ASW Machine Learning Engineer Nanodegree – Capstone Project Report
Inventory Monitoring at Distribution Centers

Davide Martizzi

Summary

In this project a deep learning model was developed with the aim of processing images from bins in a product distribution center. For each image of a bin containing either zero or more than one objects, the model is tasked with counting the number of objects. The model was deployed on AWS using the SageMaker, S3 and Lambda services. The project required too much training time to run within the budget provided by Udacity AWS credits, and was run with a budget of 50 USD from my personal AWS account. Given the constraints in terms of time and budget for this project, and as compared to the benchmark model used for comparison, the performance of the model expressed in terms of accuracy and root mean squared error (RMSE) was deemed satisfactory.

Project Scope & Domain Knowledge

The study and engineering of solutions for inventory logistic and management is a mature and thriving field [1]. Cutting-edge companies are continuously adopting innovative solutions to automate and increase the efficiency of processes such as manufacturing, quality assurance, inventory management, product distribution and marketing. The adoption of new AI-driven solutions has significantly improved this transformation, but several challenges still exist, because the performance of some of these solutions is still falls short of expectations [2]. The goal of this project is to solve a business problem related to inventory monitoring at distribution centers.

Product distributions centers often use robots to move objects as a part of their operations. In order to obtain full automation of the process and assess whether robots are operating as intended, monitoring systems are deployed to perform quality control. One of the most basic applications for quality control is counting the number of objects in each bin without human involvement, e.g., by processing of images of each bin.

In this project, a deep-learning-based computer vision model to count the number of objects in bins was developed and deployed on AWS, and its predictions were made accessible via a web app. The latter is currently available for local deployment.

Problem Statement

Formally, the problem falls in the realm of weakly-supervised object detection, the main task being the identification and counting of objects in bins within a distribution center using images. The task is weakly-supervised because it’s difficult to get detailed annotations for the bounding-box position and label of objects in the images, which are typically required to train most object detection models.

However, since we are only interested in the count of objects in each bin, some of the complexity of identifying the correct location and label for each object can be dropped. This should allow me to choose simpler model architectures and use weaker labels to perform the task.

The model takes images as an input and yield the count of objects in the image as an output.

Datasets and Inputs

The dataset that used for this project is the Amazon Bin Image Dataset: https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds. The dataset contains > 500,000 bin JPEG images and corresponding JSON metadata files describing items in the bin. Among the metadata is the count of the number of objects in each bin, which can be used to define target labels for the machine learning problem.

Due to the large size of the dataset and to the constraints in terms of budget and time, the dataset was too big to be fully utilized. For this reason, a subsample of ~50,000 images with object counts from 0 to 5 was randomly selected from the original dataset. This random subsample was split in training, validation, and test subsets, which were stored on AWS S3.

Solution Statement

The model used to solve this problem is a variant of the approach implemented in https://github.com/silverbottlep/abid_challenge (the benchmark model). The idea is to reduce the object count problem to an image classification model in which the class corresponds to the count of objects in the image.

In this implementation, the weights of the pre-trained ResNet50 classifier for ImageNet were loaded, and only train the deepest layers of the network plus the custom output layers, specifically the last few blocks of the ResNet50 architecture, plus the output layers. With this approach, one can in principle leverage the low-level image features extracted from image classification and re-purpose the rest of the network for object identification and counting. The output layer yields a softmax distribution over the possible values of a class label corresponding to the number of objects in the image.

More details of the model architecture and implementation are reported below:

- ResNet50 ImageNet classification weights are available in PyTorch. Convolutional filters that select small scale features combined with the residual neural network architecture that have been proven to be effective for classification tasks up to 1000 classes [5].
- The ResNet architecture was proposed to overcome the problem of vanishing gradients in very large neural network and converge to optimal model parameters faster [5].
- The ResNet architecture is contains `residual blocks` with two branches each. One branch applies convolution-scaling-convolution operations. The other branch applies identity or convolution operations. At the end of each block the results from both branches are summed. ResNet50 includes 9 residual blocks, and an additional final

feed-forward block. A softmax activation function can be applied to compute class probabilities.

- In the present implementation ImageNet parameters are adopted for the first 6 residual blocks and they are frozen to such values. The last two blocks are set as fully trainable.
- In the present implementation the feedforward block is substituted by two feed-forward blocks with 128 and 6 output neurons, respectively. The parameters of this portion of the network are trainable.
- The final model has 22,326,150 trainable parameters.
- Although the scope of this project is object counting, the problem has been reformulated as a classification task, so the small scale filters for ImageNet can be re-used for other classification tasks.
- The authors of the benchmark model [4] trained a ResNet34 network from scratch on a dataset ~10x larger than the one used in this project, but with the same number of classes.
- In this project, the transfer learning approach with partial training of a slightly modified ResNet50 architecture was chosen mostly for curiosity about achieving similar results with fewer data, a slightly more complex model architecture, but limitation on the number of trainable parameters.
- The final consideration for choosing this transfer learning approach is that it significantly reduces training time in instances where budget and time are limited, such as this project.

