

DoF

Deep Model Core Output Framework



Table of Contents

Storytelling.....	3
The born of the idea.....	3
Advantages of our idea.....	5
DoF - Deep Model Core Output Framework.....	6
The problem and need.....	6
Basics of the idea.....	7
Structure.....	7
Formulas.....	10
Total saved time.....	10
Benefit ratio.....	10
Total benefit ratio.....	10
Total saved space.....	10
Beneficiaries.....	11
Complete documentation.....	12
Requirements.....	12
Common notations.....	12
Class reference.....	13
class DofElement(x, y[, info=None]).....	13

class DofElementInfo(**kwargs).....	14
class DofError().....	14
class DofFile(*args[, dataset_path='./dataset'][, use_compressed=False]).....	15
class DofInfo(*args, **kwargs).....	18
Future plans.....	19
About us.....	20
cheXrad.....	21
COVID-19 dataset.....	22
COVID-19 map.....	23
COVID19CasesWorldwideDatasetFromECDC.....	24



Storytelling

Imagine, the world is better tomorrow, than today. What is the difference? What if everybody tries to make something little change? The difference is this “what if”. There are two types of inventions. One group of them makes huge impact on the peoples daily routine. Those are a game-changers. The other type of ideas are different. These are very small and basic concepts with only one goal: made the piece of world a little bit better or effective. We do not want to revolutionize people’s life, we do not want to twist upside-down the world. We want to make a tiny but significant change with our idea.



The born of the idea

We are two deep learning developers who suffer from typical developer issues: the resource scarcity and legal boundaries like General Data Protection Regulation (GDPR) in European Union. We made a healthcare software to help doctors make a better, faster, and more precise diagnosis about pulmonary disease. For his point it sounds, there is no something unique in the software. Our intent was duplex: to build a robust model or maybe a chain of networks and to reach this target with using the software in resource-scare environment. In our country the computers in hospital are old without GPU and the hospitals do not have the necessary financial support to buy new ones or to pay for cloud-based solutions. Beside the financial issues, there is a legal burden. The data that come from healthcare system can be used very straight, clear and predetermined way due to data protection of patient. Health data is protected extremely in the most countries. We made an opportunity to train our model faster and be able to use the health data without violate the regulations and without the preliminary contribution statement from patient for deep learning usage. How can these goals be reached? We optimize the training and evaluation phase by calculating the repetitive tasks only once and we anonymize the data.

Our software contains different and independent neural networks. Each of them is trained separately. Detecting pulmonary diseases on X-ray images is an image detection and image processing task. We use headless pretrained model to make the necessary detection and preprocessing tasks. However, we use different AIs, each of them builds up similarly: a headless pretrained model is connected to our network. We can save a huge amount of time with feed the pretrained model only once and save it's output. There is no need to repeat over and over the same process on the same picture again since the result will be the same every time. Our idea is born here.

If we save the output for example a ResNet or VGG network, why should we drop these data to trash at the end of the process? We decided to create a unique and new file format that not only contains the output data of headless network, but extra additional information. This container file is much more than a dataset.

Let's see an example. There is a COVID-19 X-ray dataset from github (<https://github.com/ieee8023/covid-chestxray-dataset>) that obviously contains images and metadata. However, it contains sensitive health data connected to patients. These pictures may be able to use scientific purposes, but a lot of country forbids using similar datasets for commercial usage. Using pretrained model with any type of pooling layer can anonymize the unique element of original dataset. From the result of pretrained headless model cannot be restored the original data since the pooling layers make reversibility proof destruction on data. However, there is necessary to get contribution statement from patient, but with this method the model easily wipes the dataset from personal or person-connected health. This part is very important if you handle data from patient connected to European Union due to GDPR.



Advantages of our idea

Using our container file has multiple advantages. A preprocessed data consumes less space than the original picture. The global internet bandwidth has its own limitation. During an epidemic crisis when people have to be at home and use the internet whole day to work or to get news, we have to spare with bandwidth. Big companies and web services like Facebook, Instagram, Youtube, Netflix announced to limit the dataflow in European Union by reduce quality of video stream. One megabyte of an uncompressed image 24bit-RGB image contains approximately 349 920 pixels, that equals to 486 x 720 resolution. With jpg compression the size can be reduced or the resolution can be increased. Comparing the size of image, the container file has very small size. The output of a headless model is under 10 Kilobyte. Whether we add a lot of information to it or not, the size of container file can be holded under the size of processed image and its metadata.

