered to this dataset.

Acknowledgment

In the accomplishment of completion of my project on Landmark Recognition. I would like to convey my special gratitude to Mr. / Mrs. Pankaj Kumar G of Computer Science and Engineering Department for his kind support and guidance throughout the course of this project.

I also express my sincere gratitude to the project coordinators for helping us throughout the course of this project, maintaining momentum and keeping track of all stages of the project.

My sincere gratitude goes to Dr. Jyothish K John, HOD of Computer and Engineering Department for providing us the environment to complete the project in its stipulated time.Your valuable guidance and suggestions helped me in various phases of the completion of this project. I will always be thankful to you in this regard.

Finally, I thank the university for including this project as a part of the curriculum, thereby giving me a real world project experience.

I am ensuring that this project was finished by me and not copied.

Aditya Navin Nair

Signature.

Contents

List of Figures

Chapter 1

Introduction

1. (a) **Overview**

The development in electronic device like cameras, mobile phones and tablets with inbuilt camera as well their effective cost has made it to everyone hand. It is easier to take photos in the digital format. Most of the photos taken will travelling to tourist place are posted on social networking platforms on daily base. It has resulted in huge amount photos to available online. The touristic landmarks are easily recognizable of a well-known sites and architecture as in Fig.1.There are more than 1000 tourist places around the globe which have many architecture and monuments.

Figure 1.1: Some monuments

Manually identifying the image in large scale is Time consuming and not traceable, therefore automatic content based solution is required. The main challenge is there no precise definition of what is and what not a landmark. With many Landmarks it is hard for single person to keep track of their detail all the time. With the vast amount of images over the internet which is easily be accessed.

1.2 Problem Statement

Often pictures of landmarks are seen popping up in websites and other sources without description or with unclear descriptions. This problem is mainly faced by students when they refer to the internet for projects and other study purposes. Another demographic which find this issue, are tourists. Often it can be confusing to find information about the places they visit, especially when they can't identify what or which monument they are visiting. Another problem is to recognize what type of architecture style a landmark follows. E.g.: Mughal architecture. This problem has not been tackled by many machine learning models and will be useful for architecture enthusiasts to learn about different buildings.

1.3 Objective

Landmark and architecture Recognition is mainly used for architecture enthusiasts and school children to help them identify what type of architecture was used as well as to help identify which landmark it is.

Chapter 2

Design

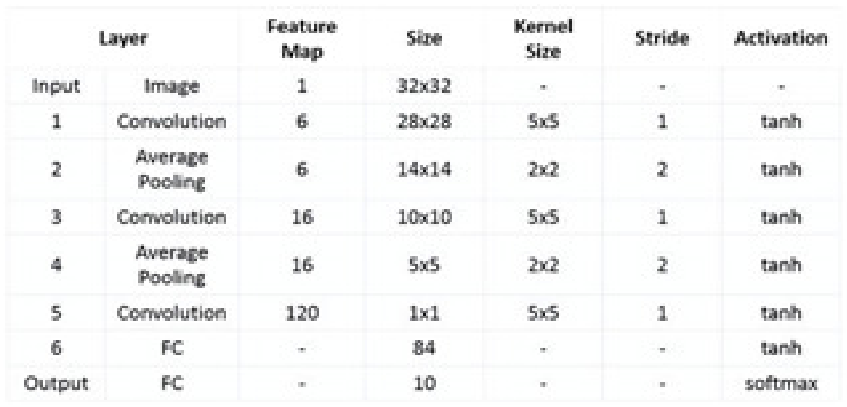
1. (a) **Implementation**

Models

2.1. LeNet

Structure of the LeNet network

LeNet5 is a small network, it contains the basic modules of deep learning: convolutional layer, pooling layer, and full link layer. It is the basis of other deep learning models. Here we analyze LeNet5 in depth. At the same time, through example analysis, deepen the understanding of the convolutional layer and pooling layer.



Layer		Feature Map	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x28	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	FC	-	84	-	-	tanh
Output	FC	-	10	-	-	softmax

Figure 2.1: Architecture of LeNet

LeNet-5 Total seven layer , does not comprise an input, each containing a trainable parameters; each layer has a plurality of the Map the Feature , a characteristic of each of the input FeatureMap extracted by means of a convolution filter, and then each FeatureMap There are multiple neurons.

INPUT LAYER

The first is the data INPUT layer. The size of the input image is uniformly normalized to 32 * 32.

Note: This layer does not count as the network structure of LeNet-5. Traditionally, the input layer is not considered as one of the network hierarchy.

C1 layer-convolutional layer:

1. **Input picture**: 32 * 32
2. **Convolution kernel size**: 5 * 5
3. **Convolution kernel types**: 6
4. **Output featuremap size**: 28 * 28 (32-5 + 1) = 28
5. **Number of neurons**: 28 *28* 6
6. **Trainable parameters**: (5 *5 + 1)* 6 (5 * 5 = 25 unit parameters and one bias parameter per filter, a total of 6 filters)
7. **Number of connections**: (5 *5 + 1)* 6 *28* 28 = 122304

S2 layer-pooling layer (downsampling layer):

1. **Input**: 28 * 28
2. **Sampling area**: 2 * 2
3. **Sampling method**: 4 inputs are added, multiplied by a trainable parameter, plus a trainable offset. Results via sigmoid
4. **Sampling type**: 6

5. **Output featureMap size**: 14 * 14 (28/2)

6. **Number of neurons**: 14 *14* 6

7. **Trainable parameters**: 2 * 6 (the weight of the sum + the offset)

8. **Number of connections**: (2 *2 + 1)* 6 *14* 14

9. The size of each feature map in S2 is 1/4 of the size of the feature map in C1.

C3 layer-convolutional layer:

1. **Input**: all 6 or several feature map combinations in S2

2. **Convolution kernel size**: 5 * 5

3. **Convolution kernel type**: 16

4. **Output featureMap size**: 10 * 10 (14-5 + 1) = 10

5. Each feature map in C3 is connected to all 6 or several feature maps in S2, indicating that the feature map of this layer is a different combination of the feature maps extracted from the previous layer.

6. One way is that the first 6 feature maps of C3 take 3 adjacent feature map subsets in S2 as input. The next 6 feature maps take 4 subsets of neighboring feature maps in S2 as input. The next three take the non-adjacent 4 feature map subsets as input. The last one takes all the feature maps in S2 as input.

