Capstone Project Report

Inventory Monitoring at Distribution Centers

1. Domain Background

Inventory monitoring in warehouses involves efficiently managing and tracking the movement, storage, and availability of goods or products within a warehouse or distribution center. It is crucial for warehouse operators to have accurate and real-time information about their inventory to optimize operations, reduce costs and streamline supply chain management.

Computer vision, a field of artificial intelligence, can greatly assist in inventory monitoring activities within warehouses. By utilizing computer vision technologies, such as cameras and image processing algorithms, several tasks can be automated and streamlined, leading to increased efficiency and accuracy in inventory management.

This project focus on inventory counting and auditing, which is only one of many tasks that can benefit from computer vision technologies in a warehouse. Computer vision can automate the inventory counting process by analyzing visual data and detecting the presence and quantity of items in a given area or container, such as bins. Together with robots, this eliminates the need for physical counting and manual record-keeping, saving time and reducing the chances of errors. Regular auditing of inventory becomes more efficient and accurate with computer vision systems in place.

2. Problem Statement

Robots play a significant role in distribution centers, where their automation capabilities greatly enhance efficiency and productivity. These robots are often referred to as autonomous mobile robots (AMRs) or collaborative robots (cobots). They are designed to work alongside human workers and perform a variety of tasks, including inventory management, order picking, packaging, and transportation of goods within the facility.

The project aims to develop a system that counts the number of objects, with a maximum limit of 5, in a bin carried by robots within a distribution center. This system will leverage computer vision technology to accurately detect and count the objects present in the bin.

The value of this project lies in its ability to enhance the efficiency and accuracy of inventory management within the distribution center. By automating the counting process, the system can provide real-time insights into the inventory levels, allowing for better decision-making and resource allocation.

3. Datasetes and Inputs

The dataset used in this project will be the Amazon Bin Image Dataset, which contains over 500,000 JPEG images and JSON metadata from bins of a pod in an operating Amazon Fulfillment Center. The bin images in this dataset are captured as robot units carry pods as part of normal Amazon Fulfillment Center operations.

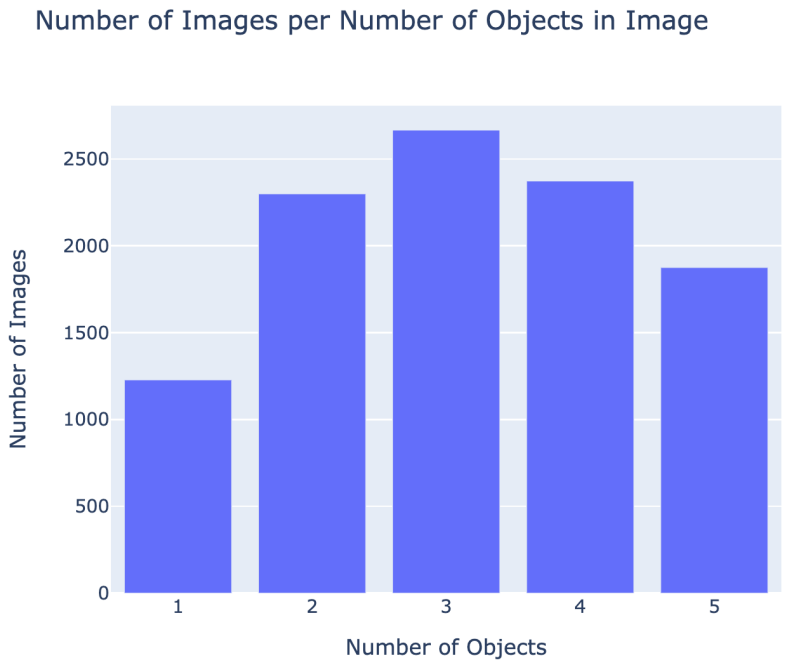
The data is distributed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 and can be retrieved from the public S3 bucket named **aft-vbi-pds**. More information about the data is available at this link.

