Python Software Foundation - Activeloop Hub GSOC Proposal 2021 - Eshan Arora

Project Description:

Automatic Generation of schema from directory

Hub is an open source package that enables Data Scientists to stream their datasets of all kinds directly to Hub. Hub aims to reduce time spent by researchers figuring out data and enable them to spend more time working on their Data Science models. Hub is capable of handling petabyte-scale datasets at great speeds by chunking data. This gives the researchers the freedom to work on their datasets from any machine.

This project aims to Auto generate Schema for a dataset when pointed to a directory, instead of having to manually define it. I am passionate and deeply committed to Hub.auto, as **I believe anything that can be automated, should be automated** and thus we will eliminate the need for manually defining schemas.

Approaching The Problem:

Schemas define the structure of the dataset. An Auto generator of Schemas should ideally follow a pipeline:

- Indexing folders from 0 to n in subdirectory
- Parse through all types of data
- Detect anomalies in dataset (use of human element)
- Have a tag for dataset (train/test/val)
- Define an accurate schema for the dataset

Currently, schema is coded in as follows:
For a labelled image dataset:

```
my_schema={"image": schema.Image(max_shape=(28,28,3), dtype="uint8"),
           "label": ClassLabel(num_classes=2)}
```

The proposed solution with Hub.auto creates this schema automatically, after detecting the types of files present in the directory. Currently Hub supports a wide range of datasets: Text, Image, Audio, ClassLabel, Polygon, Video, Bbox, Tensor, Mask, Segmentation, Sequence. The project will be able to effectively create datasets for all the above listed data types.

Positives of the existing app:

- Schemas work as desired when defined by the User
- Works with all data types (Images, ClassLabel, Text, Audio, etc.)
- The companion web app at app.activeloop.ai works in sync with schema.
- Minor progress of parsing through data in Hub.auto for (Image classification, Tabular and Audio)

Flaws that will be corrected by Hub.auto:

- Writing Schemas by users is cumbersome
  - Eliminating the learning curve with Hub.auto would **enable faster onboarding** of users to Hub.
- Hub does not treat **Train, Test and Validation** datasets differently.
  - Schemas do not know how to differentiate between datasets. Sub-schemas which can be created with an argument in function. (More on this below)
- Schemas are currently very rigid in the way they are defined and used.
  - A schema when defined for a particular dataset cannot do more than just defining the structure of the dataset.
  - With Hub.auto, users can ask Hub to **extract datasets from directories/ sub-directories** on command and in real-time. Hub.auto can also enable split dataset into train/test and auto generate schemas.
- Schemas cannot be edited once declared, unless manually rewritten.
  - If an edit has to be made to a hub dataset which requires a change in one of the arguments (for eg: max_shape). The current schema would reject such a request.
  - Hub.auto aims to **dynamically adapt and edit the current schema** to incorporate the same.

Adapting to the new codebase:

With the current situation of Hub's codebase and the recent announcement of "REBUILD DA CODE". Timeline for all features including this project Auto generation of Schema will stay flexible as the future of hub's codebase will be decided in the upcoming weeks (month of April). I understand that some features and implementation details are subjected to change with the guidance from mentors. The core functionality (deliverables) of parsing, detecting the sub directories and generation of schemas with unit tests and documentation must be completed within the time frame of the GSOC program.

Deliverables:

The goal is to create an end-user solution to generate schemas automatically:

- Implement parsing code for all datatypes (Images, Audio, Text, Tensor, etc.)
- Implement API to detect dataset through directories with multiple datasets
  - Should differentiate between pointed directories
  - Use of indexing for identifying dataset
- Implement functionality to differentiate between train, test and validation dataset
  - Use of tagging to identify the type of dataset
  - **Why tags? Use of tags is the easiest method for both the user and a future contributor to understand what is happening**
- Implement code for Human element to correct boundary cases
  - This feature is to be discussed with the mentors, as i believe eliminating manually declared schema is the aim
- Write unit tests for API
- Write documentation for the Auto schemas generator

Implementation:

- **Implement parsing code for all datatypes (Images, Audio, Text, Tensor, etc.)**

