UiT The Arctic University of Norway

Faculty of Engineering Science and Technology
Department of Computer Science and Computational Engineering

Automatic Generation of Custom Image Recognition Models

Magnus Fredheim Hanssen

Master thesis in Applied Computer Science, Narvik, May 2022

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”
–John Woods

“Without requirements or design, programming is the art of adding bugs to an empty text file.”
–Louis Srygley

Abstract

In this thesis, we examine the viability of training a convolutional neural network using synthetic data. The CNN is used for image recognition in an RMS. We create a program that can render 3D models from STL files as images with varied backgrounds. The process of creating and training an image recognition model is also automated. Lastly, the model is used for image recognition.

The report compares different methods and hyperparameters used in training a model. Transfer learning is found to be suited for synthetic datasets. Using the pre-trained feature extraction layers of the VGG-16 model, we train an image recognition model with better than 90% accuracy in the laboratory. We then demonstrate the use of this model for object detection, and suggest avenues for further development.

Acknowledgements

I would like to thank my supervisor Prof. Bernt Arild Bremdal, and co-supervisors Shayan Dadman and Halldor Arnarson for good advice and feedback over the course of my thesis work. Special thanks goes to Halldor for proposing the task, which proved to be both interesting and challenging.

I would also like to thank my classmates Tanja Henriksen, Casper Levoll-Steen and Markus Tobiassen for years of steadfast companionship, as we faced the challenges of higher education together.

Lastly, I thank my family for their love, support and regular reminders to get back to work. Without them, this paper might only have been five pages long.

Contents

List of Figures

List of Tables

List of Abbreviations

API	application programming interface
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
CNNs	Convolutional Neural Networks
DIE	Department of Industrial Engineering
IDE	Integrated Development Environment
JDK	Java Runtime Environment
JVM	Java Virtual Machine
NMS	Non Maximum Suppression
PIL	Python Imaging Library
RMS	Reconfigurable Manufacturing System
ROIs	regions of interest
VTK	Visualization Toolkit

/1 Introduction

This chapter describes the background for the fundamental project of the thesis, as well as the objectives of the project. It describes the scientific query at the core of the thesis, and gives a brief introduction to some of the central fields and disciplines for of the projects. Later sections explain existing technology, and go on to describe an investigation into the state of the art in the field. It is on the basis of these factors that we formed a strategy for the completion of the project, which is described in section 1.5.

1.1 Background

In the machine laboratory in Narvik at Department of Industrial Engineering (DIE), Halldor Arnarson is working on a Reconfigurable Manufacturing System (RMS). This system consists of a number of wheeled platforms carrying conveyor belts, 3D printers, robot arms and other parts used in manufacturing. The goal of RMS is for the manufacturing system to be responsive to changes in the market and the demands of customers, by enabling rapid and seamless scaling and convertibility.

As can be seen in his video of the system[1], Halldor's system uses a mobile robot to move the platforms that make up the RMS. His goal in doing this is to have the reconfiguration of the system be completely automatic. A stated goal is to be able to change which product is manufactured without physically interacting with the system. Such a degree of automation would enable the user to start production of different products from a remote location, and it could also enable scheduled reconfiguration whenever a production run is complete. This is desirable in that it can reduce downtime and required human labor.

One of the major challenges the system faces is that the *MiR100*, the mobile robot moving platforms around when re-configuring the manufacturing system, is not very precise in its placement. Platforms will not be placed with consistent relative positions, so any process crossing between platforms will have to account for this imprecise positioning. A proposed solution for this problem is the use of computer vision.

Halldor wishes to use robot arms with depth cameras for picking and placing different components along the assembly line. Adding this capability to the RMS would greatly decrease the amount of human intervention required at certain steps of the manufacturing process. More specifically, Halldor wishes for the robot arms to be able to recognize and pick up 3D printed parts with the use of machine learning and image recognition. Such parts will commonly be unique to a product however, and might not appear in any databases of images useful for training an image recognition model. Manually photographing printed parts in different configurations would be extremely tedious work, and it would take a long time to manually produce representative samples of every different part.

When 3D printing a part, a 3D model is used as the basis for creating instructions for the printer. Using Computer-Aided Design (CAD) and the resulting 3D models for CAD machines like lathes, plasma cutters and routers has been the industry standard for decades[2]. The ability to use 3D models from STL files for training image recognition could then allow manufacturers to use robots for picking and placing parts using computer vision.