Once deployed on AWS, the model was made accessible via a web app where a user would be able to select the ID of one of the images in the test set. The client only maintains a list of the IDs of the images in the test set. The client sends a JSON request to a lambda function which retrieves the image stored on S3 and pass it to a model endpoint. The model yields the inferred number of objects in the image, which is sent by the lambda function back to the client together with the image. Finally, the client visualizes the image and the inferred number of objects.

The general architecture of the model is shown in Figure 1. Several choices for the values of N were chosen during experimentation with the model, but eventually N=6 (object counts 0-5 in the images) was chosen to better compare to the benchmark model [4].

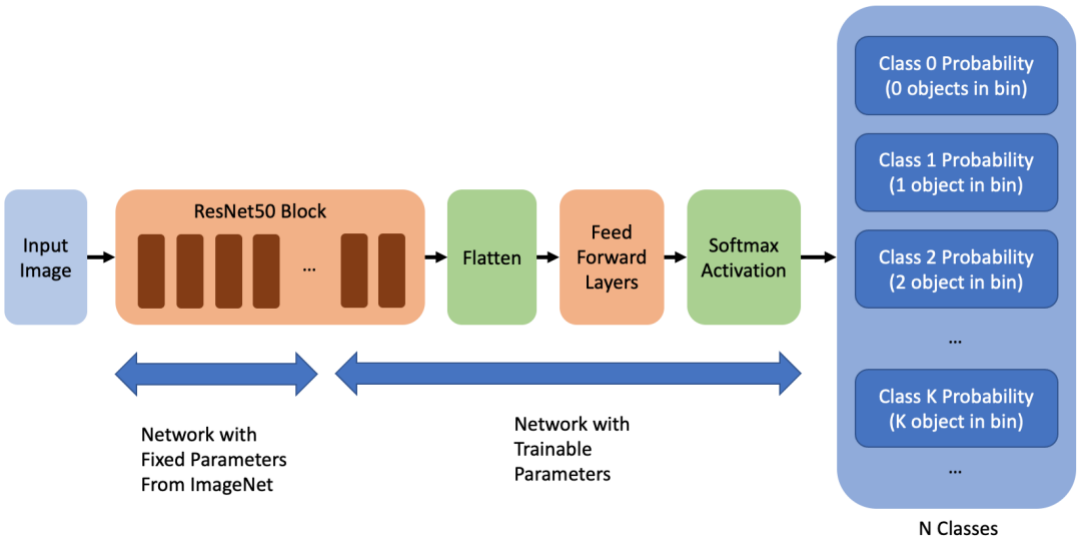

Figure 1. Model Architecture for this project.

Benchmark Model

The benchmark model chosen for this project is based on pre-trained ResNet models with custom output layers. This approach is inspired by the following implementation of a solution to the same problem: https://github.com/silverbottlep/abid_challenge.This implementation achieves an accuracy of 55.67% and a RMSE of 0.93.

In the benchmark implementation, the model is trained from scratch and uses ResNet34, whereas in this implementation the weights of the pre-trained ResNet50 classifier for ImageNet are loaded, and only the deepest layers of the network plus the custom output layers are trained.

As in the reference implementation, only images with 0 to 5 objects in the bin are considered.

Evaluation Metrics

The evaluation metrics chosen for this task are:
1. The accuracy in retrieving the exact number of objects in the image.
2. The RMSE obtained by comparing the inferred number of objects with the true one.
The metrics are tracked for training and validation sets during hyperparameter optimization sweeps. The same metrics are then used to evaluate the best model but computed on the test set.

Deployment Architecture

The design of the project, timeline, workflow, general architecture of the system, and visualization of the components that need to be developed is summarized in Figure 2.

