# Deep Learning Models of Scanner/Vision Tunnel Performance in Sortation Subsystems

by

Felix Dumont

B.S. Mathematics and Computer Science, McGill University, 2015

Submitted to the MIT Sloan School of Management and
MIT Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Master of Business Administration and
Master of Science in Electrical Engineering and Computer Science
in conjunction with the Leaders for Global Operations program
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
June 2021



Author .....
MIT Sloan School of Management and
MIT Department of Electrical Engineering and Computer Science
May 14, 2021

Certified by .....
Duane S. Boning
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Certified by .....
Negin (Nicki) Golrezaei
Assistant Professor of Operation Management
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Accepted by .....
Maura Herson
Assistant Dean, MBA Program
MIT Sloan School of Management

# Deep Learning Models of Scanner/Vision Tunnel Performance in Sortation Subsystems

by Felix Dumont

Submitted to the MIT Sloan School of Management and
MIT Department of Electrical Engineering and Computer Science
on May 14, 2021, in partial fulfillment of the
requirements for the degree of
Master of Business Administration and
Master of Science in Electrical Engineering and Computer Science
in conjunction with the Leaders for Global Operations program

## Abstract

We propose an end-to-end process and tool to deep-dive scanner issues at Amazon's sorter sites, allowing us to categorize no-reads into operational issues or actual equipment issues.

Our tool sends no-read scanner images to a separate Amazon Web Services (AWS) server and post-processes them through ResNet deep learning models tuned through Bayesian optimization to appropriately assign potential fault reasons. This program will grow the team's understanding of material handling equipment and best practices to trigger and handle exception case scenarios. A conservative entitlement is approximately $2.2MM for the pilot sites in annual savings excluding customer impact.

Scanner/Vision tunnel performance at Amazon's large crossbelt sorter sites tends to average around 80-90% read rate success, contributing to a large amount of manual rework and recirculation impacting sorter utilization. Amazon is well away from their target of 98% scanner performance for these sites. Furthermore, the mechanism to deep-dive scanner issues makes it extremely difficult to categorize them into operational issues or actual equipment issues and as a result, we have very little visibility as to no-read causes across sites and cannot properly put together a plan to improve the situation.

A user-friendly interface allows site and operations managers to see which sites are lagging behind, perform a deep-dive into the root cause of the issues and test potential operational or equipment fixes.

Thesis Supervisor: Duane S. Boning
Title: Professor, Department of Electrical Engineering and Computer Science

Thesis Supervisor: Negin (Nicki) Golrezaei
Title: Assistant Professor, MIT Sloan School of Management

# Acknowledgments

I would like to first thank Amazon for making this internship project possible through their partnership with the Leaders for Global Operations (LGO) program. In particular, this thesis would not have been the same without the constant help and guidance provided to me by Kevin Vo and Dan Hartley. I would also like to extend my gratitude to Kevin Lyons for sponsoring this project and trusting me with such an incredible opportunity. I have learned a tremendous amount over the last year and couldn't have done nearly as much without your support.

I would also like to thank my thesis advisors, Professors Duane Boning and Negin Golrezaei, who provided wonderful insights and structure throughout this project and helped me turn ideas into an impactful product. Their involvement in the LGO program and support in conducting industry research truly makes a difference! Furthermore, this thesis wouldn't have been possible without the constant support from the LGO program and the Fonds de recherche du Québec – Nature et technologies over the last two years.

Finally, while the circumstances have not made it easy, I am incredibly thankful for all the friendships I have made over my time at MIT and wish all my classmates the best. I would like to thank my family and my wife Cristina for being such a strong anchor, even in the middle of a pandemic. I owe so much of what I have accomplished to you!

# Contents

# List of Figures

# List of Tables

# Glossary

**Amazon Web Services (AWS) SageMaker** A cloud machine-learning platform allowing users to prepare data, and build, train, and deploy ML models

**canny edge** An edge detection technique developed in 1986 by John F. Canny

**class** The specific category to which a data observation belongs. Also called label.

**convolutional neural network (CNN)** A type of deep neural network that utilizes convolutions instead of matrix multiplications in some of its layers. Most commonly used for image classification tasks.

**crossbelt sorter** A common type of sortation system relying on powered belt conveyors.

**fulfillment center (FC)** FCs store products you can buy on Amazon.com and fulfill orders across the world. They are divided into sortable and non-sortable FCs. .

**hyperparameter** A machine learning model parameter whose value is set before the model training phase and dictates the learning process

**no-read** A no-read image is a picture taken of a carrier whose package's shipping label could not be read by a scanner

**residual neural network (ResNet)** A type of convolutional neural network introduced in 2015 using shortcuts between layers to simplify the network and allow a deeper representation.

**transfer learning** A machine learning technique where knowledge gained from a task is leveraged for a different but similar task.

# Chapter 1

# Introduction

This thesis focuses on the applications of computer vision for fault-processing in Amazon's sortation systems. As such, in this first chapter we introduce the reader to Amazon and its fulfillment process, as well as the scanner no-read problem and the various faults we can observe. Then, we share details on the scope of this thesis and on the solution we propose to improve the situation across the network.

## 1.1 Background

### 1.1.1 Company overview

Amazon, headquartered in Seattle, is a multinational company with established offerings in e-commerce, cloud computing, digital streaming, grocery and more. As of 2020, Amazon acquisitions cover a variety of industries and include companies such as Twitch, IMDB, Whole Foods Market, Zoox and Zappos. Amazon was founded by Jeff Bezos in Bellevue, Washington, on July 5,1994 with the goal of becoming "Earth's most customer-centric company". Amazon has been a partner of the MIT Leaders for Global Operations (LGO) program since 2003 and, as of 2019, had a total of 798,000 employees across the world and a yearly revenue over $280B.

### 1.1.2 Amazon fulfillment network

The Amazon fulfillment network is majoritarily composed of three types of sites: Inbound Cross-Docks (IXD), Fulfillment Centers (FC) and Sort Centers (SC).

FCs store products you can buy on Amazon.com and are used to fulfill orders across the world. They can also be further divided into sortable and non-sortable FCs.

Sortable sites usually contain items that can fit into 18 inch totes. These sites are the most common types of FCs and most of the newer sites will rely on Amazon Robotics to automate several parts of the process. They will also heavily rely on sortation scanners to properly redirect packages. Sort centers are usually around 800,000 square feet in size and can employ more than 1,500 full-time associates.

Non-sortable FCs mostly contain large items such as furniture or large electronic devices and range in size from 600,000 to one million square feet. Because of the nature of the items contained at these sites, non-sortable FCs tend to be significantly less automated and may not rely as much on sortation scanners.

IXDs allow Amazon to receive, prepare, and store product that comes directly from the vendor. This reduces processing time and space needed at FCs, but also gives vendors fewer delivery points of contact

SCs receive boxed packages from FCs and take the place of similar postal facilities to break down deliveries by exact delivery areas. FCs will usually group packages going to the same metropolitan area while SCs will narrow it down much further to allow for faster delivery.

There exists other specialty sites, such as Amazon Fresh or Amazon Air Hubs, but these either do not handle the same amount of volume or have a fundamentally different structure. For these reasons, we have decided to exclude them from the scope of this work.

### 1.1.3 Outbound process

The outbound process slightly differs across sites but the main steps are common as shown in figure 1-3. Once an order for an item is registered, an associate receives the instructions to pick it, either manually or through a mobile robotics fulfillment system.

Once picked, the item goes through the preparation process which involves combining the item with other items from the same order and applying labels.

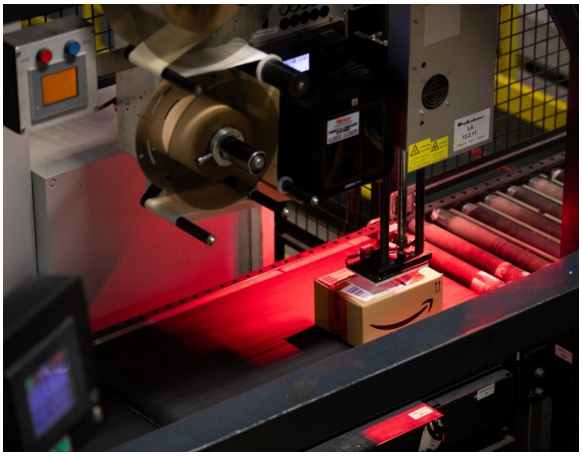


Figure 1-1: Shipping label automatically applied to a package [1]

The package will then be inducted onto the line, meaning an associate will deposit it on a conveyor belt to be sorted for shipping. This induction step can be a major source of issues for shipping, as human or equipment issues can cause the package to flip or end on the wrong carrier.

After their induction on the conveyor belt, packages go through the sortation process, where the final shipping label is applied and scanners redirect packages to the correct shipping chute.

Finally, associates carefully pack packages into each chute's truck, before the packages leave the site. Trucks have a very rigid timeline and as such will not wait for any late packages. Late packages will need to be put on the next transport, risking potential delays or expedited shipping costs from Amazon to arrive in time.

Figure 1-2: Package inducted on the main sorter (Dematic)[2]

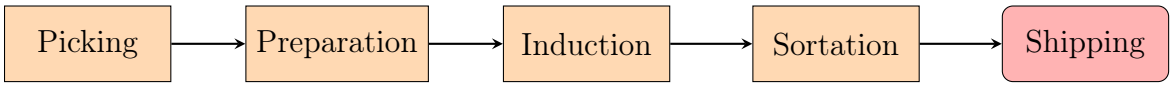
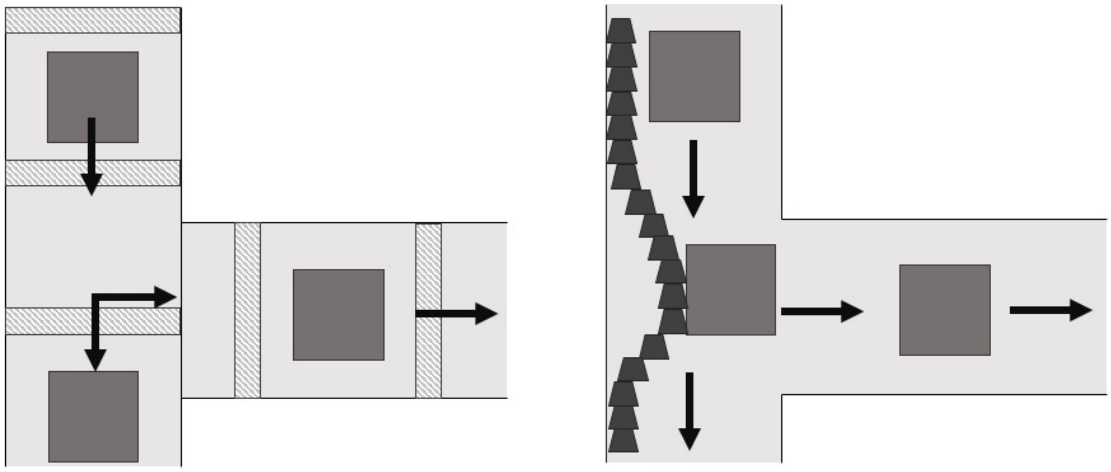



Figure 1-3: Typical outbound process in a fulfillment center

### 1.1.4 Sortation systems

Our work focuses on the sortation step of the outbound process. Sortation systems handle the identification of parcels on the conveyor systems and divert them towards the appropriate final destination. Amazon uses multiple types of sorters to this end, including crossbelt sorters and shoe sorters. Shoe sorter's carrying surface consists of conveyor flights or slats driven on the outer edges by a chain [3]. Shoe sorters can only handle parcels (boxes) since flexible or loose packaging may get caught in the shoe and crash the sorter. Crossbelt sorters instead rely on powered belt conveyors on each carrier predictably moving packages and allowing them to carry a wider range of products. Figure 1-4 shows a visual comparison between crossbelt sorters and shoe sorters.

We mainly focus on crossbelt sorter as they represent most of Amazon's volume and have a scanner defect rate up to five times as high as most shoe sorters. Crossbelt sorters' closed loop conveyors are used in several types of FCs and sort centers and carry parcels of all types (flats, cartons, polybags, smartpak and totes) which can be inducted from several places on the sorter. Sorters then rely on scanners to read shipping labels and divert parcels in the designated chutes for their final destination.



(a) Crossbelt sorter with carriers of fixed length

(b) Shoe sorter

Figure 1-4: Comparison of the main sorter types

### 1.1.5 Scanner / vision tunnels

Sortation systems heavily rely on third-party scanners from two major vendors. The most common type of scanner is a multi-sided scan tunnel, as seen in figure 1-5, designed to locate shipping labels from all six sides. All scanners attempt to find a shipping label using proprietary vendor software, and if none is spotted, a separate camera will take a picture of the carrier and save it to a local machine for Amazon to investigate.

It is important to emphasize that images uploaded for incorrect reads are therefore different from what the actual scanners captured. However, using separate cameras for this task removes some burden from the scanners and ensure that the additional uploaded images do not affect them. As a result, any change we do to the no-read cameras or modification to the image processing will not affect scanner accuracy and sorter utilization.