1.2 Objective

As stated in section 1.1, we want to automate the RMS. To achieve this end, we wish to use an image recognition system in conjunction with robot arms to automatically locate and pick up 3D-printed parts. The task given was then to create a system capable of automatically generating a custom image recognition model from a 3D model, for the purpose of using said model to locate the corresponding physical 3D printed part in the world. This objective was stated in the task description as follows:

1. Create an image recognition system based on state-of-the-art machine learning models that can be used for the purpose described above (see appendix A for full text)

2. Create a basis for training and testing by means of synthetic images using 3D models generated by the CAD system used to design and produce the parts to be recognized and classified.

The points enumerated above describe the concrete goals for the project. The scientific goal of the project is then to examine the viability and practicality of the system described. To this end, using the image recognition model for object detection is also a goal of the project.

1.3 Theory

The project for this thesis has as an objective to generate an image recognition system using state-of-the-art machine learning. This will require use of computer vision, and so this section will provide a brief introduction to the theory behind computer vision, machine learning, specific machine learning concepts used for the project.

1.3.1 Computer vision

Computer vision as a field can be said to have two related goals; to find a computational model for the human visual system, and to create a system able to perform some or all of the tasks the human visual system is naturally capable of, according to T. S. Huang[3]. Those two goals seem to approach the problem from different directions, one seeking to emulate existing systems, while the other seeks to reach parity in achievement. For the purposes of this project, "computer vision" will refer to the latter approach, as the project revolves around the creation of a computer program to take on a task previously done by a human. Computer vision is commonly achieved by use of machine learning and Convolutional Neural Networks (CNNS). Those concepts are explained later in this section.

There are a number of different problem areas within this approach to computer learning. One such area, text-parsing, can be considered a largely solved problem, as demonstrated by the product *Google Lens*[4]. Google Lens is capable of detecting and parsing both print and handwriting. Another problem area for computer vision is the mapping of 3D spaces, of particular interest for developing technology like self-driving cars[5]. For this use, a system must be capable of rapid and accurate detection of obstacles and boundaries on the path ahead, as mistakes can prove fatal when moving vehicles are involved.

1.3.2 Machine learning

Seventy years ago, in 1952, Arthur Samuel at IBM developed a program that could play checkers[6, p. 22]. This program had the ability to learn from past games, and to classify a potential move as good or bad, given the current situation of the board. Samuel went on to coin the term "machine learning", and defined it as the "field of study that gives computers the ability to learn without being explicitly programmed." This approach proved quite revolutionary, as Samuel had thus proven that a computer could find solutions to problems beyond the ability of humans to explicitly program for. In 1996, the game of checkers on an 8×8 board was weakly solved by Jonathan Scaeffer[7], but he roughly estimates the solution space to consist of 10^{17} different positions, and so a strong solution is considered prohibitively expensive. This contrast between the ability to solve a game and the ability to consistently outperform humans illustrates one of the benefits of machine learning.

As described in M. I. Jordan and T. M. Mitchell's 2015 review[8], machine learning as a discipline focuses on two related questions; how to construct a computer system capable of improving itself through experience, and which laws govern all forms of learning, be it for computers, human beings, organizations and so on. In this definition, we can see a clear parallel to the field of computer vision as described in section 1.3.1. Jordan and Mitchell goes on to state that machine learning has emerged as the technology of choice for developing computer vision, speech recognition, natural language processing and other applications. This makes intuitive sense, as the human brain, and the human capacity for learning, is quite adept at understanding visual input and the spoken word. Natural language is of course defined by naturally evolving through use by humans. When a task is believed to require a certain "human touch", a sophisticated machine learning model can often be a good substitute.

1.3.3 Artificial neural networks

An artificial neural network is a network of nodes modelled after neurons. Teuvo Kohonen, in his 1988 paper in the first issue of the *Neural Networks* journal[9], defined artificial neural networks as follows:

> "Artificial neural networks" are massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchi-

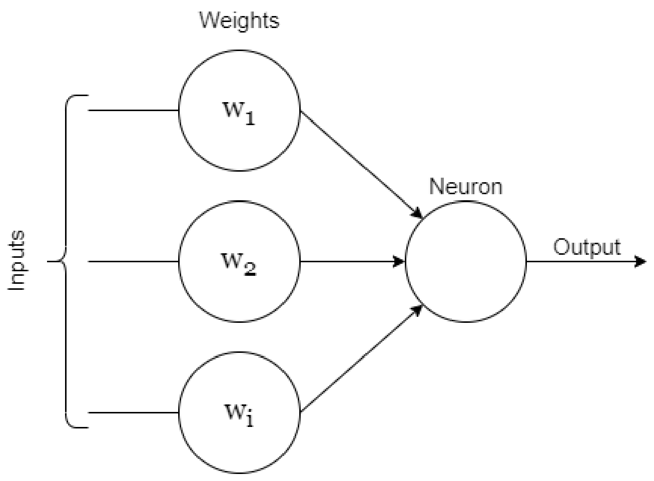


Figure 1.1: A simple diagram of an artificial neuron.

