TUM-AMI

The Task Board : https://tum-ami.atlassian.net/jira/software/projects/AMI/boards/2

Welcome to TUM AMI Group 4 Documentation Wiki!

Some important links:

1. **OneDrive**
2. **Jira**
3. **Gitlab**
4. **Meeting Link** **(every Monday 7pm, starting 13th June), Passcode: 0000**
    a. Meeting Slides here
5. **Final Report (Overleaf)**

Here's an interactive overview from our Jira
https://tum-ami.atlassian.net/jira/software/projects/AMI/boards/2/roadmap

Recently updated

You'll see the 5 most recently updated pages that you and your team create.

cProfile – How to profile your python code
Aug 23, 2022 • contributed by Cisse, Amadou

July 18th 2022
Jul 18, 2022 • contributed by Xiaogang Wang

TUM-AMI
Jul 18, 2022 • contributed by Melina Soysal

Dataset
Jul 18, 2022 • contributed by Dennis

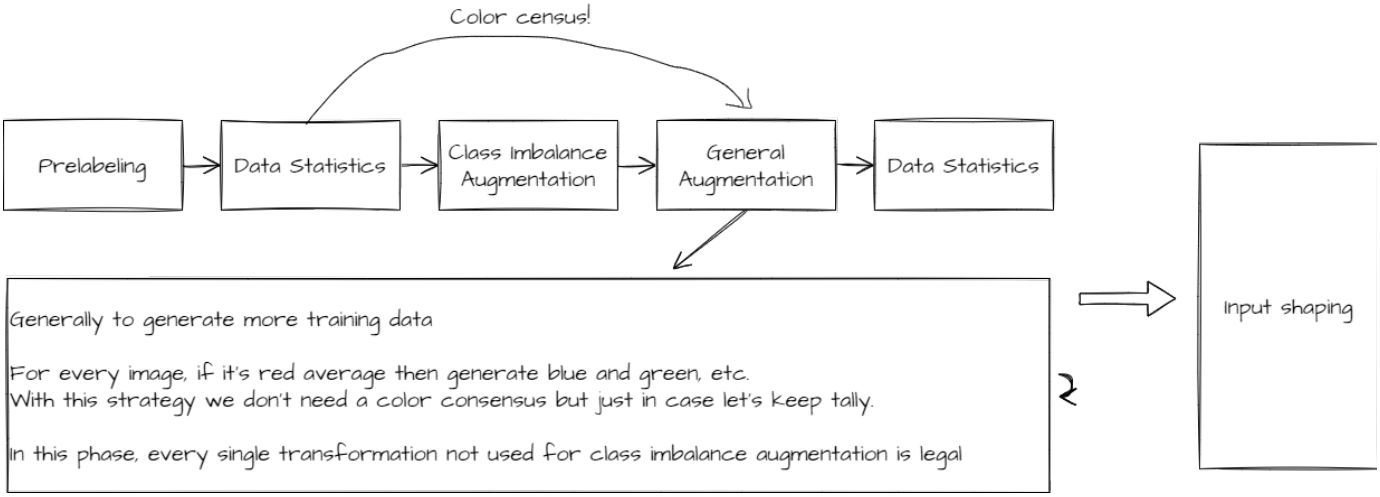
GitHub
Jul 14, 2022 • contributed by Cisse, Amadou

Documentation

Frontend

Backend

Data Preprocessing



The pptx here

The main purpose is to fix the class imbalance and to avoid making them when we generate more sample data,

Here is a rough strategy of what we could do for preprocessing for now.

Prelabeling

Every input data should be prelabeled before entering the preprocessing pipeline. (i.e. has an entry in the json)

Data Statistics

The census should return the image count of each damage type.

Each damage type should have its members broken down into red average, blue average, and green average.

The general idea is to keep class imbalances and unwanted feature extraction out.

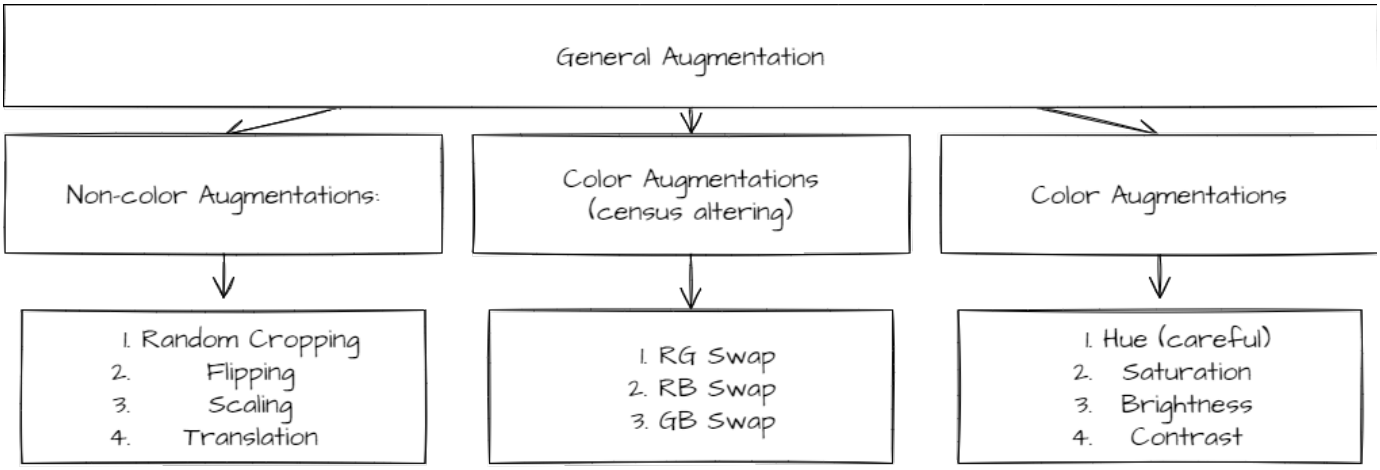
Class Imbalance Augmentation

In this phase only class imbalance is of importance. If there is more images with the color red it should be fine. By the end of the pipeline we will have the same count of red, green, and blue images.

Transformations used in this phase would be illegal in the general augmentation.

The reasoning is you would use the same transformation again on the same image after using it in this phase.

General Augmentation



Here everything we know about augmentation is to be used. Except illegal transformations we have defined.

The current strategy is:

1. Non-color transformations
2. Color transformations & everything else

For color augmentations, we'll have a count of the number of images based on its RGB average.

If it's average is red, it should generate blue and green average images (by swapping channels for example).
This way we will end up with the same number of red, green, and blue images, regardless of how imbalanced the count is.

After that any form of augmentation that doesn't shift the average should be legal.

But with this logic RGB augmentations should be pushed to the end after non-color augmentations are done.

Input Shaping

Finally before it's ready for input we'll have to reshape it into the input our model will need to receive

A resize function will do, like the one we saw in class.

Final Data Statistics

Final image census to make sure we were did a proper job augmenting the images :)

The Goal

At the end we'll have:

- A Preprocessing class
- Takes in a np.array or tensor or PIL Image (tbd)
- Generates for each damage type, a list of images(tbd)\
- A method to keep consensus count

Some augmentation will always be done (those in general augmentation)

i.e. it makes sense to keep the list using damage type.

If we're doing test-time augmentation then we need to decide on augmentation ids to keep track. (e.g 709123.png 709123_GBs_RD1.png (709123 green blue swap, random cropping 1)

To Discuss

Augmentation implementation methodology:

- Train-Time Augmentation vs Test-Time Augmentation (in memory vs in filesystem)

Dataset

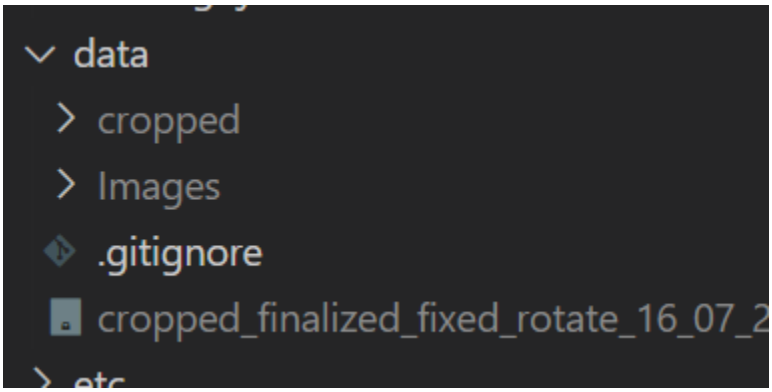
Current Status

Current Labels
dent
other
scratch
deformation

Entries under annotations hold these info now (restructured.json)

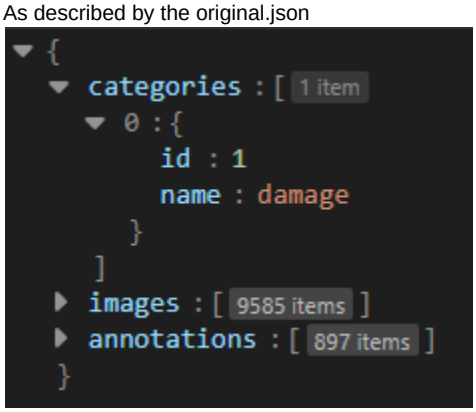
```
"images": [ ···
],
"annotations": [
    {
        "id": 504548,
        "category_id": 0,
        "image_id": 7057517,
        "filepath": "cropped/504548.png",
        "source": "Images/8f27c5f1-3ea3-4392-9bc8-585c76057cd1/858a0246-2f2d-40a9-9bcb-01ab8a93c7f5_DR80838_1630585604_1_26.jpeg",
        "bbox": [
            1050,
            649,
            65,
            52
        ]
    },
```


Put the dataset under data :



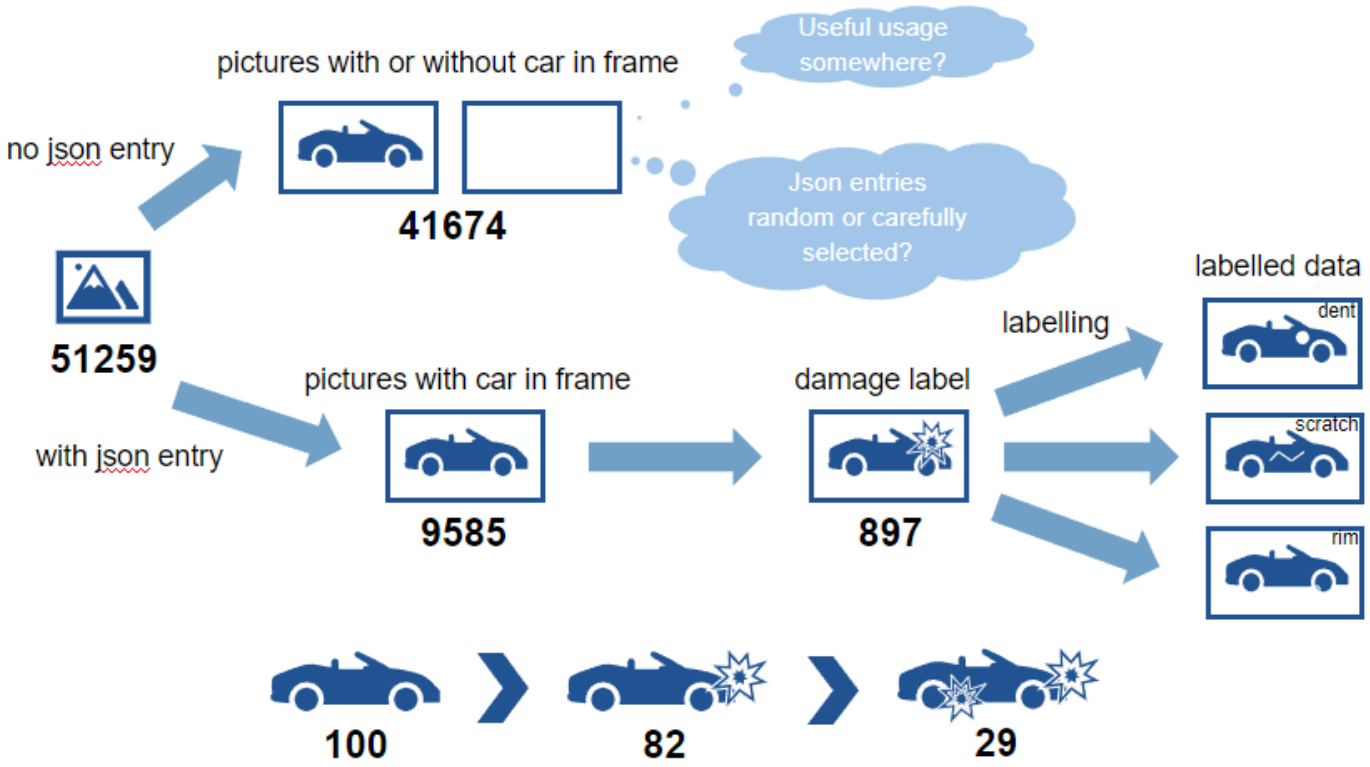
Original Dataset

As described by the original.json



There are 9585 labeled frames from 51259 total frames from 100 cars, 897 of which are damaged frames.

This script helps you go through the damaged frames (max index is 896)



Relabeling

I wrote a simple app under preprocessing branch.

Under /src/preprocessing and /notebooks/preprocessing are notebooks to :

1. regenerate the restructured.jsos annotations (cut_out_and_reparse.ipynb)
2. sort.ipynb to sort into a category folder structure
3. relabel_app.py to make relabeling easier (usable out of the box with Windows afaik)

(please don't replace the restructured.json with the cut_out_and_reparse.ipynb notebook, it's been relabeled manually.)

Meeting Notebook

General Meeting Link:

Melina Soysal lädt Sie zu einem geplanten Zoom-Meeting ein.

Thema: AMI All Hands Weekly
Uhrzeit: 13.Juni 2022 07:00 PM Amsterdam, Berlin, Rom, Stockholm, Wien
Jede Woche am Mo
13.Juni 2022 07:00 PM
20.Juni 2022 07:00 PM
27.Juni 2022 07:00 PM
4.Juli 2022 07:00 PM
11.Juli 2022 07:00 PM
18.Juli 2022 07:00 PM
25.Juli 2022 07:00 PM
1.Aug. 2022 07:00 PM
8.Aug. 2022 07:00 PM
15.Aug. 2022 07:00 PM
22.Aug. 2022 07:00 PM
29.Aug. 2022 07:00 PM
5.Sept. 2022 07:00 PM
12.Sept. 2022 07:00 PM
Laden Sie die folgenden iCalendar-Dateien (.ics) herunter und importieren Sie sie in Ihr Kalendersystem.
Wöchentlich: https://tum-conf.zoom.us/meeting/u50lfuGtqj4vEtJHwNQ8wrPy5UjV1eEUVKfN/ics?icsToken=98tyKu-hpjIiH9CQtRGBR_N5Gor4We_wiCVYj7d3tAjLMnhEcDPvY9dIBON6Ne3m

Zoom-Meeting beitreten
https://tum-conf.zoom.us/j/69898545586

Meeting-ID: 698 9854 5586
Kenncode: 903441

Meeting notes should be placed under this page:

Meeting Template

🗓️ Date

🥅 Goals

🗣️ Discussion topics

Time	Item	Presenter	Notes

✅ Action items

- [ ]

⤴️ Decisions

June 4th 2022

🥅 Goals

Update on current tasks, refine responsibilities, agree on weekly meeting

🗣️ Discussion topics

Item	Notes
Meeting preparation	• There will be one moderator which is agreed on in the previous meeting • Only topics on PowerPoint will be discussed, you can also just add slides with a headline if you want to have a discussion • PowerPoint and general meetings shall be lightweight, details in Confluence
Wiki (Dennis)	• Sign up for Confluence (https://tum-ami-wiki.atlassian.net/) • For Code Wiki you can also use markdown makes it easier for Code Documentation later • Meeting Notes are now part of the wiki • Keep Wiki more precise than PowerPoint, Power Point lightweight
Coding Style (Amadou)	• Will upload information and details on Coding Style to Wiki @all read through this file!
SharePoint (Melina)	• Only for PowerPoint (Meetings and Pecha Kucha) • Use naming convention: 2022-mm-dd_name
MLOps (Linyan)	• Short Into, more Details on Wiki and in next Meeting • See Ressource Page for Deep Dive
Jira (Gregor & Thu)	• Short How To, more Details to come in Wiki • always keep Jira up to date to let other people know of your progress

✅ Action items

- [ ] see Jira

⤴️ Decisions

- Weekly Meetings will be on Monday starting from 7PM
- Next Meeting moderator: @Linyan

June 13th 2022

🥅 Goals

Get team up to speed with several topics in the presentation:

🗣️ Discussion topics

Item	Notes
Anouar	GitLab Dockerize and CI/CD pipeline, containerized frontend and backend. Settled on yarn + Pipenv (link to Anouar's example : https://gitlab.com/anouar1073/flask_app), try it it's great
Linyan	MLOps maybe set out seperate time for us to discuss about technical ML topics
Amadou	Formatter AutoPEP8 vs Black vs Yapf yapf seems to be preferred. Formatting entire file in one go could create merge conflicts. maybe use Newline? Other Extensions: 1. Bookmark (mark code lines) 2. GitHistory & GitLens 3. Bracket Pair colorizer 4. Better Comments **we should discuss about logging in our pipeline**
Team	Pecha-Kucha Topic seems to be MLOps , PechaKucha is 20.06 Melina + Linyan

✅ Action items

- [ ] set out time to discuss presentation look at slack poll
- [ ] setup formatter VSCode wiki page to be brought up
- [ ] setup dockerize anouar
- [ ] Look at the reading tasks and if you're interested assign yourself to it people with interest other wise already assigned.
- [ ] Get yourself familiar with the data
- [ ] look into the dataset, get some insight dennis anouar amadou
- [ ] data set people also read papers on data augmentation

⤴️ Decisions

🌱 Everyone do literature research and prepare some presentation

June 20th 2022

🗓️ 20.02.2022

- @ Cisse, Amadou @ Linyan Yang @ Melina Soysal @ Xiaogang Wang @ Anouar Laouichi @ Khanh Nguyen @ Dennis @ Nils Stanelle @ Magdalena Frey @ Gregor Höhne

🥅 Goals

Get team up to speed with several topics in the presentation:

- Pecha Kucha
- Data Analysis
- Literature Research

- Update on VS Code (Autoformat, etc.)
- Update on GitLab set-up

🗣️ Discussion topics

Time	Item	Presenter	Notes

✅ Action items

- [ ] Enriching augmenting dataset Denis, (Nils)
- [ ] Pipeline Amadou, Thu, Gregor, Linyan
- [ ] CI Amadou, Nils

⤴️ Decisions

June 26th 2022 - Preprocessing

🗓️ Date

- @ Nils Stanelle
- @ Xiaogang Wang
- @ Dennis
- @ Magdalena Frey
- @ Khanh Nguyen

🥅 Tasks

- Use Linyan's skript to cut out the bboxes & generate a new JSON + pre labeling (Xiaogang until Sunday)
- Skript to change lables (Magdalena until Sunday)
- double check pre labeled data (Dennis, Nils, Thu until the monday meeting)
- Augmentation of *labeled*, cut out images (tbd)

June 27th 2022

🗓️ Date

27 Jun 2022

👥 Participants

- @ Dennis

🥅 Goals

- Decide on next milestone to accomplish within a given timeframe

🗣️ Discussion topics

Time	Item	Presenter	Notes
Data Preprocessing		Dennis	• Image relabeling in progress
MLOps TFX		Linyan	• Many tutorial available • Implementation of simple guiedline notebook • ML orchestrator not compatible with orchestrators • Explore possibilities for running in windows
CI/CD		Anour	• CI/CD setup and working

- Co-coding sessions on Tuesday 29.06 and Monday 03.07

✅ Action items

- [ ]

⤴️ Decisions

July 4th 2022

Attendees:

@ Cisse, Amadou | @ Anouar Laouichi | @ Melina Soysal | @ Linyan Yang | @ Xiaogang Wang | @ Magdalena Frey | @ Gregor Höhne

Discussions:

- Amadou
    - New folder restructuring on Confluence
    - Added a "how to contribute" section for quick onboarding.
        - Added guidelines for GitHub
        - Added guidelines for Python and Javascript
    - Gitlab repo restructuring
    - Added notebook repo to share experiments
    - Added naming conventions for file, folder, and branch names
    - What will be added:
        -
- Anouar- Pipeline
    - Pipeline is mostly ready
    - Issues with local runners
    - Everyone needs to set up local runners
- Madgalena - Data processing
    - Data is already labeled.
    - Dennis and Xiaogang found an issue with JSON files information:
        - Some ids in the JSON file refer to the same image
        - Need to make sure that the id exists or not and avoid overriding it.
        - The issue will not block data augmentation part
    - Augmentation:
        - Image resizing
        - Image preprocessing with NN (Gregor's idea)
        - Meet and define the next steps
        - Use TensorFlow or Numpy for preprocessing?
- Melina - MLFlow
    - Decided to ditch TFX, MLflow seems to be a better alternative
    - Started to set up some tutorials
        - Install MLflow
        - Sample notebook will be uploaded to showcase MLflow
        - MLflow documents the model, metrics, and batch size.
        - Important question: how to push it to git
        - Tool that allows
            - to lock versions for notebook experiments
            - Compare experiments.
- Amadou – Transfer learning
    - Added notebook for transfer learning tutorial
    - Added repo for transfer learning
    - Used keras for NNs
        - Train last layers for finetuning
        - Got good accuracy
    - In the future: use state-of-the-art NNs from Keras.
    - Should look into approaches like self-supervised learning
- Responsibilities:
    - Web application -> between Anouar and Magdalena.
    - Preprocessing -> Dennis and Xiaoagang
        - Dennis comes up with a proposal for data preprocessing by next week
    - ML -> potentially Linyan
        - Transfer learning - > Amadou
        - Self-supervised learning -> Melina
        - Active-learning -> Anouar
        - MLFlow merge (with Kubernetes?)
- Git Rules:

    - Make Nils maintainer
    - Make commits often
    - For more guidelines -> go to the general presentation slides of july 4th 2022
- Amadou -> go to confluence to more guidelines, see, Later PEP's, Tips & Tricks
- Next meeting moderator -> Anouar
- Next meeting notes -> Magdalena
- Next co-coding session-> Thursday afternoon
- Next monday meeting -> 6 pm

July 11th 2022

@🗓️ Date

🥅 Goals

Todays Presenter: @ Anouar Laouichi

🗣️ Discussion topics:

Item	Presenter	Notes
KNN Classifier	@ Melina Soysal	• Self supervised: 66% accuracy • more images for pretraining (add pictures with no defects) • try more refined classifiers
Web-App	@ Anouar Laouichi , @ Magdalena Frey	• Infrastructure is finished • Backend and frontend there • some logic is missing • some UI done (function first) (Using Material-UI) • some Text, Structure
Preprocessing	@ Xiaogang Wang	• Label: wrong right now • Finish Labeling this week (same ID, handling) • Augmentation: Non-color + color augmentation done • use models to see, what augmentation is good
Preprocessing	@ Magdalena Frey	• use zero-padding • first implementation of bilinear interpolation done, to achieve same resolution • will be included on Thursday
Transfer Learning	@ Cisse, Amadou	• Notebook done on seperate Branch • Only possible with GPU (half an hour, CPU 2 to 3 hours) • skewed data • Visualization • sizes of images, pixel distribution • used augmentation in the notebook, 3000 samples • training: 78% when training all layers
Transfer Learning Combined with Active Learning?	@ Anouar Laouichi , @ Cisse, Amadou	• maybe apply active learning, in the transfer learning approach • use it to the frontend (first transfer learning, then active learning) • use Mobilnetv2, too slow on machine, just 50% • maybe apply it to simplier models
Issues on data	@ Linyan Yang	• 780 images only • some images, label is not correct • put data away, that is not sure pictures

✅ Action items

- [ ]

⤴️ Decisions

- next moderator:

July 14th 2022 - Co-coding Session

- @ Cisse, Amadou @ Nils Stanelle @ Magdalena Frey @ Gregor Höhne

🥅 Goals

Get the modelling team up-to-date with the progress made by the data team.