Images where no complete shipping label can be identified are called no-reads and lead to heavy manual labor. Packages in no-read images will end up in a separate chute called "jackpot" and associates will need to manually investigate potential issues

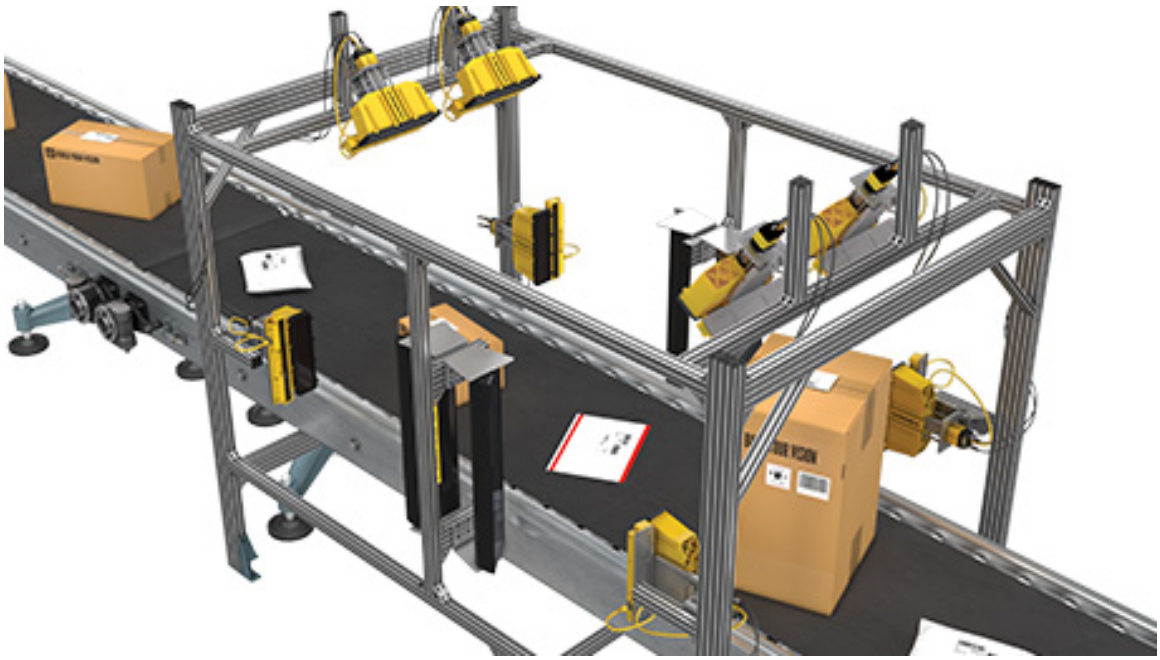


Figure 1-5: Multi-Sided Scan Tunnel (Cognex Corporation)

and send the packages to the induction step again.

## 1.2 Problem statement

With the rise of one and two day guaranteed deliveries, FCs are under increasing pressure to meet their targets and reduce operational issues. A recurring issue sites have experienced relates to scanners on sortation systems, at the end of the fulfillment process. Scanners provided by third-party vendors scan every package on the conveyor belt in order to determine what chute it should be redirected to. However, up to 10-20% of scans are unsuccessful across new sites, leading to a high ratio of unexplained no-reads. No other information is recorded along no-read scans, making it difficult to attribute them to specific equipment or operational issues.

These no-reads lead to hundreds of thousands of packages either ending up in the wrong chute or needing manual intervention every day. Both of these scenarios are very costly for Amazon but more importantly, can negatively impact the customer experience. As such, Amazon is in dire need of an automated and scalable process to deep-dive no-reads across sites and narrow down issues for site operations.

## 1.3 No-read causes

No-read images occur whenever scanners cannot properly read a unique shipping label on an image. This can occur following a variety of reasons, such as when there is no package or multiple packages on a single carrier. We detail here some of the main causes observed across North America, although each site can encounter different root causes for no-reads.

### 1.3.1 Absence of package

The first potential cause for a no-read is the absence of package on an image. This can happen if a package ended up on the wrong carrier or fell of the belt entirely. It can also indicate in some cases equipment issues or incorrect scanner triggers. If we can observe any part of a package in the image, we consider the image as containing a package.


Figure 1-6: Empty carrier

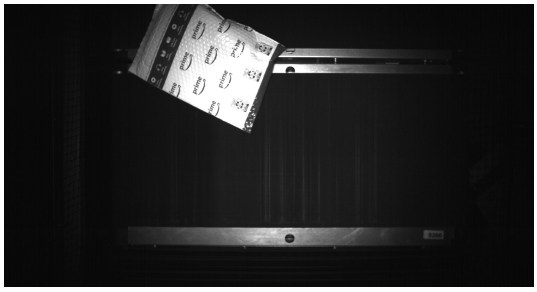
### 1.3.2 Multiple packages

Some images may instead contain multiple packages. An image with multiple packages can indicate a misaligned scanner, an incorrect induction or various crossbelt-related issues. A variety of issues can then arise from such scenarios. For instance, if only one label is read, both packages may be redirected to same chute when only one was meant to end up there. On the other hand, if more than one label are detected, no package can be redirected at all.

Figure 1-7: Multiple packages in the same image

### 1.3.3 Package position

While systems are designed with the expectation that packages will be straight in the middle of carriers, their position can often be problematic. A bad induction or system can cause packages to overlap between two carriers as in figure 4-3 or end up in the bellows which are located after every two carriers in most sites. Packages overlapping between carriers can be unpredictable and may be ejected with either of the two carriers they touch, possibly resulting in the package ending up in the wrong chute. Packages on the bellows on the other hand can be stuck and in some cases



(a) Package overlapping between carriers

(b) Package on the bellows

Figure 1-8: Incorrect package positions: packages can overlap between carriers or be stuck on the bellows

may recirculate for hours until an associate can manually remove them during the next break.

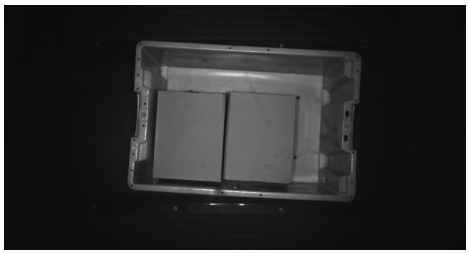
### 1.3.4 Package type

An important aspect to consider about no-reads is the type of packages observed. While different types of packages will not directly cause no-reads, some packages can behave very differently on the conveyor belt. For instance, lighter packages such as envelopes can flip easily or end up on the wrong carrier. Throughout this document we will consider four main types of packages: boxes, totes, plastic packages and Jiffy envelopes as seen in figure 1-9. Totes typically contain multiple packages but any image containing a tote should be labelled as a tote, regardless of the type of packages it holds. Another important note is that the box category includes both Amazon boxes as well as non-Amazon branded boxes shipped as is.

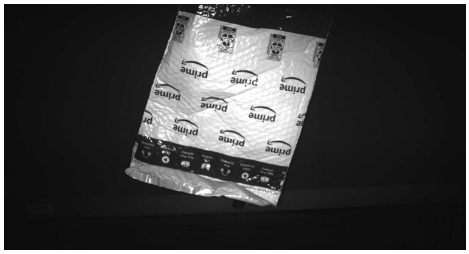
### 1.3.5 Package damage

Package damage can be a major source of concern for FCs. Indeed, not only can a damaged package cause incorrect scans, but it could also lead to spillage or damage other packages. However, since package damage can be difficult to spot from a single camera image, we did not tackle this issue in this project. Furthermore, package damage is not as common as the previously mentioned issues, hence why we ended up deprioritizing it.

(a) Box



(b) Tote




(c) Plastic

(d) Jiffy envelope

Figure 1-9: Four main types of Amazon package

### 1.3.6 Label issues

Finally, labels can also be damaged or incorrectly positioned which can lead to a partial read of the bar code. This type of issue can sometimes be difficult to spot for associates and the same package may be rejected several times in a row. Similarly to package damage though, it can be difficult to spot label issues in the scanner images and as such we will not integrate label quality in this project. However, this can be a major source of issues for some sites and will be discussed further in section 6.

## 1.4 Project overview

### 1.4.1 Goal and scope

The overall goal of this thesis and the associated six months spent at Amazon is to develop a tool allowing operators and engineers to conduct deep dives into the causes of no-reads across most of the Amazon Fulfillment network. More specifically, three components of the tool are critical for the end-users:

1. The ease of use and overall user experience

2. The number of potential causes addressed

3. The accuracy of the tool

The user experience can be measured through preliminary user tests and gathering qualitative feedback while the breadth of the tool can be evaluated through the number of issues covered from Section 1.3.

Although our goal was originally to focus on sites supplied by one of the two major vendors, the second vendor was later integrated within the scope and most fulfillment centers in North America ended up being in scope for the tool.

### 1.4.2 Approach

In order to tackle this problem, we built a pipeline in Amazon Web Services (AWS) to store and preprocess no-read images from FCs. Then, we developed a suite of deep learning models to analyze no-read images from sites and automatically label them. The following models were developed for each input image:

1. Is there at least one package present in the image?

2. Is there more than one package present in the image?

3. Are there packages on the bellows? Or overlapping between two carriers?

4. What package type can we see?

Models 2, 3 and 4 will only be executed on an image if the first model determined there was at least one package in the image.

Finally, we built a front-end interface for users to visualize trends and causes of no-reads and to help them guide their next actions.

The entire process was tested in collaboration with multiple sites in order to optimize the user experience and handover.

## 1.5 Thesis organization

This thesis is organized in six chapters, with the goal of guiding us through the process of developing and deploying the tool, as well as highlighting the uses and benefits it can generate.

**Chapter one** introduces the reader to Amazon, sortation systems, the causes of scanner no-reads as well as our proposed solution. **Chapter two** then links this research to various other past or ongoing efforts through a literature review. **Chapter three** details the data we used for this thesis as well as the processing we used. **Chapter four** documents all of the steps in the modeling process as well as technical details on the deployment of our tool to the cloud. **Chapter five** then summarizes the test results and accuracy, shows real applications of this tool among crossbelt sorters and describes the limitations and next steps. Finally, **chapter six** wraps up this thesis with some concluding remarks and next steps.

Throughout this document, a particular emphasis is given to the use of Amazon Web Services (AWS) as well as AWS SageMaker, its cloud machine learning platform. Our literature review showed us little documented uses of the latter, but we firmly believe SageMaker can greatly facilitate the development of machine learning tools and aim to display this throughout this thesis.

# Chapter 2

# Literature review

In this chapter, we introduce the reader to previous work on the topic of sortation systems, machine learning for fault detection and AWS SageMaker. Although academic work can be sparse on some of these topics given their industrial nature, some work analyzed in this section can help guide our solution. Additional details on the specific implementation and machine learning behind our solution will be present in chapter 4.

## 2.1 Sortation systems

Although sortation systems are consistently optimized, the overall design and steps have been relatively unchanged over recent years. Knill et al [3] detail the process, from the package induction all the way to the final package chute. The article dives into the difference between Amazon's main two types of sorters, shoe and crossbelt sorters. Shoe sorter's carrying surface consists of conveyor flights or slats driven on the outer edges by a chain. Crossbelt sorters instead rely on powered belt conveyors on each carrier predictably moving packages.

Innovation still found its way to sortation systems. Ken Ruehrdanz, warehousing and distribution market manager for Dematic [4] mentions that "the most significant advancement in sortation solutions today is the migration toward software architecture that treats all the associated sub-systems as one integrated sorter system". The

article further mentions software has been used to optimize every aspect of sortation systems, for instance by minimizing the gap between products to increase throughput.

Clyde E. Witt, editor in the Material Handling Management magazine, also argues that "tunnel scanners have enabled e-commerce, catalog retailing and high-speed distribution centers to achieve the throughput that customers demand" [5]. Furthermore, Cognex, a global provider of tunnel scanners and machine vision solutions acquired ViDi Systems in 2017, a company focusing on industrial machine vision as part of a push towards more advanced scanner solutions [6].

In a thorough review of automated conveyors, Boysen et al [7] describe the applications and challenges of common sortation systems from an operational research perspective. They also mention the rise of same-day delivery as a major motivation for large-scale automation. Boysen et al argue that Automated Sorting Systems (ASS) are fast, scalable, flexible and reliable, making them essential for a robust supply chain. The authors describe two main uses of ASS in the outbound fulfilment process: First, they can be used as part of the picking and packaging process and second, ASS can be used to direct packages towards transport providers in the shipping area. As part of this thesis, we will focus on this second application.

## 2.2 Machine learning for fault detection

Machine learning solutions for sortation scanners are typically either developed by third-party vendors who will carefully protect their intellectual property. As such, few details are available on the current solutions.

However, using deep learning for fault or defect detection is getting increasingly more common in both manufacturing and distribution environments. Caggiano et al [8] demonstrate how deep learning can be used for on-line defect detection in additive manufacturing, achieving a 99.4% accuracy, an improvement from the 91.2% and 96.5% accuracy obtained respectively from visual words and Histogram of Oriented Gradients, two common techniques in the field.

Past LGO theses have also leveraged deep learning for visual inspection. James

Smith (LGO 2018) [9] uses transfer learning to automate the visual inspection of uncured rubber tire assemblies, reducing manual inspection time by 16-70%. He leveraged the Inception-V3 model Our proposed approach will be similar to his final transfer learning model architecture, shown in figure 2-1.

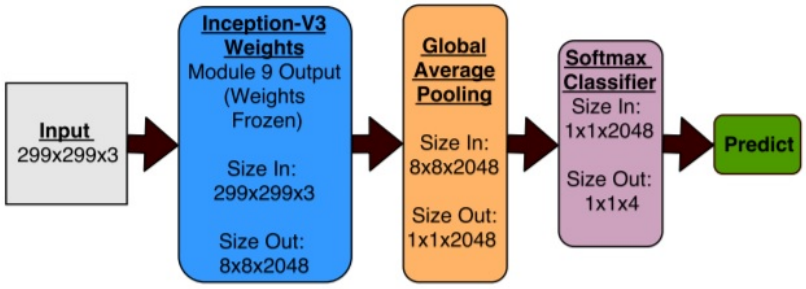



Figure 2-1: Final transfer learning model architecture used by James Smith as part of his visual inspection of uncured rubber tire assemblies

Amazon's operations engineering division, sponsor of this thesis, is also not new to the use of machine learning for fault detection. In 2019, Bidusha Poudyal investigated the use of predictive analysis in the Installation and Operational Qualification (IOQ) process and developed machine learning models to identify potentially problematic equipment, paving the way to the use of machine learning in operations engineering.

