INDIAN CURRENCY RECOGNITION USING PARALLEL COMPUTING

By

Jyoti Tiwari 17CE1135

Rohit Subramanian 17BCE1291

Amrit Omprakash 17BCE1133

A report submitted to the

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING

in partial fulfilment of the requirements for the project of the course of

CSE4014 HIGH PERFORMANCE COMPUTING

in

B.Tech. COMPUTER SCIENCE AND ENGINEERING

VIT University, Chennai

Vandalur – Kelambakkam Road Chennai

600127

October 2019

BONAFIDE CERTIFICATE

Certified that this project report entitled "INDIAN CURRENCY RECOGNITION USING PARALLEL COMPUTING" is a bonafide work of

Rohit Subramanian (17bce1291)

Amrit Omprakash (17bce1133)

Jyoti Tiwari (17bce1135)

who carried out the Project work under my supervision and guidance.

Dr. Ayesha

Professor

School of Computing Science and Engineering

(SCSE), VIT University, Chennai

Chennai – 600 127.

ACKNOWLEDGEMENT

We wish to express our sincere thanks and deep sense of gratitude to our project guide**,** Assistant Professor, School of Computing Science and Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We express our thanks to our Programme Chair Dr. Justus S, Associate Professor for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the School for their support and their wisdom imparted to us throughout the course.

We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

JYOTI ROHIT AMRIT

Abstract

Banknote identification systems, with their wide applications in Automated Teller Machines (ATMs), vending machines and currency recognition aids for the visually impaired, are one of the most widely researched fields today. The paper proposes a novel technique for recognition of Indian currency banknotes by matching an input test currency note to a trained image and identify a similarity match score between the two. The proposed technique is to be evaluated over a large data set for recognition of Indian banknotes of various denominations and physical conditions including new notes, wrinkled notes and non-uniform illumination. A parallelised approach is implemented in order to carry out the above in a quicker and more efficient manner.

Currency recognition system are on of the most researched areas in the field on image processing and Machine Learning. This project proposes a solution for recognition of multiple denominations of Indians Currency using OpenCv and Machine Learning, where the computation is being done in parallel manner.

Introduction

Object recognition is breaking into a wide scope of ventures, with use cases extending from individual security to profitability in the work environment. Object detection and recognition is applied in numerous zones of computer vision, including picture recovery, security, reconnaissance, computerized vehicle frameworks and machine assessment.

Now, currency is an important aspect of everyone's day to day life. Our lives will be at a standstill without the concept of currency. It brings in order to our civilization. Transactions happening around involve the movement of multiple denominations of currency throughout.

Moreover, given the drastic growth in technology in the modern area every single thing is getting automated. Starting from the simplest of objects to the most complex of machines, automation is playing a huge role in the market currently.

Another main problem when it comes to image processing and object detection is the computation time ,power and requirement. To process an image a lot of computing power and time is required. Therefore the efficient use of computing resources will prove to be a great boon in this domain of research. The main idea behind our project is to automate the process of recognition of currency to provide a great and useful solution to automating the recognition of currency that perform the computation in Parallel manner thus increasing the speed and reducing the computation time .

Paper currency recognition (PCR) is an important area of pattern recognition. A system for the recognition of paper currency is one kind of intelligent system which is a very important need of the current automation systems in the modern world of today. The domain of currency recognition has a wide range of real life applications. They are electronic banking, currency monitoring systems, money exchange machines, etc. This project proposes an automatic paper currency recognition system for Indian paper currency using Machine Learning Algorithm KNN(K Nearest Neighbors) in parallel fashion. The system has been implemented in python language. OpenCV has been used for image processing aspects(feature extraction, uncanny edge detection,grayscale conversion,etc) whereas the multithreading library of python is used to achieve the parallelization by the concept of multithreading.