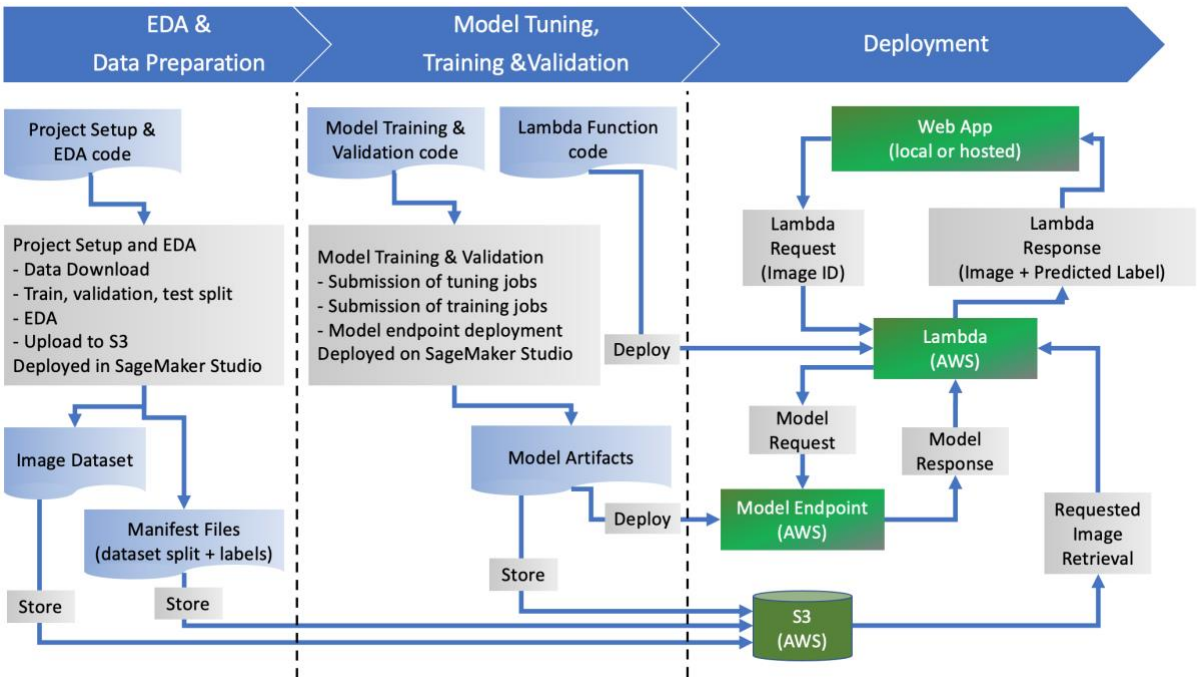

Figure 2. Project Phases and System Architecture Diagram.

The architecture of the system is very simple, and the main advantage of adopting this approach lies in the use of transfer learning to speed up the model training process while achieving competitive performance. Although simple, this architecture is relatively scalable, because it allows to take advantage of both lambda function concurrency settings and endpoint autoscaling.

Documenting steps 1 – Transfer to S3

Data transfer of the subsample of images used for this project was time consuming (it took several hours). It required fetching data from a public S3 bucket to the private one used for this project. This data transfer was deemed necessary in the first place, because the data was ultimately stored in a private S3 bucket to achieve better security.

A total of 66667 randomly selected JPEG images were transferred from the public S3 to the private one. The JSON metadata files that were used to assign labels to the images were also copied to the same S3 bucket.

Code used for the transfer between S3 buckets can be found in the `eda` folder, including the `eda/eda_preprocessing.ipynb` notebook.

Figure 3 shows proof of the successful transfer to S3.

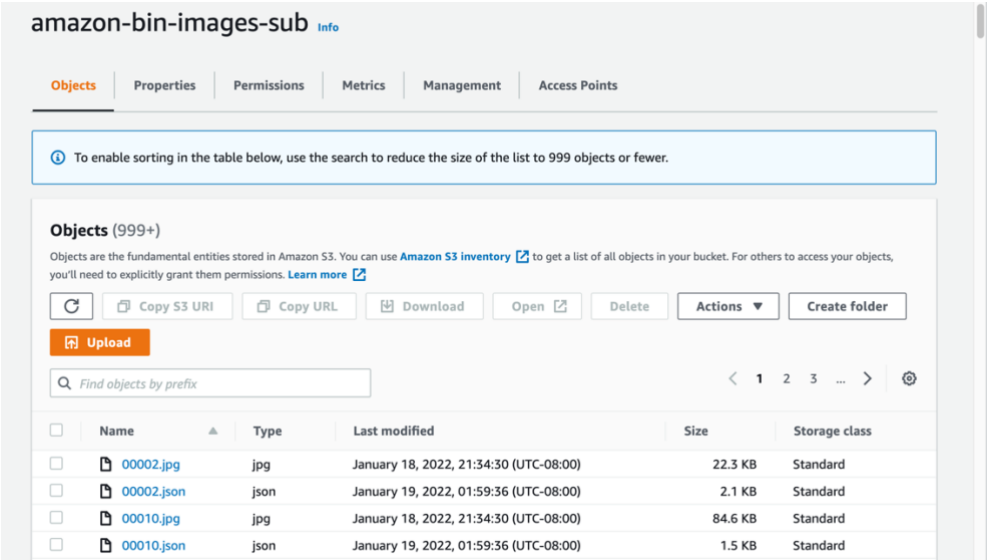

Figure 3. S3 bucket containing images and metadata.

Documenting steps 2 - Exploratory Data Analysis (EDA) - Results

EDA code can be found in the `eda` folder, including the `eda/eda_preprocessing.ipynb` notebook, which contains details of the analysis.

Summary

- The dataset is not fully balanced across the 6 classes chosen for training (0-5 items in the bin).
- The randomized 80:10:10 training, validation, testing split yields similar class distributions across the three different sets.
- Images are in standard JPEG format, with three RGB channels and pixel values 0-255.
- Images have variable height and width in terms of number of pixels; they need to be rescaled to a fixed size before being fed to the model.

Detailed report

Figure 4 shows the distribution of object counts in bins in the full dataset, and should allow the reader to appreciate that a minority of the bins contain more than a handful of object. As a matter of fact, only 25% of the bins contain more than 6 objects. This is the main consideration that led to further selecting only images with 0-5 object counts for training, validation and testing of the model. This is also a choice made by the authors of the benchmark model and facilitates comparison to their results [4].