Initially, a restricted subset of images will be utilized to test the feasibility of the idea and validate the machine learning pipeline. This subset will serve as a proof of concept to ensure that the pipeline functions as intended. Once the initial testing phase is completed and the machine learning pipeline is deemed successful, the model can then be retrained with additional data.

4. Exploratory Data Analysis

Understanding the class distribution within a dataset is crucial for various domains, including machine learning, data analysis, and decision-making processes. A plot depicting the class distribution visually represents the frequency or proportion of each class, providing insights into the balance or imbalance of classes in the data. By analyzing and visualizing the class distribution, analysts can gain a quick overview of the dataset, identify biases or imbalances, make informed decisions, and improve model performance. It helps in determining appropriate sampling strategies, guiding feature engineering and selection, and developing accurate predictive models. Ultimately, plotting the class distribution is an essential step in data analysis and machine learning tasks to ensure unbiased and effective results.

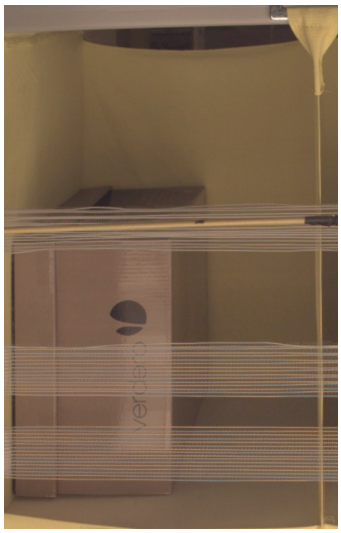
The following plot shows the frequency distribution of the image classes.



The class distribution plot reveals noticeable variations in the number of images across different classes. Notably, class 1, representing images with a single object, has fewer than half the instances compared to class 3. In order to test the machine learning pipeline we proceeded with the data as it currently is, but an opportunity for improvement is to get more data and equalize the number of instances per class.

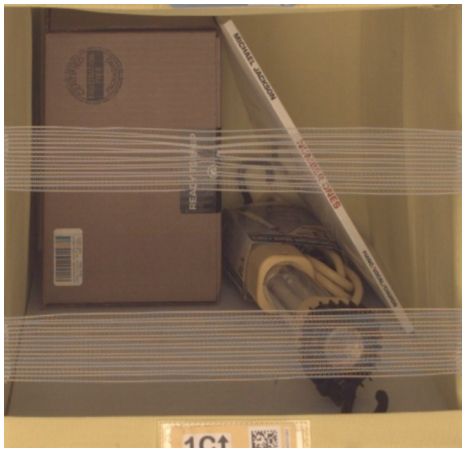
We should now proceed with inspecting some images to gain a better understanding of the data that will be fed to the model.

