# Towards Better Images.jl Ecosystem

Jiuning Chen

Github: Johnnychen94  Slack: Johnny Chen
Mentors*: Zygmunt L. Szpak, Júlio Hoffimann, Tim Holy, Christof Stocker

**Abstract**

This project aims to achieve a better ecosystem for Images.jl, an image-processing toolbox in Julia. Main contributions consist of user-friendly documentation of Images.jl ecosystem, developer manual, and more consistent, robust, and extensible APIs. Moreover, this project also serves as a subproject to bring Images.jl from pre-julia1.0 stage to post-julia1.0 stage, and eventually to Images.jl v1.0.

## Table of Contents



*Except for Dr. Zygmunt, my primary mentor, I listed other mentors according to the chronological order of their acceptance of mentoring this project. Special thanks to Prof. Tim and Mr. Christof (aka evizero), their patient guidance to my several beginning PRs in Julia community are invaluable.

# 1 Project Description

## Project Background

Images.jl is a Julia image-processing toolbox that provides a collection of out-of-box functions[1] to do image processing tasks just like scikit-image and MATLAB Image Processing Toolbox do.

However, despite the not yet benchmarked performance, this toolbox at present is still not friendly to both users and developers. Unlike other mature Julia packages such as JuMP.jl and GPUArrays.jl, Images.jl requires potential users and developers to understand the very details of its mechanism and architecture, and this becomes even harder for them without comprehensive documentation on it. Under this circumstance, many image-processing researchers are still using Python and MATLAB for their daily work.

Some apparent causes for its poor usability are:

- there are few demos or recipes in Images.jl for new users to start with;
- APIs vary greatly across Images.jl submodules, and are unintuitive to the non-experts;
- there's no image-processing-specific style guide on naming and programming, except the Julia style guide;
- there are many temporary helper functions and legacy codes everywhere;
- Images.jl is an ecosystem but it lacks a comprehensive illustration of its packages;
- coverage of trait functions are not fully tested.

Fundamentally this is because the community is still in the progress of finding the most suitable programming style to process images using Julia. We can smell a bad code design from fig. 1, the dependency overview of ImageTransformations.jl, where ImageCore.jl doesn't play its core role well.

Fortunately the problem is well-concerned in the community. Issues such as

- JuliaImages/ImageCore.jl#63 and JuliaMath/FixedPointNumbers.jl#41 – How to deal with overflow behavior of default `N0f8` type?
- JuliaImages/Images.jl#766 – Use `channelview` as possible as we can?

[1] An overview of currently implemented image-processing functionalities is shown at api comparison.

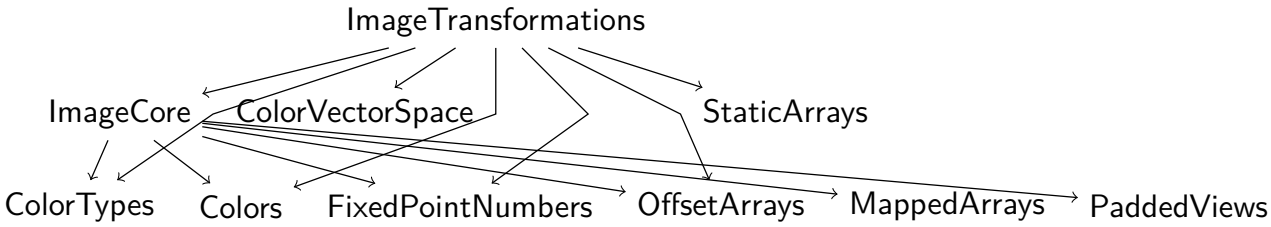



Figure 1: Dependency of ImageTransformations.jl

- JuliaImages/Images.jl#767 – Towards consistent style, part 1: a naming guide
- JuliaImages/Images.jl#772 – Revisiting the Images API
- JuliaImages/Images.jl#790 – Towards consistent style, part 2: a documentation guide
- zygmuntszpak/ImageBinarization.jl#23 – What's the appropriate argument order?
- zygmuntszpak/ImageBinarization.jl#24 – Export limited number of symbols?

discuss the coding styles and programming practice in the most generic way. Packages such as HistogramThresholding.jl and ImageBinarization.jl are examples that validate the effectiveness and usefulness of style consensus reached in those issues. For instance, in ImageBinarization.jl, one could binarize an image using any implemented methods[2] with one unified API:

```
binarize(::BinarizationAlgorithm, img::AbstractArray{<:Colorant,2})
```

[2]At the time of writing, there are 13 methods implemented.