- What has been done
- Which difficulties persist and how to solve them?

Making the preprocessing available for the modelling team.

- Data loader
- Data augmentation

🗣️ Discussion topics

Item	Presenter	Notes

State of the dataset	@ Magdalena Frey	• 200 cropps are unlabeled
		  • Images with multiple crops yielde d single crop only
		  • @ Nils Stanelle @ Dennis @ Magdalena Frey are working on this
		• 740 images are available but some cannot be labeled by hand because it is not obvious how to classify them
		  • Idea: allow classification between two classes
		  • e.g. 0.6 dent, 0.4 other
		  • This allows the classifier to learn uncertainty

Dataloader	@ Cisse, Amadou	• We want a dataloader, which resembles the image_dataset_from_directory function provided by tf. • We want the ability to specify class weights, since the dataset is unbalanced • The dataloader should not change the class distributions inherent to the dataset • @ Gregor Höhne is attacking this task, till monday Args: • directory: folder from which to read the images • val_size: percentage of total set assigned to validation set • test_size: analogously for test set • image_size: width and height as tuple • batch_size: images per batch as integer • cropping_type: as in exercise notebook • class_weights: 1- class_count/dataset_size • datatype[numpy array, tensorflow dataset] Return: • train(_ds): numpy or tf dataset of dimensions (#batches, batch_size, W, H, channels) • test(_ds): if test_size != 0 • val(_ds): if val_size != 0

Data Augmentation Module	@ Cisse, Amadou	• We want an augmentation module, which allows us to experiment with data augmentation without having to implement overhead code before doing so • @ Xiaogang Wang x @ Dennis have code available in a notebook, but not as module • Code cannot be used in its current state • Consider implementing this as a class object e.g. DataAugmentor(methods=[“nan”, “nana”], flip_method=”horizontal”, seed=42) • The transformation setup could init, the augmentation could be done by e.g. DataAugmentor.transform(train_ds) • Implementation with the following args by @ Nils Stanelle or @ Magdalena Frey or @ Xiaogang Wang or @ Dennis Args: • Dataset • Random_seed: integer to make the upsampling reproducible • target_size: upsampling size of the dataset • methods: dictionary with custom arguments where the key is the method name and the value a keyword argument dictionary (this is debatable, any better solution is highly welcome) Returns • Augmented data with dimensions (#batches, batch_size, W, H, channels)
Data augmentation research	@ Cisse, Amadou	• To understand which data augmentation method makes sense for our dataset, we need to explore this literature • e.g. random cropping might not be valid for dents or scratches, since it might crop a region in which there is no damage • e.g. is gray-scaling a valid data augmentation for our task or detrimental to the classification effort • @ Dennis @ Magdalena Frey @ Xiaogang Wang @ Nils Stanelle
Cleaning the preprocessing branch	@ Cisse, Amadou	• The branch naming convention is not respected: rename to experimental_preprocessing • Use new branches for refactor e.g. feature_data_augmentation and feature_data_loading for source module implementation • Remove empty notebooks and unused source file to prevent cluttering

✅ Action items

- [ ] Bringing the dataloader online
- [ ] bringing the data augmentation function online
- [ ] Presenting research results

⤴ Decisions

July 18th 2022

- @ Cisse, Amadou @ Nils Stanelle @ Magdalena Frey @ Gregor Höhne @ Dennis @ Linyan Yang @ Xiaogang Wang @ Anouar Laouichi @ Melina Soysal

📖 Topic

- New Json file, preprocessing status (Dennis)
- Data Loader (Gregor)
- Data Augmentor (Nils)
- CI/CD, Super-Resolution and Transfer Learning (Amadou)
- Petcha Kutcha rehearsal (Anouar and Amadou)
- Rapport and Video discussion

🗣 Discussion topics

Item	Presenter	Notes
New Json file, preprocessing status	@ Dennis	• New jason is done, can be found in Gitlab • Need to decide label and define labels: (maybe can make two versions datasets) • Create a folder for hard to distinguish categories of images, make vote for some images hard to decide the label • try DEEP AUTOAUGMENT @ Dennis
Dataloader	@ Gregor Höhne	• Dataloader is done in GItlab • make a notebbok, instructions for use. @ Gregor Höhne
Data Augmentation Module	@ Nils Stanelle	• Not done yet,will be done tomorrow or Wednesday @ Nils Stanelle • will make a notebook , instructions for use @ Nils Stanelle
Regularization by Destroying Information	@ Cisse, Amadou	• Some dents are smaller than others • Train the network for lower resolution • Increase robustness
CI/CD, Super-Resolution and Transfer Learning	@ Cisse, Amadou	• Branch name linting • File name linting • Add name of automatically generated files to exclusion list • Does everyones auto-formater setup work? • Check if you followed all steps from how to contribute

Image Resiziing	@ Cisse, Amadou	• Currently: Super-resolution using bicubic interpolation (KNN based)
		• What else can we do?
		  • Find other models
		  • e.g. Pre-trained GAN

Transfer Learning: Moving Forward	@ Cisse, Amadou	• Save relabeled data in buffer • Train on relabeled data when buffer is full • choose data augmention • Hyperparameter optimisation @ Cisse, Amadou @ Linyan Yang
• Petcha Kutcha rehearsal	@ Anouar Laouichi @ Cisse, Amadou	• There is one page no counter, need to confirm before the Pre @ Cisse, Amadou • pay attenation of “simple” @ Cisse, Amadou
• Rapport and Video discussion		• Structure of the report @ Melina Soysal • Assigning tasks after completing the structure @ Melina Soysal • Video: brainstorming , Collection of interesting ideas. Next meeting to discuss

✅ Action items

- [ ]

⤴ Decisions

Literature

Add all relevant papers in the corresponding categories. Please also add your name if you have read a paper in order for us to know who to contact in case we have questions :)

Other resources

In addition to the listed sources on the individual specific topics, you can also add other sources that you consider to be informative on a broader topic. These include books, e-learning or even YouTube channels.

Short description	Source	Responsible person (if applicable)
YouTube Channel related on modern ML/DL papers	https://www.youtube.com/c/YannicKilcher	☑ Linyan
TensorFlow 2 (E-Learning) (Free for TUM Students)	https://de.coursera.org/specializations/tensorflow2-deeplearning	☑ Linyan
Deep Learning with PyTorch (Code tutorial)	https://github.com/deep-learning-with-pytorch	☑ Linyan
Deep Learning with PyTorch (Book)	https://www.ub.tum.de/eaccess	☑ Linyan

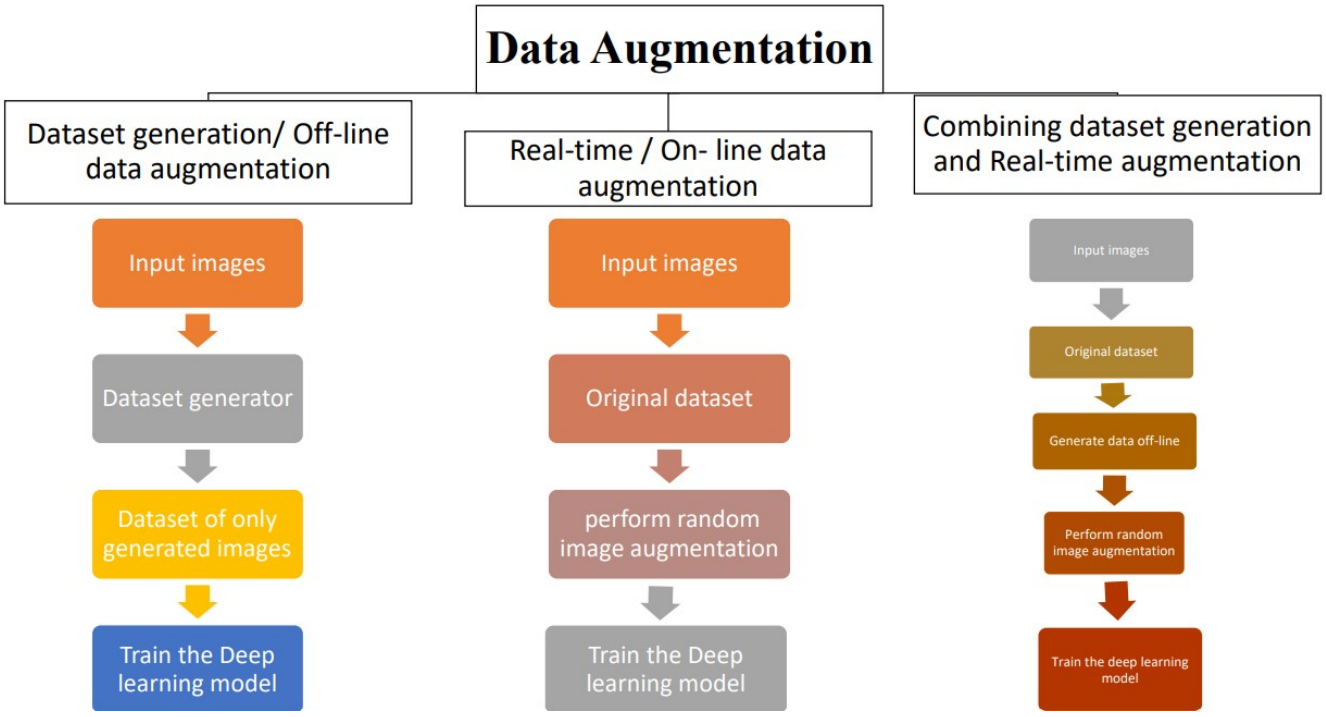
Data

Data Augmentation

In the following table, the topic of data augmentation is explained in more detail. This includes the theoretical background as well as tutorials and application examples with which this can be done.

If you add new sources or if you have read / familiarized yourself with one of the listed topics, please enter your name in the table as well. Use a checkmark to show that you are already done with this task.

Short description	Source	Responsible persons
Survey Data Augmentation Survey: Short Summary	https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0197-0	☑ Dennis
Tutorial	https://www.tensorflow.org/tutorials/images/data_augmentation	☑ Linyan
Tutorial	https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/data_augmentation.ipynb	☑ Xiaogang
Data Augmentation and Examples	https://nanonets.com/blog/data-augmentation-how-to-use-deep-learning-when-you-have-limited-data-part-2/	☑ Xiaogang



Data Augmentation Survey: Short Summary

One of the issue to solve is class imbalances in data sets, which is where data augmentation can come into play.

Potential Issues with Data Augmentation

A non-label preserving transformation could potentially
strengthen the model's ability to output a response indicating that it is not confident about its prediction.
However, achieving this would require refined labels post-augmentation.

Forms of Data Augmentation

1. Geometric
2. Color space manipulation
3. Kernel filters
4. Feature space augmentation

Transformations could result in label not being preserved i.e.

However if the label of the image after a non-label preserving transformation is something like [0.5 0.5], the model could learn more robust confidence predictions.

Allegedly constructing refined labels for every non-safe Data Augmentation is a computationally
expensive process

Geometric Augmentation

In many of the application domains covered such as medical image analysis, the biases distancing
the training data from the testing data are more complex than positional and translational variances. Therefore, the scope of where and when geometric transformations can be applied is relatively limited

Cropping results in the most accurate classifier, after that is rotating.

Color Space Manipulation

color space transformations will eliminate color biases present in the dataset in favor of spatial characteristics. However, for some tasks, color is a very important distinctive feature.

Kernel Filters

The lower-dimensional representations of image data in fully-connected layers can be extracted and isolated. Konno and Iwazume found a performance boost on CIFAR-100 from 66 to 73% accuracy by manipulating the modularity of neural networks to isolate and refne individual layers after training

Fancy PCA performs great in "Top-5" Accuracy r elative to its peers.

Kernel filters -> Sharpening/Blurring (nxn matrix insertion across an image (gaussian filter etc) -> results in blurrier image

Or high contrast vert/horizontal edge filter(?) -> sharper image

Patch Shuffle Regularization:
experiment with a unique kernel flter that randomly swaps
the pixel values in an n×n sliding window. Tey call this augmentation technique
PatchShufe Regularization. Experimenting across diferent flter sizes and probabilities
of shufing the pixels at each step, they demonstrate the efectiveness of this by achieving a 5.66% error rate on CIFAR-10 compared to an error rate of 6.33% achieved without the use of PatchShufe Regularization.

Te hyperparameter settings that achieved
this consisted of 2×2 flters and a 0.05 probability of swapping

Feature Space Augmentation

Notes

Mixing:
RGB values are averaged between pixels of two randomly selected, horizontally flipped images.
Ionue reported a reduction in error rate from 8.22 to 6.93%
when using the SamplePairing Data Augmentation technique. Te researcher found
even better results when testing a reduced size dataset, reducing CIFAR-10 to 1000 total
samples with 100 in each class. With the reduced size dataset, SamplePairing resulted in
an error rate reduction from 43.1 to 31.0%.

t better results were obtained when mixing
images from the entire training set rather than from instances exclusively belonging
to the same class

Detecting Data Issues

This section discusses how we can identify problems in our data and what we can do about them. First, some of the commonly used technical terms are listed and explained.

Drift and skew

Drift - Changes in data over time, such as data collected once a day.

Skew - Difference between two static versions, or different sources, such as training set and serving set.

Image resizing

In this section we will discuss how to handle different image sizes. To keep the input size consistent in the ML model, images of different sizes have to be adjusted accordingly. For example, smaller images can be padded with zero padding or interpolated. The techniques available and their importance in the context of Deep Learning will now be analyzed.

If you add new sources or if you have read / familiarized yourself with one of the listed topics, please enter your name in the table as well.

Short description	Source	Responsible persons
Zero-padding vs. interpolation	https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0263-7	☐ Thu

Enlarging smaller images before inputting into convolutional neural network: zero-padding vs. interpolation

https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0263-7

Labeling

Here are a few commonly used labeling tools.

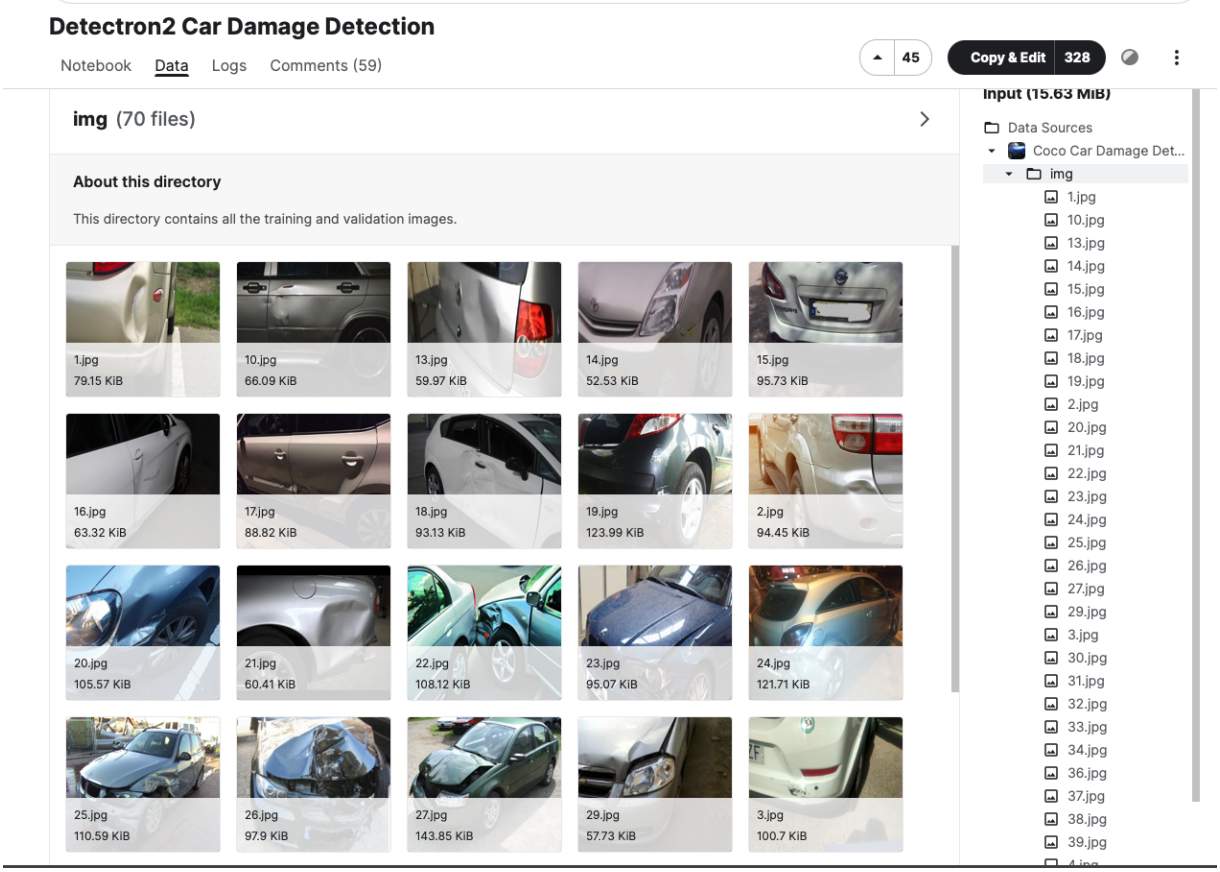
According to the latest project description, we need to design a GUI for labeling. My idea is to have an interface where people can enter images, enter categories and then generate a JSON file. The following labeling tools are available for reference.

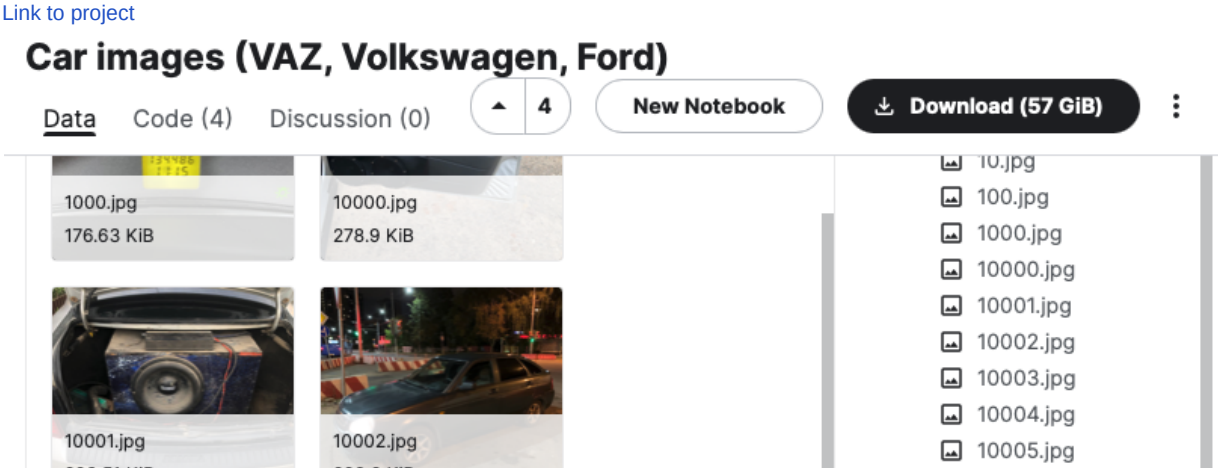
Tools	source	
The best image labeling tools for Computer Vision	awesome-data-labeling	

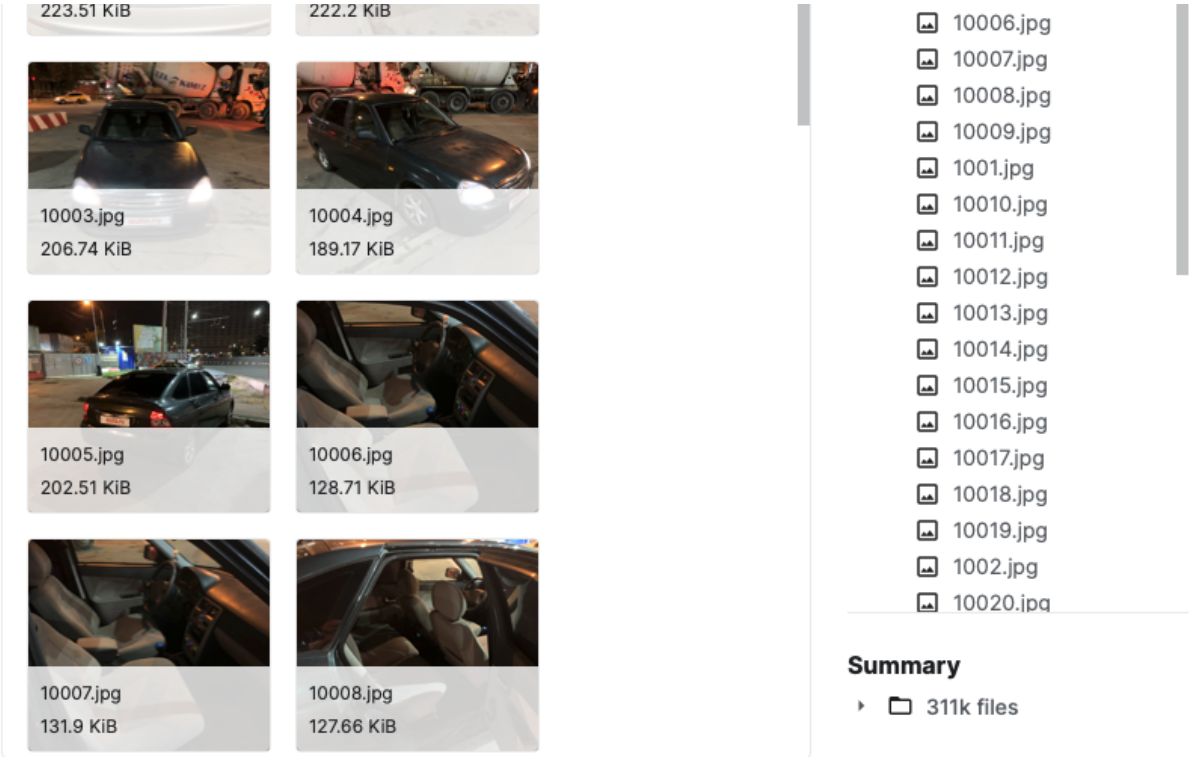
Additional training data

We might consider using other free datasets of cars and car damages.

I found some projects, that we could use on kaggle.

- Coco Car Damage Detection Dataset:
  - 70 pictures of clear damages on cars:
  - Link to dataset
  - Detectron2 Car Damaged Parts Detection (PyTorch library)
  - Link to Project
  - 

- Car Car Images
  - 311269 images of cars (also used cars with damages)
  - Link to project
  - 



- Car Image Dataset
  - 4165 files of new cars, without damages
  - categorized into brands
  - maybe also useful for training
  - Link to project

Machine Learning

Active Learning

Based on the assignment of the project, currently there is only a small amount of labeled data. One option would be to label all the data ourselves and perform ordinary supervised learning. However, it also raises the question of how sample efficient we want to train our model. Is it sufficient if we only provide a few labeled data and assist the convergence of the learning algorithm with human feedback?

If you add new sources or if you have read / familiarized yourself with one of the listed topics, please enter your name in the table as well.