Class 1

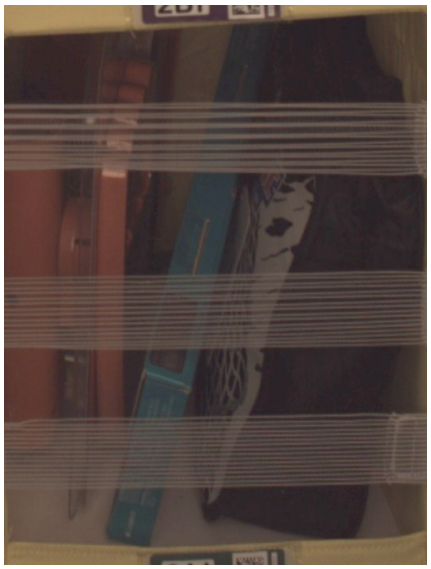


Class 2

Class 3



Class 4

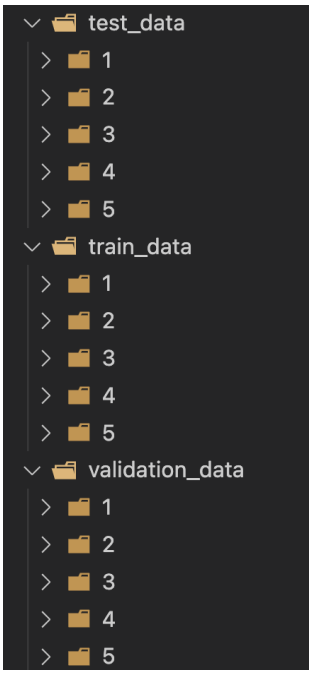


Class 5

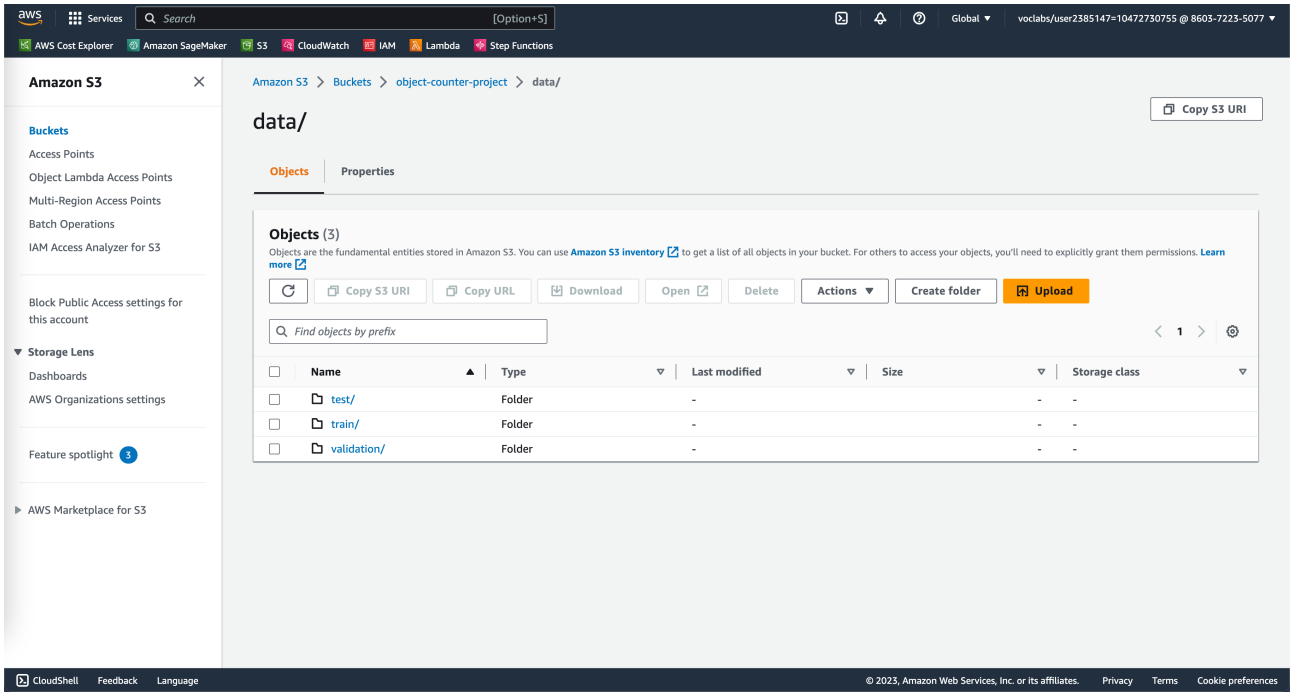
As the count of objects in an image increases, it becomes increasingly challenging, even for humans, to accurately determine the number of objects.

5. Preprocessing

Once the data has been downloaded and explored, it becomes necessary to split it into training (85% of the data), validation (10%), and testing sets (5%). Additionally, since our data loaders utilize the folder names as class labels, we have maintained the same folder structure for each partition of our data, as shown below.



