Dockerizing a PyTorch Application

In this module, we'll take a look at a sample application from the PyTorch website. The sample application does the following:

1. Load and normalizing the CIFAR10 training and test datasets using ``torchvision``
2. Define a Convolutional Neural Network
3. Define a loss function
4. Train the network on the training data
5. Test the network on the test data

We'll Dockerize the application and run it in a container. Next we'll take a look at breaking the pipeline apart into smaller steps. Then we'll build a simple Flask application where we can test our predictions.

Step 1 - Setup our sample application

Open your terminal application and navigate to your working directory. Clone the sample application.

```
$ cd /my/working/directory
$ git clone git@github.com:pytorch/tutorials.git
```

The PyTorch tutorial repository is fairly large, so let's remove the files we do not need.

```
$ mkdir docker-ml && cd docker-ml
$ cp ../tutorials/beginner_source/blitz/cifar10_tutorial.py app.py
```

Now create a requirements.txt file and add our dependencies.

```
$ touch requirements.txt
$ cat <<EOF >> requirements.txt
matplotlib
EOF
```

Step 2 - Write a Dockerfile

Open your favorite text editor or IDE and create a file named: `Dockerfile`. Now let's add the commands to create our image.

First we tell Docker what base image we would like to use.

```
FROM pytorch/pytorch:latest
```

Next we create a working directory and copy our requirements.txt file into it. Then we install our dependencies.

```
WORKDIR /workspace

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt
```

Now we need to get our code into the image. We do this by copying the app.py file into the image.

```
COPY app.py app.py
```

Finally, we tell Docker what process or command to run when the container is started.

```
CMD ["python", "app.py"]
```

Here is the full Dockerfile:

```
FROM pytorch/pytorch:latest

WORKDIR /workspace

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY app.py app.py

CMD ["python", "app.py"]
```

Step 3 - Build the image

Back in the terminal, run the following build command.

```
$ docker build -t docker-ml:full .
```

Once the build is complete, print a list of the images on your local machine.

```
$ docker images
```

Step 4 - Run the image

Now we have an image that contains our python script and it’s dependencies. Let’s run it and make sure it runs without errors.

```
$ docker run -it --rm docker-ml:full

Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
to ./data/cifar-10-python.tar.gz
170500096it [00:07, 22333760.89it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
plane truck   cat horse
/opt/conda/lib/python3.8/site-packages/torch/autograd/__init__.py:130
: UserWarning: CUDA initialization: Found no NVIDIA driver on your
system. Please check that you have an NVIDIA GPU and installed a
driver from http://www.nvidia.com/Download/index.aspx (Triggered
internally at
/opt/conda/conda-bld/pytorch_1603729096996/work/c10/cuda/CUDAFunction
s.cpp:100.)
  Variable._execution_engine.run_backward(
[1,  2000] loss: 2.202
[1,  4000] loss: 1.860
...
[2, 12000] loss: 1.263
Finished Training
GroundTruth:    cat  ship  ship plane
Predicted:    cat  ship  ship  ship
Accuracy of the network on the 10000 test images: 52 %
Accuracy of plane : 51 %
Accuracy of   car : 75 %
...
Accuracy of truck : 43 %
cpu
```

You can Ignore the UserWarning because the sample code tries to check to see if we have a GPU attached to the container which we do not.

Step 5 - Include data inside the image

If you look at the beginning of the output, you’ll see that the python application downloads the sample data. You might want to include this data in your image or you might want to store the data outside of the image. Let’s first take a look at including the data inside your image.

Let's modify the Dockerfile so that when we build our image Docker will download, unzip and include the dataset inside of our image.

Add the bold lines to your Dockerfile.

```
FROM pytorch/pytorch:latest

WORKDIR /workspace

RUN apt-get update && apt-get install -y wget
RUN mkdir data && wget -c
https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz -O - | tar -xz
-C data/

COPY requirements.txt requirements.txt
RUN pip install -r requirements.txt

COPY app.py app.py

CMD ["python", "app.py"]
```

First we update our package manager and then install wget. Next we download the tar file from www.cs.toronto.edu website using wget then we untar and unzip the file into the data directory.

Build the new image. Notice that we are now using withdata as the tag and not full.

```
$ docker build -t docker-ml:withdata .
```

After the build is finished, list the images you have on your machine and compose the size of the two images.

```
$ docker images
docker-ml    withdata   3c79513be885   13 minutes ago   4.93GB
docker-ml    full       2783960c06b5   2 days ago       4.71GB
```

As you can see, the image tagged with data is about 200MB bigger than the image tagged with full.

```
$ docker run -it --rm docker-ml:withdata
```

Now when we run our container, we will not see the download phase. We see a message that the data has already been downloaded and verified.

```
$ docker run -it --rm docker-ml:withdata
Files already downloaded and verified
```

```
...
```

Step 6 - Share and persist data using volumes

Another option when building your ML pipeline is to store your data in a volume. Docker has two options for containers to store files in the host machine, so that the files are persisted even after the container stops: volumes, and bind mounts.

Volumes are the preferred mechanism for persisting data generated by and used by Docker containers. While bind mounts are dependent on the directory structure and OS of the host machine, volumes are completely managed by Docker.

Below, we will create a volume, add our dataset to it and then connect our container to the volume so the code has access to it inside the container.

```
$ docker volume create cifar10
DRIVER    VOLUME NAME
local     cifar10
```

Now that we have a volume, let's create an image. When we run the image it will download the tar file, unpack it and store it in the `/data` directory.

We'll first create a shell script that does the downloading and unpacking.

To do this, create a new directory named: "data". Inside of this directory, create a file named: `download.sh` and add the following commands to it.

```
wget -c https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz -O -
| tar -xz -C /data
```

We need the `download.sh` file to be executable. Run the following command so we can execute the file.

```
$ chmod +x download.sh
```

Create a new Dockerfile named: `Dockerfile.data` and add the following commands to it.

```
FROM alpine:3.7

RUN apk add --no-cache wget
RUN mkdir /data

COPY download.sh download.sh
```

```
CMD ["sh", "download.sh"]
```

We are using the alpine distro because of the small size. Next we install wget and make a directory where we'll download the dataset to. Then we copy the `download.sh` script into the image and then set it as the command to run when the image is started.

Build the image:

```
$ docker build -t downloaddata -f Dockerfile.data .
```

Now let's run the container and attach the volume that we created above.

```
$ docker run --rm --volume cifar10:/workspace/data --name train
docker-ml:full
Files already downloaded and verified
```

We used a couple of flags that we have not seen before. First we use the `--rm`. This tells Docker to automatically remove the container when it exits. Meaning, we do not have to run `docker stop` and `docker rm` when we are done running the container to stop and remove it.

We also use the `--volume` flag to tell docker to attach to the volume named `cifar10` and map that volume into our container at `/workspace/data.`

The container will run as it has done before but this time, the dataset has already been downloaded and attached to the container so our application can read the data.