## 2.3 AWS SageMaker

Little formal academic research mentions AWS SageMaker, possibly because of the platform having been only launched in 2017 and the predominance of on-premise servers in the academic world. However, in this thesis we evaluate the potential applications of AWS SageMaker within both academia and industry along with the strengths and limitations of this platform. AWS SageMaker claims to serve many leading organizations worldwide such Tinder, GE Healthcare, Hotels.com, Dow Jones and Thomson Reuters. Ashok Srivastava, Chief Data Officer at Intuit, further mentions that: "with Amazon SageMaker, we can accelerate our Artificial Intelligence initiatives at scale by building and deploying our algorithms on the platform. We will create novel large-scale machine learning and AI algorithms and deploy them on this

platform to solve complex problems that can power prosperity for our customers." [10]

Khan et al [11] leveraged AWS SageMaker along with AWS DeepLens to diagnose diseases of plant leaves and achieved a 98.78% accuracy using transfer learning on a Deep Convolutional Neural Network (DCNN). The inference time of their final deployed model is of 0.349s per image. Among the algorithms considered for this task, ResNet50 performed with an accuracy of 97.85%, slightly below the DeepLens accuracy, but higher than VGG-16 at 96.89%. Furthermore, the authors mention the scalability of AWS SageMaker and cloud computing as a contributing factor to a realistic deployment of their final algorithm.

Amazon SageMaker also aims to reduce the complexity of tuning models as we will discuss in chapter 4. In December 2020, Amazon unveiled the Amazon SageMaker Automatic Model Tuning (AMT), a fully managed system for black-box optimization at scale [12]. AMT automates the hyperparameter search process and works for both built-in and custom machine learning algorithms.

# Chapter 3

# Data processing

The deep learning solution we propose in this thesis is tailored to Amazon's scanner no-read images. In this chapter, we share details about these images and their provenance, as well the steps we take to manually apply the relevant labels to them. We also provide details on the cleaning and augmentation process which allows us to later obtain an accurate classification with the algorithms described in chapter 4.

## 3.1 Data description

### 3.1.1 Site coverage

Most types of large sites have a tunnel scanner and as such we test our algorithms on multiple types of FCs as well as sort centers. However, we end up tuning the final models using large cross-belt sorters, as these are the sorters handling the most volume and where issues are the most common. Cross-belt sorters can handle most type of packages, as opposed to shoe sorters which can only handle boxes. While over a hundred sites in North America belong to these categories, we select ten sites to use to train our models, given the considerable time requirement to obtain and label the data. These sites are chosen to be fairly representative of the larger set of sites in terms of age, size and equipment. They cover both major scanner vendors as well as the three main vendors of integrated sorter solutions.

### 3.1.2 Format

All images are saved in grayscale with the resolution depending on the scanner settings but most commonly being 2048x1088. While some sites provide images from all six sides, most sites only provide a top view and so we only considered these images. The top view clearly shows the top of the package, as well as the carrier and, in some cases, the edges of adjacent carriers. However, it is worth mentioning that the brightness, angle and zoom can vary from site to site due to inconsistencies in the initial camera setup. No other logs or labels are available along with these images.

Overall, the slight variations in the quality or format of the images across sites can be significant enough to impact models. This emphasizes the importance of including multiple sites and vendors to improve the consistency of our tool. For instance, early results showed that a model trained on a single site struggled to predict labels for a second site. However, models described in chapter 5 trained on ten sites perform significantly better on an eleventh site, indicating the importance of a rich training dataset.

## 3.2 Data labelling

We manually label over 2300 images across ten sites in scope for our supervised models. These images receive labels for each of the four models we develop. Since no labelling had previous been done on no-read images, we review as a team the classification and the criteria to make sure the labelling was as objective and consistent as possible. In particular, the position label can be subjective, so we label packages according to their apparent center of mass.

Some of the labels can however be ambiguous for both humans and machines. For instance, one of the labels for each image is whether the package is centered or not in the carrier. We determine that the best approach is to assign a centered label whenever the center of mass of the package appears to be on the carrier, but this can be difficult to determine by looking at images only.

Another important aspect of our labels is that we only create classification labels

and do not perform image segmentation or any other form of labeling. Image segmentation could allow us to specifically identify where in the image are packages and bellows located but would require a significant amount of time. Given the number of images in scope, we therefore opt to only assign overall labels to perform traditional classification.

## 3.3 Data cleaning

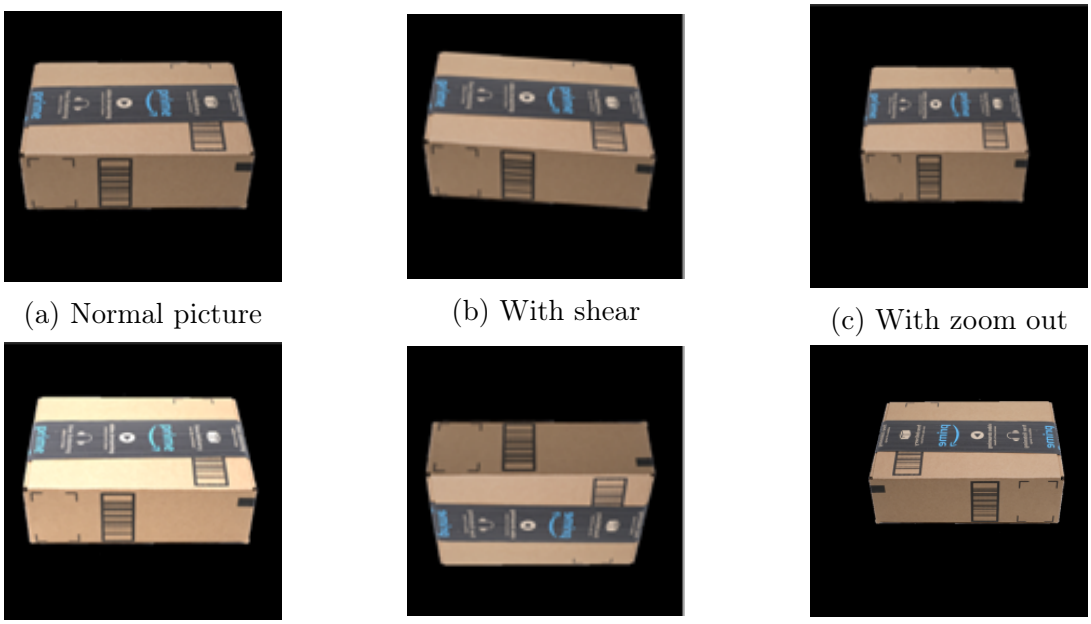
In order to ensure consistency and improve accuracy, we automatically perform data cleaning on training and inference images. All images have 10% of the left and right side cropped, as these edges do not cover the conveyor belt and only introduce noise. Then, we resize images to a 224x224 format and convert them from grayscale to an RGB format, with the grayscale value repeated across all three channels.

One of the scanners used in crossbelt sorters returns a fixed image size which facilitates the process. However, the other type of scanners returns an image of variable size, as the height of the image will depend on the trigger length for a given package. This may result in slightly distorted resized images, but the algorithms still show to be able to handle them as shown in section 5.

## 3.4 Data augmentation

While we have access to a large dataset, with millions of images generated every day, no labels are available. Since the amount of time required to label images across all of our categories is significant, we need to resort to data augmentation to amplify the size of our dataset without manual input.

This is particularly critical for our task given pre-trained image classification models have been shown to display poor invariance to small transitions, with a one pixel perturbation sometimes affecting the classification of up to 30% of images [13]. Since we expect our packages to be present anywhere in the image and possibly rotated in all kind of ways, data augmentation will help us achieve the desired robustness.



(a) Normal picture (b) With shear (c) With zoom out

(d) With brightness shift (e) With vertical flip (f) With horizontal flip

Figure 3-1: Illustration of image augmentation techniques used

In order to generate those augmented images we apply the following transformations across all of the model training sets:

1. Images are applied a zoom between 0 and -10% with a black filling. Note that we are zooming out, not in, meaning that no package will be accidentally discarded.

2. The brightness is adjusted by a random factor between -15% and +15%.

3. A shear is applied using the the Python Keras [14] library with a random factor between 0 and 5%.

4. A vertical and/or horizontal flip can be applied, with a 50% probability for each.

For images in the validation set, we only apply transformations which could happen naturally to increase the size of our image set. Hence, only the horizontal and vertical flips are applied, with the same 50% probability. All images in the final test set are left untouched.

We can visualize these augmentation techniques on an illustrative image in figure

The final data split after the augmentation process consists of 5,000 training images, 1,000 validation images and 748 images in the final test set. We sample images such that the training and validation sets are randomly sampled from the same combinations of sites, days and scanners. As such, the observed ratios and images are very similar. The test set however is selected from the same sites but using different scanners or days. As such, the ratios of observed issues and the types of packages can vary. This test set is therefore more representative of a real scenario, where accuracy may not be as high as in the validation set.

# Chapter 4

# Methodology and tool design

In this chapter, we start by detailing the training and inference pipeline which allows us to deploy this tool in production. Then, we introduce techniques to leverage the blue and red channels of our images to contain additional features, so that we can tackle our position and package count models. In the following two sections, we guide the reader through our model architecture, choice of deep learning models and hyperparameter tuning process. Finally, we describe how we leverage AWS SageMaker to deploy our models to the cloud using a multi-model inference endpoint.

## 4.1 Pipeline overview

Since the goal of this internship is to not only train machine learning models but actually deploy them in production, we have to fully leverage AWS' offerings. To this end, we develop a three step process.

First, we proceed to lead code development, such as preprocessing images, on our local machine using a private Git repository. We can then transfer the code through a mounted elastic file system to our training environment. This training environment consists of an AWS SageMaker Notebook instance containing the required image classification code. This Notebook has access to two GPU-powered cloud machines to train and tune the models. Finally, we deploy our trained models to the cloud using a SageMaker multi-model endpoint and an EC2 instance performing the preprocessing.

All of these instances can interact with our S3 bucket containing all of the no-read images. As a result, any new set of images uploaded to the S3 bucket will trigger the inference code.

While there are several components to this pipeline, it is built such that each compute or storage resource is optimized. The powerful but expensive GPU-powered instances are only used in a short timeframe for tuning, whereas the multi-model endpoint which is continuously open for inference is hosted on a cost-effective instance.

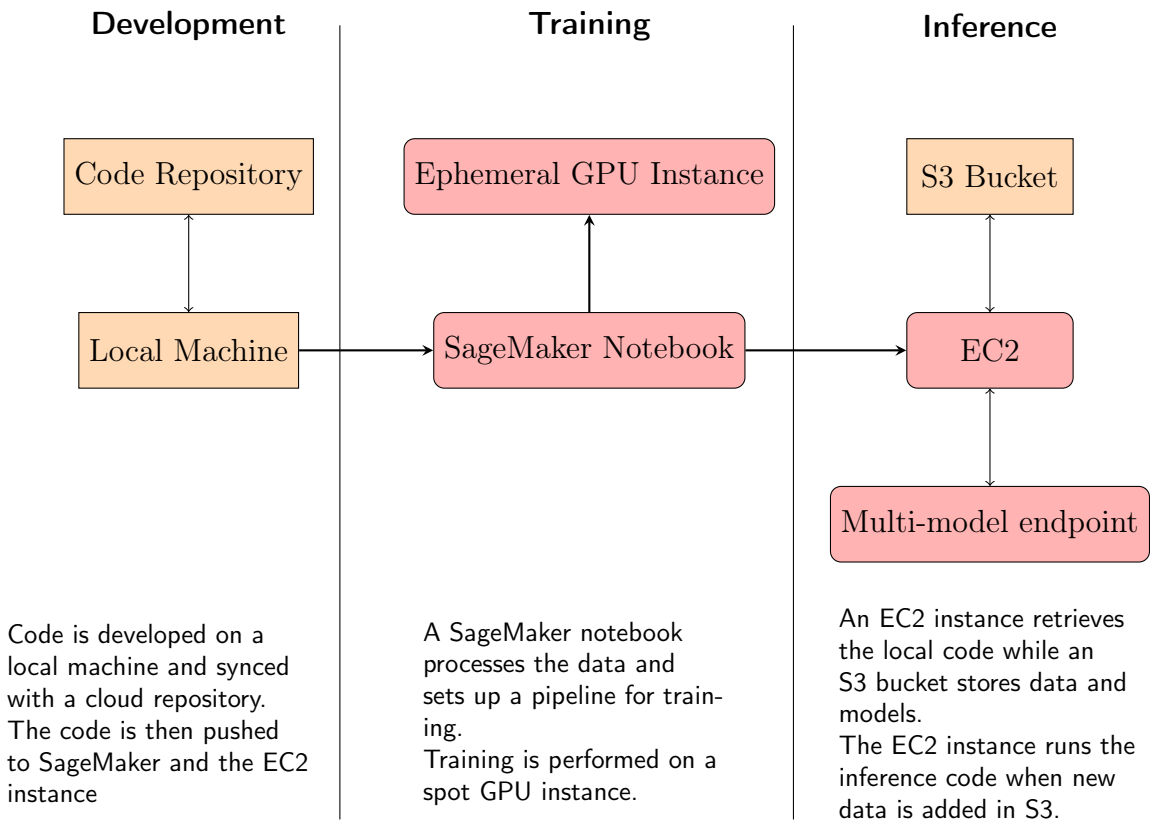



Figure 4-1: Our system architecture distributes tasks to multiple EC2 instances (in red).

Table 4.1 summarizes the AWS instances and services we use and their associated public costs. Our final production tool costs under $150 per month, with an additional estimated $190 for each month of model development. Since all of these instances are easily scalable, we can upgrade some of the resources as needed. In particular, the current endpoint uses an inexpensive t2.medium instance but could be improved to a GPU instance if the demand for the tool increases.

