dtoolAI

Release 0.1.0

May 20, 2020

Contents:

1 Introducing dtoolAI **1**
- 1.1 What is dtoolAI? . . . 1

2 Installation **3**
- 2.1 Installing with `pip` . . . 3
- 2.2 Installing with `conda` . . . 4

3 Using a trained network model **5**
- 3.1 Download the scripts needed for this tutorial . . . 5
- 3.2 Applying the network to new data . . . 5
- 3.3 Finding out about the network . . . 6

4 Training a new model **7**
- 4.1 The dataset . . . 7
- 4.2 Training a network . . . 7
- 4.3 dtoolAI and URIs . . . 8
- 4.4 Applying the trained model to test data . . . 8
- 4.5 Viewing the trained model metadata . . . 8
- 4.6 What the code is doing . . . 9

5 Retraining a model **11**
- 5.1 Part 1: With a preprepared dataset . . . 11
- 5.2 Part 2: With raw data . . . 12

6 Extending dtoolAI **15**
- 6.1 New forms of training data . . . 15

7 API documentation **17**
- 7.1 dtoolai.data . . . 17
- 7.2 dtoolai.parameters . . . 18

8 Indices and tables **19**

Python Module Index **21**

Index **23**

CHAPTER 1

Introducing dtoolAI

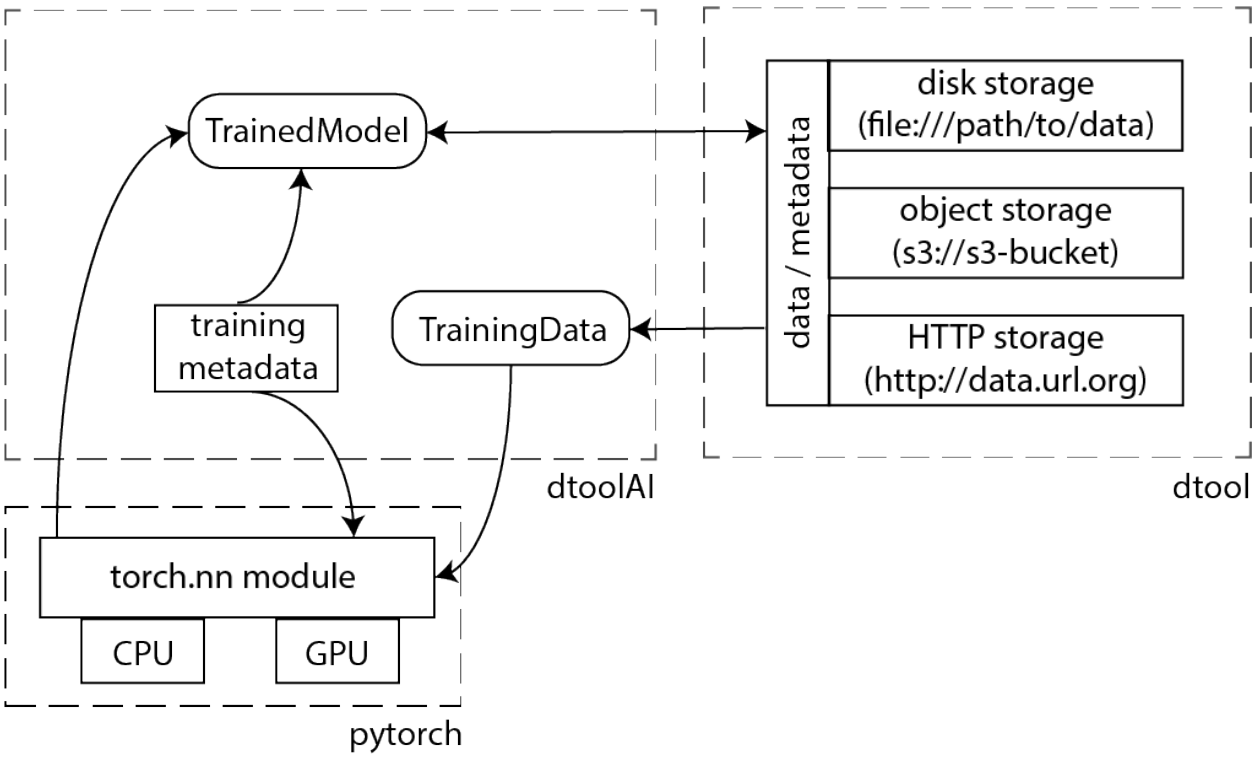
1.1 What is dtoolAI?

dtoolAI is a Python library to make reproducible AI model training and use easier. The dtoolAI package provides:

- A Python API to a set of classes and helper functions for managing Deep Learning model data, training and use.
- Scripts and command line functions for demonstrating use and automating common tasks.
- Documentation in the form of both these documents and Jupyter notebooks that show how to use the library.