  Reference Issues: #696 #771 #763

I will be adding code for parsing through all kinds of data. Currently the Image and csv have been implemented to a certain extent. To demonstrate my sound knowledge of this task I am adding the code which I have been working on for detecting Audio types below. This piece of code is to be located at Hub/hub/auto/audio/audio.py.

```
import os

import hub
import numpy as np
from hub.auto import util
from hub.auto.infer import state
from tqdm import tqdm

USE_TQDM = True


@state.directory_parser(priority=2)
def data_from_audio(path, scheduler, workers):
# Return error if package not found
   try:
       import librosa
   except ModuleNotFoundError:
       raise ModuleNotInstalledException("librosa")

   try:
       import pandas as pd
   except ModuleNotFoundError:
       raise ModuleNotInstalledException("pandas")




if not util.files_are_of_extension(path, AUDIO_EXTS):
       return None

   max_shape = (0,)
# Use dataframe to store data
   df = pd.DataFrame()
   files = util.get_children(path)
# Parse through audio files
   for audio_file in files:
       audio, sr = librosa.load(audio_file)
       df = df.append({"Audio": audio, "Sampling Rate": sr}, ignore_index=True)
       shape = audio.shape
       max_shape = np.maximum(max_shape, shape)

   max_shape = tuple([int(x) for x in max_shape])
# Define Schema
   schema = {
       "audio": hub.schema.Audio(shape=(None,), dtype="uint8", max_shape=max_shape),
       "sampling_rate": hub.schema.Primitive(dtype=int),
   }
# Convert to Hub format using transform
   @hub.transform(schema=schema, scheduler=scheduler, workers=workers)
   def upload_data(index, df):
       audio_dictionary = {}
       for column in df.columns:
           audio_dictionary[column] = df[column].iloc[index]
       return audio_dictionary

   return upload_data(range(len(df)), df=df)
```

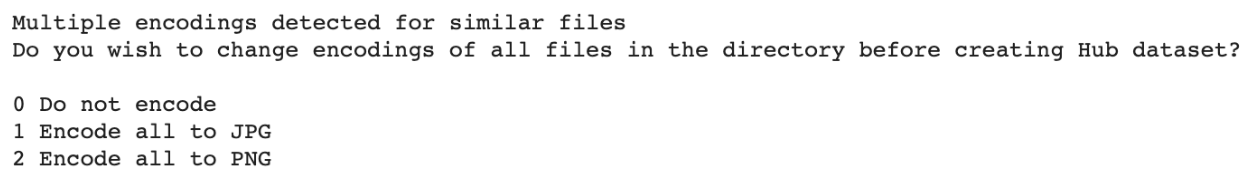
The final result should be able to parse through any kind of data in accordance to its priority digit. If Hub fails to detect the kind of data after checking for all supported types, an error should be returned. The error handling will be discussed with mentors.

- ## Use of Human element in the loop

When met the following issues in during parsing data from a directory, it is important to ask the user to generate an accurate Schema:

  - If the file extension is of the **same class** (Image) but different in formatting (JPG, PNG, BMP, etc.). An immediate encoding option must be provided to the user.
    - User can either be asked if he wants the Images to be encoded all to either of the detected formats or if they wish to ignore the formats. The users are given a choice. Should they choose to encode it to one of the detected formats. A**ll detected files must be encoded to the said format and Hub.auto should continue to function.**

Please see Image for reference:

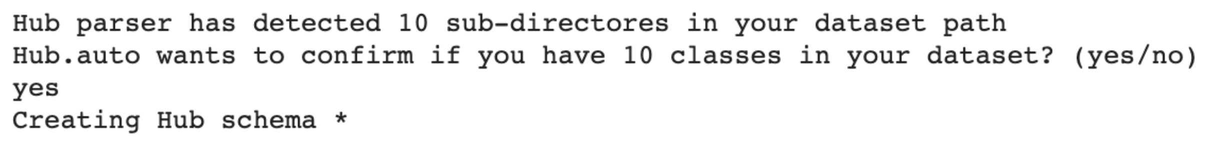


```
Multiple encodings detected for similar files
Do you wish to change encodings of all files in the directory before creating Hub dataset?

0 Do not encode
1 Encode all to JPG
2 Encode all to PNG
```

  - If the parser meets with a dilemma of treating a sub-directory as a Class label or not. Human element might be used by asking similar questions as above. [up for more discussion with mentors]

Please see Image for reference:



```
Hub parser has detected 10 sub-directores in your dataset path
Hub.auto wants to confirm if you have 10 classes in your dataset? (yes/no)
yes
Creating Hub schema *
```

- **Implement API to detect dataset through directories with multiple datasets**
    - Resources link

Currently, hub.auto parses data from `*directory_parsers*`. This piece of code at /Hub/hub/auto/infer.py checks for parsers in the `*directory_parsers*`. However it is not possible to select and bring up schema for different directories.