Table 4.1: Overview of the AWS requirements and estimated monthly costs for the pilot using public US-East server fees. Development costs are only incurred during the model development phase.

| | | Cost ($ per month) | |
|---|---|---|---|
| Instance | Task | Production | Development |
| c4.large | Handles tool requests and processes data | 72 | - |
| t2.medium | SageMaker multi-model endpoint | 40 | - |
| S3 storage | Storing data and hosting the website | 15 | - |
| t2.medium | Training SageMaker notebook | - | 40 |
| p2.xlarge | GPU Training (for 140 hours per month) | - | 150 |

## 4.2 Feature generation

One of the novelties introduced in this thesis is the use of red-green-blue channels as conveyors of additional features in the context of grayscale images. Since we only have access to grayscale images, we repeat the intensity value across the three channels. The intensity of a pixel is a single value ranging from 0 (black) to 255 (white) and can be used to represent any shade of gray. However, we can instead leverage two of these channels to store other information, namely a canny edge detection feature and a bellow highlighting technique.

### 4.2.1 Canny edge

As a first processing technique, we leverage the OpenCV [15] canny edge detection technique. Canny edge is an edge detection technique developed in 1986 by John F. Canny [16]. This technique aims at finding and highlighting edges within an image, leaving all other pixels black. This algorithm follows four steps:

1. A 5x5 Gaussian filter smoothes the image and removes the noise.

2. For each pixel, we calculate intensity gradients $G_x$ and $G_y$ in both the x and y directions. Then, we can define a pixel gradient as $G = \sqrt{G_x^2 + G_y^2}$ with angle $\theta = tan^{-1}(\frac{G_y}{G_x})$. The gradient is further rounded to be either diagonal, horizontal or vertical.

3. Non-maximum suppression removes pixels which are not a local maximum in their neighborhood in the direction of the gradient in order to thin the edges

4. Configurable thresholds for the gradients decide which pixels should have a value of 255 and which should have a value of 0 through a process called hysterisis, based on a min and a max threshold.

   We iterate through multiple threshold values over a set of 100 images and visually inspect the resulting edges. We then determine that the optimal values for the min and max gradient threshold should be respectively 40 and 75 for our dataset. Therefore, hysterisis ensures that any pixel with a gradient above 75 is considered as an edge, and so is any pixel with a gradient between 40 and 75 as long as it is adjacent to another edge pixel. Other values are discarded.

Finally, we feed the canny edge output as part of the blue channel of our red-green-blue images. This output will either be 0 (black) or 255 (blue) and make it so that edges will visually stand out in blue in the resulting images.

### 4.2.2 Carrier limit detection

The second technique we develop is a rule-based approach to detecting the metal separators between carriers. Crossbelt sorters usually have larger separators on one side where the bellows are and smaller separators between carriers. Each set of two carriers is typically between bellows. We can see an example of this in figure 4-2. Since parts of the bellows are often missing from the image frame, early iterations of our algorithms struggled to separate instances of packages on the bellows or overlapping between carriers. This motivates the addition of manual preprocessing to ensure proper emphasis of these features and more accurate evaluation of package position.

The algorithm we develop to highlight bellows looks at each image row by row and flags all rows where at least 75% of the pixels are brighter than the median image pixel intensity as these rows should contain a horizontal metal bar. Indeed, given most of the image will consist of the carrier, the median intensity will roughly correspond to the carrier's black background. We test various threshold using a set of images on

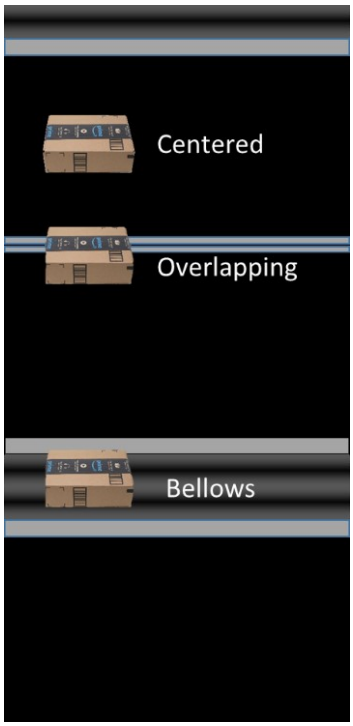



Figure 4-2: Typical crossbelt setup with bellows between sets of two carriers. Images only capture a single carrier and part of the bellows

the train set, and 75% displays the best highlighting results after a manual review of the highlighted images.

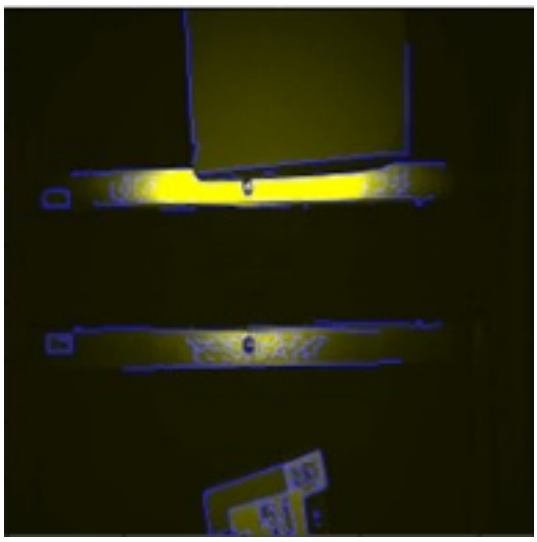
We then split the image in two horizontally and label the side with the largest horizontal bars as the bellows side and the other as the carrier side, where overlapping can happen. We finally assign a bright red value of 255 for each metal row on the carrier side and a light red value of 128 for each metal row on the bellows side.

## 4.3 Model architecture

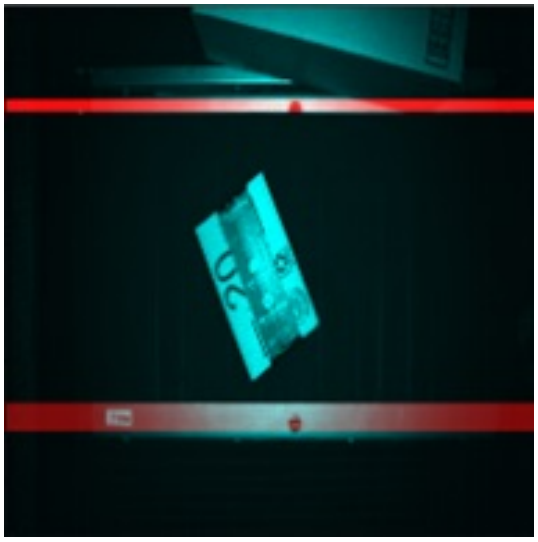
### 4.3.1 AWS SageMaker

While the first prototype of this tool was developed locally, we decide to build the final production tool using AWS SageMaker.

SageMaker is Amazon's cloud-based machine learning platform. Users can explore data and develop models in Jupyter Notebooks or run Python code directly on



(a) Canny edge processing



(b) Bellow highlighting

Figure 4-3: Preprocessing techniques

the remote EC2 instance console. SageMaker is designed to handle all steps of the development and deployment of machine learning models.

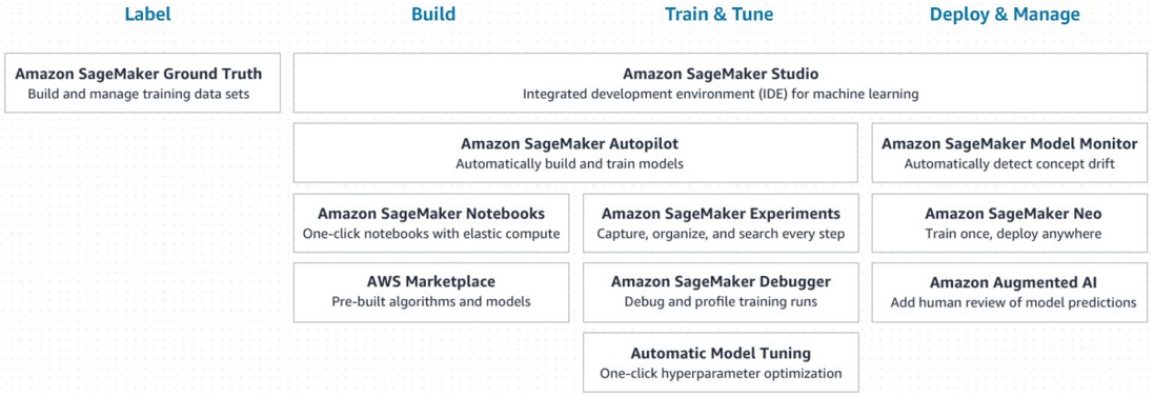



Figure 4-4: Overview of SageMaker capabilities. Source: aws.amazon.com/sagemaker

SageMaker provides three major benefits over traditional development of machine learning solutions:

1. Optimized computing resources. SageMaker allows us to leverage separate EC2 instances for exploration, model training and inference. We can therefore use a powerful but expensive GPU-powered instance to train our deep-learning models, while having a cheaper instance running at all times for inference. Users can also run collaborative Jupyter Notebooks to visualize output or investigate

results. Case studies by Amazon [17] have shown that flexible training through spot instances can reduce training costs by up to 90%.

2. Easy deployment of models to the cloud. Through SageMaker we can deploy our models to a multi-model endpoint with a built-in API. This way, any user can query our models through a few simple commands and integrate our models in their analysis.

3. Hyperparameter tuning capabilities. SageMaker provides a user-friendly interface to tune our models and compare multiple versions. This allows us to visualize the importance of multiple features and determine the optimal version of the code we want to deploy. Furthermore, all of the run results are stored in the AWS console, allowing us to keep track of all past attempts, hence increasing visibility across stakeholders as to our models and facilitating handover.

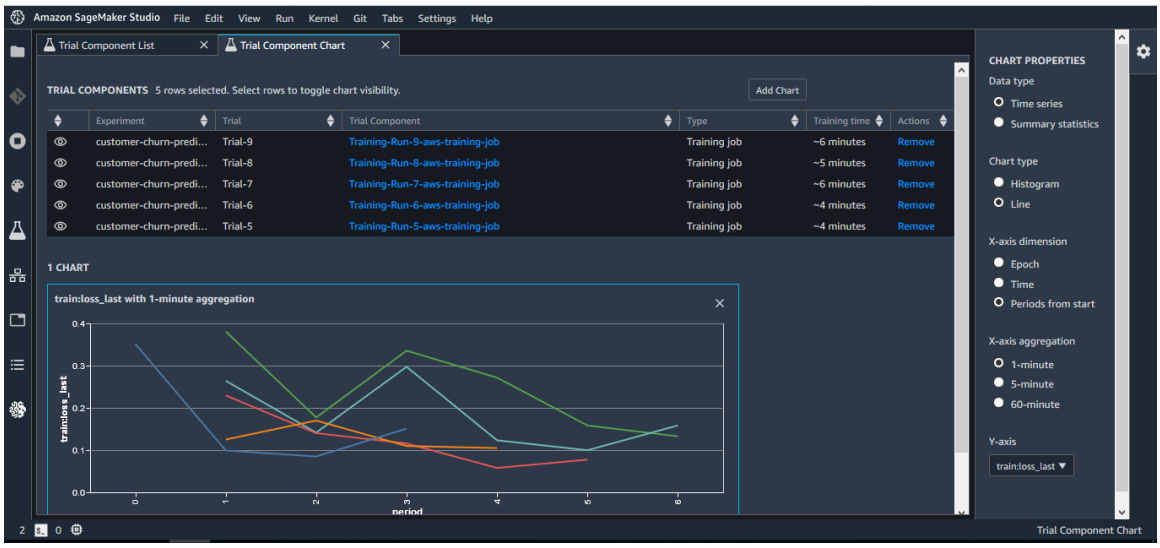



Figure 4-5: SageMaker training experiments. Source: aws.amazon.com/sagemaker

These benefits come at a cost though. Indeed, while SageMaker allows users to save time, instances deployed in SageMaker come in at a premium. As of February 2021, a US-East on-demand t3.medium EC2 instance costs $0.0416 per hour as opposed to $0.0582 within SageMaker, a 40% cost difference. Furthermore, the opacity of certain solutions or algorithms, such as the image classification algorithm or

Bayesian optimization, can be a challenge for users wanting a transparent view on the techniques used.

### 4.3.2 Convolutional neural networks

As of 2021, most state-of-the-art image classification algorithms rely on convolutional neural networks (CNNs). Neural networks, sometimes referred to as artificial neural networks (ANNs), are a family of deep learning models inspired by the humain brain. In their simplest form, they consist of an input layer, one or multiple hidden layers and an output layer, as illustrated in figure 4-6. Layers consist of hidden units, or neurons, connected to each hidden hidden unit of the following layer through an associated weight. As such, each hidden unit can be visualized as having its own linear regressions and outputs toward units of the next layer.

