# Machine Learning for Product Recognition at Ocado
— Report One —

Kiyohito Kunii, Max Baylis, Matthew Wong,
Ong Wai Hong, Pavel Kroupa, Swen Koller,
{kk3317, mgb17, mzw17, who11, pk3014, sk5317}@ic.ac.uk

Supervisor: Dr. Bernhard Kainz
Course: CO530, Imperial College London

27th January, 2018

## 1 Project Requirements

### 1.1 High-level Goals

The goal of the project is to deliver a machine learning system that can classify images of Ocado products in a range of environments. This system will consist of an image processing pipeline that demonstrates key processes and techniques necessary for classifying a subset of the Ocado product database, but will not be a complete solution suitable for deployment either in the Ocado warehouse or as part of another software product.

### 1.2 Initial Suggestions and Feasibility

It was quickly agreed that a deep learning approach would be taken in order to achieve the above requirements, motivated by the successes of deep learning frameworks in the computer vision community [1]. As with any deep learning project, the feasibility study started with an investigation into available data.

The dataset provided by Ocado consists of images that were automatically captured in their warehouse during the course of product shipment. It was suggested to use this dataset to train a Convolutional Neural Network-based image classifier. While this dataset was very large, an initial investigation into the data yielded several problematic observations:

- Most of the images showed the products in a single orientation; due to the images being taken immediately before and after a barcode scan, from a single camera angle.
- Virtually all images were from the same setting (warehouse background, lighting and equipment) resulting in a systematic bias within the dataset.
- A significant proportion of images did not feature the intended products, or included a systematic obstruction (in this case, a warehouse workers arm).

While the dataset is ideal for testing a product image recognition system, it is not suitable for training due to the inherent systematic biases described above. Indeed, generalization was an issue as outlined by the client when specifying the system - when the client tested their system on unseen environments, a 10-30% drop in accuracy was observed. It was therefore concluded that most of the projects effort should be towards mitigating the lack of unbiased training data.

[1] Image array of 4 randomly chosen images of a single product from the Ocado warehouse dataset. This is representative of the systematic pose bias (back of product always showing) and obstruction (workers arm) in the training data.

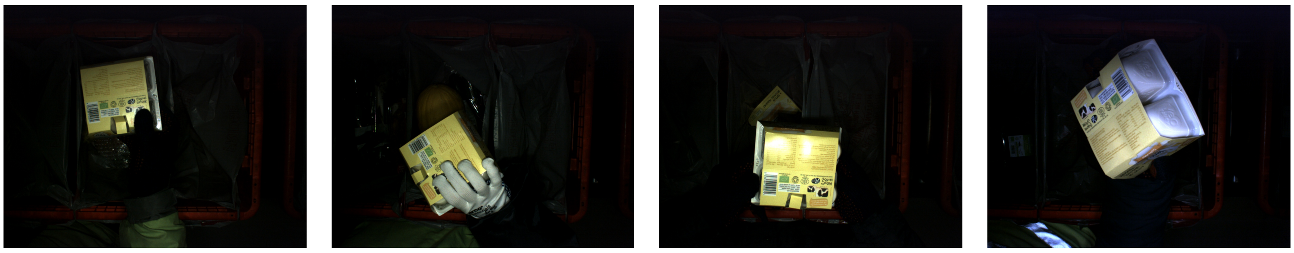

Figure 1: Ocado warehouse sample images [1]

## 2 Essential Requirements (MVP)

### 2.1 Proposed Methods

Our initial proposal is to employ a system that can self-train by generating training data from 3D models of the target products. The strategy for this system is as follows:

- The physical products will be optically scanned, and a 3D reconstruction program will be used to generate a 3D model.
- The model will be used to generate labelled training images.
- Generated images are used to train the CNN.
- The accuracy of the CNN is then evaluated using provided images.

Using 3D models is a long-standing practice in the computer vision community [2, 3, 4] and is thought to be feasible for the following reasons:

- Intra-class variation for a product is limited, and so an accurate 3D representation can be generated from a small number of products.
- Training images for new products can be easily generated without having physically to take a large amount of images.
- The technology for obtaining high-fidelity scans of products is mature and easily accessible.

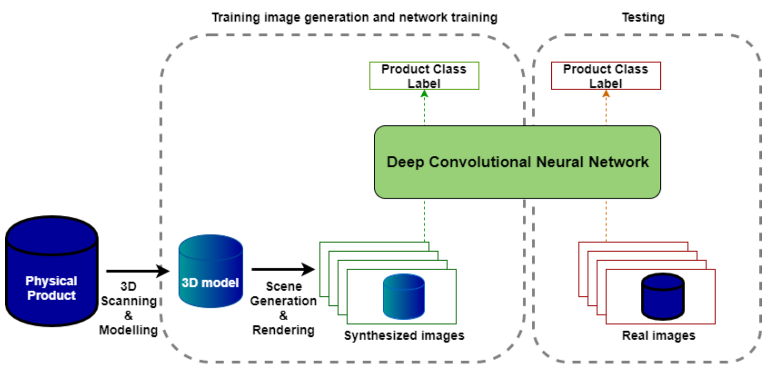

Figure 2: Training pipeline schematic

### 2.2 Data Generation (3D Modelling)

For an initial proof of concept, web-shop images will be augmented as training data. Following this, we intend to create 3D models to create large volumes of training data. Creation of the 3D models will be done in two ways: First, using a 3D scanner such as Microsoft Kinect and associated 3D Scan software. Second, images will also be taken using a conventional camera and used as input to the

Agisoft PhotoScan software as an alternative method of model creation. Consistency of image capture will be ensured by a Arduino-controlled turntable.

