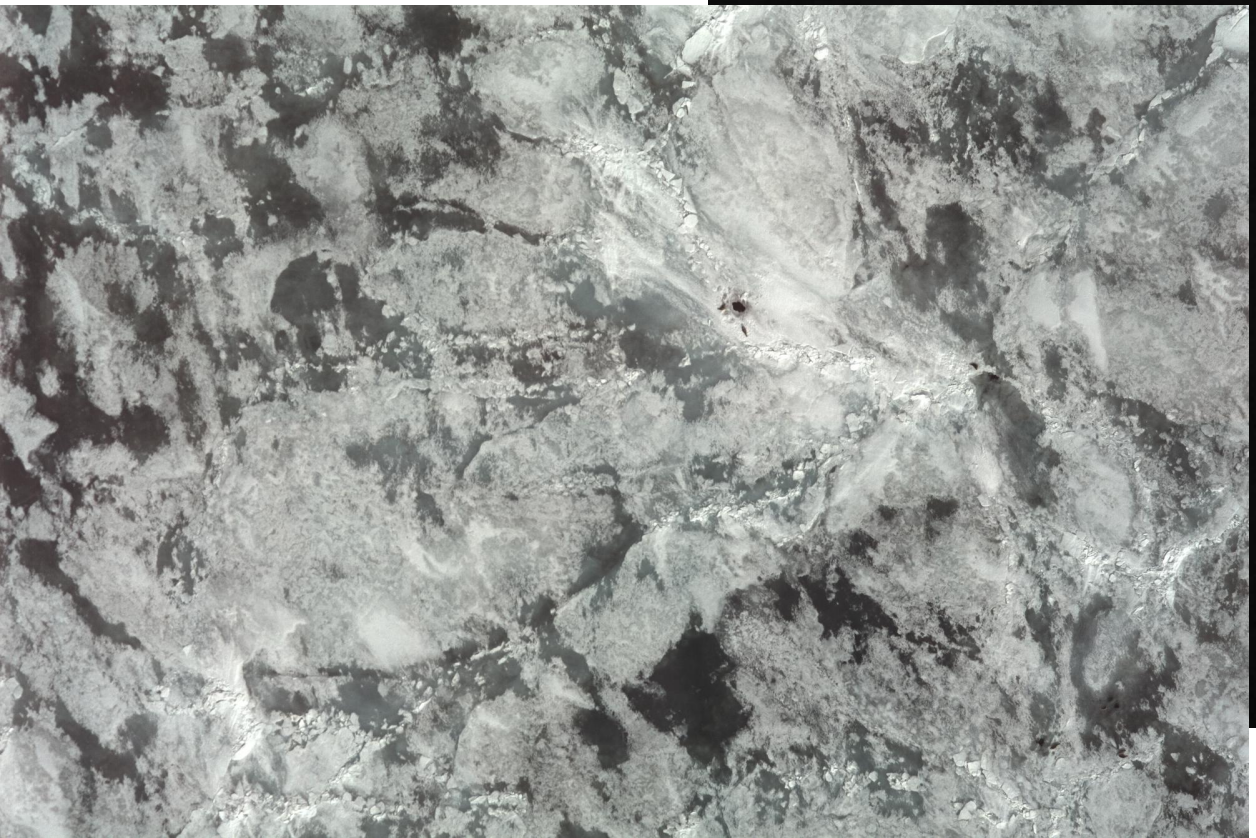
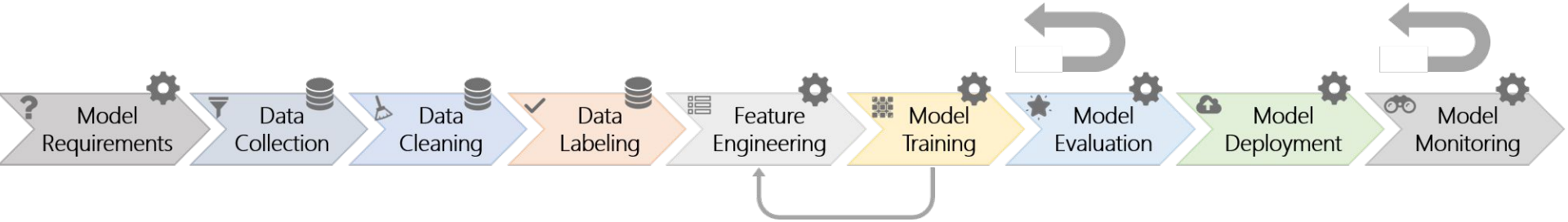
The machine learning development process in remote sensing applications

Yuval Boss

About Me

- Before this I was a software dev in the MacCoss Lab at UW and worked there on and off from 2013-2018. Worked in the lab developing Skyline, a software to aid researchers in analyzing mass spectrometer data.
- Went to school here at the UW, studied Computer Science.
- Helped develop models for the Polar Ecosystem Program while I was in school, joined full time July of this year after graduating.
- Outside of work my main passion is climbing, also enjoy skiing and most things that are outdoors. Picture from the Bugaboos back when we could go to Canada :)





“discovering, managing, and versioning the data needed for machine learning applications is much more complex and difficult than other types of software engineering”

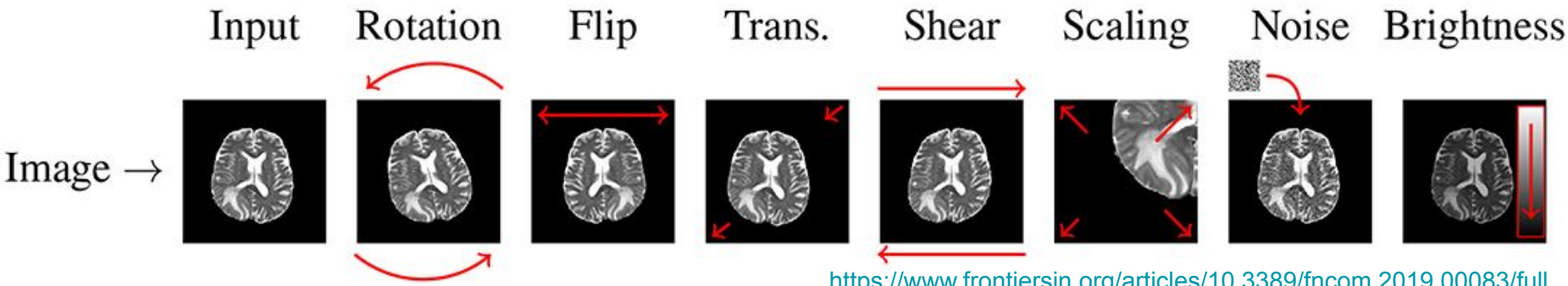
Amershi, S., Begel, A., Bird, C., Deline, R., Gall, H., Kamar, E., . . . Zimmermann, T. (2019). Software Engineering for Machine Learning: A Case Study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. doi:10.1109/icse-seip.2019.00042

Terminology

Terminology

- **Script:** A program, could be in R, python, or any other scripting language.
- **Pipeline/Data Pipeline:** Programs linked together in a linear structure to perform a larger task. Output of each program is the input to the next.
- **Machine Learning Models:** An algorithm which learns a task, sometimes the learned part is called the model. ML model's usually have two parts, one is a part defining the structure of the model, the other is a representation of what the model learned.
- **Bounding Box:** A set of 4 coordinates often accompanied by other information to describe what is seen in an image or another 2D data source.
- **Label:** More general term to describe contents of data. A label could be a point, a bounding box, a marked region in an audio clip, or any other format to describe data content.

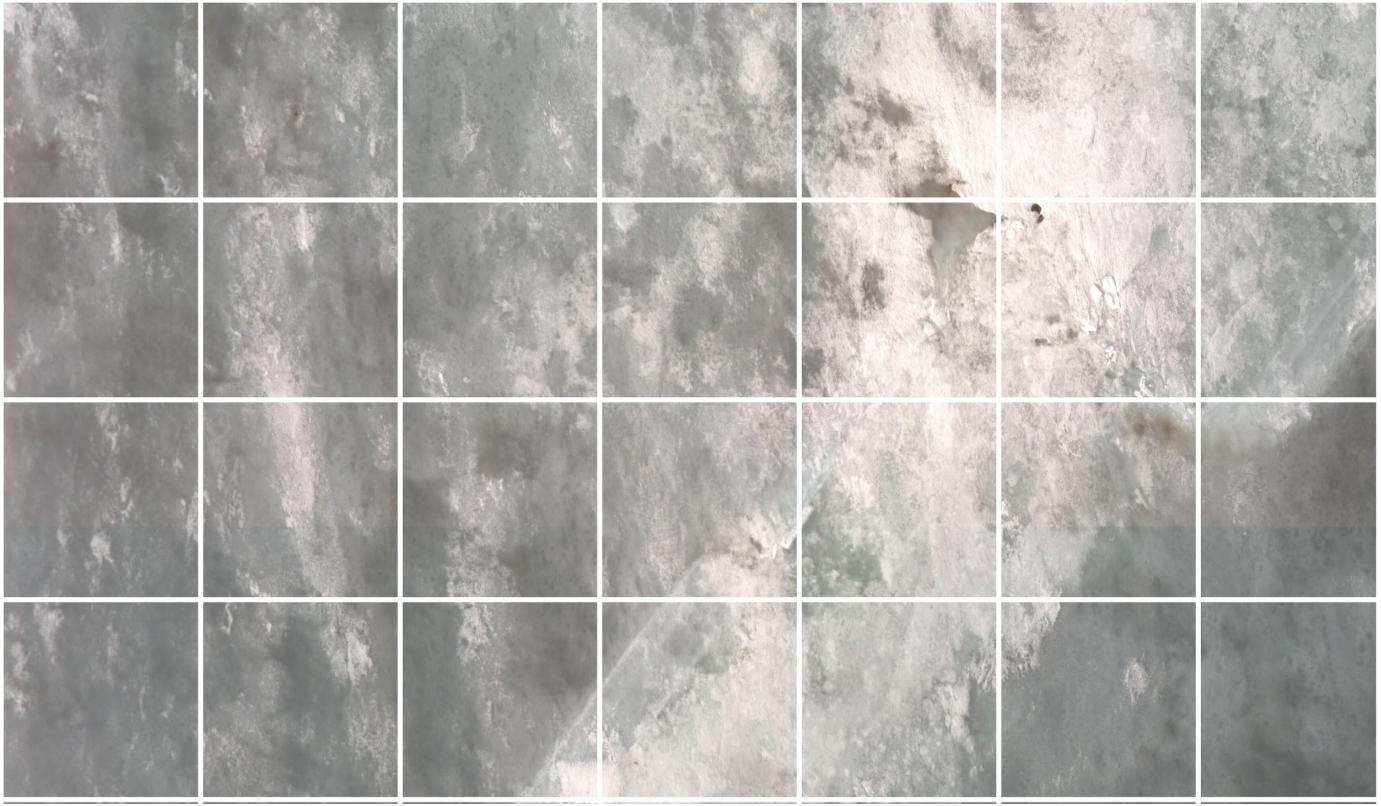
Data Augmentation



https://www.frontiersin.org/articles/10.3389/fncom.2019.00083/full

- Increase data by adding modified copies of existing data
- Increase data by creating synthetic data from existing data
- Acts as regularizer, reduces overfitting