Loading preprocessed data from container file reduces training time in two ways. First of all there is no need to run and to load the pretrained model into the memory. Secondly, we can use higher batch size which reduce the training time dramatically not just on expensive hardware but on low-end machines too.

Preprocessors are able to comment notes and add additional information or data to the container file that helps the user to reproduce the training process or build something complex. For example, the preprocessor author can note the data as train, test or valid data, or can suggest network architecture.

DoF - Deep Model Core Output Framework

The problem and need

Datasets are large and contains sensitive data in the most of cases. Sensitivity means personal and personal related health data and maybe business secrets as well. By working with health data, developers always see, datasets are heavily accessible and limited restrictions by legal regulations. For example, a lot of datasets are available only for an invited group of researchers. Second part of dataset are registration-dependant. This registration process is made by hand and are available only during working hours yet nowadays. The third kind of datasets are open to use with limitations after a short online registration process. These limitations usually are scientific purpose only, and forbidding the commercial use. The fourth kind of datasets are fully open without any limitation.

Using personal related health data means the developer or researcher handle sensitive data. States regularly make restrictions the usage of health data. In European Union the General Data Protection Regulation (GDPR) can cause competitive disadvantage.

Computers in hospitals are mostly outdated and slow. This sector frequently suffers from lack of financial resources. This means the circumstances are far away from optimal for building, training and using robust AI networks.

The internet bandwidth is a highly limited resource. During normal days, we are not facing these limitation. During quarantine or curfew everybody use of internet for work or get information. Spare with bandwidth is essential in the time of an epidemic.

However, these problems are mentioned from field of healthcare and computer vision, the problem is not particular in deep learning development. We can see these symptoms in every field of AI development.



Basics of the idea

DoF is a highly scalable dataset format which helps deep learning scientists to work with foreign or sensitive data. DoF provides fast dataset sharing and data-secure at the same time.

It is fast because during training process it's enough to run forward the dataset through the pretrained core models only once. It's not necessary to calculate these values again at the next epoch. It saves computational time. These values will be always the same because the core part of the model is no more trained.

It is secure since headless deep learning networks anonymize the unique element of the original dataset. The original data cannot be restored from the saved result, layers make reversibility proof destructions.

The DoF framework is not just a container file. It is away of thinking and sharing datasets to help each others work.