7. **The trainable parameters are**: 6 *(3* 5 *5 + 1) + 6* (4 *5* 5 + 1) + 3 *(4* 5 *5 + 1)* + *1* (6 *5* 5 +1) = 1516

8. **Number of connections**: 10 *10* 1516 = 151600

S4 layer-pooling layer (down sampling layer):

1. **Input**: 10 * 10
2. **Sampling area**: 2 * 2
3. **Sampling method**: 4 inputs are added, multiplied by a trainable parameter, plus a trainable offset. Results via sigmoid
4. **Sampling type**: 16
5. **Output featureMap size**: 5 * 5 (10/2)
6. **Number of neurons**: 5 *5* 16 = 400
7. **Trainable parameters**: 2 * 16 = 32 (the weight of the sum + the offset)
8. **Number of connections**: 16 *(2* 2 + 1) *5* 5 = 2000
9. The size of each feature map in S4 is 1/4 of the size of the feature map in C3

C5 layer-convolution layer :

1. **Input**: All 16 unit feature maps of the S4 layer (all connected to s4)
2. **Convolution kernel size**: 5 * 5
3. **Convolution kernel type**: 120
4. **Output featureMap size**: 1 * 1 (5-5 + 1)
5. **Trainable parameters / connection**: 120 *(16* 5 * 5 + 1) = 48120

F6 layer-fully connected layer :

1. **Input**: c5 120-dimensional vector
2. **Calculation method**: calculate the dot product between the input vector and the weight vector, plus an offset, and the result is output through the sigmoid function.
3. **Trainable parameters**: 84 * (120 + 1) = 10164

2.2 AlexNet

AlexNet is an incredibly powerful model capable of achieving high accuracies on very challenging datasets. However, removing any of the convolutional layers will drastically degrade AlexNet's performance. AlexNet is a leading architecture for any object-detection task and may have huge applications in the computer vision sector of artificial intelligence problems. In the future, AlexNet may be adopted more than CNNs for image tasks.

As a milestone in making deep learning more widely-applicable, AlexNet can also be credited with bringing deep learning to adjacent fields such as natural language processing and medical image analysis.

AlexNet Architecture

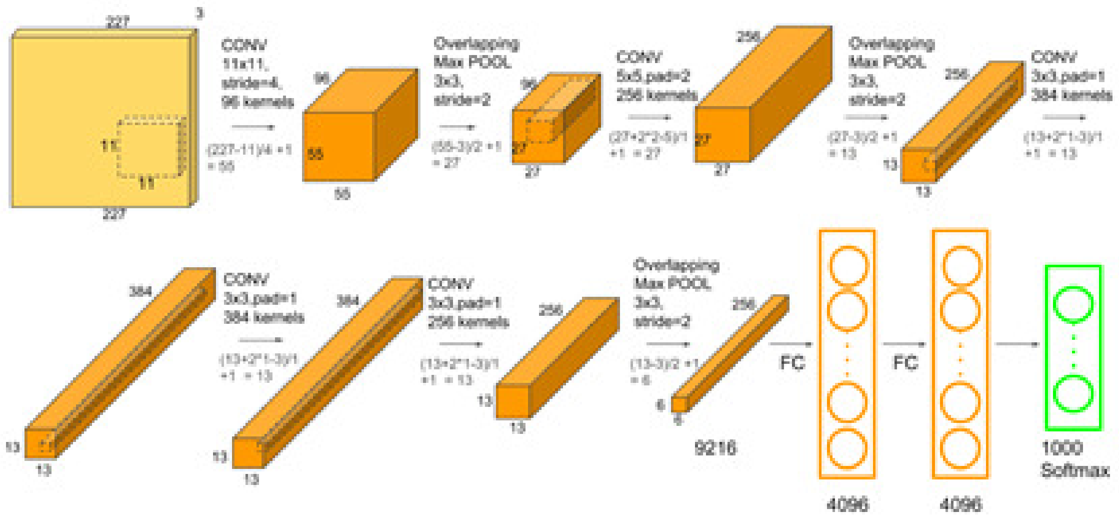


Figure 2.2: Architecture of AlexNet

AlexNet was the first convolutional network which used GPU to boost performance.
1. AlexNet architecture consists of 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer.
2. Each convolutional layer consists of convolutional filters and a nonlinear activation function ReLU.
3. The pooling layers are used to perform max pooling.
4. Input size is fixed due to the presence of fully connected layers.
5. The input size is mentioned at most of the places as 224x224x3 but due to some padding which happens it works out to be 227x227x3
6. AlexNet overall has 60 million parameters.

2.2 Versions proposed

Version 0:
Architecture – LeNet is used as an initial stage of landmark recognition. This version is expected to have a lesser accuracy as compared to other CNNs.
Libraries used - tensorflow, NumPy, Sklearn, Matplotlib

Version 1:
Architecture – Alexnet, a convolutional neural network is used where an input image example is given to the model in the form of tensor and used to identify the landmark and its architecture
Libraries used - tensorflow, NumPy, Sklearn, Matplotlib

Version 2(Not Completed):
Architecture – A CNN neural network which can classify the image with a better accuracy that version 1
Libraries used - tensorflow, NumPy, Sklearn, Matplotlib.

Chapter 3

Dataset

3.1. Dataset Information

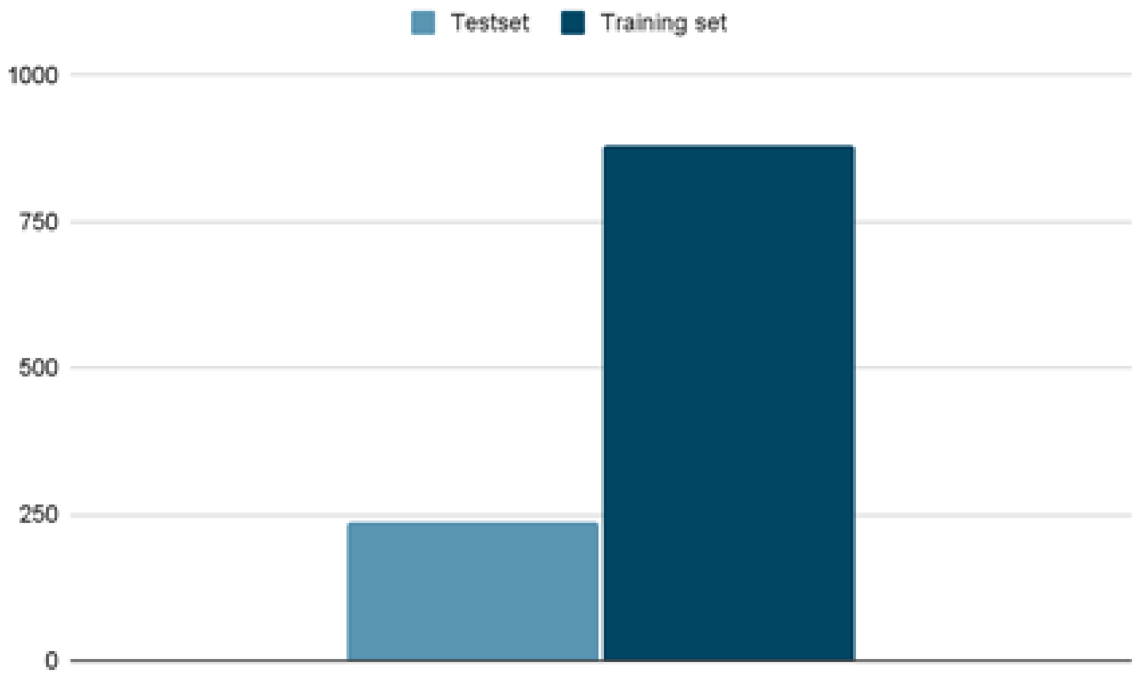


Figure 3.1: Dataset distribution graph

The dataset used comprises of images of 11 landmarks: Mysore Palace, Taj Mahal, The Golden Temple Amritsar, Red Fort, Agra Fort, Indian Parliament, Char Minar, India Gate, Rashtrapati Bhavan, Lotus Temple, Victoria Memorial

1. Mysore Palace 86

2. Taj Mahal 82

3. The Golden Temple Amritsar 84

4. Red Fort 84

5. Agra Fort 79
6. Indian Parliament 75
7. Char Minar 85
8. India Gate 86
9. Rashtrapati Bhavan 68
10. Lotus Temple 93
11. Victoria Memorial 76

3.2. Dataset Acquisition

Acquiring a dataset for training the model, one by one, is a difficult task. Hence the images required were downloaded as batches using the program mentioned in Chapter 4 of this document.