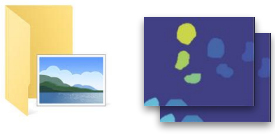
Image Chipping/Image Tiling

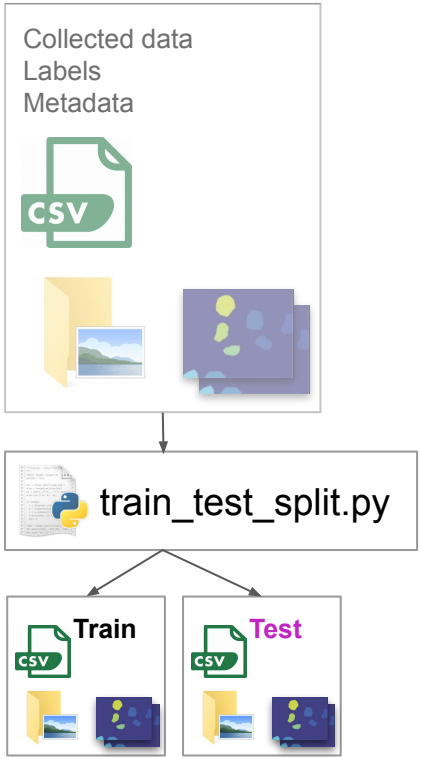


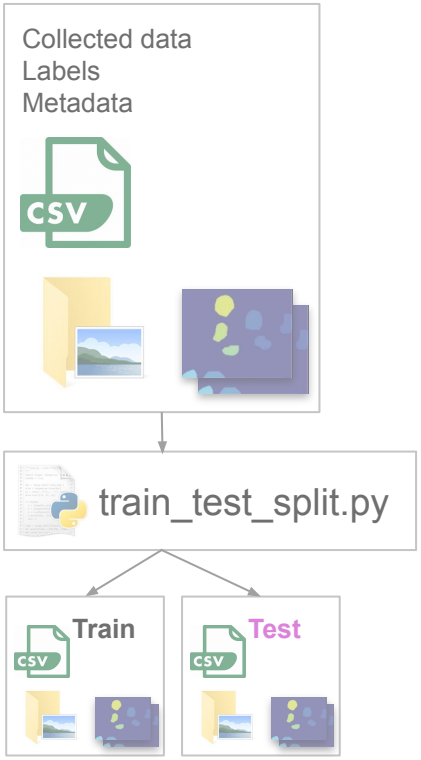
Remote sensing data

- Lots and lots and lots of data
- Sparse - often lots of useless background background
- Noisy - could be environmental factors such as clouds in images, or boat propellor noise in audio. Also technical issues with the system acquiring data.

Collected data
Labels
Metadata




Collected data
Labels
Metadata
CSV
train_test_split.py
Train
CSV
Test
CSV



ML Frameworks

ONNX

Amazon AWS

PyTorch

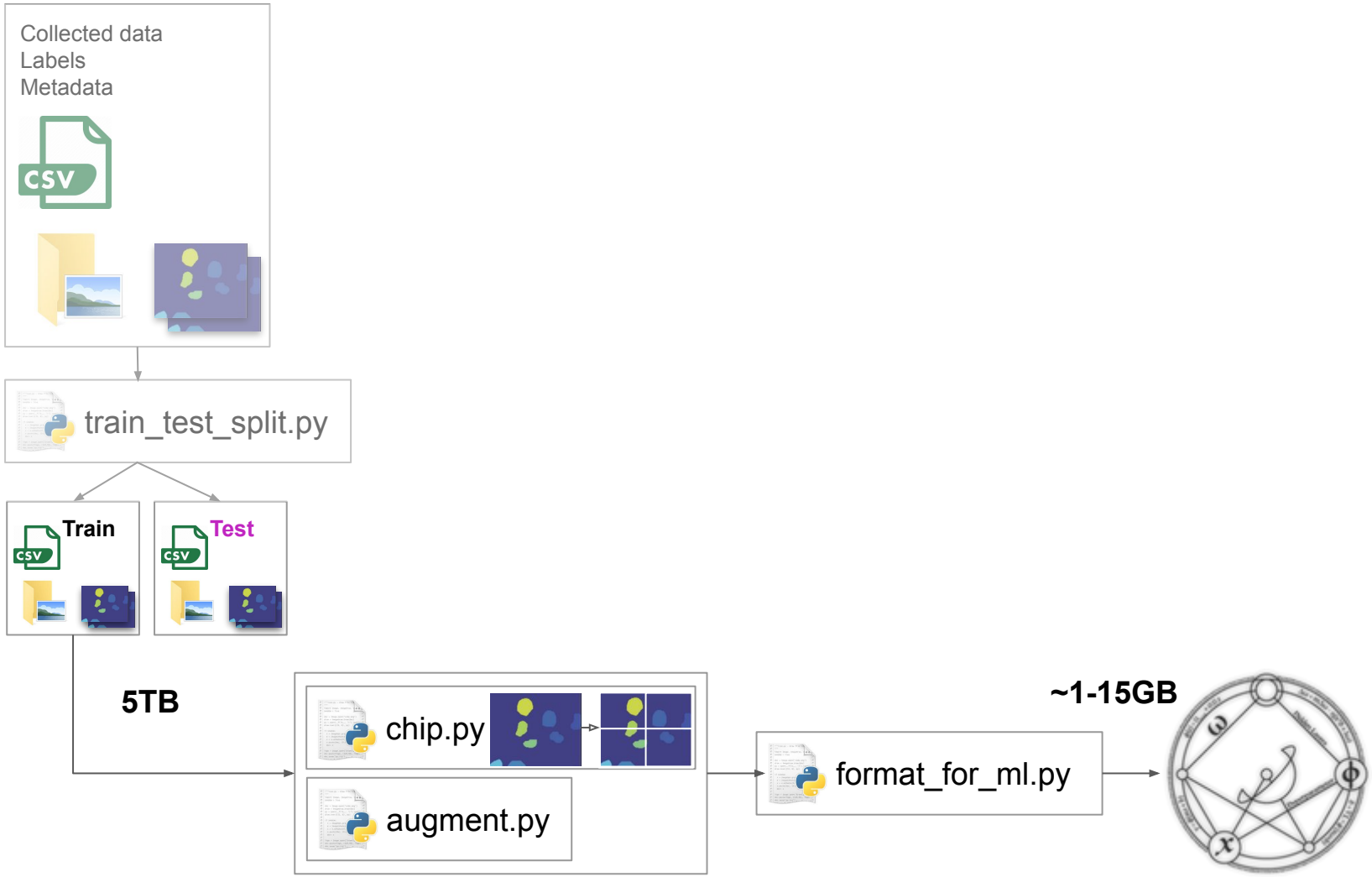
TensorFlow

VIAME

Keras

Caffe2

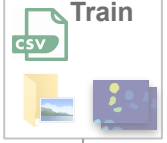
mxnet


Collected data
Labels
Metadata
CSV
train_test_split.py
Train
CSV
Test
CSV
5TB
chip.py
augment.py
format_for_ml.py
~1-15GB