> cal organizations which are intended to interact with the objects of the real world in the same way as biological nervous systems do.

The article goes on to state that there are on the order of 10^{11} neurons in the human body. The number of interconnections between those neurons is thought to be on the order of 10^{15}. On a basic level, thoughts are formed by signals travelling along paths between neurons, which activate and propagate signals at different thresholds, and due to feedback loops and nonlinearities, the neurons and the paths between them can change and allow learning.

Figure 1.1 shows a simple diagram of the layout of an artificial neuron. Such a neuron has several connected inputs, each one afforded a weight. In the commonly used McCulloch-Pitts model, as described in *Machine Learning*[6, p. 104], these weighted inputs are compared against a threshold, and an output is produced according to an activation function. A common activation function is the sigmoid function, which is continuous and smooth, and will squash inputs from any interval into the open interval $(0, 1)$. This can often be desirable, as it ensures that a single input of anomalous magnitude will not necessarily propagate through every downstream neuron. Another popular activation function is the rectified linear unit, $f(x) = \max(0, x)$, which ensures simple gradient calculations.

In fig. 1.2 can be seen a simple illustration of a neural network, with a total of 16 neurons forming a total of 60 connections. The layer in the middle is called a hidden layer, as it is not connected directly to the input nor the output. When training such a neural network, incremental changes are made to the weights of the different inputs in order to minimize loss. Neural networks can

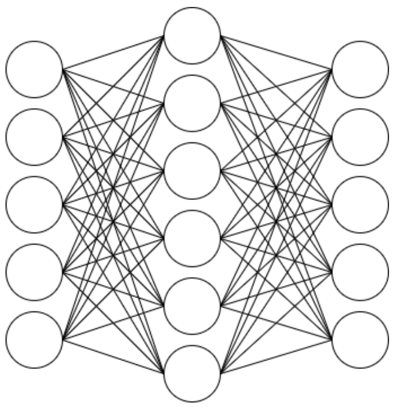


Figure 1.2: A neural network with one hidden layer.

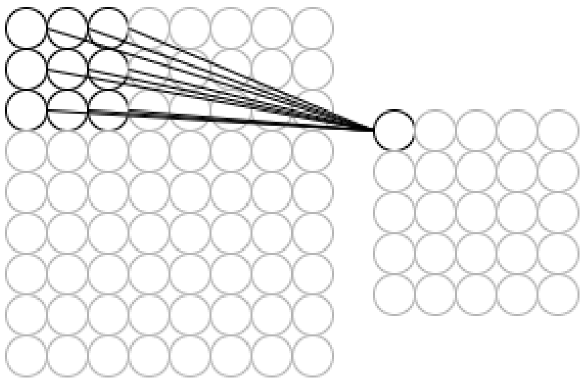


Figure 1.3: A convolution layer showing a region of the input mapped to a single neuron.

be trained using different optimization algorithms and different functions for calculating loss. The specific choices made for this project are discussed in Chapter 2.

1.3.4 Convolution and feature extraction

Section 1.3.3 briefly describes artificial neural networks, but in order to understand the use of deep learning in computer vision applications, convolutional neural networks must also be understood. The purpose of convolution is to give the network the ability to recognize important features. As explained by Jogin et al.[10], convolutional layers are useful in creating feature maps for different sets of data, and are better suited for this task than fully connected layers.