The Data Generation stage will produce labelled 3D models of 10 products in the textured .obj format. These models will include textures and colour representation that are of high enough quality to produce realistic product images in the next stage.

The Kinect can be moved freely around the product and models are generated in real time, but produce lower quality textures and colour accuracy, and needs specific Windows-based hardware and drivers. Agisoft requires more time during capture and a separate processing stage to combine these into 3D models but may produce higher quality textures.

Both methods may not perform well for transparent items such as water bottles, or for products with flexible packaging and no fixed shape. While the capture process will be automated where possible in the time allowed, it is not intended to be scalable to a larger number of products.

Our goal for this project is to demonstrate the means by which 3D models can be feasibly created and to use these 3D models as the input to our proposed pipeline. Additionally data will be gathered on how long the process takes per product, which the client can use to evaluate scalability of our method. This, however, is not the focus of our project - for the purposes of this project, the aim will be to arrive at a method that can feasibly capture quality models of a limited number (10) of products, leading to a trained CNN classifier that generalize well into various environments.

## 2.3 Data Processing (Image Rendering)

Open source software Blender will be used to render a vast range of training images for each product. For example, the pose, lighting, background and occlusions of the product in each image will all be randomised.

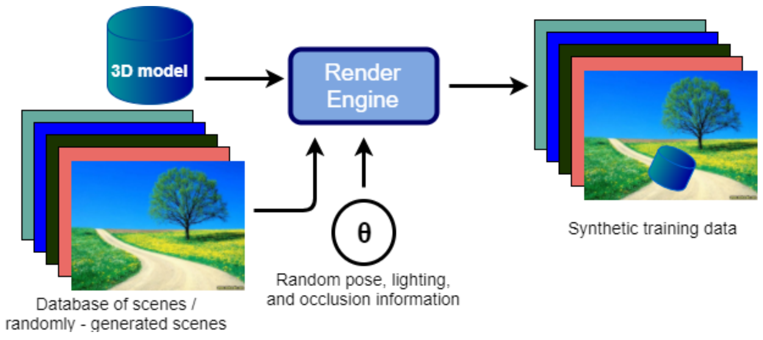
3D model
Render Engine
θ
Database of scenes / randomly - generated scenes
Random pose, lighting, and occlusion information
Synthetic training data

Figure 3: Rendering process Overview

The output images will be in the .jpeg format, which is suitable for CNN training. Blender image generation will be automated with Python scripts. This is a common practice and should not pose a problem.

A large database of realistic background images will be assembled. From this database, random images will be combined with the randomised product image to produce a unique training image. The limited size of this database could lead to repetition of the backgrounds. This might lead the CNN to focus on the background, instead of recognizing the product itself. If this proves to be a problem, 360 HDR images will be used. The product will be placed into different parts of the HDR image, leading to a more realistic variation in the background.

## 2.4 Convolutional Neural Network (CNN) Model Training

After the training data has been created, the data will immediately be used to train a CNN to produce a model capable of accurately classifying product images in a range of settings.

For the MVP the convolutional layers of an off-the-shelf InceptionV3 model will be used, whose convolutional layers are pretrained on ImageNet data [5]. This model has been shown to perform well on the ImageNet dataset and is widely used as a pretrained model. For our purposes, the dense layers of the model will be retrained. Scripts to retrain the model on our data are available and well-tested by the community [6]. Googles retraining script will be used which retrains the model using Python and Tensorflow. This is expected to involve negligible risk given the widespread use of the script. The output of the model will consist of a trained TensorFlow graph and the standardised training logbook.

## 2.5 Testing and Evaluation

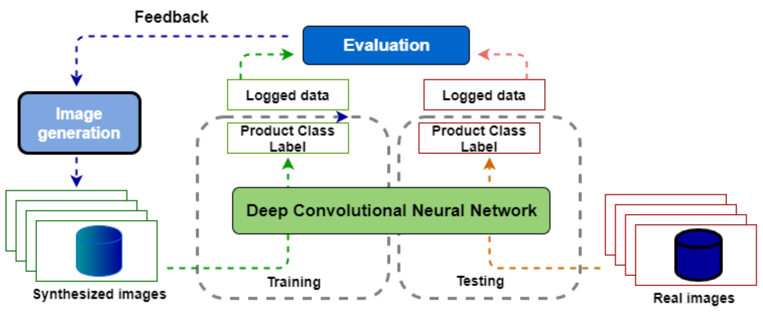

Figure 4: Test and evaluation schematic

Once the Neural Network has been successfully trained, a test and evaluation process will be necessary to identify successes and weaknesses in the preceding three steps, and to determine what changes will need to be made.