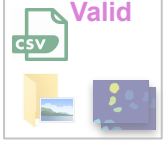
Collected data
Labels
Metadata

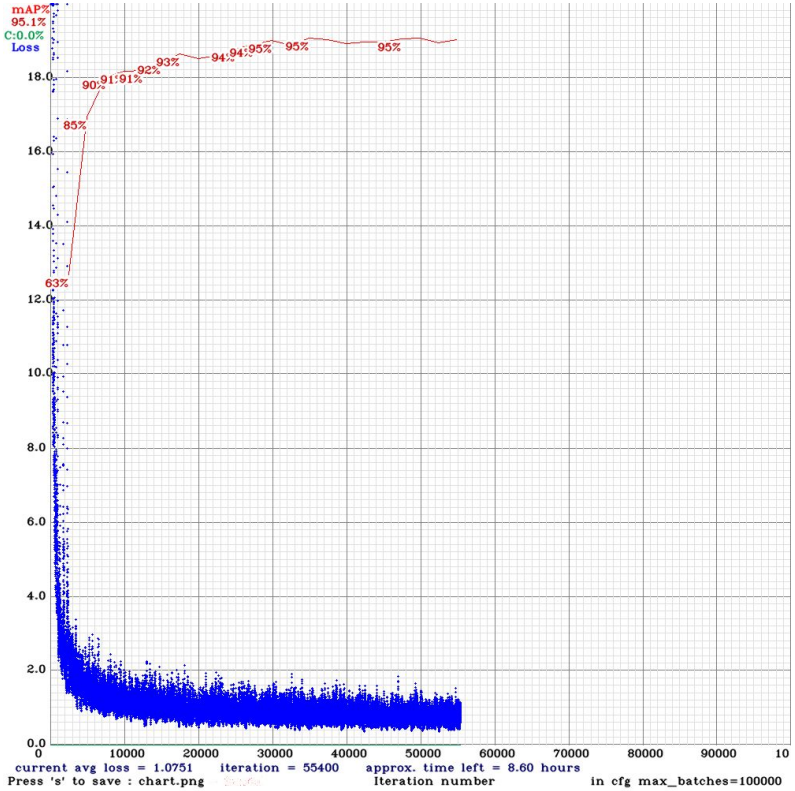
train_test_split.py

Train



Valid





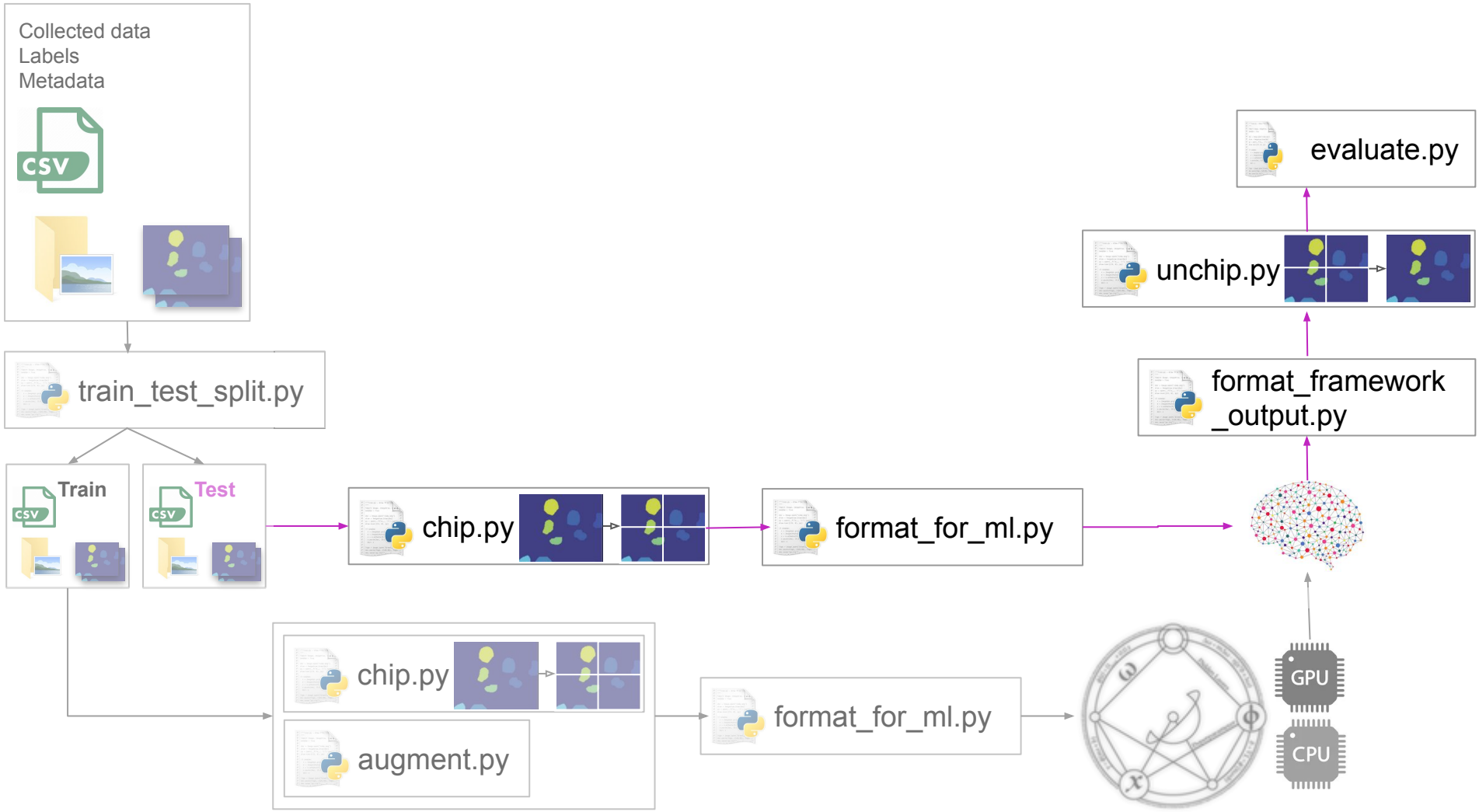
chip.py

augment.py

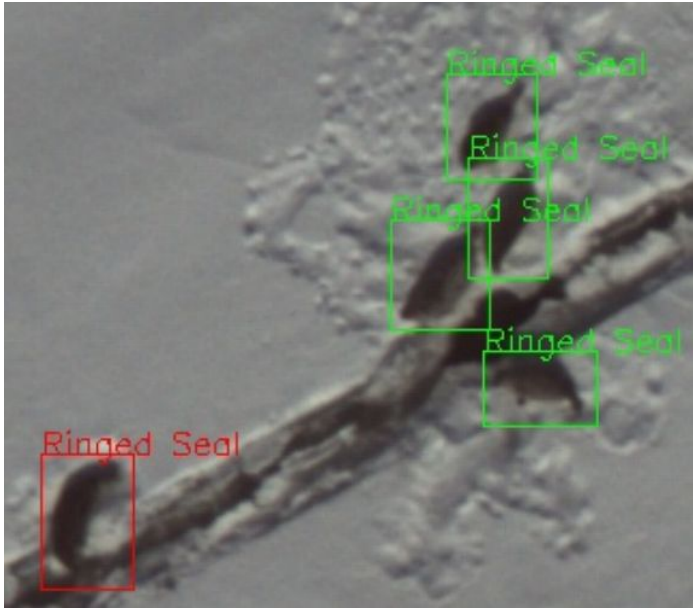
format_for_ml.py

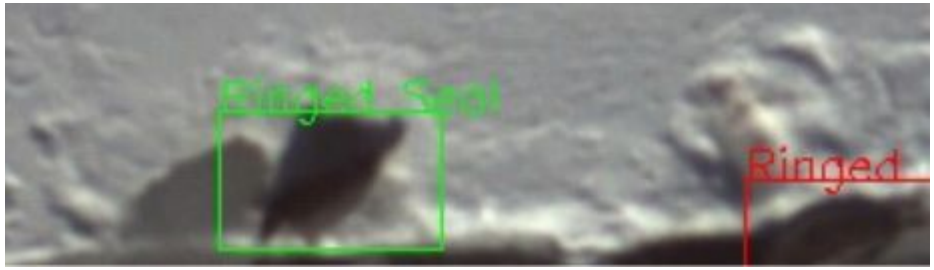
GPU

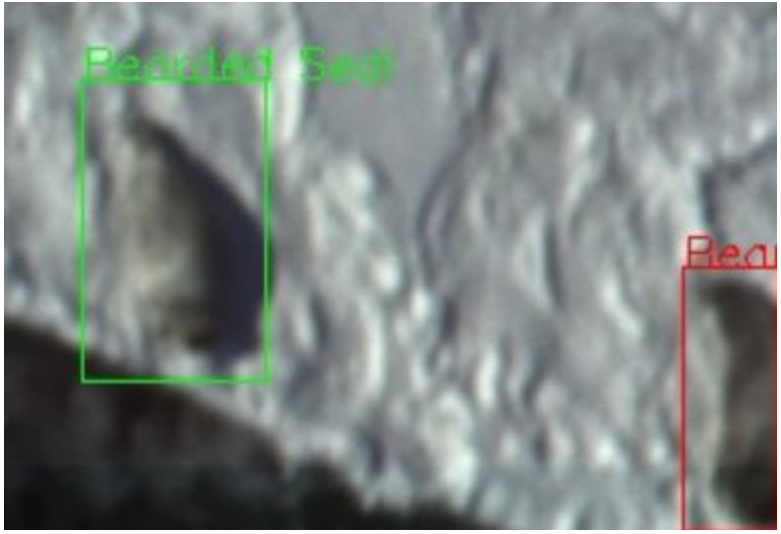
CPU


Collected data
Labels
Metadata
CSV
train_test_split.py
Train
CSV
Test
CSV
chip.py
format_for_ml.py
evaluate.py
unchip.py
format_framework
_output.py
chip.py
augment.py
format_for_ml.py
GPU
CPU

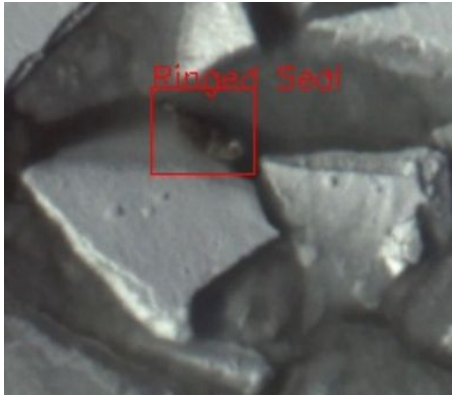

Detector from an
early attempt







False Positive
True Positive
False Negative



Common causes of poor performance in initial exploration work

1. Bad data
    a. missing labels, incorrect labels
    b. poor data quality(ex. Blurry images)
    c. Preprocessing for ml is wrong, for many types of data normalization appropriately is important.
    d. If partitioning(chipping for images) partitions on either test or train end may be an issue.
    e. Class imbalance (not exactly bad data, but needs to either be addressed in the training data or in the model)
2. Not enough data for the given approach
3. Poor augmentation
    a. Not enough augmentation
    b. Too much augmentation
    c. Not the correct type of augmentation
4. Bugs in pre/post-processing sometimes don't cause error messages, can be hard to catch
5. General consensus among those working on real applications with ML is that **data is the #1 thing.** With good data even suboptimal ML approaches can work surprisingly well.