In order to make the data accessible to SageMaker training resources, the data was uploaded to an S3 bucket. This allows the training resources to easily access and retrieve the data for further processing and model training.



6. Model Architecture

The problem stated above requires a robust solution for multi-class classification of images. The most appropriate modelo for this application is the use of Convolutional Neural Networks (CNNs), as they have revolutionized the field of computer vision by significantly improving the accuracy and efficiency of image analysis tasks.

One especially important feature of this approach is the ability to use transfer learning, which involves using pre-trained models trained on large datasets like ImageNet. These pre-trained models have learned general features from a vast amount of data and can be fine-tuned or used as a feature extractor for specific image classification tasks with limited training data. Transfer learning allows for faster convergence and improved performance, especially when labeled training data is scarce.

7. Model Benchmarking

Benchmarking models against other models is of great importance in machine learning and deep learning projects. It allows for the evaluation and comparison of

different approaches, architectures, and techniques to assess their performance and determine the most effective solution.

In our specific project, comparing the performance of a ResNet50 fine-tuned from a pre-trained model with a ResNet50 trained from scratch allows for the evaluation of the impact of transfer learning. The model trained from scratch will serve as the benchmark, providing a baseline for assessing the improvement achieved through fine-tuning. This comparison will provide valuable insights into the effectiveness of transfer learning and guide future decisions in model training and optimization.

8. Evaluation Metrics

In the project, the evaluation of model performance will involve multiple metrics, including accuracy, precision, recall, and F1-score. These metrics provide a comprehensive understanding of the model's effectiveness in handling the multi-class classification problem.

- Accuracy: Accuracy is a common metric used in classification tasks and measures the percentage of correctly classified instances out of the total number of instances. It provides an overall assessment of the model's performance across all classes. The accuracy metric will be the primary evaluation measure for the project, reflecting the general classification performance on the test dataset.
- Precision: Precision quantifies the proportion of correctly predicted positive instances (true positives) out of all instances predicted as positive (true positives + false positives). It provides insights into the model's ability to minimize false positives. Precision will be computed for each class separately, highlighting the precision of the model for each class individually.
- Recall: Recall, also known as sensitivity or true positive rate, measures the proportion of correctly predicted positive instances (true positives) out of all actual positive instances (true positives + false negatives). It evaluates the model's ability to capture all positive instances and avoid false negatives. Like precision, recall will be computed for each class, allowing the assessment of the model's recall performance on a per-class basis.

- F1-score: The F1-score is the harmonic mean of precision and recall. It provides a balanced measure that combines both precision and recall into a single metric. The F1-score considers false positives and false negatives, making it useful for evaluating the model's performance when there is an imbalance between the classes. Similar to precision and recall, the F1-score will be computed for each class individually.