The problem that we are trying to solve here is the recognition of which denomination does the given input note belong to. As we have a classification problem at hand, we decided to go with the K-nearest Neighbors Algorithm. The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems. The KNN algorithm accept that comparable things exist in proximity to eachother . In other words, comparable things are close to one another. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood—calculating the distance between points on a graph.

The features extracted from the image are plotted in a n-dimensional graph. The input image image is processed and the features of the input image are compared with that of the training data. The K close points to the point corresponding to the input image are selected and the final prediction is given to be the majority label of the k points selected.

In the following sections, the motivation behind the project, the implementation details i.e the working of the project ,experiments results and discussion and conclusion and future work are being discussed .

Motivation

Like mentioned before, the domain of object detection and recognition is a booning area of research. In this huge domain , we have selected the sub domain of paper currency recognition system. As discussed above, the scope in this field is drastic. Multiple individuals have come up with different ways to perform automation of currency recogniton. But this is not the main idea behind the project. The novelity of our idea is to come up with a cuurency recognition system that can be run in parallel among mutliple cores of a computer.

There are not many systems that exist as of now that use parallel computing the improvise and enhance the process of comparision of the input note to the already existent test notes.Our goal is to implement such a system that makes efficient comparision,of these and notes and produce accurate results.

Thus the implementation of such PCR systems to perform it’s computation in parallel fashion can increase the response time of the current PCR systems , keeping in mind that currency reognition will mainly be used for counting notes which is a very fast and quick process. Therefore, we can say that the increase in computation time of the recogniton process can prove to be a vital aspect in this domain of automation .

Implementation Details

Technological Stack:

Python: The scripting language that was used to perform this automation is Python 3. Python being a versatile programming language support multiple extensions for image processing and parallel processing .Therefore, this was our preferred choice.

OpenCV: OpenCV (Open source PC vision) is a library of programming capacities primarily went for ongoing computer vision. Initially created by Intel, it was later bolstered by Willow Garage then Itseez (which was later gained by Intel). The library is cross-platform and free for use under the open-source BSD permit. This is one of the most famous libraries for image processing ,hence it is being used here in our project for the various image processing functions like feature extraction, grayscale conversion, uncanny edge detection and other ones.

Multiprocessing python library for multithreading: The multiprocessing python library has been used to implement the parallel computing of the proposed system.

Algorithm:

Serial:

1. Store all the train images of multiple denominations in a folder.
2. Read the input image using open CV and perform orb detect and compute on it(feature detection).
3. Once the feature detection is done ,perform knn matcher on the input features to compare it to the already existing features from the train data. This will compute the distance between the input image and all the denominations (10,20,50,100,200,500,2000) seperately in an iterative manner.
4. The match that returns the lowest distance is the best match for the input image.
5. Thus, the detected denomination is printed

Parallel:

1. Store all the train images of multiple denominations in a folder.
2. Read the input image using open CV and perform orb detect and compute on it(feature detection).
3. Once the feature detection is done ,perform knn matcher on the input features to compare it to the already existing features from the train data. This will compute the distance between the input image and all the denominations (10,20,50,100,200,500,2000) in parallel fashion using threads. Now the click here is to perform the comparisions with all the denominations of the existing by separate threads(7 threads for 7 denominations).
4. All the threads are allocated a specific memory address individually to update the result(goodness of match)
5. The denomination dealt with by the thread that returns the best match is printed as the final ans.

Code:

Utils.py- Python script for all the image processing functions

```
import cv2

import math

import numpy as np

import matplotlib.pyplot as plt
```

```
from pprint import pprint



# read image as is
def read_img(file_name):
img = cv2.imread(file_name)
return img



# resize image with fixed aspect ratio
def resize_img(image, scale):
res = cv2.resize(image, None, fx=scale, fy=scale, interpolation = cv2.INTER_AREA)
return res



# convert image to grayscale
def img_to_gray(image):
img_gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
return img_gray



# gaussian blurred grayscale
def img_to_gaussian_gray(image):
img_gray = cv2.GaussianBlur(img_to_gray(image), (5, 5), 0)
return img_gray



# convert image to negative
```

```
def img_to_neg(image):
img_neg = 255 - image
return img_neg



# binarize (threshold)
# retval not used currently
def binary_thresh(image, threshold):
retval, img_thresh = cv2.threshold(image, threshold, 255, cv2.THRESH_BINARY)
return img_thresh



def adaptive_thresh(image):
img_thresh = cv2.adaptiveThreshold(image, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY, 11, 8)
# cv2.adaptiveThreshold(src, maxValue, adaptiveMethod, thresholdType, blockSize, C[, dst]) →
dsta
return img_thresh



# sobel edge operator
def sobel_edge(image, align):
img_horiz = cv2.Sobel(image, cv2.CV_8U, 0, 1)
img_vert = cv2.Sobel(image, cv2.CV_8U, 1, 0)
if align == 'h':
return img_horiz
elif align == 'v':
return img_vert
else:
```

```
print('use h or v')



# sobel edge x + y
def sobel_edge2(image):
# ksize = size of extended sobel kernel
grad_x = cv2.Sobel(image, cv2.CV_16S, 1, 0, ksize=3, borderType = cv2.BORDER_DEFAULT)
grad_y = cv2.Sobel(image, cv2.CV_16S, 0, 1, ksize=3, borderType = cv2.BORDER_DEFAULT)

abs_grad_x = cv2.convertScaleAbs(grad_x)
abs_grad_y = cv2.convertScaleAbs(grad_y)

dst = cv2.addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0)
return dst



# canny edge operator
def canny_edge(image, block_size, ksize):
# block_size => Neighborhood size
# ksize => Aperture parameter for the Sobel operator

# 350, 350 => for smaller 500
# 720, 350 => Devnagari 500, Reserve bank of India

img = cv2.Canny(image, block_size, ksize)
# dilate to fill up the numbers
#img = cv2.dilate(img, None)
return img
```

```
# laplacian edge
def laplacian_edge(image):
# good for text
img = cv2.Laplacian(image, cv2.CV_8U)
return img


# detect countours
def find_contours(image):
(_, contours, _) = cv2.findContours(image, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
contours = sorted(contours, key = cv2.contourArea, reverse = True)[:5]
return contours


# median blur
def median_blur(image):
blurred_img = cv2.medianBlur(image, 3)
return blurred_img


# dialte image to close lines
def dilate_img(image):
img = cv2.dilate(image, np.ones((5,5), np.uint8))
return img
```

```
# erode image
def close(image):
img = cv2.Canny(image, 75, 300)
img = cv2.dilate(img, None)
img = cv2.erode(img, None)
return img



def harris_edge(image):
img_gray = np.float32(image)


corners = cv2.goodFeaturesToTrack(img_gray, 4, 0.03, 200, None, None,
2,useHarrisDetector=True, k=0.04)
corners = np.int0(corners)


for corner in corners:
x, y = corner.ravel()
cv2.circle(image, (x, y), 3, 255, -1)
return image



# calculate histogram
def histogram(image):
hist = cv2.calcHist([image], [0], None, [256], [0, 256])
# cv2.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])
plt.plot(hist)
plt.show()
```

```
# fast fourier transform
def fourier(image):
f = np.fft.fft2(image)
fshift = np.fft.fftshift(f)
magnitude_spectrum = 20 * np.log(np.abs(fshift))

plt.subplot(121), plt.imshow(image, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])

plt.subplot(122), plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('FFT'), plt.xticks([]), plt.yticks([])

plt.show()



# calculate scale and fit into display
def display(window_name, image):
screen_res = 1440, 900          # MacBook Air

scale_width = screen_res[0] / image.shape[1]
scale_height = screen_res[1] / image.shape[0]
scale = min(scale_width, scale_height)
window_width = int(image.shape[1] * scale)
window_height = int(image.shape[0] * scale)

# reescale the resolution of the window
cv2.namedWindow(window_name, cv2.WINDOW_NORMAL)
cv2.resizeWindow(window_name, window_width, window_height)
```

```
# display image
cv2.imshow(window_name, image)

# wait for any key to quit the program
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Serial code:

```
from utils import *
from matplotlib import pyplot as plt



from multiprocessing import Process, Queue
import time
import subprocess
from gtts import gTTS




max_val = 8
max_pt = -1
max_kp = 0



orb = cv2.ORB_create()
```

```
#test_img = read_img('files/test_100_2.jpg')

#test_img = read_img('files/test_2000_2.jpg')

test_img = read_img('files/test_2000_3.jpeg')

#test_img = read_img('files/test_100_3.jpg')

#test_img = read_img('files/test_20_4.jpg')



# resizing must be dynamic

original = resize_img(test_img, 0.4)

display('original', original)



# keypoints and descriptors

# (kp1, des1) = orb.detectAndCompute(test_img, None)

(kp1, des1) = orb.detectAndCompute(test_img, None)



training_set = ['files/20.jpg', 'files/50.jpg', 'files/100.jpg',
'files/500.jpg','files/200.jpg','files/2000.jpg','files/10.jpg']



start=time.time()



for i in range(0, len(training_set)):

# train image

train_img = cv2.imread(training_set[i])
```

```
(kp2, des2) = orb.detectAndCompute(train_img, None)



# brute force matcher

bf = cv2.BFMatcher()

all_matches = bf.knnMatch(des1, des2, k=2)



good = []

# give an arbitrary number -> 0.789

# if good -> append to list of good matches

for (m, n) in all_matches:

if m.distance < 0.789 * n.distance:

good.append([m])



if len(good) > max_val:

max_val = len(good)

max_pt = i

max_kp = kp2



print(training_set[i], ' ', len(good))



end=time.time()

print()
```

```
if max_val != 8:
note=str(training_set[max_pt])[6:-4]
print('Detected denomination: Rs. ', note)



else:
print('No Matches')

time=end-start
print()
print('Execution time:',time)
```

Parallel code:

```
# -- coding: utf-8 --
"""
Created on Thu Oct 31 11:56:33 2019

@author: bioni
"""

from utils import *
from matplotlib import pyplot as plt
import multiprocessing as mp
from multiprocessing import Process, Queue,Pool

import glob
import threading
```

```
import subprocess
import cv2


g=[0,0,0,0,0,0,0]
def f1():
    train='files/10.jpg'
    max_val = 8
    max_pt = -1
    max_kp = 0
    train_img = cv2.imread(train)
    (kp2, des2) = orb.detectAndCompute(train_img, None)


# brute force matcher
    bf = cv2.BFMatcher()
    all_matches = bf.knnMatch(des1, des2, k=2)
    good = []
# give an arbitrary number -> 0.789
# if good -> append to list of good matches
    for (m, n) in all_matches:
        if m.distance < 0.789 * n.distance:
            good.append([m])


    if len(good) > max_val:
        max_val = len(good)
        max_kp = kp2


    train1 = len(good)
    global g
```

```
    g[0]=train1
    print(train, ':' ,train1)



def f2():
    train='files/20.jpg'
    max_val = 8
    max_pt = -1
    max_kp = 0
    train_img = cv2.imread(train)
    (kp2, des2) = orb.detectAndCompute(train_img, None)

# brute force matcher
    bf = cv2.BFMatcher()
    all_matches = bf.knnMatch(des1, des2, k=2)
    good = []
# give an arbitrary number -> 0.789
# if good -> append to list of good matches
    for (m, n) in all_matches:
        if m.distance < 0.789 * n.distance:
            good.append([m])

    if len(good) > max_val:
        max_val = len(good)
        max_kp = kp2



    train1 = len(good)
```

```
    global g
    g[1]=train1
    print(train, ':' ,train1)



def f3():
    train='files/50.jpg'
    max_val = 8
    max_pt = -1
    max_kp = 0
    train_img = cv2.imread(train)
    (kp2, des2) = orb.detectAndCompute(train_img, None)