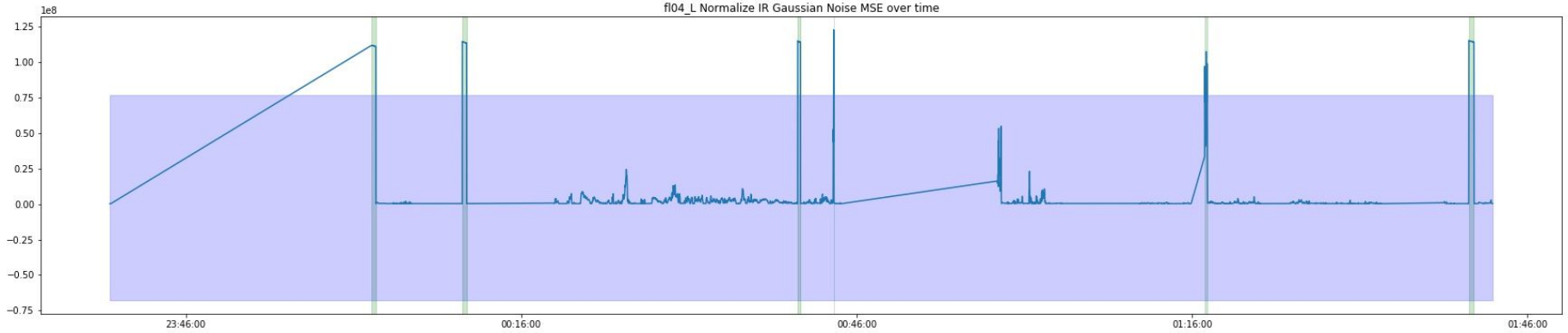
Common improvements on the model side

- Tune learning rate if loss does not converge well
- Transfer learning - I use pre-trained weights when available, usually ImageNet. Surprising how much model trained on bicycles, cups, giraffes and many more things helps for our aerial imagery.
- Possible architecture is not great for the problem
- Usually changes on the model side are for ‘fine tuning’ or during the experimentation phase.

Lots of other common issues/solutions, often the hardest part is formalizing the issue as if you are able to google it quite likely to find discussions around it.

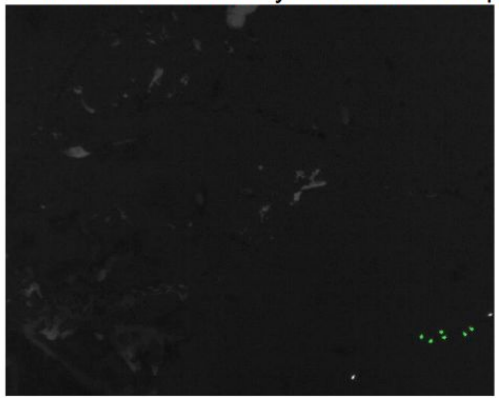
Sometimes don’t even need ML

Non Uniformity Correction Detection

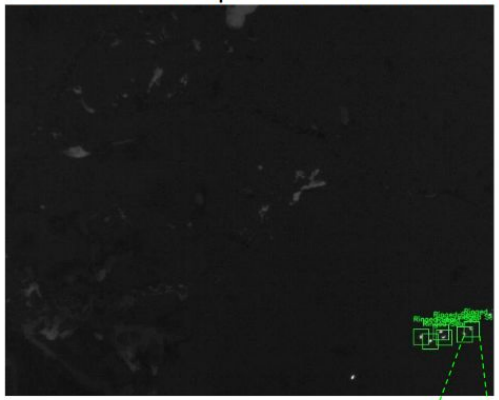


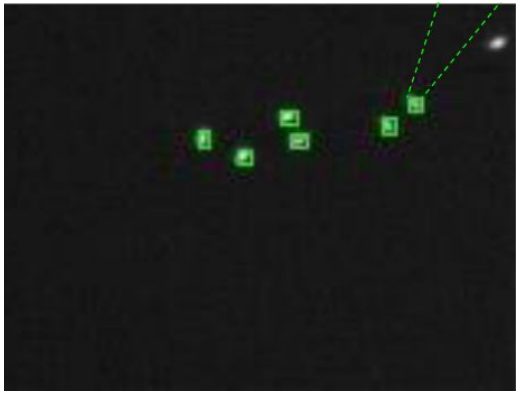
Point2Box

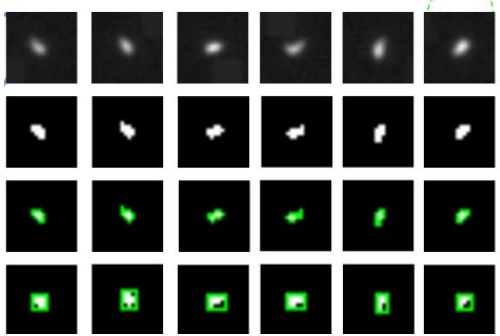
Points were manually labeled on hotspots