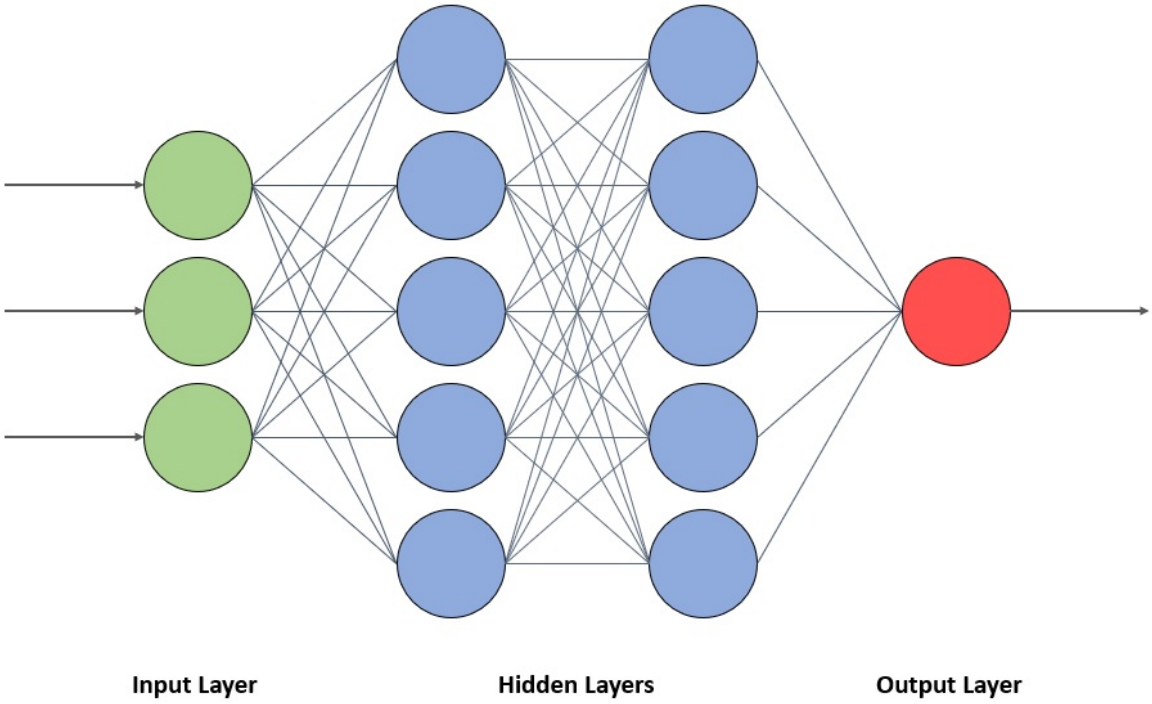



Figure 4-6: Basic artificial Neural network architecture. Models may have an arbitrary number of units or hidden layers.

CNNs are a class of neural networks specialized in computer vision applications. They can handle grid-like input such as images in a way to preserve the locality of each input. CNNs achieve this through the use of convolution and pooling layers.

Convolution layers use a kernel, a matrix of weights, sliding across the input and multiplying units in the input by the kernel's weights to obtain a feature map. Fixing and sliding this kernel across the entire input allows to detect features across multiple spatial regions of the image and effectively reduces the number of parameters required. Multiple layers of convolutions can be stacked and pooling layers are leveraged to downsample the intermediary output even further. At the end of the network, a fully connected layer then uses the output of the previous layers to compute a prediction.

As part of the this tool, we leverage CNNs for each of our models. Three main reasons guide our use of CNNs over other algorithms:

1. **Accuracy**. First and foremost, CNNs are widely used in image recognition because they provide an accurate classification typically unmatched in other algorithms. In the context of this project, accuracy is critical to provide the right insight to site operators.

2. **Robustness and consistency**. While CNNs do not fully solve the problem of robustness, convolution layers allow features to be detected across the image. With proper training practices, this means this type of algorithm will be able to detect packages no matter where they are in the image and be robust to small perturbations in the image. This scalability is also important given no two images of scanners will be the same and so our algorithms need to be flexible enough to recognize packages in new situations.

3. **Support**. AWS SageMaker's built-in image classification solutions all rely on CNNs. By leveraging pre-built and pre-trained algorithms, we can benefit from long-term support and an easy transition. Pre-trained algorithms also save the developers time and effort which they can spend on other tasks.

### 4.3.3 ResNet

Within AWS SageMaker, we leverage the Image Classification algorithm [18], which is a pretrained version of the ResNet algorithm on the ImageNet dataset [19]. We

opt for ResNet as it is currently the only image classification algorithm officially supported by AWS SageMaker and the only one for which a container template can be leveraged, although versions with 18, 34, 50, 101, 152 and 200 layers are available. ResNet models were introduced in 2015 by He et al (Microsoft research) when they outperformed the standard VGG Nets in the ILSVRC 2015 classification task. The authors principally attributed the increase in accuracy to the depth of representations. Previous attempts at increasing the depth of neural networks faced the problem of vanishing or exploding gradients [20]. While progress in regularization and learning methods addressed this problem [21], deep learning models were still crippled by decreasing experimental accuracy at higher depth [22].

In order to tackle this decreasing accuracy, He et al introduce the concept of *deep residual networks*. These networks rely on shortcut connections recasting the original layer mapping $\mathcal{F}(x)$ to $\mathcal{F}(x) + x$ and do not otherwise introduce additional computation complexity or parameters.

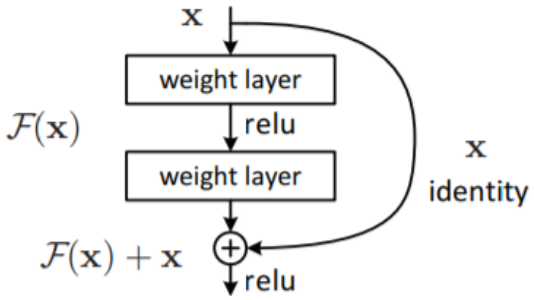



Figure 4-7: Shortcut connections in residual learning (He et al). These shortcuts allow to tackle the vanishing gradient problem.

While AWS SageMaker offers pre-trained versions of ResNet of multiple different depths, we focus our efforts on the 50 layers version. Inference with 102 layers show to be twice as long and only demonstrates minor accuracy improvement in our case. Meanwhile, versions with less than 50 layers did not provide the same level of accuracy in our early tests. Using the version with 50 layers allow us to achieve our accuracy goals while leading to a processing and inference time under 5 minutes for 1000 images on a simple CPU instance, which we deem reasonable for the initial practical purposes of our tool.

### 4.3.4 Transfer learning

SageMaker offers users the option of using the ResNet model structure and retrain the entire model. Alternatively, users can also leverage model weights obtained by training on the ImageNet dataset and instead only adjust weights in the final layers for their specific use through a process called transfer learning. According to Goodfellow et al [23], "transfer learning and domain adaptation refer to the situation where what has been learned in one setting (e.g., distribution $P_1$) is exploited to improve generalization in another setting (say, distribution $P_2$)". Multiple papers show the power of using pretrained models, such as Zeiler et al [24] whose pretrained ImageNet model beat leading results on the Caltech-101 and Caltech-256 datasets.

By using transfer learning, we freeze the weights of every layer except for the last fully connected layer and the final softmax classification. The weights of these layers have a random initialization and are the only ones tuned in the training phase.

Figure 4-8 illustrates the final model architecture, including both the frozen ResNet50 layers and the layers retrained during the transfer learning process. The first layer of ResNet50 is a convolution, followed by a pooling layer. Then, 49 different convolution layers are applied until the model ends with a final global average pooling layer, generating a downsampled array of 2048 values. A fully connected layer further combines these features into 1000 values, which are used by a softmax layer to produce a final prediction. As part of the transfer learning process, we freeze all weights except those creating the fully connected layer and the final prediction.

Furthermore, every model we create follows the exact same architecture and leverages the same pre-trained weights. The only difference in the final models is the new weights for the last two layers as well as the output classes.

While one could expect some of the overall extracted features to be similar between our application and ImageNet, our dataset does differ from the ImageNet dataset in many ways. First, scanner images are in greyscale and only consist of packages and carriers, which are not covered by ImageNet. Furthermore, the format of our images can vary and, as described in chapter 3, images need to be resized to fit the ImageNet

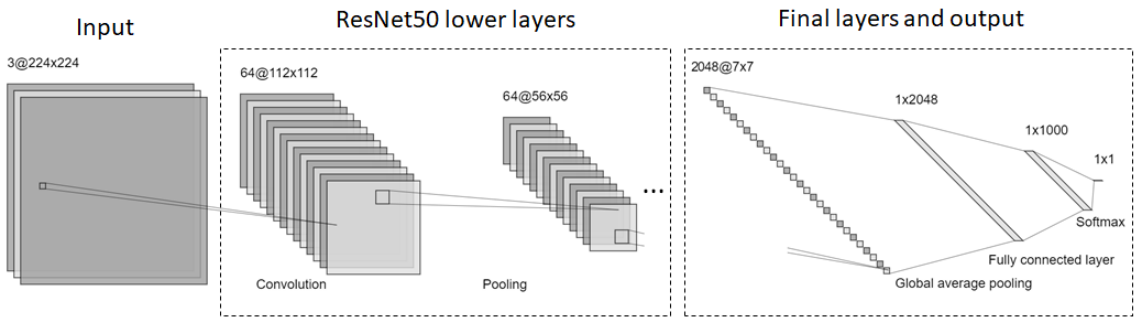



Figure 4-8: Overview of our ResNet50 model, including the lower layers using ImageNet weights and the final layers containing the fully connected and softmax layers, retrained as part of the transfer learning process

format. Still, transfer learning can provide valuable improvements in a variety of settings, in addition to reducing the model training phase, and we later show in table 4.3 the improvement associated to transfer learning over a full model retrain.

## 4.4 Hyperparameter tuning

### 4.4.1 Tunable parameters

Whether we use a pre-trained model or tune all weights, several hyperparameters can affect the speed and accuracy of our training phase. Default values for these parameters can sometimes suffice, but we can also improve our results by optimizing them for our task. More specifically, we have the freedom to adjust five main different parameters in our SageMaker image classification model:

1. The pretrained indicator. Turning this boolean flag on will leverage transfer learning while turning it off will retrain all weights.

2. The learning rate. A smaller value may prevent the model from reaching an accurate solution within our set number of epochs while a larger value may cause a suboptimal convergence.

3. The weight decay. This decay takes the form of an L2 regularization slowly leading the weights toward zero. The regularization may prevent overfit but an

incorrect value can cause a suboptimal convergence.

4. The mini-batch size. This value represent the number of training samples to be used before the weights are updated again. Smaller batches reduce the memory requirements and may allow faster convergence but will increase the overall run time and can introduce noise.

Note that we decided against tuning the momentum in order to reduce the complexity of the hyperparameter tuning phase. Furthermore, Li et al [25] also show that the default value of 0.9 has been found to perform particularly well when the target domain is very different from the transfer learning source domain. Since our pictures of packages are quite different from the usual ImageNet picture, we opt to follow their recommendation and use a momentum of 0.9.

### 4.4.2 Bayesian optimization

Hyperparameter tuning usually takes one of three forms. The most common options are to perform an exhaustive grid search across the entire set of parameters, use a random search or leverage a sequential approach by tuning each parameter one after the other. As one may expect, an exhaustive grid search usually leads to unrealistic computing time, especially for deep learning algorithms. A random search can provide good results, but there is little guarantee as to its convergence and many iterations can be required. Finally, tuning each parameter sequentially can quickly lead to a solution, but may overlook certain parameter interactions and introduces some subjectivity in deciding which parameter to tune first.

Instead, in this thesis we use Bayesian optimization to tune the hyperparameters, a technique that has gained great popularity over the last few years [26, 27].

Although the implementation of Bayesian optimization within AWS SageMaker is not revealed, we know there are multiple ways to implement these techniques. Bayesian optimization relies on probabilistic models predicting posterior means and variances for each set of hyperparameters. Gaussian Processes are commonly used for this task and can help address the tradeoff between choosing the new set of

hyperparameters from exploration or from exploitation. [28].

While this technique can lead to a local minima, its fast compute time and increased complexity compared to a random search can make it a powerful tool.

A commonly used implementation of Bayesian optimization, which we hypothesize behaves similarly to the AWS SageMaker implementation, is the bayesian-optimization package in Python by Fernando Nogueira [29]. In this package, Bayesian optimization is implemented through a gaussian process, constructing a posterior distribution of functions. For each observation consisting of a set of hyperparameters, the algorithm determines which regions are worth exploring or not, as the authors illustrate in figure 4-9. Over time, the posterior distribution improves, leading to a better set of hyperparameters.

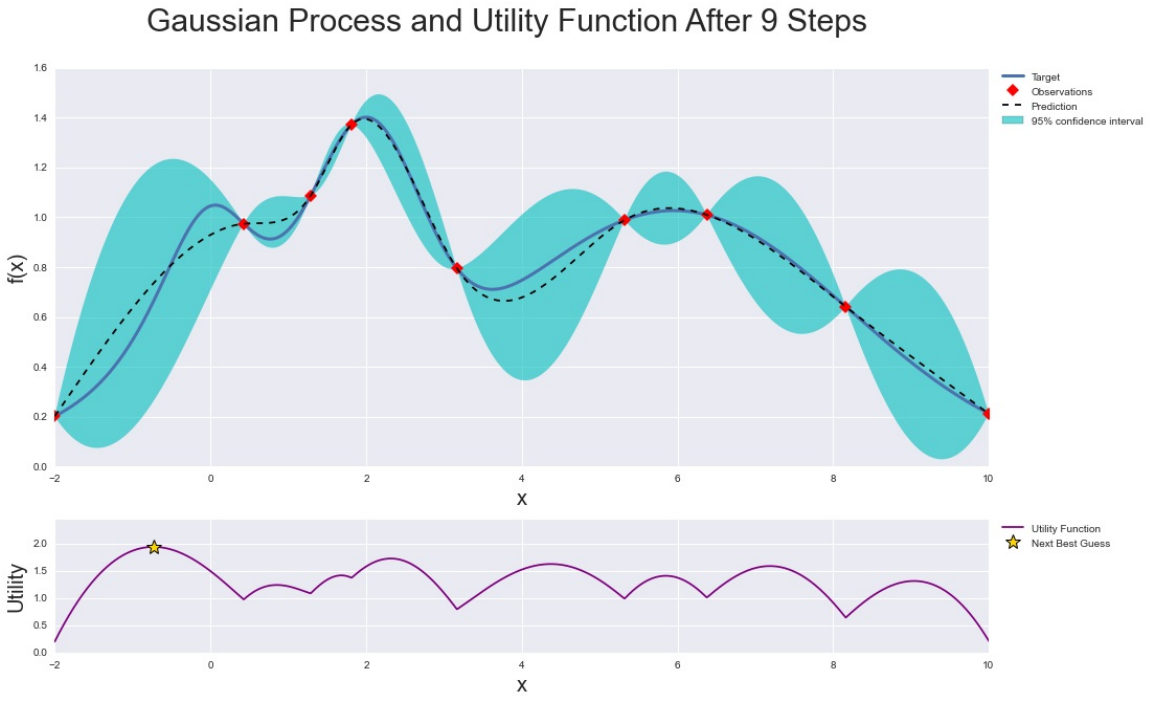



Figure 4-9: The Bayesian-Optimization package uses a gaussian process to determine which sets of hyperparameters the algorithm should explore.

