Machine Learning for Product Recognition at Ocado
— Final Report —

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

16th May, 2018

Contents

1 Acknowledgements

We would like to thank the following people, without whom this project would not have been possible:

- Dr Bernhard Kainz at Imperial College London for his dedicated supervision throughout the course of our project and for giving us valuable feedback and advice.
- Dr Fidelis Perkonigg at Imperial College London for teaching us about software engineering methodologies.
- Luka Milic and David Sharp at Ocado for sharing Ocado's data, insights and suggestions, and for hosting us at their HQ.

2 Introduction

Ocado is an online supermarket delivering groceries to customers across the UK. Their warehouses are heavily automated to fulfill more than 250,000 orders a week from a range of over 50,000 products and they rely on a variety of different technologies to facilitate customer ordering and fulfilment. As a result, they are interested in computer vision innovations that will allow them to better classify and identify products, as this technology can potentially be applied towards a wide range of different use-cases across the company.

The goal of the project was to deliver a machine learning system that can classify images of Ocado products in a range of environments. It was quickly agreed that a deep learning approach would be taken in order to achieve these requirements, motivated by the recent success of deep learning in the field of computer vision [1].

Based on discussions with Ocado and our project supervisor, we defined the customer specifications for our project shown in Table 1.

No	Description
1	Develop a classifier which is able to classify 10 Ocado products with accuracy in general environment above the Ocado baseline of 60%.
2	Develop a pipeline which successfully trains a neural network image classifier.
3	Evaluate the performance of the chosen methods.
4	Investigate failure cases to optimize performance.

Table 1: Customer Specifications

The standard workflow for a machine learning project is to train and optimise a Neural Network using an available high-quality data set. Ocado provided us with an initial data set that they captured automatically in their warehouse during the order fulfilment process. However, after considering the distinguishing features of our research problem and examining the quality of the data set available to us, we decided that this standard approach was not optimal.

Instead, we considered an alternative approach where we used 3D modelling to generate an unlimited number of 2D images, providing a large, high-quality, data set which we used to train an image classifier.

3D modelling has previously been used in deep learning to train image classifiers and object detectors (see *Existing Research: Training based on 3D Modelling* in section 4.2).

Our approach builds on the existing literature by generating 3D scans of physical objects using photgrammetery. While previous work has made use of computer-generated 3D models, our study has been the first to successfully demonstrate that this approach can be extended to 3D models acquired using photogrammetry.

Using our approach, we were able to successfully train and optimise a Convolutional Neural Network (CNN) that achieves a maximum classification accuracy of 96% on a general environment test set.

In order to also demonstrate a potential application of the system, we deployed the trained model deployed in REST API on a web server, and developed an iPhone app to showcase its use in an everyday setting.

The following sections describe the system design and implementation in more detail, and offer analysis of the final results and performance of the our image classifier.

3 Specifications

3.1 Internal Specifications

In order to fulfil the customer specifications, we defined a number of internal specifications for the individual components of our product as shown in Table 2. In aggregate, these specifications ensure the fulfilment of the client specification.

No	Dependency	Category	Description	Type	Estimate	Completed
1	N/A	Data Generation	Optically scan physical products with camera (iPhone and DSLR), and use a 3D reconstruction program (Agisoft) to generate 3D images in .obj format.	F	08/02/18	Yes
2	N/A	Image Rendering	Create a database of realistic, random colour mesh and plain colour background images in jpg or png format.	F	01/02/18	Yes
3	2	Image Rendering	Use the 3D model .obj file with Blender API to generate images of the object from different angles that show the unobstructed object. Generate 2D .jpg training images by merging the rendered products with database or randomly generated backgrounds.	F	01/02/18	Yes
4	1-3	Training	Train Tensorflow-based InceptionV3 Convolutional Neural Network (CNN) model using the training images we generated, as well as a Keras-based InceptionV3 model	F	07/02/18	Yes
5	4	Evaluation Optimisation	The trained model should be able to classify 10 products with accuracy higher than Ocado's baseline ($\tilde{6}0\%$).	F	15/03/18	Yes
6	4	Evaluation Optimisation	Evaluation results must be available on Tensorboard.	NF	07/02/18	Yes
7	6	Evaluation Optimisation	Results of experiments must be collected with experiment parameters on Tensorboard.	NF	01/04/18	Yes

Table 2: Internal Specifications (Essential), F: Functional, NF: Non-functional

No	Dependency	Category	Description	Type	Estimate	Completed
8	1-6	Image Rendering	Users are able to upload images, and get the result of classification through GUI. This will be implemented within an iPhone app.	F	30/03/18	Yes
9	4	Training	Baysian optimisation is conducted to fine-tune parameters in both model pipeline as well as rendering.	F	22/05/18	prototype
10	4	Training	Region-CNN is added to further improve the accuracy of the classification.	F	22/05/18	prototype

Table 3: Internal Specifications (Non-essential), F: Functional, NF: Non-function

In response to implementation challenges and new opportunities discovered in the process of our work, some specifications were altered over the course of our project. Our original specifications and a full details of changes made can be found in Appendix A.

4 Design

4.1 Problem Statement

The main challenge for producing an accurate classifier was the biased dataset provided by Ocado. While this dataset was very large (more than 1000 images for each product), 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, see Figure 1).

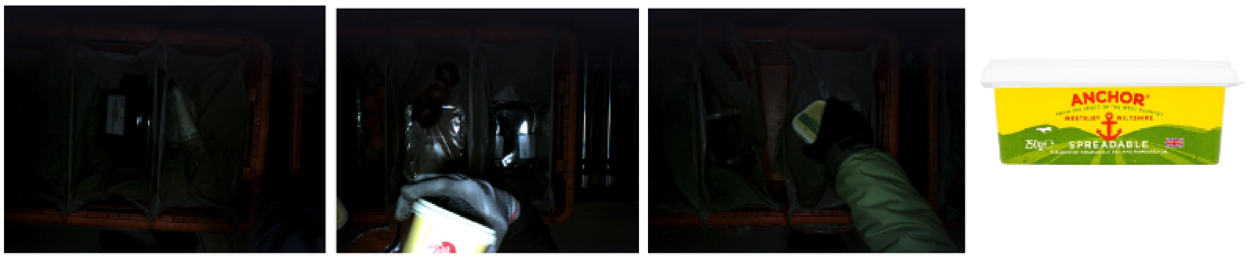


Figure 1: Examples of Ocado warehouse image data for 'Anchor Butter' (left 3 images) as well as the actual product that was meant to be depicted (from Ocado.com, right)

We realised that it in order to achieve our objective of high performance in a general environment it would not be suitable to directly train our neural network using Ocado's data set, and that a different approach would be required.

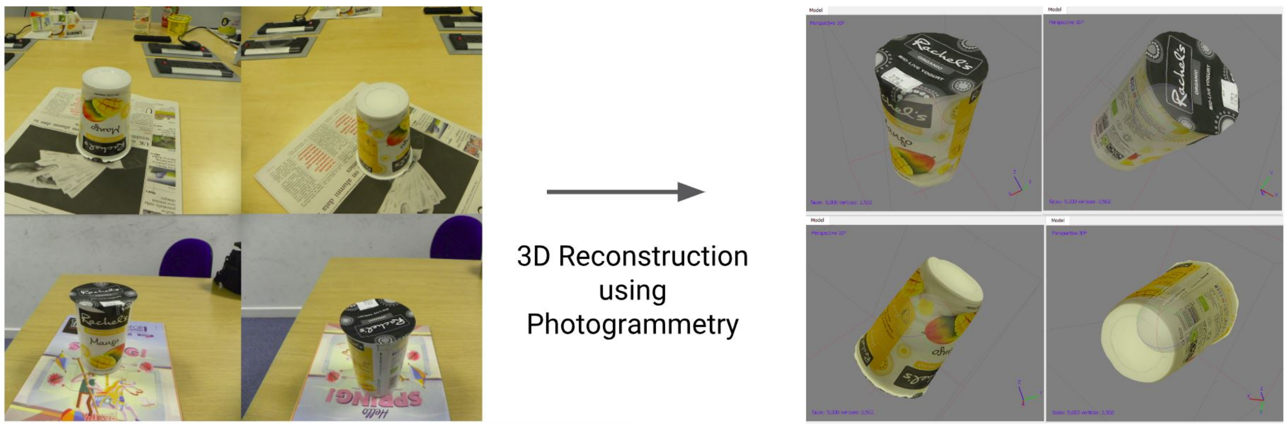


Figure 2: 3D Reconstruction

4.2 Key Observations

Following research into training data and augmentation, it became apparent that groceries exhibit a distinctive feature which differentiates product recognition from typical image classification tasks: products have very low intra-class variation. Hence we explored ways to train a neural network such that it would be able to fully capture the features of a particular product. The approach chosen for this work is acquiring 3D models of the products using photogrammetry as shown in Figure 2.

Using 3D models for image classification tasks for product recognition is thought to be feasible and effective for the following reasons:

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

While 3D modelling has long been a mainstay of computer vision research, it is only more recently that its potential applications to deep learning-based image classification have been considered. *Existing Research: Training based on 3D Modelling* provides a brief overview of the relevant papers published in this field.

Existing Research: Training based on 3D Modelling

1. Su et al., 2015 use 3D models for viewpoint estimation. They use CAD 3D models, render them such that they appear like realistic images from which they generate training data for a CNN. The CNN is trained to detect the viewpoint of objects.[2]
2. Peng et al., 2015 use a large number of 3D CAD models of objects to render realistic looking training images. The output is used to train a classifier for classifying real world images of the objects.[3]
3. Sarkar et al. 2017 similarly use 3D CAD models to re-train a pre-trained CNN to recognise real-world objects. They describe different rendering parameters including viewpoint distribution and also show the usage of different backgrounds with the rendered images.[4]

4.3 Design Choices

4.3.1 Standard Pipeline Design vs Custom Pipeline Design

Under the standard design that is applied to most deep learning projects, a pre-existing data set would be used to train a neural network, which would then be evaluated and optimised.

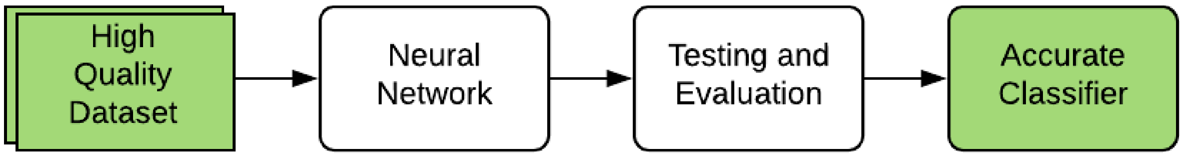


Figure 3: Standard Pipeline Design

While the standard pipeline works well when a high-quality data set is available, given the challenges described above inherent in the data set we were provided with, the standard pipeline designed was not considered to be a viable option.

Specifically, instead of training a neural network on a pre-existing data set, we decided to generate our own data and to curate our own data set using 3D modelling and Image Rendering techniques.

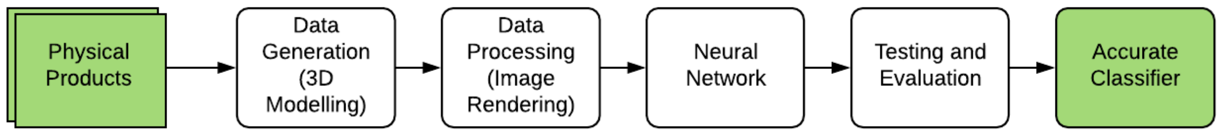


Figure 4: 3D Model Pipeline Design

4.3.2 Image Rendering

An interface between the generated 3D models and the input to the neural network was also necessary, using 3D models as the direct input to a classifier is is highly complex and would not achieve our goal of producing a scalable system for classifying 2D images.

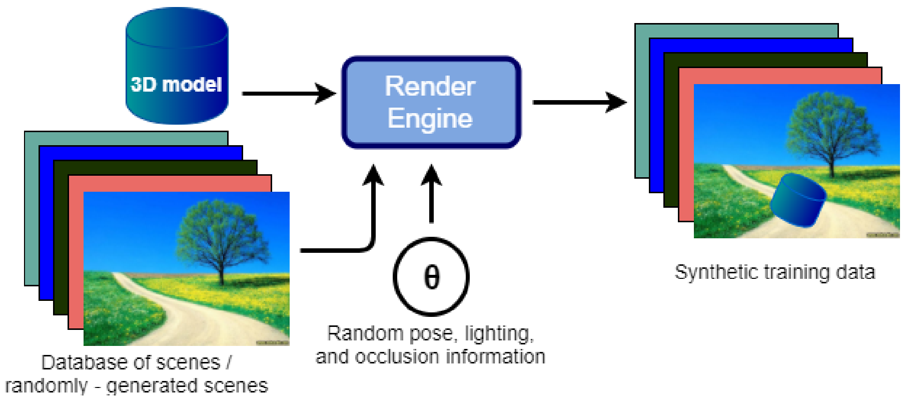

3D model
Render Engine
θ
Database of scenes / randomly - generated scenes
Random pose, lighting, and occlusion information
Synthetic training data

Figure 5: Image Rendering Schematic

We developed an image rendering system that would take a 3D model as its input and produce a set of training images as its output, given a number of rendering parameters θ as shown in Figure 5. The system would use the 3D model to produce multiple images showcasing the modelled product from all possible viewpoints, at different scale, under various lighting conditions, with different amounts of occlusion and with varying backgrounds.

A classifier trained on such generated data is expected to be robust to varying backgrounds, light conditions, occlusion, scale and pose. Furthermore, it allows the user to tailor the training set to a particular environment for which the image classifier will be deployed.

4.4 Final System Design

Our final system design shown in Figure 6 incorporated the key design choices describe above. These resulted in a custom neural network pipeline which goes from generation of 3D models to a customised evaluation suite used to optimise classification accuracy.

The individual component functionality is outlined as follows.

- **Data Generation**: provides 3D models of 10 products in .obj format. These models include textures and colour representations of the product and have to be of high enough quality to produce realistic product images in the next stage.
- **Data Processing (Image Rendering)**: produces a specified number of training images for each product which vary product pose, lighting, background and occlusions. The type of back-

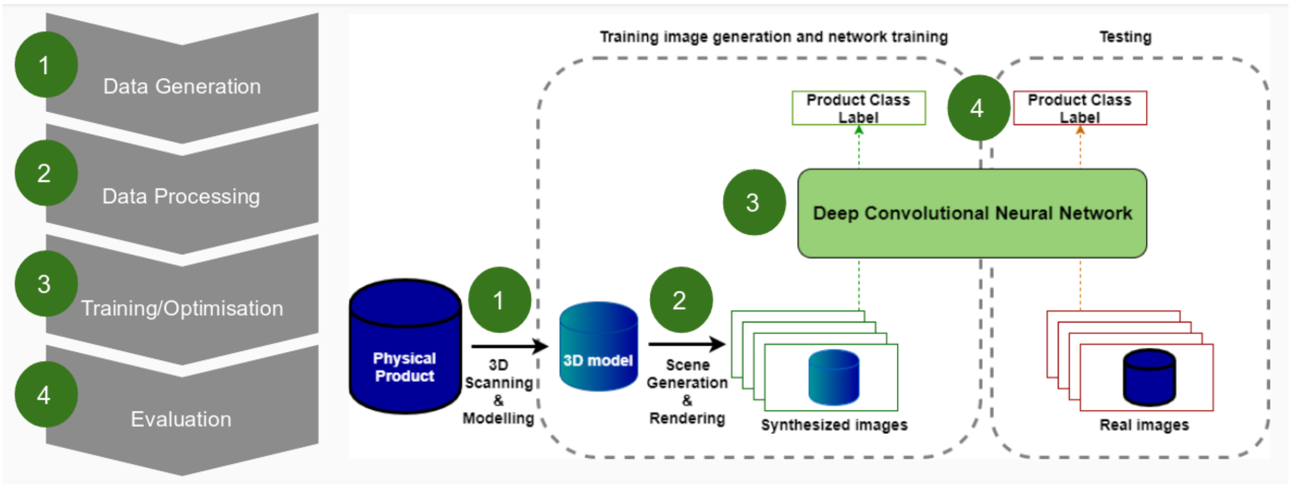


Figure 6: Final System Design

ground can be specified by the user. Both the rendered product and a background from a database are combined to create a unique training image in .jpeg format.

- **Neural Network**: the produced images are fed into a pre-trained convolutional neural network. The resulting retrained classifier should be able to classify real product images.
- **Evaluation and Optimisation**: the outlined approach to training data generation means that the training data can be tailored based on results. Therefore, a custom evaluation and optimization suite is required that is not provided in sufficient in detail in off-the-shelf solutions.
- **Integration and GUI (Extension)**: the user is able to deploy the trained neural network through an iPhone app (i.e. classify products). Further, a user can generate custom training sets and customized networks given a set of parameters using a simple script.

The following options are some of the product options which were considered but disregarded:

- **Training Suite based on Warehouse Data**: Augmentation and enhancement of the provided training data can lead to an accurate model for warehouse environments. However it would not generalise. Therefore effort of this project focuses entirely on the 3D rendering approach.
- **Generative Adversarial Networks (GANs)**: This method would allow further training data augmentation and filling gaps between training classes. Given the ability of our procedure to generate an unlimited number of data, this was a lower-priority issue for this project.
- **Training Data Augmentation**: Augmentation of both the provided training data as well as the generated training data was considered as input for the classifier. Similar to GANs, this was not made a priority for the following work due to estimated lower impact on results.
- **Training from Scratch**: Given the large amount of training data this approach generates, convolutional neural networks could be trained from randomly initialized weights. However, for this foundational work, it appears sensible to move forward with the commonly used practice of transfer learning.
- **Web based GUI**: We considered a web-based interface and built a prototype that allowed users to upload an image and receive a classification. However, we decided that an iPhone app provided a better user experience as it provided a seamless interface handling photo taking and image upload, as well as direct interaction with the API.

5 Implementation and Methodology

5.1 Software Component Breakdown

We divided our implementation of the design outlined above into five separate software components. Three of these components, BlenderAPI, RandomLib and SceneLib correspond to the Image Rendering Stage, the Keras component corresponds to the Network Training Stage, and the EvalLib component corresponds to Evaluation Stage.

A more detailed view of the software including the initial Data Generation (3D Capture) stage is presented in the data flow diagram in Figure 7. Modularity was introduced by defining interfaces between the 4 components (denoted by dotted boxes), allowing each component to be developed and tested in parallel.

The Image Rendering, Training and Evaluation stages can be operated independently through their respective Python interfaces or through a single interface that enables easy use and full automation of the pipeline with detailed logging, error reporting. This can be run by a specifying a set of parameters for a single rendering or training job or by specifying parameters for Bayesian optimisation over a number of jobs. A lightweight Slack connector is also included for convenient monitoring of long rendering and training jobs, providing automatic updates on job status and any errors.

Stage	Component	Description
Image Rendering	BlenderAPI	Wrapper around the Blender Python interface, with utilities to modify the state of the Blender environment, and generate random images of object models.
Image Rendering	RandomLib	Library to generate random variables (colors, coordinates, textures) for random object pose and background generation.
Image Rendering	SceneLib	Library to query and produce random background images from a database and merge them with object poses to create training images.
Training	Keras	A script which takes pre-trained weights for a convolutional neural network and fine-tunes these weights based on our data.
Evaluation	EvalLib	Script to test network on unseen images, and generate various evaluation metrics, including precision and accuracy, presented in Tensorboard.

Table 4: Overview of Software Components

5.2 Technical problems to be solved

Given the novel nature of the pipeline, a number of technical challenges were identified in our initial feasibility assessment of the proposed design. These are outlined briefly below and covered in more detail in the sections 5.3.

- Optical scanning of physical products to generate a 3D reconstruction is a challenging engineering problem in its own right that is largely beyond the scope of this software. This introduces reliance on existing software and techniques with well-known limitations.
- Enabling a high degree of flexibility in rendering parameters whilst ensuring rendering components remain reliable (with proper logging and error handling) is challenging during very long jobs. Longer jobs are required to render sufficiently large amounts of training data.

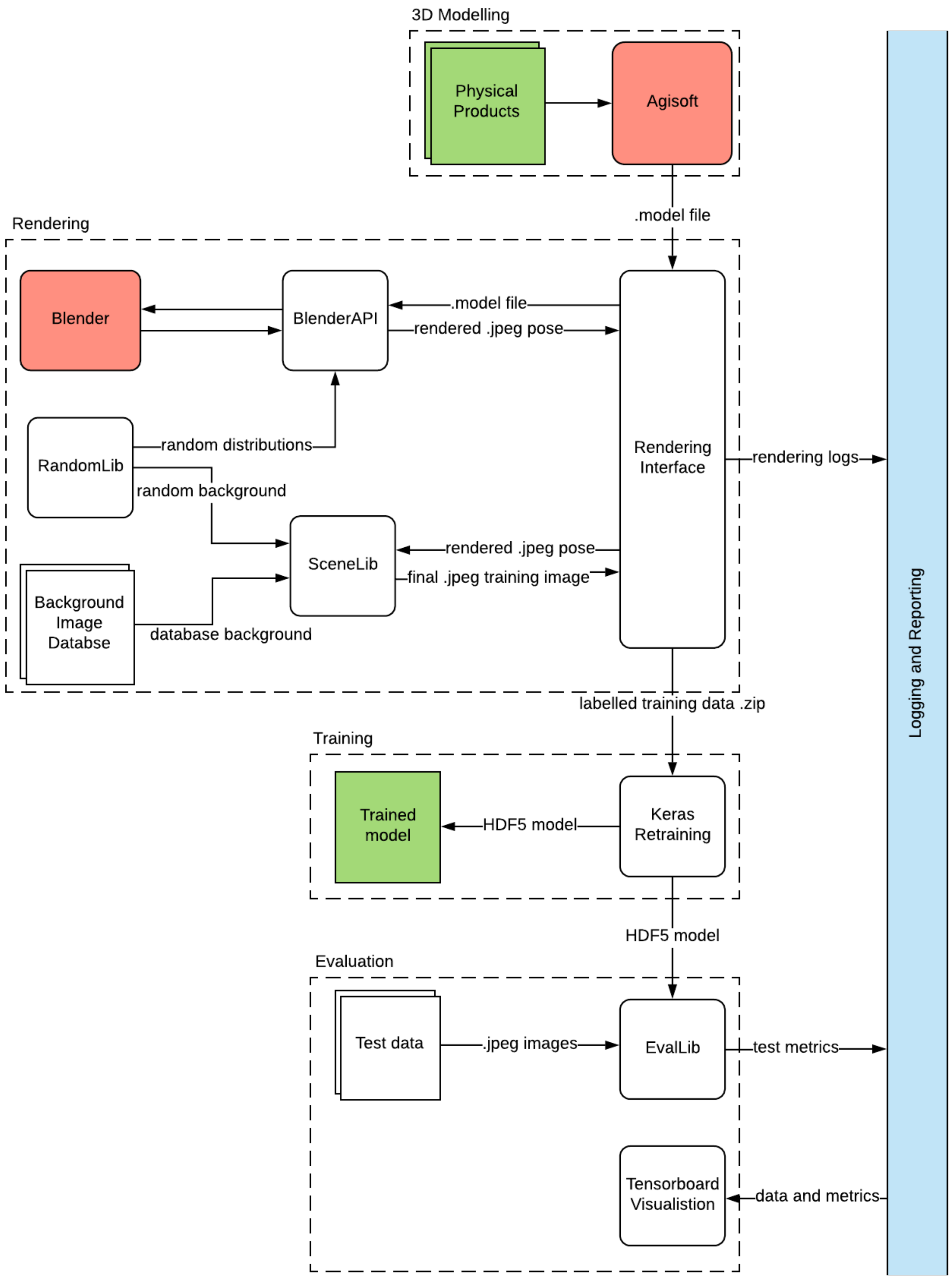


Figure 7: Data Flow Diagram for the 3D Capture, Rendering, Training and Evaulation Software Components. 3rd party software is coloured red and inputs/outputs are green. Data flows, including file formats are denoted by arrows between the libraries that were created. Dotted boxes donote each of the 4 stages.