Figure 1.3 shows an illustration of a neuron being connected to a limited region

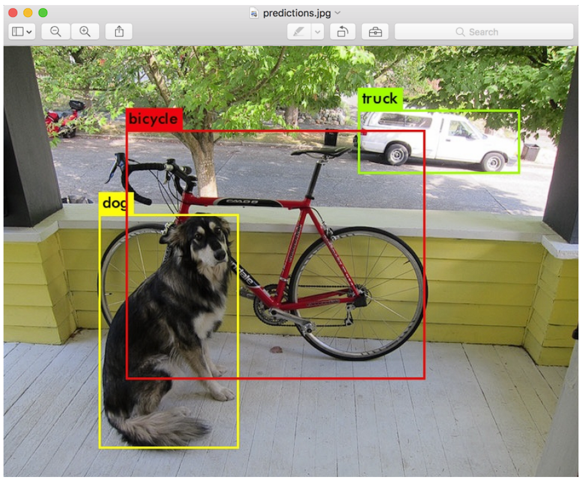


Figure 1.4: Example of YOLO predictions from Bandyopadhyay's article on the subject[11].

of an input image. Utilizing different kernels for different convolutional layers, running different filters over the input, allows different kinds of features to be discovered. In the feature extraction part of a neural network, convolutional layers are commonly alternated with max pooling layers, decreasing the dimensions of the input by only keeping the highest value for each pool. This helps abstract the input to prevent overfitting, and it also decreases the number of inputs for easier computation.

1.4 The State of the Art

A common problem in computer vision is to translate 2-dimensional images to 3-dimensional models, enabling a computer to calculate distances between objects, detect obstacles, follow roadways and so on. On his company website, Jérémy Cohen of Think Autonomous shows different ways in which computer vision is used to understand the space around self-driving cars[5]. Cohen shows how 3D information is extracted after first detecting an object in 2D photography. For such purposes, algorithms like YOLO[11] are considered state of the art, and can perform object detection in real time. YOLO is an abbreviation for *You Only Look Once*, and the algorithm proposes an end-to-end neural network that predicts bounding boxes and class labels in a single pass, hence the name. This contrasts with more traditional methods, where the classifier is typically separate from the model or system used to propose regions of interest.

An alternative to locating objects of interest in an image, and then extrapolating 3D geometries, would be to train a more traditional image recognition model based on synthesized datasets. This is the approach proposed in the project description. Investigating the topic suggests that machine learning using datasets synthesized from 3D models is quite rare outside certain, specific use cases. Jacob Sullivan describes the process of training the YOLOv3 model on 3D models of LEGO bricks[12], and tells of how he was inspired by Daniel West, who did much the same for his LEGO sorting machine. Indeed, a part of the course material taught by Prof. Bremdal in the course DTE-3606 demonstrated a simple image recognition model trained on such images of LEGO bricks.

A few researchers have experimented with synthetic datasets for other applications. In their paper, Sarkar et al.[13] showed an approach to generating images for training an image recognition model based on the *AlexNet* architecture. The training data was generated using the Visualization Toolkit (VTK) with a light affixed to the camera, and the camera orbiting the 3D model. The experiment shows that accurately texturing the 3D models used, as well as including textured backgrounds, both contribute to better recognition. They also found that cluttered background could give a number of false positives, and speculate that more chaotic backgrounds could yield a more robust image recognition model.

Another approach to the problem of generating training data is described by Heisele et al.[14]. In their experiments, they generated training data by rendering 3D models in Blender through the use of Python. The images were rendered in grayscale, without background or texturing. The image recognition model was then tested and evaluated on photographs taken of 3D prints of the models. This contrasts with the approach used by Sarkar et al., as they instead used scans of pre-existing objects, but falls closer to the nature of the problem examined in this thesis. It is worth noting that Heisele et al.[14] remarked on the difficulty of generating realistic training data. They also show that for pose-invariant object recognition, much larger datasets than are commonly used in image recognition are required for accurate predictions.

A similar task to the fundamental project of the thesis was previously attempted by Kumar et al.[15]. In this paper, Kumar et al. show methods for edge detection and the generation of bounding boxes, which are then used for the purposes of traditional image recognition. This approach does not use 3D models for generating training data, however. Upon reviewing the state of the art, it seems that using image recognition models trained with synthetic data to aid automation in robotics and flexible manufacture is a novel approach.

1.5 Strategy

The objective for the thesis project is to examine the viability of a program that generates training data, trains a model, and then uses the model to find parts in an image, as described in section 1.2. This goal can be achieved by implementing a system in accordance with the specifications, and then testing the performance of the system in order to evaluate its suitability. This section of the thesis details the three main steps to creating such a program. This section also briefly describes the strategy for experimental work.