Functionality added will work like this:

```
# Bring up schema for any dataset

ds = hub.Dataset.from_path(dataset_path = "dataset directory", index = 0 )
```

Index will refer to the folder to be searched in the given directory.

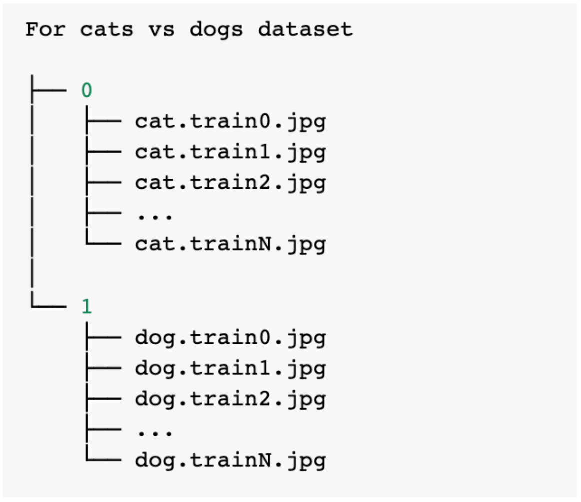
Assumption for now: The function is not obligated to look for datasets within sub-subdirectories (3 levels) as it is the user's duty to provide the correct `*dataset_path*` with subdirectories upto 2 levels. This feature can be worked at a later time interval, once basic functionality is up and running.

`from_path` methods will create indexes of folders before any parsing takes place.

```
@staticmethod
def from_path(path, scheduler="single", workers=1, index):
    # infer schema & get data (label -> input mapping with file refs)

    # TODO: search with index
    ds = auto.infer_dataset(path, scheduler=scheduler, workers=workers, index)
    return ds
```

To improve usability, an output of indexes will be displayed to the user in the terminal which displays the folder is indexing as such (for cats vs dogs dataset stored in different directories):



```
For cats vs dogs dataset

├── 0
│   ├── cat.train0.jpg
│   ├── cat.train1.jpg
│   ├── cat.train2.jpg
│   ├── ...
│   └── cat.trainN.jpg
│
└── 1
    ├── dog.train0.jpg
    ├── dog.train1.jpg
    ├── dog.train2.jpg
    ├── ...
    └── dog.trainN.jpg
```

(Image for reference)

This will resolve the indexing issue and multiple directories can be accessed separately with just a change of the argument.

Additionally, code for index based collecting of files must be added here:

```python
def infer_dataset(path, scheduler="single", workers=1, index):
    # TODO: handle s3 path

    if not os.path.isdir(path):
        raise Exception("input path must be either a directory")

    hub_path = os.path.join("./", path, "hub")

    if os.path.isdir(hub_path):
        print('inferred dataset found in "%s", using that' % hub_path)
        return hub.Dataset(hub_path, mode="r")

    root = _find_root(path)
    ds = None

    directory_parsers = state.get_parsers()
    if len(directory_parsers) <= 0:
        raise Exception("directory parsers list was empty.")

    for parser in directory_parsers:
        # TODO: look for folders based on index value


        ds = parser(root, scheduler, workers, index)
        if ds is not None:
            break

    if ds is None:
        raise Exception(
            'could not infer dataset for the root "%s". either add a new parser to'
            % root
            + "`hub.schema.auto.directory_parsers` or write a custom transform + schema."
        )

    ds.store(hub_path)
    return hub.Dataset(hub_path, mode="r")
```

- ## Implement functionality to differentiate between train, test and validation dataset

Add another argument in the method type, The user has three options for the dataset to be treated differently.