- Use of off-the-shelf CNN architectures simplified the engineering aspects of the network training component, but some complexity is still involved in optimising training parameters, supporting automation and proper integration with the rendering and evaluation sections.
- This project was the first to apply rendering to Ocado groceries, so an extensive suite of evaluation tools were needed to assess the success of our approach. This was also important in order to optimise rendering parameters.

5.3 Implementation of Individual Components

5.3.1 3D Modelling

The goal for this stage was to demonstrate a means by which 3D models can be feasibly created and to use these 3D models as an input to our proposed pipeline.

Two methods were considered for generating 3D models:

1. An infra-red scanner such as Microsoft Kinect to scan the products and associated Windows 3D Scan software to generate object files.
2. Using images taken with a conventional camera as an input to a photogrammetry software such as Agisoft PhotoScan [1] or Qlone [2] as an alternative method of model creation.

These two methods were assessed in terms of the following criteria:

- Ability to generate output models with textures and colour representation of high enough quality to produce realistic product images for the training stage.
- Consistency in quality between different product models.
- Time and number of people needed to scan a product.
- Specialised computing resources (e.g. GPU) and hardware needed.

While the Kinect capture method initially seemed promising, it quickly proved to be problematic on a number of fronts. Kinect works best with specific Windows-based hardware and drivers, while the rest of the pipeline was developed on a linux environment. Most importantly, the Kinect had lower quality textures and colour accuracy, primarily due to the size and shape of products being scanned. Whilst the Kinect works well for scanning large objects such as people, small circular objects that are common among supermarket products did not exhibit sufficient variation in depth to enable accurate camera tracking by the Kinect.[5]

Despite using less specialised hardware, photogrammetry using both Agisoft and Qlone ultimately proved superior on all our evaluation criteria. The models produced by Agisoft had very high quality textures and took around 5 minutes to photograph with a normal digital camera(requiring roughly 30 images ro reconstruct a model). The main weakness of the photogrammetry approach was that the models were typically missing the bottom of the product, but this was easily resolved by generating two models covering all features and modifying the rendering component to select the correct model automatically depending on camera position.

[1]info: http://www.agisoft.com/about/
[2]info: https://www.qlone.pro/

Photogrammetry is the process of reconstructing 3D surfaces using 2D images, which is achieved using the following steps (illustrated in Figure 8):

1. Camera Calibration. This is done automatically using matching features in the images, and estimating the most probable arrangement of cameras and features. A sparse point cloud of features of the modelled surface is calculated in the same step as the camera calibration.
2. Depth determination. This involves finding all matching features between the camera views and recovers depth information by calculating a dense point cloud.
3. A 3D mesh is then created from the dense point cloud and texture information recovered by combining information from the images.

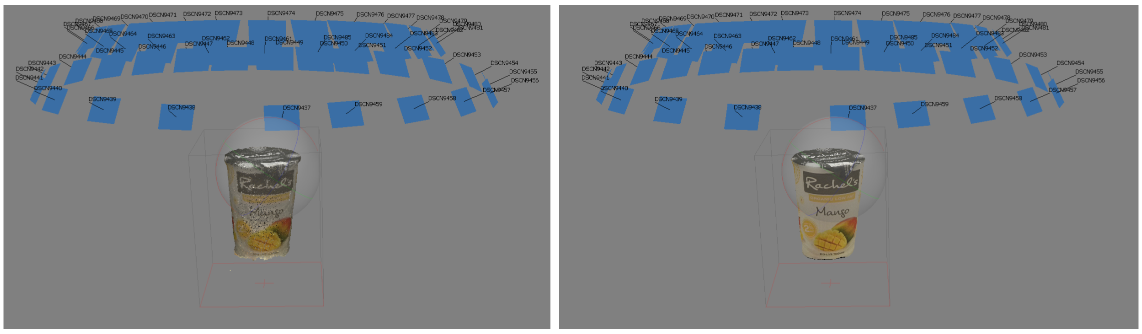


Figure 8: Screenshots showing the stages of the 3D Modelling pipeline (left: camera calibration and dense point cloud generation, right: mesh and texture generation)

5.3.2 Image Rendering

The image rendering component takes a single .model file for each product as an input, and generates a potentially unlimited number of labelled training images consisting of a rendered view of the product on a pre-existing or dynamically generated background image. This is typipcally a photograph taken in an indoor setting or random noise. Final merged images are saved in the .jpeg format, which is a suitable input for the training and evaluation components.

Several different software tools and techniques were combined into a single user-friendly Python tool. Blender[3], an open-source software package for scene rendering was used as the main rendering engined as it allows programmatic access via a bundled Python distribution. The BlenderAPI library was created to provide a high level object-oriented Blender interface for scene generation and customisation.

Using BlenderAPI, the user can controls all relevant parameters that fully define a scene:

- Camera position and angles
- Camera distance
- Lighting (intensity or equivalently distance, number of sources)

This library provides easy access and full control over the randomness and its distribution of these parameters, although one could also easily choose to make this deterministic if desired. A detailed dataflow diagram corresponding to this system is shown in Figure 9, showing the modules that control specific rendering objects. An explanation of the distributions used can be found in Section 8.4.1, and a detailed mathematical definition of these can be found in Appendix E.

[3]https://www.blender.org/

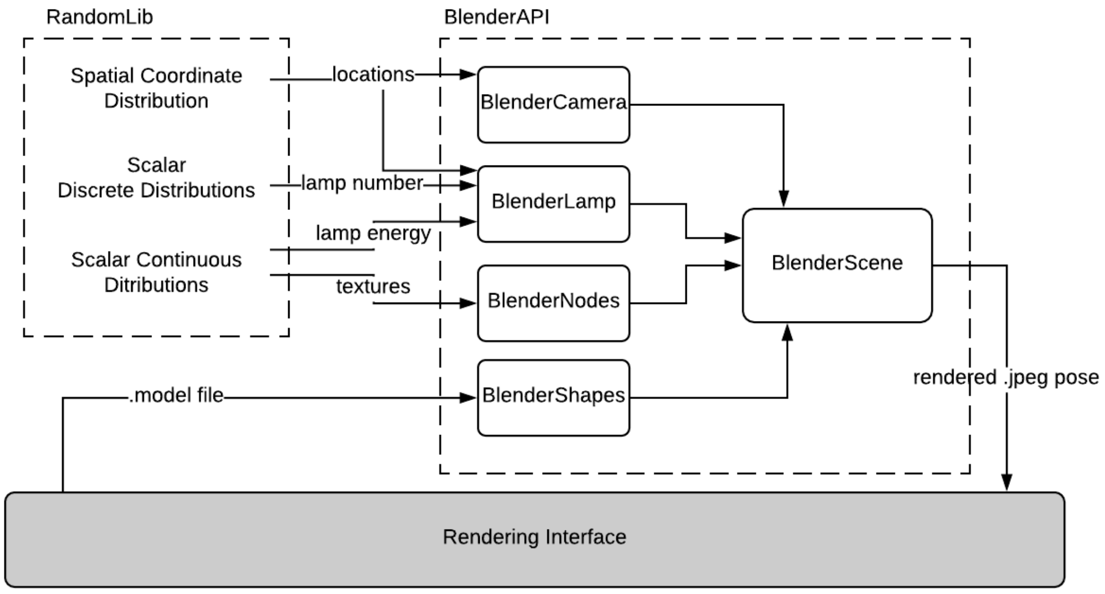


Figure 9: BlenderAPI Dataflow Diagram

The rendering process generates detailed statistics recording the lamp and camera locations used by BlenderAPI to illuminate the scene and capture product poses. Evaluation graphs (Figure 10) are generated automatically and saved for later inspection.

Each rendered product image was combined with a background in SceneLib. The Python Imaging Library and alpha composition [4] were used to stitch background images onto our foreground image. Backgrounds can be customised easily in the rendering interface, where the user is given a choice between background images taken from a database or dynamically generated random backgrounds generated using RandomLib.

For the former, a large database of realistic background images was assembled from open-access data[6]. From this database, random images were combined with the randomised product image to produce a unique training image. A possible challenge identified was that the existence of repeats might lead the CNN to focus on the background, instead of recognising the product itself. To avoid this the images are selected at random, and the database is large enough (over 80 000 distinct images) to avoid a significant amount of repetition between training images.

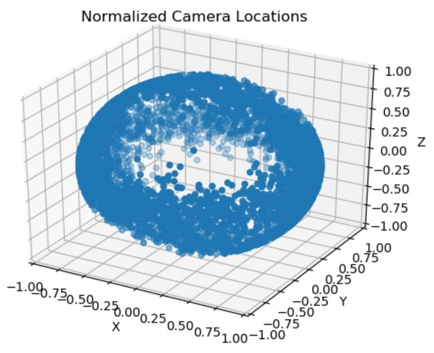
5.3.3 Convolutional Neural Network (CNN) Model Training

The Network Training stage took a generated dataset as an input, and produced a trained Neural Network that could be used to classify products from the dataset. The tool of choice for this task was a Convolutional Neural Network(CNN). These are specialized neural networks that perform transformation functions (called convolutions) on image data. Deep CNNs contain hundreds of convolutions in series, arranged in various different architectures. It is common practice to take the output of the CNN and input it into a regular neural network (referred to as the fully-connected or FC layer) in order to perform more specialised functions (in our context, a classification task).

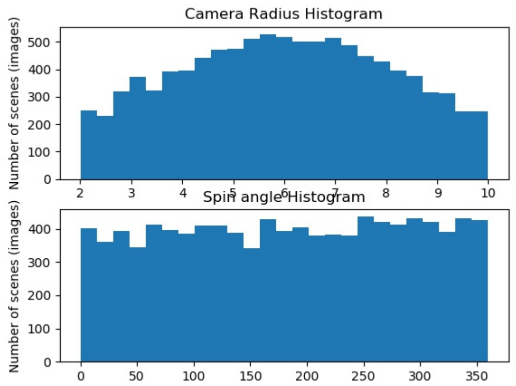
For an initial proof of concept, stock images from the Ocado website showing the products on a plain white background were augmented and used as training data for a basic model to provide an initial baseline for comparison with first models trained on rendered data.

After using 3D modelling and Image Rendering to generate a new dataset, we used this dataset to train a CNN capable of accurately classifying product images in a range of settings.

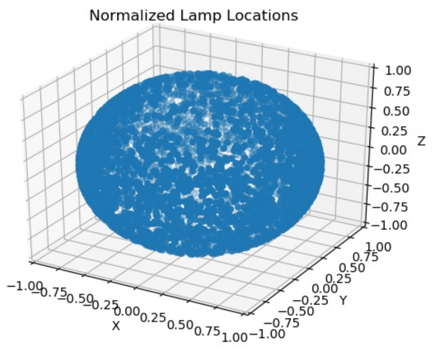
[4]Details on alpha composition: https://en.wikipedia.org/wiki/Alpha_compositing



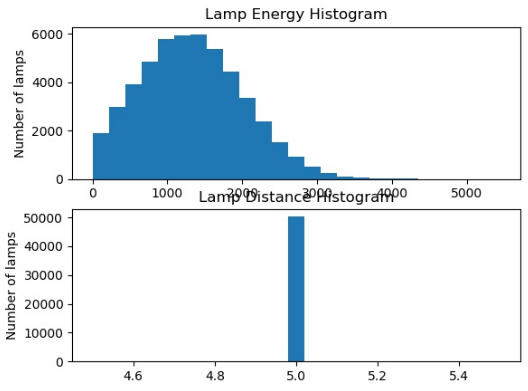
(a) 3D scatter plot showing normalized camera locations(coordinates divided by camera distance from object)



(b) Histograms showing the distribution of camera distance from object(top), and camera spin angle in degrees (bottom)



(c) 3D scatter plot showing lamp location distribution around object



(d) Histogram showing distribution of lamp energy and lamp distance from object

Figure 10: Example visualization of image rendering parameters logged during rendering process. Datapoints are logged per scene (i.e. per image generated). The histograms shows the distribution of its recorded values over all scenes.

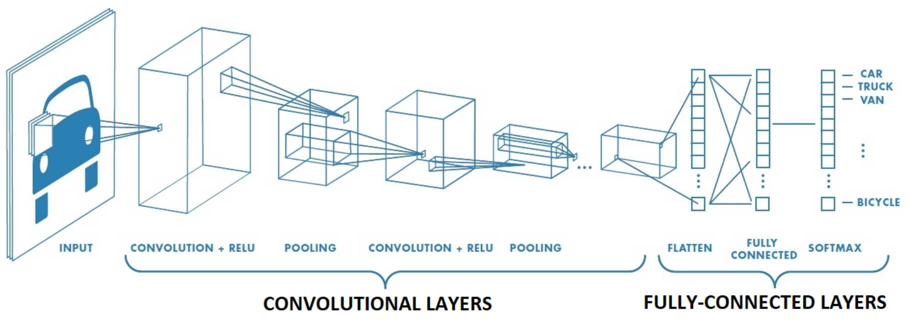


Figure 11: Illustration of the convolution layers and fully-connected layers of a CNN in a classification task (from mathworks.com)

The InceptionV3 [7] model was chosen as the basis for our network architecture. InceptionV3 was developed by Google and has shown great success in classifying images from the ImageNet dataset [8]. It is widely used by the deep learning community as a pretrained model and as a basis for further fine-tuning and retraining. The ImageNet dataset contains a large number of classes representing real-world objects; it was thus expected that InceptionV3 would also perform similarly well on our closely related task. At the later stages of the project, we also experimented with other potential models, in particular VGG[9] and ResNet[10], to determine if these networks could provide an additional boost in performance.

Each training run began by initialising our chosen network architecture with ImageNet[8] weights. This is an example of transfer learning, which helps reduce the time it would otherwise take to train a network completely from scratch.

We then proceeded to retrain the network's layers, which had the effect of optimising the network weights for our particular dataset. We retrained the Dense layers (the top layers of the network) as well as a variable number of Convolutional layers (the lower layers of the network), while keeping the remaining (if any) Convolutional layers frozen.

During each training run, the following parameters were tuned, providing data to be used at the test and evaluation stage:

- Architecture of top layers
- Number of frozen layers
- Learning rate
- Optimiser
- Momentum

Retraining was conducted using both Tensorflow and Keras, with Tensorflow used intially and Keras used in the later stages of the project. High-level python wrappers were built around these libraries which significantly simplified its use in our project:

- **Tensorflow:** Google's retraining script was used, which retrained the model using Python Tensorflow. The output of the retraining was a trained Tensorflow graph and the standardised training logbook.
- **Keras:** Keras provided a high-level framework built on top of Tensorflow, allowing retraining scripts to be written in Python. The output of the retraining was a trained Keras model stored in a H5 file.

5.3.4 Evaluation

Once the Neural Network has been successfully trained, analysis and evaluation processes were developed to assess the performance of the model. This is a crucial component of deep learning systems.

An ideal analysis and evaluation framework will provide 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 classification process, at varying levels of granularity (e.g. classification accuracy, confidence interval, confusion matrix.)
- Impact of training data variation on test performance, and metrics that can identify problems in data generation.

Given the novel nature of the pipeline, additional functionality was added to the TensorBoard visualisation and data logging tool to display misclassified images and help inspection of the training data generated in the preceding stages.

The functionality of the analysis and evaluation system can be described as follows:

- A library and scripts to run the trained CNN model on the test data specified (e.g images in general environment, or in warehouse condition), and record both prediction class label and correct class label (as well as whether the classification was correct or not).
- Log information for every misclassified images to allow us to visually analyse which test images are misclassified and how by displaying the incorrect class labels.
- Comprehensive plots of training performance (based on logged data) in the form of confusion matrix and confidence interval for all 10 products as well as overall classification accuracy.
- Report in an agreed format (Tensorboard) for these metrics.

5.3.5 Graphical User Interface (GUI)

The aim of this stage of the project was to provide a simple and intuitive GUI that a user could use to interact with our trained CNN. This would demonstrate the potential of our software, as it would show how a CNN produced using our pipeline could be applied towards a real-world use case.

We decided to implement this by developing two additional components, an iPhone app supported by a webserver handling requests to a classification API.

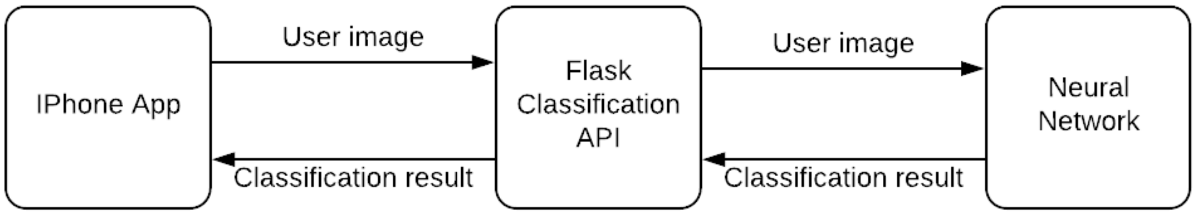


Figure 12: Interaction between iPhone App and Web Server

The iPhone app was developed in Swift 3 using Apple's Xcode Integrated Development Environment (IDE). The app was designed to provide users with the ability to take a photo with their phone camera and receive a classification result. This was implemented by sending the photo in a HTTP POST request to the classification API, which then responded with a JSON file containing the classification result.

We first implemented the basic photo capture functionality by creating a simple camera app, and built upon this by adding additional UI and design elements. HTTP networking was implemented using the open-source AlamoFire framework [11].

The server-side API was implemented with Flask, a Python-based web framework [5]. When the API received a HTTP POST request containing an image to classify, it used the image as input to the CNN. It then formatted the Neural Network's output, sending the final result back to the original client which submitted the request.

[5]info: http://flask.pocoo.org/

6 Software Engineering

6.1 Schedule

Our complete development schedule can be found in Figure 13. Our original schedule is denoted in blue, while changes to the schedule over the course of the project are denoted in yellow.

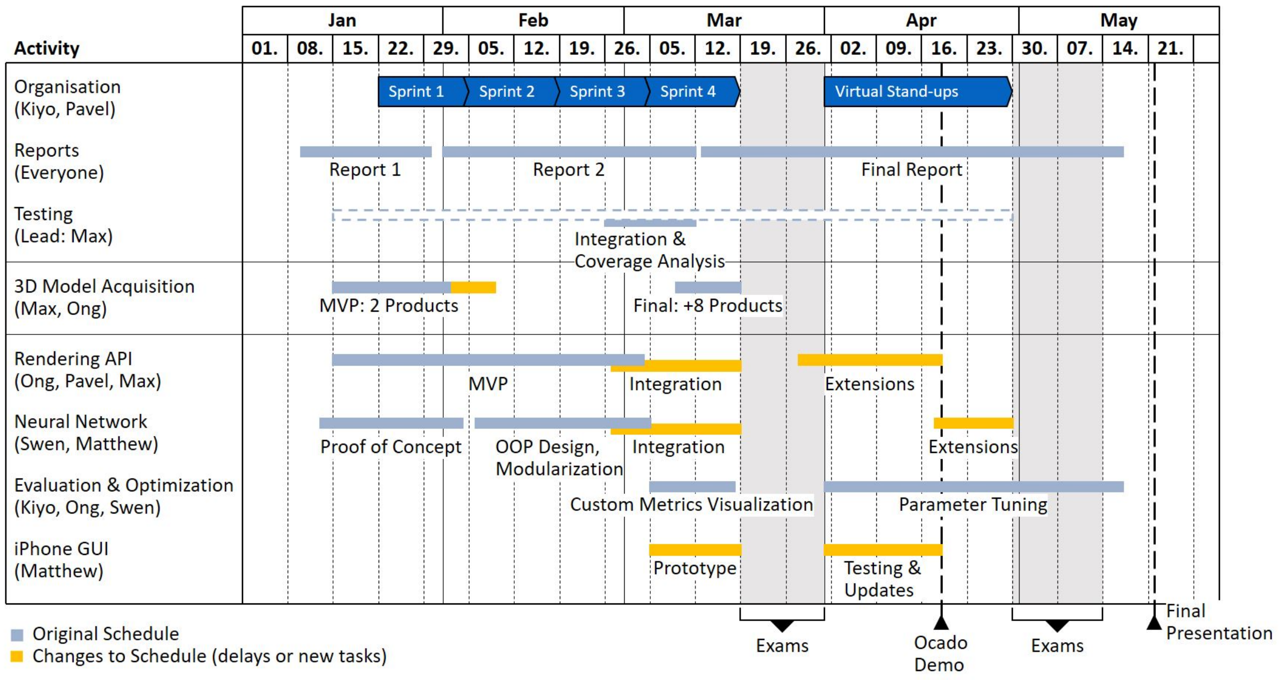


Figure 13: Project Schedule and Changes

6.2 Software Engineering Techniques

6.2.1 Agile and Team Communication

An Agile/Scrum development approach was adopted for this project. The development period was divided into 2 week sprints. Each sprint started with a sprint planning meeting. Before the meeting team members added tasks (Gitlab Issues) to the backlog. During the meeting it was decided which tasks needed to be completed in the upcoming sprint. Team members then volunteered to take the tasks from the sprint backlog. During the sprint all team members met for three 15-minute-long standup meetings each week to update the team on their progress. Each Friday evening a short write up was submitted by each member to summarize his progress. GitLab Issues and Milestones where used to document the tasks and log time spent. Before each sprint planning meeting, Pavel (Scrum Master) ensured that any unfinished tasks from last sprint were moved to the next one, while addressing the reason behind the non completion of the task.

The other primary method of communication between team members was Slack, which offered a central location for discussion and resource sharing. During the holiday periods, we also used Slack for virtual stand-ups.

6.2.2 Code management/version control system

Git was selected as our version control system. Working code was managed in a protected master branch inside the code repository. All development of new features was done in separate branches.

New features were tested, before being merged into master branch, to ensure they did not break the main branch. An overview of the sprints along with an example Gitlab issue is shown in appendix D.

6.3 Unit Testing

Our unit testing focused on ensuring reliability rather than robustness. Given the research-based (rather than client-facing) nature of our project, most inputs could generally be assumed to be of an expected nature. In other words, our unit tests ensured that our systems produced appropriate output when given expected inputs. If unexpected input was passed an error was raised. The general approach chosen was white box testing, using the Python unittest module. Implementation of the low level specification was tested, and tests were carried out with inputs partitioned into both correct and invalid inputs.

Our code base contained functions and libraries that have very different and specific uses. The unit tests written for each part were designed with these distinctions in mind to produce tests that best evaluate each specific functionality.

6.3.1 Blender API (custom wrapper) Test Strategy

The main objective of our testing was to ensure that each API call results in a correct change in the internal state of the Blender software. Every function was individually tested, and testing proceeded as follows:

- Create a clean Blender environment
- Create a class instance (if applicable) and test the functions with partitioned inputs. For instance certain methods expect normalised inputs (scalar or vectors). We test it to check if the appropriate errors are raised when illegal inputs are provided
- Inspect the output and internal state of the object/function for BlenderAPI
- Inspect the change the function has had on the Blender environment
- Verify that the correct state changes have taken place

6.3.2 RandomLib and SceneLib Test Strategy

The unit tests focused on ensuring that the individual functions input and output were compatible and that the final image was of the correct format (JPEG) and size (300*300 pixels) for CNN training. The images generated during test were retained after the test ends for visual inspection.

The generation of background and final images used a large number of randomly generated values and there was thus a range of correct values rather than single correct output. This posed the risk of correct values being generated, even when the underlying function was incorrect. To mitigate this risk the necessary tests were run multiple (5-10) times.

6.3.3 Keras and EvalLib Test Strategy

Testing of neural networks proved to be a non-trivial task as the output is not known in advance. However, we have implemented some sanity checks which ensured that the model behaves as expected and that it continues to do so after changes. First, using a few product images as training input, we checked whether the correct layers are actually being trained and undesired ones are not trained (pre-trained frozen layers). Second, we checked whether the inputs and outputs of all the layers tie into each other, to check whether the layers in the codebase are indeed connected. And lastly, we trained the entire architecture for a short period on a few images of two products to see whether it performs significantly better than random in classifying the two after training (as is expected of a working model).