1.5.1 Creating and training an image recognition model

One of the goals of the project is to create an image recognition model using convolutional neural networks and deep learning. This model is to be created and trained using the program developed over the course of the project. Requirements for this program are enumerated below:

1. The program must be capable of creating and compiling a model using a deep, convolutional neural network for image recognition

2. The program must be capable of training such a model using appropriate datasets

3. The program must be capable of saving and restoring models so trained

4. The program must be capable of evaluating and testing models using provided images

The plan for the completion of this step of the project was to investigate different architectures for suitability for the use case, and then to create a program as described above to be able to compile and train a model based on said architecture.

1.5.2 Automatic generation of datasets

An important step towards reaching the project goals is the creation of a program that can generate suitable images of an arbitrary number of 3D models with a minimum of input from the user required. A number of requirements for this program have been formulated as follows:

1. The program must be capable of loading 3D models from STL files

2. The program must be capable of calculating any number of viewpoints for generated images
3. The program must be capable of generating images of the loaded models from all calculated viewpoints
4. The program must be capable of creating suitable backgrounds for the models from photographs of relevant surfaces in the lab
5. The program must be capable of inserting such backgrounds into the images of the models
6. The program must automate the preceding steps for a selection of STL files

The goal of this step is to provide a dataset of sufficient quality to train an image recognition model capable of pose-invariant classification. As found by Sarkar et al.[13], sufficiently varied backgrounds are important to the robustness of a model.

1.5.3 Implementing object detection

As is the goal of the project, the final program must be capable of locating different object in an image. An image recognition model is by itself incapable of doing so, and will only return the confidence for a given image belonging to any given class. The final step of the project would be to create a program that brought together the components from previous steps, with the following requirements:

1. The program must be capable of taking an image with a connected camera
2. The program must be capable of detecting objects in an image
3. The program must be capable of classifying objects in an image
4. The program must be capable of returning the location and confidence for an object of a given class

1.5.4 Testing and experiments

A number of tests would have to be made to ascertain whether the project goal was met, and to achieve the scientific objective of the thesis. Some tweaks to the model, and the hyperparameters used, will be made, as well as changes to the pre-processing of training data. For those changes, training and validation loss and accuracy will be recorded and compared, and the results from an evaluation using real photography will be recorded. For the image detection system, the results will be displayed as bounding boxes, and be visually compared and analyzed.

2 Tools and Methods

This chapter describes the tools and methods used for the project. The first section concerns the different software tools and libraries used over the course of the project, and the rationale for selecting them. The second section describes how the program was made, with more concrete implementation details, and the decisions surrounding those. It also describes the process of testing and experimenting with the program.

2.1 Tools and Tool Selection

Prior to starting programming for any project, one or more programming languages must be selected. For this project, the programming language Python version 3.9 was used. While creating the object detection program, a number of different software libraries were used. The program uses *TensorFlow*[16] through *Keras*[17] for creating an image recognition model. *Python Pillow*[18] and *vtkplotlib*[19] were used for rendering synthetic training images, and later on *OpenCV*[20] with utility functions from the python library *imutils* were used for object detection.

Table 2.1: A brief comparison between relevant programming languages.

	Availability	Execution	Keras	TensorFlow	PyTorch
C++	Open	Compiled	No	Yes	Yes
Java	Open	Just-in-time	No	Yes	No
MATLAB	Proprietary	Interpreted	No	Yes	No
Python	Open	Interpreted	Yes	Yes	Yes

2.2 Programming language

When selecting which programming language to use for a project, several factors must be considered. For a project with a limited time frame, like this thesis, having to learn new programming languages would be a detriment. Secondly, the programming language chosen must have the capabilities required for the project. A different concern is the method of execution for a program written in the language, and whether such a program would require proprietary software to run.

Table 2.1 compares four programming languages that could feasibly be used for the project. *C++* is a powerful, general purpose programming language. TensorFlow 2 is built on a C++ back-end, and both TensorFlow and *PyTorch*[21], the primary deep learning software candidates as described in section 2.2.1, can be used with C++. C++ follows the standard described in *ISO 14882:2020*, and there are a number of free and open source compilers for it. This language was a strong candidate for the project.

Java is a programming language that uses the Java Runtime Environment on the host PC[22]. This environment consists of the Java Virtual Machine (JVM), core classes and supporting libraries. Programming in Java requires a Java Runtime Environment (JDK), but there are several open source or free versions. For programs contained in a single file, the JVM is capable of compiling human-readable code into byte code instructions as the program is running. This is called Just In Time compiling, and it allows the JVM to execute a program line by line as if an interpreter. This can allow for easier development, while retaining some of the advantages of compiled programs. As will become apparent, however, Java does not offer any significant advantages not found in C++ or Python.

