FOSSEE SUMMER FELLOWSHIP — PROGRESS PRESENTATION

Scilab Image Processing Toolbox Development

Presenter: Arjun E Naik | **Mentor:** Ms. Rashmi Patankar | **Guide:** Prof. Prabhu Ramachandran

Project Overview

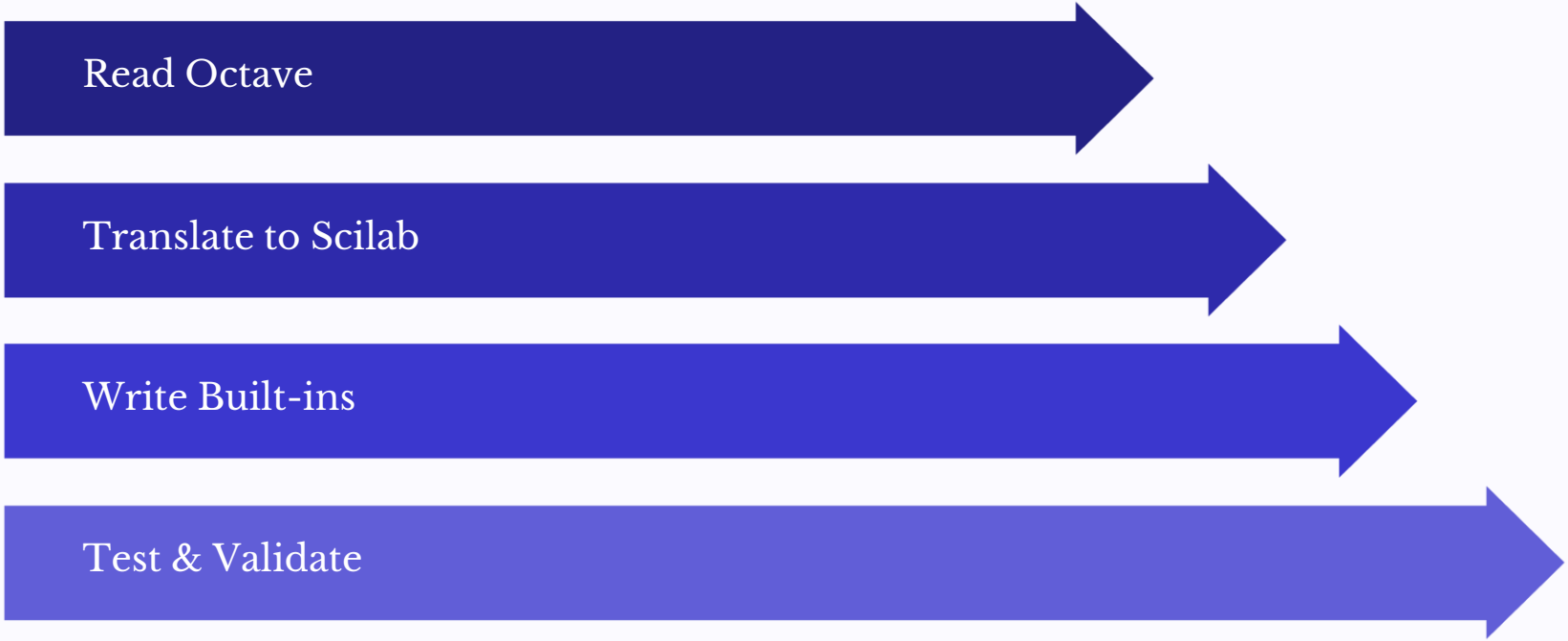
The objective of this project to **develop native Scilab implementations of Image Processing Toolbox functions** by translating and optimizing Octave equivalents.

This fellowship focus on preserving Octave compatibility while adopting algorithms to Scilab's architechture and programming model.

Core Motivation

- Expand Scilab Open-Source capabilities.
- Improve runtime performance
- Support logical and image data types natively
- Enable reproducible, self-contained Scilab code

Development Workflow



Each function follows this disciplined pipeline. Cross-validation against Octave output is the final gate before a function is marked complete.