6.3.4 iPhone app and Flask Server Testing Strategy

Unit testing for the iPhone app was conducted to cover the basic functionality of the app and its connection to the Flask web server. Given that the app was developed in a separate enviroment and language (Xcode IDE and Swift) from the rest of the project, these tests were not included in the main regression testing suite. Swift testing was conducted using the native test functionality provided by XCode. Unit tests were written for the Flask web server to ensure that all implementation functions returned results of an appropriate format so that the API as a whole would always return appropriate results.

6.3.5 Regression Testing

In order to ensure new commits did not break existing features, we performed regression testing using Gitlabs Continuous Integration system[6]. We set up a test runner on a separate virtual environment (VM) and used Docker [7] to install the dependencies into a clean testing environment for each test run (see Appendix C).

6.3.6 Code Coverage

The project's unit test suite was designed to be runnable from a single Python script with a simple command line interface which allows the user to specify which parts of the software should be included in the test run. The script is located within a separate 'test' directory to provide easier access to the individual unit tests which are located with their associated source files. This structure was necessary as different parts of the source code require different dependencies.

Tested Section	Statement Coverage	Branch Coverage
BlenderAPI	91%	78%
RandomLib	91%	80%
SceneLib	96%	92%
Keras	85%	72%
EvalLib	93%	72%
Rendering Pipeline	92%	82%
Flask Webserver	82%	100%
Overall	90%	82%

Table 5: Code Coverage Summary

Coverage.py reports statement and branch coverage for each source file; the full results of the HTML report can be found in Appendix B. Table 5 includes a combined branch and statement coverage figure for each module.

Our overall statement coverage currently stands at 90%, and overall branch coverage is 82%.

All of our coverage figures are currently at an acceptable level, but a few important observations can be made on the basis of these figures. The statement coverage is almost always better than the branch coverage as most of the untested branches are exception handling branches with few lines of code. It was deemed more important to test that the main logic (usually containing only small number

[6]https://about.gitlab.com/features/gitlab-ci-cd/
[7]https://www.docker.com/

of branches) is performing perfectly, rather than testing every possible invalid input that would raise an exception.

We excluded third party code from both testing and coverage, including Blender's bpy library and the standard Tensorflow base script.

6.4 System Testing

System testing was carried out in order to assess whether or not system as a whole meets the requirements outlined in the specifications. Our general approach was to define inputs and expected outputs corresponding to each specification (see Table 6), run the corresponding part of the program that is responsible for it, and validate the output with the expected output. For each section, different types of system testing strategies are employed depending on the requirements specified.

Spec No.	Test Type	Test Description	Validation Method	Passed
1	Usability Testing	**Input:** Real-world product **Output:** 3D model in the form of OBJ files	Visual inspection of correct file type	Yes
2	Compatibility Testing	**Input:** Request for random image **Output:** Random image sampled from database	Validation of query method.	Yes
3	Compatibility Testing	**Input:** OBJ files **Output:** Set of correct training images	Visual inspection, and validation of image properties	Yes
4	Usability Testing	**Input:** Set of training images **Output:** Trained Tensorflow/ Keras model	Variables (training accuracy, loss) converges	Yes
5	Reliability Testing	**Input:** Trained Tensorflow/Keras model, test data **Output:** Accuracy statistics	Verification of high accuracy on test sets	Yes
6	GUI Testing, Reliability Testing	**Input:** Tensorflow evaluation statistics **Output:** Tensorboard GUI display	Visual verification, validation by cross-checking values	Yes
7	Reliability Testing	**Input:** Networks from multiple runs. **Output:** Collated test results and visualizations.	Visual verification, validation by cross-checking values	Yes
8	GUI Testing, Usability Testing	**Input:** Test image **Output:** Correct classification displayed	Automatic Testing	Yes

Table 6: System Testing

6.5 Documentation Strategy

The code was documented in the following consistent way. Each file contains a short header description of its content and purpose. Each function and class is further documented at the beginning of its method body. This function documentation also contains a description of the input and output parameters. Furthermore, each module or library also contains a README file which provides high level information about the module purpose and instructions on how to use it. Unit tests were not in general documented as their name was mostly self-explanatory. In cases where it was not the case or the test was more complex, further comments with explanations were added.

7 Group Work

The division of work as well as corresponding specifications (specification details in Table 2) are as follows.

Team Members	Roles	Spec No.
Kiyohito Kunii (Group leader)	Overall Management, Model Evaluation	6,7
Pavel Kroupa (Documentation Editor)	Scrum Master, SceneLib	1,2
Ong Wai Hong	Data Generation, Optimisation	1,2,10
Swen Koller	Model Architecture, Optimisation	4,5,9
Matthew Wong	iPhone App, Model Architecture	4,8
Max Baylis	Testing and Continuous Integration, Rendering	3

Table 7: Division of work summary

- Kiyo was responsible for overall management of the project, communication with Ocade engineer team. In terms of the development, he was mainly responsible for the development of the evaluation script (Tensorboard).
- Pavel was a document editor and Scrum master. He developed the Scene Library and was in charge of the design and implementation of the integration of individual modules into a single pipeline.
- Ong contributed the development of Blender API, including the design of the architecture of the rendering engine, and the definition of algorithms to generate random scenes.
- Swen worked on the Keras high-level abstraction layer for training our CNN and on the optimisation script.
- Matthew developed the iPhone app as well as the Flask web server and API for the server-side Neural Network implementation. He also worked on developing the model architecture in Keras, and on training and optimising the network.
- Max oversaw the whole testing strategy as well as continuous integration in GitLab CI, contributed to the evaluation of 3D capture methods and development of the rendering component.

8 Final Product

8.1 Deliverables

8.1.1 Integrated Pipeline Software

The following software was developed in the course of this project:

- First, an **object-oriented custom wrapper for Blender**.
- Second, an **object-oriented wrapper for the Keras deep learning backend**.
- Third, an **evaluation suite** based on the Google TensorBoard package.
- Fourth, a **deployment solution** was provided in the form of a mobile app running a image classification service. (Detailed in Section 8.1.3).

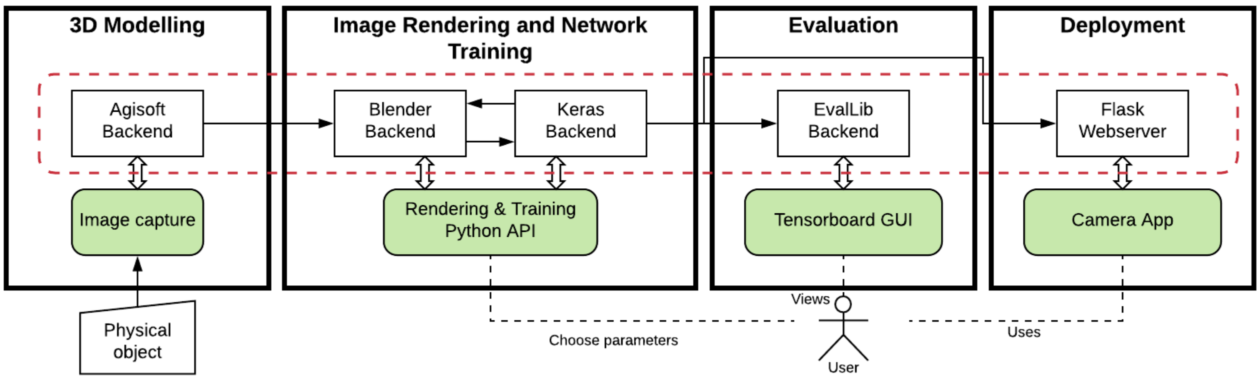


Figure 14: User diagram for training pipeline

Figure 14 illustrates the user interaction with our pipeline. The main components of the pipeline (from the data generation through to evaluation stage) are shown, with the addition of our chosen deployment method (the mobile app). User interfaces are shown in green boxes, and the underlying components supporting these are shown within the red boundary, with arrows showing the dataflow between the components.

After providing the required physical object to the data generation block, the user can control the rendering and network training processes via a script that calls the image rendering and network training API. The result of this is a trained model that can be automatically deployed to our mobile app and evaluation suite, both of which provide graphical user interfaces (Figure16 and 15).

Combined, these components and their integration match the core requirements for this project as outlined in Table 2 in the specifications.

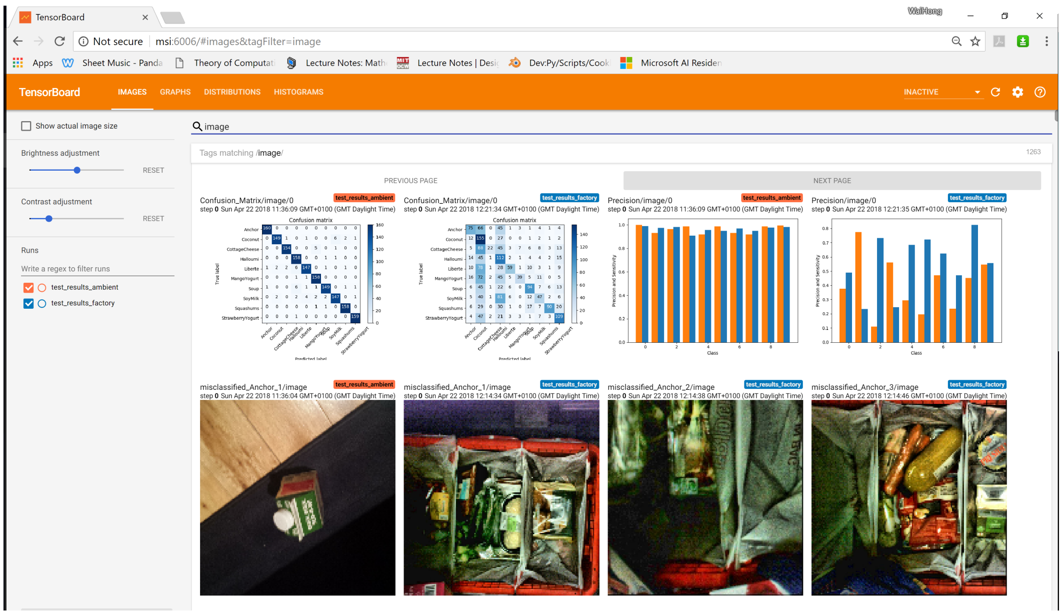


Figure 15: Screenshots of customised implementation of confusion matrices (top left) and precision/recall histograms (top right) displayed on the Tensorboard visualisation tool, with a preview of misclassified images (bottom)

8.1.2 Trained Model

We also produced a trained CNN that functions as an image classifier. The classifier trained on the rendered images (using the InceptionV3 architecture) achieves a maximum of 96% accuracy on the test set for the 10 product classes, significantly outperforming the 60% benchmark presented by Ocado (see Table 1). Section 8.4.2 provides more information about how this result was produced.

Architecture	Accuracy %	Average Precision %	Average Recall %
InceptionV3	96.19	96.24	96.18

Table 8: Results for Final Trained Model

Table 8 shows the accuracy, average precision and recall of our model on the general environment test set.

8.1.3 iPhone App and Web Server API

In addition to the core requirements, we also developed an iPhone app (Figure 16) as an extension, giving users a GUI through which they could use our Neural Network. The app is fully functional and can be installed on any iPhone running iOS 11. It could also be further extended and customised as necessary (for example, Ocado could add a "Order from Ocado" button allowing users to order the identified product from the Ocado store).

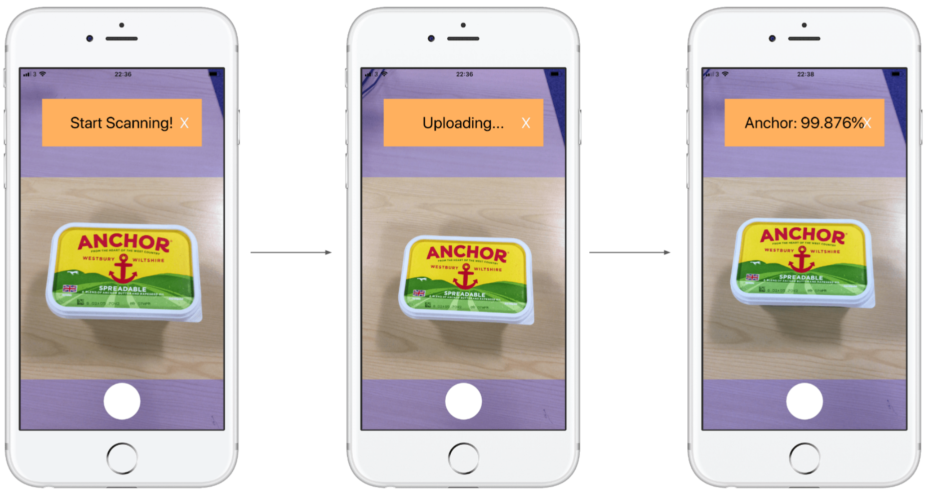


Figure 16: iPhone app

8.2 Unimplemented Extensions

Three of the initially specified potential extensions were not implemented in the course of this project: Extension to more than 10 classes, Generative Adversarial Networks and hierarchical classes (See the Table 13). The reason for this is that the key goal of this work was to demonstrate that the '3D objects to real world' recognition approach can perform well. We therefore decided to replace these initial extensions with other extensions (See Section 9) that were more relevant to this goal. Nonetheless, acquiring more than 10 classes would however be the logical next step in order to evaluate the scalability of the approach.

8.3 Product Evaluation

In this section, we evaluate the final product with respect to the (internal) specifications outlined in Section 3.1. For each specification, it is stated whether or not the specification is satisfied, how well the product does this, and any limitations and/or area for improvements that the product might have.

8.3.1 Essential Specification Satisfaction

Specification 1 : This has been achieved fully via the Data Generation pipline which allows the capture of high-fidelity models of scans using photogrammetry. Limitations that still exist include the inability to scan transparent objects (also see section 8.5).

Specification 2 : The RandomLib package of the Image Rendering suite enables generation of a large range of varied backgrounds. It performs as expected.

Specification 3 : The image rendering pipeline satisfies this requirement fully. This process has been fully automated and made simple to control due to the integration of the rendering engine with a high-level API. Limitations include extended run times, which can be mitigated by the use of a distributed approach to rendering.

Specification 4 : The Tensorflow-based training approach was exchanged for a Keras-based training library, due to its ease of use. This ease of use is further enhanced by the fact that we have developed a high-level API to access and control the training procedure.

Specification 5 : A final test accuracy of 96% on a set of general environment images proves that this has been fulfilled. A detailed analysis is provided in Section 8.4.2. A limitation that still exists with our approach is that the classifier is not robust enough to achieve similar accuracy under more challenging circumstances (Ocado warehouse images).

Specification 6 & 7 : The development of the evaluation section of the pipeline fulfills this specification fully. All metrics are logged, plotted and presented in a user-friendly Tensorboard GUI. These have proven to be very informative and useful especially for debugging and learning process evaluation. More work could be done to include more interactive plots.

8.3.2 Non-essential Specification Satisfaction

Specification 8 : Our implementation of a Flask service on a webserver that interfaces with the client camera iPhone app fulfils this specification. The final score and the classification are both reported to the user.

Specification 9 : The development of an optimisation script, which interfaces our rendering and training libraries using a 3rd party Bayesian Optimisation library[8] satisfies this specification. The script can be used to optimise a classification accuracy measure (loss/accuracy/precision) by exploring the rendering and training parameter space. See Section 9 for more information. A limitation of this system is the long runtimes of experiments that has hindered our group from using the optimisation routine to find globally optimal parameters.

Specification 10 : A region-based CNN was successfully trained and its performance evaluated. The average recall recorded (based on the top detection, provided it exceeds a certain confidence threshold) was 63%. Upon closer inspection, detection and localization of the products was mostly accurate, but classification falls short. More work should go into this to optimise its performance, as it shows great promise.

[8] https://github.com/fmfn/BayesianOptimization

8.4 Machine Learning Research and Results

The developed software served as a basis for our machine learning research. The aim of these experiments was to evaluate the feasibility of classifying real world grocery images using a classifier trained on rendered images from 3D models. The images are generated in a variety of random poses, scale, lighting and backgrounds. In particular, the aim was to evaluate the performance of this approach both on the Ocado warehouse environment as well as the generalisation performance of such an approach.

8.4.1 Experimental Methodology

The general steps for conducting experiments involved generating rendered synthetic images, training a CNN classifier, and evaluating the classification using our evaluation tool.

Rendering

Image rendering involved defining distributions for scene parameters including camera locations and lighting conditions. The camera location was defined to be evenly distributed around a ring in a spherical coordinate system. An illustration of this distribution is shown in figure 17. The reason for choosing this distribution was because these locations correspond to common viewpoints of hand-held groceries.

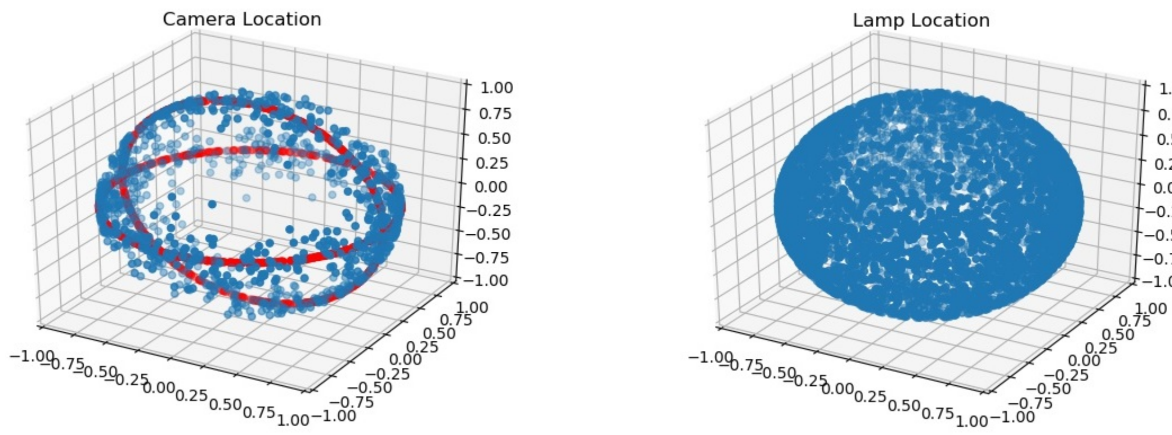


Figure 17: (Left) The rings on the shell (red rings) around which random camera locations (normalized) are sampled from (blue points). (Right) The uniform distribution of lamp locations.

The lamp locations were distributed evenly on a sphere. Additionally, the camera-subject distance and lamp energy were distributed according to a truncated normal distributions with a fixed mean and standard deviation. The number of lamps were also sampled on a uniform discrete distribution to simulate multiple light sources. Detailed description of the distributions used and their respective parameters are provided in Appendix E. Our results are based on a training set which consists of 100,000 training images (10,000 per class) with manually chosen values for the distributions of rendering parameters. Figure 18 shows a number of example images of this training set.

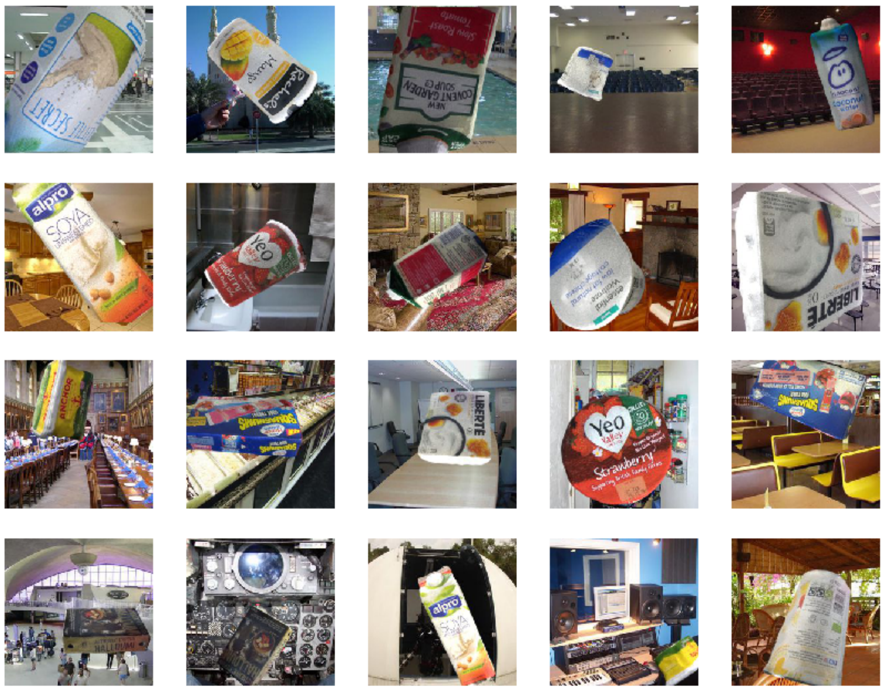


Figure 18: Example rendered images for training

Network

For network training, as described in Section 5.3.3, three different CNN architectures were tested. These were Google's InceptionV3 [7], the Residual Network (ResNet-50) [10] architecture, and the VGG-16 [9] architecture.

A FC layer with 1024 hidden nodes and 10 output nodes for each class was defined, and a standard stochastic gradient descent optimizer with momentum was used. For all of the above architectures, the only parameter that was manipulated was the number of trained convolutional layers (the FC layers are always trained), ranging from zero layers to all layers. The weights for the convolutional layers were initialized based on those trained using the ImageNet dataset [8]. All other parameters were kept constant as shown in Table 9.

Number of images (per class)	
Training	10,000
Validation (Rendered)	200
Validation (Real)	80

Learning rate (Inception V3)	Figure 19
Learning rate (VGG16, ResNet50)	0.0001
Image input size (px,px)	(224,224)
Batch size	64
Number of fully-connected layers	2
Hidden layer size	1024
Optimizer	SGD

Table 9: Constant variables for network training

Additionally, for the InceptionV3 a grid-search was carried out to fine-tine the learning rate used (shown in figure 19).

Test Data

To evaluate the classifier's generalization performance, we acquired our own test and validation set. Figure 20 displays some examples of our test set which we acquired in a variety of locations. The set contains 1600 images of 10 classes (160 images per class) from a variety of perspectives, at different distances, in different lighting conditions and with and without occlusion. These images were acquired with a number of different devices, including smartphones, DSLR cameras, and digital cameras.

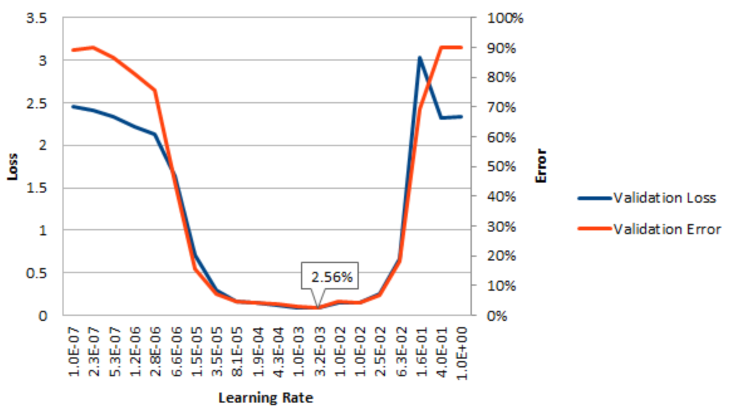


Figure 19: InceptionV3 with all convolutional layers trained: Learning Rate vs Loss and Validation Accuracy (on real images)

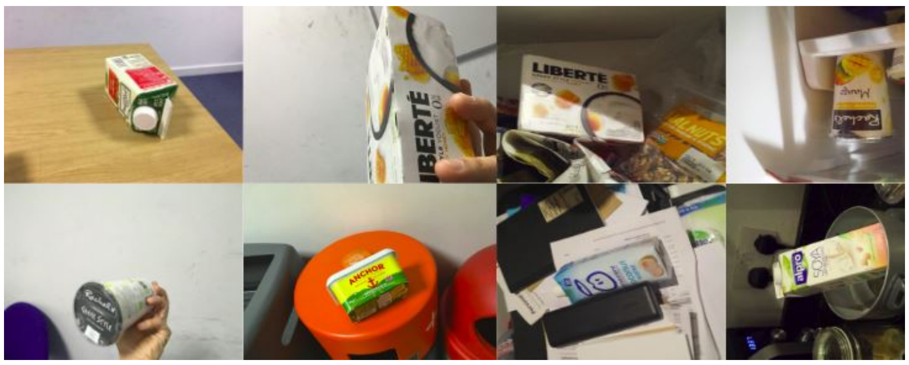


Figure 20: Proprietary General Environment Test Set

System Performance

The durations for generating training images and training each respective network architecture is shown in Table 10. The training set used in our experiments took 9 hours to generate on our Titan X machine.