Generate 20x20px boxes at location of points





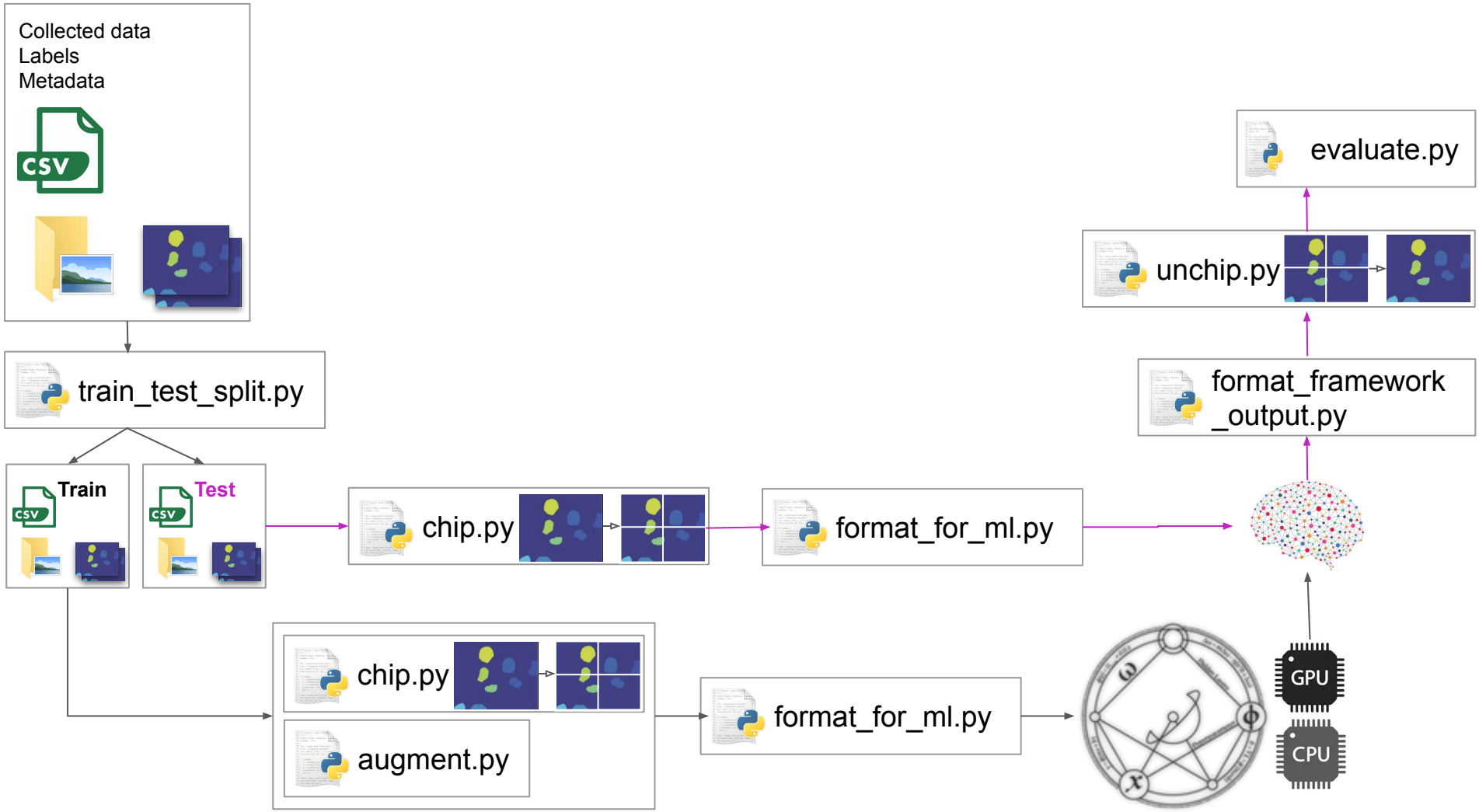


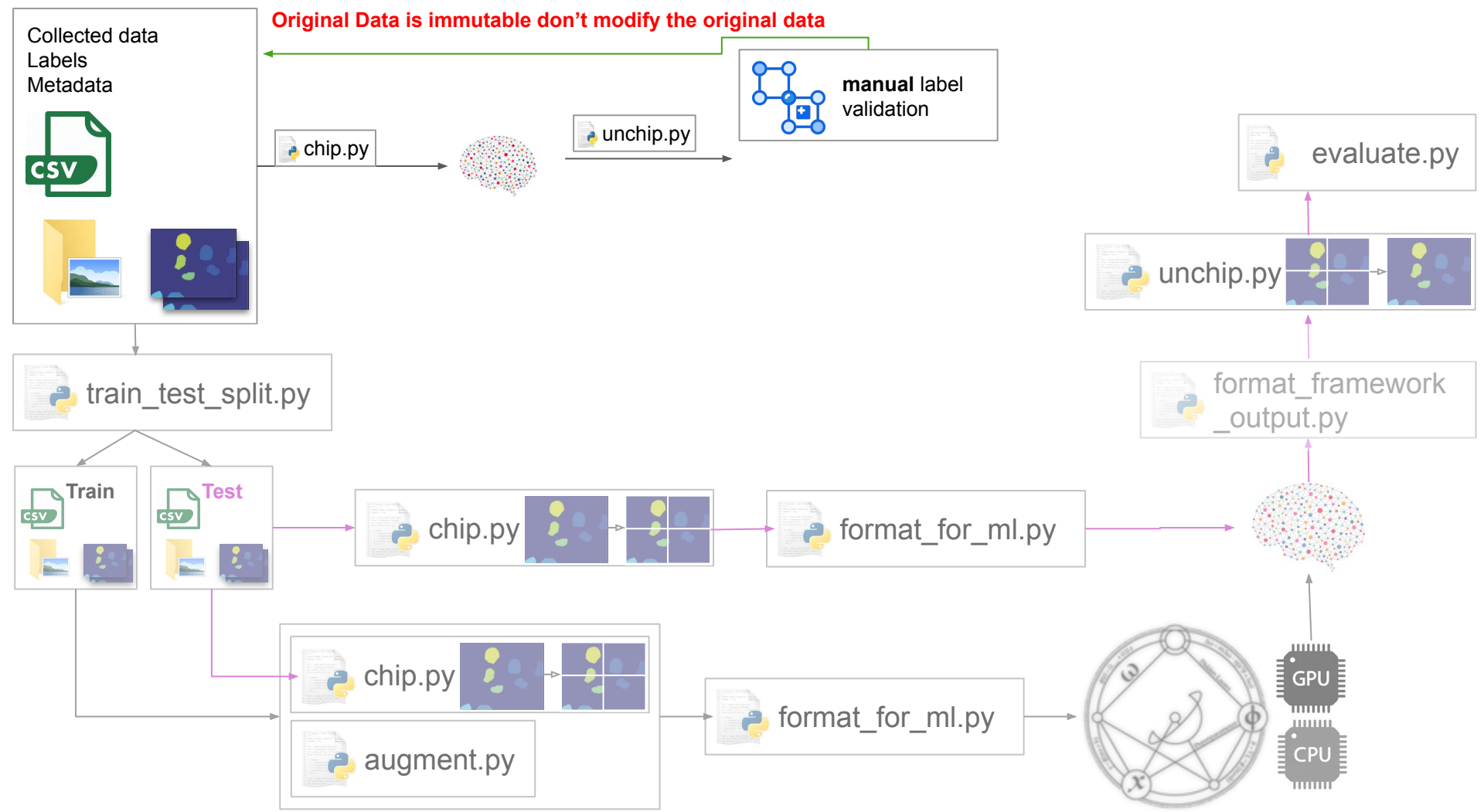
Extract Crop

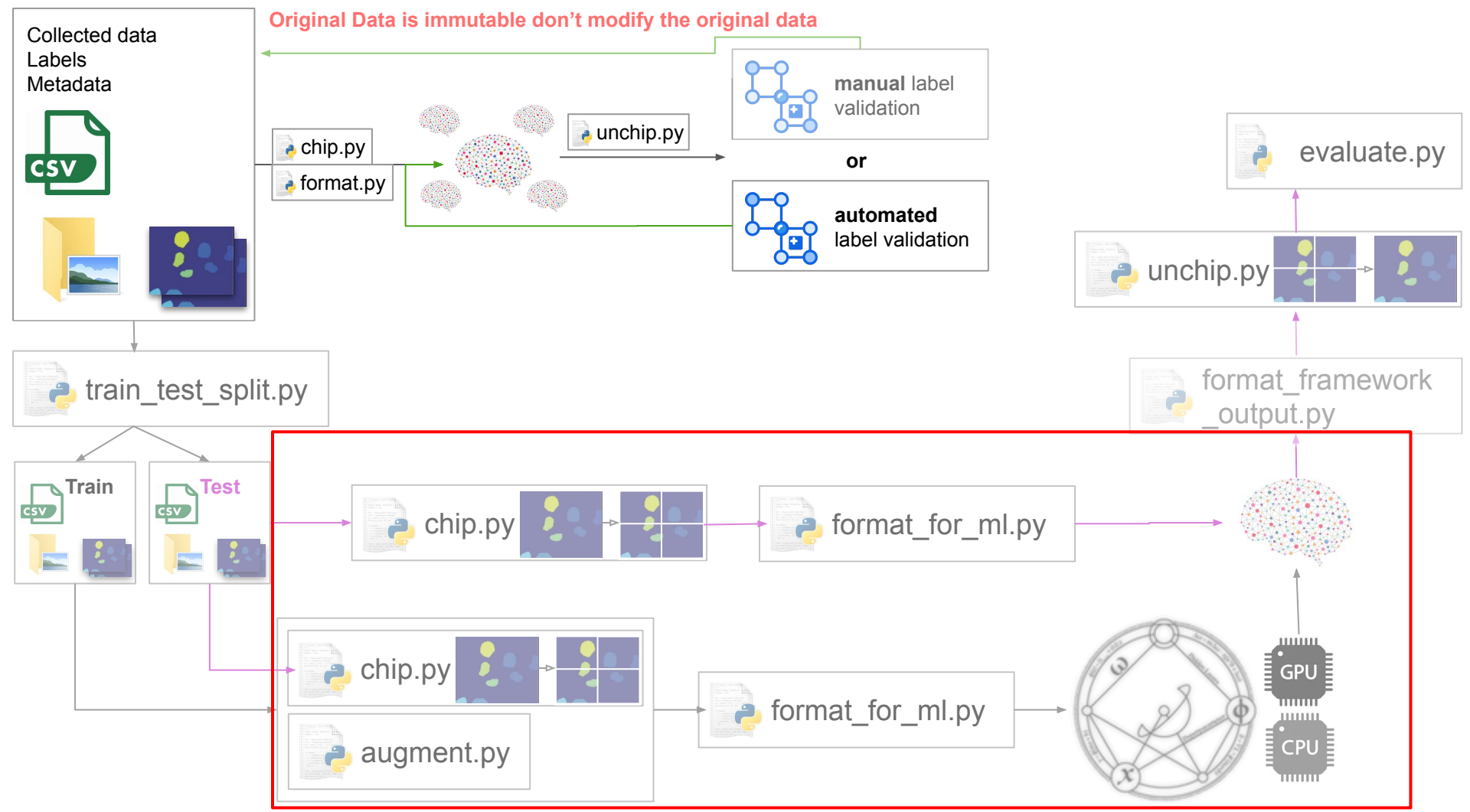
Thresholding (Percentile)