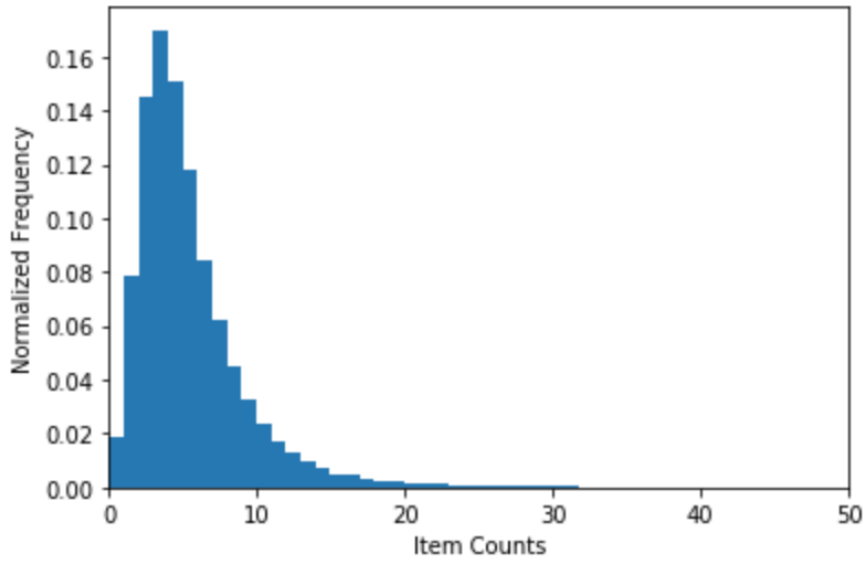

Figure 4. Item counts distribution.

After selecting images with 0-5 object counts and performing a random 80:10:10 training:validation:testing split, the following image counts were found in each sample:

- Training: 36092 images.
- Validation: 4512 images.
- Testing: 4512 images.

Figure 5 should convince the reader that the training, validation and test sets have a very similar label distribution and that they can be used for training, validation and testing of a classifier.

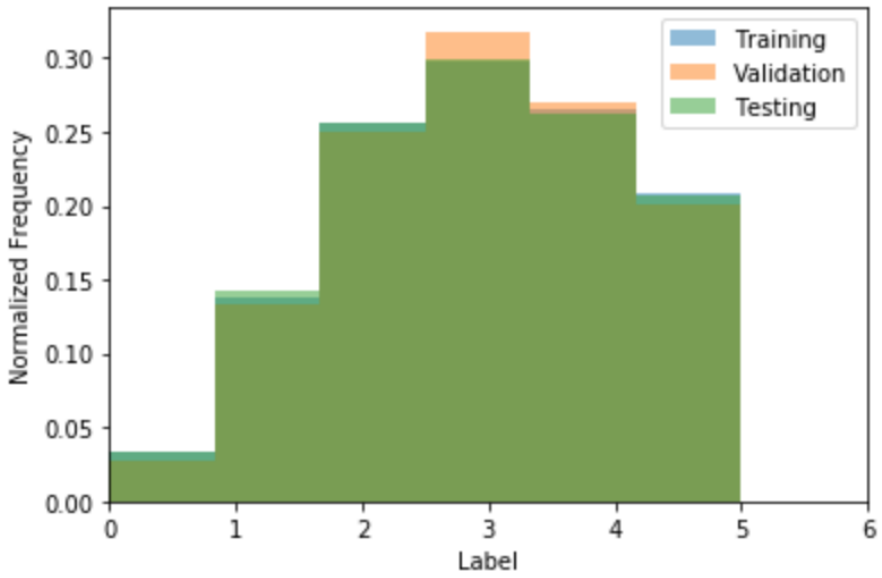

Figure 5. Label distribution in the training, validation and testing sets.

The EDA revealed that the images have the usual RGB channels with pixel values 0-255, but that image sizes are variable. In order for the deep learning model to operate, image sizes have to be standardized. Rather than pre-processing data in this phase of the project, it was decided to use the capabilities provided by PyTorch to pre-process data when passing batches to the model for training or prediction. For this reason data pre-processing is discussed in the next section.

See `eda/eda_preprocessing.ipynb` for more details.

Documenting steps 3 – Data Preprocessing and Model Training - Results

Code related to training and deployment of the model can be found in the `ml_model` folder, including the `ml_model/train_and_deploy.ipynb`, which contains code to launch hyperparameter tuning jobs, training jobs, and which deploys and tests endpoints. The entrypoint code for each task includes functions to pre-process images before feeding them to the training and prediction algorithms.

Summary

- Data pre-processing for training includes random resized cropping to 224x224 image size, data augmentation with random vertical and horizontal flips, and pixel value normalization.
- Data pre-processing for validation and testing only includes resizing to 224x224 image size and pixel value normalization.
- The PyTorch model was trained by seeking minima of the CrossEntropyLoss function (appropriate for classification tasks) with the Adam optimizer.
- Hyperparameter optimization was used to validate the model and set hyperparameters that yield best performance in terms of minimizing the loss function computed on the validation set. Validation loss = 1.3095589876174927.
- Training of the best model for 50 epochs takes > 6 hours on a GPU-accelerated `ml.g4dn.xlarge` instance.
- The model was tested on the test set, yielding
    - Test Loss Function = 1.2321901550263372.
    - Test RMSE = 1.1540604249432982.
    - Test Accuracy = 0.43515850144092216
- The model trained in this project has slightly worse performance than the baseline model propsed by https://github.com/silverbottlep/abid_challenge (Accuracy = 0.5567, RMSE = 0.930), however the differences can be attributed to multiple factors:
    - The authors of the baseline model use the full image dataset with >500,000 images. They have ~10x more training examples than in this project that uses a random subsample of ~50,000 images.
    - They train a ResNet model from scratch. In this project, transfer learning was performed, with a large part of the ResNet50 neural network whose parameters were frozen. For this reason, the baseline model is probably capable of extracting better features than in the transfer learning model trained here.
    - This project was performed with a tight limit on the training time and budget.