## Project Expectation

With these existing work, it's in the right time to revisit the whole Images.jl ecosystem and head towards a more easy-to-use Images.jl package, and more ambitiously, towards Images.jl v1.0. This project aims to solve this problem by:

1. providing more comprehensive and integrated documentation on both style guide and ecosystem illustration, and
2. drafting RFCs (Request For Comments) and pruning code base of the ecosystem accordingly.

Writing tutorials and demos of Images.jl is not included in this project since it belongs to a totally different project. Trait functions will be tested and documented carefully to support high-level API design.

This is a project on documentation and code refactoring to provide more consistent, robust and extensive APIs to both users and developers, and this is also a sub-project to Images.jl v1.0 milestone. Potentially involved packages are:

- **user entrance:** Images.jl
- **core packages:** ImageCore.jl, ImageAxes.jl and ImageMetadata.jl
- **application packages:** ImageMorphology.jl, ImageTransformations.jl, ImageDistances.jl and ImageFiltering.jl
- **new packages**[3]**:** ImageBinarization.jl, HistogramThresholding.jl, ImageInpainting.jl

Computer-vision packages (e.g., ImageTracking.jl) and plotting packages (e.g., ImageView.jl) are not under consideration of this project.

While doing the documenting and code refactoring work, this project has many side effects:

- potential bug reports and patches to all related Julia repositories;
- introduction of new image processing packages to place legacy methods in Images.jl, for example, ImageEdge.jl;
- introduction of a new image denoising package, ImageNoise.jl, as a concept-

[3]These packages are not yet imported by Images.jl.

validation experimental field.[4]

Note that these side effects are not counted as expectations of this project.

Before introducing the details of delivery schedule of this project, it's worth noting that workload and timeline of this project are hard to estimate due to two reasons:

- this project can last for arbitrarily long time, and can contain an arbitrary number of issues and tasks; refactoring code base can last forever.
- lots of repositories are get involved in this project; time delay in receiving inputs from others is significant.

Hence the purpose of this project isn't to make the ecosystem perfect, instead, it is to evolve an ecosystem that is significantly better in API and naming style, so that future development towards Images.jl v1.0 is possible.

[4]This is the author's research field.

# 2 Delivery Schedule

This project is delivered in two stages: **documenting** and **pruning**. Documenting and recording in the first stage serves as the preparation for the second stage's pruning work of Images.jl ecosystem. The existence of pruning stage is to minimize the impact of large amount of backward incompatible API changes, which means backward compatible PRs can happen at any stage.

Concerning the GSoC timeline, documenting stage begins from April 22[5] to June 24 (weeks 1-10), and the pruning stage starts from July 1 to August 26 (weeks 12-22). Week 11 serves as a buffer week. Phase 1 evaluates the documenting stage, and Phase 2 and Final evaluates the pruning stage.

## 2.1 Documenting

### Stage Expectations

The primary purpose of this stage is to provide traceable records for the next stage's pruning work. There are three types of records generated in this stage: **ecosystem documentation**, **developer manual**, and **RFCs**.

Ecosystem documentation illustrates the scope of image ecosystem and relationships between different relevant packages; it helps users and developers to understand this ecosystem and its fundamental principles quickly. Developer manual consists of the style guide, best practice as well as other related community-operating rules; it gives a documented reference to developers to solve potential conflicts. RFCs with detailed lists of API changes and porting operations are proposed as traceable records for the pruning work in next stage.