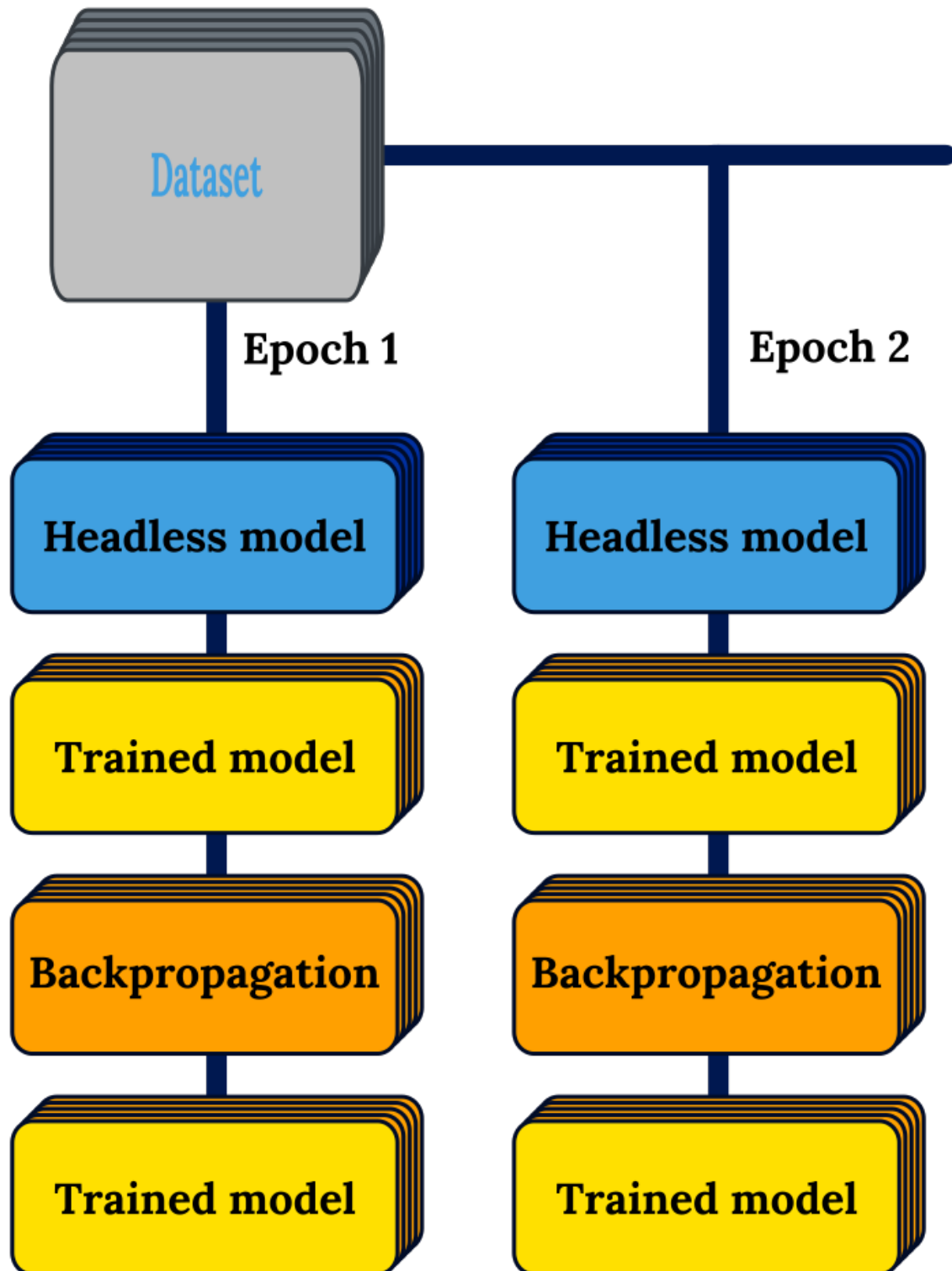


Structure

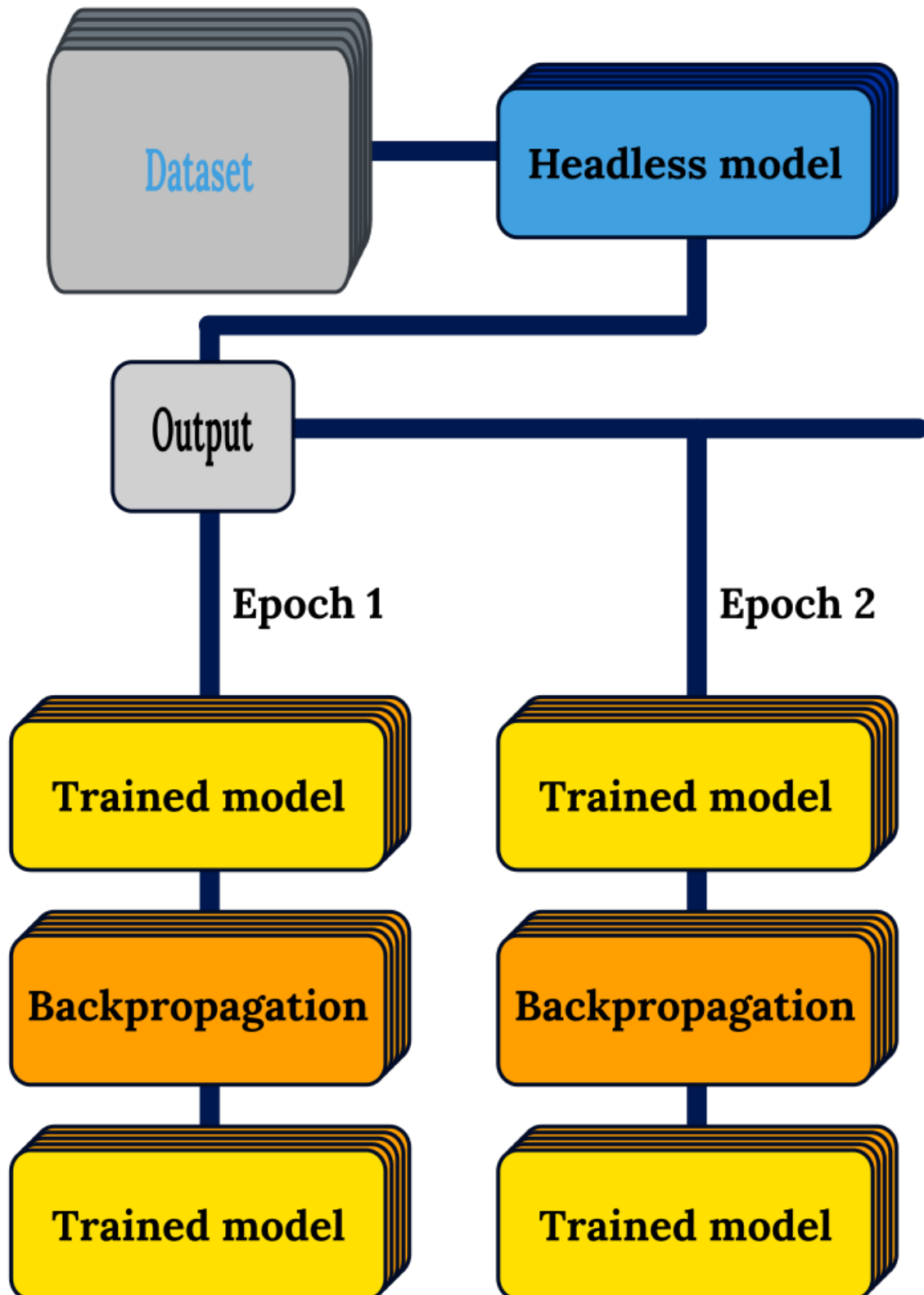
The DoF module contains 5 classes:

- DofElement to hold dataset element in DoF style
- DofElementInfo to store dataset element level information
- DofError for error handling with unique exception
- DofFile to manage DoF file and/or DoF dataset
- DofInfo to store dataset level information

normal train process



DoF train process





Formulas

DIS = DoF info size

EC = Epoch count

IC = Image count

IS = Image size

HNCT = Headless network computation time

HOLT = Headless output loading time

HOS = Headless output size

TNCT = Trained network computation time

Total saved time

$(\text{HNCT} - \text{HOLT}) * \text{EC} - \text{HNCT}$

Benefit ratio

$(\text{TNCT} + \text{HOLT}) / (\text{HNCT} + \text{TNCT})$

Total benefit ratio

$((\text{TNCT} + \text{HOLT}) * (\text{EC} - 1) + (\text{HNCT} + \text{TNCT})) / ((\text{HNCT} + \text{TNCT}) * \text{EC})$

Total saved space

$(\text{IS} * \text{IC}) - (\text{HOS} * \text{IC}) - \text{DIS}$

Beneficiaries

The solution is good for deep learning developers who are able to train model faster, download less data during dataset sharing and use less storage. Shortening the training by calculating headless model output only once includes a possibility to use the saved time for building and training bigger model.

Preprocessors are able to licence their own work and add notes to the DoF container to help guiding the end user or the next developer. This opens the new form of cooperation between developers. In this moment the preprocessor state dissolves into the dataset making or deep learning developing. With DoF method there will be a whole new field for programmers and developers.

The community gains benefit by reducing the size of datasets. During quarantine or curfew a lot of people use the internet for working, gathering information, maintaining social connections. Less data results less loaded internet network and ISPs can serve the users' need smoothly.

Data owners should not share their sensitive data or business secrets. It is enough to provide access to their business partners (developers, related companies) for the headless model output. This output is irreversible. If they want to make more secure solution, there is a chance to build, train and use new headless models in DoF system.

In healthcare system patients do not have to worry about data-protection since their image-based medical data cannot be traced back. However, it seems to be necessary for maximizing the data-protection to handle prudent the information section and metadata in DoF container file.



Complete documentation

Requirements

The python module DoF requires nothing than libraries from the Standard Library. However to do training and validation you'll need at least Deep Learning frameworks like PyTorch and efficient side modules like NumPy. DoF itself counts on NumPy's functionality even if it doesn't require it be imported. Saving data in NumPy instead of saving it in Pytorch means about 0.25 % less storage needs.

DoF uses from Standard Library: json, parts of os and parts of os.path, pickle, parts of ZipFile

Common notations

The variable name x is used to sign the output of the core model. The letter x is selected because core model's output is used as training model's input. Therefore we consider x a good naming.

The variable name y is used to sign the target of the training, test or validation.

Class reference

class DofElement(x, y[, info=None])

Represents a dataset element. Its main advantage is that it handles dataset elements in two different ways.

x (object | string) The x value of the element or the path to the file with the value. String type values are always considered as paths.

y (object) The corresponding y value. Since y values are usually much smaller than x values are it is stored in memory only.

info (DofElementInfo | None) Additional information to the element or None if no additional information is available. This parameter is optional.