Detailed report

Because the number of images in the sample has been reduced by a factor ~10 compared to the full sample, data pre-processing for training included data augmentation processes that should help the model generalize. Resized cropping allows the model to be less dependent on translations of the same image. Since the target variable, object count in each image, is independent on image orientation, data augmentation was also performed via random vertical and horizontal flips. Finally, a normalization step was included.

For validation, testing and prediction, only resizing and normalization were applied because there is no need for the additional steps.

Fine tuning of the model was performed by seeking a minimum of the CrossEntropyLoss function computed on the validation set. Training was performed with the Adam optimizer. The list and ranges of hyperparameters that were explored is reported below:

- Number of epochs, continuous range: [5, 50].
- Learning rate, continuous range: [0.001, 0.01].
- Batch size, categorical: [32, 64].

Bayesian optimization was used to explore this hyperparameter space with 6 runs. Ultimately, the model with best valuer for the CrossEntropyLoss function computed on the validation set.

Figure 6 to Figure 9 below show documentation of the hyperparameter tuning and final training steps. In particular, Figure 8 shows hyperparameters of the best model.

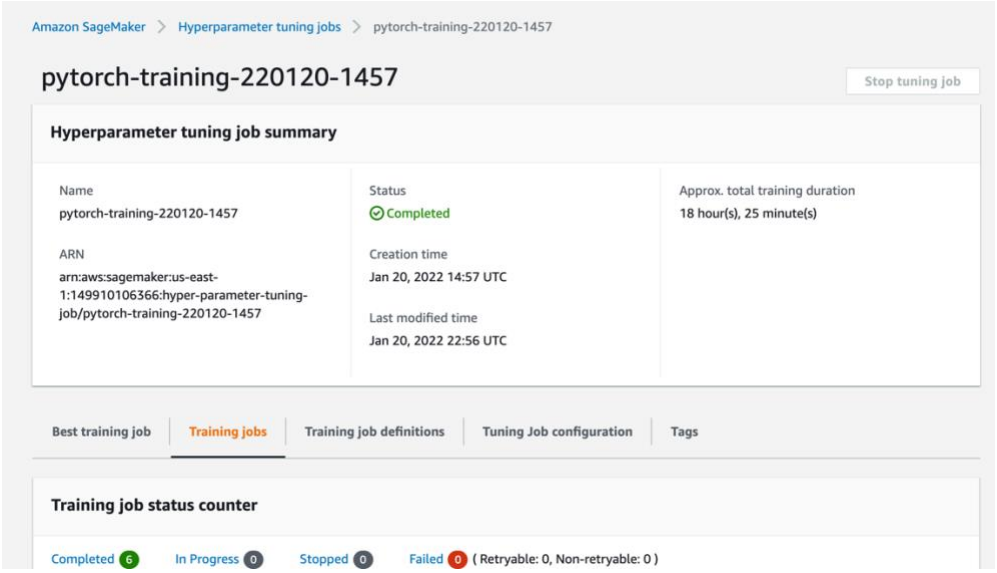

Figure 6. Successful hyperparameter tuning job, part 1.

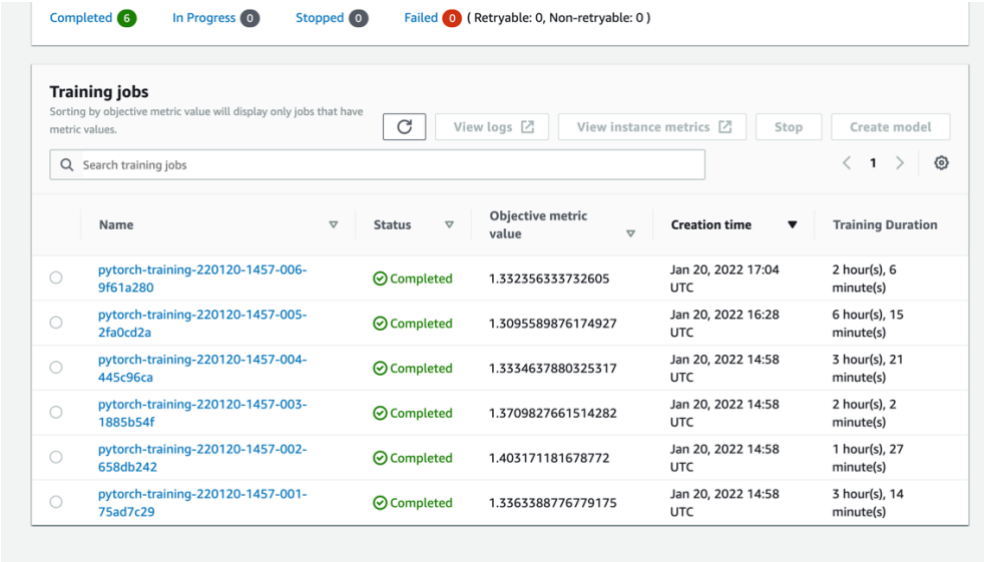

Figure 7. Successful hyperparameter tuning job, part 1.