# brute force matcher
    bf = cv2.BFMatcher()
    all_matches = bf.knnMatch(des1, des2, k=2)
    good = []
# give an arbitrary number -> 0.789
# if good -> append to list of good matches
    for (m, n) in all_matches:
        if m.distance < 0.789 * n.distance:
            good.append([m])

    if len(good) > max_val:
        max_val = len(good)
        max_kp = kp2
```

```
    train1 = len(good)
    global g
    g[2]=train1
    print(train, ':' ,train1)



def f4():
    train='files/100.jpg'
    max_val = 8
    max_pt = -1
    max_kp = 0
    train_img = cv2.imread(train)
    (kp2, des2) = orb.detectAndCompute(train_img, None)

# brute force matcher
    bf = cv2.BFMatcher()
    all_matches = bf.knnMatch(des1, des2, k=2)
    good = []
# give an arbitrary number -> 0.789
# if good -> append to list of good matches
    for (m, n) in all_matches:
        if m.distance < 0.789 * n.distance:
            good.append([m])

    if len(good) > max_val:
        max_val = len(good)
        max_kp = kp2
```

```
    train1 = len(good)
    global g
    g[3]=train1
    print(train, ':' ,train1)

def f5():
    train='files/200.jpg'
    max_val = 8
    max_pt = -1
    max_kp = 0
    train_img = cv2.imread(train)
    (kp2, des2) = orb.detectAndCompute(train_img, None)

# brute force matcher
    bf = cv2.BFMatcher()
    all_matches = bf.knnMatch(des1, des2, k=2)
    good = []
# give an arbitrary number -> 0.789
# if good -> append to list of good matches
    for (m, n) in all_matches:
        if m.distance < 0.789 * n.distance:
            good.append([m])

    if len(good) > max_val:
        max_val = len(good)
        max_kp = kp2
```

```
    train1 = len(good)
    global g
    g[4]=train1
    print(train, ':' ,train1)


def f6():
    train='files/500.jpg'
    max_val = 8
    max_pt = -1
    max_kp = 0
    train_img = cv2.imread(train)
    (kp2, des2) = orb.detectAndCompute(train_img, None)

# brute force matcher
    bf = cv2.BFMatcher()
    all_matches = bf.knnMatch(des1, des2, k=2)
    good = []
# give an arbitrary number -> 0.789
# if good -> append to list of good matches
    for (m, n) in all_matches:
        if m.distance < 0.789 * n.distance:
            good.append([m])

    if len(good) > max_val:
        max_val = len(good)
        max_kp = kp2

    train1 = len(good)
```

```
    global g
    g[5]=train1
    print(train, ':' ,train1)


def f7():
    train='files/2000.jpg'
    max_val = 8
    max_pt = -1
    max_kp = 0
    train_img = cv2.imread(train)
    (kp2, des2) = orb.detectAndCompute(train_img, None)


# brute force matcher
    bf = cv2.BFMatcher()
    all_matches = bf.knnMatch(des1, des2, k=2)
    good = []
# give an arbitrary number -> 0.789
# if good -> append to list of good matches
    for (m, n) in all_matches:
        if m.distance < 0.789 * n.distance:
            good.append([m])


    if len(good) > max_val:
        max_val = len(good)
        max_kp = kp2


    train1 = len(good)
    global g
```

```
    g[6]=train1
    print(train, ':' ,train1)




max_val = 8
max_pt = -1
max_kp = 0

orb = cv2.ORB_create()

#test_img = read_img('files/test_100_2.jpg')
#test_img = read_img('files/test_2000_2.jpg')
test_img = read_img('files/test_2000_3.jpeg')
#test_img = read_img('files/test_100_3.jpg')
#test_img = read_img('files/test_20_4.jpg')