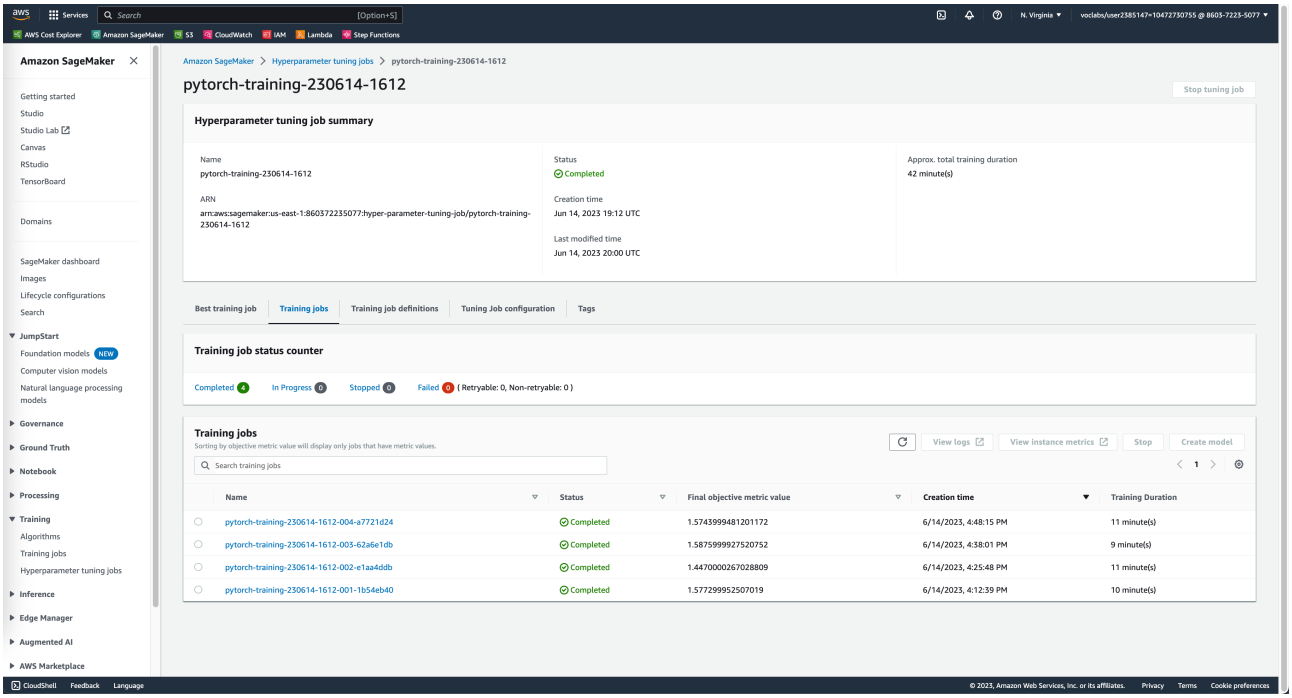
9. Hyperparameter tuning

Hyperparameter tuning plays a critical role in optimizing the performance of machine learning models. It involves finding the best set of hyperparameters, which are parameters that define the model's architecture or behavior and are not learned from the data. Among the essential hyperparameters, batch size and learning rate hold particular importance in model training of CNNs.

The batch size determines the number of data samples processed before updating the model's parameters during each iteration. It impacts both the model's convergence speed and the computational resources required. Tuning the batch size allows finding a balance between processing efficiency and the quality of parameter updates. A larger batch size can speed up training but may sacrifice fine-grained updates, while a smaller batch size can provide more accurate updates but at the cost of increased training time and memory usage.