We can also visualize the impact of using Bayesian optimization for the "at least one package" model. Table 4.2 shows the three hyperparameters we are optimizing for this model in the training phase. A full grid search would likely require four values of the mini-batch size and at least six of each of the learning rate and weight decay,

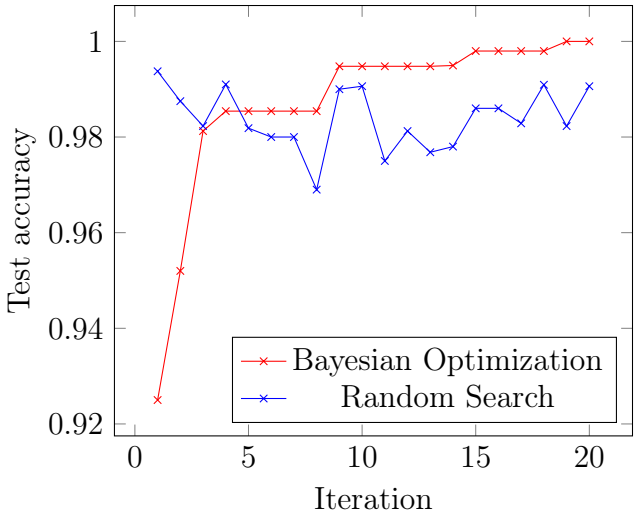



Figure 4-10: The test accuracy for the "at least one package" model improves sharply through Bayesian optimization and reaches 100% after 19 iterations, outperforming a random search.

Table 4.2: Hyperparameters for the at least one package model optimized using Bayesian optimization

| Parameter | Min value | Max value | Final Value |
|---|---|---|---|
| Mini-batch size | 8 | 64 | 8 |
| Learning Rate | 0.0001 | 0.1 | 0.00061 |
| Weight decay | 0.00001 | 0.001 | 0.00056 |

leading to a minimum of 144 full training jobs. Considering the run time for each iteration is between five and fifteen minutes on a GPU instance, this option would significantly slow down the model development process. Similarly, a random search does not achieve the same accuracy as Bayesian optimization across twenty iterations, as shown in figure 4-10.

Perrone et al [12] further analyze the efficiency of Bayesian Search in AWS SageMaker, comparing the performance of random search and Bayesian optimization. They show that Bayesian optimization not only achieves faster convergence than a random search but also a higher AUC.

### 4.4.3 Model evaluation and metrics

The end goal of the tool is to achieve the lowest aggregated error at a site and day level. For example, if the true number of images containing at least one package for a site is 70% on a given day, we want our aggregated forecast to be as close to this value of 70% as possible. End users are not concerned by image-level classification and as such will not be impacted if incorrect classifications cancel each other at the aggregated level. Our initial discussions with end-users conclude that an absolute error of 5% on the aggregate values would be deemed acceptable, as all major trends would be preserved. So, in our example, an aggregated forecast between 65% and 75% is considered sufficiently accurate.

While our goal is to achieve low aggregated error, we cannot use this as a metric for individual images in our modeling phase and only accuracy is currently supported as a metric for image classification in AWS Sagemaker. As a result, we use accuracy throughout our modeling and reporting process and verify that it indeed leads to a low aggregated error as well as seen in chapter 5.

### 4.4.4 Model selection

We train and tune all four models using both a full retrain and a transfer learning process and display the results in table 4.3. Transfer learning produce an accuracy improvement for each model and is thus selected over a full retrain. We further train each model using four processing techniques to determine which leads to the most accurate results.

Interestingly, the addition of canny edge helps our multiple package model, indicating that this technique may help tell apart packages in images. Similarly, highlighting bellows provide a 4.8% absolute improvement in accuracy for the package position model which relied on determining if a package ends up on the bellows or on the carrier.

Since our final pipeline is able to handle different preprocessing for each model, we can select the most accurate technique for each model to be used both during

Table 4.3: Model validation accuracy by processing technique

| | Full retrain | Transfer learning | | | |
|---|---|---|---|---|---|
| | Original | Original | Highlight bellows | Canny edge | Both |
| Empty Carrier | 95.0% | 100% | - | - | - |
| Multiple Packages | 95.5% | 96.4% | 96.9% | 98.0% | 95.4% |
| Package Type | 91.0% | 98.1% | 97.4% | 97.5% | - |
| Package Position | 87.3% | 89.5% | 94.3% | 92.0% | 91.3% |

training and inference. So, our final empty carrier and package type model do not use additional processing while our multiple packages and package position models use canny edge and bellow highlighting respectively.

## 4.5 Multi-model inference endpoint

Once all models are trained, we deploy a multi-model inference endpoint using SageMaker. The endpoint is effectively a separate AWS EC2 instance hosting all of the trained models and allowing easy inference through an API.

We opt for a multi-model endpoint which can store multiple models on the same instance, hence significantly reducing the overhead costs. The resources are shared across all models and users can simply specify in the API which of the models they wish to query. Models are stored as compressed files in an S3 bucket, with all past versions being preserved. This specific type of endpoint is particularly cost-effective in our case, given the models will be sparsely queried and may have significant idle time. In fact, given we expect the tool to be used only a few times per day at most at the start, we opt for an inexpensive CPU instance for inference, which could be upgraded to one or multiple GPU instances in the future if needed.

The implementation of this type of endpoint involve the creation of a Docker [30] image for our custom model and our pre-processing requirements. A Docker image contains all of the required software and dependencies to allow our code to execute in a given environment, regardless of the infrastructure. Docker images are easy and fast to deploy and become containers once running.

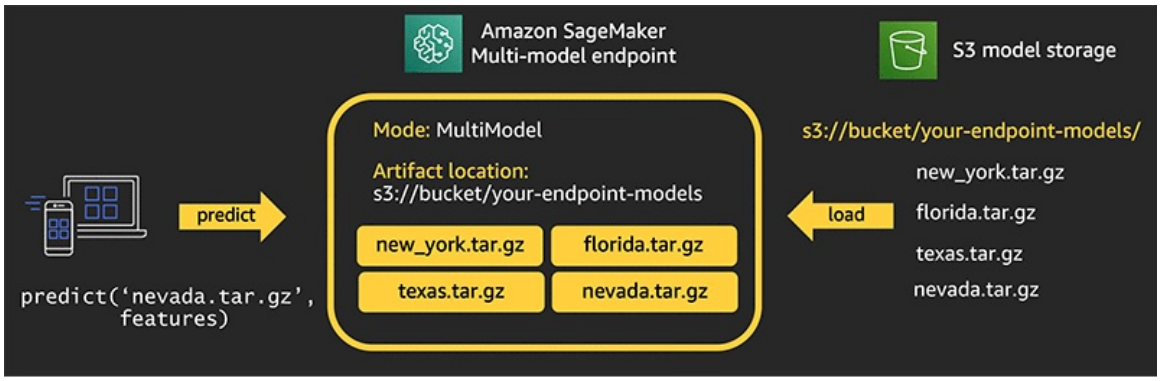



Figure 4-11: Example of a SageMaker multi-model endpoint. Source: aws.amazon.com/sagemaker

# Chapter 5

# Results and insights

In this section, we share details about our user interface, including how it is deployed and how we report top-1 accuracy as a proxy for confidence. We also analyze the final testing accuracy for each model and summarize some of the accuracy challenges remaining. We then guide the reader through a case study from a large site with recurring issues and the results and implications of our tool deployment there. Finally, we discuss some of the remaining limitations of our solution, such as the lack of real-time output and the need for a manual launch by site operators.

## 5.1 User interface

### 5.1.1 Static S3 website

Users need a simple way to understand use cases, upload their images and launch the tool. To this effect, we create and deploy a simple online no-read deep dive tool. The landing page leverages the Bootstrap open-source CSS framework and a simple JavaScript back-end verifies the input format and uploads images to the data S3 bucket. Users also have access to a detailed manual of how to obtain images and how to interpret any output.

Furthermore, we can host this static website directly on an Amazon S3 bucket. Static websites can contain HTML code and client-side JavaScript, and launching

such a page on S3 only requires creating an index page and activating the hosting feature. Any employee with the right internal permissions can then access the tool. A sanitized overview of the landing page can be seen in figure 5-1.

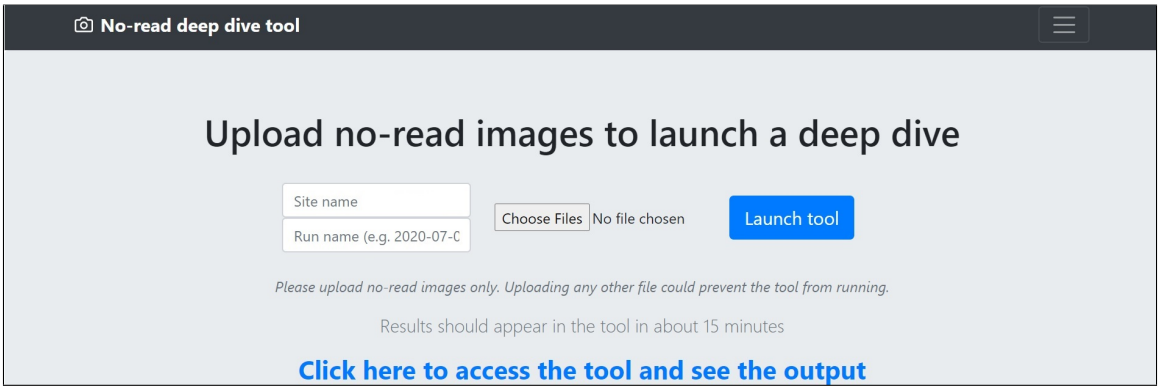



Figure 5-1: Static no-read S3 website

### 5.1.2 QuickSight overview

The final output is displayed on a custom Amazon QuickSight page. QuickSight is a cloud-powered business intelligence solution allowing users to create interactive dashboards and reporting [31]. Quicksight offers easy integration to most relational data stores such as MySQL, Amazon S3 and Amazon Redeshift. As an Amazon product, QuickSight is available for all Amazon employees and hence allows us to easily share insights across the organization.

The tool's dashboards include no-read rates over time as well as the models output. Users can filter the data for any site and day and can also download the data in Excel. They can also drill down into specific issues, for instances by selecting only certain types of packages or issues and analyzing their trends over time.

A redacted example of this dashboard can be seen in figure 5-2

### 5.1.3 Top-1 probability reporting

Site users often ask how can they know if the models are confident in their output. This motivated us to add a proxy for confidence reporting on the final site under the

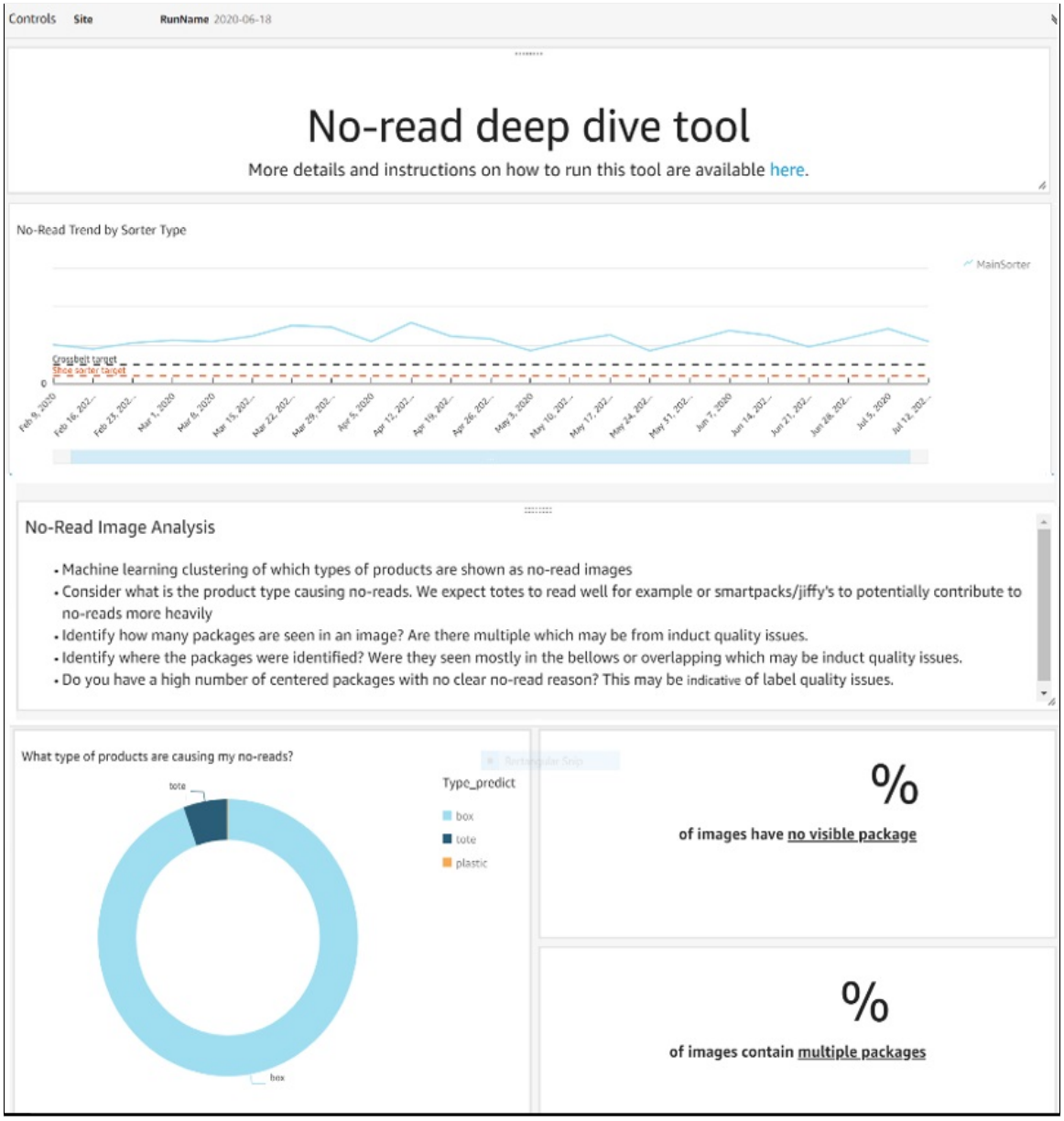



Figure 5-2: QuickSight Output example. Users can scroll down for more insights can also click to drill down into specific issues.

form of the average top-1 probability for a prediction set. For a single image, the top-1 probability is simply defined as the output probability for the image's top class.