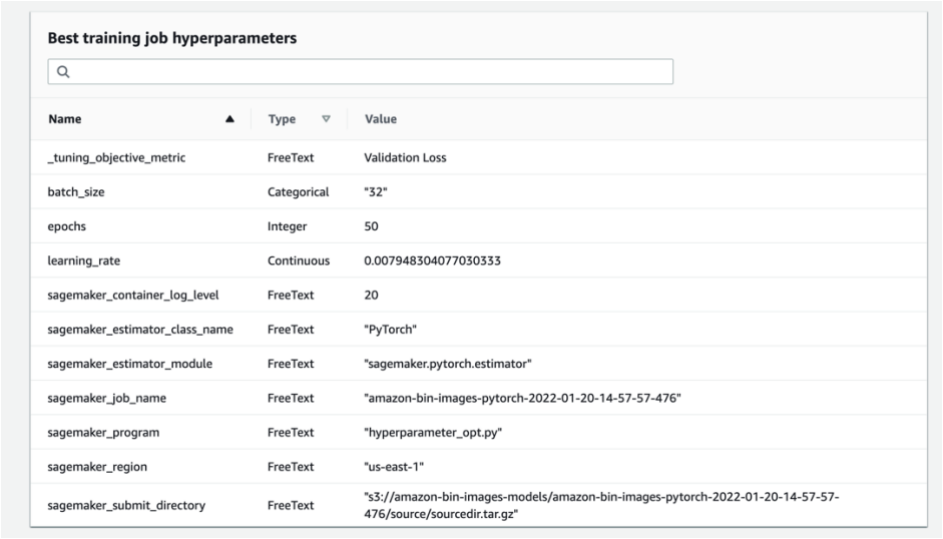

Figure 8. Hyperparameters of the best model.

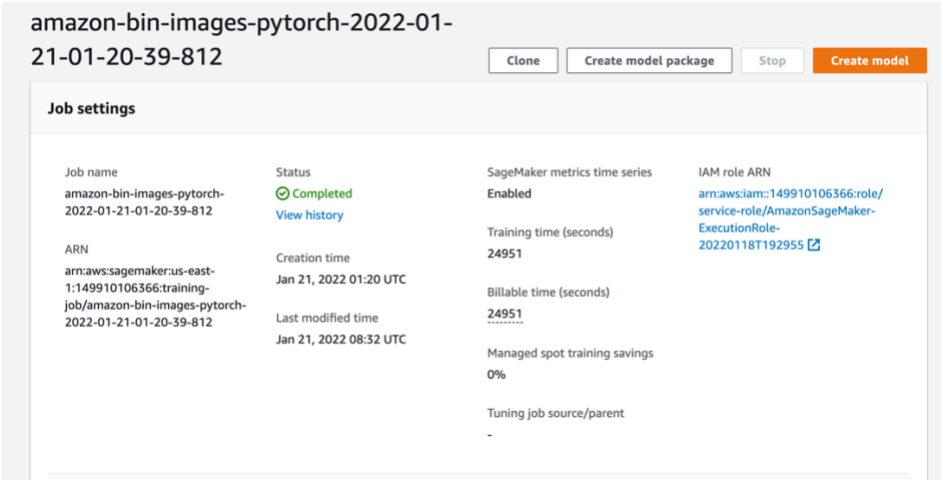

Figure 9. Successful training job for the model with best hyperparameters.

Figure 10 to 12 show direct comparison of the training and validation average loss, accuracy and RMSE as a function of time during training of the best model. These figures should convince the reader that performance of the model on these two sets converge to similar results by the end of the training process.