Chapter 4

Source Code

4.1 Training Model

```
import tensorflow as tf
from google.colab import drive
drive.mount('/content/drive')
cd drive/MyDrive
import os,cv2
import numpy as np
import matplotlib.pyplot as plt
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
from keras import backend as K
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Flatten
from keras.layers.convolutional import Convolution2D, MaxPooling2D, Conv2D
from tensorflow.keras.optimizers import SGD,RMSprop
PATH = os.getcwd()
print(PATH)
# Define data path
data_path = PATH + '/dataset-landmark'
data_path1 = PATH + '/dataset-landmark_test'
data_dir_list = os.listdir(data_path)
print(data_dir_list)
img_rows=128
img_cols=128
```

```
num_channel=1
num_epoch=20
dataimg_train=tf.keras.utils.image_dataset_from_directory(
    directory=data_path,
    labels='inferred',
    label_mode='categorical',
    class_names=None,
    color_mode='rgb',
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=1,
    validation_split=None,
    subset=None,
    interpolation='bilinear',
    follow_links=False,
    crop_to_aspect_ratio=False,
)
dataimg_valid=tf.keras.utils.image_dataset_from_directory(
    directory=data_path1,
    labels='inferred',
    label_mode='categorical',
    class_names=None,
    color_mode='rgb',
    batch_size=32,
    image_size=(256, 256),
    shuffle=True,
    seed=1,
    validation_split=None,
    subset=None,
    interpolation='bilinear',
   follow_links=False,
   crop_to_aspect_ratio=False,
)
#Test list test_list = [ ”/content/drive/MyDrive/Demo_Images_landmarks/LotusTemple.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/IndiaGate.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/IndianParliament.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/IndianParliament1.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/MysorePalace.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/MysorePalace1.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/RashtrapatiBhavan.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/RashtrapatiBhavan1.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/RedFort.jpeg”,
”/content/drive/MyDrive/Demo_Images_landmarks/VictoriaMemorial.jpeg” ]
#CNN Architecture
model = Sequential()
model.add(tf.keras.layers.Conv2D(6, kernel_size=(5, 5), activation=’relu’, in-
put_shape=(256, 256, 3)))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(tf.keras.layers.Conv2D(16, kernel_size=(5, 5), activation=’relu’))
model.add(tf.keras.layers.Conv2D(384, kernel_size=(3,3), strides= 1,
          padding= ’same’, activation= ’relu’,
          kernel_initializer= ’he_normal’))
model.add(tf.keras.layers.Conv2D(256, kernel_size=(3,3), strides= 1,
          padding= ’same’, activation= ’relu’,
          kernel_initializer= ’he_normal’))
model.add(tf.keras.layers.MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(120, activation=’relu’))
model.add(tf.keras.layers.Dense(11, activation=’softmax’))
model.compile(loss=tf.keras.metrics.categorical_crossentropy, optimizer=tf.keras.optimizers.Adam(), met-
rics=[’accuracy’])
model.summary()
model.fit(dataimg_train, epochs=15)
model.save(”savedweights_final”)
from keras.models import load_model
model =load_model(”savedweights_final”)
score = model.evaluate(dataimg_valid)
print(’Test Loss:’, score[0])
print(’Test accuracy:’, score[1])
from keras.models import load_model
from keras.preprocessing import image
import matplotlib.pyplot as plt
import numpy as np
import os
def load_image(img_path, show=False):
    img = image.load_img(img_path, target_size=(256, 256))
    img_tensor = image.img_to_array(img)                    # (height, width, channels)
   img_tensor = np.expand_dims(img_tensor, axis=0)       # (1, height, width, chan-
nels), add a dimension because the model expects this shape: (batch_size, height, width, chan-
nels)
    img_tensor /= 255.
 # imshow expects values in the range [0, 1]
    if show:
        plt.imshow(img_tensor[0])
        plt.axis(’off’)
        plt.show()
    return img_tensor
if __name__ == ”__main__”:
    # load model
    #model = load_model(”savedweights_final”)
```

```
# image path
img_path = test_list[0]
# load a single image
new_image = load_image(img_path)
# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

4.2 Image Scraper

4.2.1 main.py

```
# Import libraries
import os
import concurrent.futuresfrom GoogleImageScraper
import GoogleImageScraperfrom patch
import webdriver_executable
defworker_thread(search_key):
image_scraper = GoogleImageScraper(
webdriver_path, image_path, search_key, number_of_images, headless,
min_resolution, max_resolution)
image_urls = image_scraper.find_image_urls()
image_scraper.save_images(image_urls)
# Release resources
del image_scraper
if __name__ == "__main__": # Define file path
webdriver_path = os.path.normpath(os.path.join(os.getcwd(),'webdriver',webdriver_executable()))
image_path = os.path.normpath(os.path.join(os.getcwd(), 'photos'))
# Add new search key into array ["cat","t-shirt","apple","orange","pear","fish"]
search_keys = ["Indian Parliament Building"]
# Parameters
```

```
number_of_images = 500    # Desired number of images
headless = True    # True = No Chrome GUI
min_resolution = (0, 0) # Minimum desired image resolution
max_resolution = (9999, 9999) # Maximum desired image resolution
max_missed = 1000 # Max number of failed images before exit
number_of_workers = 1 # Number of ”workers” used
# Run each search_key in a separate thread
# Automatically waits for all threads to finish with concurrent.futures.ThreadPoolExecutor(max_workers=
number_of_workers) as executor:
executor.map(worker_thread, search_keys)
```

4.2.2 GoogleImageScraper.py

```
#import selenium driversfrom selenium
import webdriverfrom selenium.webdriver.chrome.options
import Options
#import helper librariesimport timeimport urllib.request
import os
import requests
import io from PIL
import Image
#custom patch libraries
import patch
class GoogleImageScraper():
def __init__(self, webdriver_path, image_path, search_key=”cat”, number_of_images=1,
headless=True, min_resolution=(0, 0), max_resolution=(1920, 1080), max_missed=10):
#check parameter types
image_path = os.path.join(image_path, search_key)
if (type(number_of_images)!=int):
print(”[Error] Number of images must be integer value.”)
return
if not os.path.exists(image_path):
```