This reporting process follows a few steps. First, for each model and image we define the top-1 prediction probability $p_{m,i}$, the probability for the chosen class:

$$p_{m,i} = \max_{k \in K_m} p_{m,i,k} \tag{5.1}$$

Where $K_m$ represents the set of output classes of model $m$ and $p_{m,i,k}$ the output probability for model $m$, image $i$ and class $k$. We then define the top-1 probability $Top\text{-}1_m$ of model $m$ as:

$$Top\text{-}1_m = \frac{1}{n}\sum_{i=1}^{n} p_{m,i} \tag{5.2}$$

Where $n$ represents the number of images in the inference set. In other words, $Top\text{-}1_m$ is the average across the inference set of the probability for the chosen class for each image. A high probability for a given model and inference data set indicates that on average the probability of the highest class is high itself. As such, we can use this metric as an indicator of the overall trained model confidence and compute it for any inference set.

However, we earlier described this technique as proxy for confidence. Indeed, a model predicting labels with high top-1 probability does not necessarily mean that the output is correct. Models could be both highly confident and wrong, for instance if they face a type of package they have never seen before. Under most circumstances though, having a low top-1 probability should raise a flag to the user as there may be an issue with the input or an unexpected format.

Our proposed solution to report top-1 probability while simplifying the reporting is to categorize the model output in high, medium or low confidence buckets. The thresholds for high and medium top-1 probability are respectively set at 95% and 85%. These thresholds are determined across a series of site tests and are deliberately chosen to be conservative.

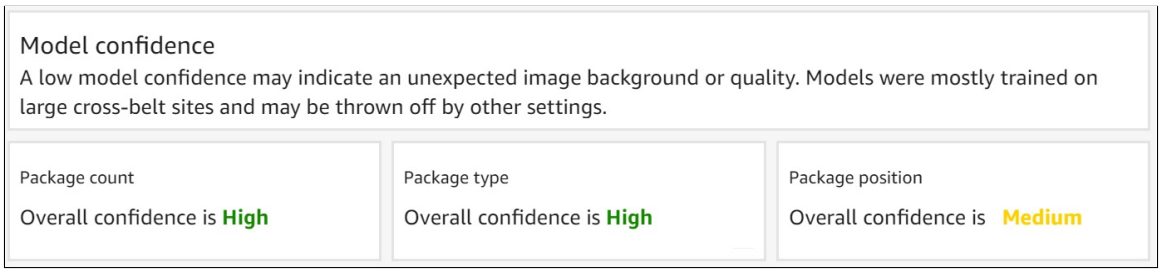



Figure 5-3: The model top-1 probability is dynamically displayed in QuickSight for each input. Users can investigate images producing a low confidence and verify that the format and characteristics are as expected.

Table 5.1: Final model validation and test accuracy and aggregated worst class error on the test set

| | Final processing | Val. accuracy | Test accuracy | Agg. test error |
|---|---|---|---|---|
| Empty Carrier | No processing | 100% | 98.7% | 0.6% |
| Multiple Packages | Canny edge | 98.0% | 96.2% | 1.6% |
| Package Type | No processing | 98.1% | 88.3% | 1.8% |
| Package Position | Highlight bellows | 94.3% | 85.3% | 1.8% |

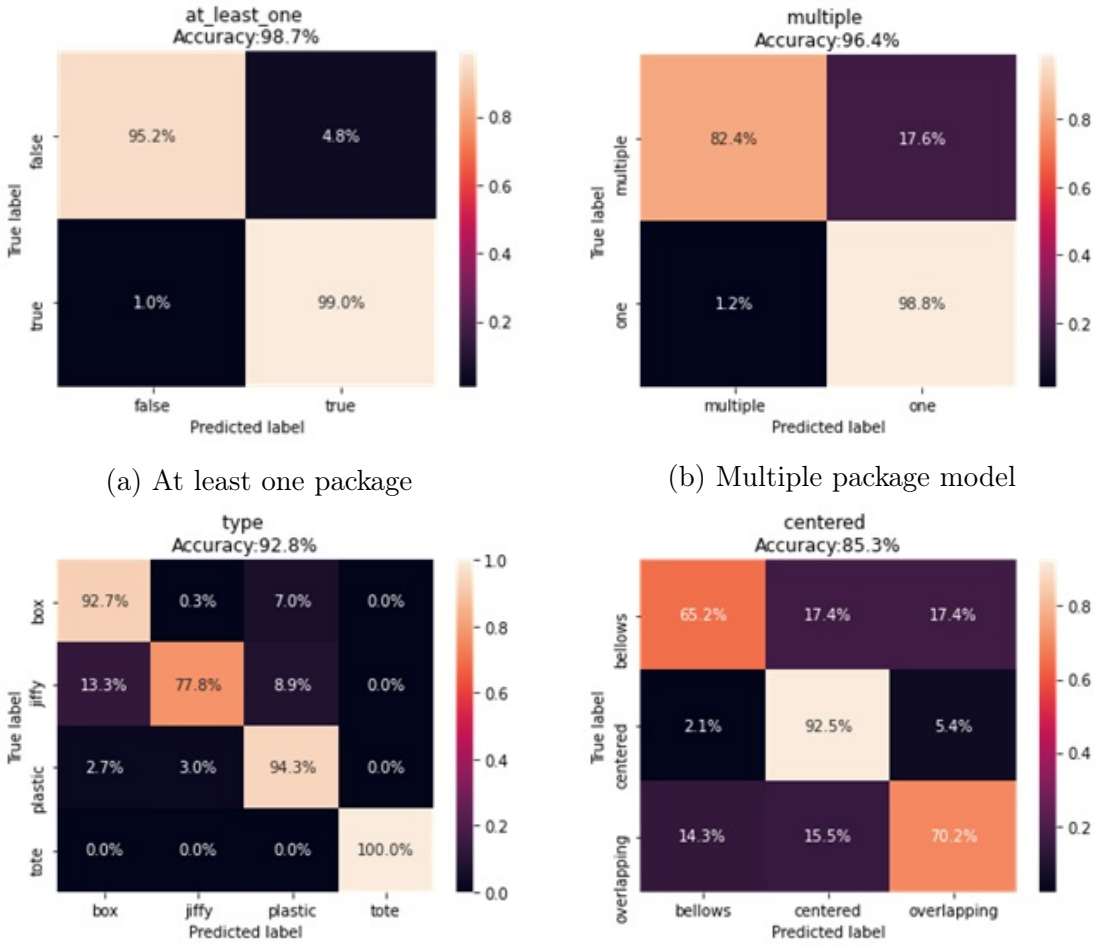
## 5.2 Final testing accuracy

Table 5.1 shows the final processing technique for each model along with the accuracy results. First, we notice accuracy in the test set can be up to 10% lower than in the validation set. Since the validation set is constructed from the same sites as the train set, as opposed to the test set, we do expect a drop in accuracy. However, once aggregated, all four models perform well below our initial 5% target error threshold, with the highest error being at 1.8% for the package type and position models. Furthermore, since these two models contain respectively four and three output classes, as opposed to two classes for the first two models, accuracy is expected to be lower.

While our initial model, trained on a single site, struggled to accurately assign labels for a second site, our final model is more consistent. Using ten sites for training leads us to be able to meet our target error within every site in scope, whether they are used in the training phase or not. Therefore, our final model generalizes well to sites not used in the training data and can scale up across the entire network.

The confusion matrices in figure 5-4 show our predictions against the actual values. For instance, the top row of the at least one model tells us that among false predictions (images with no packages) we correctly identify them as having no package 95.2% of the time, but think they had a package 4.8% of the time. This helps us understand which classes can be confused with each other as well as checking that our overall accuracy is not due to class imbalance.

These confusion matrices help us identify potential issues or classes that can still confuse the models. We can analyze the results for each of the four confusion matrices in figure 5-4 before coming up with overall conclusions.




(a) At least one package

(b) Multiple package model

(c) Package type

(d) Package position

Figure 5-4: Confusion matrices across each of the four models. The rates are computed on the test set.

### 5.2.1 Final accuracy - At least one package

This first model aiming at telling apart images containing no packages and images containing at least one presents the highest accuracy across models. This model has a precision of 95.4%, meaning that 95.4% of images predicted as having a package do have one. Similarly, it has a recall of 99%, indicating that the model finds 99% of images containing at least one package.

While the model meets all of our accuracy goals, we are still interested in determining when it fails to prevent future issues. For instance, by looking at the images, we can see that a common cause for misclassification is the presence of dirty carriers. Figure 5-5 shows an example of such a dirty carrier. Given a carrier can be dirty or soiled in many different ways, it can be challenging for an algorithm to perfectly learn to recognize dirty carriers. However, we include all of the examples we could find in our final training data set.

Figure 5-5: Carriers can be dirty, causing models to think there is a package on each image.

### 5.2.2 Final accuracy - Multiple packages

With a precision of 98.6% and a recall of 82.4%, our second model predicts which images contain multiple packages. However, 17.6% of images with multiple packages are predicted as having only one, while only 1.2% of images with one package are predicted as having multiple. This difference can be explained by two things.

First, there is a slight class imbalance for this model with 85.2% of images having a single package. Since our goal is to maximize accuracy, smaller classes will be penal-

ized. However, this is a tradeoff users are aware of and willing to accept. Ultimately, given the high no-read rates observed, classes with a lower frequency can only explain a small portion of no-reads, and thus a larger relative error is acceptable for those.

Second, there are instances where unexpected image or carrier conditions can cause the algorithms to believe there are multiple packages. Figure 5-6 show one those situations, where a package was left next to the conveyor belt and accidentally caused the algorithm to think there were multiple packages on each image.


Figure 5-6: A package on the side on the conveyor belt (bounded in red) can throw off the algorithm.

### 5.2.3 Final accuracy - Package type

As we now move from two output classes to four, the package type model still displays good accuracy but more challenges than the previous two models. Within the test set, boxes represent 49.7% of images and plastic packages 38.3%, while jiffy and totes only represent 6.5% and 5.5% of packages respectively. This imbalance explains why boxes and totes tend to have a higher accuracy, whereas 13.3% of jiffy envelopes are predicted as boxes.

By examining cases of images the algorithm thought were plastic packages while they were jiffy envelopes, we can see a significant number of partial packages. Figure 5-7 shows an example of a partial jiffy envelope, which the model often confuses for

a plastic package. This problem partly arises from the lack of color in the images generated by the scanners; otherwise the white and blue pattern of plastic packages can be easily discerned from the beige envelopes.


Figure 5-7: Models can be confuse jiffy envelopes, as in this image, with plastic packages when they are only partially visible.

Interestingly, totes have a perfect prediction rate. This is likely due to them being very different from the other types of packages and typically being centered in packages.

### 5.2.4 Final accuracy - Package position

Our final model is the package position model. This model has the lowest accuracy at 85.3% and is also the model we spent the most time iterating. While a label such as a package type is ultimately clear, a package position is more ambiguous in nature. Different humans can come up with slightly different labels. Furthermore, algorithms and humans alike can be confused by the bellows and overlapping labels. As showed in figure 4-2, packages are considered on the bellows if their apparent center of gravity is on the bellows between sets of two carriers. On the other hand, if they overlap with the thinner metal bars between carriers, we label them as overlapping. The challenge with this classification arises from the fact that images are often missing these metal bars or the bellows. So, the classification can sometimes rely on analyzing the rest of

the image or even using one's judgment.

Part of this ambiguity can be resolved through preprocessing. In table 4.1 we show how highlighting bellows improved the accuracy of the package position on the validation set from 87.3% to 94.3%. Most of this accuracy improvement comes from the algorithm now being able to tell apart the two faulty situations.

As for similar models, there is a slight data imbalance that can impact accuracy for smaller classes. In our test set, 69.2% of images contain packages that are centered, while overlapping and bellows situation represent respectively 24.2% and 6.6% of the cases. This explains why the centered labels are the most accurate, while the error is higher for the two fault classes. Still, the algorithm is able to find over two-thirds of faults while correctly identifying 92.5% of centered packages. Most importantly, once aggregated, the errors tend to cancel out. For this specific model, the aggregated test error is 1.8%. This means that for all of the three ratios mentioned earlier (69.2%, 24.2% and 6.6%), our forecast was 69.6%, 22.4%, 8.0% and the biggest absolute error observed across these was 1.8%. And for our user base, reporting an overlapping rate of 24.2% of 22.4% for a site will result in the same actions taken and the same impact. Furthermore, the error is consistent across each site individually.