### Stage Workflow

This stage is divided into two periods: **discussion period** and **RFC drafting period**. Ideally, this stage ends after the Phase 1 Evaluation with regard to GSoC timeline. However, since a lot of repositories are involved in this project, which makes the timeline hard to be stuck to, the following timeline serves in

[5] Although the coding officially begins from May 27, the author will start this project as soon as he's available.

a flexible way.

The discussion period begins from April 22 to June 9 (weeks 1 to 7). In this period the community shares ideas and thoughts on the future of APIs and on best practices.

In the beginning of the discussion period, an ecosystem documentation be add to juliaimages.github.io as soon as possible to reach a consensus on the future of Images.jl, this consensus shall serve as the fundamental principle to all future discussion and development. Ideally, the current Images.jl maintainer, i.e., Tim Holy, is supposed to participate in.[6]

The RFC drafting period begins from May 27 to June 23 (weeks 6 to 9)[7]. Based on previous discussions, one or more RFCs are drafted and discussed in this period. The last week of this stage is used for evaluation, merging, and announcement of RFCs.

From weeks 1 to 7, many discussions happen simultaneously in the following way:

1. **Code Review:** dig into source codes of repositories of images ecosystem to find anything that's likely in need of changing. Other mature Julia packages, and image-processing libraries in other languages such as scikit-image and MATLAB Image Processing Toolbox are references.
2. **Issue Open:** open an issue for anything worth a discussion, e.g., legacy codes, misplaced codes, codes with bad practice, and undocumented practices and decisions.
3. **Decision Making:** The conventional rules are taken: a decision is made when consensus is reached, otherwise the current maintainer of Images.jl make the decision. If a decision can't be made before June 16 (Week 8), it'll be dropped as future work.
4. **Record:** all approved, rejected, and future-work proposals are documented in a temporary repository - GSoC2019_Document. Developer manual is drafted to juliaimages.github.io when there are enough decisions made.

[6] In case of maintainer being busy on other work, the author will draft a document based on his understanding and post it to the maintainer to get a feedback.

[7] This doesn't mean RFCs can't be drafted before May 27.

From weeks 6 to 9, RFC drafting[8] happen simultaneously in the following way:

1. **Code Review:** for each approved proposal, find all involved code pieces, and give a solution to it according to the developer manual. The principle of code review is to rigorously stick to decisions made in the discussion period – either there's one principle or no principle.
2. **RFC Post:** post the draft-version of RFC in juliaimages.github.io as WIP PR.
3. **RFC Review:** if there's an issue with any item in the proposed RFC, suspend the related items and go back to the discussion workflow until a decision is made.
4. **RFC Approval and Announcement:** After approval of RFCs, they are merged as records and announced to the community via slack and discourse. RFC merging and announcement only happen in the last two weeks in case there are more items to be added.

RFC details on how the code base is pruned are described in section 2.2.

### Stage Evaluation

Four items are evaluated at the end of this stage, i.e., Phase 1 Evaluation:

- 3/10: activity on issues and discussions
- 2/10: ecosystem documentation
- 2/10: developer manual
- 3/10: RFCs

A score of 6/10 stands for Evaluation Pass.

[8] A RFC Template is available in the Tensorflow community, and 20180827-api-names.md is a good API-renaming RFC example.

## 2.2 Pruning Code Base

After the RFCs being approved and announced to the community, the pruning stage begins. Ideally, this stage begins from July 1 to August 26 (weeks 12-22).

### Stage Expectations

The pruning stage is to clean the code base according to the RFC operation guide. There are three types of pruning work:

- symbol renaming, move, and removal – backward incompatible
- API changes – backward incompatible
- API enhancement – backward compatible