```
print("[INFO] Image path not found. Creating a new folder.")
os.makedirs(image_path)
#check if chromedriver is updated
while(True):
try:
#try going to www.google.com
options = Options()
if(headless):
options.add_argument('–headless')
driver = webdriver.Chrome(webdriver_path, chrome_options=options)
driver.set_window_size(1400,1050)
driver.get("https://www.google.com")
break
except:
#patch chromedriver if not available or outdated
try:
driver
except NameError:
is_patched = patch.download_lastest_chromedriver()
else:
is_patched = patch.download_lastest_chromedriver(driver.capabilities['version'])
if (not is_patched):
exit("[ERR] Please update the chromedriver.exe in the webdriver folder ac-
cording to your chrome version:https://chromedriver.chromium.org/downloads")
self.driver = driver
self.search_key = search_key
self.number_of_images = number_of_images
self.webdriver_path = webdriver_path
self.image_path = image_path
self.url = "https://www.google.com/search?q=%s&source=lnms&tbm=isch&sa=X&ved=2ahUKEwie44_Anq
self.headless=headless
self.min_resolution = min_resolution
self.max_resolution = max_resolution
self.max_missed = max_missed
def find_image_urls(self): print(”[INFO] Gathering image links”)
image_urls=[]
count = 0
missed_count = 0
self.driver.get(self.url)
time.sleep(??)
indx = 1
while self.number_of_images > count:
try:
#find and click image
imgurl = self.driver.find_element_by_xpath(’//*[@id=”islrg”]/div[1]/div[%s]/a[1]/div[1]/img’%(str(indx)))
imgurl.click()
missed_count = 0
except Exception:
#print(”[-] Unable to click this photo.”)
missed_count = missed_count + 1
if(missed_count>self.max_missed):
print(”[INFO] Maximum missed photos reached, exiting...”)
break
try:
#select image from the popup
time.sleep(??)
class_names = [”n3VNCb”]
images = [self.driver.find_elements_by_class_name(class_name)
for class_name in class_names if
len(self.driver.find_elements_by_class_name(class_name)) != 0 ][0]
for image in images:
#only download images that starts with http
```

```
src_link = image.get_attribute(”src”)
if((”http” in src_link) and (not ”encrypted” in src_link)):
print(f”[INFO] {self.search_key} \t #{count} \t {src_link}”)
image_urls.append(src_link)
count +=1
break
except Exception:
print(”[INFO] Unable to get link”)
try:
#scroll page to load next image
if(count%3==0):
self.driver.execute_script(”window.scrollTo(0, ”+str(indx*60)+”);”)
element = self.driver.find_element_by_class_name(”mye4qd”)
element.click()
print(”[INFO] Loading next page”)
time.sleep(??)
except Exception:
time.sleep(??) indx += 1
self.driver.quit()
print(”[INFO] Google search ended”)
return image_urls
def save_images(self,image_urls):
#save images into file directory
print(”[INFO] Saving image, please wait...”)
for indx,image_url in enumerate(image_urls):
try:
print(”[INFO] Image url:%s”%(image_url))
search_string = ”.join(e for e in self.search_key if e.isalnum())
image = requests.get(image_url,timeout=5)
if image.status_code == 200:
with Image.open(io.BytesIO(image.content)) as image_from_web:
```

```
try:
filename = ”%s%s.%s”%(search_string,str(indx),image_from_web.format.lower())
image_path = os.path.join(self.image_path, filename)
print(f”[INFO] {self.search_key} \t {indx} \t Image saved at: {image_path}”)
image_from_web.save(image_path)
except OSError:
rgb_im = image_from_web.convert(’RGB’)
rgb_im.save(image_path)
image_resolution = image_from_web.size
if image_resolution != None:
if image_resolution[0]<self.min_resolution[0] or
image_resolution[1]<self.min_resolution[1] or
image_resolution[0]>self.max_resolution[0] or
image_resolution[1]>self.max_resolution[1]:
image_from_web.close()
#print(”GoogleImageScraper Notification: %s did not meet resolution require-
ments.”%(image_url))
os.remove(image_path)
image_from_web.close()
except Exception as e:
print(”[ERROR] Download failed: ”,e)
pass
print(”——————————————————————”)
print(”[INFO] Downloads completed. Please note that some photos were not
downloaded as they were not in the correct format (e.g. jpg, jpeg, png)”)
```

4.2.3 patch.py

```
#!/usr/bin/env python3
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.common.exceptions import WebDriverException, SessionNotCre-
```

```
atedException
import sys
import os
import urllib.requestimport reimport zipfileimport stat
from sys import platform
def webdriver_executable():
if platform == ”linux” or platform == ”linux2” or platform == ”darwin”:
return ’chromedriver’
return ’chromedriver.exe’
def download_lastest_chromedriver(current_chrome_version=””):
def get_platform_filename():
filename = ”
is_64bits = sys.maxsize > 2**32
if platform == ”linux” or platform == ”linux2”:
# linux filename += ’linux’
filename += ’64’ if is_64bits else ’32’
elif platform == ”darwin”:
# OS X
filename += ’mac64’
elif platform == ”win32”:
# Windows...
filename += ’win32’
filename += ’.zip’
return filename
# Find the latest chromedriver, download, unzip, set permissions to executable.
result = False
try:
url = ’https://chromedriver.chromium.org/downloads’
base_driver_url = ’https://chromedriver.storage.googleapis.com/’
file_name = ’chromedriver_’ + get_platform_filename()
pattern = ’https://.*?path=(\d+\.\d+\.\d+\.\d+)’
```

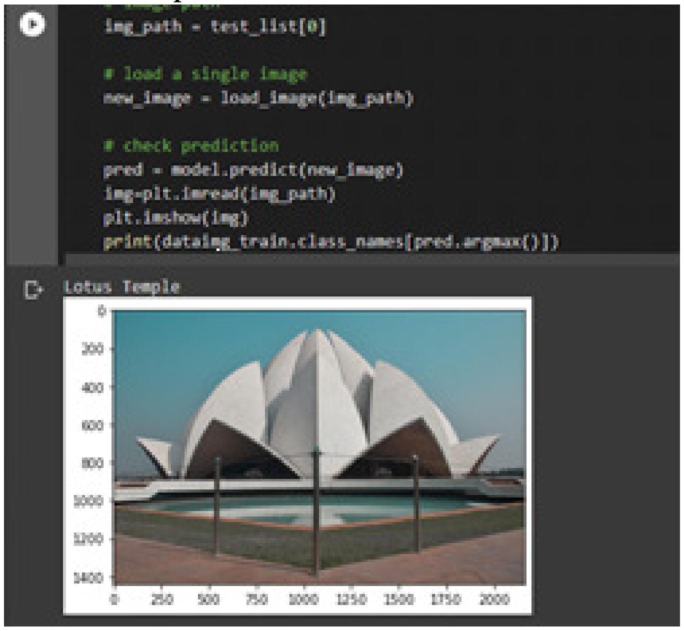
```
# Download latest chromedriver.
stream = urllib.request.urlopen(url)
content = stream.read().decode(’utf8’)
# Parse the latest version.
all_match = re.findall(pattern, content)
if all_match:
# Version of latest driver.
if(current_chrome_version!=””):
print(”[INFO] updating chromedriver”)
all_match = list(set(re.findall(pattern, content)))
current_chrome_version = ”.”.join(current_chrome_version.split(”.”)[:-1])
version_match = [i for i in all_match if re.search(”∧%s”%current_chrome_version,i)]
version = version_match[0]
else: print(”[INFO] installing new chromedriver”)
version = all_match[1]
driver_url = base_driver_url + version + ’/’ + file_name
# Download the file.
print(’[INFO] downloading chromedriver ver: %s: %s’% (version, driver_url))
app_path = os.path.dirname(os.path.realpath(__file__))
chromedriver_path = os.path.normpath(os.path.join(app_path, ’webdriver’, web-
driver_executable()))
file_path = os.path.normpath(os.path.join(app_path, ’webdriver’, file_name))
urllib.request.urlretrieve(driver_url, file_path)
# Unzip the file into folder
with zipfile.ZipFile(file_path, ’r’) as zip_ref:
zip_ref.extractall(os.path.normpath(os.path.join(app_path, ’webdriver’)))
st = os.stat(chromedriver_path)
os.chmod(chromedriver_path, st.st_mode | stat.S_IEXEC)
print(’[INFO] lastest chromedriver downloaded’)
# Cleanup.
os.remove(file_path)
```

