DFFML: Integrating Image Processing into DFFML

Python Software Foundation: Data Flow Facilitator for Machine Learning

About Me

Name	Saksham Arora
Github	sakshamarora1
Gitter	sakshamarora1
Email	
Time Zone	IST (GMT +5:30)
Phone	
LinkedIn	Saksham Arora

Education and Background

University	Maharaja Agrasen Institute of Technology, GGSIPU
Location	Delhi, India
Program	B.Tech in Information Technology
Year	2nd
Expected Graduation	2022

Code Contribution in DFFML

Contribution	Issue	Pull Request	Status
Added logging for testing It was my first PR for this project so I was getting familiar with the codebase of the project and how things work.	#156	#161	Closed, Merged
Replaced -features with -model-features Updated the documentation to reflect the changes made in the code and get a better understanding of the workflow.	#250	#251	Closed, Merged
Resolved negative value error for n_jobs parameter Found why the n_jobs parameter for the scikit models wasn't able to take negative value as an argument and helped in fixing the code.	#191	#266	Closed, Merged
Added new Scikit Models Added 15 new Scikit Models for both Classification(6) and Regression(9). List of Added Models.	#272	#276	Closed, Merged
Added IDX1 and IDX3 source files Found a way to read the MNIST dataset from the IDX source files and train the existing models on them for Digit Prediction.	-	#294	Closed, Merged
Reading PNG Images as arrays for MNIST Prediction Used the Python Imaging Library (PIL) to read PNG images as arrays of the same dimensions as the MNIST dataset for predicting Digits (in PNG format) on models trained on MNIST Dataset.	#371	#399	Closed, Merged

Project Information

1. **Organisation:** Python Software Foundation

2. **Sub-Organisation:** DFFML

3. **Project Abstract:**

Integrating Image Pre-Processing and Computer Vision into DFFML to train, test and predict on already existing machine learning models.

4. **Detailed Description:**

DFFML is a machine learning based project which provides APIs for training and testing datasets using various machine learning frameworks such as scikit-learn, tensorflow, etc; making it easier for anyone to input there datasets to train and test upon no matter the knowledge of the user in programming.

At present, there is no way of training and testing the existing machine learning models in DFFML on image datasets, so I have selected 2 image processing python libraries OpenCV and Scikit-Image to wrap in DFFML.

Most of the machine learning algorithms are very likely to overfit in the training process. Part of the problem is that visual data is very complex, therefore models tend to have high dimensions of input and have to have a lot of parameters to fit. And when there isn't enough training data available, overfitting happens really fast.

The project is divided into 2 parts:

1. Wrapping the Image Processing Libraries
2. Implementing High Level Operations

Note:
Even though I will be wrapping all the sub-modules and functions in OpenCV and Scikit-Image, here are a few functions to provide an idea of the operations that will be performed using them.

Wrapping the Image Processing Libraries

OpenCV

Even though I will be wrapping the complete library, here are some of the important functions in OpenCV that are popularly used for image pre-processing and take the highest priority:

Image Filtering			
bilateralFilter	filter2D	dilate	Sobel
blur	GaussianBlur	morphologyEx	pyrUp
boxFilter	Laplacian	pyrDown	Scharr
buildPyramid	medianBlur	erode	sepFilter2D
sqrBoxFilter	spatialGradient	pyrMeanShiftFiltering	
Geometric Image Transformations			
resize	linearPolar	getAffineTransform	warpPolar
remap	logPolar	warpPerspective	ConvertMaps
undistort	warpAffine		
Image Thresholding			
adaptiveThreshold	threshold		
Drawing Functions			
arrowedLine	ellipse	polylines	drawContours
circle	line	rectangle	drawMarker
Color Space Conversions		Color conversions provided in OpenCV	
Feature Detection			
Canny	cornerHarris	HoughCircles	HoughLines
Miscellaneous			
blendLinear	watershed	Contour Operations	grabCut
distanceTransform	floodFill	matchTemplate	integral
Cascade Classifiers for face and eye detection			

Scikit-Image

A few important features present in Scikit-Image are not available in OpenCV and vice versa. So, here are a few popular functions from this library:

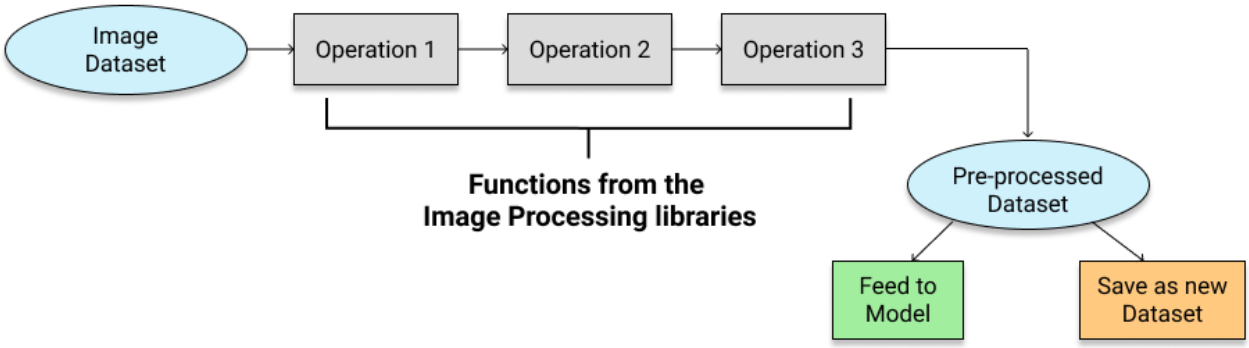
scikit.feature			
canny	peak_local_max	local_binary_pattern	haar_like_feature
daisy	hessian_matrix	Blob functions	Corner functions
scikit.filters			
inverse	sobe	laplace	hessian
wiener	scharr	rank_order	unsharp_mask
gaussian	prewitt	gabor_kernel	meijering
frangi	roberts	threshold	sato
filters.rank			
autolevel	modal	enhance_contrast	otsu
bottomhat	mean	pop	majority
equalize	subtract_mean	sum	entropy
gradient	median	tophat	noise_filter
maximum	minimum	enhance_contrast_percentile	
scikit.morphology			
erosion	skeletonize	max_tree	white_tophat
dilation	watershed	flood_fill	black_tophat
opening	thin	convex_hull	
closing	Remove_small_holes	Remove_small_objects	
scikit.restoration			
wiener	Denoising functions	unwarp_phase	richardson_lucy
scikit.transform			
hough_circle	hough_line	hough_ellpse	integrate
warp	radon	resize	rotate
skimage.util			
Image as any datatype	crop	invert	random_noise

Operation Workflow

The second part of integrating these libraries into DFFML is the operation workflow, that is, defining the operation flow schema for the various features and filters in the above mentioned libraries and defining how the high level operations and low level operations will work and how the user will be able to use these operations.

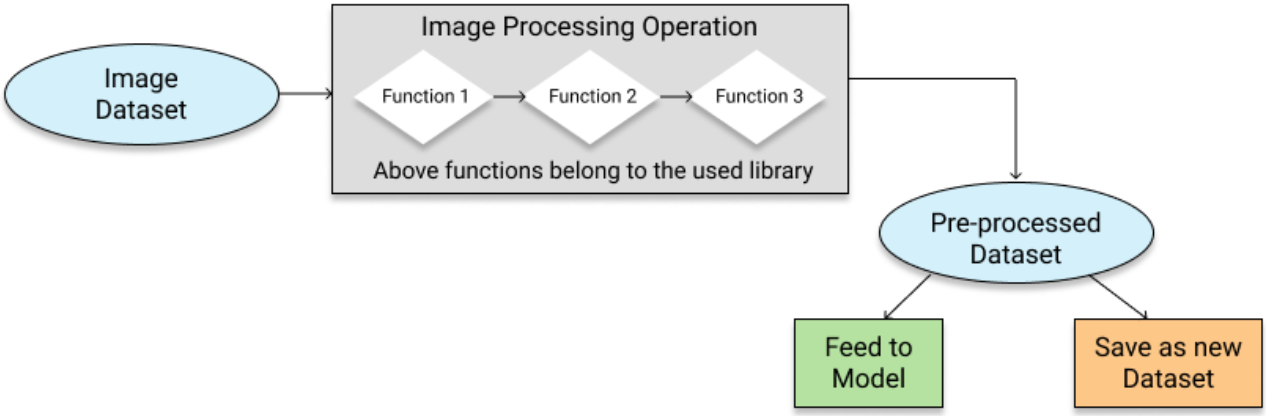
Approach 1:

In this approach, all the image processing functions are treated as individual operations.