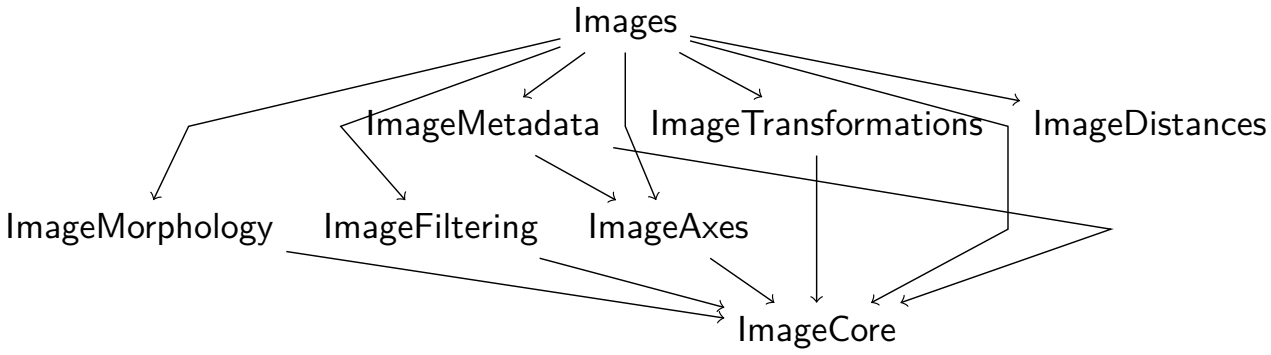
For the ease of tracking pruning progress, a project/milestone is set in Images.jl to track the progress, and each pruning-related PR/issue will be tagged.

### Stage Workflow

Challenges during this stage are backward incompatibility and complex package dependencies. This section focus on strategies to address these problems.

One strategy of the pruning work is to start from packages with the least dependencies to that with the most dependencies. Using terms from section 1, we start from core packages (e.g., ImageCore.jl), to application packages (e.g., ImageTransformations.jl), and finally to the user-entrance package, i.e., Images.jl. New packages are easy to be handled since they're not officially included in Images.jl ecosystem yet.

Figure 2: Dependency Hierarchy of Images.jl Ecosystem




Another strategy is to do all the pruning work in separate branches to

reduce the influence brought by its backward incompatibility. In other words, the workflow of pruning work is:

1. create a separate branch api-prune in each involved repositories, set up CI environment.
2. port all methods and symbols in separate branches – this is backward incompatible
3. merge each branch into master, and tag a minor version to each repository.
4. freeze minor version for one or two months to let downstream packages upgrade their code base. In the meantime, do backward compatible API enhancement
5. remove deprecated symbols, methods and their tests, tag a minor version

Due to limited time of GSoC, steps 1-3 are counted as a part of GSoC project, and steps 4-5 belong to future work. Since lots of repositories are involved in this project, it's highly possible that this project ends in step 2.

Porting methods from package A to package B takes the following routine:

1. implement new methods and unit tests in package B
2. deprecate old methods in package A; these codes will be deleted after at least two minor releases.

### Stage Evaluation

With regard to GSoC timeline, this stage includes the Phase 2 and Phase final evaluations. Two attributes can be uses to evaluate the progress: milestone progress (percentage of merged PRs) and absolute number of opened PRs. Both of these are essential to evaluate the success of this project: the former helps evaluate the completeness of this project and the latter helps evaluate the workload and difficulty of this project. Without the latter, one might try to just finish his project without concerning the greater Images.jl v1.0 progress.

The Phase 2 evaluation shall focus on checking if the pruning work begins as expected. The Phase final evaluation shall focus on checking if the major part of pruning work is done.

# 3 About the Author

My name is Jiuning Chen, a third-year student in the School of Mathematical Sciences, East China Normal University, Shanghai. I'm currently doing research related to image processing, computer vision, convex optimization, and machine learning. It's worth noting that I got my bachelor's degree as a philosophy student, this experience makes me a good candidate to think about the future of Images.jl ecosystem from a more generic and broader view – I have good feelings about what needs improvement and what doesn't.

50+ hours per week can be guaranteed on this project since there's no other internship or vacation plan this summer. After GSoC, at least 20+ hours per week can be guaranteed to continue the long-term Images.jl v1.0 project since I've passed the Ph.D. candidate examination in ECNU – no compulsory work in the first academic year.

I learned Julia in Aug 2018 and become a contributor to Julia community since Oct 2018, and plan to make more contributions as a member of this community. Currently I'm one of the developers in juliaimages.github.io.

