Assignment: ML Data Engineer Co-op, Spring 2025

The Nokia AIMS team

October 2024

1 Introduction

One of the key metrics for a warehouse is space utilization—the amount of free space in the warehouse dictates its ability to efficiently manage incoming inventory and optimize workflows, much like how free disk space in your computer is crucial for performance.

We're counting on your expertise to help build this feature for a massive warehouse operating in New Jersey. The challenge involves building a computer vision algorithm/model that classifies a location in the warehouse as 'empty' or 'filled', based on images taken from a drone flying in the warehouse. This algorithm needs to be deployed as a REST API service, so that it can seamlessly integrate with the rest of the system.

To help you out, we've split your mission into three separate tasks.

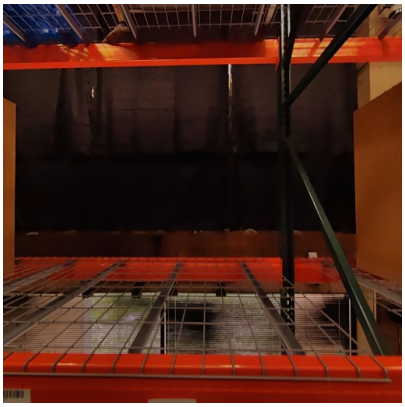
2 Task 1 - Training the model

Your first task is to train an image classification model that, given a single 512px x 512px RGB image, classifies whether the location shown in the image is 'empty' or 'filled'.

In the '`training_images/`' directory, you will find an '`images`' directory, containing **1007 512x512 JPEG images** you can use to train your model. The images are captured by a high-resolution camera on-board a drone that flies vertically along the warehouse aisles, like a vertical lawn-mower.

You will also find a '`labels.csv`' file, which contains the 'empty' or 'filled' labels against each image. 'Empty' locations in the warehouse have no visible boxes/items, whereas 'Filled' locations have at least one box/item. **However, note that if a location has an empty 'pallet', it is considered empty !**

See Figure 1 for some examples.



(a) An 'empty' Location

(b) A 'filled' location

(c) An 'empty pallet' - considered an empty location

Figure 1: Some examples from the training dataset.

2.1 Notes

- Feel free to use any learning framework (PyTorch/Tensorflow/JAX/etc.) of your choice.
- Feel free to use any publicly available architectures and model checkpoints that might suit this task. Don't forget to cite the original authors in your documentation!
- Avoid the use of end-to-end pipelines (such as `transformers.pipeline`, `lightning.Trainer`, etc.). In particular, your code must contain the following:
  - A forward pass of the image input through the model.
  - Calculation of the loss (you may use methods provided by the framework for this).
  - A backward pass through the model using the loss (you may use methods provided by the framework for this).
  - Calculation of the accuracy per training step.
- After the end of each epoch of training, print out the mean training loss, mean training accuracy and mean validation accuracy.
- You may submit your training code as a Jupyter or Google Colab notebook.

Once you have sufficiently trained your model, save or export your model to the 'checkpoints/' directory.

3 Task 2 - Serving the model

Now that you have a trained model, it's time to deploy it for real-world use. Most models are served using an API service.

Your API service must expose the following two endpoints over HTTP:

- [**GET**] `/author`:
  Request parameters - No parameters
  Response (`application/json`) - `{"author": <your email address here>}`
- [**POST**] `/classify`:
  Request parameters (`Content-Type:multipart/form-data`) - `{"image":<binary JPEG image file>}`
  Response (`application/json`) - `{"class":<string (either 'empty' or 'filled')>}`

4 Task 3 - Deploying the service

Now, it's finally time to deploy your service using Docker containers.

Edit the 'Dockerfile' file provided in the repository, and install all the dependencies for your service in this Docker image. Use the appropriate base image for your service.

When run, your Docker container must initialize your API service and expose it via port 8080. Your endpoints must be accessible at `http://localhost:8080/<endpoint>` on the host machine.

5 Wrapping Up

Finally, add a `README.md` file, with the following details:

- Your name and email ID
- The structure of your codebase (mention where the model, service and other files are located)
- Your classification model's architecture (If you used a pre-trained model, your motivation behind choosing a specific model)
- Your impressions on the training dataset (are the classes well-balanced, etc.)? Did you have to modify/augment the dataset to improve your model? If so, what were the modifications, and how do you think they affected the model?
- Metrics for your final model checkpoint. Include all metrics you think are relevant.
- Instructions and dependencies to build the Docker image for your API service.
  Include a sample command, with any environment variables or build flags required.
- Instructions and system requirements to run your Docker container.
  Include a sample command, with any environment variables or runtime flags required.
- Anything else you'd like us to know regarding your code or model.

6 Evaluation Notes

- Your model will be evaluated on a hidden benchmark test dataset, through the API service. Test images will be posted to the prediction endpoint as per the request parameters.
- Like in the real world, your model's test accuracy is not the sole factor in evaluation. Compute efficiency of the service, resource usage (CPU/RAM) of the Docker container, readability of the codebase, etc. play a role, among other factors.
- All submissions will be evaluated on a system equipped with a 48-core Intel Xeon Silver 4214 CPU, and an NVIDIA RTX A4000 GPU (16GB VRAM).
  To take advantage of the GPU during evaluation, your Docker image — and the Docker container when it's run — must be configured to support GPU usage. Please ensure you install all necessary GPU-related dependencies and use the appropriate flags to enable GPU support at runtime.
  Submissions specifically optimized for CPU are welcome and will be evaluated equally.

If you have any questions, feel free to reach out to nikhilanj.pelluri@nokia.com or prasanth.ananth@nokia.com.
Have fun !