Short description	Source	Responsible person
Blog post	https://www.datacamp.com/tutorial/active-learning	☐ @ Linyan Yang
Survey	https://arxiv.org/abs/2009.00236	☐ @ Linyan Yang
Book Chapter Active Learning: Learning with Limited Data	Learning with Limited Data	☐ @ Melina Soysal
Variational Adversarial Active Learning	https://arxiv.org/pdf/1904.00370.pdf	☐ @Linyan Yang

Active Learning: Learning with Limited Data

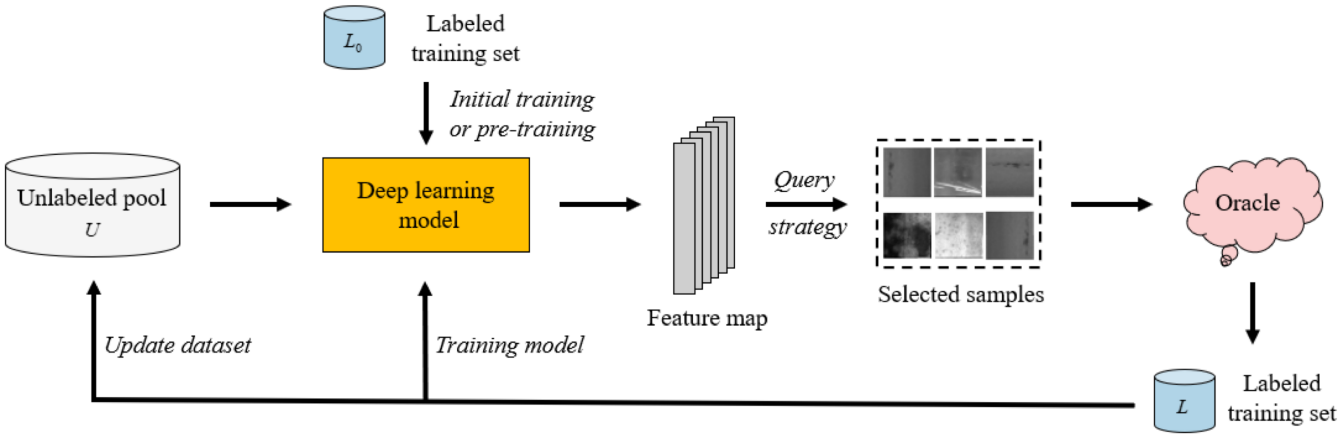
This book chapter has also been a reading assignment in the lecture

- When having a lot of data one needs to find a way to **smartly label** the data: When using active learning one starts to label a few samples, trains a model on these samples and uses this model to label the other samples. The model flags the data points that were harder to predict to request for human confirmation of the labels. This can be an iterative process such that the model performance trends upwards.
- **Random sampling** / passive learning: randomly start labeling data (+ can be used as baseline, jumpstart model; - if labeling uninformative / random data, more data might be necessary for same model performance)
- **Uncertainty sampling**: select data points to label on how uncertain they are to predict

    - Margin sampling: label points closest to decision boundary, label very informative data points (**fundamental trade-off in active learning: informativeness vs. representativeness**)
    - Entropy sampling: a strong confident misclassification should be more punished than a weakly confident misclassification, label those points with high entropy, results in a more representative dataset compared to margin sampling
    - Other: pick from dense regions with many datapoints
- Active learning is dependent on **learner** (linear, nonlinear) and **strategy**
    - to prevent choosing the wrong learner (which leads to bad performance, more labeling isn't improving the model), one can use a "committee" of diverse learners

Deep Active Learning Survey: Short Summary

Through areas where labeling data requires expert knowledge, learning algorithms using a few labels was incentivized and reignited research interest in active learning (AL). AL reduces the cost of sample annotation while retaining learning capabilities of DL.



Deep Learning

- This survey contains a short summary of the history of DL.
- Supervised DL has a strong greedy attribute to the data.

Active Learning

- Goal: Obtaining as many performance gains as possible by labeling as few samples as possible.
- Finding useful unlabeled samples to hand over to the oracle (e.g. human annotator) for labeling.
- Approaches can be divided into **membership query synthesis, stream-based selective sampling and pool-based AL.**
- Main difference between stream-based selective and pool-base sampling:
    - stream-based selective sampling makes an independent judgement on wether each sample in the data stream need to query labels of unlabeled samples. **Normally targeted at smaller devices** due to storage limitations.
    - pool-based chooses the best query based on the evaluation and ranking of the entire dataset. **More common strategy**.
- **Design of the query rules is crucial** for the performance of the AL method. Different methods include
    - Uncertainty-based approach
    - Diversity-based approach
    - Expected model change
    - hybrid query strategies

Choosing a query strategy is often not simple as the also include potential problems. Sampling based on uncertainty might result in sampling bias und might not be representative of the dataset. Diversity based strategies might lead to increased labeling costs.

The necessity and challenge of combining DL and AL

- Model uncertainty in DL
- Insufficient data for labeled samples
    - Research in generative networks for data augmentation or assigning pseudo labels to high confidence samples
- Processing pipeline inconsistency

Query Strategy Optimization in DeepAL

- Batch Mode DeepAL
    - Replaces the one-by-one sample query with a batch based query strategy.
- Uncertainty-based and Hybrid Query Strategies
    - Considers multiple sample attributes in addition to uncertainty ranking.
- Deep Bayesian Active Learning
    - Based on bayesian convolutional neural networks.
- Density-based methods
    - Attemps to find a core subset representing the distribution of the entire dataset.

To be continued …

Self-supervised Learning

In order to teach children what an elephant is, we can show them some pictures of elephants and they will (almost) always be able to classify an elephant correctly. We do not need to show them thousands of pictures of elephants in different scenarios. However, a learning algorithm is not able to do this. If it does not receive enough diverse samples, it will not be able to recognize the animal correctly. The strict limitation of machine learning potential is therefore limited by the amount of labeled data. The difference to humans is that AI does not have so-called " common knowledge" like humans. Yann LeCun calls this the dark matter of intelligence.

With the help of self-supervised learning, we want to bypass this obstacle. It is not possible to label everything in the world. But maybe it is not necessary. Examples of successful self-supervised learning algorithms are Bert in NLP or Dino in Object Detection.

Short description	Source	Responsible person
Blog post	https://blog.tensorflow.org/2022/02/boost-your-models-accuracy.html	☐ @ Linyan Yang
Tutorial Siamese Network	https://keras.io/examples/vision/simsiam/	☐ @ Linyan Yang
Blog post / YouTube Video Dark Matter of Intelligence	https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence/	☑ @ Linyan Yang
Github / Paper / YouTube Video Dino	https://github.com/facebookresearch/dino	☑ @ Linyan Yang
Contrastive Self-Supervised Learning	https://ankeshanand.com/blog/2020/01/26/contrative-self-supervised-learning.html	☑ @ Linyan Yang
The Illustrated Self-Supervised Learning	https://amitness.com/2020/02/illustrated-self-supervised-learning/	☑ @ Linyan Yang
Self-supervised learning and computer vision	https://www.fast.ai/2020/01/13/self_supervised/	☑ @ Linyan Yang
Paper Generative or Contrastive?	https://arxiv.org/abs/2006.08218	☐ @ Khanh Nguyen

Self-supervised learning: The dark matter of intelligence

- **Supervised learning: bottleneck** as requires large amount of labelled data —> for many tasks not possible, need to find a different way for AI to understand reality and generalise
- people / animals learn with **common sense**, children only need few images of an animal to recognise them (good generalisation ability) vs. machine needs a lot of examples —> humans rely on previously acquired background knowledge
- **self-supervised learning:** learn from lot of data to recognise & understand patterns; already good success in NLP pre-training
- self-supervised = **predictive learning:** obtains **supervisory signals** from data itself, e.g. by **underlying structure**, general technique: predict unobserved / hidden part of input from observed / unhidden part (e.g. beginning / end of sentence)
    - for **text**: mask/blank word, then AI needs to find word that fits the context
    - for **images**: considerably harder to represent **uncertainty** in **images**, no technique to represent suitable probability distribution over high-dimensional continuous space (e.g. video frames)
- **energy-based model**: given two inputs, tells us **how incompatible** these are (the higher the more incompatible)
    - 1.: show examples that are compatible and train for low energy
    - 2.: ensure if x & y are incompatible then high energy (harder part)
- **joint embedding/ siamese networks**: composed of two (almost) identical copies of the same network, feed one with x, other one with y, third module computes energy between resulting embedding vectors, adjust parameters if input similar such that energy is lower, hard for different inputs x and y to receive high energy. problem: **collapse** if energy is not higher for non matching x and y than matching x and y
- How to **prevent collapse**?
- **Contrastive energy based** SSL: adjusting parameters such that for incompatible pairs x, y high energy, but computationally expensive as high dimensionality for images / video, e.g.: latent-variable predictive architectures
- Non-contrastive energy-based SSL: not explicitly pushing up energy of incompatible pairs

Representation Learning

Representation learning is concerned with training machine learning algorithms to learn useful representations, e.g. those that are interpretable, have latent features, or can be used for transfer learning. To summarise, use the algorithm to obtain features of the data and then use the

features for further work. which then proceeds to the next step, such as our classification task. In image classification and target detection, representation learning can be mainly classified as supervised learning, unsupervised learning, self-supervised learning, Semi-Supervised Learning, etc.

Title	Short description	Source	Responsible person
Representation Learning | Papers With Code	The concept of representation learning is introduced and popular articles in the field are presented. In addition to this, a large number of open source related algorithms are presented.	https://paperswithcode.com/task/representation-learning	@ Xiaogang Wang
A Survey on Semi-, Self- and Unsupervised Learning in Image Classification	For picture classification, this article focuses on several types of unsupervised learning, semi-supervised learning, to obtain features of pictures.	https://arxiv.org/abs/2002.08721	@ Xiaogang Wang
Representation Learning: A Review and New Perspectives	This paper reviews recent work in the area of unsupervised feature learning and deep learning, covering advances in probabilistic models, auto-encoders, manifold learning, and deep networks.	https://paperswithcode.com/paper/representation-learning-a-review-and-new	@ Xiaogang Wang

Transfer Learning

Title	Short description	Source	Responsible person
Example Project Transfer Learning Tutorial	In this notebook, we use our project data to fine tune a pre-trained model.	https://www.tensorflow.org/tutorials/images/transfer_learning	@ Nils Stanelle @ Linyan Yang

Transfer Learning Tutorial

Transfer Learning Tutorial

This tutorial demonstrates how to perform a simple transfer learning task with the data generated by Xiaogang.(https://ami2022group4.slack.com/archives/C03LKBAAJKW/p1656146461494769)

This tutorial is based on a tutorial on the TensorFlow page. You can therefore also use the following page as a reference. (https://www.tensorflow.org/tutorials/images/transfer_learning). The following tutorial is not about creating a perfect model, but only about showing how a transfer learning task could look like.

1. Imports

2. Setting up your directory

3. Applying Transfer Learning

4. Fine Tuning

Before you start the tutorial please download the above linked zip file from Xiaogang to your current path and unzip it.

---

1. Imports

Required imports and loading image data.

```
import json
import os
from PIL import Image

import tensorflow as tf
import matplotlib.pyplot as plt
```

```
# get current working directory
cwd = os.getcwd()

# open JSON file
with open(os.path.join(cwd, 'cropped_images_with _label', 'annotated_functional_test3_fixed_new.json')) as f
    data = json.load(f)
```

```
# get informations
categories = data['categories']
images = data['images']
annotations = data['annotations']

print(f'The classes are: {[x["name"] for x in categories]}')
```

```
The classes are: ['dent', 'other', 'rim', 'scratch']
```

2. Setting up your directory

The following block depends on your operating system.

```
from IPython.display import clear_output

# windows

! mkdir "cropped_images_with _label/images"
! mkdir "cropped_images_with _label/images/dent"
! mkdir "cropped_images_with _label/images/other"
! mkdir "cropped_images_with _label/images/rim"
! mkdir "cropped_images_with _label/images/scratch"

# unix
# ! mkdir "cropped_images_with _label\images"
# ! mkdir "cropped_images_with _label\images\dent"
# ! mkdir "cropped_images_with _label\images\other"
# ! mkdir "cropped_images_with _label\images\rim"
# ! mkdir "cropped_images_with _label\images\scratch"

clear_output()
```

2.2 Move all files to the correct folder

If you have already moved the images you can skip this part.

```
# loop over all annotations
for annotation in annotations:

    try:
        # get image informations
        image_id = annotation['image_id']
        category_id = annotation['category_id']
        category_name = categories[int(category_id)]['name']

        # move file to correct directory
        from_path = os.path.join(cwd, 'cropped_images_with _label', 'cropped_images', str(image_id) + '.png'
        to_path = os.path.join(cwd, 'cropped_images_with _label', 'images', category_name, str(image_id) + '

        # move file
        os.replace(from_path, to_path)
    except:
        pass
```

3. Applying Transfer Learning

3.1 Load data from disk

The data is then split into a training and validation set.

```
image_path = os.path.join(cwd, 'cropped_images_with _label', 'images')

train_ds = tf.keras.utils.image_dataset_from_directory(directory=image_path, validation_split=0.3, subset='t
val_ds = tf.keras.utils.image_dataset_from_directory(directory=image_path, validation_split=0.3, subset='val
```

```
Found 731 files belonging to 4 classes.
Using 512 files for training.
Found 731 files belonging to 4 classes.
Using 219 files for validation.
```

3.2 Configure the dataset for performance

Use buffered prefetching to load images from disk without having I/O become blocking. To learn more about this method see the data performance guide.

```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.prefetch(buffer_size=AUTOTUNE)
```

3.3 Use data augmentation

When you don't have a large image dataset, it's a good practice to artificially introduce sample diversity by applying random, yet realistic, transformations to the training images, such as rotation and horizontal flipping. This helps expose the model to different aspects of the training data and reduce overfitting.

```
data_augmentation = tf.keras.Sequential([
  tf.keras.layers.RandomFlip('horizontal'),
  tf.keras.layers.RandomRotation(0.2),
])
```

3.4 Rescale pixel values

In a moment, you will download tf.keras.applications.MobileNetV2 for use as your base model. This model expects pixel values in [-1, 1], but at this point, the pixel values in your images are in [0, 255]. To rescale them, use the preprocessing method included with the model.

```
rescale = tf.keras.layers.Rescaling(1./127.5, offset=-1)
```

3.5 Create the base model from the pre-trained convnets

You will create the base model from the MobileNet V2 model developed at Google. This is pre-trained on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes.

```
# Create the base model from the pre-trained model MobileNet V2
base_model = tf.keras.applications.MobileNetV2(input_shape=(224, 224, 3),
                                               include_top=False,
                                               weights='imagenet')
```

3.6 Freeze the convolutional base

It is important to freeze the convolutional base before you compile and train the model. Freezing (by setting layer.trainable = False) prevents the weights in a given layer from being updated during training. MobileNet V2 has many layers, so setting the entire model's trainable flag to False will freeze all of them.

```
base_model.trainable = False

# Let's take a look at the base model architecture
# base_model.summary()
```

3.7 Add a classification head

To generate predictions from the block of features, average over the spatial 5x5 spatial locations, using a tf.keras.layers.GlobalAveragePooling2D layer to convert the features to a single 1280-element vector per image.

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
```

3.8 Apply a tf.keras.layers.Dense layer to convert these features into a single prediction per image

```
prediction_layer = tf.keras.layers.Dense(4)
```

Build a model by chaining together the data augmentation, rescaling, base_model and feature extractor layers using the Keras Functional API. As previously mentioned, use training=False as our model contains a BatchNormalization layer.

```
inputs = tf.keras.Input(shape=(224, 224, 3))

x = data_augmentation(inputs)
x = rescale(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)

outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 224, 224, 3)]     0

 sequential (Sequential)     (None, 224, 224, 3)       0

 rescaling (Rescaling)       (None, 224, 224, 3)       0

 mobilenetv2_1.00_224 (Funct  (None, 7, 7, 1280)       2257984
 ional)

 global_average_pooling2d (G  (None, 1280)             0
 lobalAveragePooling2D)

 dropout (Dropout)           (None, 1280)              0

 dense (Dense)               (None, 4)                 5124

=================================================================
```

```
Total params: 2,263,108
Trainable params: 5,124
Non-trainable params: 2,257,984
_________________________________________________________________
```

3.9 Compile the model

Compile the model before training it.

```
base_learning_rate = 0.0001
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

model.summary()
```

3.10 Check the initial performance

```
initial_epochs = 10

loss0, accuracy0 = model.evaluate(val_ds)

print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))
```

```
7/7 [==============================] - 6s 634ms/step - loss: 5.4365 - accuracy: 0.3562
```

3.11 Train the model

```
history = model.fit(train_ds,
                    epochs=initial_epochs,
                    validation_data=val_ds)
```

```
Epoch 1/10
16/16 [==============================] - 19s 1s/step - loss: 5.7116 - accuracy: 0.3711 - val_loss: 5.0557
- val_accuracy: 0.4155
Epoch 2/10
16/16 [==============================] - 18s 1s/step - loss: 5.7344 - accuracy: 0.4531 - val_loss: 5.0238
- val_accuracy: 0.4886
Epoch 3/10
16/16 [==============================] - 17s 1s/step - loss: 5.2400 - accuracy: 0.4844 - val_loss: 4.7333
- val_accuracy: 0.4932
Epoch 4/10
16/16 [==============================] - 18s 1s/step - loss: 5.2432 - accuracy: 0.4336 - val_loss: 4.4295
- val_accuracy: 0.5068
Epoch 5/10
16/16 [==============================] - 18s 1s/step - loss: 5.3314 - accuracy: 0.4473 - val_loss: 4.0970
- val_accuracy: 0.5251
Epoch 6/10
16/16 [==============================] - 18s 1s/step - loss: 5.2111 - accuracy: 0.4863 - val_loss: 3.9137
- val_accuracy: 0.5160
Epoch 7/10
16/16 [==============================] - 19s 1s/step - loss: 4.4624 - accuracy: 0.5176 - val_loss: 3.9028
- val_accuracy: 0.5297
Epoch 8/10
16/16 [==============================] - 18s 1s/step - loss: 4.9377 - accuracy: 0.4902 - val_loss: 3.8271
- val_accuracy: 0.5479
Epoch 9/10
16/16 [==============================] - 18s 1s/step - loss: 4.5658 - accuracy: 0.5137 - val_loss: 3.6440
- val_accuracy: 0.5434
Epoch 10/10
16/16 [==============================] - 18s 1s/step - loss: 4.5670 - accuracy: 0.4883 - val_loss: 3.6344
- val_accuracy: 0.5525
```

3.12 Learning curves

Let's take a look at the learning curves of the training and validation accuracy/loss when using the MobileNetV2 base model as a fixed feature extractor.

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

plt.figure(figsize=(8, 5))
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.ylabel('Accuracy')
plt.ylim([0,1])
plt.title('Training and Validation Accuracy')
```

```
Text(0.5, 1.0, 'Training and Validation Accuracy')
```

4. Fine Tuning

In the feature extraction experiment, you were only training a few layers on top of an MobileNetV2 base model. The weights of the pre-trained network were not updated during training.

One way to increase performance even further is to train (or "fine-tune") the weights of the top layers of the pre-trained model alongside the training of the classifier you added. The training process will force the weights to be tuned from generic feature maps to features associated specifically with the dataset.

Also, you should try to fine-tune a small number of top layers rather than the whole MobileNet model. In most convolutional networks, the higher up a layer is, the more specialized it is. The first few layers learn very simple and generic features that generalize to almost all types of images. As you go higher up, the features are increasingly more specific to the dataset on which the model was trained. The goal of fine-tuning is to adapt these specialized features to work with the new dataset, rather than overwrite the generic learning.

4.1 Un-freeze the top layers of the model

All you need to do is unfreeze the base_model and set the bottom layers to be un-trainable. Then, you should recompile the model (necessary for these changes to take effect), and resume training.

```
base_model.trainable = True

# Let's take a look to see how many layers are in the base model
print("Number of layers in the base model: ", len(base_model.layers))

# Fine-tune from this layer onwards
fine_tune_at = 100

# Freeze all the layers before the `fine_tune_at` layer
for layer in base_model.layers[:fine_tune_at]:
    layer.trainable = False
```

```
Number of layers in the base model:  154
```

4.2 Compile the model

As you are training a much larger model and want to readapt the pretrained weights, it is important to use a lower learning rate at this stage. Otherwise, your model could overfit very quickly.

```
model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate/10),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(),
              metrics=['accuracy'])

model.summary()
```

4.3 Continue training the model

If you trained to convergence earlier, this step will improve your accuracy by a few percentage points.

```
fine_tune_epochs = 10
total_epochs =  initial_epochs + fine_tune_epochs

history_fine = model.fit(train_ds,
                         epochs=total_epochs,
                         initial_epoch=history.epoch[-1],
                         validation_data=val_ds)
```

```
Epoch 10/20
16/16 [==============================] - 28s 2s/step - loss: 3.9778 - accuracy: 0.5703 - val_loss: 3.0664
- val_accuracy: 0.5936
Epoch 11/20
16/16 [==============================] - 23s 1s/step - loss: 3.3941 - accuracy: 0.6016 - val_loss: 2.9292
- val_accuracy: 0.6073
Epoch 12/20
16/16 [==============================] - 23s 1s/step - loss: 3.2680 - accuracy: 0.5977 - val_loss: 2.5669
- val_accuracy: 0.5890
Epoch 13/20
16/16 [==============================] - 24s 1s/step - loss: 3.1989 - accuracy: 0.6172 - val_loss: 2.4456
- val_accuracy: 0.6164
Epoch 14/20
16/16 [==============================] - 24s 1s/step - loss: 2.8382 - accuracy: 0.6250 - val_loss: 2.4065
- val_accuracy: 0.6438
Epoch 15/20
16/16 [==============================] - 26s 2s/step - loss: 2.4659 - accuracy: 0.6016 - val_loss: 2.2495
- val_accuracy: 0.6393
Epoch 16/20
16/16 [==============================] - 26s 2s/step - loss: 2.3909 - accuracy: 0.6582 - val_loss: 2.0386
- val_accuracy: 0.6484
Epoch 17/20
16/16 [==============================] - 25s 2s/step - loss: 1.9048 - accuracy: 0.6113 - val_loss: 1.3595
- val_accuracy: 0.5479
Epoch 18/20
16/16 [==============================] - 24s 1s/step - loss: 1.3964 - accuracy: 0.5742 - val_loss: 1.1564
- val_accuracy: 0.4566
Epoch 19/20
16/16 [==============================] - 24s 2s/step - loss: 1.2094 - accuracy: 0.5547 - val_loss: 1.1204
- val_accuracy: 0.5068
Epoch 20/20
16/16 [==============================] - 25s 2s/step - loss: 1.3040 - accuracy: 0.5078 - val_loss: 1.1458
- val_accuracy: 0.5068
```

4.4 Append performance to previous results

```
acc += history_fine.history['accuracy']
val_acc += history_fine.history['val_accuracy']

plt.figure(figsize=(8, 5))
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.ylim([0, 1])
plt.plot([initial_epochs-1,initial_epochs-1],
          plt.ylim(), label='Start Fine Tuning')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
```

```
Text(0.5, 1.0, 'Training and Validation Accuracy')
```

Guidelines

CI/CD

Visit our CI/CD guidelines and instructions to learn how to setup your own local runner or to learn about the pipeline.

ML/Ops

Visit our MLOps guidelines to learn how to handle different stages of ML related project development. This includes theory, TensorFlow, Kubernetes and MLFlow.

Python in Practice

Visit our python guidelines to learn about PEP8, common pitfalls to avoid, docstrings and additional useful libraries.

Wiki & Documentation

Visit our wiki & documentation guidelines to learn how to use the confluence and what to consider when adding to our documentation.

Expand all Collapse all

CI/CD

CI/CD is essential to a successful software project of any kind. Using CI/CD helps us avoid time consuming bugs on our own branches and later on contaminating other branches.

In this wiki you will find the most essential topics of CI/CD you need to know.

Local Runner

CI/CD is a computational task. Therefore it needs a hardware to execute on. While the LDV offers the prospect of GPU runners in the future, for current developments, we still need to use our own resources. This can be done through CI/CD runners operating on your local machine. We try to have at least one runner up at any time, but in case none is available and you need to urgently move some code, you will find the instructions here.

GitLab Pipeline

Visit our guidelines on GitLab pipelines to learn more about the topic.