## Contribution to the community

My main interest is to improve the Images.jl, in the meantime, there are some small patches to Flux.jl and julia. The major part of my contribution to JuliaImages is on issues: 25% pull requests, 47% issues, 19% commits, 9% code reviews. This is because I don't think Images.jl is at its good status for new features until we pull this GSoC project off.

My previous PRs can help proving that I'm qualified to make progress on this project. Unless explicitly noted, all following PRs are non-trivial and merged.

- Enhancement: JuliaLang/julia#29626[9]

  Support `repeat` at any dimension as an enhancement

  reviewed by Matt Bauman

- Feature: JuliaImages/ImageTransformations.jl#58

  Implement a robust `imrotate` function

[9] I only learned Julia for two months at that moment – a proof of my good understanding on julia

reviewed by Christof Stocker and Tim Holy

- Enhancement: JuliaImages/ImageTransformations.jl#59

  Enhance `imresize` with more delicate dispatch design

  Rewrite all the test cases of `imresize`

  reviewed by Christof Stocker and Tim Holy

- Patch: JuliaImages/ImageDistances.jl#8

  fix a module design bug caused by misunderstanding of method overriding

  reviewed by Júlio Hoffimann

- Patch: JuliaImages/juliaimages.github.io#50

  fix the paralyzed autodeploy problem

  reviewed by Christof Stocker

- Documentation: JuliaImages/TestImages.jl#35

  initialize docstring of `testimage`, which is the only exported function

  reviewed by Tim Holy

- Documentation(RFC, pending): zygmuntszpak/ImageBinarization.jl#25

  redesign the docstring structure of all codes in ImageBinarization.jl

  this PR serves as a first attempt to JuliaImages/Images.jl#790 – Towards consistent style, part 2: a documentation guide

  not yet reviewed because the maintainers are busy lately

- Patch(pending): FluxML/Flux.jl#710

  reimplement `activations` to replace the current broken one

  reviewed by Mike J Innes

- Patch: FluxML/Flux.jl#372

  restrict argument type of `Dense`, initialize test cases for `layers/basic.jl`

  this PR is quite trivial but necessary

  reviewed by Mike J Innes

- Patch: FluxML/Flux.jl#371

  add support for ≈ and fix dispatch bug on ==

  reviewed by Mike J Innes

I've also authored some related issues:

- JuliaImages/Images.jl#790 – Towards consistent style, part 2: a documentation guide

- zygmuntszpak/ImageBinarization.jl#24 – Export limited number of symbols?
- JuliaImages/ImageDistances.jl#7 – Redesign the type hierarchy
- JuliaImages/Images.jl#766 – Use `channelview` as possible as we can?
- JuliaImages/Images.jl#765 – Move `difftype` to ImageCore.jl
- JuliaImages/Images.jl#792 – reduce Images.jl dependency complexity

## Other Experiences

- Independently set up the whole self-hosted research platform from scratch for my supervisor's laboratory

  5000 lines of markdown user&admin documentation is written by myself.
- *De facto* maintainer of the deep learning servers of the School of Mathematical Sciences, and that of a laboratory in Computer Sciences Department;
- Proudly created and maintained the homepage of my supervisor.
- Head teaching assistant of courses: "Deep Learning and Action (Fall 2018)" and "Digital Image Processing (Spring 2019)".

## Self Evaluation

- **Mathematics & Image Processing (8/10)**: my current research is on image denoising using variational model and deep learning;
- **Linux (7/10)**: heavy usage of docker, bash, git and vim in my daily work to maintain the servers;
- **Matlab (8/10)**: the only programming language used throughout my early-stage of research;
- **Julia (7/10)**: fully understand and stick to the philosophy of Julia, but lack of real and large project experience;
- **Related packages (6/10)**: familiar with other image-processing packages but haven't dig into the source code of them yet.

## Education Background

**2016-Present (Postgraduate)** Study on image processing and computer vision in School of Mathematical Sciences, East China Normal University, and supervised by Prof. Fang Li.

**2013-2016 (Undergraduate)** Bachelor of philosophy, Department of Philosophy, Shanghai University.

**2011-2013 (Undergraduate)** Study on metal material in School of Material Sciences, Shanghai University.