# resizing must be dynamic
# resizing must be dynamic
original = resize_img(test_img, 0.4)
display('original', original)
```

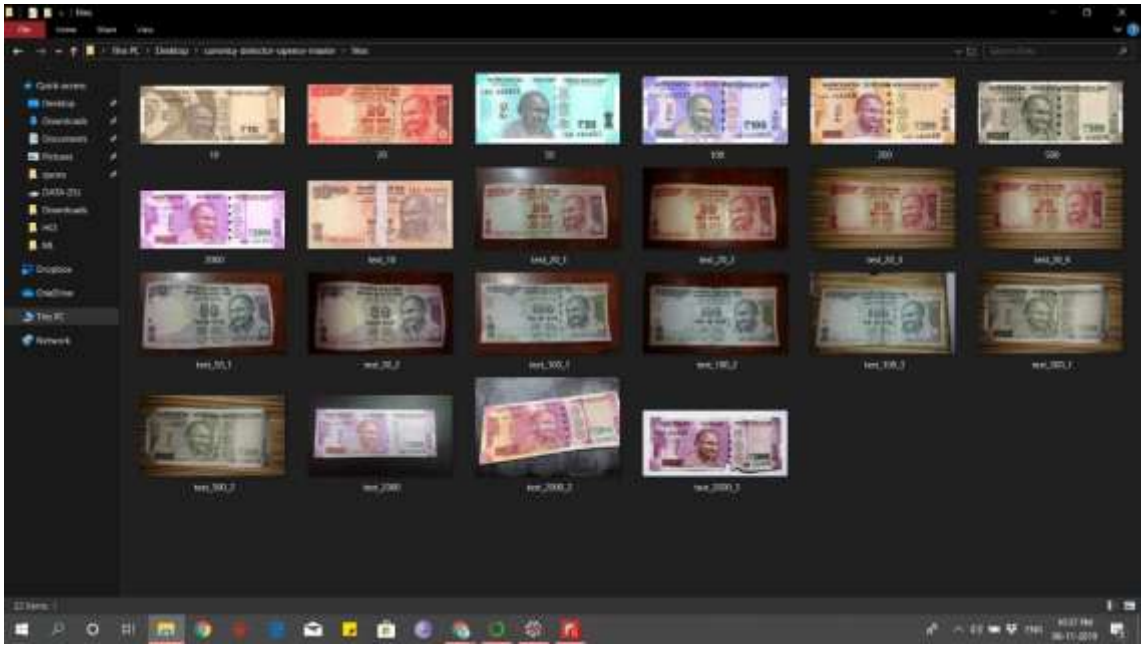
```
# keypoints and descriptors
# (kp1, des1) = orb.detectAndCompute(test_img, None)
(kp1, des1) = orb.detectAndCompute(test_img, None)



t1 = threading.Thread(target=f3)
t2 = threading.Thread(target=f4)
t3 = threading.Thread(target=f1)
t4 = threading.Thread(target=f2)
t5 = threading.Thread(target=f5)
t6 = threading.Thread(target=f6)
t7 = threading.Thread(target=f7)

    # starting thread 1
t1.start()
    # starting thread 2
t2.start()



    # starting thread 1
t3.start()
    # starting thread 2
t4.start()

t5.start()
t6.start()
t7.start()
```

```
    # wait until thread 1 is completely executed
t1.join()
    # wait until thread 2 is completely executed
t2.join()

  # wait until thread 1 is completely executed
t3.join()
    # wait until thread 2 is completely executed
t4.join()

t5.join()
t6.join()
t7.join()
val=g.index(max(g))
if(val==0):
    print("Detected Denomination:10")
if(val==1):
    print("Detected Denomination:20")
if(val==2):
    print("Detected Denomination:50")

if(val==3):
    print("Detected Denomination:100")

if(val==4):
    print("Detected Denomination:200")

if(val==5):
```

```
    print("Detected Denomination:500")
if(val==6):
    print("Detected Denomination:2000")
```

Experimental Results & Discussion

The test data consisted of images of all the old and new notes of denominations 10,20,50,100,200,500,2000. Given below is the picture of the train set.



Now, let's test our system on notes .

Denomination 2000:
Firstly, let's test the system with 2000 denomination. The input picture that is sent to the sytem is :

Serial:

Output of the serial system with an execution time of **0.24S**