- Local Runner
- GitLab Pipeline

Local Runner

- Start and Stop your Local Runner:
- View Local Runner in GitLab:
- Set-Up
  - eduVPN
  - Runner for your OS
    - Windows
    - MacOS
    - Ubuntu
  - Config.toml

Start and Stop your Local Runner:

To start and stop an already installed local runner for GitLab, follow the following instructions:

1. Open the “Command prompt” **with administrative rights**
2. Switch to the directory:

```
cd C:\GitLab-Runner
```

3. Execute **gitlab-runner.exe start** to start the local runner

```
gitlab-runner.exe start
```

4. Make sure the eduVPN is connected
5. (for MacOS) Docker needs to be running on the runner host machine.

6. Execute **gitlab-runner.exe stop** to stop the local runner

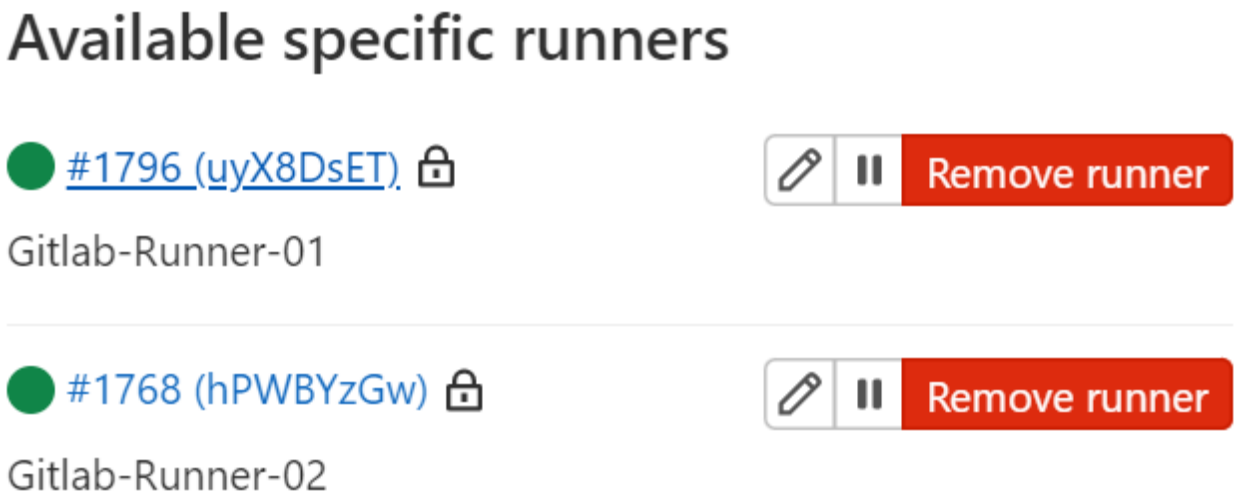
```
gitlab-runner.exe stop
```

View Local Runner in GitLab:

Note: Viewing available local runners through the GitLab UI is only possible with the role of a Maintainer!

To find a listing of available runners, follow the following steps:

1. Go to **Settings** > CI/CD
2. **Expand** the Section **Runners**
3. All local Runners are shown:



Set-Up

eduVPN

Before installing your local runner, make sure to install eduVPN in order to connect to the LDV servers. Download instruction can be found under the following link:

https://doku.lrz.de/display/PUBLIC/VPN+-+eduVPN+-+Installation+und+Konfiguration

Runner for your OS

- Windows
- MacOS
- Ubuntu
- config.toml

Windows

1. Create the folder “C:\GitLab-Runner”
2. **Download** the binary for your computer (64-bit or 32-bit) and put into the folder you created
3. Open the “Command prompt” **with administrative rights**
4. Direct to the directory: “cd C:\GitLab-Runner”
5. Execute

```
gitlab-runner.exe install
gitlab-runner.exe register
```

- 1st argument: https://gitlab.ldv.ei.tum.de/
- 2nd argument: GR13489412Q5cdUxzSPsikrdgkkU-
- 3rd argument: Gitlab-Runner-XX
- 4th argument: [optional] press Enter
- 5th argument: [optional] press Enter
- 6th argument: docker
- 7th argument: docker:latest

6. Open the config.toml file and change:
   - The **volumes** variable to match the code snippet below
   - The **privileged** variable to true
   - Do not copy the code snippet in its entirety, as some settings might be unique to your machine

MacOS

1. Download the binary for your system:
   - For **Intel-based** systems:

     ```
     sudo curl --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-darwin-amd64"
     ```

     For **Apple Silicon-based** systems:

     ```
     sudo curl --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-darwin-arm64"
     ```
2. Give it permissions to execute:
   - `sudo chmod +x /usr/local/bin/gitlab-runner`
3. Execute

   ```
   gitlab-runner register
   ```

   - 1st argument: https://gitlab.ldv.ei.tum.de/
   - 2nd argument: GR13489412Q5cdUxzSPsikrdgkkU-
   - 3rd argument: Gitlab-Runner-XX
   - 4th argument: [optional] press Enter
   - 5th argument: [optional] press Enter
   - 6th argument: docker
   - 7th argument: docker:latest
4. Open a terminal and switch to the current user.

   ```
   su - <username>
   ```

5. Install GitLab Runner as a service and start it:
   - `cd ~`
   - `gitlab-runner install`
   - `gitlab-runner start`
6. Open the config.toml file and change:
   - The **volumes** variable to match the code snippet below
   - The **privileged** variable to true
   - Do not copy the code snippet in its entirety, as some settings might be unique to your machine

Ubuntu

1. Download the binary for your system:
   - For **Intel-based** systems:

     ```
     sudo curl --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-darwin-amd64"
     ```

     For **Apple Silicon-based** systems:

```
sudo curl --output /usr/local/bin/gitlab-runner "https://gitlab-runner-downloads.s3.amazonaws.com/latest/binaries/gitlab-runner-darwin-arm64"
```

2. Give it permissions to execute:
   - `sudo chmod +x /usr/local/bin/gitlab-runner`
3. Execute

```
gitlab-runner register
```

   - 1st argument: https://gitlab.ldv.ei.tum.de/
   - 2nd argument: GR13489412Q5cdUxzSPsikrdgkkU-
   - 3rd argument: Gitlab-Runner-XX
   - 4th argument: [optional] press Enter
   - 5th argument: [optional] press Enter
   - 6th argument: docker
   - 7th argument: docker:latest
4. Open a terminal and switch to the current user.

```
su - <username>
```

5. Install GitLab Runner as a service and start it:
   - `cd ~`
   - `sudo gitlab-runner install --user <username>`
   - `sudo gitlab-runner start`
6. Open the config.toml file at the protected location /etc/gitlab-runner and change:
   - The **volumes** variable to match the code snippet below
   - The **privileged** variable to true
   - Do not copy the code snippet in its entirety, as some settings might be unique to your machine

Config.toml

This is an example toml file, showing you how to configure your runner.

```
• concurrent = 1
  check_interval = 0

  [session_server]
    session_timeout = 1800

  [[runners]]
    name = "Gitlab-Runner-21"
    url = "https://gitlab.ldv.ei.tum.de/"
    token = "sCfgLctsQAp37u1sQ_7C"
    executor = "docker"
    [runners.custom_build_dir]
    [runners.cache]
      [runners.cache.s3]
      [runners.cache.gcs]
      [runners.cache.azure]
    [runners.docker]
      tls_verify = false
      image = "docker:latest"
      privileged = true
```

```
    disable_entrypoint_overwrite = false
    oom_kill_disable = false
    disable_cache = false
    volumes = ["/var/run/docker.sock:/var/run/docker.sock","
/cache"]
    shm_size = 0
```

GitLab Pipeline

Pipelines are the top-level component of continous integration, delivery and deployment.

Pipelines comprise:

- Job, which define what to do. For example, jobs that compile or test code.
- Stages, which define when to run the jobs. For example, stages that run tests after stages that compile the code.

Jobs are executed by runners. Multiple jobs in the same stage are executed in parallel, if there are anough concurrent runners.

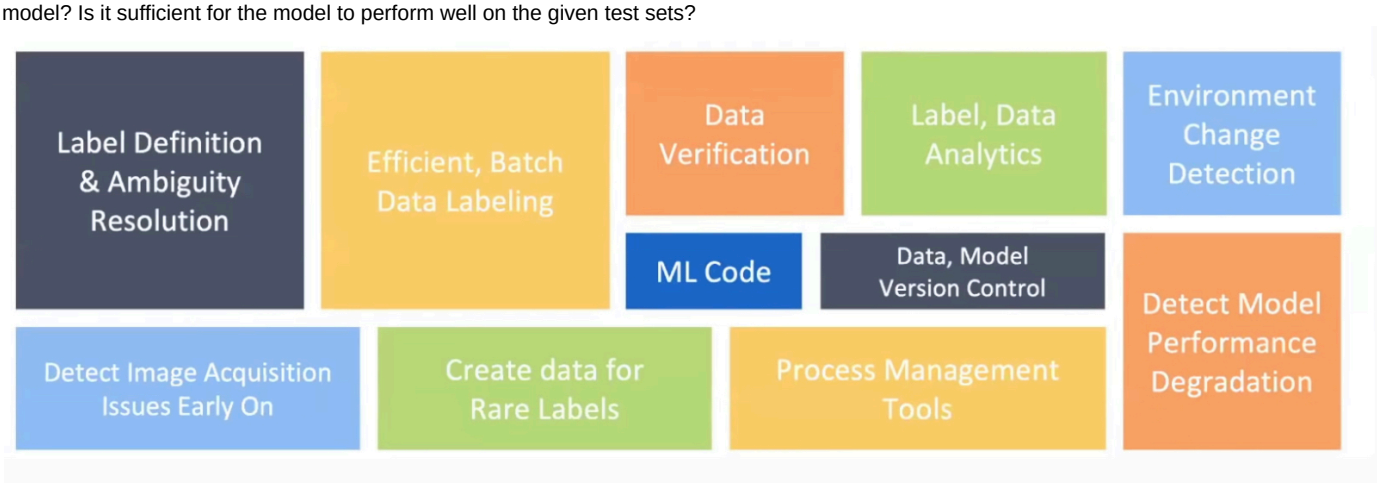
If all jobs in a stage succeed, the pipeline moves to the next stage.

If any job in a stage fails, the next stage is not executed and the pipeline ends early.

In gerneral, pipelines are executed automatically and require not intervention once created. However, there are also times when you can manually interact with the pipeline.

MLOps

How does the process of a machine learning project look like? Does the role of a machine learning practitioner end with the completion of the model? Is it sufficient for the model to perform well on the given test sets?



These and many other questions are to be answered in the following section. With this section we try to define a structure similar to DevOps with which we guide a ML project from start to finish. Topics in this section aim to identify flaws in these processes and to provide a best practice recommendation. In addition, publications and tutorials are provided for the individual topics, which provide a deeper understanding of the subject matter. This section is therefore useful for those who are already familiar with the project and are looking for answers to questions such as " how to handle skewed datasets" and many others.

Summary of our recommendations:

- Align labelling
- Start with rather simple, well documented and already used model as a baseline
- Focus equally on Data and Model

01 - Introduction

What is MLOps?

MLOps is a set of practices that aims to deploy and maintain machine learning models in production reliably and efficiently. The word is a compound of "machine learning" and the continuous development practice of DevOps in the software field. Machine learning models are tested and developed in isolated experimental systems. When an algorithm is ready to be launched, MLOps is practiced between Data Scientists,

DevOps, and Machine Learning engineers to transition the algorithm to production systems. Similar to DevOps or DataOps approaches, MLOps seeks to increase automation and improve the quality of production models, while also focusing on business and regulatory requirements. MLOps started as a set of best practices bit is slowly evolving into an independent approach to ML lifecycle management.

The ML Project Lifecycle:

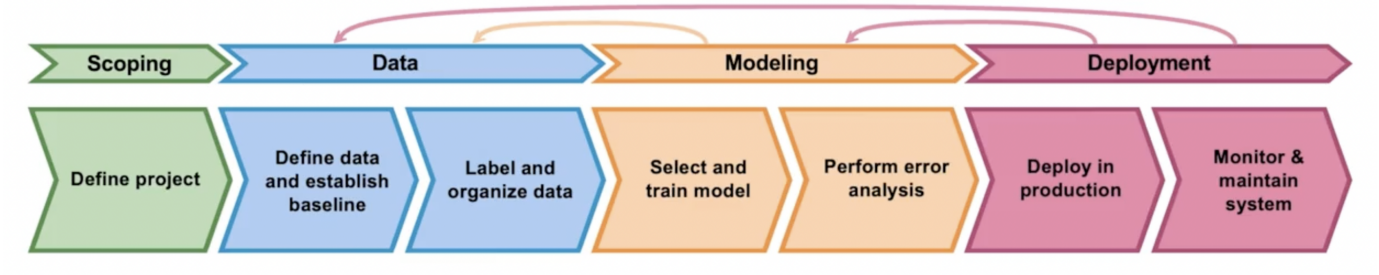
A machine learning project is basically composed of four phases: Scoping, Data, Modelling and Deployment.

During the **scoping** phase the general scope and goals of the project are defined.

In the **data** phase it is decided which data will be collected and how it shall be used. Additionally it will be decided on a constant labelling and organising of the data.

The next phase is the **modelling** phase. In this phase the model will be selected, trained, analysed and optimised. If necessary it is possible to go back to the data phase to either collect more data or refine the current labelling and organisation of the data. As soon as one is satisfied with the model performance during testing one can go further to deployment/ production and test the model in a "real-world" scenario.

The last phase is the **deployment** phase. It consists of the deployment itself as well as the monitoring and maintaining after the deployment. It may be necessary to go back to the Data or Modeling phase to refine or update the system, e.g. after a sensor changes and the data now looks different. As basically all software products, it is also necessary to update the ML algorithm on a regular basis to account for system/behavioural changes and to optimise the system.



MLOps Goals:

- faster experimentation and model development
- faster deployment of updated model into production (not that relevant for our project)
- Quality assurance

02 - Scoping

Notes:

- Data Centric vs. Model Centric
- General Recommendation: Focus on data and use model with "reasonable effort" e.g. use open source model for prototyping and refine if first test show promising start
- Software engineering issues (realtime or batch, cloud or edge, compute resources, latency and throughput, logging, security and privacy)
- Degrees of automation

03 - Data

Guidelines:

1. Consistent Labelling:
    a. Consistent labelling is important to achieve maximum performance of our model. As the data we received is mostly not labelled we should agree on consistent labelling

2. Data Changes: Be aware of changes in the data over time.
    a. Gradual Changes: AI in Fashion: Fashion Trends change over time, NLP: Language changes over time (slang, new words, etc.)
    b. Sudden Shock: With COVID a lot of consumers changed their online shopping behaviour and AIs had to be retrained
3. Drift:
    a. Concept Drift: Mapping from input to output changes (e.g.: detect more classes than before)
    b. Data Drift: Input changes (e.g.: NLP trained on adult voices, now a lot of children using the app, requires retraining)

04 - Model

05 - Deployment

Notes:

- first deployment vs maintenance
- Deployment cases
    - New: small amount of data then ramp it up
    - Until how: task performed manually, now add ML for assist/replacement
    - Replace previous ML
- Generally: Gradual ramp up to allow easy rollbacks
- Modes of deployment
    - Shadow mode deployment
    - Canary deployment
    - Blue green deployment
- Dashboards for monitoring (start by brainstorming everything that can go wrong, define metric for each case and create dashboard)
    - Software Metrics
    - Input Metric (e.g.: fraction of missing data)
    - Output Metric

06 - Kubernetes and Kubeflow

Both of these concepts go with each other. To understand Kubeflow, one needs to have a good grasp of what Kubernetes is since Kubeflow builds on top of Kubernetes' infrastructure.

Kubernetes

Kubernetes is a portable, extensible, open-source platform for managing, or rather, orchestrating containerized workloads and services.

The need for a platform like Kubernetes arose due to the fact that modern software is shifting from a monolithic running entity to a collection of independent microservices, where each microservice runs in a separate container. Kubernetes is useful in this case because it turns physical or virtual hosts (servers) into a platform that:

- Hosts containerized services, providing them with compute, storage, and network resources
- Provides automatic management of large numbers of containerized applications, running on hundreds, or even thousands of containers.

Use case of Kubernetes

Kubernetes is useful when the project that is developed follows a multi-container architecture. The developers plan out every aspect of this cluster of containers, namely:

- How they fit and work together
- How many of each component should run
- What happens when challenges arise, e.g a lot of users are logging in at once

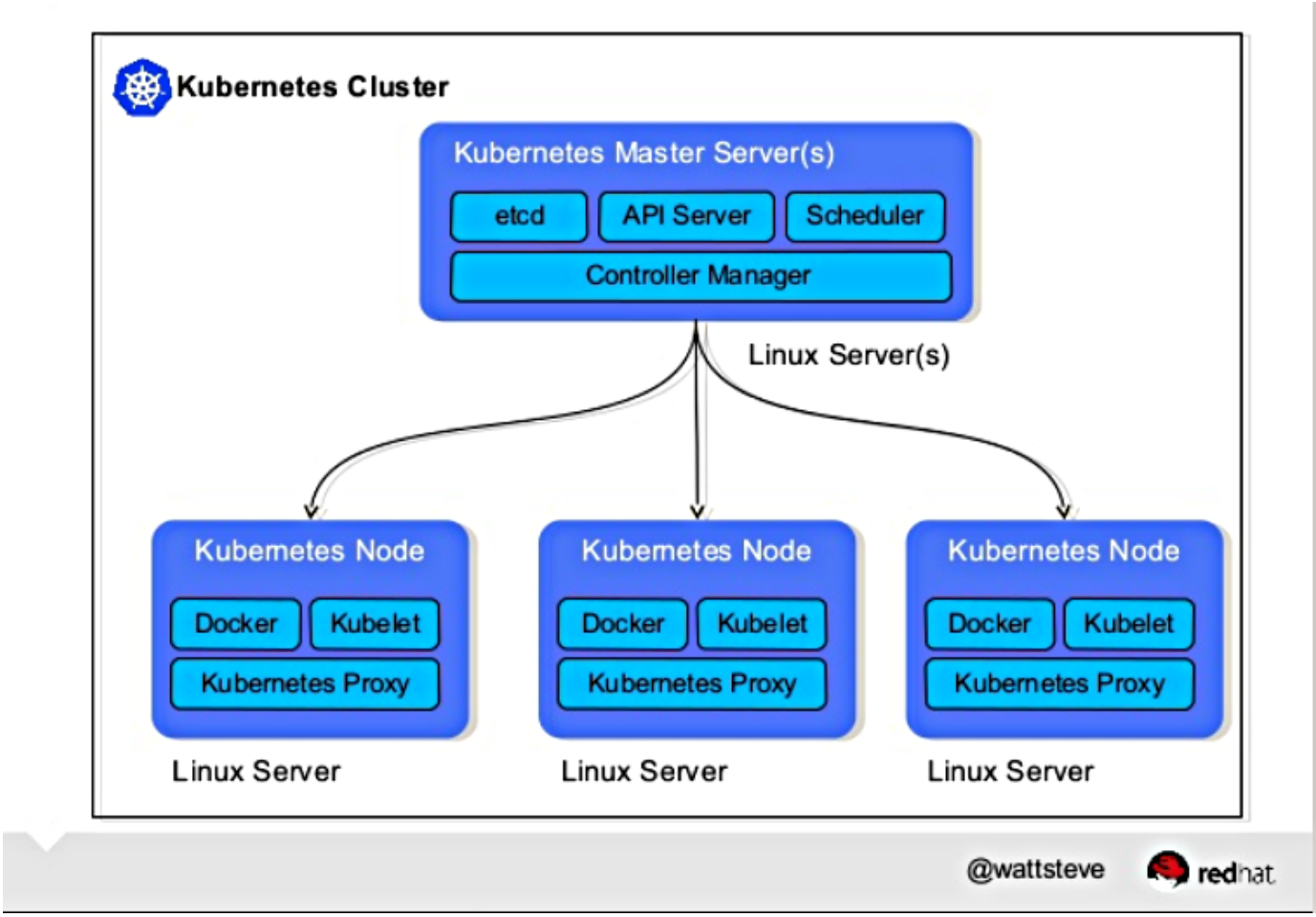
The collection of containers for the project is stored in a local or remote container registry.

The behavior of the collection of containers is captured in text files as configurations.

Kubernetes does the job of evaluating, implementing, and maintaining this configuration until told otherwise.

Kubernetes cluster

Kubernetes architecture comprises a cluster of nodes hosting the project.



There is one master node (also called the control plane), which itself runs the API server, the scheduler, the controller manager, and etcd store, which is a highly available store for shared configuration and service discovery.

For the rest of the nodes, each one hosts part of the distributed application. This is done by leveraging docker or similar container technology. Furthermore, the nodes run two additional pieces of software, Kubernetes Proxy, which gives access to the running application. and Kubelet, which receives commands from the control plane.

Kubernetes runtime environment

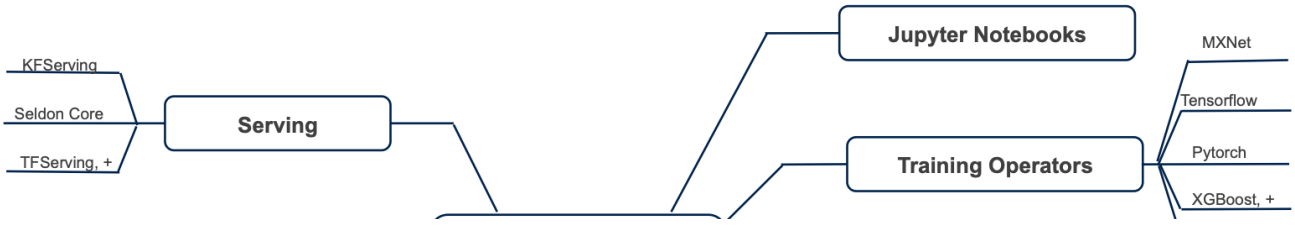
Kubernetes also runs almost anywhere, on a wide range of Linux operating systems (worker nodes can also run on Windows Server). A single Kubernetes cluster can span hundreds of bare-metal or virtual machines in a data center, private, or any public cloud. Kubernetes can also run on developer desktops, edge servers, microservers like Raspberry Pis, or very small mobile and IoT devices and appliances.

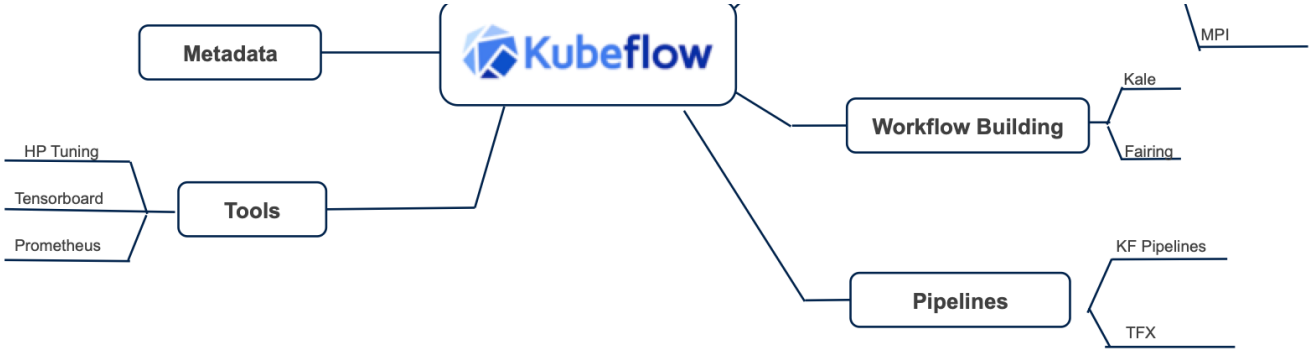
Kubeflow

Kubeflow is a central dashboard that provides a platform for data scientists and engineers to leverage Kubernetes to develop, deploy and monitor their models in production.

Motivation to use Kubeflow

it can be difficult to manage machine learning applications, platforms, and resource considerations. ML apps have a different footprint than other deployments like web and mobile. For example, if we consider resource allocation, it can be difficult not to either assign too little or too much processing power because an ML training phase is compute-intensive, while the inference phase is lightweight. Existing solutions that are great in other domains don't capture the nuances needed for ML.





Kubeflow focuses on providing a standard for deploying enterprise ML apps. this standard is build around three principles:

Composability

Kubeflow provides flexibility in choosing what is needed in any particular ML project. It makes sure that the steps that are part of the pipeline are independent and configurable depending on the specifics of the machine learning frameworks and libraries.

Portability

Kubeflow handles the specifics of the platform to enable the developer to focus on the data and model.

the runtime environment can range from a laptop to a training rig or cloud server.