- Train
- Test
- Validation

```python
# Bring up schema for any dataset

ds = hub.Dataset.from_path(dataset_path = "dataset directory", index = 0, type = "train" )
```

```python
def infer_dataset(path, scheduler="single", workers=1, index, type):
    for parser in directory_parsers:
        # TODO: look for folders based on index value


        ds = parser(root, scheduler, workers, index, type )
        if ds is not None:
            break

    if ds is None:
        raise Exception(
            'could not infer dataset for the root "%s". either add a new parser to'
            % root
            + "`hub.schema.auto.directory_parsers` or write a custom transform + schema."
        )

    ds.store(hub_path)
    return hub.Dataset(hub_path, mode="r")
```

```python
@staticmethod
def from_path(path, scheduler="single", workers=1, index, type):
    # infer schema & get data (label -> input mapping with file refs)

    # TODO: search with index
    ds = auto.infer_dataset(path, scheduler=scheduler, workers=workers, index, type)
    return ds
```

A train-test split function can also be incorporated from `*scikitlearn*` directly into Hub, which would enable creation of both the dataset and the respective schema. Transforms could also be incorporated for Image resizing/rotation etc. or Text pre-processing with tokenization, noise removal, handling cases, etc.

This idea needs to be further discussed with the mentors to know if it is in scope of this GSOC project,
As of now there is no way for the user to know which dataset in Hub is having a train tag or a test tag. Users have to create multiple datasets with different names.

`*type*` is an arbitrary name. Final name of the argument will be discussed with the mentors. Although it adds no new function to the dataset, it will allow the user to create a dataset schema with a unique tag to differentiate between the parts of a single dataset.

An alternate approach would be to ask the user to explicitly rename the folders to "train" and "test". If Hub.auto finds folder names as such, it would treat the dataset accordingly. However if folders are named arbitrarily, the user is offered to split it to train and test before a schema is created.
Train and Test datasets could be helped by keys instead of an argument in the function.

Ideally, the sub-datasets with tags "train", "test" or "validation" must be stored in the same dataset. Currently it is not possible to do this with Hub. However it is to be discussed with the mentors if keeping the dataset under such tags is possible. This would allow easy visualization as follows:
the user can login to web app>pick dataset>choose tag "train">view train dataset.
Note that this is not a sub-dataset, it is a single dataset but split into 3 parts and the schema would stay the same apart from the `type` tag and `max_shape`.

- **Write unit tests for API**
    - I will write tests for all the code I will contribute in this project and make sure auto generation of schemas work smoothly and handle multiple boundary cases including Multiple data types detected in `*dataset_path*`
    - Multiple data types within a single dataset_path can confuse Hub.auto as to what schema is being considered. A human interaction can be added to the loop by asking the user if he/she is looking for a specific data type to be considered.

- **Write documentation for the Auto schemas generator**
    - Currently all documentation is written with sphinx, Hub 2.0 will be using gitbook and I would contribute to adding documentation for the Auto schema generator. I have previously contributed to writing documentation for Hub. **All my contributions are listed below.**

Management of Coding Project:

- I will take active participation in the development of Hub.auto in accordance with the new code base of Hub 2.0.
- From recent talks with Dyllan and Davit, Hub 2.0 wants the schemas to be exclusively automatic for the most part. Hence I will be a part of REBUILD DA CODE from April itself so that I am in sync with new ideas and have a thorough understanding of the new codebase.
- Every week, I will discuss my progress and project development with my mentors.
- Will regularly tweet my learnings and work done during the GSOC period.
- I will write a blog about my experience contributing to Hub at the end of the summer.
- In the worst case scenario, the time allotted for writing documents may be used for other purposes in the project. However, I assure you that I will put my full effort to avoid any such situation.
- Finally, I will create a presentation/report of my work done during the GSOC period at Hub.