```
result = True
except Exception:
print(”[WARN] unable to download lastest chromedriver. the system will use
the local version instead.”)
return result
```

Chapter 5

Output Images

5.1 Lotus Temple

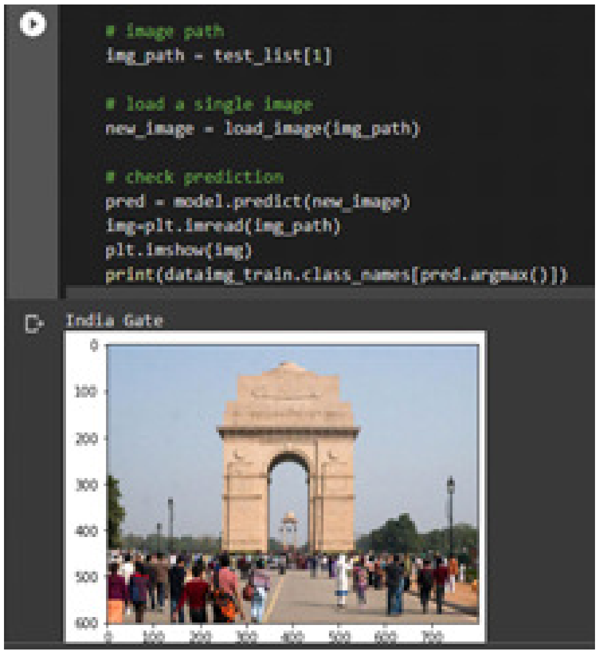


```
img_path = test_list[0]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Lotus Temple
```

Figure 5.1: Lotus Temple

5.2 India Gate

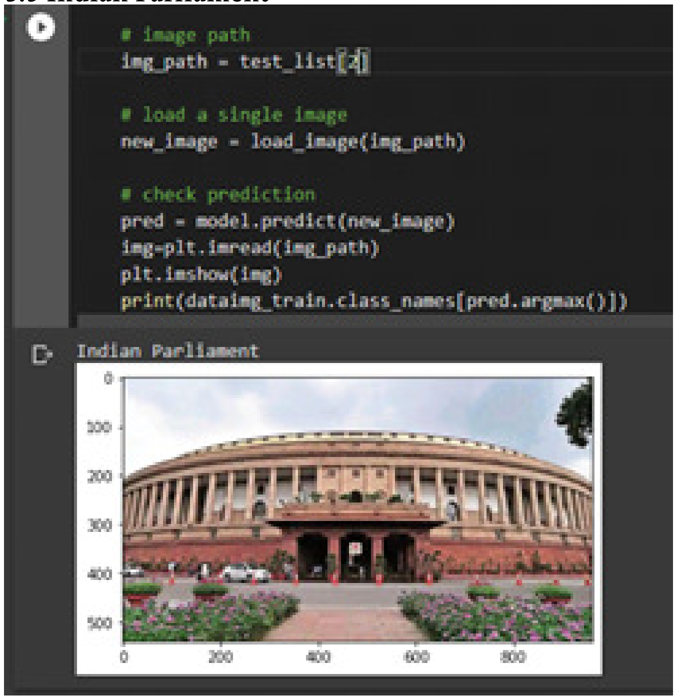


```
# image path
img_path = test_list[1]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
India Gate
```

Figure 5.2: India Gate

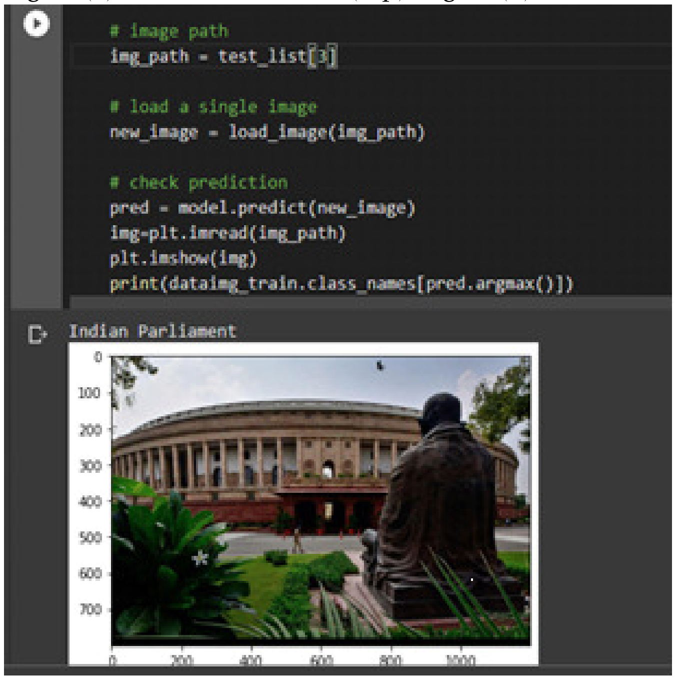
5.3 Indian Parliament



```
# image path
img_path = test_list[2]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Indian Parliament
```

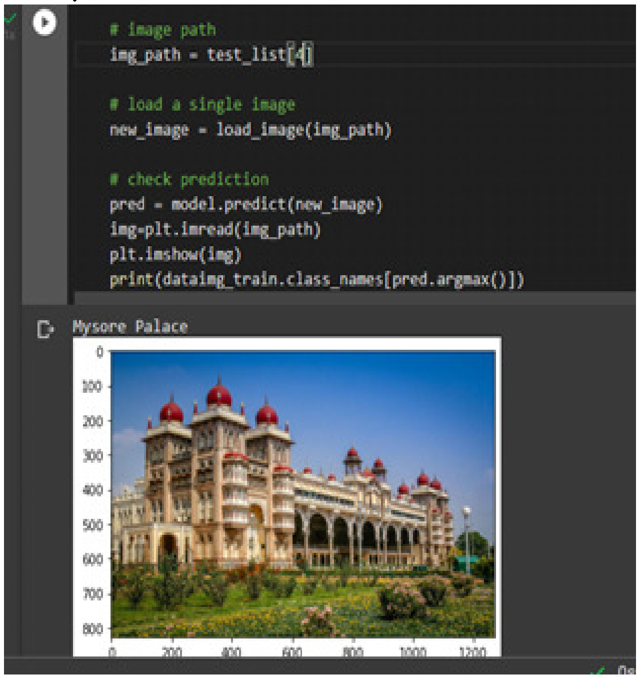
Fig 5.3(a): Indian Parliament (top), Fig 5.3(b): Indian Parliament (below)