In general, the documentation and scripts use image recognition to demonstrate the library, but the lower level functions for packaging data and models, as well as capturing training metadata can be used for a wide range of problem domains.

dtoolAI makes use of dtool, a library for lightweight data management to work with different data sources such as S3, Azure, HTTP and local filesystem. dtoolAI uses pytorch for implementation of AI models.


TrainedModel
training metadata
TrainingData
dtoolAI
data / metadata
disk storage (file:///path/to/data)
object storage (s3://s3-bucket)
HTTP storage (http://data.url.org)
dtool
torch.nn module
CPU
GPU
pytorch

CHAPTER 2

Installation

dtoolAI requires Python version 3 and Pytorch.

Warning: Install Pytorch before installing dtoolAI. For information on how to install Pytorch this see the Pytorch getting started guide for details. Version 1.4.0 of Pytorch and 0.5.0 of torchvision are definitely compatible with dtoolAI.

For Windows users, we'd recommend using `conda` to instal pytorch and torchvision, as per instructions below.

2.1 Installing with `pip`

You can install dtoolAI via the pip package manager:

```
pip install dtoolai
```

To understand the examples, it's also useful to install the dtool meta package. This makes it easier to work with datasets created by dtoolAI:

```
pip install dtool
```

Running the example notebooks in the code repository also requires Jupyter:

```
pip install jupyter
```

Finally, if you want to run the test suite in the code repository, you'll need pytest.

```
pip install pytest
```

2.2 Installing with conda

You can install dtoolAI with conda as follows:

```
conda install pytorch==1.4.0 torchvision==0.5.0 -c pytorch
conda install dtoolcore dtool-http dtoolai -c dtool
```

This first installs a version of Pytorch known to work with dtoolAI. If you would like to install the whole `dtool` command line suite, you'll need to use pip:

```
pip install dtool
```

CHAPTER 3

Using a trained network model

In this first example, we'll look at how to apply a trained network to an image that's new to the network. We'll then look at how dtoolAI allows us to find out information about the data on which the model was trained and how it was trained.

3.1 Download the scripts needed for this tutorial

The scripts for this tutorial can be found in https://github.com/JIC-CSB/dtoolai. The easiest way to get access to them is to clone the git repository.

```
$ git clone https://github.com/JIC-CSB/dtoolai.git
```

The examples in this documentation assumes that you are working from within the downloaded git repository. The command below updates the working directory to this location.

```
$ cd dtoolai
```

3.2 Applying the network to new data

Let's start by trying to classify a new image. The image below is available in `./docs/source/non_mnist_three.png`.

Now run the script `apply_model_to_image.py` in the `scripts/` directory of the dtoolAI repository on the image, e.g.:

```
$ python scripts/apply_model_to_image.py http://bit.ly/2tbPzSB ./docs/source/non_
↪mnist_three.png
Classified /Users/hartleym/Downloads/three.png as 3
```

We've applied an existing model to a new image.

3.3 Finding out about the network

We can also find out about the network and how it was trained. For this, we'll use the command `dtoolai-provenance` that's provided when you install the dtoolAI package. This command displays data about a trained model including the training data URI. It then attempts to follow that URI to give more information about the training data:

```
$ dtoolai-provenance http://bit.ly/2tbPzSB
Network architecture name: dtoolai.simpleScalingCNN
Model training parameters: {'batch_size': 128,
'init_params': {'input_channels': 1, 'input_dim': 28},
'input_channels': 1,
'input_dim': 28,
'learning_rate': 0.01,
'loss_func': 'NLLLoss',
'n_epochs': 10,
'optimiser_name': 'SGD'}
Source dataset URI: http://bit.ly/2NVFGQd
Source dataset name: mnist.train
Source dataset readme:
---
dataset_name: MNIST handwritten digits
project: dtoolAI demonstration datasets
authors:
- Yann LeCun
- Corinna Cortes
- Christopher J.C. Burges
origin: http://yann.lecun.com/exdb/mnist/
usetype: train
```

Here we see that model's network architecture is `simpleScalingCNN` from the dtoolAI package, some more information about the training parameters then, at the bottom, some information about the training data for the model.