Functions

Function	Description
`entropy`	Shannon entropy of image histogram
`makelut`	Creation of Lookup table given condition
`entropyfilt`	Local entropy filtering over sliding neighborhoods
`rangefilt`	Local range (max − min) filtering
`stdfilt`	Local standard deviation filtering
`applylut`	LUT-based neighborhood transform
`bweuler`	Euler number computation for binary images
`checkerboard`	Synthetic checkerboard test-pattern generator
`roicolor`	Region-of-interest extraction via hue/color range
`grayslice`	Partitions grayscale intensity range into intervals
`fftconv2`	2-D convolution using FFT
`otsuthresh`	Global threshold computation using Otsu's method
`wavelength2rgb`	Converts visible light wavelengths into RGB
`bwmorph`	Morphological operations on binary images

Functions:

1.**entropy()** - measures the amount of information (randomness) present in an image.

Calling Sequence:
Out = entropy(I,nbins);
whre,
I – real or logical array
nbins – positive integer scalar

Output:

E – scalar double

2.**makelut()** - pre-computes the output of an arbitrary neighbourhood function for *every possible* binary input pattern, storing the results in a vector that can later be applied pixel-by-pixel at constant cost

Calling Sequence:
Out = makelut(fun,n);
whre,
fun – condition function to create lookup table.
n - neighbhourhood length.

Output:

lut – column vector of doubles, length $2^{(n^2)}$

3.**entropyfilt()** - slides a neighbourhood window (called **domain**) across every pixel of an image and computes the **Shannon entropy** of the intensity distribution within that window.

Calling Sequence:
E = entropyfilt(I);
E = entropyfilt(I, domain);
E = entropyfilt(I, domain, padding);

whre,
I – numeric or logical matrix
domain – numeric matrix (defalt ones(9,9))
Padding – padding to image (default – symmetric)

Output:

E – Local entropy at every pixel (bits). Size same as I.

Functions:

4.**rangefilt()** - non-linear spatial image processing operation that replaces each pixel with the **local range** — the difference between the maximum and minimum pixel values within a defined neighbourhood.

Calling Sequence:
retval = rangefilt(I)
retval = rangefilt(I, domain)
retval = rangefilt(I, domain, padding)
whre,
I – real or logical array
domain – numeric or logical matrix
Padding – pad to image.

Output:

E – double matrix.

5.**stdfilt()** - slides a neighbourhood window (called the **domain**) across every pixel of an image and computes the **local standard deviation** of intensity values within that window

Calling Sequence:
S = stdfilt(I)
S = stdfilt(I, domain)
S = stdfilt(I, domain, padding)
whre,
I – real or logical array
domain – numeric or logical matrix
Padding – pad to image.

Output:

lut –doubles matrix,same length as I.

6.**applylut()** - applies a **neighbourhood look-up table (LUT)** to a binary image. For every pixel in the image it examines the pixel's neighbourhood, converts that neighbourhood into an integer index, and replaces the pixel's output value with the entry at that index in the LUT

Calling Sequence:
A = applylut(BW, LUT)
whre,
BW – logical matrix (any size >=3)
LUT – numeric column vector length of 512 .

Output:

A – double matrix . Same size as BW.

Functions:

7.**bweuler()** - The **Euler number** (also called the Euler characteristic or Euler–Poincaré number) is a fundamental topological property of a binary image.
E = C - H
where **C** is the number of connected foreground components (objects) and **H** is the total number of enclosed holes across all objects. Background regions that touch the image border
are **not** counted as holes

Calling Sequence:
eul = bweuler(BW)
eul = bweuler(BW, n)

whre,
BW – real or logical matrix.
n – integer scaler (4 or 8)

Output:

E –scaler double

8.**checkerboard()** - generates a synthetic **checkerboard test pattern**

Calling Sequence:
checkerboard(side, M, N, P, ...);

whre,
side – non-negative integer (default 10)

M , N,P – single numeric integer.
No. of tiles along eah dimension.

Output:

lut –doubles matrix,2d or nd.

9.**roicolor()** - **Region of Interest by Colour** (roicolor) produces a binary mask that identifies every element in an input array whose intensity value either lies within a closed range or matches one of a set of discrete target values.

Calling Sequence:
BW = roicolor(A, low, high)
BW = roicolor(A, v)

whre,
BW – logical or numeric matrix
Low – lower bound.
High – upper bound.
V – numeric vector row or column..

Output:

BW – double matrix . Same size as A.

Functions:

10.**grayslice()** - partitions the intensity range of a grayscale image into a set of **intervals** (or **levels**) and produces a labeled image in which each pixel is replaced by the index of the interval it falls into.
Scalar input n → divides the range into n equally spaced levels.
Vector input v → uses the provided thresholds to define irregular intervals.