```
# image path
img_path = test_list[3]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Indian Parliament
```

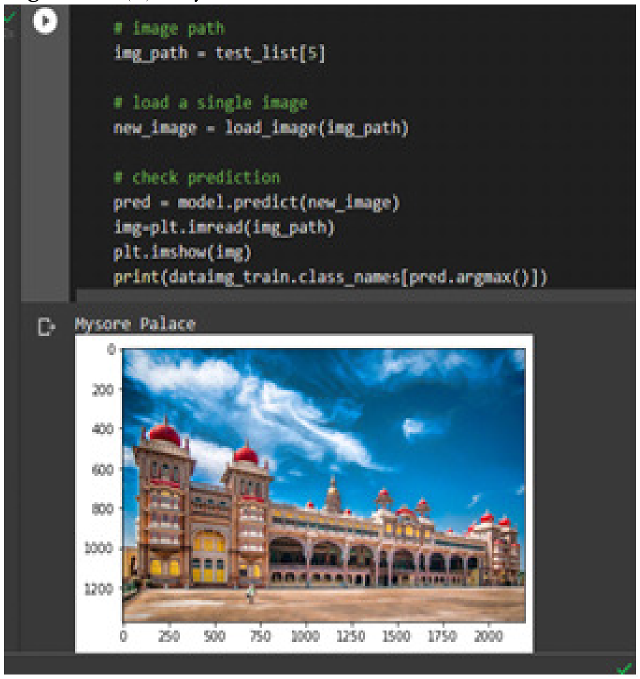
5.4 Mysore Palace



```
# image path
img_path = test_list[4]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Mysore Palace
```

Figure 5.4(a): Mysore Palace

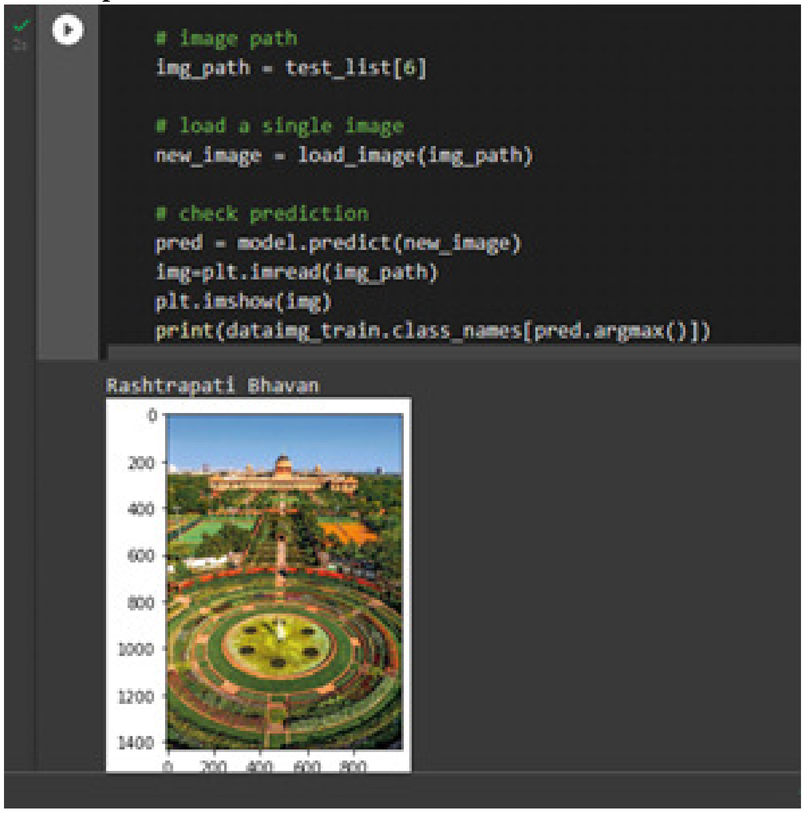


```
# image path
img_path = test_list[5]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Mysore Palace
```

Figure 5.4(b): Mysore Palace

5.5 Rashtrapati Bhavan

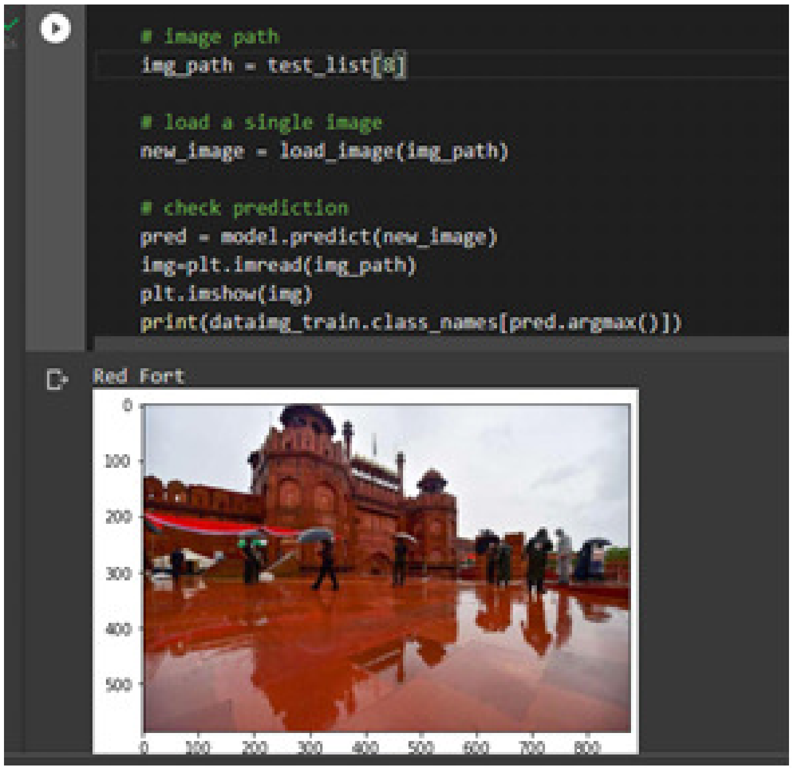


```
# image path
img_path = test_list[6]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Rashtrapati Bhavan
```

Figure 5.5: Rashtrapati Bhavan

5.6 Red Fort

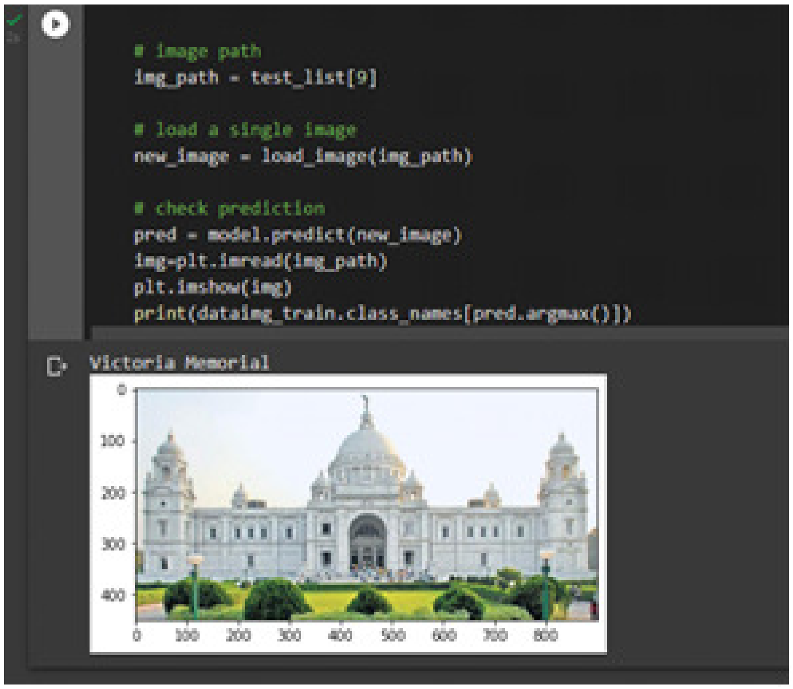