Next, we'll look at how to train a model like this one.

CHAPTER 4

Training a new model

In this example we'll look at one of the "hello world" example problems of training deep learning networks - handwritten digit recognition. We'll use the MNIST dataset, consisting of 70,000 labelled handwritten digits between 0 and 9 to train a convolutional neural network.

4.1 The dataset

In this case, we've created a dtool DataSet from the MNIST data. We can use the dtool CLI to see what we know about this DataSet:

```
$ dtool readme show http://bit.ly/2uqXxrk
---
dataset_name: MNIST handwritten digits
project: dtoolAI demonstration datasets
authors:
- Yann LeCun
- Corinna Cortes
- Christopher J.C. Burges
origin: http://yann.lecun.com/exdb/mnist/
usetype: train
```

This tells us some information about what the data are, who created them, and where we can go to find out more.

4.2 Training a network

We'll start by using one of the helper scripts from dtoolAI to train a CNN. Later, we'll look at what the script is doing.

```
mkdir example
python scripts/train_cnn_classifier_from_tensor_dataset.py http://bit.ly/2uqXxrk␣
↪example mnistcnn
```

This will produce information about the training process, and then report where the dataset with the trained model weights have been written, e.g.:

```
Wrote trained model (simpleScalingCNN) weights to file://N108176/Users/hartleym/
↪projects/ai/dtoolai-p/example/mnistcnn
```

4.3 dtoolAI and URIs

In the example above, when we specified where the trained model should be written, we provided two parameters to the script with values `example` and `mnistcnn`. The second of these, `mnistcnn` gives the name of the output model, the first `example` is a **base URI**. This concept is explained in more detail in the dtool documentation, we'll give a short summary here.

In general when we create model training datasets and trained models, we want to store these in permanant HTTP accessible object storage with persistent URIs. However, since this requires setting up Amazon S3 or Microsoft Azure storage credentials, for simplicity we'll work with filesystem URIs in these examples. URIs on filesystem disk are something of a special case. Properly qualified file URIs have a form like the example above:

file://N108176/Users/hartleym/projects/ai/dtoolai-p/example/mnistcnn

For convenience's sake, we allow file URIs to be expressed as filesystem paths. As such the URI above can be simplified to `./example/mnistcnn`, and dtool will internally convert this into a full URI.

4.4 Applying the trained model to test data

The simplest way to test our model is on another preprepared dataset - this allows us to quickly apply the model to many ready-labelled images and calculate its accuracy.

We have provided the MNIST test data as a separate dtool DataSet for this purpose, and we can apply our new model to this dataset like this:

```
$ python scripts/apply_model_to_tensor_dataset.py \
    ./example/mnistcnn http://bit.ly/2NVFGQd
7929/10000 correct
```

If we want to improve the model's accuracy, we could try training it for longer. For example, to train it for 5 epochs (loops through the training dataset) rather than one, we can run our script again:

```
$ python scripts/train_cnn_classifier_from_tensor_dataset.py \
    http://bit.ly/2uqXxrk example mnistcnn_epochs_5 --params n_epochs=5
```

This will train the model for longer.

4.5 Viewing the trained model metadata

One of the core features of dtoolAI is capture of references to training data and metadata about the training process. Let's look at how we access those captured data for our newly trained model.

dtoolai provides a helper script, `dtoolai-provenance` for this purpose. This will show a model's training metadata, the references to its training data, then the metadata for those training data.

```
$ dtoolai-provenance example/mnistcnn/

Network architecture name: dtoolai.simpleScalingCNN
Model training parameters: {'batch_size': 128,
'init_params': {'input_channels': 1, 'input_dim': 28},
'input_channels': 1,
'input_dim': 28,
'learning_rate': 0.01,
'n_epochs': 1,
'optimiser_name': 'SGD'}
Source dataset URI: http://bit.ly/2uqXxrk
Source dataset name: mnist.train
Source dataset readme:
---
dataset_name: MNIST handwritten digits
project: dtoolAI demonstration datasets
authors:
- Yann LeCun
- Corinna Cortes
- Christopher J.C. Burges
origin: http://yann.lecun.com/exdb/mnist/
usetype: train
```

We can see that the model dataset contains both information about how the model was trained (learning_rate, n_epochs and so on) as well as the reference to the training data, which we can follow to show its provenance.