Scalability

Scalability means that the project can use more resources when they are needed, and release some if they are not. Every environment can have different computing resources, like CPUs, TPUs, and GPUs. Kubeflow leverages Kubernetes to optimize resource use and minimize manual scaling effort.

Machine learning stages with Kubeflow

there are two main stages when developing a machine learning project.

experimental phase

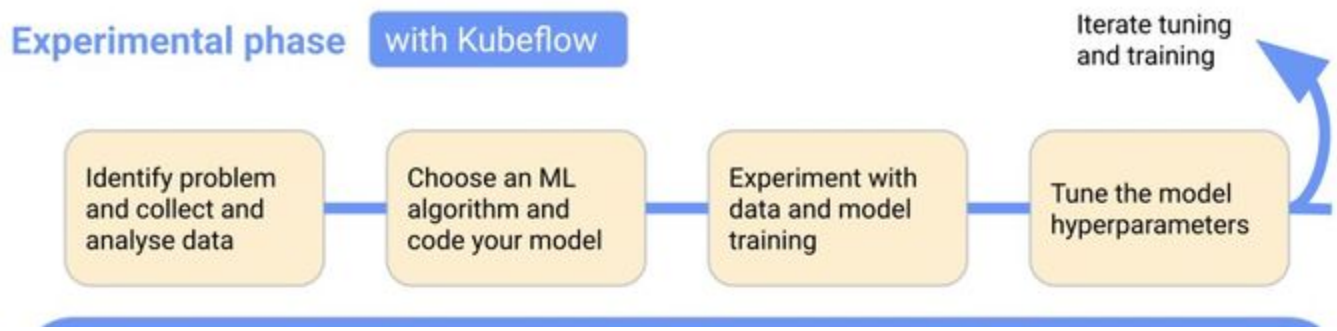
In the experimental phase, you develop your model based on initial assumptions, and test and update the model iteratively to produce the results you're looking for:

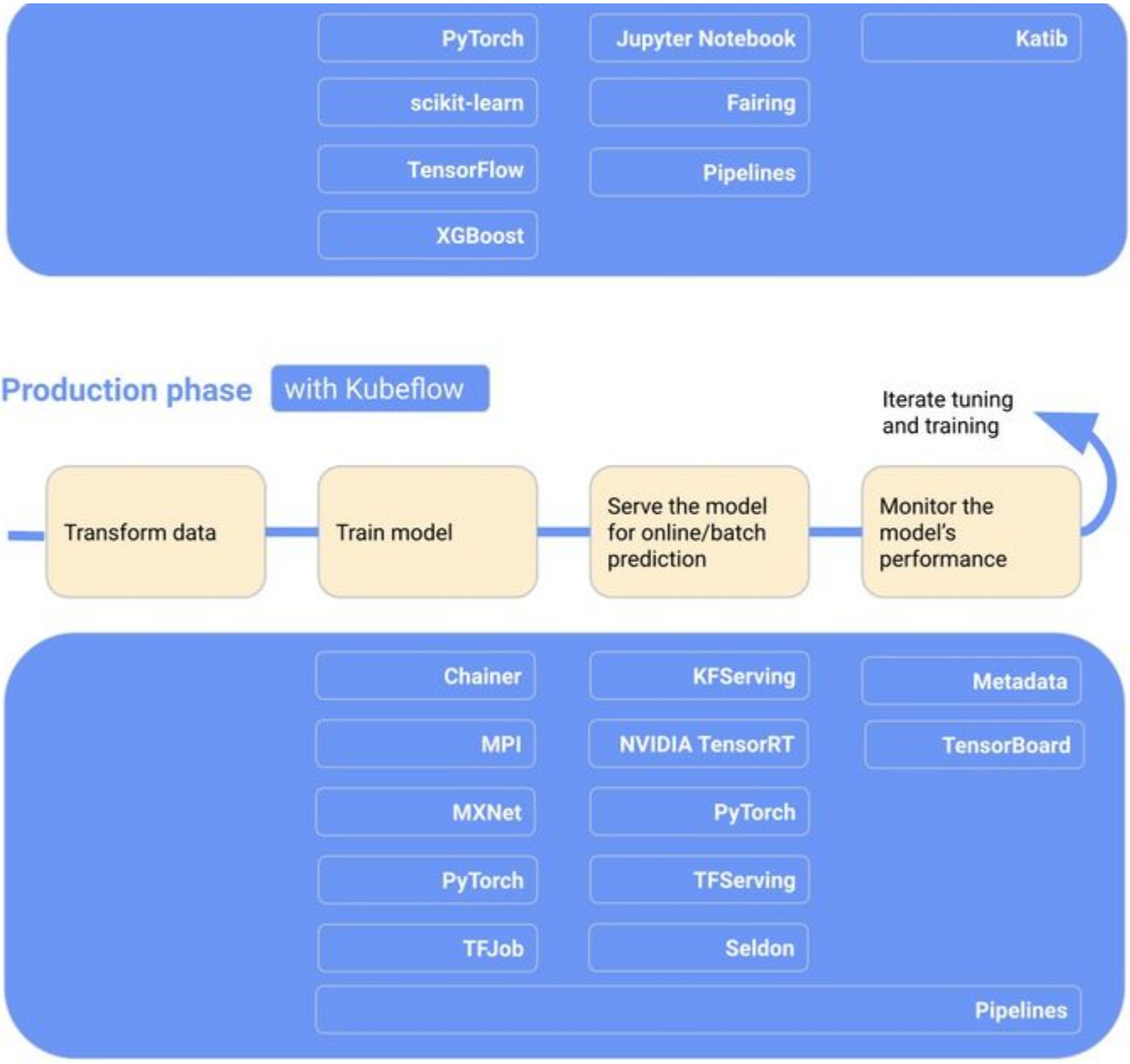
- Identify the problem you want the ML system to solve.
- Collect and analyze the data you need to train your ML model.
- Choose an ML framework and algorithm, and code the initial version of your model.
- Experiment with the data and with training your model.
- Tune the model hyperparameters to ensure the most efficient processing and the most accurate results possible.

Production phase

In the production phase, you deploy a system that performs the following processes:

- Transform the data into the format that your training system needs. To ensure that your model behaves consistently during training and prediction, the transformation process must be the same in the experimental and production phases.
- Train the ML model.
- Serve the model for online prediction or for running in batch mode.
- Monitor the model's performance, and feed the results into your processes for tuning or retraining the model.





Kubeflow includes services for spawning and managing Jupyter notebooks. Use notebooks for interactive data science and experimenting with ML workflows. Furthermore, Kubeflow Pipelines is a platform for building, deploying and managing multi-step ML workflows based on Docker containers.

Kubeflow offers several components that you can use to build your ML training, hyperparameter tuning, and serving workloads across multiple platforms.

Set-up

Windows:

Minikube:

1. Download and run the installer for the latest release.
2. Add the `minikube.exe` binary to your PATH.
   Make sure to run PowerShell as Administrator.

```
$oldPath = [Environment]::GetEnvironmentVariable('Path',
[EnvironmentVariableTarget]::Machine)
if ($oldPath.Split(';') -inotcontains 'C:\minikube'){ `
  [Environment]::SetEnvironmentVariable('Path', $('{0};C:
\minikube' -f $oldPath), [EnvironmentVariableTarget]::Machine) `
}
```

3. Reopen PowerShell
4. Type `minikube` to see all command options for minikube.

Kubernetes:

1. Download the latest release v1.24.0.
2. Download the `kubectl` checksum file:

```
curl -LO "https://dl.k8s.io/v1.24.0/bin/windows/amd64/kubectl.exe.
sha256"
```

3. Validate the `kubectl` binary against the checksum file:
   Using PowerShell to automate the verification using the `-eq` operator to get a `True` or `False` result:

```
$($(CertUtil -hashfile .\kubectl.exe SHA256)[1] -replace " ", "") -
eq $(type .\kubectl.exe.sha256)
```

4. Append or prepend the `kubectl` binary folder to your `PATH` environment variable.
5. Test to ensure the version of `kubectl` is the same as downloaded:

```
kubectl version --client
```

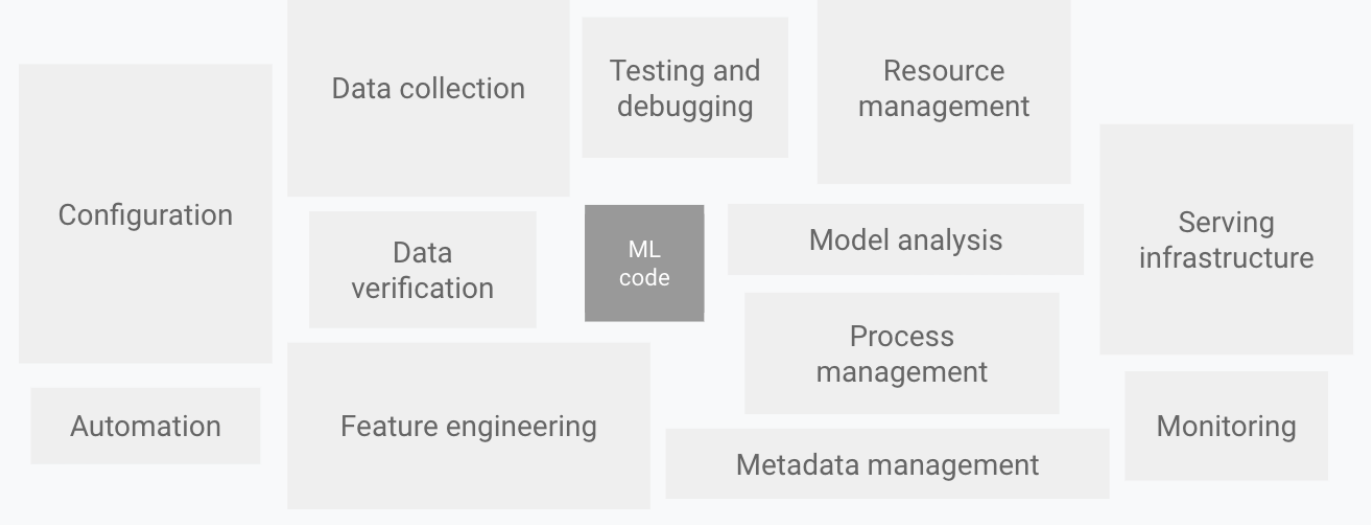
Get started (with kubectl):

1. Start minikube with `minikube start`
2. Get status of minikube with `minikube status`
3. Get status of…
   … nodes with `kubectl get nodes`
   … pods with `kubectl get pod`
   … services with `kubectl get services`
4. Create pod:

```
kubectl create deployment <docker-image-name> --image=<docker-
image>
```

07 - TensorFlow Extended (TFX)

There is much more to the machine learning software than the actual learning algorithm. Often, the algorithm makes up only a fraction of the entire code. A pipeline is used to organize the infrastructure around the ML code in a well-structured manner.



When you're ready to move your models from research to production, use TFX to create and manage a production pipeline.

Short description	Source	Responsible person
Starting Point for TFX	https://www.tensorflow.org/tfx	
Introduction videos for TFX	https://www.youtube.com/watch?v=Mxk4qmO_1B4	
YouTube Tutorial on TFX Workflow and corresponding Notebook	https://www.youtube.com/watch?v=VrBoQCchJQU&t=1826s https://colab.research.google.com/gist/rafiqhasan/2164304ede002f4a8bfe56e5434e1a34/dl-e2e-taxi-dataset-tfx-e2e.ipynb	@ Linyan Yang
Talk on TFX Serving	https://www.youtube.com/watch?v=4mqFDwIdKh0&t=281s	@ Linyan Yang
TFX: ML Metadata	https://www.tensorflow.org/tfx/guide/mlmd?hl=en	
	https://cloud.google.com/architecture/architecture-for-mlops-using-tfx-kubeflow-pipelines-and-cloud-build	

08 - MLFlow

Python in Practice

This space offers our team the possibility to gather and exchange best practices, tips, tricks, tools and setups for the development in python. Explore the space and get familiar with the standard of Group4.

PEP8

Get to know the PEP8 style convention, which will keep our code neat, tidy and readable throughout the project. PEP8 has been endorsed by the python community and is frequently used in corporate setting. Show your future employer that you know how to make code look like poetry!

Docstrings

As a team, we settled for the aesthetic and functional Google docstring style. This will keep our code documentation clean and up-to-date. Check out our docstring templates and learn how to compose your own!

Later PEP's, Tips & Tricks

Style is not everything! Learn how to avoid common python mistakes and improve your coding with libraries and best practices after PEP8. Did you know that there are over 590 PEPs?

- PEP 8

- Docstrings
- Later PEP's, Tips & Tricks

!Disclaimer! Most content has been repurposed from https://peps.python.org/ and https://realpython.com/ !Disclaimer!

PEP 8

Before starting to write code any python code, one should be familiar with the PEP 8 style convention. Compiled by the creators of python and endorsed by the python community, this is the official python style guide. While the PEP8 manifest is a lengthy document, we will have a look at the most important rules and practices.

Naming Conventions

Learn how to correctly name variables, functions and other items to keep your code readable and avoid cluttering.

Code Layout

Learn how to layout your code to highlight the functioning and segmentation of your functional blocks. Python does not use braces but whitespaces can do!

Line Length, Breaking & Indentation

Do you like using split screen for coding? Let's make the split screen experience a dream by keeping our lines short. Line breaks are beautiful… but only with the right indentation!

Comments

Everybody wants to know what you are doing? Tell them with comments! Learn how to use comments while avoiding to clutter.

Whitespaces in Expressions and Statements

You have a difficult time reading a statement? Maybe whitespaces can fix it! Learn how to highlight line functionality using whitespaces.

Programming Recommendation

Learn how to avoid the most commited errors with the standard library.

- Naming Conventions
- Code Layout
- Line Length, Breaking & Indentation
- Comments
- Whitespaces in Expressions and Statements
- Programming Recommendation

> *Beautiful is better than ugly.*
> *Explicit is better than implicit.*
> *Simple is better than complex.*
> *Complex is better than complicated.*
> *Flat is better than nested.*
> *Sparse is better than dense.*
> *Readability counts.*
> *Special cases aren't special enough to break the rules.*
> *Although practicality beats purity.*
> *Errors should never pass silently.*
> *Unless explicitly silenced.*
> *In the face of ambiguity, refuse the temptation to guess.*
> *There should be one-- and preferably only one --obvious way to do it.*
> *Although that way may not be obvious at first unless you're Dutch.*
> *Now is better than never.*
> *Although never is often better than* right *now.*
> *If the implementation is hard to explain, it's a bad idea.*
> *If the implementation is easy to explain, it may be a good idea.*
> *Namespaces are one honking great idea -- let's do more of those!*
>
> — The Zen of Python

Naming Conventions

> *"Explicit is better than implicit."*
>
> — The Zen of Python

When you write Python code, you have to name a lot of things: variables, functions, classes, packages, and so on. Choosing sensible names will save you time and energy later. You'll be able to figure out, from the name, what a certain variable, function, or class represents. You'll also avoid using inappropriate names that might result in errors that are difficult to debug.

Naming Styles

The table below outlines some of the common naming styles in Python code and when you should use them:

Type	Naming Convention	Examples
Function	Use a lowercase word or words. Separate words by underscores to improve readability.	`function`, `my_function`
Variable	Use a lowercase single letter, word, or words. Separate words with underscores to improve readability.	`x`, `var`, `my_variable`
Class	Start each word with a capital letter. Do not separate words with underscores. This style is called camel case.	`Model`, `MyClass`
Method	Use a lowercase word or words. Separate words with underscores to improve readability.	`class_method`, `method`
Constant	Use an uppercase single letter, word, or words. Separate words with underscores to improve readability.	`CONSTANT`, `MY_CONSTANT`, `MY_LONG_CONSTANT`
Module	Use a short, lowercase word or words. Separate words with underscores to improve readability.	`module.py`, `my_module.py`
Package	Use a short, lowercase word or words. Do not separate words with underscores.	`package`, `mypackage`

Note: Never use l, O, or I single letter names as these can be mistaken for 1 and 0, depending on typeface:

In addition to choosing the correct naming styles in your code, you also have to choose the names carefully. Below are a few pointers on how to do this as effectively as possible.

How to Choose Names

Choosing names for your variables, functions, classes, and so forth can be challenging. You should put a fair amount of thought into your naming choices when writing code as it will make your code more readable. The best way to name your objects in Python is to use descriptive names to make it clear what the object represents.

When naming variables, you may be tempted to choose simple, single-letter lowercase names, like x. But, unless you're using x as the argument of a mathematical function, it's not clear what x represents.

Example:

```
>>> # Not recommended
>>> x = 'John Smith'
>>> y, z = x.split()
>>> print(z, y, sep=', ')
'Smith, John'
```

This will work, but you'll have to keep track of what x, y, and z represent. It may also be confusing for collaborators. A much clearer choice of names would be something like this:

```
>>> # Recommended
>>> name = 'John Smith'
>>> first_name, last_name = name.split()
>>> print(last_name, first_name, sep=', ')
'Smith, John'
```

One is able to jump into the code at the thrid line and still understand what is being done.

Similarly, to reduce the amount of typing you do, it can be tempting to use abbreviations when choosing names.

Example:

```
# Not recommended
def db(x):
    return x * 2
```

At first glance, this could seem like a sensible choice. db() could easily be an abbreviation for double. However, this naming practice will force a team member to look at the definition of the function, loosing valuable time.

The following example is much clearer.

```
# Recommended
def multiply_by_two(x):
    return x * 2
```

The same philosophy applies to all other data types and objects in Python. Always try to use the most concise but descriptive names possible.

Code Layout

> *“Beautiful is better than ugly.”*
>
> — The Zen of Python

How you lay out your code has a huge role in how readable it is.

Blank Lines

Vertical whitespace, or blank lines, can greatly improve the readability of your code. Code that’s bunched up together can be overwhelming and hard to read. Similarly, too many blank lines in your code makes it look very sparse, and the reader might need to scroll more than necessary.

Surround top-level functions and classes with two blank lines. Top-level functions and classes should be fairly self-contained and handle separate functionality. It makes sense to put extra vertical space around them, so that it’s clear they are separate:

```
class MyFirstClass:
    pass


class MySecondClass:
    pass


def top_level_function():
    return None
```

Surround method definitions inside classes with a single blank line. Inside a class, functions are all related to one another. It’s good practice to leave only a single line between them:

```
class MyClass:
    def first_method(self):
        return None

    def second_method(self):
        return None
```

Use blank lines sparingly inside functions to show clear steps. Sometimes, a complicated function has to complete several steps before the return statement. To help the reader understand the logic inside the function, it can be helpful to leave a blank line between each step.

In the example below, there is a function to calculate the variance of a list. This is two-step problem, indicated by the white space as separation. Additionally, we add a white space before the return value, to make it clearer what is being returned.

```
def calculate_variance(number_list):
    sum_list = 0
    for number in number_list:
        sum_list = sum_list + number
    mean = sum_list / len(number_list)

    sum_squares = 0
    for number in number_list:
        sum_squares = sum_squares + number**2
    mean_squares = sum_squares / len(number_list)

    return mean_squares - mean**2
```

Line Length, Breaking & Indentation

> *“There should be one—and preferably only one—obvious way to do it.”*
>
> — The Zen of Python

Line Length and Line Breaks

PEP 8 suggests lines should be limited to 79 characters. This is because it allows you to have multiple files open next to one another, while also avoiding line wrapping.

Of course, keeping statements to 79 characters or less is not always possible. PEP 8 outlines ways to allow statements to run over several lines.

Python will assume line continuation if code is contained within parentheses, brackets, or braces:

```
def function(arg_one, arg_two,
             arg_three, arg_four):
    return arg_one
```

If it is impossible to use implied continuation, then you can use backslashes to break lines instead:

```
from mypkg import example1, \
    example2, example3
```

However, if you can use implied continuation, then you should do so.

If line breaking needs to occur around binary operators, like + and *, it should occur before the operator. This rule stems from mathematics.

```
# Recommended
total = (first_variable
```

```
          + second_variable
          - third_variable)

# Not Recommended
total = (first_variable +
          second_variable -
          third_variable)
```

Indentation

Indentation, or leading whitespace, is extremely important in Python. The indentation level of lines of code in Python determines how statements are grouped together.

The key indentation rules laid out by PEP 8 are the following:

- Use 4 consecutive spaces to indicate indentation.
- Prefer spaces over tabs.

Tabs vs. Spaces

As mentioned above, you should use spaces instead of tabs when indenting code. You can adjust the settings in your text editor to output 4 spaces instead of a tab character, when you press the Tab key.

Python 3 does not allow mixing of tabs and spaces. Therefore, if you are using Python 3, then these errors are issued automatically:

```
$ python3 code.py
  File "code.py", line 3
    print(i, j)
               ^
TabError: inconsistent use of tabs and spaces in indentation
```

PEP 8 recommends that you always use 4 consecutive spaces to indicate indentation.

Indentation Following Line Breaks

When you're using line continuations to keep lines to under 79 characters, it is useful to use indentation to improve readability. It allows the reader to distinguish between two lines of code and a single line of code that spans two lines.

Align the indented block with the opening delimiter:

```
def function(arg_one, arg_two,
             arg_three, arg_four):
    return arg_one
```

Sometimes you can find that only 4 spaces are needed to align with the opening delimiter. This will often occur in if statements that span multiple lines as the if, space, and opening bracket make up 4 characters. In this case, it can be difficult to determine where the nested code block inside the if statement begins:

```
x = 5
if (x > 3 and
    x < 10):
    print(x)
```

In this case, PEP 8 provides two alternatives to help improve readability:

```
# Add a comment after the final condition. Due to syntax highlighting
in most editors, this will separate the conditions from the nested code:

x = 5
if (x > 3 and
    x < 10):
    # Both conditions satisfied
    print(x)

Add extra indentation on the line continuation:

x = 5
if (x > 3 and
        x < 10):
    print(x)
```

An alternative style of indentation following a line break, frequently used to declare sequences, is a hanging indent. This is a typographical term meaning that every line but the first in a paragraph or statement is indented.

```
list_of_numbers = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
]
```

Where to Put the Closing Brace

Line continuations allow you to break lines inside parentheses, brackets, or braces. It’s easy to forget about the closing brace, but it’s important to put it somewhere sensible. Otherwise, it can confuse the reader. PEP 8 provides two options for the position of the closing brace in implied line continuations:

Line up the closing brace with the first non-whitespace character of the previous line:

```
list_of_numbers = [
    1, 2, 3,
    4, 5, 6,
    7, 8, 9
    ]
```

Line up the closing brace with the first character of the line that starts the construct:

```
list_of_numbers = [
    1, 2, 3,
```

```
    4, 5, 6,
    7, 8, 9
]
```

Comments

> *“If the implementation is hard to explain, it’s a bad idea.”*
>
> — The Zen of Python

You should use comments to document code as it’s written. It is important to document your code so that you, and any collaborators, can understand it. When you or someone else reads a comment, they should be able to easily understand the code the comment applies to and how it fits in with the rest of your code.

Here are some key points to remember when adding comments to your code:

- Limit the line length of comments and docstrings to 72 characters.
- Use complete sentences, starting with a capital letter.
- Make sure to update comments if you change your code.

Block Comments

Use block comments to document a small section of code. They are useful when you have to write several lines of code to perform a single action, such as importing data from a file or updating a database entry.

PEP 8 provides the following rules for writing block comments:

- Indent block comments to the same level as the code they describe.
- Start each line with a # followed by a single space.
- Separate paragraphs by a line containing a single #.

Here is a block comment explaining the function of a for loop. Note that the sentence wraps to a new line to preserve the 79 character line limit:

```
for i in range(0, 10):
    # Loop over i ten times and print out the value of i, followed by a
    # new line character
    print(i, '\n')
```

Sometimes, if the code is very technical, then it is necessary to use more than one paragraph in a block comment:

```
def quadratic(a, b, c, x):
    # Calculate the solution to a quadratic equation using the quadratic
    # formula.
    #
    # There are always two solutions to a quadratic equation, x_1 and
x_2.
    x_1 = (- b+(b**2-4*a*c)**(1/2)) / (2*a)
    x_2 = (- b-(b**2-4*a*c)**(1/2)) / (2*a)
    return x_1, x_2
```

If you’re ever in doubt as to what comment type is suitable, then block comments are often the way to go. Use them as much as possible throughout your code, but make sure to update them if you make changes to your code!