```
# image path
img_path = test_list[8]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Red Fort
```

Figure 5.6: Red Fort

5.7 Victoria Memorial

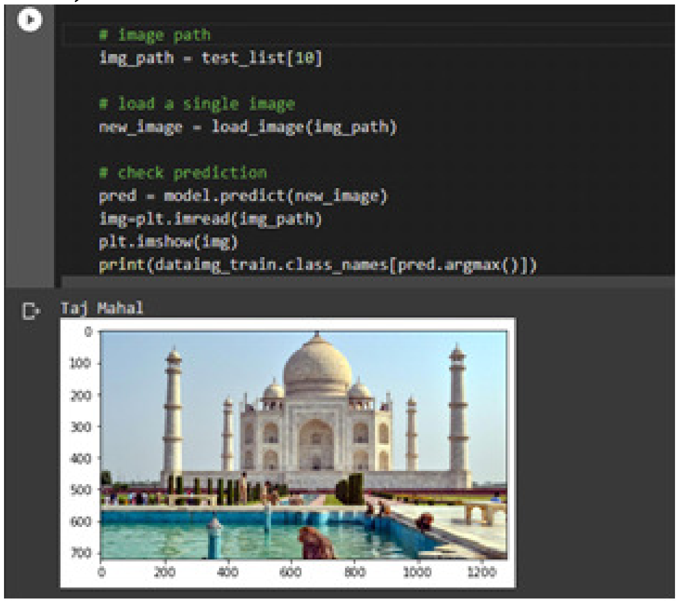


```
# image path
img_path = test_list[9]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Victoria Memorial
```

Figure 5.7: Victoria Memorial

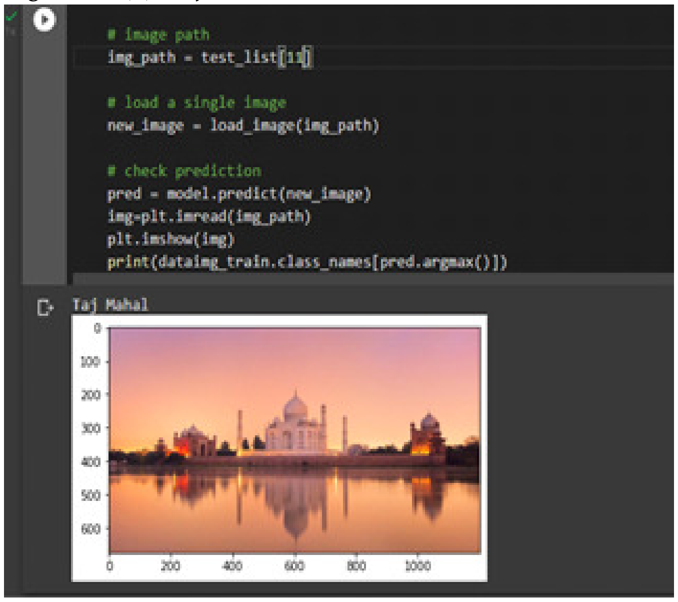
5.8 Taj Mahal



```
# image path
img_path = test_list[10]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Taj Mahal
```

Figure 5.8(a): Taj Mahal

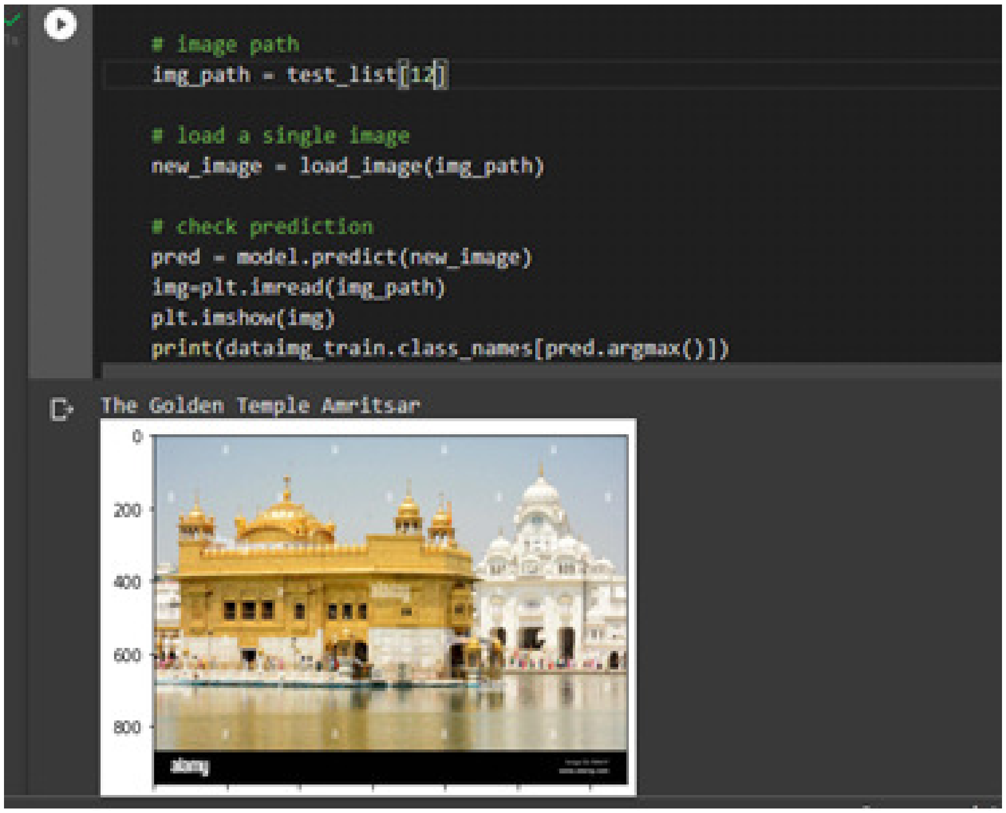


```
# image path
img_path = test_list[11]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Taj Mahal
```

Figure 5.8(b): Taj Mahal

5.9 The Golden Temple Amritsar

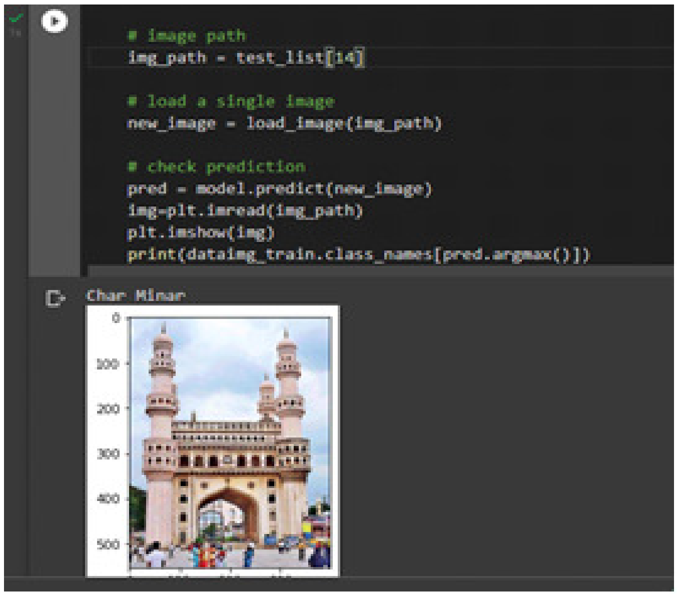


