


[https-deeplearning-ai](#) / [machine-learning-engineering-for-production-public](#) Public
[Code](#) [Issues 1](#) [Pull requests](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#)
[main](#)

...

[machine-learning-engineering-for-production-public](#) / [course4](#) / [week2-ungraded-labs](#) / [C4_W2_Lab_1_FastAPI_Docker](#) / [with-batch](#) / [README.md](#)
 **andres-zartab** C4 W2 uglS - Reorder webserver uglS

[History](#)
 2 contributors  
 127 lines (78 sloc) | 6.72 KB

...

Adding batching to the server

Now that you saw how to code a webserver that serves a model for inference it is time to implement some code that allows for getting predictions in batch. This is very important because right now your server can only handle one prediction per request and you are losing this feature that most machine learning predictors have been optimized to perform.

Begin the `cd` ing into the `with-batch` directory. If you are currently within the `no-batch` directory you can use the command `cd ../with-batch`.

Update the server

Remember that this code can be find within the `app/main.py` file.

First of all you will need two extra imports to handle prediction by batches. Both are related to handling list-like data. These are `List` from `typing` and `conlist` from `pydantic`. Remember that `REST` does not support objects like numpy arrays so you need to serialize this kind of data into lists. The imports will now look like this:

```
import pickle
import numpy as np
from typing import List
from fastapi import FastAPI
from pydantic import BaseModel, conlist
```

Now you will modify the `Wine` class. It used to represent a wine but now it will represent a batch of wines. To accomplish this you can set the attribute `batches` and specify that it will be of type `List` of `conlist`s. Since FastAPI enforces types of objects you need to explicitly specify them. In this case you know that the batch will be a list of arbitrary size but you also need to specify the type of the elements within that list. You could do a List of Lists of floats but there is a better alternative, using pydantic's `conlist`. The "con" prefix stands for `constrained`, so this is a constrained list. This type allows you to select the type of the items within the list and also the maximum and minimum number of items. In this case your model was trained using 13 features so each data point should be of size 13:

```
# Represents a batch of wines
class Wine(BaseModel):
    batches: List[conlist(item_type=float, min_items=13, max_items=13)]
```

Notice that you are not explicitly naming each feature so in this case the **order of the data matters**.

Finally you will update the `predict` endpoint since loading the classifier is the same as in the case without batching.

Since scikit-learn accepts batches of data represented by numpy arrays you can simply get the batches from the `wine` object, which are lists, and convert it to numpy arrays before feeding it to the classifier. Once again you need to convert the predictions to a list to make them REST-compatible:

```
@app.post("/predict")
def predict(wine: Wine):
    batches = wine.batches
    np_batches = np.array(batches)
    pred = clf.predict(np_batches).tolist()
    return {"Prediction": pred}
```

With these minor changes your server is now ready to accept batches of data! Pretty cool!

Building a new version of the image

The Dockerfile for this new version of the server is identical to the previous one. The only changes were done within the `main.py` file so the Dockerfile remains the same.

Now, while on the `with-batch` directory run the docker build command:

```
docker build -t mlep4w2-ugl:with-batch .
```

Since the initial layers are identical to the ones of the previous image, the build process should be fairly quick. This is another great feature of Docker called `layer caching`, which caches layers for newer builds to yield lower build times.

Notice that the image name stays the same, but the tag changed to specify that this version of the server supports batching.

Cleaning things up

To check out the two images you have created you can use the following command:

```
docker images
```

Sometimes when checking all of your available images you will stumble upon some images with names and tags that have the value of `none`. These are intermediate images and if you see them listed when using the `docker images` command it is usually a good practice to prune them.

Another way to check if you have these images in your system is to run this command:

```
$(docker images --filter "dangling=true" -q --no-trunc)
```

If there is no output it means there where no such images. In case there were some you can prune them by running the following command:

```
docker rmi $(docker images --filter "dangling=true" -q --no-trunc)
```

Now if you run again the `docker images` command you should not see those intermediate images. Things are much cleaner now!

Running the server

Now you can run a container out of the image using the following command:

```
docker run --rm -p 81:80 mlepc4w2-ugl:with-batch
```

Notice that this time, port 80 within the container maps to port 81 in your local host. This is because you probably haven't stopped the server from part 1 of this lab which is still running on port 80. It also serves to showcase how port mapping can be use to map from any port in the container to any port in the host.

Now head over to localhost:81 and you should see a message about the server spinning up correctly.

Make requests to the server

Once again it is time to test your server by actually using it for prediction. In the same manner as before there are some examples of batches of data within the `wine-examples` directory.

To get the predictions for a batch of 32 wines (found in the `batch_1.json` file) you can send a `POST` request to the server using `curl` like this:

```
curl -X POST http://localhost:81/predict \  
-d @./wine-examples/batch_1.json \  
-H "Content-Type: application/json"
```

Now you should see a list with the 32 predictions (in order) for each one of the data points within the batch. Nice work!

Stopping the servers

To step to servers and the containers they are running in, simply use the key combination `ctrl + c` in the terminal window where you started the process.

Alternatively you can use the `docker ps` command to check the name of the running containers and use the `docker stop name_of_container` command to stop them. Remember that you used the `--rm` flag so once you stop these containers they will also be deleted.

Do not delete the images you created since they will be used in an upcoming lab!

Congratulations on finishing this ungraded lab!

Now you should have a better understanding of how web servers can be used to host your machine learning models. You saw how you can use a library such as FastAPI to code the server and use Docker to ship your server along with your model in an easy manner. You also learned about some key concepts of Docker such as `image tagging` and `port mapping` and how to allow for batching in the requests. In general you should have a clearer idea of how all these technologies interact to host models in production.

Keep it up!