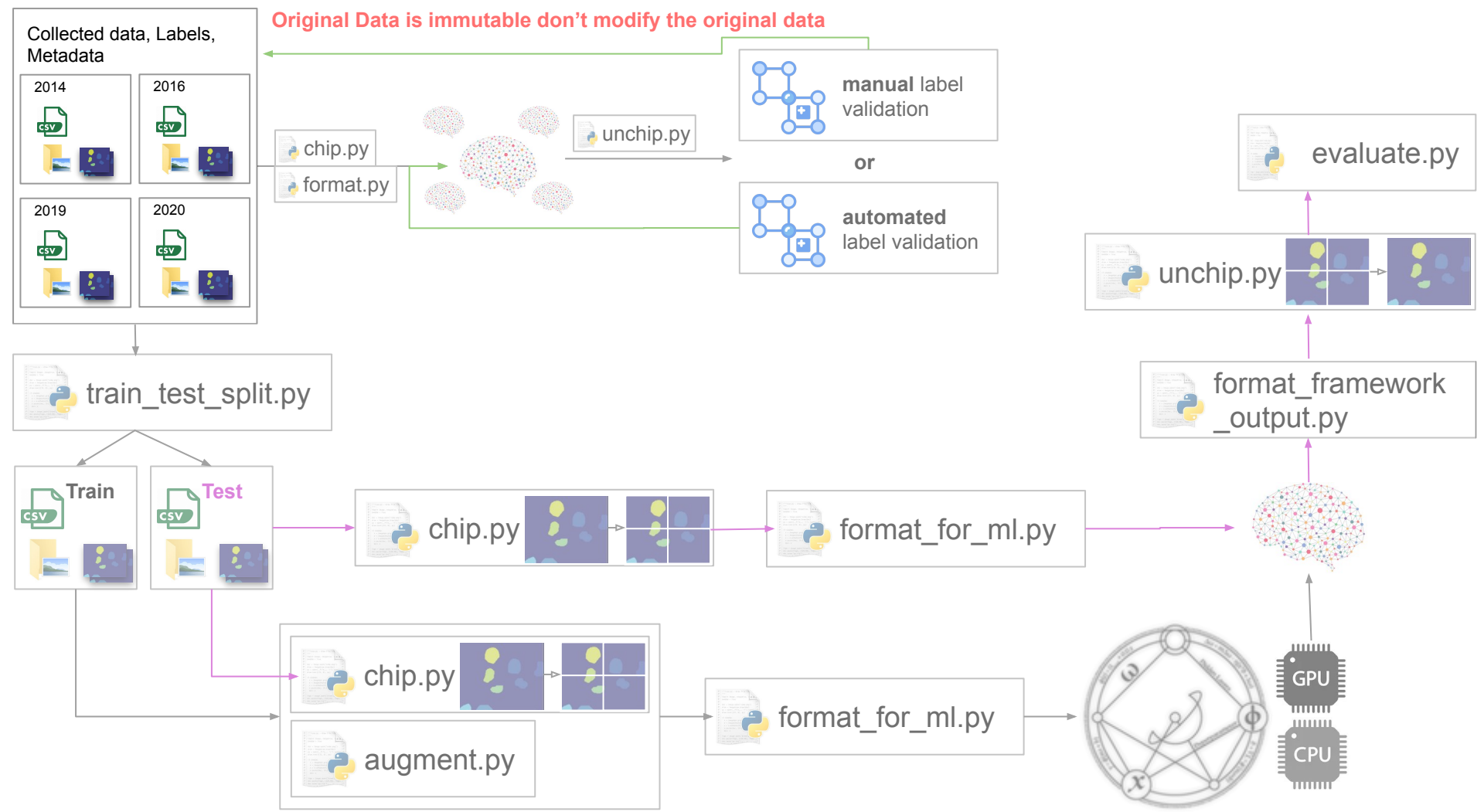
Find Borders/Contours

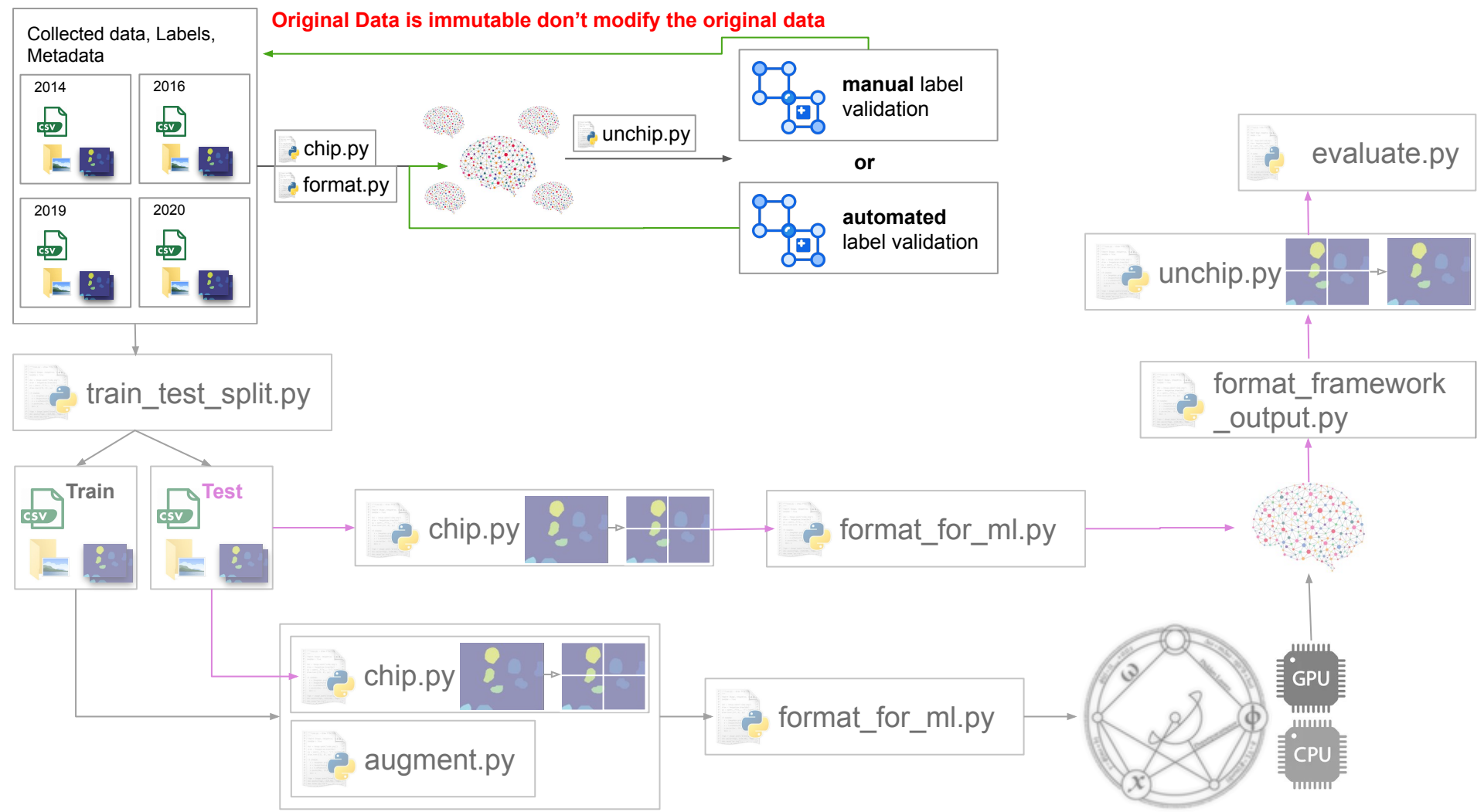
Extrapolate Box

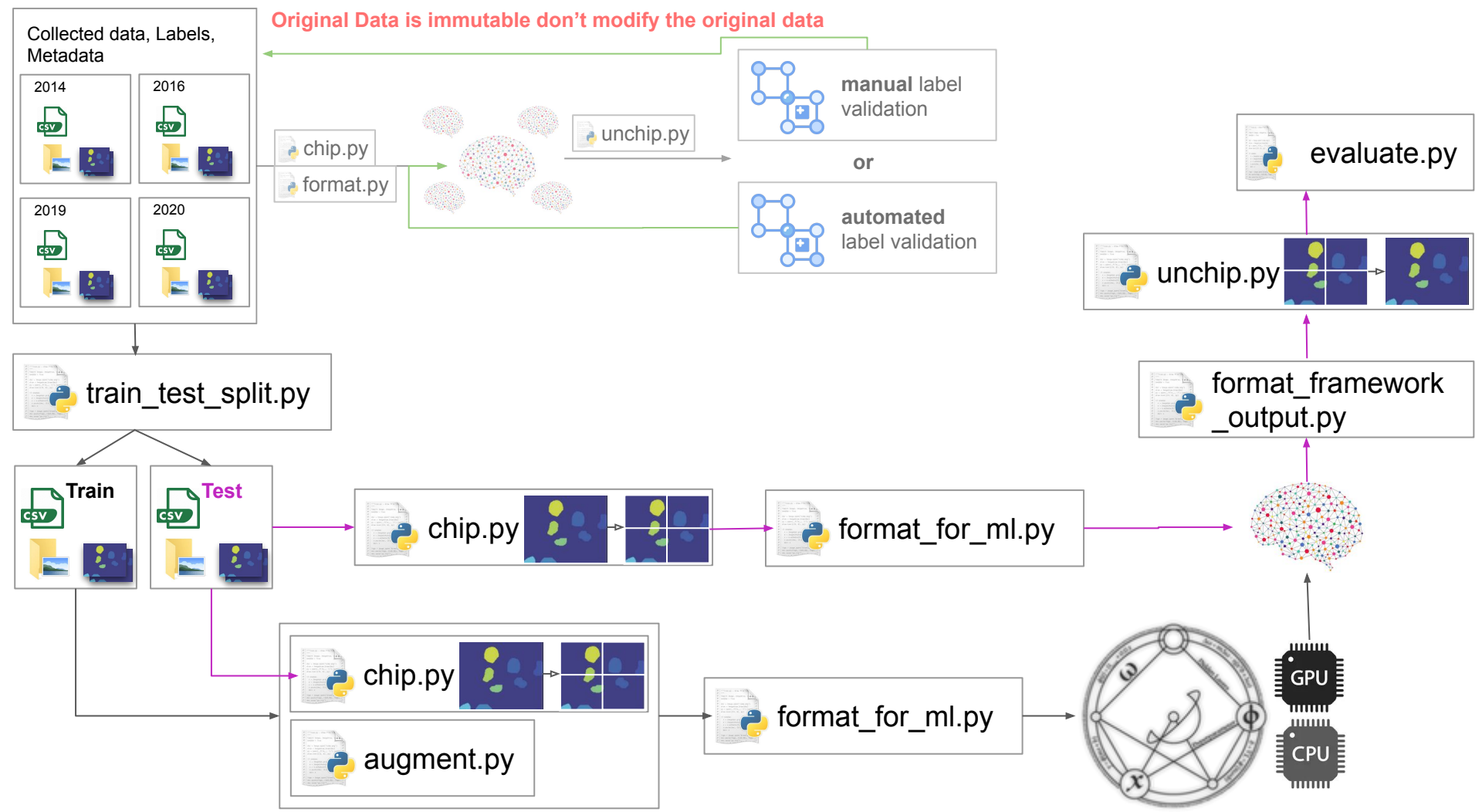

Collected data
Labels
Metadata
CSV
train_test_split.py
Train
CSV
Test
CSV
chip.py
format_for_ml.py
chip.py
augment.py
format_for_ml.py
GPU
CPU
unchip.py
evaluate.py
format_framework
_output.py


Original Data is immutable don't modify the original data
Collected data
Labels
Metadata
CSV
chip.py
unchip.py
manual label validation
evaluate.py
unchip.py
format_framework_output.py
train_test_split.py
Train
Test
chip.py
format_for_ml.py
chip.py
augment.py
format_for_ml.py
GPU
CPU

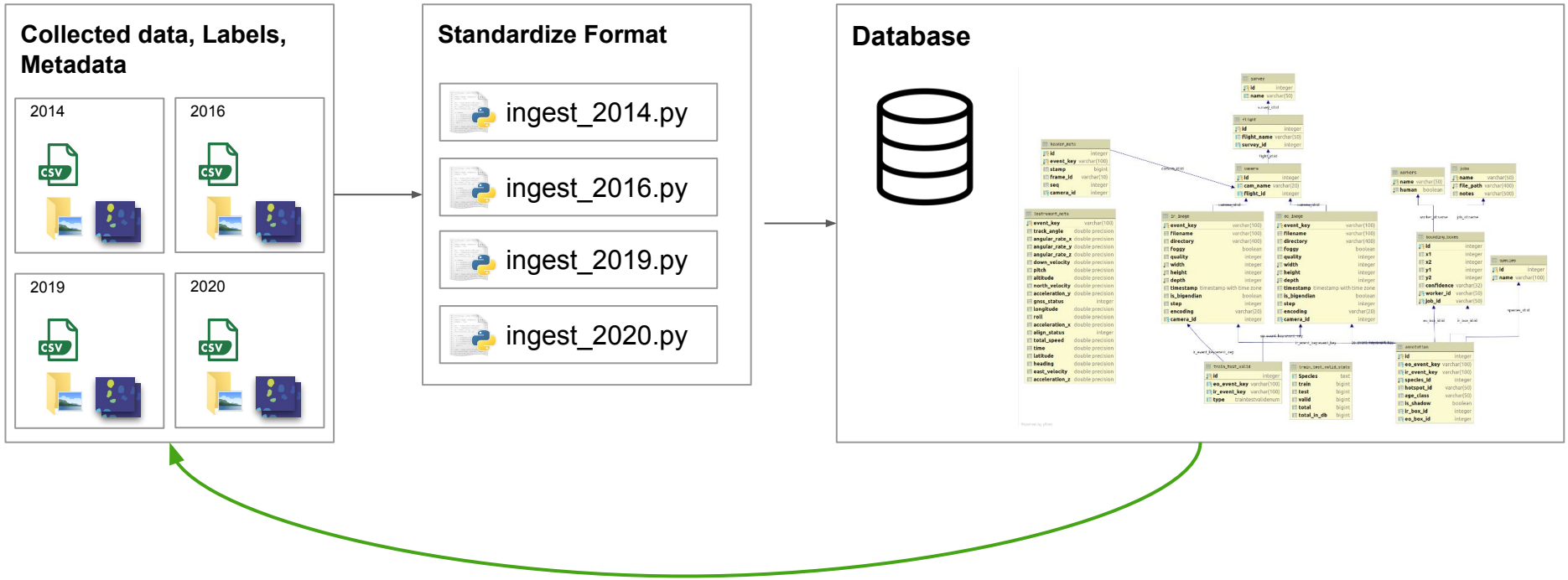

Collected data
Labels
Metadata
CSV
Original Data is immutable don't modify the original data
chip.py
format.py
unchip.py
manual label validation
or
automated label validation
evaluate.py
unchip.py
format_framework_output.py
train_test_split.py
Train
Test
chip.py
format_for_ml.py
chip.py
augment.py
format_for_ml.py
GPU
CPU