```
# image path
img_path = test_list[12]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
The Golden Temple Amritsar
```

Figure 5.9: The Golden Temple Amritsar

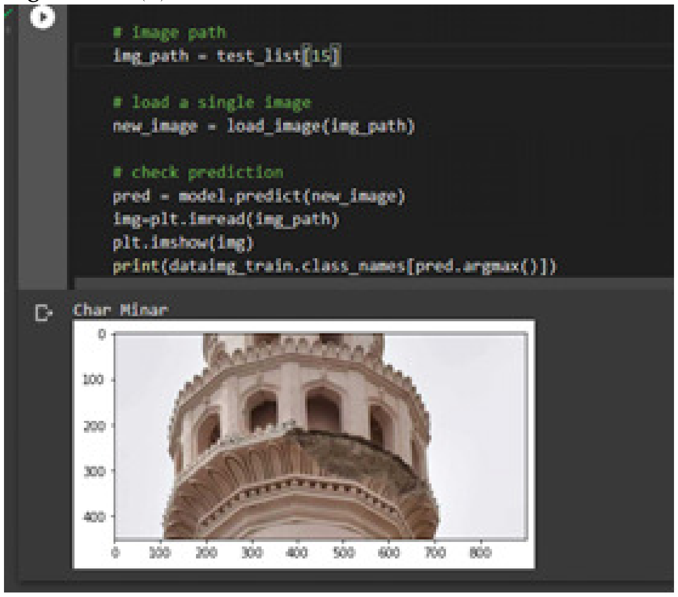
5.10 Char Minar



```
# image path
img_path = test_list[14]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Char Minar
```

Figure 5.10(a): Char Minar

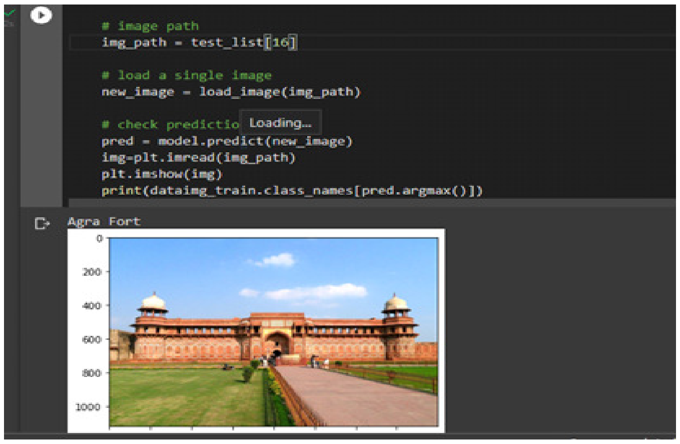


```
# image path
img_path = test_list[15]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Char Minar
```

Figure 5.10(b): Char Minar

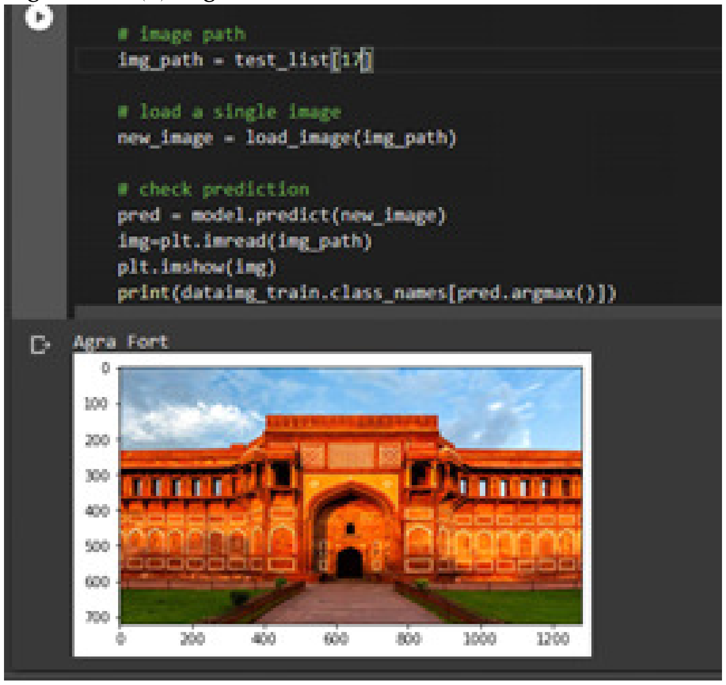
5.11 Agra Fort



```
# image path
img_path = test_list[16]

# load a single image
new_image = load_image(img_path)

# check predictio Loading...
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Agra Fort
```

Figure 5.11(a): Agra Fort

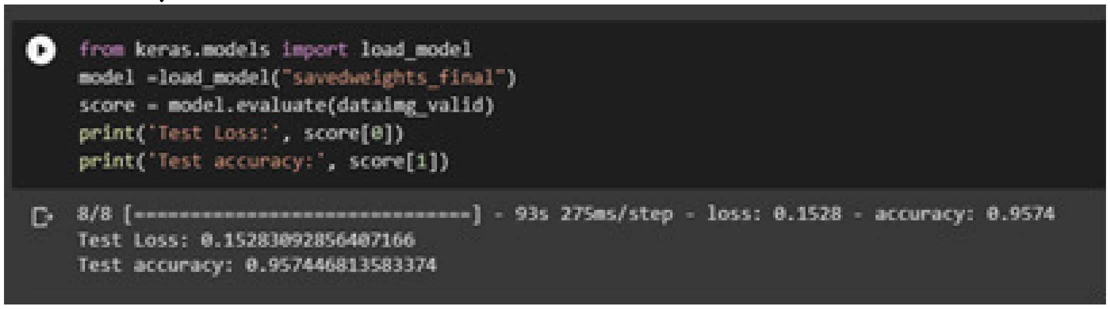


```
# image path
img_path = test_list[17]

# load a single image
new_image = load_image(img_path)

# check prediction
pred = model.predict(new_image)
img=plt.imread(img_path)
plt.imshow(img)
print(dataimg_train.class_names[pred.argmax()])
```

```
Agra Fort
```

Figure 5.11(b): Agra Fort

5.12 Accuracy



```
from keras.models import load_model
model =load_model("savedweights_final")
score = model.evaluate(dataimg_valid)
print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
8/8 [==============================] - 93s 275ms/step - loss: 0.1528 - accuracy: 0.9574
Test Loss: 0.15283092856407166
Test accuracy: 0.9574468135833 74
```

Figure 5.12: Accuracy measurement on validation set

Bibliography

1. ImageNet Classification with Deep Convolutional Neural Networks-https://papers.nips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html

2. Dive Into Deep Learning, d2l.ai