DofElement.info()

Gets information of the element or returns None if no information exists.

DofElement.islink()

Returns True if element x value is represented as link or False if the x value is in memory.

DofElement.link()

Gets the link of the x value of the element. Calling this function on a memory based instance leads to error.

DofElement.x()

Gets the x value of the element no matter if it is in memory or in file.

DofElement.y()

Gets the y value of the element.

class DofElementInfo(kwargs)**

Represents information of unique dataset element. Canonical usage is not modify the data after instantiation but there's no tool in python to forbid this. However due to clean coding style it worth to act this way.

****kwargs** (key, value pairs) Keys and values to add basic and more advanced information to dataset element. Required keys are:

author: author, owner of the specific element

author_contact: contact address (email, phone, homepage, etc.) to the author of the element

source: link to the source of the element

license: license of the element

Giving further keys and values are available and suggested too.

class DofError()

Represents a Dof module exception. There are various exceptions thrown by the module. Distinct exception is made for cleaner coding purposes since you can separate clearly module errors from any other errors.

class DofFile(*args[, dataset_path='./dataset'], use_compressed=False)

Represents a DoF manager. A DoF manager handles a compressed DoF file or a DoF file based dataset. However the DoF manager is not a subclass of PyTorch's DataSet class it can be used in PyTorch DataLoader class as representation of a dataset so don't have change your training routine in PyTorch to use DoF.

DoF manager has two different modes at the moment. Read mode and write mode. In read mode the manager have 2 different behavior. If file name is given it opens the file and extracts its content to dataset folder. If file name is omitted the manager uses a DoF dataset folder only. In write mode file name is required. The manager collects the data to save it at the end.

Positional arguments [file], mode ([string], string) Required argument(s). In read mode 1 argument DofFile.READ is enough if dataset folder contains a fully extracted DoF dataset. If 2 argument is given, 1st argument is treated as the name of the DoF file to read or write and the 2nd argument is the access mode selector.

dataset_path (string) Path to the dataset to locate incoming or outgoing files. This parameter is optional since it has default value the folder name “./dataset”.

use_compressed (bool) Sets whether to use dataset compressed or uncompressed. This value is optional. At the moment setting this value doesn't have any effect because compressed mode is not implemented yet.

DofFile.append(data)

Appends an element or list of elements to dataset.

data (DofElement or numpy.ndarray like object) Dataset element or list of dataset elements like a batch for example. In case of a single element the element should be DofElement in case of multiple elements, the elements should implement .shape() and .tolist() functions like numpy.ndarray does.

DofFile.check()

Checks if DoF dataset has every needed element. At the moment this function is implemented for read mode with uncompressed behavior only.

DofFile.delete(id)

Deletes an element from the dataset. In read mode the element is deleted for this specific session only. In write mode deleting an element means the element will be deleted from the saved dataset too.

id (int) The index (postion) of the element in the dataset.

DofFile.filename()

Gets the file name string bound to the dataset. If file name was omitted, it returns an empty string.

DofFile.info([newvalue=None])

Gets or sets the dataset level information.

newvalue (DofInfo or None) If DofInfo instance is given, it is set as new dataset info. If None is given, the current dataset info is returned.

DofFile.mode([mode=None])

Gets or sets the Dof manager mode.

mode (String or None) If mode is DofFile.READ or DofFile.WRITE new mode will be set in the future but at the time this is not allowed. If None is given the current mode is returned.

DofFile.save([remove_dataset_dir=False])

Saves the dataset to a DoF file.

remove_dataset_dir (bool) Whether or not dataset directory should be removed at the end of save.

DofFile.use_compressed([newvalue=None])

Gets or sets the Dof uncompressed mode.

newvalue (bool or None) If newvalue is bool the new mode will be set in the future but at the moment it is not implemented yet. If None is given it returns the current use_compressed state.

DofFile.__getitem__(id)

Gets an item. This method provides compatibility with Pytorch's DataSet and DataLoader.

Id (int) The id of the element to return.

DofFile.__iter__()

Gets an iterable instance, gets the instance itself. This method provides iterable functionality of DoF manager.

DofFile.__len__()

Gets the length of the dataset. This method serves for compatibility with iterators and PyTorch as well.

DofFile.__next__()

Gets next element from dataset. This method provides iterable functionality of DoF manager.

class DofInfo(*args, **kwargs)

Represents information of the whole dataset. Canonical usage is not modify the data after instantiation but there's no tool in python to forbid this. However due to clean coding style it worth to act this way.

args (positional arguments is [dict]) List if dict with length 1. (This way is reserved for DofFile read mode's JSON based instantiation only.)