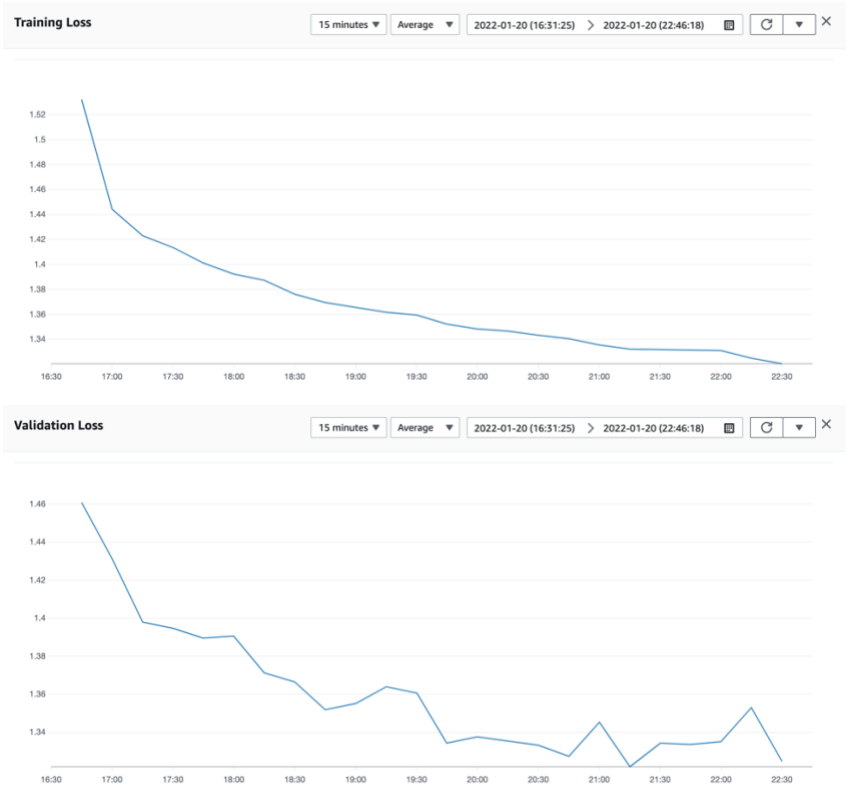

Figure 10. Training vs. Validation average loss during training of the best model.

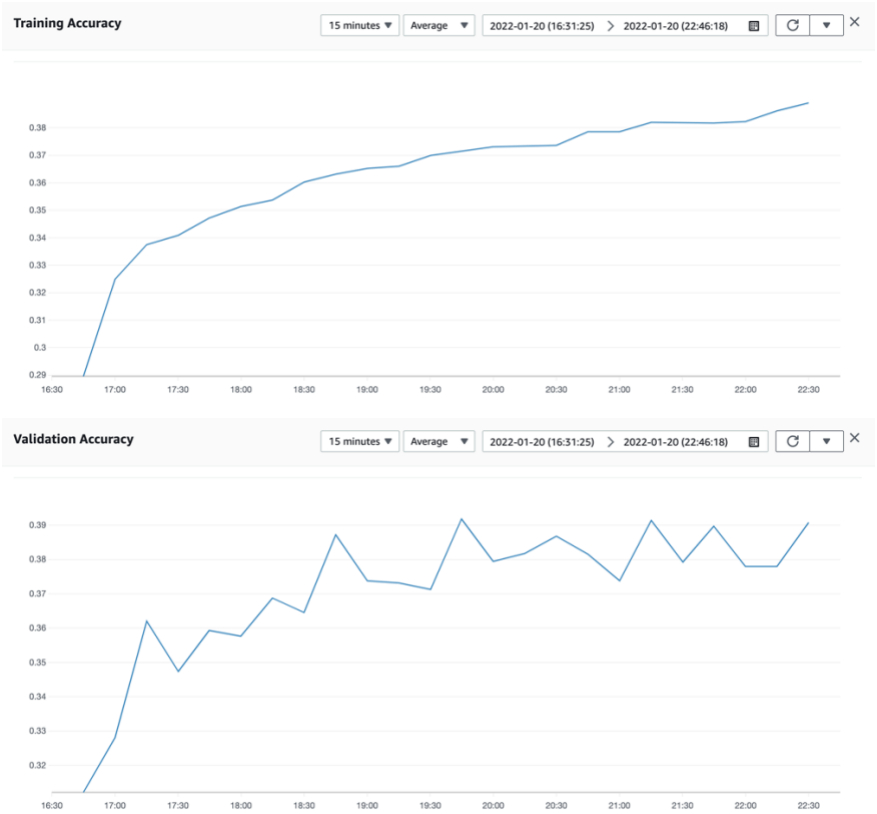

Figure 11. Accuracy during training of the best model.

Figure 12. RMSE during training of the best model.

Training for the final model was performed with hooks for the debugger and profiler, but a glitch caused the files they produced to be corrupted within S3, and therefore not retrievable. Due to the fact that training takes more than 6 hours on relatively expensive GPU-accelerated `ml.g4dn.xlarge` instances, the training job was not repeated to re-capture debugger and profiler outputs. The performance of the final model on the test set reported above was extracted from the CloudWatch logs.

Figure 13. CloudWatch logs for the final training jobs.

label	accuracy	rmse
0	0.927419	0.515877
1	0.741155	0.605684
2	0.546306	0.855903
3	0.389680	1.067141
4	0.162107	1.283517
5	0.420308	1.669321

Table 1. Performance of the model for each class.

Table 1 disassembles where the model shines and where the model fails. In particular, for images that contain 0-2 objects the performance of the model is quite good. RMSE quantifies how much the predicted counts deviate from the actual numerical value on average, and again the performance is better for classes 0-2. These results are qualitatively in line with those found in the benchmark model [4], even if the authors find better performance for most classes. The reason for this loss is that images capture only the surface of a bin, and there may be objects hidden below the surface that would obviously not be countable by this model. Furthermore, due to image quality (blurriness, distortions, bad cropping, etc.), even humans may struggle to reliably count objects in some of the images.