An ideal test and evaluation framework will provide us with knowledge of the following:

- Obvious faults in the previous stages of the pipeline (bad quality of rendering, non-converging loss in training etc.)
- Metrics to assess the performance of the identification process, at varying levels of granularity (e.g. accuracy between classes, within classes and over all classes)
- Influence of training data on the final test performance, and metrics that can identify problems in data generation/augmentation

It is feasible, and known to be crucial to have a test and evaluation framework in place for deep learning systems. Many machine learning frameworks provide support for diagnostic tools. The TensorFlow framework provides a visualization and data logging tool for machine learning training and test.

The boundaries of the test and evaluation system can be described as follows:

- A library and scripts to run CNN on the test data, and log information for every test image.
- Comprehensive plots of test and training performance (based on logged data).
- Library to calculate and plot any metrics identified as important, given a test log from the testing process.
- Report in an agreed format (Tensorboard) for these metrics.

## 2.6 Non-Essential Requirements (Extension)

Several non-essential features that could be implemented to improve the performance and functionality of our pipeline.

- Extend the model to recognize more than initial 10 products.
- Introduce a class hierarchy (e.g meat) to enable broader classification of very similar products.

- Investigate the use of generative adversarial networks (GAN). A Wasserstein GAN could be used to generate training images of corner cases between similar classes [7].
- Develop a GUI-based tool to to enable live demonstration of the classifier.

## 3 Software Engineering Processes

### 3.1 Development Paradigm

An Agile/Scrum development approach will be adopted for the project. The development period will be divided into 2 week sprints, with each sprint beginning with a sprint planning meeting where tasks are added to the backlog and assigned a duration agreed by the group. Team members will volunteer to take on tasks from the sprint backlog and attend three standup meetings each week to update the team on their progress. GitLab Issues and Milestones will be used to document the tasks, and will be checked in the review meeting at the end of each sprint.

### 3.2 Code management/version control system

Git has been selected as our version control system. This consists of a Gitlab Project containing separate repositories for planning and product code. Working code will be managed in a protected master branch inside the code repository. Pull/Merge requests will be handled by Kiyo (group leader) and Ong, both of whom have experience with Git in production systems. All development of new features will be done in separate branches. New features must pass all tests, before being merged into master branch.

### 3.3 Testing

Unit testing will be continuously performed on any written code, this includes any code/scripts that are written for the data generation system, and the training and test system. Furthermore, it is proposed that the Continuous Integration system on Gitlab be used to automate these tests.

### 3.4 Communication

The primary method of communication between team members is Slack, which offers a central location for monitoring of documents, commits and shared resources. Google Drive is used to share and collaborate on documents, and to collect helpful resources. Tasks are tracked and assigned with Gitlab issue tracking. Administrative tasks are coordinated by Kiyo, and distributed by various members of the group with Pavel as the Document Editor and Scrum master.

Our supervisor acts as the sole customer for the purposes of requirement specification, progress tracking and evaluation of the final system. We will meet our supervisor once every two weeks, and communicate over email for interim updates and issues.

The project is being undertaken in collaboration with Ocado, who will provide product image data and related metadata from their systems. Ocado has also offered to provide advice and further relevant data should we require it. Communication with Ocado primarily takes place using email and video-conferencing, with email used for routine communication and video calls scheduled at critical junctures in the project.

### 3.5 Schedule

The tentative development schedule is as follows. Between Week 3 and Week 5 of Spring Term, the team will be split into three pairs and will work on three tasks in parallel: Data Generation, Data Rendering, and CNN Model Training. During Week 6 - Week 7 we will integrate each part, and in Week 7 - Week 8 we will implement Testing and Evaluation. Further work will be determined by our findings during the evaluation period.

## References

[1] W. Rawat and Z. Wang, “Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review,” *Neural Computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[2] H. Su, C. R. Qi, Y. Li, and L. Guibas, “Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views,” may 2015.

[3] S. Savarese and L. Fei-Fei, “3D generic object categorization, localization and pose estimation,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2007.

[4] X. Peng, B. Sun, K. Ali, and K. Saenko, “Learning Deep Object Detectors from 3D Models,” in *ICCV ’15 Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, dec 2015.

[5] J. Shlens, “Research Blog: Train your own image classifier with Inception in TensorFlow,” 2016.

[6] Google, “How to Retrain Inception’s Final Layer for New Categories — TensorFlow,” 2017.

[7] M. Arjovsky, S. Chintala, and L. Bottou, “Wasserstein GAN,” 2017.