**kwargs (key, value pairs) Keys and values to add basic and more advanced information to dataset. Required keys are:

coremodel_family: Identification name of the core model (e.g. "ResNet").

coremodel_type: Subclass name of the core model (e.g. "50").

coremodel_common: True or False Whether the core model is common or not. Being common means that it's available in most Deep Learning frameworks.

original_author: Author, owner of the original dataset.

original_source: Link (or other availability) to the original dataset.

original_license: License of the original dataset.

dof_author: Author (compiler, owner) of the DoF dataset.

dof_author_contact: Contact address (email, homepage, phone, etc.) to the author of the DoF dataset.

dof_source: Link to the DoF dataset.

dof_license: License of the DoF dataset.

In case if coremodel_common is False further keys are required:

coremodel_source_architecture: Link to the surce of the core model.

coremodel_source_weightsandbiases: Link to the wights and bias data of the core model architecture.

Giving further keys and values are available and suggested too.



Future plans

Our current project is based on PyTorch, but we will be working on extending to other frameworks like TensorFlow, Keras.

Our zip container file works in two modes: create and use. We would like to make a third usage: modify. This will open the possibility of a new type of cooperation between researchers. They will be able to modify, refine the work of each other in a time-effective way by skipping the whole building process and build only the new elements or arguments.

If we see through the current situation and want to make something timeless, we have a lot of opportunity as a developer community. Using container files is good and can improve our community, but we can do more. These files can be gathered and labeled on central servers or stored metadata in blockchain to help researchers to search, share, publish datasets or to cooperate with each other. This dream is beyond our limitation and the frame of this hackathon, but it would be good if one day it will be born.

About us



Axel ORSZÁG-KRISZ Dr.

<https://hyperrixel.com/en/axel>



Richárd Ádám VÉCSEY Dr.

<https://hyperrixel.com/en/richard>

We are two deep learning developers and data scientists. We work a lot of different field of AI developing, but our main objective is a computer vision in healthcare. Nowadays, most companies like to hire developers who are outsiders, who have other skills and education. We are lawyers who have jumped from law to programming and deep learning. We had a lot of opportunity from Facebook and Udacity. These companies maintain scholarship programs to get knowledge people around the world. We were two of them. Computer Vision Nanodegree or Deep Learning Nanodegree were supported by Facebook. Besides the knowledge, we got a lot of new viewpoint and the possibility to help others. We are thankful and during the COVID-19 epidemic, we are trying to get back something to the community and to the people around the Globe. Under these sign, we created the following projects.

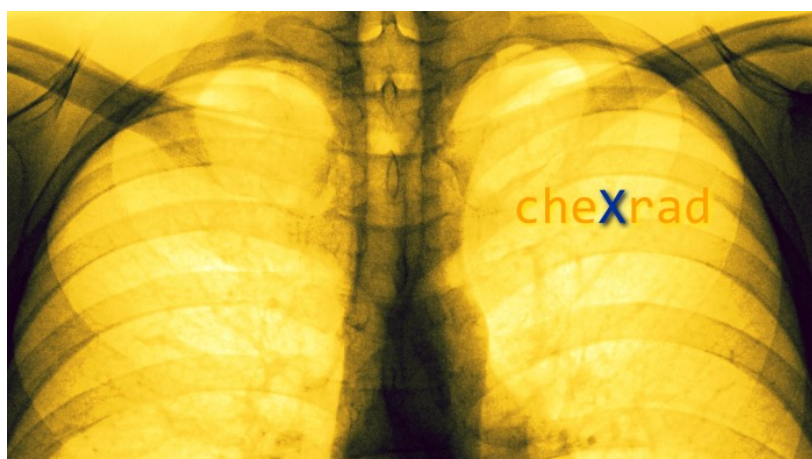


CheXrad is pulmonary application with 4 different artificial intelligence models. It can be trained very fast with CoCoLo technology. The software's goal is to support doctors' decisions with analyzing x-ray images.

Each artificial intelligence network is trained separately so their knowledge can be specialized or generalized as well. The neural network analyze the content of an image independently from each other too. The result can be a simple decision about the X-ray picture: healthy – not healthy. In case of illness the concrete disease is also communicated.