Potential methods to improve performance in future experiments:

- Use a bigger dataset, or the full dataset. There is potential to learn better features from a more representative dataset. More time and a bigger budget would be needed.
- Perform a more thorough exploration of the hyperparameter space. More time and a bigger budget would be needed.
- Make more blocks in the ResNet architecture trainable as in the benchmark model. More time and a bigger budget would be needed.

Documenting steps 4 - Deployment

The model was deployed on a `ml.m5.large` instance as a tradeoff between price and computation time, as documented in Figure 14. See `ml_model/train_and_deploy.ipynb` for more details.

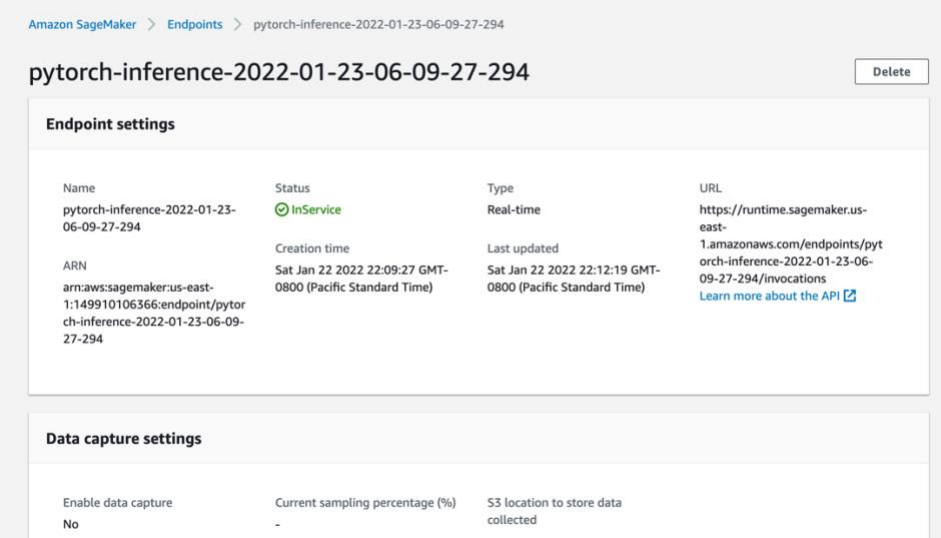

Figure 14. Snapshot of a deployed inference endpoint.

The second part of the deployment required deploying a Lambda function that performs a few operations:

- Receive name of requested image.
- Fetch a requested image from S3.
- Pass the image to the model endpoint and obtain a prediction.
- Serve image and prediction as an output.

Figure 15 documents deployment of the Lambda function.

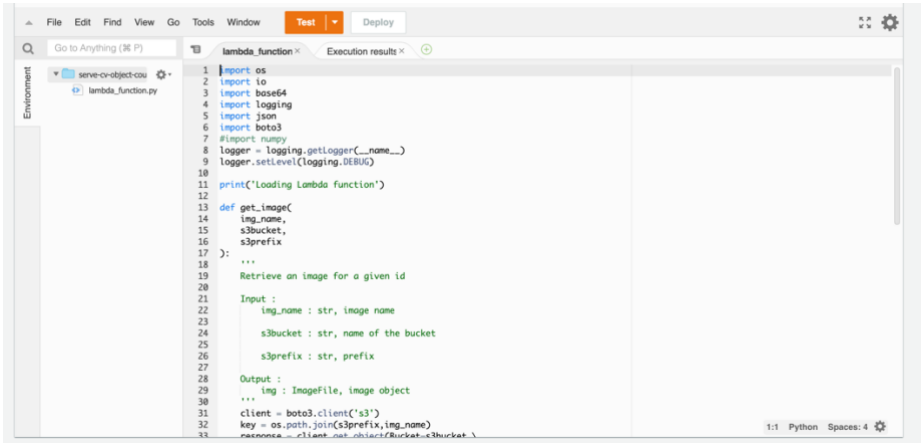

Figure 15. The Lambda function deployed for this project.

Documenting steps 5 – Web App

The final step of the project involved developing a simple web app that:

- Allows the user to choose an image from a pre-populated list, in this case the images in the test set.
- Invokes the Lambda function to retrieve the selected image and the model prediction for the object counts.
- Visualizes image and prediction as an output.

The app was developed using the Flask framework for Python and Bootstrap was used as a framework for the website's templates. In short, this framework allows the developer to define Python functions for each website view and

A video demo of the web app in operation is included in this repository under `cv-object-counter-app_demo.mov`.

References

[1] Williams, Brent & Tokar, Travis. (2008). A Review of Inventory Management Research in Major Logistics Journals. The International Journal of Logistics Management. 19. 212-232. 10.1108/09574090810895960.

[2] Baldassari. (2021). Industry 4.0 reality check: Separating hype from reality. https://flex.com/resources/industry-4-reality-check-separating-hype-from-reality.

[3] Amazon Bin Image Dataset. https://github.com/awslabs/open-data-docs/tree/main/docs/aft-vbi-pds.

[4] Benchmark model. https://github.com/silverbottlep/abid_challenge.

[5] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.