Inline Comments

Inline comments explain a single statement in a piece of code. They are useful to remind you, or explain to others, why a certain line of code is necessary. Here’s what PEP 8 has to say about them:

- Use inline comments sparingly.
- Write inline comments on the same line as the statement they refer to.
- Separate inline comments by two or more spaces from the statement.
- Start inline comments with a # and a single space, like block comments.
- Don't use them to explain the obvious.

Below is an example of an inline comment:

```
x = 5  # This is an inline comment
```

Sometimes, inline comments can seem necessary, but you can use better naming conventions instead. Here's an example of two equivalent statements:

```
# Not recomended
x = 'John Smith'  # Student Name

# Recomended
student_name = 'John Smith'
```

Finally, inline comments such as these are bad practice as they state the obvious and clutter code:

```
empty_list = []  # Initialize empty list

x = 5
x = x * 5  # Multiply x by 5
```

Inline comments are more specific than block comments, and it's easy to add them when they're not necessary, which leads to clutter. You could get away with only using block comments so, unless you are sure you need an inline comment, your code is more likely to be PEP 8 compliant if you stick to block comments.

Whitespaces in Expressions and Statements

> *"Sparse is better than dense."*
>
> — The Zen of Python

Whitespace can be very helpful in expressions and statements when used properly. If there is not enough whitespace, then code can be difficult to read, as it's all bunched together. If there's too much whitespace, then it can be difficult to visually combine related terms in a statement.
Whitespace Around Binary Operators

Adding Whitespaces

Surround the following binary operators with a single space on either side outside of parameter assignments:

- Assignment operators (=, +=, -=, and so forth)
- Comparisons (==, !=, >, <. >=, <=) and (is, is not, in, not in)
- Booleans (and, not, or)

```
# Recommended
def function(default_parameter=5):
    # ...
```

```
# Not recommended
def function(default_parameter = 5):
    # ...
```

When there's more than one operator in a statement, adding a single space before and after each operator can look confusing. Instead, it is better to only add whitespace around the operators with the lowest priority, especially when performing mathematical manipulation. Here are a couple examples:

```
# Recommended
y = x**2 + 5
z = (x+y) * (x-y)

# Not Recommended
y = x ** 2 + 5
z = (x + y) * (x - y)
```

You can also apply this to if statements where there are multiple conditions:

```
# Recommended
if x>5 and x%2==0:
    print('x is larger than 5 and divisible by 2!')

# Not recommended
if x > 5 and x % 2 == 0:
    print('x is larger than 5 and divisible by 2!')
```

No matter how you do it, you must use the same amount of whitespace either side of the operator.

The following is not acceptable:

```
# Definitely do not do this!
if x >5 and x% 2== 0:
    print('x is larger than 5 and divisible by 2!')
```

In slices, colons act as a binary operators. Therefore, the rules outlined in the previous section apply, and there should be the same amount of whitespace either side. The following examples of list slices are valid:

```
list[3:4]

# Treat the colon as the operator with lowest priority
list[x+1 : x+2]

# In an extended slice, both colons must be
# surrounded by the same amount of whitespace
list[3:4:5]
list[x+1 : x+2 : x+3]
```

```
# The space is omitted if a slice parameter is omitted
list[x+1 : x+2 :]
```

In summary, you should surround most operators with whitespace. However, there are some caveats to this rule, such as in function arguments or when you're combining multiple operators in one statement.

When to Avoid Adding Whitespace

In some cases, adding whitespace can make code harder to read. Too much whitespace can make code overly sparse and difficult to follow. PEP 8 outlines very clear examples where whitespace is inappropriate.

The most important place to avoid adding whitespace is at the end of a line. This is known as trailing whitespace. It is invisible and can produce errors that are difficult to trace.

The following list outlines some cases where you should avoid adding whitespace:

1. Immediately inside parentheses, brackets, or braces:

```
# Recommended
my_list = [1, 2, 3]

# Not recommended
my_list = [ 1, 2, 3, ]

Before a comma, semicolon, or colon:

x = 5
y = 6

# Recommended
print(x, y)

# Not recommended
print(x , y)
```

2. Before the open parenthesis that starts the argument list of a function call:

```
def double(x):
    retured
tuple = (1,)

# Not recommended
tuple = (1, )
```

3. To align assignment operators:

```
# Recommended
var1 = 5
var2 = 6
```

```
some_long_var = 7

# Not recommended
var1          = 5
var2          = 6
some_long_var = 7

# Not recommended
list [3]
```

4. Between a trailing comma and a closing parenthesis:

```
# Recommended
tuple = (1,)

# Not recommended
tuple = (1, )
```

5. To align assignment operators:

```
# Recommended
var1 = 5
var2 = 6
some_long_var = 7

# Not recommended
var1          = 5
var2          = 6
some_long_var = 7
```

Make sure that there is no trailing whitespace anywhere in your code.

Programming Recommendation

> *“Simple is better than complex.”*
>
> — The Zen of Python

Do not compare to True and False

Don’t compare Boolean values to True or False using the equivalence operator. bool can only take values True or False. It is enough to check the boolean value through it if statement:

```
# Not recommended
my_bool = 6 > 5
if my_bool == True:
    return '6 is bigger than 5'

# Recommended
```

```
if my_bool:
    return '6 is bigger than 5'
```

Empty sequences falsy

Use the fact that empty sequences are falsy in if statements. If you want to check whether a list is empty, you might be tempted to check the length of the list. If the list is empty, it's length is 0 which is equivalent to False when used in an if statement. However, in Python any empty list, string, or tuple is falsy. We can therefore come up with a more efficient implementation:

```
# Not recommended
my_list = []
if not len(my_list):
    print('List is empty!')

# Recommended
my_list = []
if not my_list:
    print('List is empty!')
```

is not rather than not ... is

Use use the former rather than the latter in if statements. If you are trying to check whether a variable has a defined value, there are two options. The first is to evaluate an if statement with x is not None, as in the example below:

```
# Recommended
if x is not None:
    return 'x exists!'

# Not recommended
if not x is None:
    return 'x exists!'
```

While both options will be evaluated correctly, the first is simpler, so PEP 8 encourages it.

Not being None is not being True

Don't use if x: when you mean if x is not None:. Sometimes, you may have a function with arguments that are None by default. A common mistake when checking if such an argument, arg, has been given a different value is to use the following:

```
# Not Recommended
if arg:
    # Do something with arg...
```

This code checks that arg is truthy. Instead, you want to check that arg is not None, so it would be better to use the following:

```
```

```
# Recommended
if arg is not None:
    # Do something with arg...
```

The mistake being made here is assuming that not None and truthy are equivalent. You could have set arg = []. As we saw above, empty lists are evaluated as falsy in Python. So, even though the argument arg has been assigned, the condition is not met, and so the code in the body of the if statement will not be executed.

Check preffixes and suffixes

Use .startswith() and .endswith() instead of slicing. If you were trying to check if a string word was prefixed, or suffixed, with the word cat, it might seem sensible to use list slicing. However, list slicing is prone to error, and you have to hardcode the number of characters in the prefix or suffix. It is also not clear to someone less familiar with Python list slicing what you are trying to achieve:

```
# Not recommended
if word[:3] == 'cat':
    print('The word starts with "cat"')

# Recommended
if word.startswith('cat'):
    print('The word starts with "cat"')
```

Similarly, the same principle applies when you're checking for suffixes.

Docstrings

Together, we chose to use the Google docstring style. Let's have a look at this docstring style together!

- Module Docstring
- Function Docstring
- Module Function Docstring
- Generator Docstring
- Exceptions
- Classes

Module Docstring

This type of docstring is located at the head of a module entry point. It allows to explain the use case and gives brief examples. When flying over the documentation, this type of docstring will be a major vantage point.

```
# -*- coding: utf-8 -*-
"""Example Google style docstrings.

This module demonstrates documentation as specified by the `Google
Python
Style Guide`_. Docstrings may extend over multiple lines. Sections are
created
with a section header and a colon followed by a block of indented text.

Example:
    Examples can be given using either the ``Example`` or ``Examples``
    sections. Sections support any reStructuredText formatting,
including
    literal blocks::
```

```
        $ python example_google.py

Section breaks are created by resuming unindented text. Section breaks
are also implicitly created anytime a new section starts.

Attributes:
    module_level_variable1 (int): Module level variables may be
documented in
        either the ``Attributes`` section of the module docstring, or
in an
        inline docstring immediately following the variable.

        Either form is acceptable, but the two should not be mixed.
Choose
        one convention to document module level variables and be
consistent
        with it.

Todo:
    * For module TODOs
    * You have to also use ``sphinx.ext.todo`` extension

"""
```

Function Docstring

The function docstring should briefly summarize the use case of a function and the nature of of inputs and return values.

```
def function_with_pep484_type_annotations(param1: int, param2: str) ->
bool:
    """Example function with PEP 484 type annotations.

    Args:
        param1: The first parameter.
        param2: The second parameter.

    Returns:
        The return value. True for success, False otherwise.

    """
```

Module Function Docstring

A modul function is a function which is intended for third party use outside of the module. The documentation should therefore be more lengthy to facilitate the understanding to individuals who are not familiar with the modules internal operation.

```
def module_level_function(param1: bool, param2=None: bool, *args,
```

```
**kwargs) -> bool:
    """This is an example of a module level function.

    Function parameters should be documented in the ``Args`` section.
The name
    of each parameter is required. The type and description of each
parameter
    is optional, but should be included if not obvious.

    If \*args or \*\*kwargs are accepted,
    they should be listed as ``*args`` and ``**kwargs``.

    The format for a parameter is::

        name (type): description
            The description may span multiple lines. Following
            lines should be indented. The "(type)" is optional.

            Multiple paragraphs are supported in parameter
            descriptions.

    Args:
        param1 (int): The first parameter.
        param2 (:obj:`str`, optional): The second parameter. Defaults
to None.
            Second line of description should be indented.
        *args: Variable length argument list.
        **kwargs: Arbitrary keyword arguments.

    Returns:
        bool: True if successful, False otherwise.

        The return type is optional and may be specified at the
beginning of
        the ``Returns`` section followed by a colon.

        The ``Returns`` section may span multiple lines and paragraphs.
        Following lines should be indented to match the first line.

        The ``Returns`` section supports any reStructuredText
formatting,
        including literal blocks::

            {
                'param1': param1,
                'param2': param2
            }

    Raises:
        AttributeError: The ``Raises`` section is a list of all
```

```
exceptions
            that are relevant to the interface.
        ValueError: If `param2` is equal to `param1`.

    """
    if param1 == param2:
        raise ValueError('param1 may not be equal to param2')
    return True
```

Generator Docstring

The functional difference a function and a generator is refelceted in their respective docstring. The generator docstring explains the yielded values.

```
def example_generator(n: int) -> int:
    """Generators have a ``Yields`` section instead of a ``Returns``
section.

    Args:
        n (int): The upper limit of the range to generate, from 0 to
`n` - 1.

    Yields:
        int: The next number in the range of 0 to `n` - 1.

    Examples:
        Examples should be written in doctest format, and should
illustrate how
        to use the function.

        >>> print([i for i in example_generator(4)])
        [0, 1, 2, 3]

    """
    for i in range(n):
        yield i
```

Exceptions

```
class ExampleError(Exception):
    """Exceptions are documented in the same way as classes.

    The __init__ method may be documented in either the class level
    docstring, or as a docstring on the __init__ method itself.

    Either form is acceptable, but the two should not be mixed. Choose
one
```

```
    convention to document the __init__ method and be consistent with
it.

    Note:
        Do not include the `self` parameter in the ``Args`` section.

    Args:
        msg (str): Human readable string describing the exception.
        code (:obj:`int`, optional): Error code.

    Attributes:
        msg (str): Human readable string describing the exception.
        code (int): Exception error code.

    """

    def __init__(self, msg, code):
        self.msg = msg
        self.code = code
```

Classes

```
class ExampleClass(object):
    """The summary line for a class docstring should fit on one line.

    If the class has public attributes, they may be documented here
    in an ``Attributes`` section and follow the same formatting as a
    function's ``Args`` section. Alternatively, attributes may be
documented
    inline with the attribute's declaration (see __init__ method below).

    Properties created with the ``@property`` decorator should be
documented
    in the property's getter method.

    Attributes:
        attr1 (str): Description of `attr1`.
        attr2 (:obj:`int`, optional): Description of `attr2`.

    """

    def __init__(self, param1: str, param2: int, param3: bool) -> None:
        """Example of docstring on the __init__ method.

        The __init__ method may be documented in either the class level
        docstring, or as a docstring on the __init__ method itself.

        Either form is acceptable, but the two should not be mixed.
```

```
Choose one
        convention to document the __init__ method and be consistent
with it.

        Note:
            Do not include the `self` parameter in the ``Args`` section.

        Args:
            param1: Description of `param1`.
            param2: Description of `param2`. Multiple
                lines are supported.
            param3: Description of `param3`.

        """
        self.attr1 = param1
        self.attr2 = param2
        self.attr3 = param3  #: Doc comment *inline* with attribute

        #: list of str: Doc comment *before* attribute, with type
specified
        self.attr4 = ['attr4']

        self.attr5 = None
        """str: Docstring *after* attribute, with type specified."""

    @property
    def readonly_property(self) -> None:
        """str: Properties should be documented in their getter method.

        Returns:
            Will always return smth lmao

        """
        return 'readonly_property'

    @property
    def readwrite_property(self) -> list:
        """:obj:`list` of :obj:`str`: Properties with both a getter and
setter
        should only be documented in their getter method.

        If the setter method contains notable behavior, it should be
        mentioned here.

        """

        return ['readwrite_property']

    def example_method(self, param1: int, param2: str) -> bool:
        """Class methods are similar to regular functions.
```

```
        Note:
            Do not include the `self` parameter in the ``Args`` section.

        Args:
            param1: The first parameter.
            param2: The second parameter.

        Returns:
            True if successful, False otherwise.

        """
        return True

    def __special__(self):
        """By default special members with docstrings are not
included."""
        pass


    def _private(self):
        """By default private members are not included.

        Private members are any methods or attributes that start with an
        underscore and are *not* special. By default they are not
included
        in the output.


        """
        pass

    def _private_without_docstring(self):
        pass
```

Later PEP's, Tips & Tricks

In this space you will find additional best practices form later PEPs and the python standard library, which will improve you python experience and code robustness. Have a look around!

Pathlib

Ever noticed how cluttered the os module is? Ever struggled handling paths in python? The pathlib has your back, whit API and object oriented AND intuitive implementations!

List Comprehension

Are your nested for loops slow? Did you which you could write more compact code? Reduce your line usage and increase your execution parallelism with list comprehensions!

Handling Files

Ever wondered why closing files is important? Already used a context manager? No? Learn about file handling!

cProfile

Why is my code so slow? Lets find out using the time module? Don't waste your time on time! Use the cProfiler to learn why your code is under performing in depth and generate cute graphics.

Development Mode

It works! But is it safe? Learn how to improve your code robustness using pythons in-build development mode.

- Pathlib
- List Comprehension
- Handling Files
- cProfile – How to profile your python code
- Development Mode
- Interfaces

Pathlib

- Construction
- Representing
- Properties
- Deriving new paths
    - Joining
    - Making the path relative
    - Sequence-like access
    - Parents
- Querying
- File metadata
- Directory walking
- File opening

With its publication and subsequent endorsment by the python comunity in the PEP 411, the pathlib module became the defacto standard for handling paths with python3. Contrary to its predecessor os.path, which is a submodule to the clutterd os standard library, is uses an object oriented approach to path handeling. Let's briefly discuss its some of the modules features.

```
>>> p = Path('/home/antoine/pathlib/setup.py')
>>> p.name
'setup.py'
>>> p.suffix
'.py'
>>> p.root
'/'
>>> p.parts
('/', 'home', 'antoine', 'pathlib', 'setup.py')
>>> p.relative_to('/home/antoine')
PosixPath('pathlib/setup.py')
>>> p.exists()
True
```

Construction

We will present construction and joining together since they expose similar semantics.

The simplest way to construct a path is to pass it its string representation:

```
>>> PurePath('setup.py')
PurePosixPath('setup.py')
```

Extraneous path separators and "." components are eliminated and additional arguments are automatically joined:

```
>>> PurePath('a///b/c/./d/')
PurePosixPath('a/b/c/d')
>>> PurePath('docs', 'Makefile')
PurePosixPath('docs/Makefile')
```

Joining semantics are similar to os.path.join, in that anchored paths ignore the information from the previously joined components:

```
>>> PurePath('/etc', '/usr', 'bin')
PurePosixPath('/usr/bin')
```

However, with Windows paths, the drive is retained as necessary:

```
>>> PureWindowsPath('c:/foo', '/Windows')
PureWindowsPath('c:/Windows')
>>> PureWindowsPath('c:/foo', 'd:')
PureWindowsPath('d:')
```

Also, path separators are normalized to the platform default:

```
>>> PureWindowsPath('a/b') == PureWindowsPath('a\\b')
True
```

The classmethod cwd() creates a path object pointing to the current working directory in absolute form:

```
>>> Path.cwd()
PosixPath('/home/antoine/pathlib')
```

Representing

To represent a path (e.g. to pass it to third-party libraries), just call str() on it:

```
>>> p = PurePath('/home/antoine/pathlib/setup.py')
>>> str(p)
'/home/antoine/pathlib/setup.py'
>>> p = PureWindowsPath('c:/windows')
>>> str(p)
'c:\\\\windows'
```

Properties

Several simple properties are provided on every path (each can be empty):

```
>>> p = PureWindowsPath('c:/Downloads/pathlib.tar.gz')
>>> p.drive
'c:'
>>> p.root
'\\'
>>> p.anchor
'c:\\\\'
>>> p.name
'pathlib.tar.gz'
>>> p.stem
'pathlib.tar'
>>> p.suffix
'.gz'
>>> p.suffixes
['.tar', '.gz']
```

Deriving new paths

Joining

A path can be joined with another using the / operator:

```
>>> p = PurePosixPath('foo')
>>> p / 'bar'
PurePosixPath('foo/bar')
>>> p / PurePosixPath('bar')
PurePosixPath('foo/bar')
>>> 'bar' / p
PurePosixPath('bar/foo')
```

As with the constructor, multiple path components can be specified, either collapsed or separately:

```
>>> p / 'bar/xyzzy'
PurePosixPath('foo/bar/xyzzy')
>>> p / 'bar' / 'xyzzy'
PurePosixPath('foo/bar/xyzzy')
```

A joinpath() method is also provided, with the same behaviour:

```
>>> p.joinpath('Python')
PurePosixPath('foo/Python')
```

Changing the path's final component
The with_name() method returns a new path, with the name changed:

```
>>> p = PureWindowsPath('c:/Downloads/pathlib.tar.gz')
>>> p.with_name('setup.py')
PureWindowsPath('c:/Downloads/setup.py')
```

The with_suffix() method returns a new path with the suffix changed. However, if the path has no suffix, the new suffix is added:

```
>>> p = PureWindowsPath('c:/Downloads/pathlib.tar.gz')
>>> p.with_suffix('.bz2')
PureWindowsPath('c:/Downloads/pathlib.tar.bz2')
>>> p = PureWindowsPath('README')
>>> p.with_suffix('.bz2')
PureWindowsPath('README.bz2')
```

Making the path relative

The relative_to() method computes the relative difference of a path to another:

```
>>> PurePosixPath('/usr/bin/python').relative_to('/usr')
PurePosixPath('bin/python')
```

Sequence-like access

The parts property returns a tuple providing read-only sequence access to a path's components:

```
>>> p = PurePosixPath('/etc/init.d')
>>> p.parts
('/', 'etc', 'init.d')
```

Parents

The parent property returns the logical parent of the path:

```
>>> p = PureWindowsPath('c:/python33/bin/python.exe')
>>> p.parent
PureWindowsPath('c:/python33/bin')
```

The parents property returns an immutable sequence of the path's logical ancestors:

```
>>> p = PureWindowsPath('c:/python33/bin/python.exe')
>>> len(p.parents)
3
>>> p.parents[0]
PureWindowsPath('c:/python33/bin')
```

```
>>> p.parents[1]
PureWindowsPath('c:/python33')
>>> p.parents[2]
PureWindowsPath('c:/')
```

Querying

match() matches the path against a glob pattern. It operates on individual parts and matches from the right:

```
>>> p = PurePosixPath('/usr/bin')
>>> p.match('/usr/b*')
True
>>> p.match('usr/b*')
True
>>> p.match('b*')
True
>>> p.match('/u*')
False
```

File metadata

The stat() returns the file's stat() result; similarly, lstat() returns the file's lstat() result (which is different iff the file is a symbolic link):

```
>>> p.stat()
posix.stat_result(st_mode=33277, st_ino=7483155, st_dev=2053,
st_nlink=1, st_uid=500, st_gid=500, st_size=928, st_atime=1343597970,
st_mtime=1328287308, st_ctime=1343597964)
```

Higher-level methods help examine the kind of the file:

```
>>> p.exists()
True
>>> p.is_file()
True
>>> p.is_dir()
False
>>> p.is_symlink()
False
>>> p.is_socket()
False
>>> p.is_fifo()
False
>>> p.is_block_device()
False
>>> p.is_char_device()
False
```

Directory walking

Simple (non-recursive) directory access is done by calling the iterdir() method, which returns an iterator over the child paths:

```
>>> p = Path('docs')
>>> for child in p.iterdir(): child
...
PosixPath('docs/conf.py')
PosixPath('docs/_templates')
PosixPath('docs/make.bat')
PosixPath('docs/index.rst')
PosixPath('docs/_build')
PosixPath('docs/_static')
PosixPath('docs/Makefile')
```

This allows simple filtering through list comprehensions:

```
>>> p = Path('.')
>>> [child for child in p.iterdir() if child.is_dir()]
[PosixPath('.hg'), PosixPath('docs'), PosixPath('dist'), PosixPath
('__pycache__'), PosixPath('build')]
```

Simple and recursive globbing is also provided:

```
>>> for child in p.glob('**/*.py'): child
...
PosixPath('test_pathlib.py')
PosixPath('setup.py')
PosixPath('pathlib.py')
PosixPath('docs/conf.py')
PosixPath('build/lib/pathlib.py')
```

File opening

The open() method provides a file opening API similar to the builtin open() method:

```
>>> p = Path('setup.py')
>>> with p.open() as f: f.readline()
...
'#!/usr/bin/env python3\n'
```

List Comprehension

As proposed with PEP 202, list comprehension supports nesting and if/else statements. Since the notation is more concise it is preferred by PEP8. Additionally the python backend tends to parallelize list comprehension, since none of the iterations depend on each other. This behaviour is similar to what is expected of the map() function.

It is recommended to use list comprehension wherever possible. Lets have a look at the syntax with the help a few use cases of list comprehension.

Unpacking Generators

```
>>> print [i for i in range(10)]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

List comprehension with if statement

```
>>> print [i for i in range(20) if i%2 == 0]
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18]
```

Nesting List Comprehension

```
>>> nums = [1, 2, 3, 4]
>>> fruit = ["Apples", "Peaches", "Pears", "Bananas"]
>>> print [(i, f) for i in nums
           for f in fruit]
[(1, 'Apples'), (1, 'Peaches'), (1, 'Pears'), (1, 'Bananas'),
 (2, 'Apples'), (2, 'Peaches'), (2, 'Pears'), (2, 'Bananas'),
 (3, 'Apples'), (3, 'Peaches'), (3, 'Pears'), (3, 'Bananas'),
 (4, 'Apples'), (4, 'Peaches'), (4, 'Pears'), (4, 'Bananas')]
```

A line break after the inner loop allows for better readability.

We can add if statements in the same fashion for nested list comprehension. Adding line breaks keeps the complicated code readable.