Timeline: **[To stay flexible, parsing could be implemented at an earlier interval upon discussion with mentors]**

- **Community Bonding Period (May 17 - June 7):**
    - Finalize the structure for Hub 2.0
    - Discuss with the mentor, a plan for going about the implementation of Hub.auto

    - Enhance my knowledge on the path of the new codebase
    - If there are more tags needed to produce accurate datasets, except `type` and `index`.
    - This period can be used to plan, finalize the parts of the proposal that are not very clear
    - Figure out Clustered Hierarchical Chunking (CHC) for Hub 2.0
- **Week 1 - 2 (June 7 - June 21)**
    - Implement index based directory searching for all folders present in a directory
    - Implement of display of directories with tag `*index*` = 0,1,etc.
    - Write test for the same
    - Clean up code

- **Week 3 - 4 (June 21 - July 5)**
    - Implement train/ test/ val tag for schema datasets
    - Implement output of directory structure to the user in the terminal
    - Write tests for redundancy and boundary cases
    - Clean up code and documentation

- **Week 5 (July 5 - July 12)**
    - Discuss with mentor and Implement if any changes that need to be done
    - Run multiple test cases
    - Make the feature robust and error free
    - Clean up code, improve code quality and unit tests
    - Prepare for evaluation

- **Evaluation July 12 - July 16**

- **Week 6 - 7 (July 17 - July 31)**
    - Implement parsing code for all data types supported by Hub Schemas with `index` and `type` tags.
    - Write unit tests for all parsers
    - Integrate Human element in the loop with tests

    - Complete testing and make the code error free

- **Week 8 (July 31 - Aug 6)**
    - Finalize boundary cases for Auto schema
    - Clean up code and improve code quality
    - Implement documentation of Hub.auto

- **Week 9 (Aug 6 - Aug 16)**
    - Complete final documentation of Hub.auto
    - Final unit tests
    - Prepare and Submit Final Evaluation

- **Final Evaluation**

Open Source contributions:

Hub

- Add Auto schema for Audio files
    - Link to PR - #763
- Add Working with Images documentation and Google Colab file
    - Link to PR - #743
    - Link to Colab file - link
- Add Schema documentation for Images, Audio, Text, Bbox,
    - Link to PR - #654, please find my work on Activeloopai's official docs here
    - Link to Colab file - link
- Hub v0 contributions
    - Link to PR - #95
    - Link to Colab file - link

Personal Information:

Name: Eshan Arora
University: Sardar Vallabhbhai National Institute of Technology, Surat
Email: thisiseshan@gmail.com
Personal Website: https://thisiseshan.dev
GitHub: https://github.com/thisiseshan
LinkedIn: https://www.linkedin.com/in/thisiseshan/
Resume: link
Location: Muscat, Oman
Timezone: GST (UTC + 4:00)

Previous work:

I am a self-taught Machine Learning developer, with previous experience in Internships at G4S and Universal E-Business Solutions. I am passionate about computer vision and have worked on 2 research projects in CV:

- MusicGAN - Generate Music from Music using Generative Adversarial Networks (link)
- Colorization of Grayscale Images using GANs - Use of U-Net architecture in GANs to produce colorized images (present)

Additionally, I have self-taught myself through multiple online courses on Coursera [Details in Resume].
I have experience in developing Decentralized Apps on the Ethereum blockchain Blockchain using Solidity and Stellar Blockchain.
When I started out programming in High School I developed and published 2 Mobile Apps on the AppStore and 1 App on Play Store. (Developed my me)

- ISWK - AppStore Play Store [1000+ downloads]
- Chatty Stickers - AppStore

Obligations/Schedule Conflicts:

I have **NO** obligations this summer and will be able to devote my full time to work on this project. I will be able to contribute 40+ hours every week towards this project. Due to travel to India in June or July (uncertainty due to the pandemic), the number of hours I am able to spend everyday might reduce for a short time but I will make up for them on weekends and make sure to give my 100% in completing this project.

Why Hub?

Hub and I share common interests in the field of data handling on the go. Python is my go-to language and since Hub is almost completely built on Python. I also believe that this is a great organization that will help me kick off with GSOC! I want to increase my open source contribution and level myself up as a Machine Learning Developer. I am familiar with its code base and my recent talks with wonderful people such as Dyllan and Davit have made me confident that this is where I want to spend my time contributing and refining my skills.

Post GSOC work:

Once I am done with this project, I will complete my documentation if it is pending, explaining what every step in my code is doing. I will also encourage my classmates and juniors to contribute more towards Open Source and will share my experience with the world in a blogpost about my journey of GSOC 21 with Hub.