Calling Sequence:
s = grayslice(I);
s = grayslice(I,n);

whre,
I – numeric matrix.
n – integer scaler or vector (default 10)

Output:

E –uint8 or double labled image.

11.**fftconv2()** - performs **2-D convolution** using the Fast Fourier Transform (FFT). The convolution is computed in the frequency domain, which is significantly faster than spatial convolution for large kernels

Calling Sequence:
X = fftconv2(a, b)
X = fftconv2(a, b, shape)
X = fftconv2(v1, v2, a)
X = fftconv2(v1, v2, a, shape)

whre,
a,b – numeric or logical matrices.

V1,v2 – numeric vectors.

Shape – for same size output (default :"full")
Output:

X –doubles matrix.

12.**otsuthresh()** - computes the **global threshold** of a histogram using **Otsu's thresholding method**. The algorithm determines the threshold that maximizes the **between-class variance**, thereby separating the histogram into two classes with maximum discrimination.

Calling Sequence:
T = otsuthresh(hist)
whre,
hist - numeric vector – histogram.
Output:

X –normalised otsu threshold between 0 and 1.

Functions:

13.**wavelength2rgb** - wavelength2rgb converts one or more **visible light wavelengths** (measured in nanometers) into their corresponding **RGB colour representation**

Calling Sequence:
rgb = wavelength2rgb(wavelength)
rgb = wavelength2rgb(wavelength, out_class)
rgb = wavelength2rgb(wavelength, out_class, gamma)
where,

wavelength – numeric array in nano meters.
Out_class – double,uint8,uint16 datatypes.
Gamma – scalar numeric. Gamma correction factor.

Output:
rgb– numeric array rgb representation of input wavelengths

14.**bwmorph()** - Apply a named morphological transform to a binary image, n times (or until convergence with n = %inf)

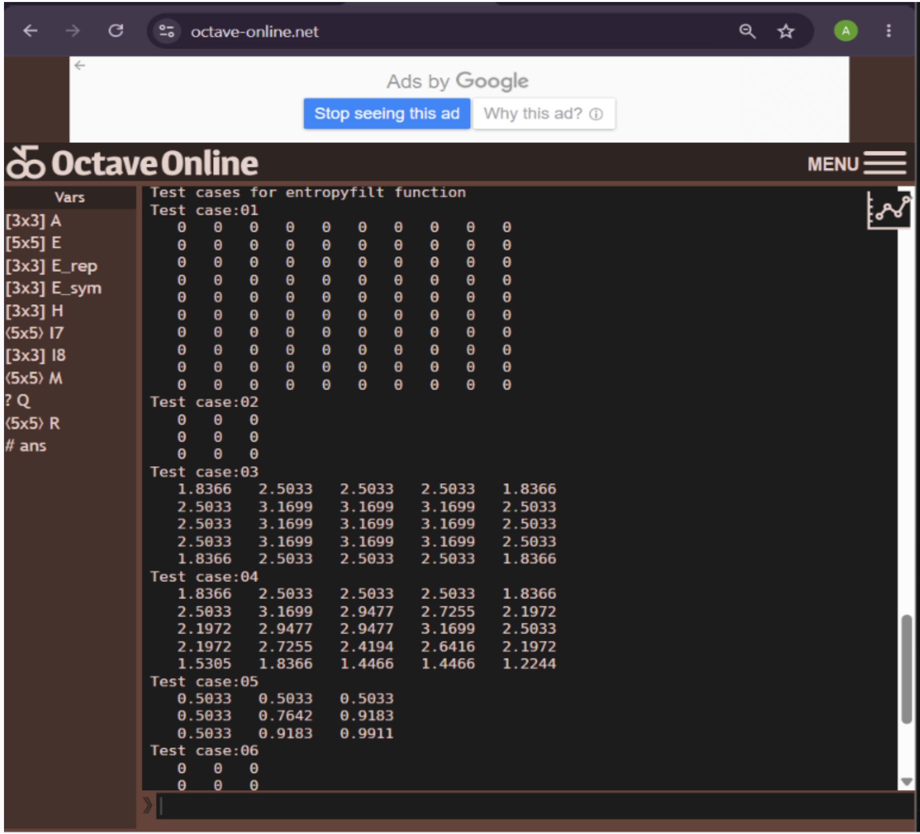
Calling Sequence:
bw2 = bwmorph(bw, operation)
bw2 = bwmorph(bw, operation, n)
whre,
bw – input binary image.
Operation name – like erode,dialate,thin,tophat,bothat etc.