```
files/20.jpg   10
files/50.jpg   14
files/100.jpg   6
files/500.jpg   11
files/200.jpg   7
files/2000.jpg   25
files/10.jpg   6

Detected denomination: Rs.  2000

Execution time: 0.23847270011901855
```

Parallel:

Output of the parallel system with an execution time of **0.11S**

```
files/100.jpg : 6
files/50.jpg : 14
files/2000.jpg : 25
files/10.jpg : 6
files/200.jpg : 7
files/500.jpg : 11
files/20.jpg : 10

Detected Denomination:2000

Execution time: 0.1137244701385498
```

Difference in time =0.24-0.11=0.13S

Denomination 100:

Next, let's test the system for 100 rupee notes. The input image is as follows.

Serial:

*Output of the serial system with execution time of **0.24S***

```
files/20.jpg   5
files/50.jpg   8
files/100.jpg   10
files/500.jpg   7
files/200.jpg   9
files/2000.jpg   5
files/10.jpg   6

Detected denomination: Rs.  100

Execution time: 0.23895645141601562
```

Parallel:

Output of the parallel system with an execution time of **0.13S**

```
files/100.jpg : 16
files/10.jpg : 14
files/50.jpg : 6
files/2000.jpg : 5
files/200.jpg : 2
files/20.jpg : 10
files/500.jpg : 2

Detected Denomination:100

Execution time: 0.13164925575256348
```

```
files/100.jpg : 16
files/10.jpg : 14
files/50.jpg : 6
files/2000.jpg : 5
files/200.jpg : 2
files/20.jpg : 10
files/500.jpg : 2

Detected Denomination:100

Execution time: 0.13164925575256348
```

Difference in time =0.24-0.13=0.11S

Denomination 20:

Input image:

Serial:

*Output of the serial system with execution time of **0.52S***

```
files/20.jpg   14
files/50.jpg   8
files/100.jpg   7
files/500.jpg   6
files/200.jpg   5
files/2000.jpg   9
files/10.jpg   6

Detected denomination: Rs.  20

Execution time: 0.523193359375
```

Parallel:

Output of the parallel system with an execution time of **0.21S**

```
files/200.jpg : 5
files/20.jpg : 14
files/2000.jpg : 9
files/50.jpg : 8
files/100.jpg : 7
files/10.jpg : 6
files/500.jpg : 6
Detected Denomination:20
Execution time: 0.20940876007080078
```

Difference in time =0.52-0.21=0.31S

Denomination	Serial	Parallel	Difference
100	0.24S	0.13S	0.11S
2000	0.24S	0.11S	0.13S
20	0.52S	0.21S	0.31S

Average difference=(0.11+0.13+0.31)/3=**0.183S**

Conclusion & Future work

A parallel model for currency recognition system using digital image processing has been successfully created to improve efficiency by producing more accurate results and increased success rates. The proposed technique has been evaluated over various denominations and physical conditions including new notes, wrinkled notes and non-uniform illumination. Various image processing methodologies have been adopted to design and build an efficient recognition system. By using digital image processing and parallel constructs, analysis of paper currency is more efficient on the basis of cost and time consumption compared to existing systems.

As a part of future scope and extension of this project a modular approach with parallelism can be implemented for verification of these currency notes using distinct and unique features of Indian currency notes such as central numeral, RBI seal, colour band and identification mark for the visually impaired and employs algorithms optimized for the detection of each specific feature. These features could be individually verified using separate processes or threads. This project could be further extended to recognise and verify more denominations and for other currencies as well.