4.6 What the code is doing

We provide the Jupyter notebook TrainingExplained.ipynb to show how the training script uses dtoolAI's library functions and classes to make capturing training metadata and parameters easier. This notebook's available here, or if you have a local copy of the dtoolAI repository, in the `notebooks` directory.

CHAPTER 5

Retraining a model

Deep learning models are powerful, but can be slow to train. Retraining let us take a model that has already been trained on a large dataset and provide it with new training data that update its weights. This can give accurate models much faster than training from scratch, with less data.

Let's look at how to do this using dtoolAI.

In this example, we'll take a model called ResNet, that's been trained on a large image dataset, and retrain it to classify new types of images that the network has not seen before.

5.1 Part 1: With a preprepared dataset

In this example, we'll use the CalTech 101 objects dataset. We provide a hosted version of this dataset in a suitable format. If you have the `dtool` client installed, you can view information about this hosted dataset like this:

```
$ dtool readme show http://bit.ly/3aRvimq

dataset_name: Caltech 101 images subset
project: dtoolAI demonstration datasets
authors:
- Fei-Fei Li
- Marco Andreetto
- Marc 'Aurelio Ranzato
reference: |
L. Fei-Fei, R. Fergus and P. Perona. One-Shot learning of object
categories. IEEE Trans. Pattern Recognition and Machine Intelligence.
origin: http://www.vision.caltech.edu/Image_Datasets/Caltech101/
```

This version of the CalTech data contains just two object classes - llamas and hedgehogs. We'll train a network to be able to distinguish these.

5.1.1 Retraining the model

Since we have data available, we can immediately run the retraining process. dtoolAI provides a helper script to apply its library functions for retraining a model and capturing metadata:

```
$ mkdir example
$ python scripts/retrain_model_from_dataset.py http://bit.ly/3aRvimq example hlama
```

After some information about the training process, you should see some information about where the model has been written:

```
Wrote trained model (resnet18pretrained) weights to file://N108176/Users/hartleym/
↪projects/ai/dtoolai-p/example/hlama
```

5.1.2 Applying the retrained model to new images

Let's evaluate the model. We can first try evaluation on a held-out part of our training dataset. This dataset contains metadata labelling some parts of the dataset as training data and some as evaluation data. Our evaluation script takes advantage of this labelling to score the model:

```
$ python scripts/evaluate_model_on_image_dataset.py example/hlama http://bit.ly/
↪3aRvimq
Testing model hlama on dataset caltech101.hedgellamas
23/25 correct
```

Now we can test the newly trained model. Try downloading this image:

https://en.wikipedia.org/wiki/File:Igel.JPG

Then we can apply our trained model

```
python scripts/apply_model_to_image.py example/hlama Igel.JPG
```

5.2 Part 2: With raw data

We saw above how we could retrain a model using data that's already been packaged into a dataset. Now let's look at how we can work with raw data, by first packaging it then applying the same process.

5.2.1 Gathering data

You can use any collection of images. For this example, we'll again use the Caltech 101 objects dataset. which is available here.

Download the dataset somewhere accessible and unpack it.

5.2.2 Converting the data into a DataSet

dtoolAI provides a helper script to convert a set of named directories containing images into a dataset suitable for training a deep learning model.

To use this script, we first need to set up our images in the right layout. The script requires images to be in subdirectories, each of which is named for the category it represents, e.g.:

```
new_training_example/
├── category1
│   ├── image1.jpg
│   └── image2.jpg
├── category2
│   ├── image1.jpg
│   └── image2.jpg
└── category3
    ├── image1.jpg
    └── image2.jpg
```

We can then use helper script provided by dtoolAI, `create-image-dataset-from-dirtree` to turn this directoy into a training dataset.

Assuming that the images are in a directory called `new_training_example`, and that the directory `example` exists and that we can write to this directory, we run:

```
create-image-dataset-from-dirtree new_training_example example retraining.input.
↪dataset
```

or, under Windows:

```
create-image-dataset-from-dirtree.exe
```

This will create a new dataset and report its created URI:

```
Created image dataset at file:///C:/Users/myuser/projects/dtoolai/example/retraining.
↪input.dataset
```

In this example, we're creating the dataset on local disk, so we would need to copy it to persistent world accessible storage (such as Amazon S3 or Azure storage) when we publish a DL model based on this dataset. If you have S3 or Azure credentials set up, you can create persistent datasets directly using the script described above, changing the `example` directory to a base URI as described in the dtool documentation.