Collected data, Labels, Metadata
2014
2016
2019
2020
Original Data is immutable don't modify the original data
chip.py
format.py
unchip.py
manual label validation
or
automated label validation
evaluate.py
unchip.py
format_framework_output.py
train_test_split.py
Train
Test
chip.py
format_for_ml.py
chip.py
augment.py
format_for_ml.py
GPU
CPU


Original Data is immutable don't modify the original data
Collected data, Labels, Metadata
2014
2016
2019
2020
chip.py
format.py
unchip.py
manual label validation
or
automated label validation
evaluate.py
unchip.py
format_framework_output.py
train_test_split.py
Train
Test
chip.py
format_for_ml.py
chip.py
augment.py
format_for_ml.py
GPU
CPU


Collected data, Labels, Metadata
2014
2016
2019
2020
Original Data is immutable don't modify the original data
chip.py
format.py
unchip.py
manual label validation
or
automated label validation
evaluate.py
unchip.py
format_framework_output.py
train_test_split.py
Train
Test
chip.py
format_for_ml.py
chip.py
augment.py
format_for_ml.py
GPU
CPU

Standardizing data access is key



If possible never modify or delete original data. Version control for big data is hard but there is a lot of attention by industry on this and open source solutions exist. Challenge is storage requirement is multiplied when versioning.

Benefits of using a SQL/NoSQL database

- Constraints!
  - Provides certainty that data is what you define your expectation to be
  - When constraints are broken errors are thrown which has helped me a lot to mitigate bugs that could lead to major issues down the line.
  - Example could be any bounding box in the database must have x1<x2 & y1<y2, seems like common sense but when bringing together different datasets that use different formats sometimes mistakes are made, constraints help mitigate that.
- Collaboration
  - If things change, things are changed for everyone working with the data.
  - Improved access, can allow users access with different privilege (reader, admin, etc..)
  - More complex schemas can be used for versioning of information
- Modularity/Scalability
  - Downstream compatibility, all code only needs to rely on one data specification
  - Improved productivity - easier to build and experiment with

Aggregated Database

Now we don't have to worry about formats or modifying original data and can be free to work on the data.

chip.py

format.py

unchip.py

manual label validation

or

automated label validation

evaluate.py

unchip.py

format_framework_output.py

Data has standard format, is clean, and trusted from ML perspective.

train_test_split.py

Train

Test

chip.py

format_for_ml.py

chip.py

augment.py

format_for_ml.py

GPU

CPU

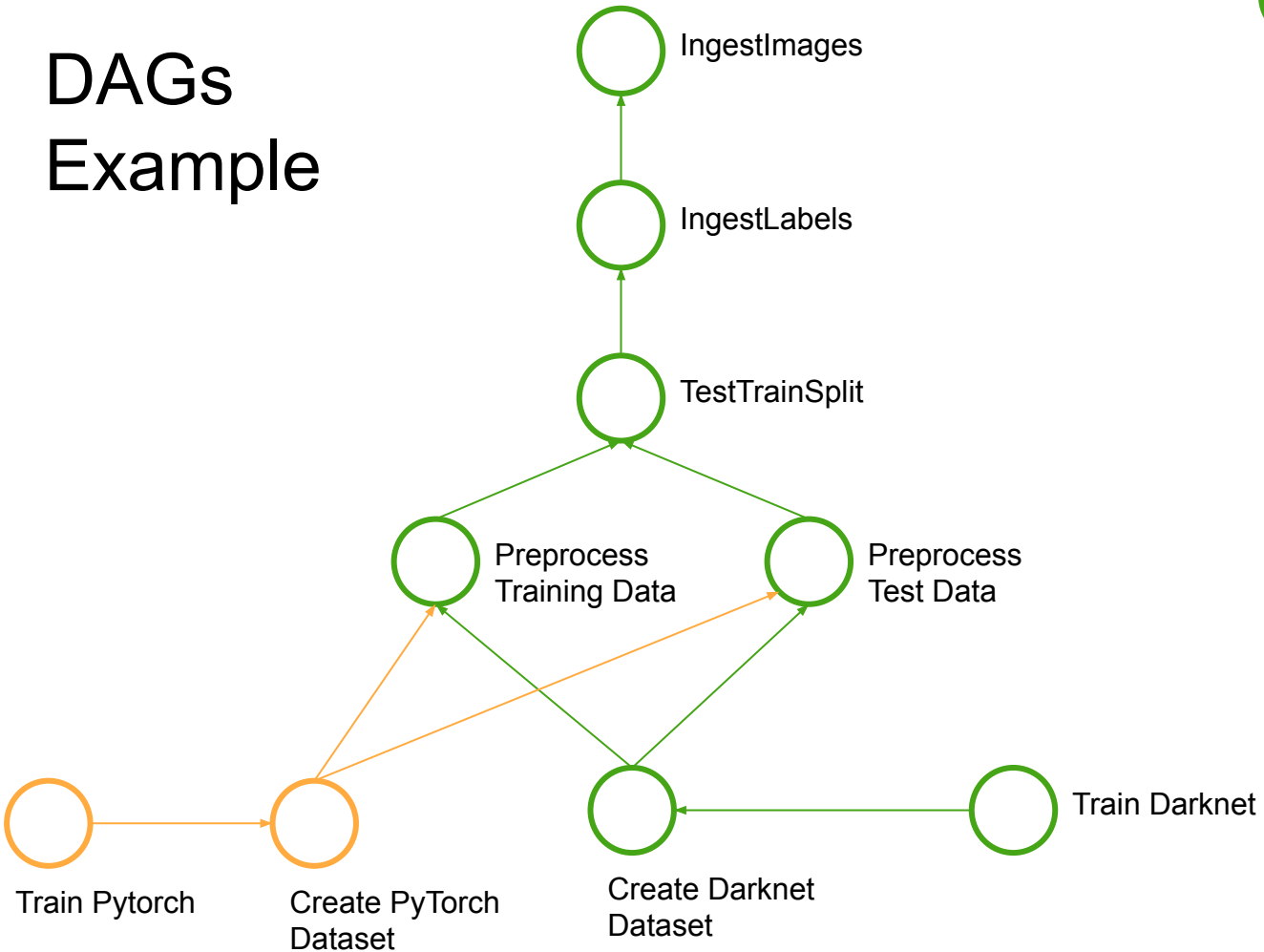
Pipeline development lessons

- Scripts and notebooks are good for experimentation and presentation, not for development. Pipelines should be treated as software.
- Learning is continuous, pipeline development also continuous
- Having to re-run processing code regularly can be a huge hit to productivity, reusing processed data/artifacts saves a lot of time
- Reproducibility is hard
- Concurrency is time consuming and difficult to do well
- Test test test
- Pipelines are just DAGs (Directed Acyclic Graph)

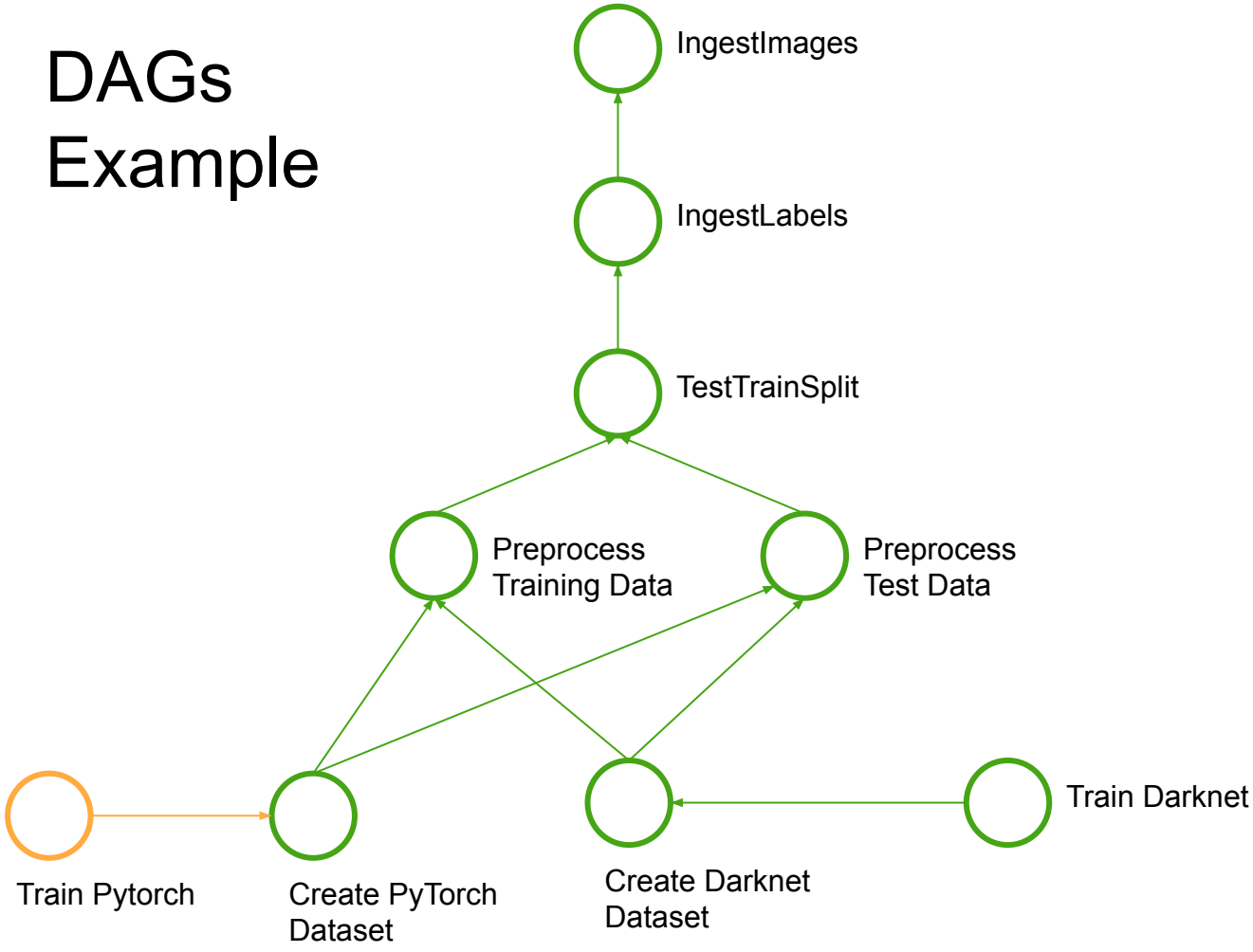
Obvious solution as usual is to find an open-source pipeline frameworks on github!