The four separated neural networks work like healthcare counselor group of several specialists which only purpose is to support the doctor. It gives not only a second opinion but also third and fourth opinion of the actual case. The result of each network will be evaluated and visualized after the decision process. Multi-level hierarchy is the key so the computer could decide even if it did not see the actual problem during the training process.

Logo:



Repository is available here:

<https://github.com/hyperrixel/cheXrad>

COVID-19 dataset

We download the daily dataset from ECDC (European Centre for Disease Prevention and Control) and make a standardized csv from this dataset. It contains the additional and cumulated confirmed cases and death number per days and countries. The standardized structure supports the fast processing. It can be filtered fast and easily. Every common programs and modules for datascientists or journalists be able to handle csv format from pandas in python through R till Excel.

Dataset contains 6 columns in the following order:

date | areacode | cumulated cases | cumulated death | additional cases | additional death

Example:

date	areacode	cumulated cases	cumulated death	additional cases	additional death
2020-03-29	20	536	30	41	6
2020-03-29	27	1187	2	17	0
2020-03-29	30	1061	32	95	4
2020-03-29	31	9762	639	1159	93
2020-03-29	32	9134	353	1850	64
2020-03-29	33	37575	2314	4611	319
2020-03-29	34	72248	5690	8189	832

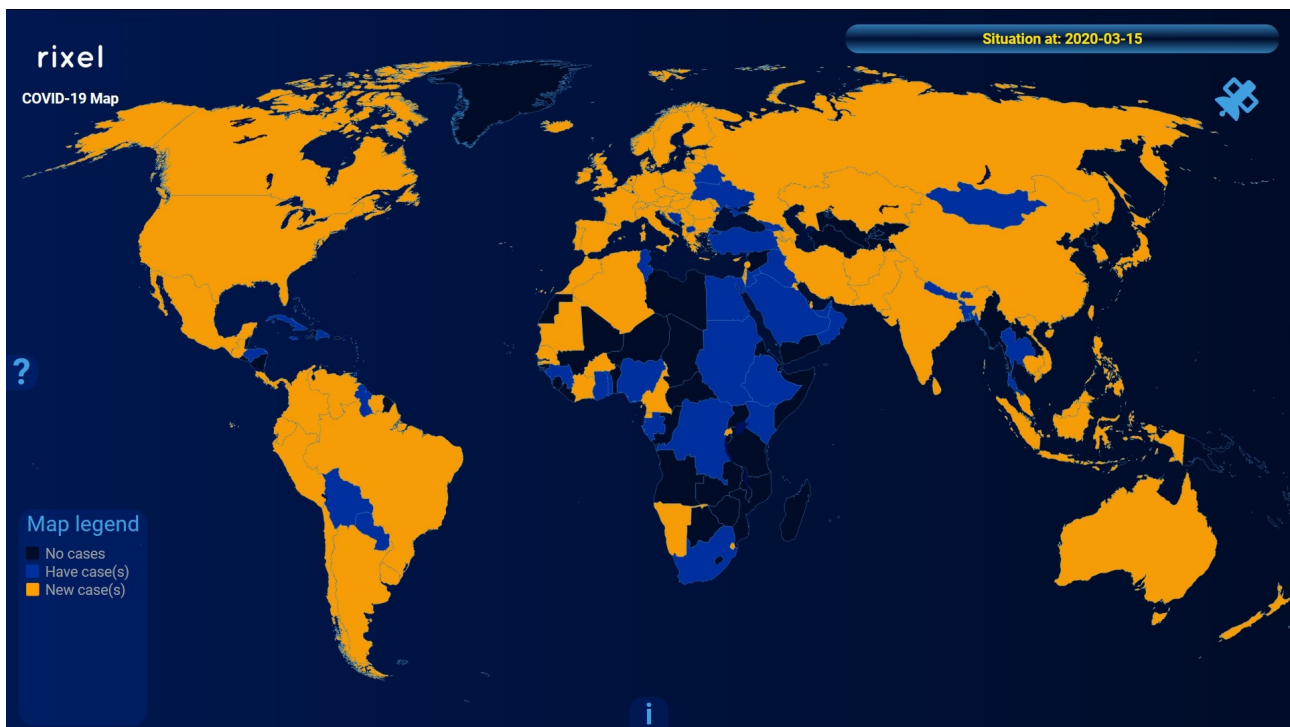
Repository is available here:

<https://github.com/hyperrixel/covid19map-dataset>

COVID-19 map

Besides the daily dataset-making routine, we wanted to create a simple visualization tool that helps people to understand the raw numbers and add visualization ideas to other datascientists. There are a lot of techniques to process an epidemic dataset, we show an example one of them.