5.2.3 Retraining on the new dataset

Now that we've created our training dataset, we can run the same training script that we used above on our new dataset, e.g.:

```
python scripts/retrain_model_from_dataset.py file:///C:/Users/myuser/projects/dtoolai/
↪example/retraining.input.dataset example new.model
```

CHAPTER 6

Extending dtoolAI

dtoolAI provides everything needed to train image classification networks "out of the box". Different types of Deep Learning network will require both new models and possibly classes for training data.

6.1 New forms of training data

dtoolAI provides two classes for managing training data - `TensorDataSet` and `ImageDataSet`. Our examples use these to train models and capture provenance.

The class should:

- Inherit from `dtoolai.data.WrappedDataSet`. This ensures that it provides both the methods required by Pytorch (to feed into the model) and dtoolAI (to capture metadata).
- Implement `__len__` which should return how many items are in the dataset.
- Implement `__getitem__`, which should return either `torch.Tensor` objects or numpy arrays that Pytorch is capable of converting to tensors.

Instances of this class can then be passed to `dtoolai.training.train_model_with_metadata_capture`.

CHAPTER 7

API documentation

7.1 dtoolai.data

class dtoolai.data.**ImageDataSet**(*uri*, *usetype='train'*)
Class allowing a collection of images annotated with categories to be used as both a Pytorch Dataset and a dtool DataSet.

class dtoolai.data.**TensorDataSet**(*uri*)
Class that allows numpy arrays to be accessed as both a pytorch Dataset and a dtool DataSet.

dim
The linear dimensions of the tensor, e.g. it is dim x dim in shape.

input_channels
The number of channels each tensor provides.

class dtoolai.data.**WrappedDataSet**(*uri*)
Subclass of pytorch Dataset that provides dtool DataSet methods.

This class mostly provides methods that consumers of DataSets require, and passes those methods onto its internal DataSet object.

Parameters **uri** – URI for enclosed dtool DataSet.

dtoolai.data.**coerce_to_fixed_size_rgb**(*im*, *target_dim*)
Convert a PIL image to a fixed size and 3 channel RGB format.

dtoolai.data.**create_tensor_dataset_from_arrays**(*output_base_uri*, *output_name*, *data_array*, *label_array*, *image_dim*, *readme_content*)
Create a dtool DataSet with the necessary annotations to be used as a TensorDataSet.

Parameters

- **output_base_uri** – The base URI where the dataset will be created.
- **output_name** – The name for the output dataset.

- **data_array** (*ndarray*) – The numpy array holding data.
- **label_array** (*ndarray*) – The numpy array holding labels.
- **image_dim** (*tuple*) – Dimensions to which input images should be reshaped.
- **readme_content** (*string*) – Content that will be used to create README.yml in the created dataset.

Returns The URI of the created dataset

Return type URI

dtoolai.data.**scaled_float_array_to_pil_image**(*array*)

Convert an array of floats to a PIL image.

Parameters **array** (*np.ndarray*) – Array representing an image. Expected to be float and normalised between 0 and 1.

Returns A PIL Image object created from the array

7.2 dtoolai.parameters

class dtoolai.parameters.**Parameters**(***kwargs*)

Class holding key/value parameter data.

This class is designed to make working with a mixture of JSON, string and object data easier.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

dtoolai.data, 17
dtoolai.parameters, 18

Index

C

coerce_to_fixed_size_rgb() (*in module dtoolai.data*), 17
create_tensor_dataset_from_arrays() (*in module dtoolai.data*), 17

D

dim (*dtoolai.data.TensorDataSet attribute*), 17
dtoolai.data (*module*), 17
dtoolai.parameters (*module*), 18

I

ImageDataSet (*class in dtoolai.data*), 17
input_channels (*dtoolai.data.TensorDataSet attribute*), 17

P

Parameters (*class in dtoolai.parameters*), 18

S

scaled_float_array_to_pil_image() (*in module dtoolai.data*), 18

T

TensorDataSet (*class in dtoolai.data*), 17

W

WrappedDataSet (*class in dtoolai.data*), 17