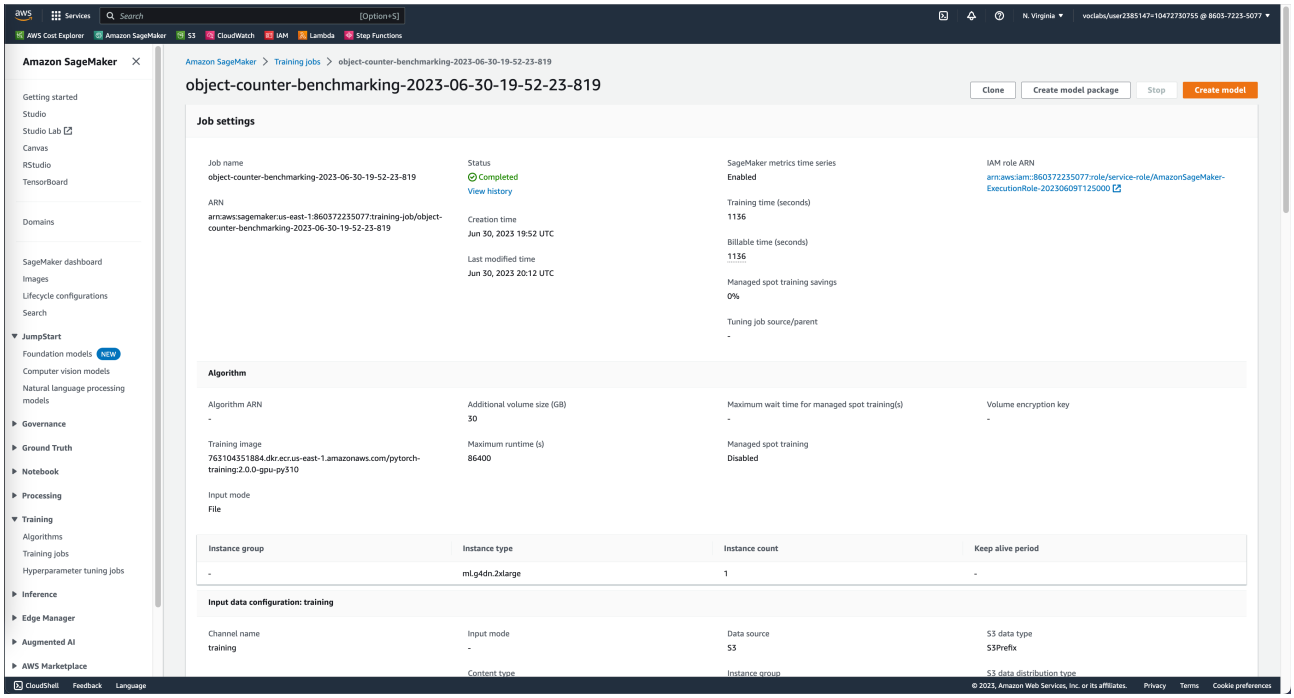
The learning rate controls the step size taken during each parameter update, influencing how quickly the model converges and whether it reaches an optimal solution. Setting an appropriate learning rate is crucial because a high learning rate may cause the model to overshoot the optimal solution or even diverge, while a low learning rate may result in slow convergence or getting stuck in suboptimal solutions. By tuning the learning rate, one can strike a balance between convergence speed and the quality of the final model.

In this project, a preliminary search was conducted to gain insights into how different parameters influenced the model training. Out of the four runs performed, it was observed that the learning rate had a substantial impact on the final test loss, with the best value determined as 0.003. Since a limited number of trials were available to train with various batch sizes, a pragmatic choice was made to utilize an intermediate value of 32.



10. Model training

After performing hyperparameter tuning, the model was retrained with the best hyperparameters for 20 epochs.



During the course of 20 epochs, the validation accuracy exhibited a slight improvement, rising from an initial value of 0.3141 to a higher value of 0.3593. Additionally, the model's performance on the test set yielded an accuracy of 0.3584. While these advancements demonstrate progress, it is evident that further improvements are required before deploying the model in a production environment.

To enhance the accuracy of the model, several strategies can be pursued. First, extending the training process by running additional epochs can provide the model with more opportunities to refine its learned representations and optimize its decision boundaries. This extended training period allows the model to gradually learn from the training data, potentially leading to enhanced generalization capabilities.

Moreover, augmenting the available training data can contribute to performance gains. By introducing a more diverse and comprehensive dataset, the model can encounter a wider range of patterns and variations, enabling it to learn more robust and discriminative features.

The usage statistics below were generated from the profiler report. It shows that the GPU was underutilized, with a maximum utilization of only 18% throughout the training process. The reason for the underutilization of the GPU may be attributed to the small batch size used during training. A small batch size means that fewer samples are processed in parallel, leading to lower GPU utilization.

node	metric	unit	max	p99	p95	p50	min
algo-1	Network	bytes	106776169.25	70	0	0	0
algo-1	GPU	percentage	26	19	19	18	0
algo-1	CPU	percentage	95.27	83.95	52.66	50.51	0.12
algo-1	CPU memory	percentage	7.55	7.46	7.45	7.44	1.27
algo-1	GPU memory	percentage	18	13	13	12	0
algo-1	I/O	percentage	11.79	9.12	8.42	0	0

11. Model performance evaluation

From the results of the model_benchmark notebook that are shown below, we can see that the model trained for only 10 epochs and with no transfer learning performs very poorly. It always predicts the majority class and with this behaviour it achieves an test accuracy of 0.2563 and average test loss of 1.5794.