Rendering (100K images)		
Samples	Resolution (px)	Runtime (hours)
64	224	4
128	224	6
64	300	5.5
128	300	9

Training (3.75K steps @ batch size 64)	
Architecture	Runtime (min)
VGG-16	86
InceptionV3	82
ResNet-50	83

Table 10: Performance: The runtimes for rendering and network training. Note that rendering runtimes depend heavily on rendering settings as shown below. Hardware used was an Nvidia GTX Titan X GPU.

8.4.2 Results and Discussion

A first attempt involved training an InceptionV3 CNN with no trained convolutional layers. A problem observed was the stalling of losses associated with the validation data (which comprised of a rendered set of images, as well as the real images), shown in figure 21, when the training loss kept decreasing, but also stalling at a relatively high value. The final validation accuracy after 19.8K steps at a batch size of 64 was 58%, and training accuracy was 70%.

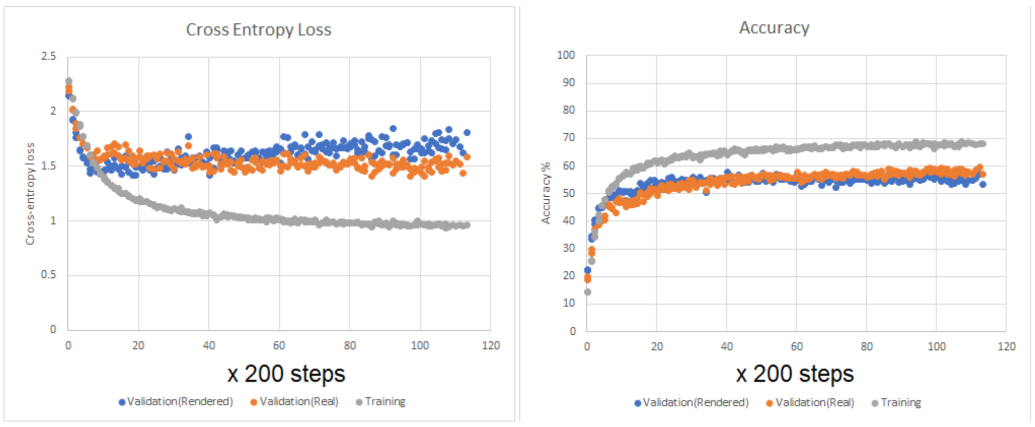


Figure 21: Convergence plots for InceptionV3 network with zero unfrozen layers

Further attempts saw progressively more layers set to trainable. These yielded much better validation accuracies, and ensured that training accuracy always converged to 100%. This trend is shown in Figure 22.

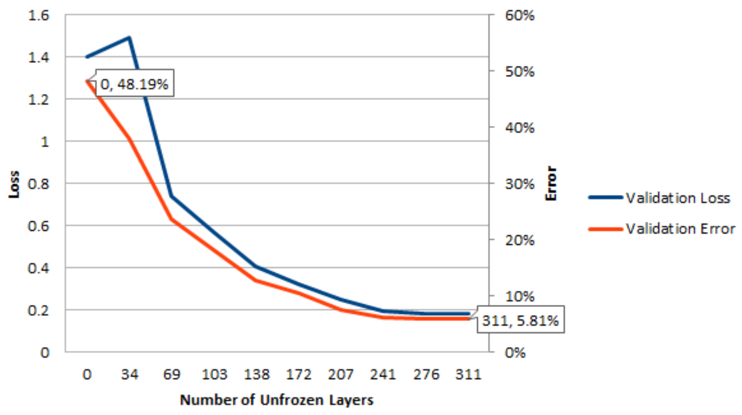
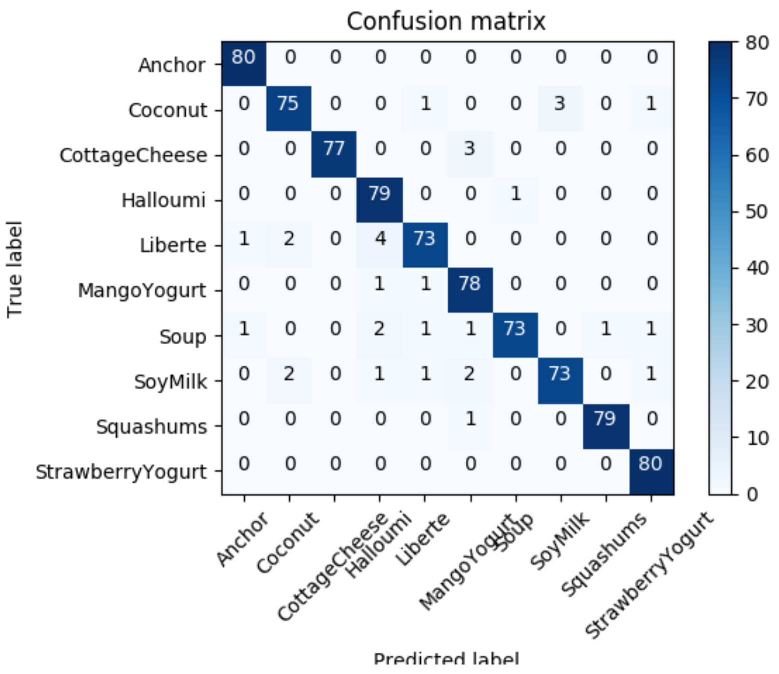


Figure 22: Number of Unfrozen Layers vs. Validation Accuracy and Loss (before tuning learning rate) on InceptionV3

Finally, a test accuracy of 96% on our proprietary test set was achieved using the InceptionV3 architecture with the optimized learning rate and all convolutional layers being retrained. The confusion matrix for this network is shown in figure 23. Appendix F also contains a histogram of confidences per class for this test set.



	Anchor	Coconut	CottageCheese	Halloumi	Liberte	MangoYogurt	Soup	SoyMilk	Squashums	StrawberryYogurt
Anchor	80	0	0	0	0	0	0	0	0	0
Coconut	0	75	0	0	1	0	0	3	0	1
CottageCheese	0	0	77	0	0	3	0	0	0	0
Halloumi	0	0	0	79	0	0	1	0	0	0
Liberte	1	2	0	4	73	0	0	0	0	0
MangoYogurt	0	0	0	1	1	78	0	0	0	0
Soup	1	0	0	2	1	1	73	0	1	1
SoyMilk	0	2	0	1	1	2	0	73	0	1
Squashums	0	0	0	0	0	1	0	0	79	0
StrawberryYogurt	0	0	0	0	0	0	0	0	0	80

Figure 23: Confusion Matrix using InceptionV3 on proprietary general environment test set

Experimentation with different CNN architectures yielded favorable and consistent results, consistently reporting accuracies above 80%. These results are shown in Table 11.

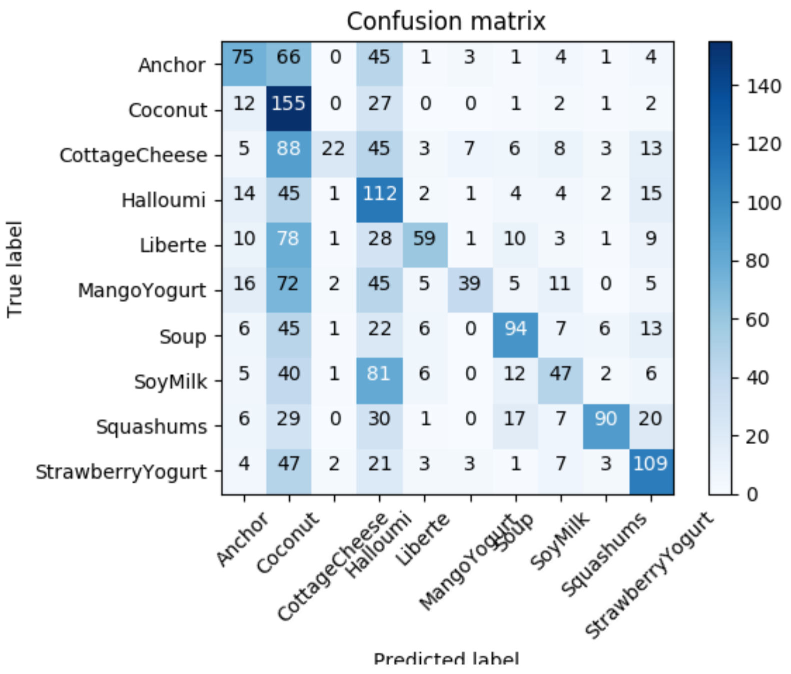
Architecture	Accuracy %	Average Precision %	Average Recall %
VGG16	84.21	85.05	84.18
ResNet50	95.81	95.81	95.86
InceptionV3	96.19	96.24	96.18

Table 11: Summary of testing results for different architectures

These results prove that a classifier can be successfully trained entirely based on 3D rendered objects acquired using photogrammetry. To our knowledge this the first time photogrammetry-based 3D models were used to train a CNN. [9]

However in terms of accuracy on the Ocado warehouse dataset, our method's result remained below the benchmark. This dataset differs in the following ways, previously discussed in section 2: multiple learned classes in an image, severe occlusion, light bias, and empty images. The performance on the warehouse dataset was 40% and stays significantly below the client warehouse environment benchmark. The results are partially explained by the fact that the rendered training set was optimized for a general environment. However it also shows that the 3D modelling approach based on photogrammetry and without depth information, as presented in this report, might introduce too much noise to the features for it to produce a classifier which performs well under adverse conditions (such as significant occlusion). The confusion matrix for warehouse images is shown in figure 24.

[9]see [2] for viewpoint estimation use case of CAD-based rendered images to train a CNN, see [4] for CAD-based object recognition using training data based on 3D rendered images



	Anchor	Coconut	CottageCheese	Halloumi	Liberte	MangoYogurt	Soup	SoyMilk	Squashums	StrawberryYogurt
Anchor	75	66	0	45	1	3	1	4	1	4
Coconut	12	155	0	27	0	0	1	2	1	2
CottageCheese	5	88	22	45	3	7	6	8	3	13
Halloumi	14	45	1	112	2	1	4	4	2	15
Liberte	10	78	1	28	59	1	10	3	1	9
MangoYogurt	16	72	2	45	5	39	5	11	0	5
Soup	6	45	1	22	6	0	94	7	6	13
SoyMilk	5	40	1	81	6	0	12	47	2	6
Squashums	6	29	0	30	1	0	17	7	90	20
StrawberryYogurt	4	47	2	21	3	3	1	7	3	109

Figure 24: Confusion Matrix using InceptionV3 on warehouse images

Initial experiments show that the accuracy for warehouse could be increased by at least 6% when introducing occlusion into the training image generation. Further research is necessary to determine the impact of occlusion in training image generation on performance in the warehouse environment.

8.5 Limitations

Throughout the research conducted into this approach, a number of limitations became apparent.

First, the 3D Modelling based on photogrammetry creates noise in reconstruction in the case of transparent features and in the case of large unicolor areas. This noise reduces classification performance when combined with a challenging environment and likely also when scaled to more classes. This could be mitigated by combining photogrammetry with other information sources. For example, there are cost efficient products for the fast acquisition of 3D objects such as Occipital's Structure product. It uses an infrared iPad mount to combine information from a typical camera with information from an infrared sensor to create more accurate 3D models. Similarly, there is an industrial-grade RGBD scanners which also combine depth and colour information to reconstruct 3D objects, offered by companies such as Ametek.

Second, the process of acquiring and 3D models and preparing them for rendering required roughly 15 minutes of manual work per product. An industry-ready solution for classification based on 3D scanning will require a more sophisticated set for acquiring the product photos in order to keep manual effort at a minimum. This includes investments into appropriate hardware such as the above mentioned industrial grade 3D scanners as well customising the set up for automation for this use case.

9 Exploratory Efforts and Further Research

This section covers extensions which are currently work in progress and are meant as a starting point for further research. We present our preliminary findings which show great promise.

9.1 Object Detection with Region-CNNs

Our approach opens up the opportunity to train object detectors and segmentation algorithms given that our pipeline can produce pixel-level annotations of products. This is possible as the placement of the object of interest in the image is fully under our control, and can be logged automatically. For this application, we have logged the object bounding box. This information can then be compared with the detector estimated bounding box. This provides a huge advantage as no manual pixel annotation is necessary as is the case with currently available datasets, eg. the Microsoft Coco dataset [12]. First experiments using RCNNs on our dataset show strong results when performing detection on our general environment test set.

The chosen architecture for this task is the RetinaNet architecture[13]. This is a one-stage architecture that runs detection and classification over a dense sampling of possible sub-regions (or 'Anchors') of an image. The authors have claimed that it outperforms most state-of-the-art one-stage detectors in terms of accuracy, and runs faster than two-stage detectors (such as the Fast-RCNN architecture [14]). A 'detection as classification' (DAC) accuracy metric was calculated by using the following formula:

$$DAC = \frac{\sum_i^n TP_i}{n} \tag{1}$$

Where the quantity TP_i is summed over every image i, and is calculated:

$$TP_i = \begin{cases} 1 & \text{if the top 3 detections for image i contain the correct class} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

The same proprietary general environment test set specified in Section 8.4.1 was used to test the accuracy of this learning task, when the same set of rendered training images were used.

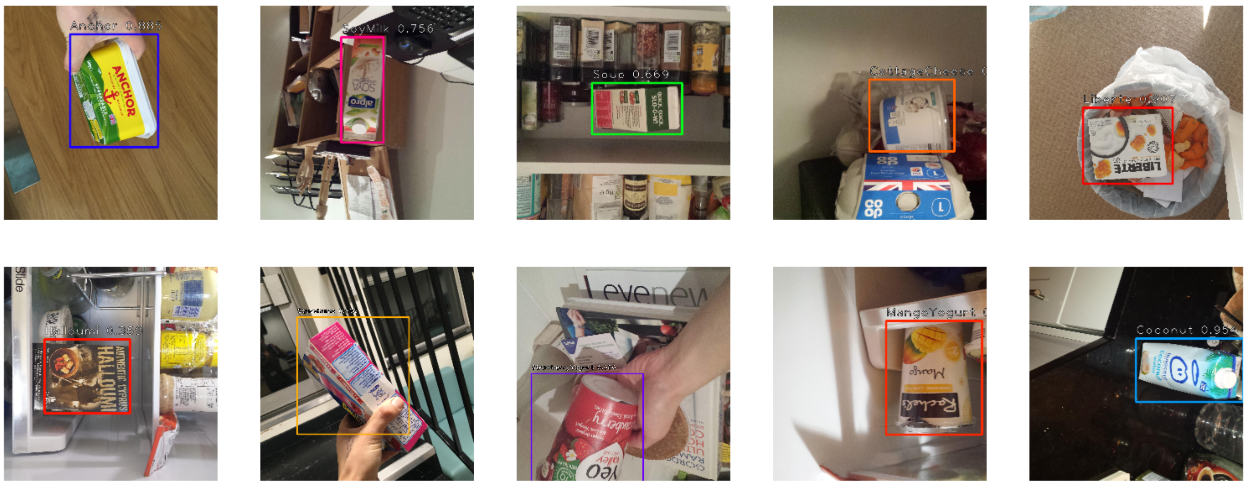


Figure 25: Detection bounding boxes and confidence scores for classification

The reported DAC accuracy for the test set was 64%. Upon closer inspection of the test results, it was discovered that the detection bounding boxes were mostly accurately calculated. However, the main factor driving the score down was incorrect classifications. This classification loss was less pronounced in the rendered image validation dataset, giving a final accuracy of 75%.

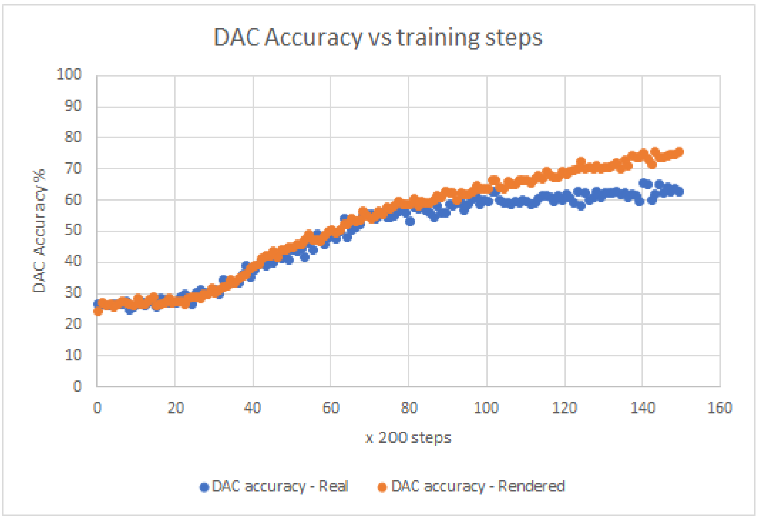


Figure 26: Real and rendered image validation scores for product detection, logged every 200 training steps. The DAC accuracy for real images deviates from the rendered accuracy at around 20K steps.

The logged validation accuracy for both real and rendered data is shown in Figure 26, and indicates a deviation between how the network perceives real images of objects, and rendered images. This is a surprising finding, given near-perfect performance on the classification task (Section 8.4.2). This shows that our current set of rendering parameters might not be as robust as previously thought, and can be dependent on the learning task. We feel that some optmization of the rendering pipeline can be done in order to optimize against the detection task, potentially making it more robust against a larger variety of learning tasks.

9.2 Bayesian Optmisation

The presented setup contains a lot of parameters for both rendering and training of the model which can be optimised. For this purpose, we built a Bayesian Optimisation script to tune hyperparameters of both rendering and training. Using this setup, suitable parameters for performance on the warehouse dataset can be explored. Optimal parameters can be found for small subset of the classes (for example the presented 10 classes). It is expected that these parameters will be then transferable when scaling the rendering to any number of classes, while decreasing the time necessary for the optimisation.

9.3 Further Research

Both the 3D-acquisition scalability challenge and the noise resulting from photogrammetry justify further investigation. In particular, different methods for 3D model acquisition could be explored such as the above mentioned fusion of photogrammetry with depth information from infrared sensors.

Another area of investigation is the class-scalability of this method, i.e. whether classification accuracy declines significantly when introducing more classes in training. This is a particular concern for this method since the photogrammetry approach introduces noise to the features on which the CNN is trained. This can potentially be mitigated by using depth information for model acquisition, as outlined above.

10 Conclusion

Early on in our work it became apparent that the provided dataset will not allow a classifier to extract the distinguishing features of the classes well. At the same time, groceries possess the unique property of low intra-class variation. As such, this justified approach to the problem which attempted to capture all features of a class exhaustively. This could either be done with non-biased data acquisition 'in the field', similar to what Ocado is doing in their warehouse or it can be done in a 'sterile' environment, where a product is photographed in a large number of poses. 3D modelling offers a third approach to the problem which is more scalable than the 'sterile' acquisition from hundreds or thousands of perspectives. From 40-60 images per product, an unlimited amount of training data can be generated.

In the course of this project, we developed an original pipeline that goes from data acquisition, over data generation to training a CNN classifier. This is a novel combination of several tools including photogrammetry for 3D scanning and the use of a graphics engine for training data generation. The 96% accuracy achieved on the general environment test set demonstrates that photogrammetry-based 3D models can be successfully used to train an accurate classifier for real-world images. To our knowledge this the first time photogrammetry-based 3D models were used to train a CNN.

This work may serve as a basis to build a classifier which can be deployed on a large scale grocery dataset for any particular use case. Next steps for such deployment are acquiring more 3D models and optimising the rendering parameters for the particular environment. This can be done using the Bayesian Optimisation script provided. Automated Optimisation in the scope of this project was limited given the amount of GPU hours required to do such a parameter search for a search space that spans over seven rendering dimensions.

The exploratory work with region-based CNNs also demonstrates a unique favorable aspect of generating training data based on 3D models. For every traininig image, there is pixel-level annotation of the object available. This facilitates the training of real-world object detector which has advantages over a simple detector in certain use cases. Initial results show that it is feasible to train such a network based on our image generation approach.

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] 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.

[4] K. Sarkar, K. Varanasi, and D. Stricker, "Trained 3D models for CNN based object recognition," 2017.

[5] Microsoft, "Kinect for Windows 1.7," vol. 188670, pp. 1–9, 2017.

[6] J. Xiao, J. Hays, K. A. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 3485–3492, June 2010.

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," sep 2014.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[9] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," sep 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," dec 2015.

[11] A. S. Foundation, "Alamofire - GitHub," 2018.

[12] T.-Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, and P. Dollár, "Microsoft COCO: Common Objects in Context," may 2014.

[13] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017.

[14] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

Appendix A Original Specifications and Details of Changes

No	Dependency	Category	Description	Type	Estimate
1	N/A	Data Generation	Optically scan physical products with a Kinect, and use a 3D reconstruction program (Microsoft 3D scan) to generate 3D images in OBJ format.	Functional Essential	01/02/18
2	N/A	Image Rendering	Create a database of realistic background images in jpg or png format.	Functional Essential	01/02/18
3	2	Image Rendering	Use the 3D model obj file with Blender API to generate images of the object from different angles. By merging the images with the database of backgrounds, generate 2D jpg training images.	Functional Essential	01/02/18
4	1-3	Training	Train Tensorflow-based InceptionV3 CNN model using the training images we generated	Functional Essential	07/02/18
5	4	Evaluation Optimisation	The trained model should be able to classify 10 products with accuracy higher than Ocado's baseline (60%).	Functional Essential	15/03/18
6	4	Evaluation Optimisation	Evaluation results must be available on Tensorboard.	Non-Functional Essential	07/02/18
7	1-6	Image Rendering	Users are able to upload images, and get the result of classification through GUI.	Functional Non-Essential	30/03/18

Table 12: Original Internal Specifications (Essential)

No	Description
1	Extend the model to recognise more than initial 10 products
2	Introduce a class hierarchy (e.g meat) to enable broader classification of very similar products (e.g chicken thigh and chicken legs)
3	Investigate the use of generative adversarial networks (GAN). A Wasserstein
4	Develop a GUI-based tool to enable live demonstration of the classifier.

Table 13: Original Interal Specifications (Non-Essential)

No	Dependency	Category	Description	Type	Estimate	Completed
1	N/A	Data Generation	Optically scan physical products with camera (iPhone and DSLR), and use a 3D reconstruction program (Agisoft) to generate 3D images in OBJ format.	F	08/02/18	Yes
2	N/A	Image Rendering	Create a database of realistic, random colour mesh and plain colour background images in jpg or png format.	F	01/02/18	Yes
3	2	Image Rendering	Use the 3D model obj file with Blender API to generate images of the object from different angles that show the unobstructed object. By merging the images with the database of backgrounds, generate 2D jpg training images.	F	01/02/18	Yes
4	1-3	Training	Train Tensorflow-based InceptionV3 Convolutional Neural Network (CNN) model using the training images we generated, as well as a Keras-based InceptionV3 model	F	07/02/18	Yes
5	4	Evaluation Optimisation	The trained model should be able to classify 10 products with accuracy higher than Ocado's baseline ($\tilde{6}0\%$).	F	15/03/18	Yes
6	4	Evaluation Optimisation	Evaluation results must be available on Tensorboard.	NF	07/02/18	Yes
7	6	Evaluation Optimisation	Results of experiments must be collected with experiment parameters on Tensorboard.	NF	01/04/18	Yes

Table 14: Changes to Original Specifications (Essential), changes to original specifications are shown in red.