Approach 2:

In this approach, there will be only one operation (opencv or scikitimage in this case), to which the functions will be passed as parameters of the larger operation.



I'd like to go with the 2nd approach as it will be a better way for the user to use the operations and will give them more freedom to use the functions available.

Users will be able to implement any number of operations on their image dataset for which the flow will be provided by them.

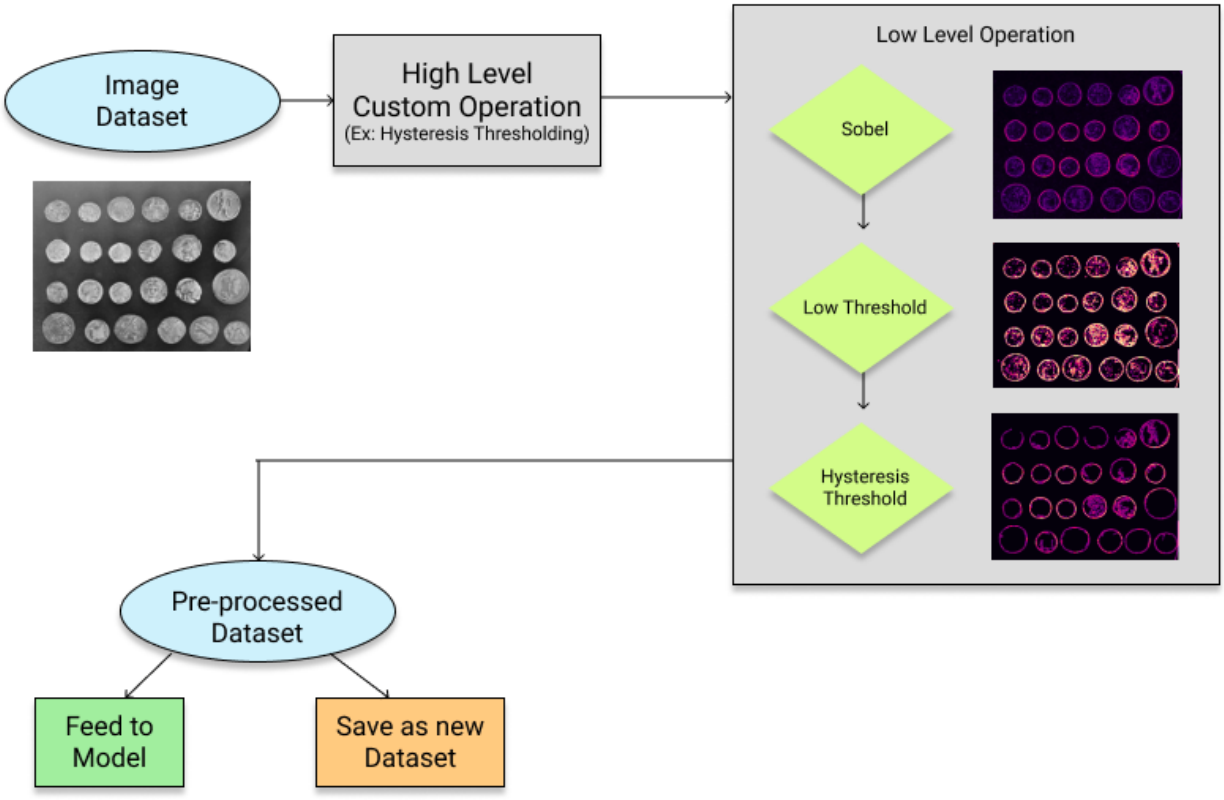
For instance, here are a few examples:

- A simple example would be Image Scaling, the resize feature in Scikit-Image where a trained model will be able to predict on any image regardless of its original aspect ratio.
- Another example would be the Canny edge detection method, in which it returns the edges in the image which can be fed as a feature to the model.

High Level Operations

Other than this, I plan on adding a few Custom Operations which will act as high level operations implementing a predefined flow of the image processing functions (i.e. the low level operations). The user will pass the values of functional parameters of the high level operation.

The working of a custom operation:



A simple example:
For further understanding of High level operations, let us look at a face detection example where:

- Step 1 : Face Detection using Cascade Classifiers.