Class	Precision	Recall	F1
1	0.0000	0.0000	0.0000
2	0.0000	0.0000	0.0000
3	0.2563	1.0000	0.4080
4	0.0000	0.0000	0.0000
5	0.0000	0.0000	0.0000

Average Test Accuracy	0.2563
Average Test Loss	1.5794

We can now compare the values form the benchmark model against the model we have just trained:

Class	Precision	Recall	F1
1	0.5882	0.4918	0.5357
2	0.3525	0.3772	0.3644
3	0.3050	0.4586	0.3664
4	0.3982	0.3814	0.3896
5	0.2121	0.0753	0.1111

Average Test Accuracy	0.3584
Average Test Loss	1.4073

We can see that the model trained for 20 epochs, along with transfer learning, demonstrates significantly better performance compared to our benchmark model. Not only does the model now exhibit improved class discrimination (rather than merely guessing the majority class), but it has also shown an increase in accuracy and a decrease in test loss.

However, our model is still far from being useful for our application, as it achieves an accuracy of only 0.3584. There are several steps we can take to further enhance its performance. Firstly, we could allocate more computing power to train the model by increasing the number of epochs. Additionally, we can augment our training data and ensure an equal number of images per class.

Nevertheless, these results indicate that we are moving in the right direction, and there is still untapped potential for improvement in our model.

12. Model deployment and querying

Finally, although our model still requires improvement before being deployed in a production environment, we can proceed with finalizing the prototype of our machine learning pipeline by making the trained model accessible through an endpoint. To achieve this, we retrieve the output path of the model data generated during the training job and create a PyTorchModel object. It is important to provide the necessary instructions for serializing and deserializing the image data.

The code required to perform inference within an AWS SageMaker environment is available in the 'deploy.py' script, which is utilized during endpoint setup.

Next, we randomly sample images from our test dataset and compare the predicted values with the actual labels. In the example below, we observe that the endpoint successfully outputs a predicted class; however, it does not correspond to the correct class. It is worth noting that this particular image poses a significant challenge in accurately identifying the number of objects, even for a human.

```
Sending the following image to AWS SageMaker endpoint:
```

```
Expecting the following class: 5
Predicted class: 1
```

13. Conclusion

In this project, we have successfully developed a comprehensive machine learning workflow leveraging the capabilities of AWS (Amazon Web Services) services. Our primary objective was to train and deploy an image classification model using these services, and we have achieved that goal.

To streamline our workflow, we made use of AWS's machine learning services. We employed Amazon SageMaker, a fully managed service, to simplify the model training process. SageMaker offered a range of built-in algorithms and pre-configured environments, enabling us to quickly prototype and iterate on our models. We were able to leverage SageMaker's automatic model tuning capabilities to optimize hyperparameters and enhance the model's performance.

In addition to the computational resources, we leveraged AWS's storage services, such as Amazon S3, to store and manage our training data. S3 provided us with durable and scalable object storage, ensuring that our data was readily accessible and protected throughout the training process.

To further improve the model with more computing power, we can increase the number of training epochs to allow for better parameter fine-tuning and capturing complex patterns. Utilizing high-performance GPUs can expedite the training process and potentially enhance performance. Distributed training across multiple instances can accelerate convergence and yield improved results. Exploring more sophisticated model architectures, conducting extensive hyperparameter tuning, and leveraging ensemble learning techniques can also contribute to enhancing the model's performance. However, it's important to consider the associated costs and resource requirements when allocating additional computing power.

Overall, by utilizing AWS services, we established an end-to-end machine learning workflow that encompassed data storage, compute resources, model training, and monitoring. The integration of these services enabled us to efficiently train an initial model, bringing us closer to achieving our project objectives.