No	Dependency	Category	Description	Type	Estimate	Completed
8	1-6	Image Rendering	Users are able to upload images, and get the result of classification through GUI. This will be implemented within an iPhone app.	F	30/03/18	Yes
9	4	Training	Baysian optimisation is conducted to fine-tune parameters in both model pipeline as well as rendering.	F	22/05/18	prototype
10	4	Training	Regional CNN is added to further improve the accracy of the classification.	F	22/05/18	prototype

Table 15: Changes to Original Spoecifications (Non-essential), changes to original specifications are shown in red.

A.1 Details of Changes to Original Specifications

Changes made to each specification over the course of the project are described below. Full details on the design and implementation of each of the specifications can be found in Sections 4 and 5 respectively.

A.1.1 Data Generation (Spec. 1)

The completion of this task was delayed by one week due to significant issues with the initial data generation method; using a Kinect device to capture 3D models proved infeasible as the Kinect software was unable to generate 3D models of a high enough quality. After trying multiple different alternatives, the specification was updated to allow for the use of specialised 3D modelling software (Agisoft), which proved to be successful.

A.1.2 Image Rendering (Specs. 2, 3)

Minor challenges were initially encountered when we discovered that real life background images appeared to be under-performing in tests of our neural network. We hypothesised that this was due to the nature of the background images used, specifically the different light gradient of the background as compared to the object. To test this hypothesis we updated our specification to produce a new set random coloured mesh backgrounds and a new set of plain coloured backgrounds, in addition to our original background set. The source of the problem was later discovered to be in the training process so the realistic background remained the main source of backgrounds.

Separately, the 3D scanning process also introduced another issue - as the objects had to be placed on a desk to be scanned, they were not scanned from the bottom. The 3D model showed a black patch instead. While this could be mitigated by producing two 3D models, one scanned with the product in its normal orientation and the other with the product upside-down, the product pose generation process had to be altered so that images showing the obscured side were not generated. Our specification was updated to reflect this new requirement.

A.1.3 Model Training (Specs. 4, 5)

After successfully completing the training of a two class InceptionV3 model in Tensorflow, it was decided to update the specification to use Keras, rather than Tensorflow, for all remaining work. Keras is a high-level deep learning framework with a Tensorflow backend which we concluded it was a better fit for the needs of our project. While Tensorflow provided a high degree of control and customisability, its complexity also slowed development down. Updating the specification to use Keras allowed us to significantly speed up future development, and was a change that did indeed pay off. Nonetheless, as some intial work was needed to transition from Tensorflow to Keras, the estimated delivery dates for these specifications were pushed back slightly.

A.1.4 Evaluation and Hyperparameter Tuning (Specs. 6, 7)

We were able to fulfil our initial specification for the evaluation and optimisation section of the pipeline. The software was able to create, export and visualise custom plots and metrics onto the Tensorboard tool. However, we then realised that in order to successfully optimise our pipeline, it was necessary that evaluation and optimisation be carried out over the course of multiple runs, and collated and collectively visualised in one central location, leading us to add a new specification detailing this (specification 7).

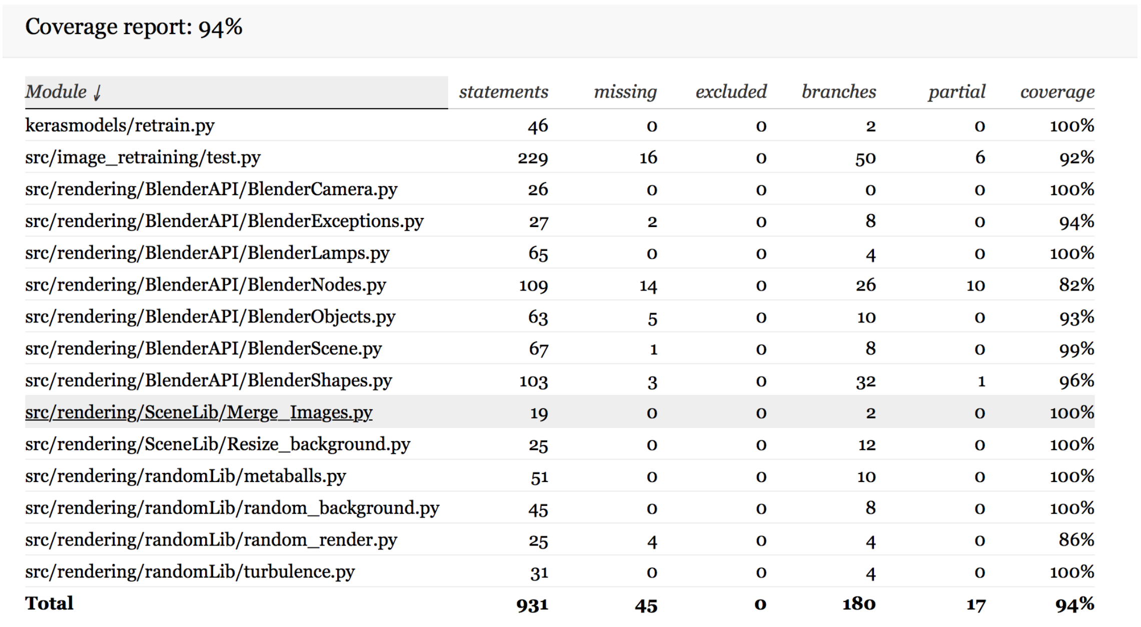
A.1.5 Final Product (Spec. 8)

The goal of our first extension was to implement a GUI which allowed users to upload images and receive classification results in an intuitive way. We decided to implement this in the form of an iPhone app connected to an API running on a Flask web server, which would allow us to illustrate a potential practical use case of our deep learning model.

A.1.6 Further Optimisation (Spec. 9, 10)

Once our essential specifications were completed, we decided to replace the rest of our original extensions with two new extensions designed to maximise the accuracy of our deep learning model. First, as the pipeline had a large number of hyper-parameters that needed to be tuned, Bayesian optimisation was a logical step to explore in order to improve accuracy of our classifier. Second, the 3D rendering based approach yielded pixel-level annotations for images, which allowed us to explore region-based CNNs for object detection. These new extensions were more closely related to our key objectives, and were thus more beneficial to our project than the other extensions previously considered.

Appendix B Test Coverage Table


Coverage report: 94%

Module ↓	*statements*	*missing*	*excluded*	*branches*	*partial*	*coverage*
kerasmodels/retrain.py	46	0	0	2	0	100%
src/image_retraining/test.py	229	16	0	50	6	92%
src/rendering/BlenderAPI/BlenderCamera.py	26	0	0	0	0	100%
src/rendering/BlenderAPI/BlenderExceptions.py	27	2	0	8	0	94%
src/rendering/BlenderAPI/BlenderLamps.py	65	0	0	4	0	100%
src/rendering/BlenderAPI/BlenderNodes.py	109	14	0	26	10	82%
src/rendering/BlenderAPI/BlenderObjects.py	63	5	0	10	0	93%
src/rendering/BlenderAPI/BlenderScene.py	67	1	0	8	0	99%
src/rendering/BlenderAPI/BlenderShapes.py	103	3	0	32	1	96%
src/rendering/SceneLib/Merge_Images.py	19	0	0	2	0	100%
src/rendering/SceneLib/Resize_background.py	25	0	0	12	0	100%
src/rendering/randomLib/metaballs.py	51	0	0	10	0	100%
src/rendering/randomLib/random_background.py	45	0	0	8	0	100%
src/rendering/randomLib/random_render.py	25	4	0	4	0	86%
src/rendering/randomLib/turbulence.py	31	0	0	4	0	100%
Total	**931**	**45**	**0**	**180**	**17**	**94%**

Figure 27: Coverage.py HTML output

Appendix C Gitlab Continuous Integration Pipeline

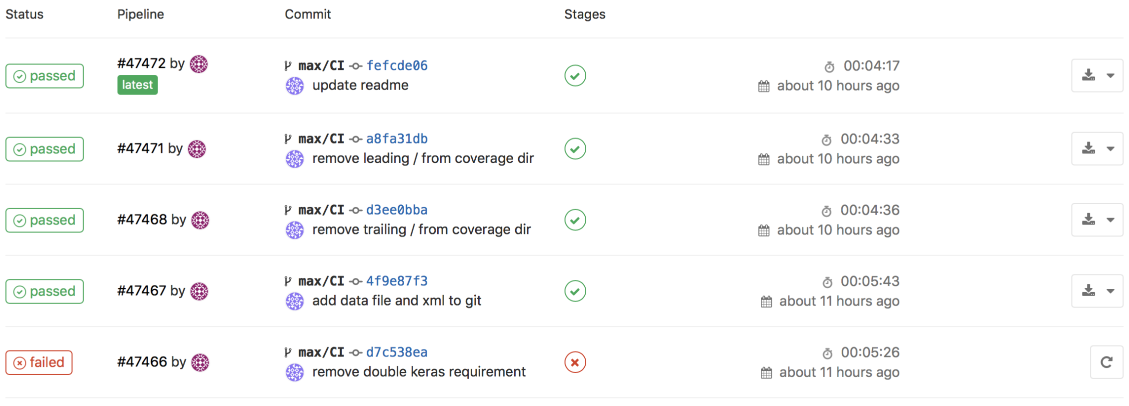
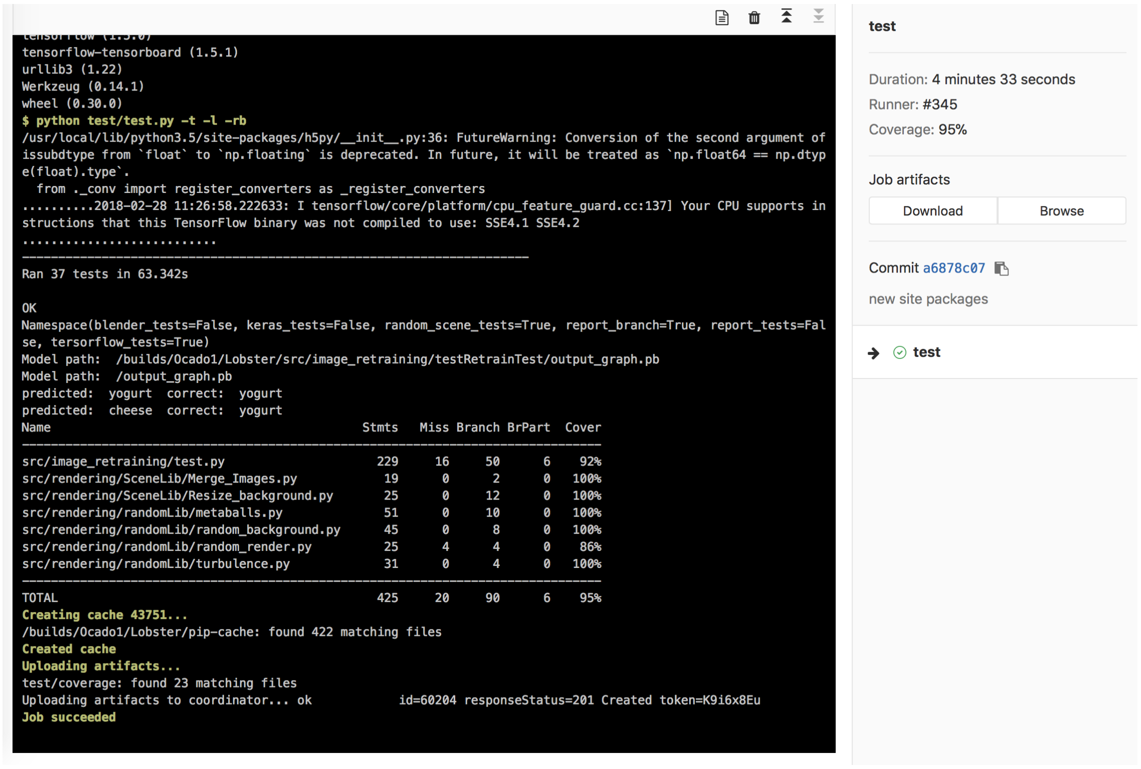

Status	Pipeline	Commit	Stages		
passed	#47472 by latest	max/CI fefcde06 update readme	✓	00:04:17 about 10 hours ago	
passed	#47471 by	max/CI a8fa31db remove leading / from coverage dir	✓	00:04:33 about 10 hours ago	
passed	#47468 by	max/CI d3ee0bba remove trailing / from coverage dir	✓	00:04:36 about 10 hours ago	
passed	#47467 by	max/CI 4f9e87f3 add data file and xml to git	✓	00:05:43 about 11 hours ago	
failed	#47466 by	max/CI d7c538ea remove double keras requirement	✗	00:05:26 about 11 hours ago	

Figure 28: A screenshot from Gitlab's 'Pipelines' section displaying past commits.



```
tensorflow-tensorboard (1.5.1)
urllib3 (1.22)
Werkzeug (0.14.1)
wheel (0.30.0)
$ python test/test.py -t -l -rb
/usr/local/lib/python3.5/site-packages/h5py/__init__.py:36: FutureWarning: Conversion of the second argument of
issubdtype from `float` to `np.floating` is deprecated. In future, it will be treated as `np.float64 == np.dtyp
e(float).type`.
  from ._conv import register_converters as _register_converters
..........2018-02-28 11:26:58.222633: I tensorflow/core/platform/cpu_feature_guard.cc:137] Your CPU supports in
structions that this TensorFlow binary was not compiled to use: SSE4.1 SSE4.2
...........................
----------------------------------------------------------------------
Ran 37 tests in 63.342s

OK
Namespace(blender_tests=False, keras_tests=False, random_scene_tests=True, report_branch=True, report_tests=Fal
se, tersorflow_tests=True)
Model path:  /builds/Ocado1/Lobster/src/image_retraining/testRetrainTest/output_graph.pb
Model path:  /output_graph.pb
predicted:  yogurt  correct:  yogurt
predicted:  cheese  correct:  yogurt
Name                                           Stmts   Miss Branch BrPart  Cover
--------------------------------------------------------------------------------
src/image_retraining/test.py                     229     16     50      6    92%
src/rendering/SceneLib/Merge_Images.py            19      0      2      0   100%
src/rendering/SceneLib/Resize_background.py       25      0     12      0   100%
src/rendering/randomLib/metaballs.py              51      0     10      0   100%
src/rendering/randomLib/random_background.py      45      0      8      0   100%
src/rendering/randomLib/random_render.py          25      4      4      0    86%
src/rendering/randomLib/turbulence.py             31      0      4      0   100%
--------------------------------------------------------------------------------
TOTAL                                            425     20     90      6    95%
Creating cache 43751...
/builds/Ocado1/Lobster/pip-cache: found 422 matching files
Created cache
Uploading artifacts...
test/coverage: found 23 matching files
Uploading artifacts to coordinator... ok            id=60204 responseStatus=201 Created token=K9i6x8Eu
Job succeeded
```

test

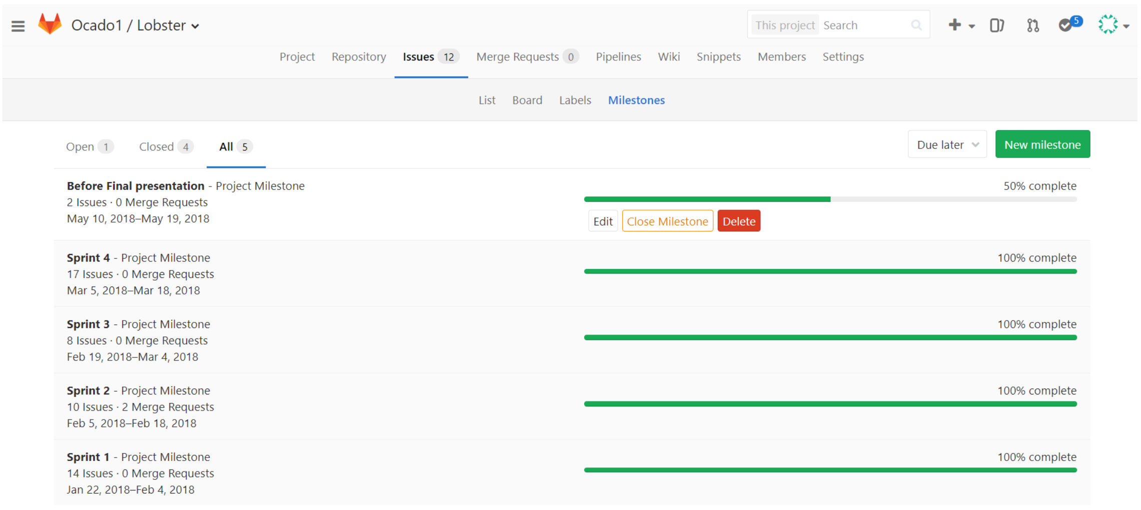
Duration: 4 minutes 33 seconds
Runner: #345
Coverage: 95%

Job artifacts

Commit a6878c07
new site packages

Figure 29: A screenshot from Gtilab CI showing a test runner automatically loading dependencies and running tests on the EvalLib, SceneLib and RandomLib sections.

Appendix D Gitlab issues



Before Final presentation - Project Milestone
2 Issues · 0 Merge Requests
May 10, 2018–May 19, 2018
50% complete

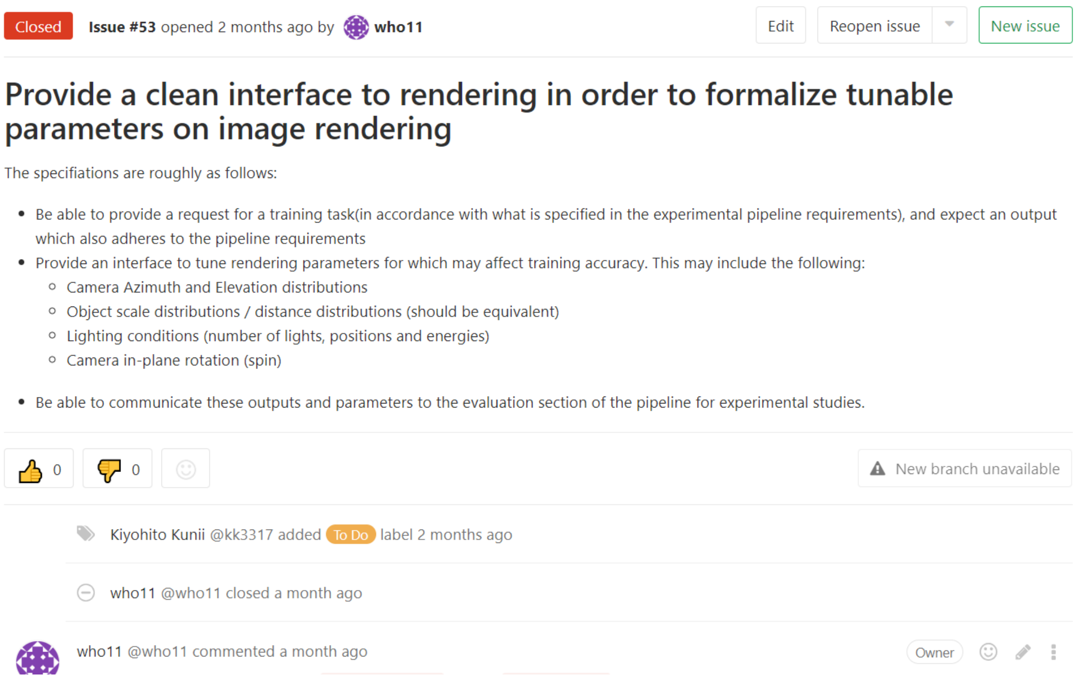
Sprint 4 - Project Milestone
17 Issues · 0 Merge Requests
Mar 5, 2018–Mar 18, 2018
100% complete

Sprint 3 - Project Milestone
8 Issues · 0 Merge Requests
Feb 19, 2018–Mar 4, 2018
100% complete

Sprint 2 - Project Milestone
10 Issues · 2 Merge Requests
Feb 5, 2018–Feb 18, 2018
100% complete

Sprint 1 - Project Milestone
14 Issues · 0 Merge Requests
Jan 22, 2018–Feb 4, 2018
100% complete

Figure 30: A screenshot from Gitlab's Milestone overview, showing our Sprint timeline and progress.


Closed **Issue #53** opened 2 months ago by **who11**

Provide a clean interface to rendering in order to formalize tunable parameters on image rendering

The specifiations are roughly as follows:

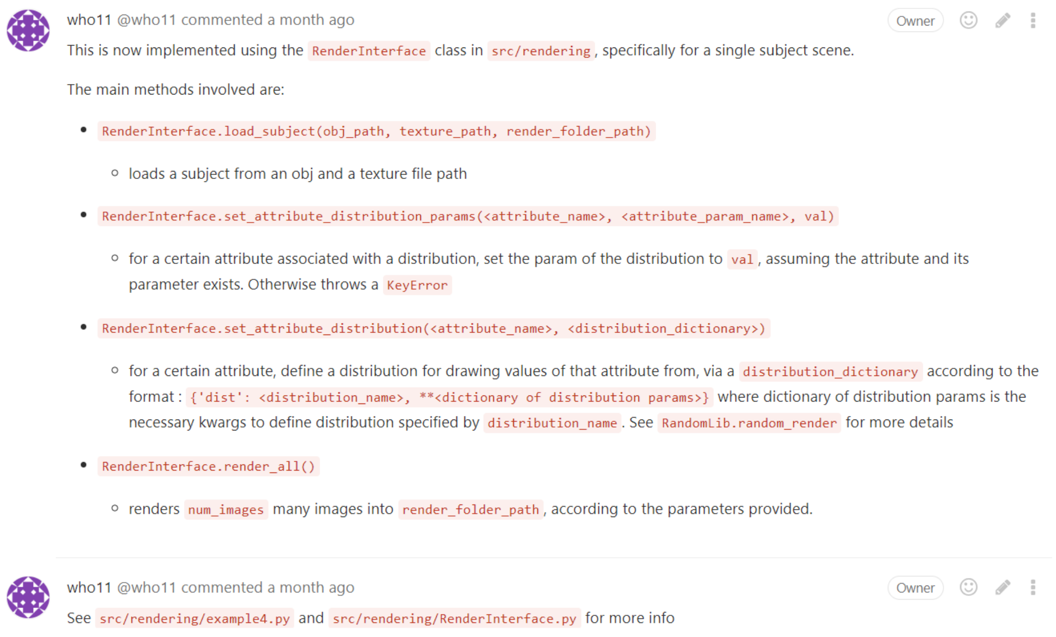
- Be able to provide a request for a training task(in accordance with what is specified in the experimental pipeline requirements), and expect an output which also adheres to the pipeline requirements
- Provide an interface to tune rendering parameters for which may affect training accuracy. This may include the following:
  - Camera Azimuth and Elevation distributions
  - Object scale distributions / distance distributions (should be equivalent)
  - Lighting conditions (number of lights, positions and energies)
  - Camera in-plane rotation (spin)
- Be able to communicate these outputs and parameters to the evaluation section of the pipeline for experimental studies.

Kiyohito Kunii @kk3317 added To Do label 2 months ago

who11 @who11 closed a month ago

who11 @who11 commented a month ago

Figure 31: An example of a typical Gitlab issue, describing a task.


who11 @who11 commented a month ago

This is now implemented using the `RenderInterface` class in `src/rendering`, specifically for a single subject scene.