```
>>> print [(i, f) for i in nums
           for f in fruit
           if f[0] == "P"]
[(1, 'Peaches'), (1, 'Pears'),
 (2, 'Peaches'), (2, 'Pears'),
 (3, 'Peaches'), (3, 'Pears'),
 (4, 'Peaches'), (4, 'Pears')]
>>> print [(i, f) for i in nums
           for f in fruit
           if f[0] == "P"
           if i%2 == 1]
[(1, 'Peaches'), (1, 'Pears'), (3, 'Peaches'), (3, 'Pears')]
```

```
>>> print [i for i in zip(nums, fruit)
           if i[0]%2==0]
[(2, 'Peaches'), (4, 'Bananas')]
```

Handling Files

We will briefly have a look at best proctice for opening files and the risks of not closing files.

When opening a file in python, the interpreter makes a systme call to the OS, requesting a file handle. Using this file handle, the interpreter will perform all subsequent calls, including the request to close the file.

When opening a file, we will want to use a context mangaer.

```
with Path("hello.txt").open("w") as file:
    file.write("Hello, World!")
```

The context manager will handle the file operations towards the operating system and make sure, that the file is being closed even if the process crashes. This prevents us from losing data or reaching the open file limit during runtime.

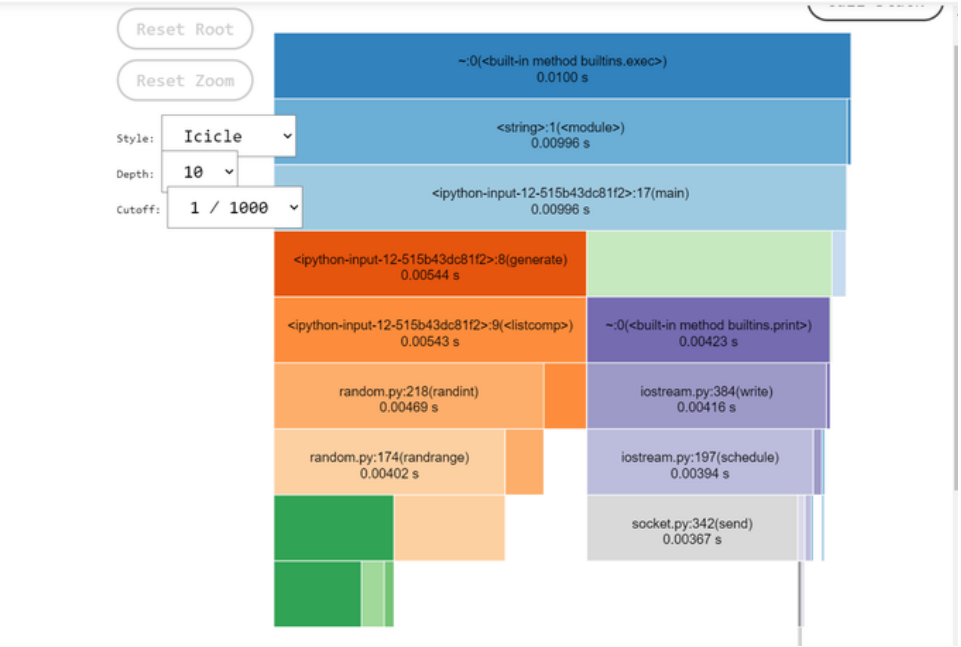
Only operations which rely on the file being open should be handled in the context manager.

Another equivalent implementation can be achiebed with the try...finally syntax. The finally block executes no mater if the try operation executes or not.

```
try:
    file = open("hello.txt", mode="w")
    file.write("Hello, World!")
finally:
    file.close()
```

cProfile – How to profile your python code

Reducing code runtime is important for developers, especially in machin learning. Python Profilers, like cProfile helps to find which part of the program or code takes more time to run. Lets have a look at the process of using cProfile module for extracting profiling data, using the pstats module to report it and snakeviz for visualization.



1. Why do we need Python Profilers ?

The main aspect while writing any code, especially when deploying, is that it should consume the lowest computational time and cost.

For our case, an operation executed in a data generator may be executed a few million times during the project runtime. Cumulatively, event the smallest difference in runtime can end up making an impact on our model selection process performance. However there is also a monetary aspect to runtime.
This is especially important when you run code on cloud services like AWS, Google Cloud or Azure, where there is a defined cost associated with the usage of computing resources. If you have two pieces of code that give the same result, the one that takes the least time and resource is usually chosen.

Let's say you have an algorithm that takes a lot of time to run. And you want to reduce the code run time. The first question that might crop up is:

Why does my code take so long to run?

Python Profilers can answer that question. It tells you which part of the code took how long to run. This lets you focus on that particular part and achieve efficiency.

2. Introduction to cProfile

cProfile is a built-in python module that can perform profiling. It is the most commonly used profiler currently.

What can cProfile do for me?

- It gives you the total run time taken by the entire code.
- It also shows the time taken by each individual step. This allows you to compare and find which parts need optimization
- cProfile module also tells the number of times certain functions are being called.
- The data inferred can be exported easily using pstats module.
- The data can be visualized nicely using snakeviz module. Examples come later in this post.

3. How to use cProfile ?

cProfile provides a simple run() function which is sufficient for most cases. The syntax is cProfile.run(statement, filename=None, sort=-1).

You can pass python code or a function name that you want to profile as a string to the statement argument.

If you want to save the output in a file, it can be passed to the filename argument. The sort argument can be used to specify how the output has to be printed. By default, it is set to -1(no value).

Let's call cProfile.run() on a simple operation.

```
import numpy as np
cProfile.run("20+10")
```

Output:

```
3 function calls in 0.000 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000    0.000    0.000 <string>:1(<module>)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.
exec}
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
```

Above you passed a simple addition code as a statement to the run() function of cProfile. Let's understand the output.

- Line no.1: shows the number of function calls and the time it took to run.

- Line no.2: Ordered by: standard name means that the text string in the far right column was used to sort the output. This could be changed by the sort parameter.
- Line no. 3 onwards contain the functions and sub functions called internally. Let's see what each column in the table means.
  - ncalls : Shows the number of calls made
  - tottime: Total time taken by the given function. Note that the time made in calls to sub-functions are excluded.
  - percall: Total time / No of calls. (remainder is left out)
  - cumtime: Unlike tottime, this includes time spent in this and all subfunctions that the higher-level function calls. It is most useful and is accurate for recursive functions.

You could see that it isn't very complex because the operation we did is simple.

4. Profiling a function that calls other functions

Now let's try profiling on a code that calls other functions. In this case, you can pass the call to main() function as a string to cProfile.run() function.

```
# Code containing multiple dunctions
def create_array():
  arr=[]
  for i in range(0,400000):
    arr.append(i)

def print_statement():
  print('Array created successfully')


def main():
  create_array()
  print_statement()


if __name__ == '__main__':
    cProfile.run('main()')
```

Output:

```
Array created successfully
         400041 function calls in 0.091 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.004    0.004    0.091    0.091 <ipython-input-10-
4dd6137cfe06>:12(main)
        1    0.059    0.059    0.087    0.087 <ipython-input-10-
4dd6137cfe06>:3(create_array)
        1    0.000    0.000    0.000    0.000 <ipython-input-10-
4dd6137cfe06>:8(print_statement)
        1    0.000    0.000    0.091    0.091 <string>:1(<module>)
        3    0.000    0.000    0.000    0.000 iostream.py:195(schedule)
        2    0.000    0.000    0.000    0.000 iostream.py:307
(_is_master_process)
        2    0.000    0.000    0.000    0.000 iostream.py:320
```

```
(_schedule_flush)
        2    0.000    0.000    0.000    0.000 iostream.py:382(write)
        3    0.000    0.000    0.000    0.000 iostream.py:93
(_event_pipe)
        3    0.000    0.000    0.000    0.000 socket.py:357(send)
        3    0.000    0.000    0.000    0.000 threading.py:1062
(_wait_for_tstate_lock)
        3    0.000    0.000    0.000    0.000 threading.py:1104
(is_alive)
        3    0.000    0.000    0.000    0.000 threading.py:506(is_set)
        1    0.000    0.000    0.091    0.091 {built-in method builtins.
exec}
        2    0.000    0.000    0.000    0.000 {built-in method builtins.
isinstance}
        1    0.000    0.000    0.000    0.000 {built-in method builtins.
print}
        2    0.000    0.000    0.000    0.000 {built-in method posix.
getpid}
        3    0.000    0.000    0.000    0.000 {method 'acquire' of
'_thread.lock' objects}
        3    0.000    0.000    0.000    0.000 {method 'append' of
'collections.deque' objects}
   400000    0.028    0.000    0.028    0.000 {method 'append' of
'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
```

Observe the above output. The function loacted within our small module have as filename <ipython-input-10-4dd6137cfe06>. The other functions are located deeper within the python backend

5. How to use Profile class of cProfile

What is the need for Profile class when you can simply do a run()?

Even though the run() function of cProfile may be enough in some cases, there are certain other methods that are useful as well. The Profile() class of cProfile gives you more precise control. Let's see a simple example.

By default, cProfile sorts its output by "standard name". This means that it sorts by the filename(far right column). If you think of it, it's actually not so useful, especially for complex functions.

How to use Profile to modify reports?

If your aim is to find the time-consuming parts, it would be helpful to sort the outputs as per ncalls. To do this,

1. First, initialize an instance of Profile class.
2. After that, call the enable() method of the profiler to start collecting profiling data.
3. After that, call the function you want to profile.
4. To stop collecting profiling data, call the disable() method.

How to report the data collected ?

The pstats module can be used to manipulate the results collected by the profiler object. First, create an instance of the stats class using pstats. Stats. Next, use the Stats class to create a statistics object from a profile object through stats= pstats.Stats(profiler).Now, to sort the output by ncalls, use the sort_stats() method as shown below. Finally to print the output, call the function print_statss() of stats object.

```
# How to use Profile class of cProfile
def create_array():
```

```
  arr=[]
  for i in range(0,400000):
    arr.append(i)

def print_statement():
  print('Array created successfully')


def main():
  create_array()
  print_statement()

if __name__ == '__main__':
    import cProfile, pstats
    profiler = cProfile.Profile()
    profiler.enable()
    main()
    profiler.disable()
    stats = pstats.Stats(profiler).sort_stats('ncalls')
    stats.print_stats()
```

Output:

```
Array created successfully
         400039 function calls in 0.094 seconds

   Ordered by: call count

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
   400000    0.034    0.000    0.034    0.000 {method 'append' of
'list' objects}
        3    0.000    0.000    0.000    0.000 {method 'acquire' of
'_thread.lock' objects}
        3    0.000    0.000    0.000    0.000 {method 'append' of
'collections.deque' objects}
        3    0.000    0.000    0.000    0.000 /usr/local/lib/python3.6
/dist-packages/ipykernel/iostream.py:93(_event_pipe)
        3    0.000    0.000    0.000    0.000 /usr/local/lib/python3.6
/dist-packages/ipykernel/iostream.py:195(schedule)
        3    0.000    0.000    0.000    0.000 /usr/local/lib/python3.6
/dist-packages/zmq/sugar/socket.py:357(send)
        3    0.000    0.000    0.000    0.000 /usr/lib/python3.6
/threading.py:1104(is_alive)
        3    0.000    0.000    0.000    0.000 /usr/lib/python3.6
/threading.py:506(is_set)
        3    0.000    0.000    0.000    0.000 /usr/lib/python3.6
/threading.py:1062(_wait_for_tstate_lock)
        2    0.000    0.000    0.000    0.000 {built-in method posix.
getpid}
```

```
        2    0.000    0.000    0.000    0.000 {built-in method builtins.
isinstance}
        2    0.000    0.000    0.000    0.000 /usr/local/lib/python3.6
/dist-packages/ipykernel/iostream.py:307(_is_master_process)
        2    0.000    0.000    0.000    0.000 /usr/local/lib/python3.6
/dist-packages/ipykernel/iostream.py:320(_schedule_flush)
        2    0.000    0.000    0.000    0.000 /usr/local/lib/python3.6
/dist-packages/ipykernel/iostream.py:382(write)
        1    0.000    0.000    0.000    0.000 {built-in method builtins.
print}
        1    0.000    0.000    0.000    0.000 <ipython-input-1-
66b56f7cc511>:6(print_statement)
        1    0.004    0.004    0.094    0.094 <ipython-input-1-
66b56f7cc511>:10(main)
        1    0.055    0.055    0.090    0.090 <ipython-input-1-
66b56f7cc511>:1(create_array)
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
```

You can see that the above output is different from previous and is sorted by ncalls. You can sort the output in various other ways.

6. How to export cProfile data?

By default, the output of the profiler is simply printed out. But, you can use store the extracted data of profiling in a file as well. How to export the data/report?

The pstats module comes to use here.

After creating a Stats instance, pass the profiler as input to it as shown below. After that, use dump_stats() method to store it to any file by providing the path.

```
# Export profiler output to file
stats = pstats.Stats(profiler)
stats.dump_stats('/content/export-data')
Now, let’s consider a bit more lengthier example to organize the
profiler output better. Let’s create a profile for the below code and
print the report.
```

```
# Using cProfile.Profile example
import random

def print_msg():
    for i in range(10):
        print("Program completed")

def generate():
    data = [random.randint(0, 99) for p in range(0, 1000)]
    return data

def search_function(data):
```

```
    for i in data:
        if i in [100,200,300,400,500]:
            print("success")

def main():
    data=generate()
    search_function(data)
    print_msg()

if __name__ == '__main__':
    import cProfile, pstats
    profiler = cProfile.Profile()
    profiler.enable()
    main()
    profiler.disable()
    stats = pstats.Stats(profiler).sort_stats('tottime')
    stats.strip_dirs()
    stats.print_stats()
    stats.dump_stats("profile.prof")
```

```
Program completed
Program completed
Program completed
Program completed
Program completed
Program completed
Program completed
Program completed
Program completed
Program completed
    5552 function calls in 0.003 seconds

   Random listing order was used

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1000    0.000    0.000    0.000    0.000 {method 'bit_length' of
'int' objects}
       20    0.000    0.000    0.000    0.000 {built-in method posix.
getpid}
       21    0.000    0.000    0.000    0.000 {method 'acquire' of
'_thread.lock' objects}
     1268    0.000    0.000    0.000    0.000 {method 'getrandbits' of
'_random.Random' objects}
       21    0.000    0.000    0.000    0.000 {method 'append' of
'collections.deque' objects}
       20    0.000    0.000    0.000    0.000 {built-in method builtins.
isinstance}
       10    0.000    0.000    0.001    0.000 {built-in method builtins.
```

```
print}
        1    0.000    0.000    0.003    0.003 <ipython-input-30-
2a521dc30378>:15(main)
        1    0.000    0.000    0.003    0.003 <ipython-input-30-
2a521dc30378>:6(generate)
        1    0.000    0.000    0.000    0.000 <ipython-input-30-
2a521dc30378>:10(search_function)
        1    0.000    0.000    0.001    0.001 <ipython-input-30-
2a521dc30378>:3(print_msg)
        1    0.000    0.000    0.003    0.003 <ipython-input-30-
2a521dc30378>:7(<listcomp>)
       21    0.000    0.000    0.000    0.000 iostream.py:93
(_event_pipe)
       21    0.000    0.000    0.000    0.000 iostream.py:195(schedule)
       20    0.000    0.000    0.000    0.000 iostream.py:307
(_is_master_process)
       20    0.000    0.000    0.000    0.000 iostream.py:320
(_schedule_flush)
       20    0.000    0.000    0.000    0.000 iostream.py:382(write)
       21    0.000    0.000    0.000    0.000 socket.py:357(send)
        1    0.000    0.000    0.000    0.000 {method 'disable' of
'_lsprof.Profiler' objects}
     1000    0.001    0.000    0.002    0.000 random.py:173(randrange)
     1000    0.001    0.000    0.002    0.000 random.py:217(randint)
     1000    0.001    0.000    0.001    0.000 random.py:223(_randbelow)
       21    0.000    0.000    0.000    0.000 threading.py:1104
(is_alive)
       21    0.000    0.000    0.000    0.000 threading.py:506(is_set)
       21    0.000    0.000    0.000    0.000 threading.py:1062
(_wait_for_tstate_lock)

<pstats.Stats at 0x7f58db5659e8>
```

7. How to visualize cProfile reports?

However reading the stats alone may be a daunting task for a more complex function call. A good solution to get a clear picture of the profiling data is to visualize it.

A best tool available at the moment for visualizing data obtained by cProfile module is SnakeViz.

Installing the module

Let's install it through the below command.

```
!pip install snakeviz
Collecting snakeviz
[?25l  Downloading <https://files.pythonhosted.org/packages/a2/9a
/6c753d20af6f177d3cbdb05a4b2e4419db4ec021c50ba86aa0d13a784a5c/snakeviz-
2.1.0-py2.py3-none-any.whl> (282kB)
[K     || 286kB 2.8MB/s
[?25hRequirement already satisfied: tornado>=2.0 in /usr/local/lib
```

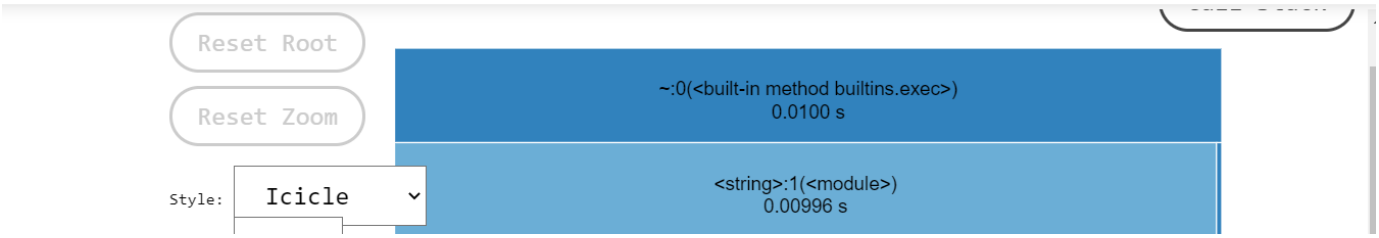
```
/python3.6/dist-packages (from snakeviz) (5.1.1)
Installing collected packages: snakeviz
Successfully installed snakeviz-2.1.0
For Ipython notebooks like google colab and Jupyter, you can load the
SnakViz extension using %load_ext snakeviz command.
```

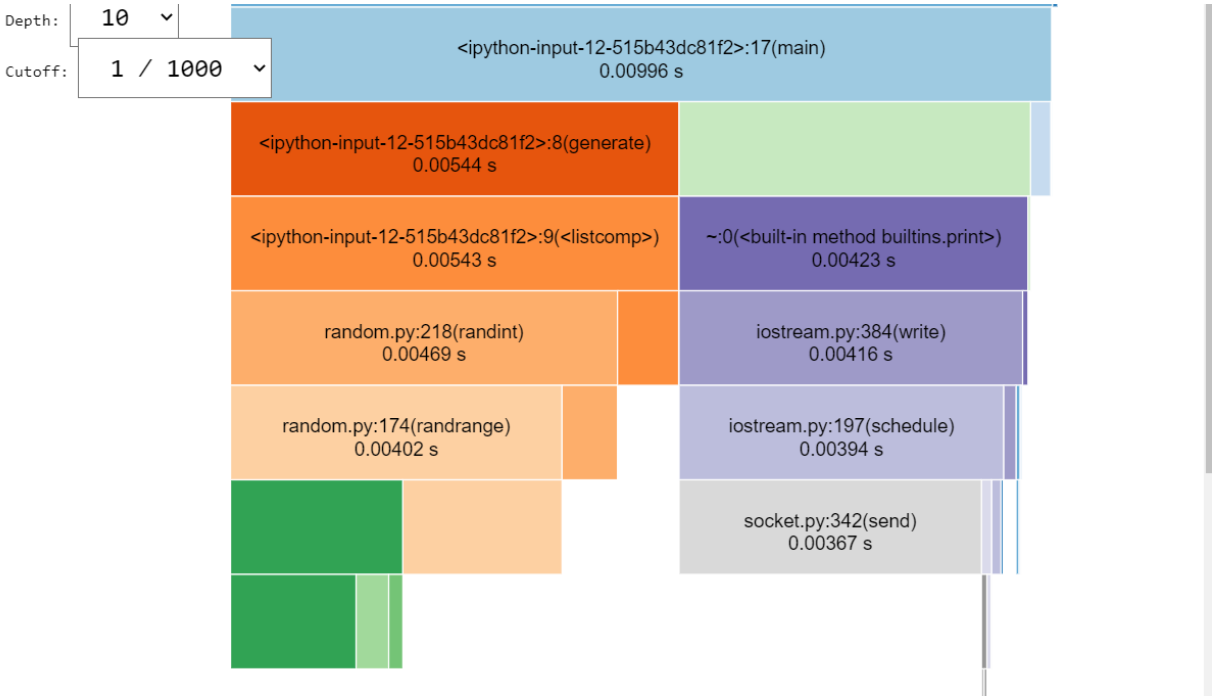
After this, call the function or program's profiling you want to visualize through the %snakeviz <filename>. The filename can be either the entire python script or call to a particular function.

In the below code, I have written a main() function which calls several basic functions like creating an array and searching for specific elements. Now, to visualize the profiling data of the entire program I can use the command %snakeviz main() .

```
# Code to test visualization
import random
# Simple function to print messages
def print_msg():
    for i in range(10):
        print("Program completed")

# Generate random data
def generate():
    data = [random.randint(0, 99) for p in range(0, 1000)]
    return data

# Function to search
def search_function(data):
    for i in data:
        if i in [100,200,300,400,500]:
            print("success")

def main():
    data=generate()
    search_function(data)
    print_msg()
```

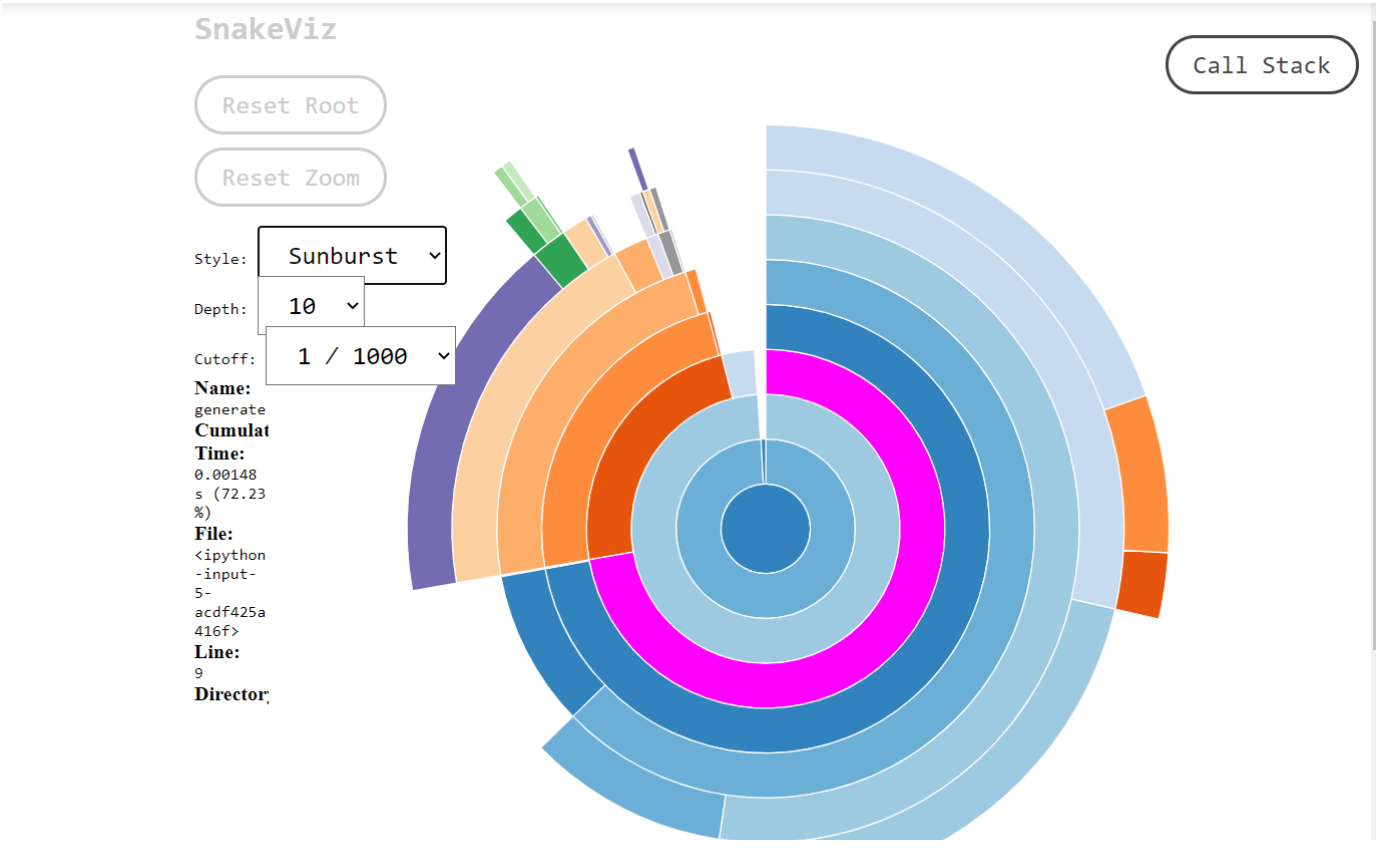
```
%load_ext snakeviz
%snakeviz main()
cProfile Visualization - Snakeviz ( Icicle)
cProfile Visualization – Snakeviz ( Icicle)
SnakeViz has two visualization styles, ‘icicle’ and ‘sunburst’.
```





By default, it's icicle. icicle, the fraction of time taken by a code is represented by the width of the rectangle. Whereas in Sunburst, it is represented by the angular extent of an arc. You can switch between the two styles using the "Style" dropdown. For the same code, let me show you the Sunburst style visualization too.