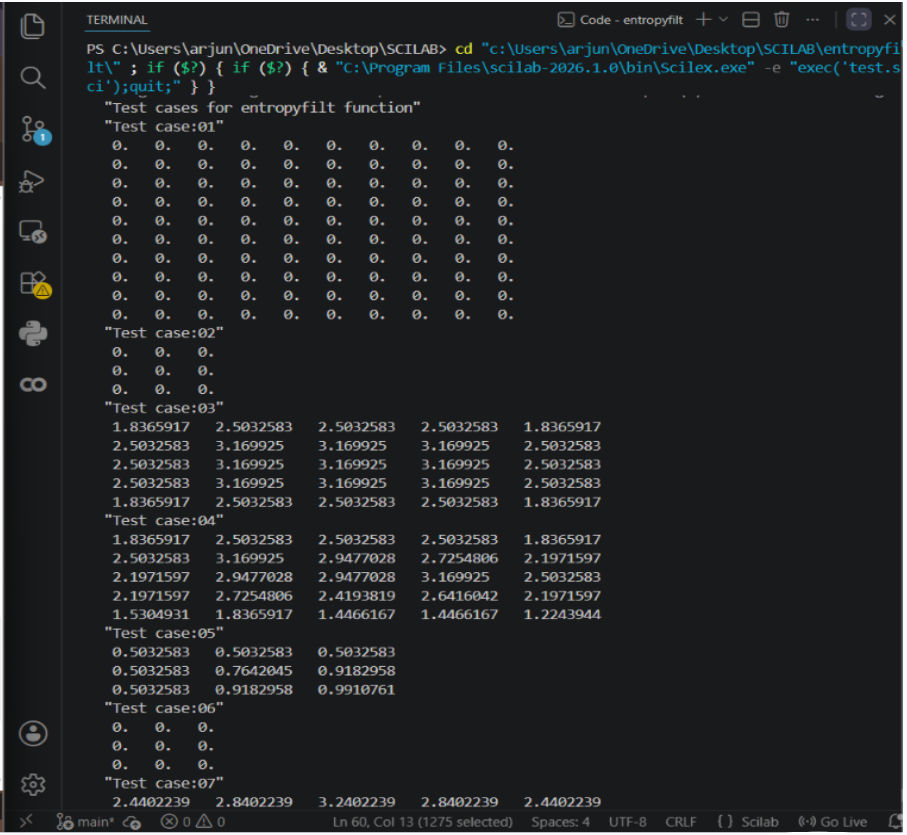
N –no of iterations.

Output:

bw2 –Boolean matrix.

Example function execution: entropyfilt()





Key Technical Challenges

No Native Logical Type

Scilab lacks a native `logical` type — explicit `bool2s()` conversion required throughout.

Argument Count Mismatch

Octave's `nargin`/`nargout` map to Scilab's `argn(2)`/`argn(1)`.

Convolution Kernel Orientation

Column-major reshaping and `conv2` kernel flipping required 180° pre-rotation in `applylut`.

Empty-Dimension Array Collapse

Scilab's `zeros()` collapses empty-dimension arrays to `0×0`, unlike Octave which preserves shape.

Missing Built-ins

Functions like `im2uint8`, `imhist`, `immse`, and `getrangefromclass` required custom narrow-scope helpers.

Naming Collisions

Scilab's flatter namespace caused collisions with existing macros (e.g., `cell2mat`), requiring renaming.

Documentation Approach

Every function ships with a consistent `README` following a five-part template, ensuring reproducibility and ease of adoption in lab and classroom settings.

01

Overview

Concise description of function purpose and mathematical basis.

02

Calling Sequence

All valid syntax variants with parameter signatures.

03

Parameter Reference

Type, shape, and valid range for each input and output.

04

Worked Examples

Reproducible Scilab snippets with expected console outputs.

05

Octave Compatibility Notes

Known behavioural differences, edge cases, and limitations.

Future Scope & Suggested Developments

Deep Learning Extension

Integrate deep learning interference binding directly With image toolbox to support native object detection And segmentation tasks.

Develop the function has computational advantage

Reduce the time and space complexity with modern discovered techniques.

Resolve Remaining Compatibility Issues

Investigate and address outstanding FOSSEE Scilab-Octave Toolbox compatibility gaps to reduce bridge dependency further.

Performance Benchmarking

Systematically benchmark native Scilab implementations against Octave-bridged calls to quantify performance gains.