The main methods involved are:

- `RenderInterface.load_subject(obj_path, texture_path, render_folder_path)`
  - loads a subject from an obj and a texture file path
- `RenderInterface.set_attribute_distribution_params(<attribute_name>, <attribute_param_name>, val)`
  - for a certain attribute associated with a distribution, set the param of the distribution to `val`, assuming the attribute and its parameter exists. Otherwise throws a `KeyError`
- `RenderInterface.set_attribute_distribution(<attribute_name>, <distribution_dictionary>)`
  - for a certain attribute, define a distribution for drawing values of that attribute from, via a `distribution_dictionary` according to the format : `{'dist': <distribution_name>, **<dictionary of distribution params>}` where dictionary of distribution params is the necessary kwargs to define distribution specified by `distribution_name`. See `RandomLib.random_render` for more details
- `RenderInterface.render_all()`
  - renders `num_images` many images into `render_folder_path`, according to the parameters provided.

who11 @who11 commented a month ago

See `src/rendering/example4.py` and `src/rendering/RenderInterface.py` for more info

Figure 32: Gitlab Issue closing comment, explaining what was achieved in this task for further reference.

Appendix E Rendering Detailed Parameters and Calculations

Coordinates for camera location and lamp locations were mainly calculated using spherical coordinates. These are calculated by considering three variables - the azimuth θ, elevation ϕ, and radius ρ. They are related to cartesian coordinates:

$$x = \rho \cos(\theta) \sin(\phi)$$
$$y = \rho \sin(\theta) \sin(\phi)$$
$$z = \rho \cos(\phi)$$

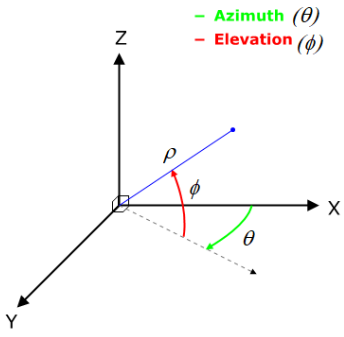


Figure 33: Illustration of the variables in spherical coordinates, and their relation to cartesian coordinates.

To generate a random distribution of locations, one has to define an appropriate distribution for each variable. The chosen distributions were as follows:

$$\rho \sim \mathcal{T}(\mu_\rho, \sigma_\rho, a_\rho, b_\rho)$$
$$\phi \sim \mathcal{T}(0, \sigma_\phi, -\pi/2, \pi/2)$$
$$\theta \sim \mathcal{U}(0, 2\pi)$$

Where $\mathcal{T}(\mu, \sigma, a, b)$ is the truncated normal distribution, with mean μ and standard deviation σ. a and b define the limits of the distribution, for which the PDF function is zero outside the limits. So if $X \sim \mathcal{T}(\mu, \sigma, a, b)$, then $P(X = x|a \leq x \leq b) = \mathcal{N}(\mu, \sigma)$. This set of variables define a distribution around a ring in the X-Y plane (with a normal Z), and the width of the ring can be controlled by specifying σ_ϕ. If $\mathbf{x} = (x, y, z)$ is drawn from the distribution, one can 'flip' the ring to have normal aligned with the X axis by doing:

$$\mathbf{x}' = \mathbf{R_y}(\pi/2)\mathbf{x}$$

Where $\mathbf{R_y}(\pi/2)$ is the rotation matrix that rotates the point about the axis y, $\pi/2$ radians. This way points can be distributed around multiple rings (specified by their normals). For the purposes of this project, camera was distributed about the rings with the Y and Z axes as normals.

Table 16: Parameters for camera and lamp location distributions

Camera location (normals - Y, Z)		Lamp location (normals - Z only)	
σ_ϕ rad	$\pi/18$	σ_ϕ rad	$\pi/6$
μ_ρ blender units	6	μ_ρ blender units	6
σ_ρ blender units	3	σ_ρ blender units	3
a_ρ blender units	2	a_ρ blender units	2
b_ρ blender units	10	b_ρ blender units	10

Lamp locations were distributed according to a truncated normal distribution with the following parameters:

Appendix F Histogram of Confidences General Environment Test Set

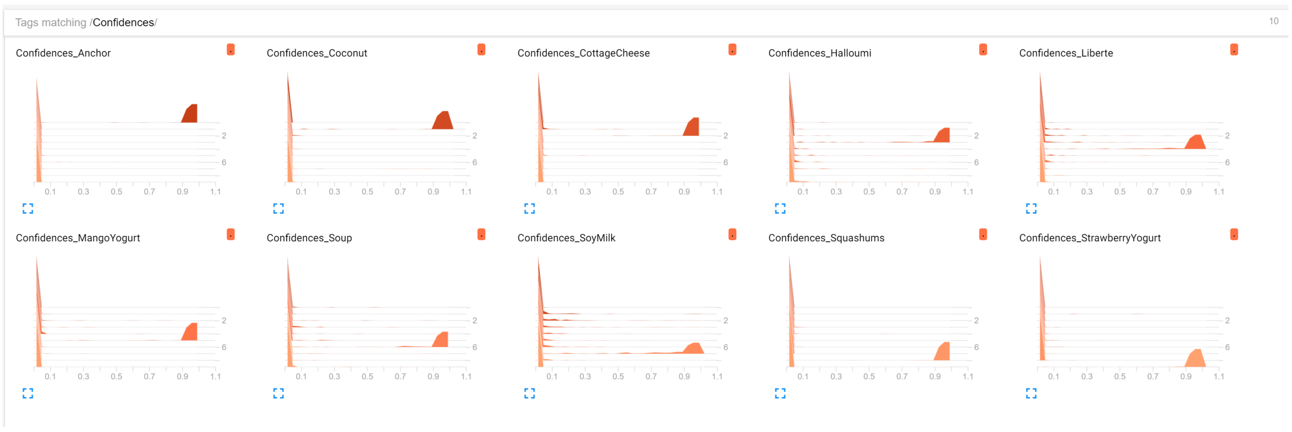


Figure 34: Histogram of confidences for 10 classes, when tested with the InceptionV3 architecture

Appendix G Log Book

This logbook contains the following resources that map the development process:
1) Write Ups - weekly summary written by each member about the progress on his tasks.
2) Minutes - taken during every meeting. Includes both team only meeting as well as meeting with supervisor.

G.1 Weekly Write Up

Every member of the team summarized his weekly progress in a Friday write up. The purpose of the write up was to give an overview of the progress to other team members. It was also used as a method to spot potential problems and unexpected delays. This allowed us to address these issue early, before they heavily influenced the progress of the project.

G.1.1 Week one

Kiyo

What you did this week
- Gitlab admin (repos/issues/Gitlab access restrictions task assignments)
- Set up first report and splits tasks
- Explore potential use of GAN for our extension
- prepare for meeting with Bernhard
- revise Python (especially numpy, panda etc) and Tensorflow

What is your task next week
- Revise Tensorflow - Study GAN - Build pipleline with Swen - Write the draft of the first report.

How much time you spent this week
- 10 hours

All of your tasks on track? (if there is any problems, describe it)
- Yes

Matthew

Items Completed This Week

Research
- Spatial Transformer Networks
- During the course of my research into the topic, I came across the concept of Spatial Transformer Networks. This idea was developed at Google DeepMind, and are modules that can be added on to a standard CNN that aim to reduce its sensitivity to geometrical variance.
- Given the huge amount of geometrical variance in the Ocado dataset (product are pictured at numerous angles and orientations), I realised that Spatial Transformer Networks could potentially be of great use to our project.
- I also found a similar concept known as TI-Pooling, which seeks to achieve the same results in a slightly different way.
- Reference paper: https://papers.nips.cc/paper/5854-spatial-transformer-networks.pdf
- Spatial Transformer Networks tensorflow implementation: https://github.com/kevinzakka/spatial-transformer-network/blob/master/README.md
- TI Pooling tensorflow implementation: https://github.com/dlaptev/TI-pooling

Experimentation, Coding and Implementation:
- Barcode scanning
- We had previously hypothesised that automatically reading product barcodes where available might be a good way to increase accuracy and to provide an additional layer of confirmation.
- I conducted some experiments to test this hypothesis. I tested barcodes from the Ocado dataset on various image barcode readers (specifically a few different python implementations on github, as well as a few commercial-grade solutions). Unfortunately, none of the barcode readers were successful in detecting/reading the Ocado barcodes (even though they were successful on other test images). I concluded that this was because the barcodes in the Ocado dataset were of very limited quality, due to the poor lighting conditions, low image resolution, and motion blur, making successful detection almost impossible.
- Text Recognition
- We also hypothesised that reading the text on product labels might prove promising in providing additional confirmation of a product.
- I conducted some experiments to test this hypothesis. Using Microsoft Azure's Text Recognition API, I was able to successfully read text from a test image. While the end-result was not perfect (a good number of characters were not read correctly), there were sufficient correctly read characters that one could reasonably see how this method might prove useful as a secondary means of confirming the validity of a product classification.

Tasks for Next Week

Research
- Continue watching Stanford CS231n

Experimentation, Coding and Implementation:
- Make robotic turntable with Max
- Set up the Kinect to capture 3D models of objects placed on the turntable
- (If we manage to secure products) Begin making 3D models of our initial 10 products
- Complete initial report
- Max and I will assist the other pairs if we (hopefully!) don't encounter delays with implementing the 3D modelling system

Time spent this week
- Research: 5 hours
- Experimentation: 2-3 hours

Tasks on track? - Yes

Max

Tasks completed this week
- Continued general research into machine learning techniques.
- Continued research into CNNs and GANs.
- Research into KinectFusion, other methods for capturing 3D models.
- Revised Python and Numpy.
- Began collating resources posted on slack channel for a Git repo.

Tasks for next week
- Build and test turntable and other capture equipment with Matthew.
- Experiment with different capture strategies with Kinect and cameras in order to obtain 3D models

of products.
- Matthew and I will help one of the other pairs if this is completed early (currently scheduled for Saturday 12th).
- Complete first draft of relevant report sections.

Time spent this week 5 hours

All of your tasks on track? Yes

Ong

What you did this week
- Explored feasibility of training a classifier from a limited set of data
- Explored feasibility of extracting 3D models of products for training data generation
- Explored 3D rendering software and potential for pipeline integration

What is your task next week
- Figure out how to build a good model-to-training data interface via 3D modelling software
- Define specifications and requirements on this interface
- Work with Pavel on building software through Blender
- Write the draft of the first report.

How much time you spent this week 10 hours

All of your tasks on track? (if there is any problems, describe it) Yes

Pavel

What you did this week:
Secretary tasks:
- Clean and upload minutes from both meetings.
- Set up Mendeley.

Technical research:
- Study one shot learning.
- Learn basics about GAN and prepare for meeting with Bernhard

What is your tasks for next week:
- Figure out how precisely to organize Mendeley and write up a manual.
- Add everyone else to Mendeley.
- Watch Udacity course.
- Discuss with Ong approach on 3D rendering and reconstruction and begin work on it.
- Start thinking about the content of my report section.

Total time spent: 6h + 3 hours meetings

Swen

What you did this week?
- Preparing for Meeting with Bernhard:
- Research into CNNs (Stanford CS231n), GANs

- Research into good vs bad training data
- Learning Git and Python

What is your task next week?
- Write my section of report
- build basic working pipeline with Tensorflow based on InceptionV3

How much time did you spend this week: - ca. 5 hours + 3 hours meetings

G.1.2 Week two

Kiyo

What you did this week
- First report draft
- GPU4 environment setup

What is your task next week
- GPU4 environment
- pipeline coding
- Finalize the first report and submit

How much time you spent this week
- 13 hours

All of your tasks on track? (if there is any problems, describe it)
- Yes

Matthew
Items Completed This Week
Report
- Wrote section on Preliminary Research for first draft of report.
- Restructured report into a new format that was ultimately used for submission to Bernahrd.

Experimentation, Coding and Implementation:
- Successfully constructed automated programmable turntable with Max.

Tasks for Next Week

Experimentation, Coding and Implementation:
- Select and order 10 products from Ocado.
- Set up the Kinect to capture 3D models of objects placed on the turntable.
- Begin making 3D models of at least one product.

Time spent this week
- Report: 5 hours
- Turntable Construction: 3 hours
- Meeting: 2 hours

Tasks on track? - Yes

Max
Tasks completed this week
- Built/tested turntable with Matthew
- Completed relevant sections of first report
- Helped review and edit first report to appropriate length
- Continued ML research/python
- Drafted product list (to be revised!)

Tasks for next week
- Preliminary test of turntable/Kinect with Ong or Pavel's laptop (I'll try with mine first but don't think it'll get far).
- Agree product list
- Order products from Ocado
- Start image capturing
- Further tasks to be decided during Scrum meeting

Time spent this week 10 hours (including meetings)

All of your tasks on track? Yes - provided Mathew and I start scanning next week.

Ong
What you did this week
- First report draft
- Blender scripting
- Render engine tweaking
- Wrote some scene randomization algorithms
- Camera path algorithms

What is your task next week
- More blender scripting, perhaps get on to importing 3rd-party models
- Background scene generation
- (Hopefully) be able to import 3D models of products, and generate some preliminary images
- Define a good randomization algorithm

How much time you spent this week 13 hours

All of your tasks on track? (if there is any problems, describe it) Yes

Pavel
What you did this week
- Finished Udacity course
- Searched literature for Optimal Camera positioning
- Image Blendering paragraph for first report
- Learning with Blender
- Manual Creation of Background and 3D object scene. Plus .jpg export of the final image
- Searched for possible background databases
- Some admin overhead(minutes, Mendeley)

What is your task next week
- Write a script that will add background, add 3D object and export to jpg
- Investigate different Blender native backgrounds
- Prepare first sprint

How much time you spent this week - 14 hours

All of your tasks on track? (if there is any problems, describe it)
- As of now, Yes, but I am having trouble with Blender Python scripting
- I will ask Ong for help as it seems he is much more skilled in this

Swen
What you did this week
- Kiyo and I got our Classifier Training Script Running on the assigned GPU04 machine:
    - Initial failed attempts getting our planned Anaconda environment to run
    - Set up Python Virtual environment, linked Tensorflow-GPU lib to CUDA
    - Downloaded Initial Training Data in Group Directory and redirected Training Script
    - Test run of the classifier
- Wrote a section of the first report and reviewed other sections.

What is your task next week
- To be discussed

How much time you spent this week - ca. 10 hours (incl meetings on Wed/Thu)

All of your tasks on track? (if there is any problems, describe it) - Yes

G.1.3 Week three

Kiyo
What you did this week
- First report update/submit
- retrain.py refactor

What is your task next week
- add final layer
- Display misclassified images
- Talk to Bernhard for the next step with Ocado
- Survive Tensorflow hell in general

How much time you spent this week - 7 hours

All of your tasks on track? (if there is any problems, describe it) -Yes

Matthew
Items Completed This Week

Report
- Proof-read final draft of report.
Experimentation, Coding and Implementation:
- Tested Kinect for 3D scanning and concluded it was not a viable solution.

- Successfully created 3D models using the Qlone app.

Tasks for Next Week

Experimentation, Coding and Implementation:
- Create models of the objects sent over by Ocado.
- Join Kiyo and Swen on the Tensorflow team.

Time spent this week
- Report: 2 hours
- 3D modelling: 3-4 hours
- Meeting: 2 hours

Tasks on track? - Yes

Max
Tasks completed this week
- Tested turntable/Kinect with Ong's laptop
- Got Windows 10 PC with admin rights from to install 3D model creation software
- Contacted Ocado to arrange product order
- Proofed Report 1

Tasks for next week
- collect products and do the scanning
- join rendering team

Time spent this week 6 (incl meetings)
All of your tasks on track? -Yes

Ong
What you did this week
- Managed to secure an Agisoft license, which enabled me to export meshes into Blender
- Managed to set up Blender Node network in Python, for controlling Blender Mesh appearance (Texture and material properties)
- Proof of concept: rendered 1000s of images of an imported mesh
- Spotted and solved a problem with rendering: camera spin was not randomized, causing subject to always be tilted at a certain angle when viewed from a certain position.
- Helped with experimenting with Kinect for 3D scanning
- Refactored the Blender Library, which will be documented over the weekend

What is your task next week
- Analysing randomized parameters and making sure that it is correct
- Look into randomizing more parameters (lighting intensities and introducing reflective surfaces?)
- Document the Blender Library in preparation of takeover by assigned group members
- Lay out specifications to test the Blender Library on
- Look into refactoring tensorflow code with Kiyo and Swen

How much time you spent this week - ca. 8 hours **All of your tasks on track? (if there is any problems, describe it)** - Yes

Pavel
What you did this week
- Prepare Sprint planning meeting
- Add background image to objects in Blender
- Design a new method of Image rendering based on RGBa and implement its basics in Python

What is your task next week
- Figure out the best way to rescale object or background image in order to get a realistic sized object
- Create a database of background images
- Automate the rendering so that large amount of data can be produced
- Read upon occlusion methods (pixel dropout, cover up) and figure out the feasibility of each
- Possibly (Implement pixel dropout)

How much time you spent this week - 11 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Swen
What you did this week
- Prepared for first sprint meeting: specifying the requirements for pipeline
- Researched Keras functionality

What is your task next week
- Set up our own Keras Model

How much time you spent this week - ca. 5 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

G.1.4 Week four

Kiyo
What you did this week
- Finish Tensorboard export
- Start adding fully-connected layer

What is your task next week
- Finish fully-connected layer

How much time you spent this week - 7 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Matthew
Items Completed This Week

Experimentation, Coding and Implementation:
- Purchased Qlone 3-month license.
- Created 3D models (OBJ files and textures) of 5 products for Ong, Pavel, and Max to use while waiting for the actual Ocado products to arrive.
- Completed initialization procedure for using Tensorflow on gpu04 and began looking through the retrain.py script

Tasks for Next Week

Experimentation, Coding and Implementation:
- Work with Swen on implementing a retrain script in Keras.
- Create 3D models of the Ocado products when they arrive.
- Contact the print service at Imperial to order a A1 size scanning mat.

Time spent this week
- Tensorflow: 3 hours
- 3D modelling: 2-3 hours
- Meeting: 2 hours

Tasks on track? - Yes

Max
Tasks completed this week
- ordered products from Ocado
- started working on unified rendering script

Tasks for next week
- finish rendering script

Time spent this week 4h
All of your tasks on track? -Yes

Ong
Tasks completed this week
- Finished basic documentation of BlenderAPI
- Refactored BlenderAPI
- Finalized randomization of variables in rendering

Tasks for next week
- Move on to writing a Tensorflow test script
- Write script to log evaluation metrics

Time spent this week 7h

All of your tasks on track?
Yes. Am anxious to get started on the test script in order to have a fully working pipeline

Pavel
Tasks completed this week
- Written a script that resizes background images and joins them with ocado poses.
- Created a provision database of approx 900 images.
- Downloaded a SUN database of 131072 images.
- Did some preliminary research into occlusion

Tasks for next week

- Finalize the format of the database of background images
- Do the occlusion
- Prepare next sprint

Time spent this week 5

All of your tasks on track?
Almost. I might not be able to finish occlusion (or fully incorporate it into the pipeline), but I was kind of expecting this to happen.

Swen
Tasks completed this week
- Listed Features of our retraining script
- Implemented basic new script in Keras

Tasks for next week
- Extend Keras Script
    - Splitting Data into Test and Validation
    - Produce Test Metrics
    - Print misclassified images

Time spent this week 4h

G.1.5 Week five

Kiyo
What you did this week
- Implement tunable fully-connected final layer

What is your task next week
- Keras/Tensorflow test integration
- Fix final layer bug

How much time you spent this week - 5 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Matthew
Items Completed This Week

Experimentation, Coding and Implementation:
- Learnt how to use Keras: trained models on the MNIST, CIFAR10 and Ocado datasets (including re-training an InceptionV3 model and training a simpler CNN from scratch).
- Created 3D models (OBJ files and textures) of Ocado products.
- Used 3D models (two classes, Halloumi and Yogurt) to re-train InceptionV3-based model and tested model on data from the Ocado dataset - achieved 73% accuracy!!!

Tasks for Next Week

Experimentation, Coding and Implementation:
- Keeping working on Keras to make it do more cool stuff!

Time spent this week
- Keras: 10 hours (7 hours learning and 3 hours working on training on 3D models)
- 3D modelling: 2-3 hours
- Meeting: 2 hours

Tasks on track? - Yes

Max
Tasks completed this week
- collected products from Ocado
- set up with Blender/rendering
- got rendering script working
- downloaded newest data from Ocado

Tasks for next week
- define and set up testing strategy
- set up code coverage tool
- prepare outline for report 2
- integrate rendering script with Pavel's new data/folder structures

Time spent this week 6h
All of your tasks on track? Yes

Ong
What you did this week
- Implement visualization of test result metrics in 'test.py'
- Implemented random background generation
- Captured Agisoft models

What is your task next week
- Integrate MVP (Minimum viable pipeline)
- Sort out remaining issues with rendering
- Run experiments for MVP

How much time you spent this week - 6 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Pavel
What you did this week
- Investigated the properties of Sundatabase and came up with a new specification based on its properties
- Downloaded most of the database and written a script that process the images into usable format
- Generated Demo

What is your task next week
- Produce training images! and finally automate the process
- Make occlusion, unless we decide not to do it based on Bernhard comments
- Fully support further training by providing desired images

How much time you spent this week - 5 hours

All of your tasks on track? (if there is any problems, describe it)
- Yes. There is a problem with my tasks always changing.
- E.g changing the database format, Bernahrd being skeptical about the need of occlusion.
- So I am making slower progress in total scheme of things than I wanted

Swen
Tasks completed this week
- Extended Keras Script, fixed evaluation bug
- Helped Ong scanning products

Tasks for next week
- Bring Keras script to TF retrain.py level

Time spent this week 7h

G.1.6 Week six

Kiyo
What you did this week
- Implemented Tensorboard custom callbacks on Keras

- What is your task next week
- Continues Tensorboard
- Unit test on retrain_test.py
- Write the second report on CNN part

How much time you spent this week - 7 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Matthew
Items Completed This Week

Experimentation, Coding and Implementation:
- Tested Keras models trained with Qlone data on multiple test sets.
- Created Flask web application that uses the Qlone-trained model to classify an image uploaded by the user.
- Ran tests on Qlone model using Tensorflow

Tasks for Next Week

Experimentation, Coding and Implementation:
- Unit Tests
- Writing Report 2

Time spent this week
- Keras: 5 hours
- Web app: 10 hours
- Tensorflow: 1 hour
- Meeting: 2 hours

Tasks on track? - Yes

Max
Tasks completed this week
- experimented with generating central documentation page
- more work on rendering script.
- got coverage.py working on all project modules. I have excluded build in/installed python libraries but still need to specify more files/folders to exclude (e.g. retrain.py)
- prepared outline for report 2
- set up a new test runner from GSC and got it talking to GitLab. I now have a basic continuous integration script running on a test repo and will set this up for Lobster once I have the dependencies sorted out (will do this weekend).

Tasks for next week
- integrate our final list of files to test/skip into coverage.py and GitLab CI (this weekend)
- get coverage figure appearing on GitLab
- write tests for BlenderAPI

Time spent this week - 8h

All of your tasks on track? - Yes

Ong
What you did this week
- Wrote tests for BlenderAPI
- Ran tests on Tensorflow
- Wrote unittests for tensorflow evaluation script

What is your task next week
- Identify and write more tests for the evaluation script
- Help with any unittests tasks on rendering
- Outline requirements for experimental pipeline (with Pavel)
- Write the second report

How much time you spent this week - 7 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Pavel
What you did this week
- Created a database of background images ready to use
- Generated training data for MVP
- Sorted the problem with images not showing and broken .png
- Started to design the whole pipeline

What is your task next week
- Write unit tests for all of my code and RandomLib
- Design the whole pipeline
- Write my bits to the second report

How much time you spent this week - 9 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Swen
Tasks completed this week
- Ran Keras tests for MVP

Tasks for next week
- Testing for Report 2
- Work on Keras script

Time spent this week -4h

G.1.7 Week seven

Kiyo
What you did this week
- Unit test on test_test.py
- Second report (specification section)

What is your task next week
- Continues Tensorboard

How much time you spent this week - 7 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Max
Tasks completed this week
- wrote single testing script with basic command line interface (-b to include blender, -t to include test files).
- script also does coverage and combines into a single report
- created testing branch on Gitlab and merged in tests (TODO: Ong's latest and Keras, and a final merging Sunday evening to generate coverage table for second report draft)
- got CI running with dependencies/overall coverage figure appearing in readme and installed Blender on test runner
- practice with Blender and wrote tests for BlenderNodes and BlenderLamp

Tasks for next week
- Draft my sections of the report
- Add testing strategy comments to BlenderAPI tests

Time spent this week -10h
All of your tasks on track? -Yes

Ong
What you did this week
- Unit testing for BlenderAPI, supervision of all BlenderAPI work
- Consulted with Pavel on the pipeline

What is your task next week
- Write my part of the report
- Design the rendering part of the pipeline, in accordance with Pavel's design.