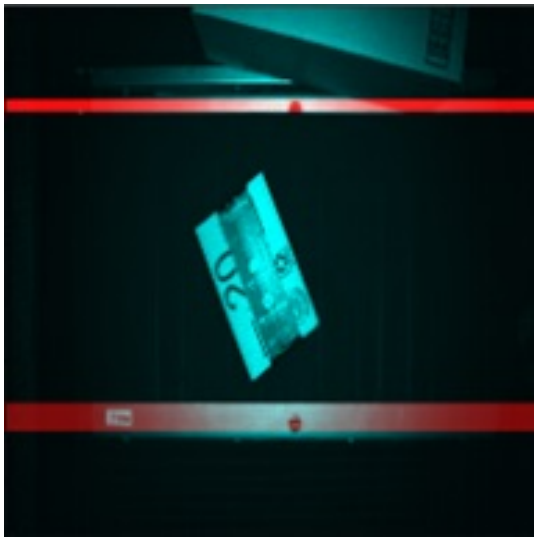
### 5.2.5 Final accuracy - Summary

Overall, all of our models meet our 5% aggregated error goal. However, achieving the best accuracy possible is critical to ensure models will scale well across new sites and require as little maintenance as possible. To this end, we identify and highlight the following remaining challenges and issues:

1. The models get most of the centered packages correctly, but in the case of an overlapping or bellows package, tend to equally predict either of the remaining two classes.

    - This is likely partly due to the subjectivity of these classes. If a package slightly touches the bellows, we will not label is as being on the bellows but the algorithm may think it is. Furthermore, algorithms may confuse

(a) Initial grayscale image



(b) Image with bellows highlighted in dark red and overlapping metal bar in bright red.

Figure 5-8: Pre-processing through red highlighting can help models learn the difference between bellows and overlapping

packages overlapping and on the bellows given images may sometimes only capture a small portion of either extremities, making it difficult for both humans and machines to identify separator and label images.

2. Jiffy packages can be mistaken for plastic packages. Given our grayscale images this is not entirely surprising. One may consider splitting plastic into polybags and smartpaks and see if it affects the result.

Some of these issues may be solved through better modeling or additional training data, although the last few iterations of our models saw little change there. In the absence of improvement, and realizing that these challenges could slightly affect some outputs, we lay down explanations and warnings in the user guides about these situations.

## 5.3 Site case study

### 5.3.1 Situation

Soon after deploying the tool, the team from a large IXD with a crossbelt sorter reached out to us to test it. The site had recurring major issues with scanner no-reads and had seen its no-read rate increase up to 10% in the first half of 2020. We extracted over 10,000 images from the site across June and July, uploaded them to the tool and helped investigate the issues. In this section, we share some preliminary findings for this site, while omitting identifiable details and exact numbers for confidentiality purposes.

### 5.3.2 Model results

We use the web interface to load images from two different scanners over two months into our tool. After a run time of about 15 minutes, the site dashboard is populated with all the relevant machine learning outputs. While analyzing the results, we identify two main insights:

1. A significant amount of no-read images contain no package at all. This number is very high compared to other sites and could be due to packages ending up in the netting and other induction issues.

2. There are hundreds of non-centered boxes and totes. We typically tend to see these issues with lighter packages such as smartpaks / jiffys at most sites, but very rarely in this order of magnitude for boxes and totes. The system is usually designed to work with a significantly higher accuracy across heavy packages.

These two points are significantly different from comparable sites and also far from Amazon's targets. While it may have been difficult to determine this from manually looking at a few images, the tool allows users to quantify these problems and benchmark them in a matter of minutes, something they would not have been able to do before.

### 5.3.3 Business and operational implications

We shared our preliminary findings with site operations. By going through the images containing no package, we realized this site had significant issues with dirty carriers causing false triggers. A team cleaned up all carriers during a maintenance pause and results from the following weeks showed a decrease in images containing no packages. Engineers also tracked down boxes that were not centered and realized that one of the induct stations had a broken belt causing the bad placement of boxes.

The tool and results were shared with all of the site operations, including the building general's manager. Following the positive feedback, operations managers agreed to use the tool in the future if there are further issues with no-read rates.

## 5.4 Limitations

### 5.4.1 Model limitations

Due to a mix of data and time limitations, some potential causes for no-reads are not yet tackled. We cover the causes which represented the best combination of effort required and potential benefits but were not able to include the following models, in order of importance:

1. Is there a full label on the package?

2. Is there any damage to the package or the label?

3. Can a partial label be observed?

4. Are carriers dirty or damaged?

The first three models listed above would likely require additional data sources, such as side no-read cameras. Indeed, labels will be scanned if they are on the side of the package so any model will need to include these angles as well, which would increase the resulting complexity. Still, we leave detailed instructions to the team on

how these models could be developed as well as requests for additional engineering resources.

Furthermore, the tool is optimized for crossbelt sites, and accuracy may not be as high for other types of sites (e.g. sort centers). Finally, the only data currently used are images from the top scanners, and all images need to be manually inputed. No other angles or logs are integrated in the models.

### 5.4.2 Usage limitations

One of the major limitations is that while the model is fully working and deployed, it is currently only meant to be an ad-hoc tool. Sites experiencing issues with no-reads can launch the tool and obtain insights. However, the tool still requires a manual launch and does not run in real time. As part of our final handover, we propose a way to remove the manual aspect of the tool by fully integrating it within all sites daily operations. As such, the tool could be connected in real-time to any site and kick-off whenever no-read rates go above a certain threshold.

This proposed upgrade will require a team of data engineers to create the required pipelines with all sites as well as an upgrade of the inference instances to accommodate the additional traffic.

# Chapter 6

# Conclusion and future work

While the tool is currently deployed across North America, there are multiple ways it could be further improved. In this final chapter, we cover the immediate next steps to support and leverage this tool but also lay out a long-term vision to take this tool to the next level and magnify its impact across Amazon's fulfillment network.

## 6.1 Next steps

This tool finally allows site operators and engineers to better understand potential root causes of no-reads without having to manually investigate images or involve vendors. However, for the impact to be fully realized, more models and a better integration within everyday operations will be necessary. With this in mind, we built the entire solution in such a way that retraining models, adding new models or scaling up the number of images processed can be done without major modifications to the current pipeline. As a result, we have identified and are recommending the following next steps, in order of importance:

1. Incorporate this tool as part of the regular maintenance and operations investigations into no-read rates

    - Several demonstrations have already been done with various levels of potential users but operations engineering will need to raise awareness and

actively push this tool whenever sites have concerns about no-reads.

2. Mandate deep-dives using this tool as part of the new site qualification process

    - While the first recommendation is reactive, we also need to be proactive. Integrating this tool in the qualification process will allow us to detect any issues early and act quickly. This will also help standardize the scanner and no-read camera setup which currently greatly varies by site.

3. Expand the tool coverage by adding additional models for damaged or absent labels.

    - The current models cover the main causes of no-reads and scanner issues but other types of models could also help. Among those, models on label damage or absence could push the deep-dive further.
    - Adding models requires a significant amount of time and knowledge about machine learning, but instructions and guides have been developed to this effect and shared with the new owners.

4. Pilot the upgrade of current scanners to 3D cameras allowing depth perception

    - Some sites have shown interest in testing new hardware and software allowing us to use 3D cameras and perform live error classification. While these are still to be tested, we should investigate their efficiency and measure their ability to reduce no-reads.

Additionally, two common causes of no-reads were cited across the sites we have visited. Since the scope of this project was limited to the tool development, we did not pilot specific no-reads solputions and would encourage investigating the following:

1. Installing bellow guards at the end of each bellows has shown promising results across sites to reduce no-reads and prevent packages from ending up in the bellows.

2. Mandating a minimum weight of 100g for smartpaks, as it is the case in Europe, could greatly help reduce the number of these packages having induction issues or ending up on the bellows.

    - While there would be a cost from shifting away from smartpaks, the tool could help process engineering quantify the impact.

Finally, this tool provides the framework that will allow various groups to analyze the situation and weigh the impact of no-reads against the costs of various solutions. Among the most important analyses we recommend performing:

1. What solutions can we implement and are they worth it? E.g. new scanner, new belts, bellow guards, etc. By how much would they reduce no-read and how much would they cost?

2. What types of package hurt me the most (e.g. smartpaks)? What is the current cost of no-reads for them and would it be worth shifting some of the volume away to other package types?

## 6.2 Long-term vision

We have demonstrated this tool to several groups within Amazon and worked on determining potential long-term uses or additions to this tool. We have identified multiple features which would greatly increase its value, although at the expense of heavy development work :

1. Determine a way to link results to specific induct stations or equipment

2. Automatically kick off the tool and send alerts when no-reads reach a certain rate across a site

3. Have an automatic data engineering pipeline allowing us to skip the manual image upload

4. Include several other models leveraging side images

All of these improvements can bring great benefits to users and although were not in the scope of our work, we would highly recommend investigating their individual required investment. Furthermore, operations engineering is assessing whether any team could take over this tool and lead it towards this desired state.

# Bibliography


[1] Kellie Garnett. The surprising way one Amazon warehouse tour ended.

[2] Dematic. Smart solutions for returns; We optimise your Supply Chain., November 2016.

[3] Bernie Knill. The art and science of sortation. *Material Handling Engineering*, 54(9):61–64, September 1999.

[4] Lorie King Rogers. Sortation: Reliability meets scalability. *Modern Materials Handling*, 67(5):36–39, May 2012.

[5] Clyde E. Witt. Tunnel scanners: keeping pace with distribution. *Material Handling Management*, 55(11):4–13, October 2000.

[6] James Carroll. Cognex acquires deep learning machine vision software company ViDi Systems, April 2017.

[7] Nils Boysen, Dirk Briskorn, Stefan Fedtke, and Marcel Schmickerath. Automated sortation conveyors: A survey from an operational research perspective. *European Journal of Operational Research*, 276(3):796–815, August 2019. Publisher: Elsevier B.V.

[8] Alessandra Caggiano, Jianjing Zhang, Vittorio Alfieri, Fabrizia Caiazzo, Robert Gao, and Roberto Teti. Machine learning-based image processing for on-line defect recognition in additive manufacturing. *CIRP Annals - Manufacturing Technology*, 68(1):451–454, January 2019. Publisher: Elsevier Ltd.

[9] James Thomas Howard Smith. Deep Learning for Automated Visual Inspection of Uncured Rubber. June 2018.

[10] Amazon SageMaker customers.

[11] Asim Khan, Umair Nawaz, Anwaar Ulhaq, and Randall W. Robinson. Real-time plant health assessment via implementing cloud-based scalable transfer learning on AWS DeepLens. *PLOS ONE*, 15(12):e0243243, December 2020. Publisher: Public Library of Science.

[12] Valerio Perrone, Huibin Shen, Aida Zolic, Iaroslav Shcherbatyi, Amr Ahmed, Tanya Bansal, Michele Donini, Fela Winkelmolen, Rodolphe Jenatton, Jean Baptiste Faddoul, Barbara Pogorzelska, Miroslav Miladinovic, Krishnaram Kenthapadi, Matthias Seeger, and Cédric Archambeau. Amazon SageMaker Automatic Model Tuning: Scalable Black-box Optimization. *arXiv:2012.08489 [cs, stat]*, December 2020. arXiv: 2012.08489.

[13] Aharon Azulay and Yair Weiss. Why do deep convolutional networks generalize so poorly to small image transformations? *arXiv:1805.12177 [cs]*, December 2019. arXiv: 1805.12177.

[14] François Chollet and others. *Keras*. 2015.

[15] OpenCV. Open Source Computer Vision Library. 2015.

[16] John Canny. A Computational Approach To Edge Detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679–698, December 1986.

[17] Mark Roy and Urvashi Chowdhary. Save on inference costs by using Amazon SageMaker multi-model endpoints, November 2019.

[18] AWS SageMaker Developer Guide, Image Classification Algorithm, https://docs.aws.amazon.com/sagemaker/latest/dg/image-classification.html, July 2020.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[20] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[21] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

[22] Kaiming He and Jian Sun. Convolutional Neural Networks at Constrained Time Cost. *arXiv:1412.1710 [cs]*, December 2014. arXiv: 1412.1710.

[23] Ian Goodfellow and Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016. \url{http://www.deeplearningbook.org}.

[24] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. *arXiv:1311.2901 [cs]*, November 2013. arXiv: 1311.2901.

[25] Hao Li, Pratik Chaudhari, Hao Yang, Michael Lam, Avinash Ravichandran, Rahul Bhotika, and Stefano Soatto. Rethinking the Hyperparameters for Fine-tuning. 2020.

[26] Vu Nguyen. Bayesian Optimization for Accelerating Hyper-Parameter Tuning. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 302–305, June 2019.

[27] Mercy Prasanna Ranjit, Gopinath Ganapathy, Kalaivani Sridhar, and Vikram Arumugham. Efficient Deep Learning Hyperparameter Tuning Using Cloud Infrastructure: Intelligent Distributed Hyperparameter Tuning with Bayesian Optimization in the Cloud. In *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, pages 520–522, July 2019. ISSN: 2159-6190.

[28] N. Srinivas, A. Krause, S. M. Kakade, and M. W. Seeger. Information-Theoretic Regret Bounds for Gaussian Process Optimization in the Bandit Setting. *IEEE Transactions on Information Theory*, 58(5):3250–3265, May 2012. Conference Name: IEEE Transactions on Information Theory.

[29] Fernando Nogueira. Bayesian Optimization Python Package, 2020.

[30] Merkel, Dirk. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014:2, 2014.

[31] What Is Amazon QuickSight? - Amazon QuickSight.