DAGs Example

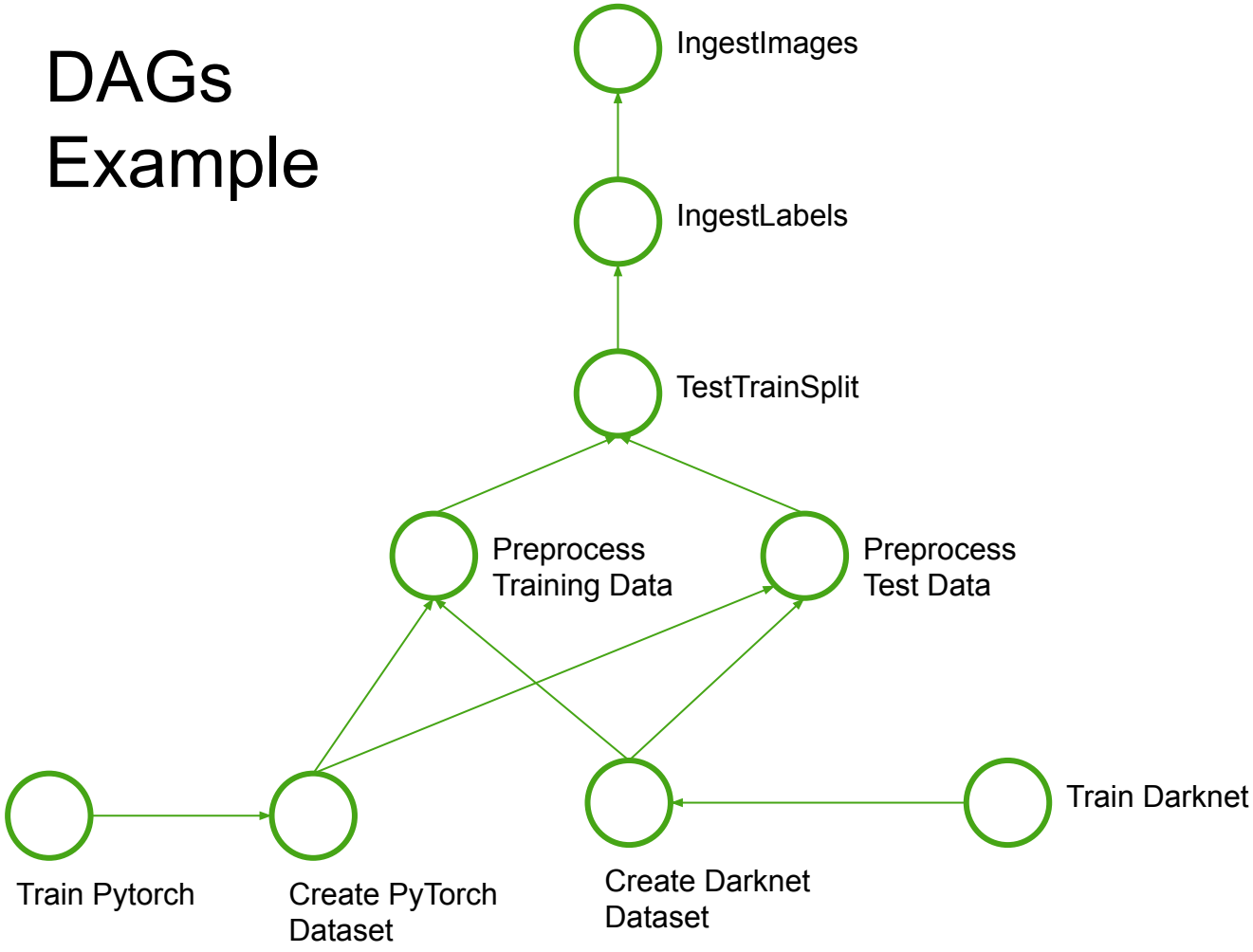
"TaskB dependent on TaskA"



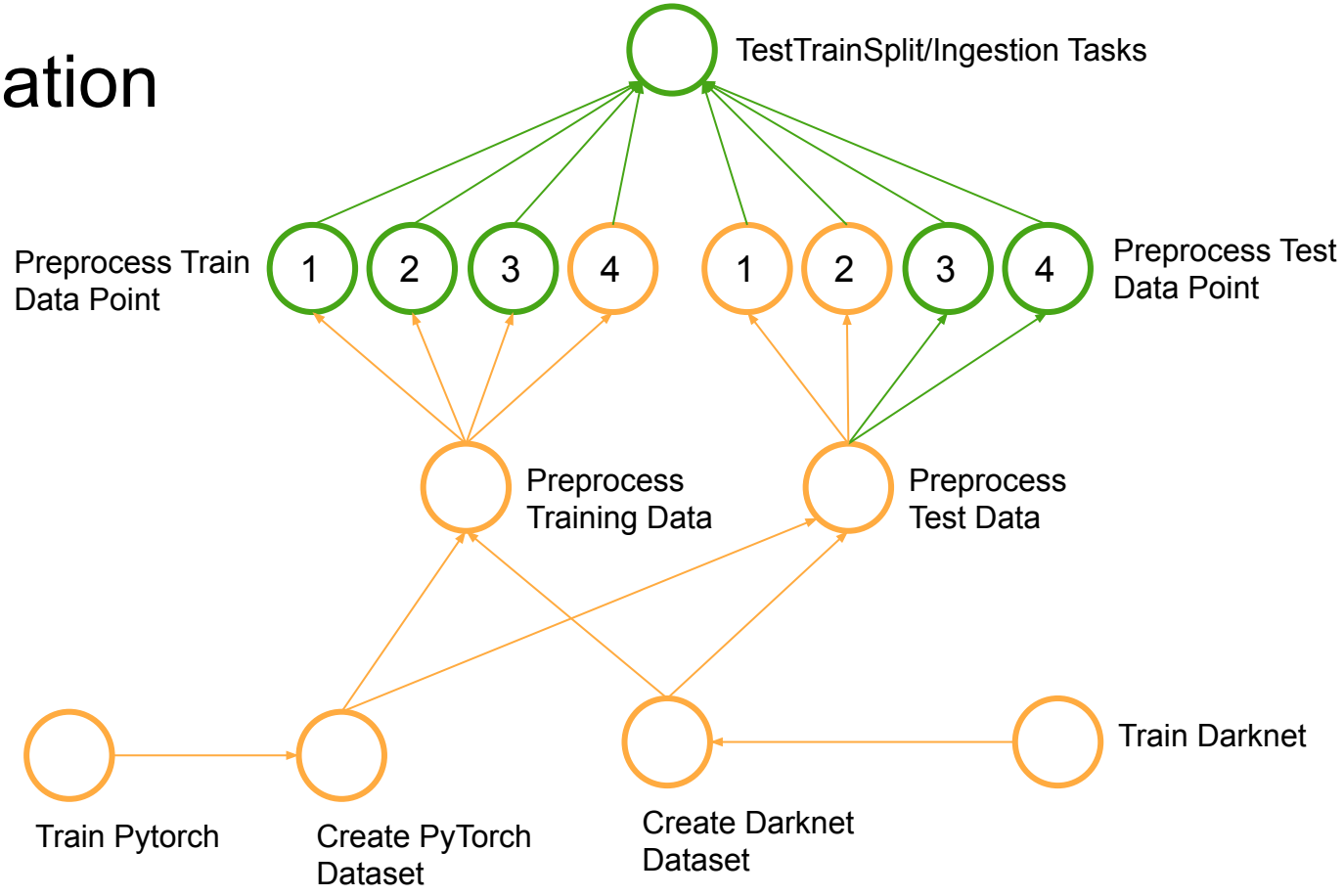
DAGs Example



DAGs Example



DAGs Parallelization Example



Pipeline Reproducibility

```
[trainSetConfig]
only_manual_reviewed = True
background_ratio = 1
bbox_padding = 1
species_filter = ["Ringed Seal", "Bearded Seal"]
remove_chip_if_species_present = ["UNK Seal"]

[testSetConfig]
only_manual_reviewed = True
background_ratio = 1
bbox_padding = 0
species_filter = ["Ringed Seal", "Bearded Seal"]

[validSetConfig]
only_manual_reviewed = True
background_ratio = 1
bbox_padding = 1
species_filter = ["Ringed Seal", "Bearded Seal"]
remove_chip_if_species_present = ["UNK Seal"]

[chipConfig]
chip_w= 512
chip_h = 512
chip_stride_x = 400
chip_stride_y = 400
label_overlap_threshold = .6
```

Pipelines as DAGs frameworks

- Spotify's Luigi
- Apache Airflow
- Netflix's Metaflow
- PipelineX/Kedro
- And many more… github.com/pditommaso/awesome-pipeline

In Conclusion

- Biggest improvements to results are made when focusing on the data.
- Biggest challenges in working with the data often are the software and data engineering challenges.
- Machine Learning is continuous.
- The step of actually training an ml model can be a lot more straightforward than people think.

Frameworks/Tools mentioned

- https://github.com/spotify/luigi
- https://github.com/sqlalchemy/sqlalchemy
- https://github.com/AlexeyAB/darknet
- https://github.com/VIAME/VIAME

Other Useful Links

- Jenny Bryan's How to Name Files
- Understanding How Image Quality Affects Deep Neural Networks https://arxiv.org/pdf/1604.04004.pdf
- paperswithcode.com
- https://carpentries.org/workshops/ - No experience with this organization but it came up the other day and looks very useful. From their website: "[These workshops] are for people who work with data in their research and want to learn how to code and organise their projects to work more effectively and reproducibly with data".