How much time you spent this week - 10 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Pavel
What you did this week
- Unit tests on RandomLib, SceneLib
- Design the whole pipeline. Do a write up of the design

What is your task next week
- Write my part of the report
- Finish the design of the pipeline and implement it

How much time you spent this week - 15 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Swen
Tasks completed this week
- Wrote Keras Unittests
- Report

Tasks for next week
- Finish Unittests
- Finish Report
- OOP for Keras

Time spent this week -10h

G.1.8 Week eight

Kiyo
What you did this week
- Report
- Partition test

What is your task next week
- Rewrite EvalLib in Keras

How much time you spent this week - 9 hours

All of your tasks on track? (if there is any problems, describe it) -Yes

Matthew
Items Completed This Week

Experimentation, Coding and Implementation:
- Created iphone app
- Completed some sections of Report Two

Tasks for Next Week

Experimentation, Coding and Implementation:
- Keras tasks

Time spent this week
- iphone app: 10 hours
- Report: 3 hours
- Meeting: 3 hours

Tasks on track? - Yes

Max
Tasks completed this week
- my sections of the report
- test merging
- coverage things
- proofed report

Tasks for next week
- help Pavel with rendering interface
- VGG model
- submit Friday writeup on Friday

Time spent this week -6 hours
All of your tasks on track? -Yes

Ong
Tasks completed this week
- Started work on rendering part of pipeline
- Report

Tasks for next week
- Finish class structure for rendering pipeline

Time spent this week -7h

Pavel
What you did this week
- Report

- Unit testing
- Sprint preparation

What is your task next week
- Finalize the pipeline design
- Implement the pipeline design

How much time you spent this week - 5 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Swen
Tasks completed this week
- Wrote Keras Wrapper
- Report

Tasks for next week
- Figure out how to report Keras Metrics in Tensorboard

Time spent this week -7h

G.1.9 Week nine

Kiyo
What you did this week
- Rewrite keras_eval

What is your task next week
- N/A

How much time you spent this week - 3 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Matthew
Items Completed This Week

Experimentation, Coding and Implementation:
- Added extra feature to iphone app (clear button to remove previous classification)
- Helped out with integrating Keras model into Tensorboard

Tasks for Next Week

Experimentation, Coding and Implementation:
- Studying for exams!!

Time spent this week
- iphone app: 3 hours
- Tensorboard: 3 hours

- Meeting: 2 hours

Tasks on track? - Yes

Max
Tasks completed this week
- rendering pipeline with Pavel

Tasks for next week

Time spent this week -8 hours
All of your tasks on track? Yes

Ong
What you did this week
- Help rewrite keras_eval
- Finalize rendering pipeline and interface
- Produce test plots for initial feasibility study
- Looked into rendering optimizations

What is your task next week
- Not fail exams

How much time you spent this week
- 7 hours
All of your tasks on track? (if there is any problems, describe it)
- Yes

Pavel
What you did this week
- Finish Rendering pipeline design
- Implement the pipeline

What is your task next week - Study

How much time you spent this week - 13 hours
All of your tasks on track? (if there is any problems, describe it) - Yes

Swen
Tasks completed this week
- Added Functions to Keras Wrapper
- Added functionality to interface with zip file as input
- Did research on gathering confidence data from Keras eval

Tasks for next week
- integration in pipeline

Time spent this week -6

G.2 Minutes

For every meeting minutes were kept for further reference in a shared Google drive. Minutes were taken for both team only and supervisor meetings. Agenda for the meetings was assembled in advanced and discussed during the meeting. Where relevant, a list of tasks with assigned team members was assembled at the end of the minutes as an overview.

G.2.1 27.11.2017

Meetings with: Dr. Bernhard Kainz
Feel free to add your notes
Project: Machine Learning for Product recognition at Ocado Technology

We will be given access to Ocado Data, and can discuss lots of things with them, but Bernhard would like us to do more than what just Ocado wants, so he will supervise it as well. The image classification should be doable and not too hard according to Bernhard so he is very much interested in what we can do from there. Some ideas include: testing it outside of warehouse environment, using it as recycling app, doing a counter on tablets used from larger pack in order to make order when X are left We can use PyTorch or Tensorflow.

Auto captioning Meta-data of products

How much interaction do we expect to do with Ocado vs Bernhard? At the beginning will be mainly with Ocado as we need to collect/analyze their data, later with Bernhard. Ocado already trained their model using their data and the accuracy is 60-70%. They want us to improve their model further. Bernhard said we could retrain using open source data, such as ImageNet, instead of using their own data. Ocado is thinking that we could implement the DL model such that it confirms what the robot is picking is right, since this confirmation of the picking is currently done by human.

G.2.2 9.1.2018

First meeting in 2018
Date: 9.1.2018
Time: 13:00-15:00
Present: Everybody

Goals of today:
1) Prepare for meeting with Bernhard
   a) Go over the issues in gitlab
2) Internal regular meeting
3) How to split tasks
4) Minutes

Deliverables/products

Don't overpromise. Overdeliver
Highlight problems: Chicken thighs vs chicken breasts (very similar)

Ocado used: trained both on image nets(went well), and only on their images(did terribly)
We will use frozen Conv net and unfreeze last layer, or use ours(transfer learning). If we have more data and time, we can unfreeze more layers.
We will play with the last layer, the fully connected layer.

If we want to do detection, we need recognition. For recognition we will use pretrained model, with our own fully connected layer.

There will be two parts to the project. First will be getting a MVP done, with fully functional pipeline and accuracy higher than Ocado. Second, will be extensions. There are several ways in which we can go about extensions, which will be investigated later.

MVP:
Recognition: Pretrained + Our own fully connected layer for recognition
Goal: Accuracy better than Ocado
Software:(training pipeline)
Stages: Read ,Clean the images, Augment, Load into queue, training pipeline, shell script, Working pretrained model with our own fully connected layer.

Cleaning could include (Background subtraction, Product zooming), but need further discussion of its advantages(e.g. Will it not make our model unreliable in other environments?). Don't need to zoom on the product, as the convolution is done on every scale, but for that we need images on every scale. Ong is not sure about this. For now, leaving as potential zooming.

Extensions: GAN, potentially one-shot-learning if promising, capsule model, YOLO(You only look once), RNNDetection (video), application(recipe recommendation,),

Data augmentation, training data preparation:
Should we try and use GAN? Will it increase the accuracy with bad training data (not good images) - we could use the pure images from the website.
What Ong did: Inception in TensorFlow Cov layer, bottleneck with outputs being features, not classes (2048 features, not necessarilly 0 to 1, e.g. not probabilities) it is not the final layer.
Took website pictures, augement them heavily, and use them. Give very high accuracy(85%)

Problems with training images: all products have only one side, as image taken just after barcodescan. Training images consist of video stream, which is according to Ong a problem. Bad for training, if we pick a certain part of the stream for training and the rest of testing, it will be very simillar, without the model actually being that good. But they always take four images, so we know they belong to one sequence and do stuff about it. So technically we have only one fourth of the data for training.

On 500 testing data for binary clasificator, Ong achieved 75% with one training picture(from the website), which was augmented into 300 images. However this was only binary(chicken vs vegetable). We have very little intraclass variation - advantage

Plan for mitigating the problem with bad data: We get a big batch of training data from image augmentation of the website image. From that we have a good representation of skew, glare brightness. Feed this training data to CNN. Thus we use image augmentation alongside GAN to generate large amount of high quality training images for the classifier. We feed the augmented images into Discriminator along with real training data. We have no way to get unbaised data. GAN will give us a way to generate lots of training data for the actual clasifier.

We need training data of nothing to be able to confidently say there is nothing in the image. "Not sure what in image" is not equal to "sure that nothing in image". If there is low probability with all of the classes, what to do next? Expert input e.g. tell human I have no idea what to do, tell me what is happening

Clasifier architecture: CNN classifier - Inception We will use TensorFlow.

Other Agenda: Meeting with Bernhard probably once in a two weeks(after each sprint) Our own meeting once a week, with a quick write up half a week later. Use Gitlab Issues a lot!

Questions for Bernhard:
What interface: give .pp file, interface to .pp file, API, shell?? GUI, WEBapp
What to do about the hand?
Would the advantage of one-shot learning (learnig with little data) be beneficial for our case?
PCA/Whitening is not being used in practice?
Normalization not necessary for our images, because they have the same scale
To what kind of data do we compare our accuracy against when we measure our result, does it include images where the front of the product cannot be seen?
Is it better to use a few good data and do one-shot learning instead of using lots of data?

Do we need to decide one learning strategy, or multiple ones?

Learning strategy 1: Train top layers of vanilla CNN and validate accuracy

Learning strategy 2: Try something fancy like one-shot learning and see if it has higher accuracy (or maybe same accuracy with less data input/shorter training time)

A benefit of building a more flexible image pipeline that can train on lots of poor quality images is that its performance could (presumably) improve in the future as Ocado takes more pictures (especially if those pictures are from different angles etc). Is that a legitimate aim for us, or are we only trying to achieve high accuracy using the images they have now?

Agenda for the next meeting:
Development strategy (Scrum, XP programming).
Learnig strategy

G.2.3 11.1.2018

Meeting with Dr. Bernhard Kainz
11/01/2018 9:00 (originally scheduled for 10/01/2018)
Present: Everybody

First report should be mainly about background research. Overview of what literature we have red. Not much work done is actually expected. We could mention current state of art. Overview of WasserStein GAN.

In terms of problems with chicken tights vs chicken wings, it might be a good idea to define a hierarchy of classes. E.g. meat-¿chicken-¿wings, We could use the hierarchy from Ocado webpage. However this would require some data curation. Another possible way to go is to try to read the barcode. However this could only be a second check to increase the reliability of the evaluation. There are some barcode detectors out there, should investigate how useful. Getting training data should not be a problem at all.

Good way to go is the way Ong suggested. Render 3D image, using agisoft photoscan(if we dont find open source software, it should not be problem to buy a license for this).
Adversial training of the Discriminator in GAN. Use the 3D rendered images as real images and train for edge cases using the Generator images.

Wasserstein GAN should be easy to implement. We should be careful about data explosion (the

amount of training data necessary might explode exponentially).
For rendering 3D images we should study the best angles from which to take the images(**Fibonacci sampling** on a sphere, although apparently the effect might not be huge).

The key to everything is to get good training data.
Our model should ideally work in both the warehouse and outside (customers houses). Thus it would be good to have training data from which we can generalize. Background generation in GAN could be useful, however there could be the problem of the object being sharp, and the background blurry, which would not give a good generalisation. Or we could train on images with no background

For our networks, we could initialize the weights with a pretrained network.
THere is a lightstage downstairs, but not sure if they would be happy if we used it. Also not sure if necessary. Trying different lighting might result in data explosion.

Blender for rendering.
Kinect for 3D reconstruction.

In terms of products:

- Ocado used image nets with Inception and reached accuracy about 50%.
- Anything above that is good, although we should aim over 80%.
- GAN does not have to be part of the MVP.
- For MVP we should pick some nice set of products, which we can get from Ocado (lobster, caviar,..).

In terms of final report:

- We are not expected to provide Ocado with code ready to run.
- We should do a proof of concept and then write the report of what we did. We are also not expected to come up with some new crazy research, but rather apply current state of art on this problem, try to find the best way and then evaluate and report on it. For the presentation, it might be nice to have a life demo. (A Barchart with probabilites of each subclasses).

Admin: We should be able to get access to uni GPUs or credit for AWS.
Do not overload Ocado with meetings, show some independence. Send them our first report and then when we actually have something, or need something.
Meeting once every two weeks with Dr. Kainz.
Send our draft of the report to Dr. Kainz for feedback.

Swens notes:
Creating subclasses:
- Classify as fridge product good enough?
One idea:
- Two nets, one gives opinion on barcode, one from photo?
Problem: Good training data
- Adversarial example generation?
- Wasserstein GANs, easier to train than DCGANs / just implement from github
    o No need to understand the math behind it
- Posnet (Cambridge) learn the camera pose for each image
Adversarial Approach:
- Unclear to me where this comes in useful
For sampling:

- Polar coordinates sampling vs Fibonacci sampling, read up

Hierachy:
- Multiple labels
  - o Vegetable
  - o Green vegetable
  - o Broccoli
- Helps the training

First Report:
- Put in text what we have done now
- Few lines on how CNNs work, what the state of the art is, how were gonna train is
- Maybe GANs

MVP:
- Pipeline
- 10 product
- Robustly detect 10 products
- Proof the pipeline works
- Select nicely behaving images

R&D project, we dont deliver any software
- What was our hypothesis, how we implemented in and how we evaluated it
- Live demo would spice things up
- Using a smartphone camera, hold into camera, overlays name with python open cv
- Bar chart with probability
- If hierarchy from webshop, then show both bar charts
- Access to GPU machine in lab
- Azure/aws credits somewhere in our documents, WJK might have credits for azure

Not overload ocado, show independence
- Write first report, which we will also send to ocado
- Send draft to Bernhard, he gives feedback, then we meet

Agisoft license:
- Explore alternatives, then potentially buy it (write to Bernhard)

Models:
Light stage
Blender for rendering
Microsoft Kinect for 3D reconstruction: get from Bernhard

G.2.4 17.01.2018

17.1.2018 minutes
13:15-14:40
Present: Everybody

Goals:
Discuss First report
Progress update

First report
There are three parts of the report:
External specification
MVP
Development approach used(Scrum)
Version control specification

We should also include Feasibility, boundaries, completion dates for low level design According to Bernhard this is R&D project, so we will not pass the code to anyone. The final outcome of this project is a report.
External specification: Contains Schedule, Requirements, Assess difficulties/risks
The report was promised to be sent to Bernhard by Friday

High level:
Have requirements, specifications. The goal is to have a complete pipeline as explained below, which is able to learn classifying products in different environments.

4 parts of the pipeline and then two parts of the evaluation process:
Data generation
3D modeling
Rendering
Generating training data from 3D models
CNN (or other network)
Training accurate model
Evaluation
Result visualisation. (E.g. percentages assigned to different products). Metrix to measure success. Allows conclusions to be reached, from which improvement of the product can be achieved/ the effectivness of our approach evaluated.

Lower Level:
Asses Feasibility, Boundaries,

We have nice design, but how did we get from Ocado to this. Start from the task given to us as high level. Define the problem with given data and move towards a solution that we proposed to mitigate this problem-¿ Describe our design.

We looked at high level goal, the resource they gave us. Not suitable data for the task. That data part is not feasible. But making image recognizer is feasible, so we came up with a new solution. Put it at the start of the report.

Deadlines:
Data generation 2 weeks
Rendering 3 weeks
Network 3 weeks
Further one and half weeks for incorporating all the bits together.
Followed by a one and half weeks of testing.

Boundaries:
Data generation: section will provide a labelled 3D models in .stl . 10 products. Including Mash and texture.
Rendering:
Provide labeled .jpeg images. Pictures are of good quality(over 90% of the training images, shows at least 80% of the product)
Network: output is a .pb file and output.txt. Input Jpeg
Evaluation: input .pb file output.txt; output accuracy and training graphs; confidence of test images

Things to consider for Rendering: How much of the product should be in the picture. What is the desired quality of the pictures. (How much of the product will be in the picture. Majority of good ones, but some worse could be useful).

Should start thinking about which products we will use. Max and Matthew will do. We will probably pick some distinct products at first. However a discussion might be still useful.
Put a timetable at the end of specifications, which will lead into development approach.
Each pair continuous in its current work, but focuses on the report first. Matthew will reorganize the current report by tonight so that most of the work on it can be done tomorrow and sent of by Friday to Bernhard.

Who does what by when:
Matthew and Max will update the structure of the first report based on above by Thursday Afternoon, and Kiyo will send it to B on Friday.
Apart from that, each pair will continue to work on their tasks.

G.2.5 18.01.2018

18.1.2018 First report draft meeting
18:00-19:00
Present: Max, Swen, Ong, Kiyo, Pavel
Agenda:
Prepare first report draft so it can be sent to Bernhard
Discuss development strategy to be used (Scrum/XP)

First report draft:
The whole group went over the whole report and commented and changed needed sections.

Discuss development strategy to be used:
Scrum will be used with stand ups on Monday, Wednesday and Friday. If necessary, writeups over Slack can be done on the other days.
First sprint will start on 22.1. And will take 2 weeks.
Pavel will prepare the necessary things for Sprint planning meeting, along with a doodle to set up a date.

To dos:
Everybody will go over the draft before Friday noon, so Kiyo can send it to Bernhard
Max will add a part about Scrum into the report
Pavel will set up a doodle for Sprint planning meeting and prepare all issues for that meeting.

G.2.6 24.01.2018

Minutes for meeting on 24.1.2018
Present: Max, Kiyo, Swen, Matthew, Ong, Pavel
Start: 13.10
End: 14.40

Completed:

Finalising the first report
Team members confirmed all Mendeley reference added for first report
Changes made to google doc by Kiyo
Changes to be integrated into Latex by Kiyo.

The standup meeting:
Everyone will copy in rest of group to future email communications.
Swen - has created Retraining script. Will try to do retraining with Keras. In progress. Issue from Ong: tensorport equivalent?
Kiyo - same as Swen. Requested help.
Ong- Has written blender script to render 1000 random images. Needs OBJ file and a texture mapping from Max and Matthew.
a) Suggest Max and Matthew join Pavel with rendering.
b) This is because TenserFlow has steep learning curve, meaning better to have a smaller number of people knowing low level code.
c) Ong would like to move to CNN to finish library.
d) Pavel would prefer not to do more admin

Gitlab Tasks/Sprint Planning:
Worked through all issues, confirming if in correct sprint
Tasks specific notes decided during meeting to be added to gitlab notes.
Gitlab times:
a) Time estimate will be added when a task is started
b) Time spent on a task will recorded as people work
c) Estimates and total time spent will be review at the next sprint meeting
Everyone will add detailed comments, issues, progress to gitlab issues in comments
Everyone confirmed they are happy with the allocation of issues
Standups will be on Mon, Wed, Fri
Sprint review will be on Friday - to be organised by Pavel

G.2.7 31.01.2018

Minutes 31.1.2018
13:00-14:05
Present: Everybody

Agenda for this meeting:
Make sure everybody has something to do
Discuss a Demo show, e.g. some time where differents part of the group will be showing their demo. (Maybe after MVP is done)
Discuss possible meeting with Bernhard
Stand up

Stand up:
Pavel just pushed changes from last time
Ong, optimized camera placement with respect to object. Range is adequate. The object size is between 3* and 0.3 times scaled.
Randomized lightning also adequate.
Refactor library and added documentation (src/rendering/readme.md)
Should open source the API for Blender

Helper Kiyo

Max: What did Luca say about products:
He would pick products that dont look diferently when consumed.
Product should be delivered to Max. Max gave them list of times, havent heard from Luca back yet.

Swen:
Working on Keras.
Question if we should switch to Keras, integrate with Tensorflow or abandon at all
Keras has working Tensorboard
Demonstration of Keras code

Matthew:
Set up a GPU. Ready to go. Moving to Keras

Kiyo:
Tensorboard now shows misclassified images

Demo and sprint review next Wednesday - Pavel prepares
Pavel and Max will finish the Rendering pipeline. Next sprint they will move to some work on the network.

In regards to Keras/tensorflow. We will use tensorboard for MVP as it is close to finish (some work to be done on tensorboard and final layer). At the same time Matthew and Swen will start building similar network in Keras. Tensorflow will be done by Ong and Kiyo. The reason is that Tensorflow is Google's code so it will not be easy to manipulate the network. If we have our own Keras code, we are more flexible in changing it.

The problem with the testing is that on some images there are more than one product(or the product is totally concealed). A potential fix suggested Matthew. There are always four images in succession. So find the part that is changing (through velocity) and crop the image to contain only this part. -Matthew creates Git issue for next Sprint.

Discuss with Bernhard:
Can we change the testing data? E.g. remove images that dont show the product, or show more than one product Image rendering. Show training images to him and discuss Occclusion (e.g. should it just be a black patch?)
Pavel will bring the images to the meeting
Pavel will make sure the github is being updated, issues closed, time logged Checked before Friday standup.

G.2.8 07.02.2018

Meeting 07/02/2018
13:00-15:00
Present: Everyone

Agenda for meeting:

Sprint review:
Show Demo, if possible, everybody should show some output of his work, e.g. part of the pipeline and the results of his work.
Each person can evaluate how much he is happy with last sprint (e.g. if he wants to move to other parts of the pipeline, if he needs help...).
Overall we will discuss what we can do better in next sprint
Sprint Planning meeting:
Discuss, where we want to be at the end of next sprint. Especially mention key tasks and functionality that needs to be done.
Move any leftovers from Sprint 1 to Sprint 2
Go over each task in backlog and make sure we all agree on what each task entails. Make sure no task is missing.
Let people volunteer for tasks.
Make sure all tasks that need to be done are taken
Bernhard meeting preparation:
Make a list of things we want to update Bernhard on
Make list of demos we want to show
Make list of questions we want to ask and discuss
Bake a cake for Bernhard so he likes us more

Going through finished tasks:
For Keras code we need to figure out what metrics we want the code to output as the default are just two basic metrics which might not be enough for us. - Issue
For object rendering we need to ensure that the object is scaled to one meter cubed so that its size is representative -issue
Printing misclassified images is going well but still needs work to be done
Both Keras and Tensorflow runs. Tensorflow is MVP ready. Keras needs some additional features to be deployable as MVP follow up.
For the Keras, we want some API that we define few arguments and make it run. Ong: Be able to write at most 10 lines in main file and run the whole thing without too much trouble for the user.
Kiyo is working on specifying our own fully connected layer.
We have problem with having object poses showing the not scanned side. This shows as a black outline in the training images and is a not good for training.

Where we want to be done after sprint two:
The aim for Sprint two is to take the parts we have at the moment and put them together. Be able to get some results for at least some classes (not necessarily all ten).
We should try to run a test by Monday. So we have some idea how well we are doing.

Deadline: Monday we want to have three products tested. Training test and Validation will come from image rendering. Tested on Ocado. Training: Thousand rendered per class split into validation and training. Testing 300 images per class from Product images dataset. Output: Confusion matrix, sensitivity precision, accuracy

People should start thinking about testing. Use coverage.py. Rendering definitely needs testing. Think what other parts can be tested.

Documentation should be done in a uniform way so we can generate it quickly and easily See: src/rendering/Blender_objects.py for example. Figure out how to generate the docs to see if generated correctly. Sphinx documentation format.

Next Wednesday we should discuss extensions. The extension should only be done in case of good accuracy. Good accuracy is priority one.

Discussing what tasks need to be done in Sprint 2. Goal is Monday have some results to discuss. Does not have to be optimal

Questions to ask Bernhard:
How to solve the problem with not scanned side. We are missing the bottom side of imaging during scanning
Two classes taken pictures from two orientation
Use just the best pose
Ask about the call. Is the other group being going to be there
Should we present something? some results?
Show demos, Pavel generated images
Ask about our decision about Tensorflow, Keras

To dos:
We need to find a tripod. Ask Photo soc, 4D recording room. Pavel and Max bring cameras.
Scanning on Friday 2-4. Matthew will book room. Ong and Matthew will be here from start, Pavel and Swen after databases finishing at 3
Pavel and Max agree on final Rendering Pipeline
Everybody should think about what values we should log during training the pipeline. We need to think this out really hard as it is important for report.
People will start putting their comments into the issue for the logbook.