```
cProfile visualization
cProfile Visualization – SnakeViz ( Sunburst )
```



Development Mode

The Python Development Mode introduces additional runtime checks that are too expensive to be enabled by default. It should not be more verbose than the default if the code is correct; new warnings are only emitted when an issue is detected.

It can be enabled using the -X dev command line option or by setting the PYTHONDEVMODE environment variable to 1.

ResourceWarning Example

The example script below does not close the file explicitly. By default, Python does not emit any warning.

```
import sys

def main():
    fp = open(sys.argv[1])
    nlines = len(fp.readlines())
    print(nlines)
    # The file is closed implicitly

if __name__ == "__main__":
    main()
```

Example using README.txt, which has 269 lines:

```
$ python3 script.py README.txt
269
```

Enabling the Python Development Mode displays a ResourceWarning warning:

```
$ python3 -X dev script.py README.txt
269
script.py:10: ResourceWarning: unclosed file <_io.TextIOWrapper
name='README.rst' mode='r' encoding='UTF-8'>
  main()
```

In addition, enabling tracemalloc shows the line where the file was opened:

```
$ python3 -X dev -X tracemalloc=5 script.py README.rst
269
script.py:10: ResourceWarning: unclosed file <_io.TextIOWrapper
name='README.rst' mode='r' encoding='UTF-8'>
  main()
Object allocated at (most recent call last):
  File "script.py", lineno 10
    main()
```

```
  File "script.py", lineno 4
    fp = open(sys.argv[1])
```

Interfaces

Interfaces play an important role in software engineering. As an application grows, updates and changes to the code base become more difficult to manage.

At a high level, an interface acts as a **blueprint** for designing classes. Like classes, interfaces define methods. Unlike classes, these methods are abstract. An **abstract method** is one that the interface simply defines. It doesn't implement the methods.

Wiki & Documentation

Best Practices

You should create a document if it fulfills a clear, important, and immediate goal of your overall project efforts, this purpose may be short term or long term.

The goal is to ensure that readers understand how the system works so they can confidently evolve it over time, not to produce a mound of documentation that they may or may not use.

The goal is to ensure that your users work with your system effectively, not that they have a **pretty help system** available to them. The result of documentation is to enable others and not bury them with paragraphs.

Documentation attempts to achieve the following:

1. Clear & concise refresher of the reasons behind the decisions/design that you made
2. Getting fellow developers up to speed with the code you wrote

If you think you'll likely to forget whatever thing you wrote then simply write a few lines about it.

The issue with documentation is sometimes it's not very clear to what level of detail should things be documented, too detailed then it could be cumbersome/irrelevant, not enough detail would defeat the purpose of keeping information. Here are some points to help you figure out the answer to that.

Best Practices for Simplification

The following practices will help you to simplify the documentation that you write:

1. Keep documentation just **simple enough**, but **not too simple**
2. Write the fewest documents with **least overlap**
3. Put the information in the most appropriate place and form

Keep Documentation just Simple Enough

The best documentation is the simplest that gets the job done. Don't create a fifty-page document when a five page one will do. Don't create a five-page document when five bullet points will do. Don't create an elaborate and intricately detailed diagram when a sketch will do. Don't repeat information found elsewhere when a reference will do. Write in point form. Document only enough to provide a useful context.

Write the Fewest Documents with Least Overlap

Strive to travel as light as you possibly can, writing on just enough documentation for the situation at hand. One way to achieve this is to break a big topic into from bite-sized documentations.

Agile teams often use Wikis like this. One page described the user interface architecture for our system, a page which included a user interface flow diagram and appropriate text describing it. The advantage was that this information was defined in one place and one place only, so there is no opportunity for overlap.

Put the Information in the Most Appropriate Place and Form

Where will somebody likely want a piece of documentation? Is that design decision best documented in the code, added as note on a **diagram**, or best placed in an **external document**? Is a specific requirement best captured as part of a use case, in a business rule specification, or as an executable test?

The answer to these sorts of question should be driven by the needs of the customer of that information.

You should strive to record the information once where it enhances your work the most.

Treat Documentation Like a Requirement

Documentation should be estimated , prioritized, and put on your work item stack along with all other work items to be addressed.

The need to write a document clearly is a requirement just like the need to write a feature. Any investment we make in documentation is investment that we could have made in new functionality, and vice versa, so someone should make a conscious decision as to how much that investment (if any), should actually be.

However,

Well-written documentation supports organizational memory effectively, but is never better than direct communication during a project.

References

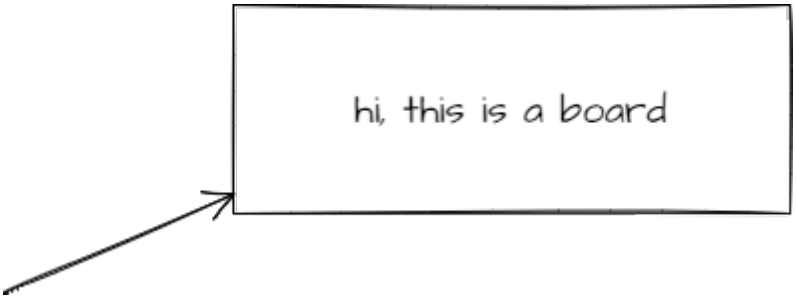
http://agilemodeling.com/essays/agileDocumentationBestPractices.htm

https://developers.google.com/style

Plugins

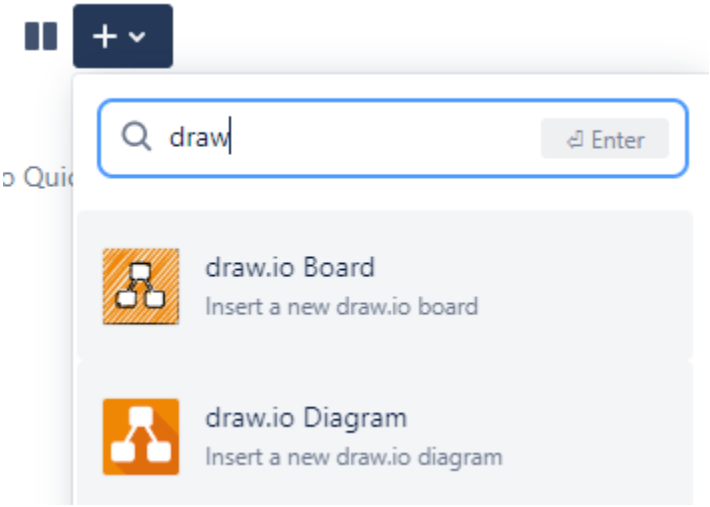
List of available plugins and some short guides on how to use them.

draw.io Quickstart



Introduction

A diagram is inserted like any other macro in a Confluence page. (Click the Plus ("**+**"), and then search)

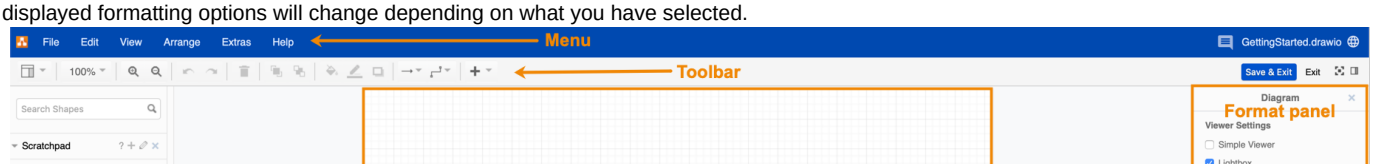


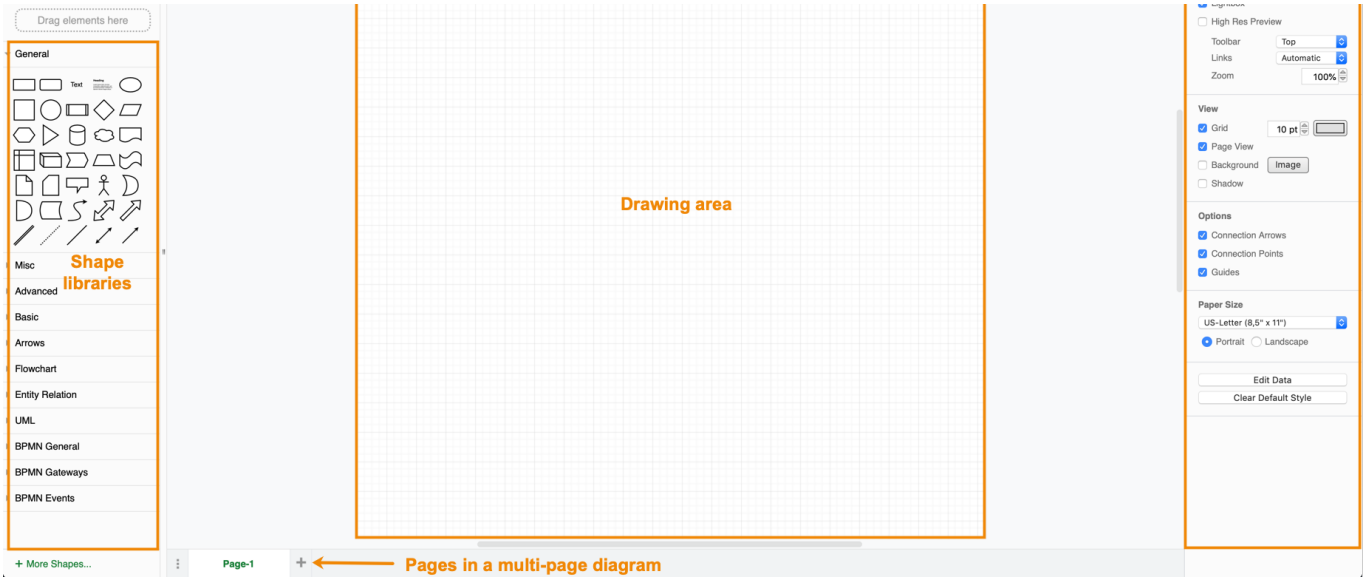
Or type slash /draw and it the macro should pop up.

Create a new page in Confluence, or edit a page that you have permission to edit.

The Interface

A diagram typically consists of shapes, connectors, and text. Depending on what you work with, the available tools in the toolbar and the displayed formatting options will change depending on what you have selected.





Toolbar

The toolbar is displayed directly under the draw.io menu. The available tools will change depending on what you are working with in your diagram.

Drawing area

The drawing area is in the middle – what you see in this area is what is displayed in your Confluence page after you save it. There are guide lines to help you align each component in your diagram.

You can move around the drawing area with the mouse wheel or the scroll bars on the sides of the drawing area. Or you can click and hold the right mouse button and drag the drawing area around.

To zoom in and out, use the two magnifying glass icons in the toolbar. Alternatively, press and hold the **Alt** key and use the mouse wheel to zoom.

Shape library

The shape library is displayed on the left hand side of the drawing area and contains a variety of shapes, connectors, and text organized into categories. You can expand or collapse these categories as needed.

To add a shape to your diagram, simply click on a shape in the symbol library, or click and drag it into the diagram.

Shapes, text, and connectors in the drawing area are collectively called objects.

You can add your own shapes to a custom library, if you can't find the ones you need.

Format panel

When you select an object, you can customize how it appears, text included within the shape, or the arrangement and size in the format panel on the right hand side of the drawing area.

When nothing is selected, the drawing area itself may be customized.

The contents of the format panel change depending on what you have selected.

More from their official docs

How to Contribute

This wiki page will help you to get an overview on what needs to be done and what you need to consider before you can begin contributing code.

GitHub

When contributing to our GitHub, there are a few things you will need to consider. Learn about the structure of the repository and its conventions here.

GitHub

Python

Visit the following pages, to get on page with other contributors.

Setup

Python Auto Formater

Guidlines

PEP 8

Pathlib

cProfile – How to profile your python code

JavaScript

Visit the following pages get on page with other contributors.

Setup

Linting/Formatting javascript in Vscode

Linting/Formatting javascript in Vscode

ESlint for formatting and linting

Eslint is highly a customizable linting extension for Javascript. Unlike in the case of Python, auto formatters and linters in Javascript happen to be very flexible to the point where there is no difference in choosing any popular extension over the other. Most of them can be customized to output the same code appearance. In this section, we discuss how to set up ESlint. I took the liberty of choosing this extension since it is very popular and has some advantages over other extensions like Prettier, in the sense that it also provides static code analysis.

Prettier for formatting

As stated on the official website, Prettier is:

- An opinionated code formatter
- Supports many languages
- Integrates with most editors
- Has few options

ESLint Vs Prettier

- ESlint is not only a code formatter, it also helps developers to find coding errors. For Example, ESLint will give you a warning if you use a variable without declaring it. Prettier doesn't have such an ability.
- Also, ESLint will let you know what's wrong with your code formatting and give you options to fix the issue. Then you can select one from those options. Prettier, on the other hand, doesn't care about you at all. It simply formats all your code to a different structure format-wise.
- On the other hand, this whole rewriting process in Prettier prevents the developer from making any mistakes.
- max-len, no-mixed-spaces-and-tabs, keyword-spacing, comma-style these are some popular formatting rules in Prettier.
- In addition to the above type of rules, ESLint also considers code quality rules such as no-unused-vars, no-extra-bind, no-implicit-globals, prefer-promise-reject-errors.

Overall, these methods seem to complement each other, while having some similarities.

How to set up ESlint

We'll use npm to install this extension, so NodeJS needs to be installed on the system first.

to install ESlint, run:

```
npm install -g eslint
```

then initialize ESlint for the particular project:

```
eslint --init
```

a prompt will appear with multiple-choice options:

I chose:

Check syntax, find problems, and enforce code style Javascript Modules (import/export) React Browser Use Popular style guide Airbnb JSON

but of course, this is from what I read to be the most usual choices, we can always discuss other configurations.

after finishing the setup, a configuration file will be generated in the root directory of the project under the name: .eslintrc.json

the customization of the coding syle will go under the property “rules“

For the linting to work properly, the vscode extension: ESlint needs to be installed from the vscode extensions marketplace.

Installing the Prettier Extension

To work with Prettier in Visual Studio Code, you’ll need to install the extension. Search for Prettier - Code Formatter.

Userful links:

https://eslint.org/

https://github.com/airbnb/javascript

https://www.youtube.com/watch?v=slEtjbr8sVY&t=249s

https://prettier.io/

Python Auto Formater

An auto formater is a tool which allows you to remove PEP8 errors automatically, but also streamline the look of your code. This prevents the linter to throw errors when you are pushing the code to the repository.

Yapf

The yapf auto formater is based of the 'clang-format', which not only tries to resolve PEP8 errors but also generates good looking code. With its numerous customization options and focus on looks, the yapf auto formater lies in between its to main competitor, autopep8 and black in terms of uniformization . While autopep8 has a singular focus on removing PEP8 without offering uniformization, the black unapologetically enforces its own code style.

The google python style guide improves on the PEP8 style to define what better looking code should be like. Together, we chose to base our code style on the google style, with the help of yapf.

Setting up The Formater

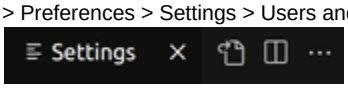
To set up the formater, you simply need to have it installed in the python environment, which you will be using in your VSCode. To do so, open a terminal in VSCode and run:

```
pip install yapf
```

Workspace Settings

To integrate the auto formater into our project, we are going to make us of a .vscode/settings.json. This file will contain shared settings for the entire team.

While this allows us to provide you with the necessary setting to make the auto formater work remotely, this will overrule your user settings. To alleviate this issue, copy your settings to the file in the work space (.vscode/settings.json) from your user settings. Find your user settings at File > Preferences > Settings > Users and access the json by clicking on the icon at the top right of the tab next right to Settings.



Usage

VSCode Settings

Yapf will format the code as specified by the VSCode settings. The available options are:

- OnPaste: only the pasted code
- OnType: Formats the line after you are done editting
- OnSave: Formates the entire file after editing it

The default settings we provide you with are OnPaste and OnType, since the OnSave setting is prone to creating merge conflicts.

Disable Yapf

Yapf will do its best to generate the best formatting for your code. However, this is not always possible since the yapf algorithm needs to balance between PEP8, the line length limit and the google style guide. If the result of a formating degrades the readability of the statement too much, you can disable the formating by placing a # yapf: disable comment behind the statement.

Example:

We are trying to unpack a large amount of return values, however the statement violates the line length limit.

```
```

```
X_train, X_test, X_val, y_train, y_test, y_val = datasets.
train_test_split(X_subjects,

y_subjects,

test_size=0.2,


val_size=0.1,

concatenate=False)
```

To alleviate this issues, we want unpack the return values into a tuple, but yapf does its thing and the result looks something like this.

```
(X_train, X_test, X_val, y_train, y_test,
 y_val) = datasets.train_test_split(X_subjects,
                                    y_subjects,
                                    test_size=0.2,
                                    val_size=0.1,
                                    concatenate=False)
```

Your OCD kicks in and you feel really bad about the look of this. You can now fix it by doing:

```
(X_train, X_test, X_val,
 y_train, y_test, y_val) = datasets.train_test_split(X_subjects,
                                                     y_subjects,
                                                     test_size=0.2,
                                                     val_size=0.1,
                                                     concatenate=False)
# yapf: disable
```

GitHub

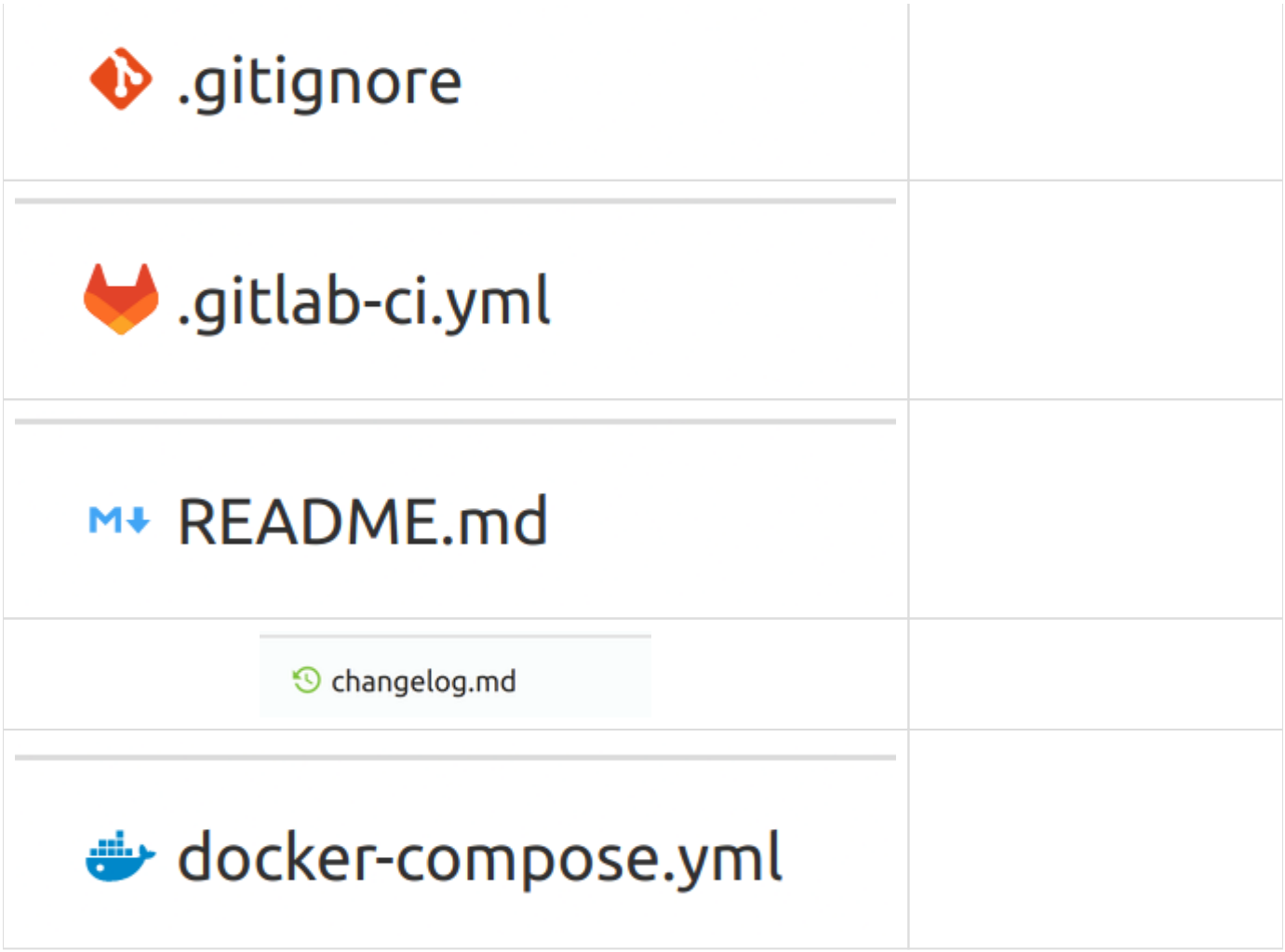
Our code lives in our GitHub! This wiki addresses how to navigate and contribute to our shared code.

Structure

Lets have a brief look at our folder structure.

	Common settings for the VSCode editor

	Folder in which the data is stored locally. The folder is excluded from version control to keep our git light
	Folder for settings and environment setup files.
	Frontend source folder.
	
	Store your research, demonstration and tutorial noteboooks here.
	
	Folder containing custom python module.
	



Conventions

Lets have a look at our naming conventions.

Folder Names:	snake_case
File Names:	snake_case
Branch Names:	feature_snake_case_{jiraid} fix_snake_case docker_snake_case experimental_snake_case_{jiraid} develop master

Other Resources

Things that could be useful can be posted here:

Default Templates

xx

How-to article

🟦 Instructions

ℹ️ Highlight important information in a panel like this one. To edit this panel's color or style, select one of the options in the menu.

📋 Related articles

Master project documentation

This template is brought to you by Mural, a visual collaboration app.

❓Unmet needs

🏹 Objectives

👥 User personas

💪 Jobs we want to cover

- When I
- I want to
- So I can

📜 Some history

👀 Constraints

🗺️ Explorations + Decisions

🗓️ Releases

Release Name	Value it adds	Scope	Status	Completed date
			TO DO / IN PROGRESS / BLOCKED / WAITING FOR FEEDBACK / DONE	

👣 Next steps

- [ ] *SET A STATUS*

📈 Impact

🔗 Other documents