Image downloaded from https://www.mediaweek.com

- Step 2 : Copying the face rectangles using the coordinates generated from step 1.


and more

- Step 3 : Resizing all the faces to a common aspect ratio.

- Step 4 : Either feed the preprocessed data into a model or save it as a new dataset(using writing/saving function).

These operations could also be implemented manually by the user but providing a custom operation for this task will ease things which is what DFFML targets to achieve.

Dynamic loading of functions :

➤ Scikit Image has the feature skimage.lookfor, which can be used for dynamic loading of the functions instead of importing all the functions all at once at the start.

- ➤ While there is no such feature available in OpenCV, dynamic loading can be done by using the configparser from the python standard library. A *config.ini* file will be used which will contain all the necessary information.
- ➤ If dynamic loading of functions doesn't work out, then there is the option of importing the functions all at once.

With this project, DFFML will be able to train, test and predict on a large number of popular datasets for example, VIsualQA, ImageNet, Open Images Dataset.

Weekly Timeline

Pre-GSoC (Upto May 4):

- Make more contributions through issues and features to further my understanding of codebase.
- Brush up all the necessary topics for the libraries to wrap around DFFML.
- Get more familiar with the operation workflow of the project.

Community Bonding (May 4 - June 1):

- Communicate with the mentors on different ideas for the image processing stuff and get input on how it can be integrated in a user friendly way.
- Communicate with other selected applicants about their projects and how we can be of help to each other.
- Keep working on issues and start working on wrapping OpenCV library.

> Phase 1 of 3 (Wrapping OpenCV)

Week 1 (June 1st - June 8th):

- Start with the wrapping of the most important functions in OpenCV as discussed with the mentors.

Week 2 (June 8th - June 15th):

- Continue wrapping of OpenCV library.
- Start working on the dynamic loading of functions.

Week 3 (June 15th - June 22nd):

- Continue the same as Week 2.
- Start updating the documentation and add examples for image pre-processing.

Week 4 (June 22nd - June 29th):

- Finish the wrapping of OpenCV.
- Finish the documentation of OpenCV functions usage.
- Write test cases for OpenCV.

> Phase 2 of 3 (Wrapping Scikit-Image)

Week 5 (June 29th - July 6th):

- Start wrapping of Scikit-image.
- Start working on the dynamic loading.

Week 6 (July 6th - July 13th):

- Continue the same as Week 5.
- Start updating the documentation and add examples for image pre-processing.

Week 7 (July 13th - July 20th):

- Finish the wrapping of Scikit-Image.
- FInish updating the documentation.
- Write test cases for Scikit-Image.

Week 8 (July 20th - July 27th):

- A spare week to finish anything left in wrapping of OpenCV or Scikit-Image.
- Start preparing for implementation of custom high level operations.

> Phase 3 of 3 (High Level Operations and Documentation)

Week 9 (July 27th - August 3rd):

- Start adding custom High Level Operations as discussed with mentors.

Week 10 (August 3rd - August 10th):

- Continue the same as Week 9.

- Start documenting the working of these operations.

Week 11 (August 10th - August 17th):
- Continue to add new custom high level operations.
- Documenting their working.

Week 12 (August 17th - August 24th):
- Finish working on high level operations.
- Complete the documentation & tests and add examples.

Final Week (August 24th - August 31st):
- Wrap things up if anything is left in wrapping of the libraries of the High level operations.
- Prepare a final summary and organize the work into a presentable form.

Alternate Timeline:
After wrapping OpenCV in the first month, if the functions suffice the need, I will shift the wrapping of Scikit-Image to the 3rd month and will focus on the high level operation workflow.

Stretch Goals

After finishing the wrapping of OpenCV, Scikit-Image and finish the high level operation workflow, I plan to work on the Output Formatting issue #335 as this issue is important when dealing with image datasets.
I will discuss with the mentors and work on the dynamic loading of scikit models.

Other Commitments

- End Semester Examination (For 2 weeks)
  - Due to COVID-19, the dates are uncertain. I will update the mentors a week or 2 before my exams start.
  - Time dedicated to GSoC during exams: 3-4 hours per day.
  - Other than this, I have no exclusive plans for this summer.

Are you applying for other projects in GSoC?

No, I am only applying for this project.

Further Contribution

DFFML is a machine learning based project which aligns with my interest in the field, so I will be more than happy to stay a part of the community even after the GSoC to keep contributing and learning.