Screenshot:



Visualization is available here:

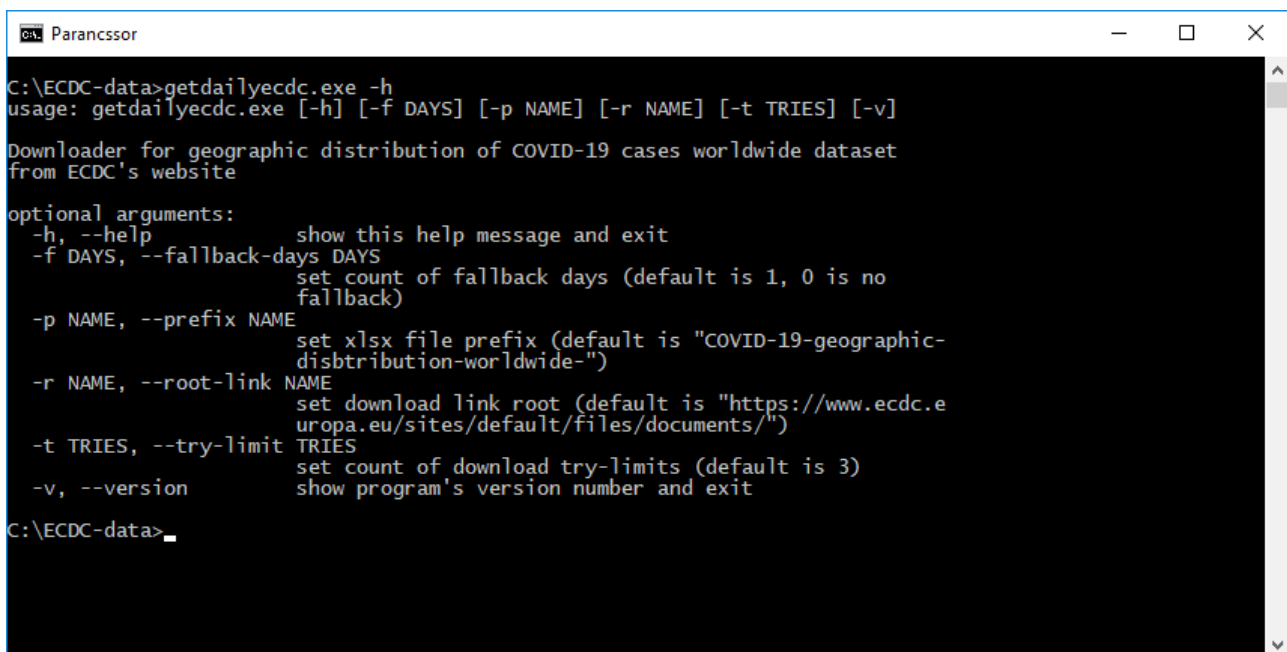
<https://www.hyperrixel.com/covid19map/>



COVID19CasesWorldwideDatasetFromECDC

The goal of this project is to provide downloader and loader for geographic distribution of COVID-19 cases worldwide dataset from ECDC's website. This project includes multiple files. These can be used as module or as script based on the user's need. As a module `eccdloader.py` downloads the latest dataset and loads it into pandas dataframe. As a script it just downloads the latest dataset. It is possible to finetune the parameters with `argparser`. This method works with the executable file too. For researchers who are not familiar with python, we provide an executable version of script.

Screenshot:



```
C:\ECDC-data>getdailyecdc.exe -h
usage: getdailyecdc.exe [-h] [-f DAYS] [-p NAME] [-r NAME] [-t TRIES] [-v]

Downloader for geographic distribution of COVID-19 cases worldwide dataset
from ECDC's website

optional arguments:
  -h, --help            show this help message and exit
  -f DAYS, --fallback-days DAYS
                        set count of fallback days (default is 1, 0 is no
                        fallback)
  -p NAME, --prefix NAME
                        set xlsx file prefix (default is "COVID-19-geographic-
                        disbtribution-worldwide-")
  -r NAME, --root-link NAME
                        set download link root (default is "https://www.ecdc.e
                        uropa.eu/sites/default/files/documents/")
  -t TRIES, --try-limit TRIES
                        set count of download try-limits (default is 3)
  -v, --version          show program's version number and exit

C:\ECDC-data>
```

Repository is available here:

<https://github.com/hyperrixel/COVID19CasesWorldwideDatasetFromECDC>