G.2.9 08.02.2018

Meeting with Bernhard

Issue: Issue is background just random noise, instead of occlusion or natural background using various elimination positions/light sources.
Problem with random background is need to adapt illumination to background. So from learning perspective random noise would have the same effect.
In real environments, overall lighting conditions have an effect. Model includes sampling of all conditions, sun positions, light positions etc. Illumination constant across the image.
Introducing wrong steep non-differential component.
But for this project too hard automatically to put background illumination (of stock photo) on foreground object.

Suggested approach for basic proof of concept:
Key Question: if just random noise: how good is if just one-class classifier. This would show general approach works. Can model identify in a random environment (e.g. warehouse). Is this better than just train on warehouse images?
If this works, then move to images taken kitchen. Naive, but sufficient comparison: compare our model with another model trained on warehouse images on test data on products in e.g. a kitchen. This would be our proof of concept.
For a basic approach, could even train from model gif on slack and see how it does (background not great and logo, but he thinks might not have terrible performance).

Over-the-top approach:

Model indoor scenes e.g. whole virtual environment of kitchen. Render product in model with lighting. Retracer (?), other sophisticated approaches.
Only extra thing you learn is to ignore clutter, but performance increases.
But an additional add-on - too much for main project component.

Choice of background images: Issue: it will learn silhouettes where random images are outdoor images etc.
Issue is constant bias across the image. We have different gradients.
Possibility: ignore background by implement ignore component (-1 constant in architecture). But highly overfit on object as won't have other features.
He thinks random feature noise (across whole image or local features, which are just learned to be ignored) will perform fine.
If we have to pick image for background, use roads/grass because not as much illumination bias. Would have the bias in kitchen bedroom images. Maximum he would go to is kitchen background, with images/tests also on random background.

Next steps for project:
1. Train on random background.
    - Build a minimum pipeline and see if it trains at all.
    - Constant background or randomly changing. Tiling.
    - For constant background, he expects overfitting.
    - Validation set is synthetic images, test is warehouse.
2. Try warehouse image training and compare results.
3. Add features / improve
    - sim real environment
    - GAN
    - Backgrounds
    - Illumination (if focusses on product but can't identify the rendering or illumination)

Binary classification:
- Binary class first. Don't need more as can assume if it works on binary can work more more.
- Can do more, but object or not is fine for first stage.
    - Should do one product vs no product. No product could be random or warehouse for other products.
    - So we need a background class. For proof of concept can be product vs product, but ideal is representative background.

Data processing:
Our main focus is: data acquisition + general augmentation approach. He expect us to be worse in warehouse but better on transferability.
The more data augmentation, the more invariant in data. Normally translation rotation flipping, which makes model ignore spacial distribution.
Risk overfitting as we have so many augmentation options. Constant background with light noise and blur might be enough.

Model creation:
Bottom of products:
Make sure hole not visible in training data. Just use slightly truncated hemispheres. Be very careful about the fillers in occluded side.

Artefacts on models:

See how much influence it has once we build the model. Might be worth mixing data: Ours + warehouse images. This would be a bit biased to warehouse environment but might be better than just using 3D model images. Lots of opportunity for experimentation! Tweak proportion.

Approach:
So throw it all in but keep focus on judging what variants/approaches actually contribute.
Other possible add on: can do some things on his light stage. Small single figure boost in final performance maybe.
Final conclusion for project: the approach works, but just needs more manpower.

Keras/tenserflow:
He doesn't mind. Can take foreign code or write your own.
Don't put too much work in architecture dev. Simple VGG would do fine if data is right. Mobile net might also be interesting.
Models are interchangeable. VGG/inception performance difference very small.
He like Keras for soft engine. Simplicity of testing.
Our project is 90% data management, evaluation, tweaking. Maybe 10% coding architecture.

Monday meeting:
Monday meeting: brief explain approach.

G.2.10 14.02.2018

Meeting 14/02/2018
13:00-15:00
Present: all
Agenda for meeting 14/02:
Standup:
Discuss interface between image rendering and CNN.
Discuss software engineering tasks that need to be done to scale up!
Image generation:

- Systematic (like in the qlone images) or random (Matthew is of the opinion that systematic is better) (Ong is of the opinion that some balance can be struck, this can probably be determined through extensive experimentation).

Web app, iphone app, DEMO
Discuss what to do now. E.g. how to regroup, what tasks to focus on (scaling to more classes...)
Come up with a system of training, testing, data generation, so that it is consistent IMPORTANT
Formalize initial results, then immediately focus on software engineering tasks
Second report

Standup:
What do we actually need to log. Logs from Training as well as from testing.
SPinghx seem a bit more complicated than expected, so for now we will just use consistent commenting and if we have time we will do something with it.

Strategy for further work:
For getting processes automated we need to define some further interfaces. From discussion a system was crudely designed that Pavel will fully develop. For each training set we will have a folder that

will contain all the images, all the parameters used for generating the data. For each test run on that model the results and evaluations will be stored in the folder under that tests folder.

There seems to be problem when converting models from .h5 to .pb the accuracy drops rapidly. The pb file predicts most of the stuff to be yoghurt. Very inconsistent. Something must have been lost in transition between Tensorflow to Keras. The easiest thing is probably to get everything work in Keras.

Translation from T to Keras is infeasible. We should do everything in Keras now. Some of the work on Tensorflow wasn't wasted as it can be reused.

Sun images are underperforming at the moment. This reason might be that we are using just one layer which might not be enough for the separation of the object from the background.
We also want to run Qlone (Matthews app) with background as well.

Matthew wants to implement an Iphone app that would use our classifier. Definitely something we want, but what is the priority? This sprint is very busy because of report two, so will have to do later

Report two:
Want to send it to Bernhard on the 26. So everybody should put their parts together by then.
Max produced a rough draft
Prototype in Tensorflow
We need to have a lot of testing done as it is main part of the report:
Rendering: Ong+Pavel
Keras: Swen+Matthew
Tensorflow: Kiyo
Chapters of report divided between people
dont need to test retrain.py but need to test everything

To do: Pavel: Produce proper training data for MVP (The same amount data)
Swen and Ong run the test and generate output.
Pavel will design the whole pipeline
Kiyo will design the evaluation matrix.
Do MVP and formalize results: Use tensorflow with two classes and create an output of the test.
Pavel and Matthew will talk about the specification about what the training people can ask the rendering people Use unittest module - Max will study this further
Use unittest and run test for python, everybody

G.2.11 21.02.2018

Meeting 21/02/2018
Time 13:00
Present: Everybody

Agenda for meeting 21/02: Discuss the deployment of qlone images:
Should we generate object poses directly from qlone, or should we get the .obj files and use the same system as for Agisoft obj files?
Would it be good to have a completely separate object generation for comparison?
Second report draft discussion

Stand up:
Mostly testing
Ong will need to enhance the BlenderAPI to catch some edge cases. For example a wrong object reference is accepted by some of the functions, even though it should not.
Ong: Add comments about general approach to testing. Blender API is 86% coverage.
Keras is not working atm.
We have Coverage.py tables
Coverage has exclude, so that we can exclude the tests from the coverage count. Right now we have two coverage tables, but it should be doable to combine them.
Swen is unsure about some thing about testing main in keras. Will ask Fidelis

Qlone images:
We will abandond qlone alltogether in favour of Agisoft as it offers better accuracy. The increase in workload is reasonable for only ten products.

Getting ten products:
So Max and Matthew will spend one afternoon to get models of all 10 products. Quickly generate training image and push it into Keras. This will be a proof of concept and a basic accuracy measure for our improvements. The reason for doing this is that qlone is fast in scanning the images and producing the training data.

Report:
The update to specification will be as follows: A single table of the original specification with updates in red. Further comments can be put below

Prototype:
Need one section for the prototype.
Do we need UML? Probably not
Need to update them on our progress.
Write the results of the two class prototype. Refer back to the specification of what was already implemented. We can add a done collumn to the specification table to be able to show what was already done.

Schedule: Kiyo can talk about schedule along the specification update.

System Testing:
With respect to specification what test do we do?
Table:
Collumns: No, Specs, Test, Validation, Meet specs.
Rows: Input, Output

Get keywords in.
Performance testing: measure time of the running
Usability test
If we have space we could some mention of prolifing. E.g. where the bottlenecks are.
We will have nice big tables

Unit Testing:
Probably the main focus. Talk about different components
Put table into appendix showing all of the unit tests

Split into different section. Discuss how to test the section in general.
Testing sections:
Tensorflor (Kiyo)
Keras (Swen)
BlenderAPI(Ong)
Background +Merging (Pavel)
randomLib(Pavel)

We should test not only positive cases but also raising exceptions, edge cases.
Try to have some exception catching. And test for it. Probably too much work to do it everywhere. Or in comment just state that the behaviour is undefined for these inputs.

Integration testing:
Test that everything puts together. The prototype is proof of this. E.g. the prototype works. The whole pipeline.
Put comments to the tests. Talk about partitioning e.g. what inputs and why are you testing.

Coverage:
Make sure that we have high branch coverage. We currently measure statement coverage. Need to make sure branch coverage is good as well.

Regression testing:
Max is doing it. He will include it in the report

To do:
Everybody:
Merge tests into master as soon as done
Everyone look at system testing and comment
Kiyo: Update the specification table and schedule.
Swen: Send email to Fidelis about his question + should we put every a large table of every unit test into appendix. Ask if we should migrate our repo. Update on progress of the prototype.
Pavel: Updates specification. Unit test oversee. Write initial bit of the Unit test section.
Matthew: Update specification and System testing.
Ong: Update specification and create System testing table.
Max: Update specification. Get Code coverage and Regression testing.
Someone: Migrate the repo to the given one, if we need to.

Deadline: Try by Friday, Full deadline by Monday.

G.2.12 26.02.2018

Minutes 26/02/2018
11:30-13:00
Present: Everybody
Agenda:
Go through second report draft and prepare a version to send out to Bernhard and consult with Fidelis.

Discussing the prototype. How to present our product that is a system integrity test. Need to add a small paragraph that talks about how our product meets the specification. Must ensure that dependencies in the specification table follow one another. E.g. A which depends on B, the due date of B must be after A.

Things to ask Fidelis:

- Ask how much detail do we need to go into the update on specification (especially the update on specification paragraphs)
- Will he be looking at the code? E.g. can we put only the most important bits into the report and if he is interested in the details he can find details in the comments of the tests.
- Is he going to mark the page with details of unit test
- Is 5 page the hard limit?? Yes
- How detailed must the test strategy section be? Do we need to detail each section or is just one overall description enough?
- Do we also need give some background of the codebase in order to justify testing strategy?
- In the unit test section, should we give a detail description of an example unit test, or is general comments enough? If yes, what should be the ratio between talking in detail about concrete UT versus general remarks.
- Is the report supposed to be in past or present tense? Weve ended up writing the unit test section is past tense, and the system testing section in present tense. In a way this makes sense because weve already done the unit tests but havent done the system tests, but at the same time the incongruity is quite glaring.
- Do we only need to justify non-100% coverage if coverage numbers are very low?

Todo:
In update on progress, make emphasis on what the problem was and follow with
Pavel cuts shorter the second paragraph in Image rendering
Ong updates evaluation in Specs updates
Matthew will add Final Product section by moving it from appendix
Pavel will look at the UT vs ST so that we do not say similar
Pavel will create Scene lib and ensure everything runs
Rename test.py to tensor_eval.py

G.2.13 28.02.2018

Minutes 28/02/2018
13:00-15:00
Present everybody

Agenda:
Report 2, incorporate Fidelis advice
Sprint 3

We go quickly through the report and discuss what we should change. Changes are implemented by individual people later. We should include 4 images and draw images between them to show the pipeline Kiyo moves to Latex tomorrow. Get your changes done by tomorrow 5p.m. so Kiyo can merge it in late evening. However still able to make changes after that. However we want to see how many pages it takes in latex.

Sprint:
Pavel leads the work on Pipeline, calls for Max and Swen for help.
Kiyo leads the work on Keras evaluation by callback to tensorboard. Put misclassified images into tensorboard. Remake tfeval to be compatible with Keras.
Ong and Matthew work on getting the ten scans.

3 Priorities:
Get 10 products 3D models.
Finish Pipeline
Get evaluation matrix in Keras and other callbakcs logs

Todo:
Everybody make it shorter. Just focus on what was the problem, why it was problem and how we approached it
Swen add Essential vs non-Essential labels to requirements
Kiyo Swen combine tensorflow and keras sections UT
Pavel combines RandomLib and SceneLib
Pavel add intro to Unit test about how we dont focus on Robustness and performance as we have research project.
Matthew will change the system testing table
Max: Add paragraph about why our coverage is so high. We have only few branches and only not many small functions
Kiyo moves to Latex

G.2.14 07.03.2018

07/03/2018 minutes
present: everybody
13:00-14:00
Agenda:
standup
Prepare meeting with Bernhard

standup:
Swen working on keras/setup retrain.py
Kiyo should merge his branch back into Swens
Ong: Tensorboard, Blender

Scanning of the products:
Speedup by two people taking the images
Need two tripods from CSG
Pavel and Max will bring cameras
Ong might want to run experiments over exams if it is automated enough

Bernhard meeting:
What do we show him:
Show him:
Quick demo of the render pipeline

Explain the two part pipeline
Show him tensorboard
IPhone App
Ask him:
Scaling problem
Why image size influences conversion (Matthew)
Introduction of background class (to decrease confidence of wrong classes)
Introduction of dropout?
What about using Mobile Net instead of Inception? (Losses some accuracy for time) (lower priority as our model is quite fast)
Extensions:
Grid search or Random search for hyperparameters
Iphone App, add some more features, lilke a buy button that takes you to Ocado website.
Video of going into sainsbury, scanning a product and finding a better priced on Ocado.

G.2.15 08.03.2018

Meeting with Bernhard 11AM
Thursday 8th March 2018
Present: All

Reports
The final presentation is the most important element of our mark. Bernhard will mark it but this will be moderated by other staff.
We need to demonstrate a significant coding component (i.e. that it is not just trivial putting together pre-existing software)
System might work fine as it is, but for report/marking we need to sell/present as a piece of software.
Look for areas where we can demonstrate coding. APIs, documentation.
Over document/sell our code.
Show that our development process is efficient.
Discuss what was the most challenging coding part?
Present our solution.
Quantify the gains we made over our original approach.
Discuss limitations of our approach.

Pipeline Integration
B suggested generating images on the fly and passing directly to training:
Input would be obj file
Would sacrifice training time for memory gains
B agreed that keeping them separate fine but still need to show proper integration

To satisfy the coding element:
Implement APIs to pass data between components
Calculate rendering time, space requirement etc
Document and quantify/justify the decision
We need to sell our pipeline as single piece of software
Could be a script/web frontend, server API
Needs to be single input/output for client - obj file input and model out
Let user specify parameters easily
Well documented

Training:
B reiterated that our focus is on data, not model comparison.
We will still need to do lots of parameter tuning
Could use other frameworks (B suggested resnet as a second)
Make an ensemble, which could consist of different frameworks or multiple Inceptions with different parameters

Data generation:
Parameters
Sample model focal distances
Make use of rendering parameters where we can
Can tune parameters to particular environments. We should use a systematic approach:
  Outdoors
  In lecture theatre etc
  In a bin (blurry, so maybe put a bit of motion blur rendering?)

  In our report we should document locations, results, what we did about it.

Backgrounds:
Could introduce a bias field into background to simulate a strong light on one side. Separate background scenes.

Parameter Tuning:
In response to Ongs question about tuning hyperparameters, B suggested simple gradient descent. Use an ensemble. All models work well enough so dont spend too much time on it.

Final product/Demo:
B very happy with supermarket price comparison use case. Could give Ocado better data on customer needs.
Could optimize for supermarket lighting etc.
Make a video.

G.2.16 28.03.2018 Skype meeting

Meeting agenda:
Sprint meeting
Tasks relating to last sprint
Max to open merge request on pipeline after tidying up
Pavel to ensure separate pipelines are ready, confirm training pipeline details with Swen.
Max to meet with Ong to do scanning Wednesday afternoon. Max will book a room. Kiyo will let us know where the products are.

The report:
We will focus on the demo for now, but Kiyo will assign some points for people to think about. Wont start writing yet, as we will need to talk about the work we are about to do.

Next sprint:
Images to be cropped/resized so they dont need to squished
Matthew will make a square overlay for iphone app
Matthew will provide additional test images from flask

Tasks allocated in the other Google doc. First named person to create GitLab issue.
How to do stand-up (fix dates and time) that works for everyone
Standups will be slack chats every few days. Kiyo will announce dates.
We will have one Skype chat per week?
Prepare for Ocado demo and final presentation
Kiyo will speak to the other group to find out what we need to do
Matthew wants it to be perfect
Discuss closer to the time
How to delegate tasks remotely

G.2.17 08.04.2018

Agenda for next meeting:
Stand-up and go through issue boards.
Discuss the result of the first experiment (hopefully).
Briefly discuss what we need to do between after exams and final demo
Discuss what to prepare for the demo at Ocado (short presentation)

Meeting 10:00 08/04/2018
Present: Everybody

We have Slack notifications when errors happen during training/rendering. Sends a message when an exception is raised. Error handling was done extensively so now the code actually breaks. Need to make sure this is done properly throughout the code. Image cropping on the warehouse images. Cut images down to square. Some problems where there are not four consequential images. Flask app does not crash anymore, but there might be some hardcoded stuff still. But we will probably keep that. App is stable. Add buy on Ocado
Warehouse test set was taking lot of space, was moved from Manhattan project. It was probably the only copy on group folder. Ong can put it back up by tomorrow.

Swen written a function that takes in all parameters we can tune. Optimization was interrupted by a memory problem on GPU. Optimization is running again now. Swen will do a write up about what he is doing. It is problematic as ten class are taking a lot of time to train 1-5 hours per evaluation. We will wait for the first results to come back to see how good it is, before making any further decisions. Then we can discuss what we need to change. Swen will draft up the strategy and we will comment on it.

Logging experiments:
Every run is stored as one line in csv file, with all the parameters as collumn.
We will save both trained model and data into bitbucket automatically if possible.
Current validation accuracy is about 56% but it should improve a lot.
On Friday we managed to generate 200K images. Good stress test.

There is a draft of the final report. The main work on it would be done after the exams. Right now we should focus on UT, documentation, cleaning up the repository.
CTO came to Peters group visit so we should prepare properly.
Kiyo created a draft for the presentation.
We will split the presentation into sections where each will talk about certain aspects:
Pavel/Kiyo/Ong: Introduction, how we formulated the problem, why we did that,...
Ong: The actual technology

Swen: Training+ optimization+ results+ evaluation
Matthew: API+ app
Problem, Solution, Results, Product, Future development as presentation structure
Show warehouse images next to ambient ones to show that it is almost impossible to generalise from warehouse to ambient.
Use lots of pictures, optimistic graphs. Systematic error with warehouse images.
Prepare presentation by next sunday call.

To do:
Everybody: Do unit tests, Do error handling, Presentation preparation
Swen: Write up the testing strategy.
Matthew: Will have a look at the Keras UT

G.2.18 14.04.2018

Meeting 14/04/2018
Present: Kiyo, Max, Swen, Pavel, Matthew, Ong
Agenda for this meeting:
Discuss the result of parameter tuning
Presentation check
Warehouse accuracy
Indoor vs outdoor scenes for image rendering
Train ticket

Parameter tuning:
There was problem that we had to relax some inception layers, which should not be necessary. In Tensorflow it was not necessary, in Keras it would not converge.
With only top layer the best accuracy was 55%.
More improvement should come from render side. The network is probably slowly approaching its maximum. Some overfitting was spotted when relaxing Inception layers.
Conclusion: Some issue we do not understand and we will have to find out what is happening. However we are still getting a good model and retraining the whole network is not potentially a problem as we can generate very large number of data to train a deep network.

Another parameter to fine tune: Number of layers to relax
Swen has a script that can automatically change that parameter
Matthew has a script that allows to change parameters during training.
Network should be fine, we should optimize data generation

Highest accuracy: real images 83% Main problems: Small images, images from bottom, viewpoints we do not cover too much in the object pose generation
Region based CNN could improve our method and should not be that hard to implement?
A bounding box around the product and nothing else.

Warehouse accuracy:
Ong will put the dataset on bitbucket.
Use current model on the warehouse dataset.
We need to crop the warehouse dataset first, otherwise it will be squashed, as the aspect ratio is wrong. This will lead to very poor results.
We have a script that locates the product and crop the image into a square image.

We might need to brighten the warehouse images or darken the training images to improve accuracy. We are missing warehouse images for one of the products of the ten. It is Liberty that we are missing. Maybe they did not get downloaded correctly, so they might be somewhere.

Indoor vs Outdoor scenes for image rendering
We have separate sets, but they are small. Manually can increase the size, but would require some manual work.
Lets not worry about this for now. If we need to squeeze more accuracy we might consider doing this.

Presentation Check:
Modelling loss 11% The difference between validation and test set
Network loss 6%
We will probably get very low number on the warehouse dataset. We should convince them that after optimization this value can improve dramatically. Stress out that we are not yet done with rendering optimization. We have lots of ideas for improvement but not enough time computational power.It would be nice to check the demo before. Products are with Kiyo. Matthew needs photos of the products to test it on.
We will practice the presentation on Wednesday before meeting with Bernhardt.

Train ticket:
We will discuss with Bernhard if he is going with us.
Then we will either buy a group safe or if there is no group discount, we will use railcards and individual tickets.

Todo:
Max will look for Liberte images
Ong will upload the warehouse dataset
Matthew will crop it
Ong will run Keras Eval
Pavel Liberty in space for Swen presentation
Matthew will add rejection region to the app
On Wednesday film a video of the demonstration in case something brakes
Max will add some price comparison slide
Everyone Update the slides
Meeting at 3pm on Wednesday to train the presentation

G.2.19 11.05.2018

Minutes 11/05/2018
Present: Kiyo, Pavel, Matthew, Swen, Max, Ong
Agenda:
Check the remaining tasks
Assign the final report
Schedule for the next 5 days

We have a draft for the report with structure of the report We have an UML diagram + content of first report
Take the nice design image and make it more specific. E.g. specify that this step is done by this library, this is done by this software package.
Then pick a good core part, and do an UML for this. Probably do the Blender API.

Lets try and answer every question in the group project specification

Specification:
What was meant to be a first report
Table of specification, show of extension versus base product

Design: What we wanted to do. Top level concept: Data generation, Data processing, Classification, Evaluation
After that talk about the design choices of each component

Early design/background research. Motivation for methodology. Mention trying out barcode scanning, labels recognition, using GAN, augmentation, Spatial invariance, one shot learning, Tensorflow vs Keras. Main design consideration was using 3D modelling rather than using their data. Why we picked CNN.

Methodology
How we do what we decided to do. Mostly report two.
Unit tests, we are going to need to make them work. Otherwise the content of report two should mainly cover the text. Schedule needs to be updated.

Final product:
What we actually achieved
The biggest section, all the results, all that we manage to accomplish, We have delivered on what we promised. We should highlight the extensions we did: App, Detection, Also highlight this in the spec.

Need to be careful about overlaps. For example between Design/ final product

Discussing what we still need to do. Talk about how it does not perform that well on Ocado dataset. Lets however try to improve it in remaining time.
Lets do grid search for learning rate, relaxed layers. With rendering parameters we can just discuss that we went with a general distribution and it proved to work.
There is one integrated pipeline. E.g. one file with two function calls that run everything from object files to trained network and evaluation.

Who will do for the report:
Ong and Swen have some coding tasks still to do.
Put stuff from the presentation and the other two reports into the final report
Kiyo: Introduction (brief), Specification, and Group work
Matthew: Design (oversee)
Max: Methodology(oversee) Pavel: Logbook
Ong, Swen: Final product

To do:
Everybody clean their code and fix UT
Proper documentation. Mainly for interface functions, which might be expected to be seen by the marker. E.g. the function used to actually run the pipeline.