MATLAB is largely included for the sake of completeness, and because it is a common choice of programming language for a number of engineering problems. The programming language is pretty specialized, and while there are a number of tool kits available for different applications[23], the risks of locking the project into an ecosystem of proprietary software seemed quite severe.

The final candidate language was Python, an open source, high level and general purpose programming language. Like C++, Python can use TensorFlow and PyTorch, but Python can also use TensorFlow with Keras. Python follows a different philosophy to C++; whereas C++ was developed oriented towards embedded systems with a focus on performance and efficiency, Python has different design principles as related in “The Zen of Python”[24]. The programming language is designed to be simple, sparse, readable and obvious. The ability to execute a program line by line was also considered a positive, as the ability to effectively debug and tweak the program was considered more important than achieving the highest possible degree of efficiency. The *pip* package installer also makes installing packages for Python easy and quick.

For those reasons, Python was chosen as the sole programming language for the project, as all project requirements should be within the capabilities of a Python program. Python version 3.9 was selected because at the time, it was the latest version of Python supported by Keras, and it was suspected that other packages might not be updated for 3.10 and later versions. The specific version used was Python 3.9.4, as that was the version installed on the computer used for development, but the specific release number should not have an effect on the execution of the program

2.2.1 Deep learning software

When selecting which software to use for the deep learning portion of the project, there were only really two options that seemed to merit closer consideration; TensorFlow 2 and PyTorch. A description of the TensorFlow interface and implementation can be found in the 2015 paper “TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems” [25]. TensorFlow is commonly used with Keras, an API built on top of TensorFlow 2, meant to make interacting with the TensorFlow platform easier and with fewer user actions. TensorFlow is used by a number of companies, like Google, Twitter, PayPal, GE Healthcare and Intel[16]. It is a popular platform for many applications.

PyTorch is a library for deep learning and neural networks that has seen increased use and activity in later years. As can be seen on the PyTorch GitHub page[26], the frequency of commits to the project increased massively around 2016. A sentiment frequently heard is that PyTorch is a very good, up-and-coming platform that nonetheless lacks some of TensorFlow’s matured functionality. Unlike TensorFlow, PyTorch was built to be deeply integrated into Python. Some design principles seem to echo the previously mentioned Zen of Python, with a focus on intuitive and easy use.

Whereas TensorFlow requires a model to be built and compiled before training can start, PyTorch enables the user to make changes to the model graph arbitrarily. This could potentially be useful for rapid experiments using different models. TensorFlow is more mature than PyTorch, and more guides and articles about deep learning and neural networks using TensorFlow are available. It is difficult to say definitively which platform is better than the other. In the end, the choice of TensorFlow with Keras was made due to its ubiquity and my previous experience with the platform. Defaulting to a known and tested platform seemed the better choice, in this case.

2.2.2 Image rendering and manipulation

The initial plan for rendering the 3D models was to use VTK, an open source toolkit for displaying and manipulating scientific data[27]. In addition to displaying scientific data, VTK is capable of rendering 3D models and saving the image to a file. Blender was considered for rendering images, but was rejected as it is considered less lightweight and practical than a Python package of more limited scope. When browsing example code and guides to learn about VTK, however, it was decided to use the vtkplotlib library, as it is built on top of VTK to wrap the "ugliness" into friendly and clean functions[19]. This library seemed easier to use, and more in line with the Zen of Python. Following this philosophy usually helps with playing into the strengths of Python.

Python Pillow is a fork of the Python Imaging Library (PIL), which was previously discontinued. It is considered one of the standard Python libraries for manipulating images, and was chosen for its ability to apply an alpha mask while inserting one image on top of another and resize the resulting image. OpenCV was later used for the object detection program, as it is perhaps the most commonly used Python library for extracting images from a web camera, and was otherwise convenient for creating image pyramids, sliding windows and displaying the results of the object detection. The *imutils* package provides some convenient functions to use for computer vision alongside OpenCV, and provided a function for non-maxima suppression. The tools described in this paragraph were not decided on during the preliminary project, but were instead investigated and used as the need arose.

2.3 Methods

This section relates how the functionality described in section 1.5 was implemented. The development of the program is divided into three different phases, each of which are described in their subsection. The section also de-

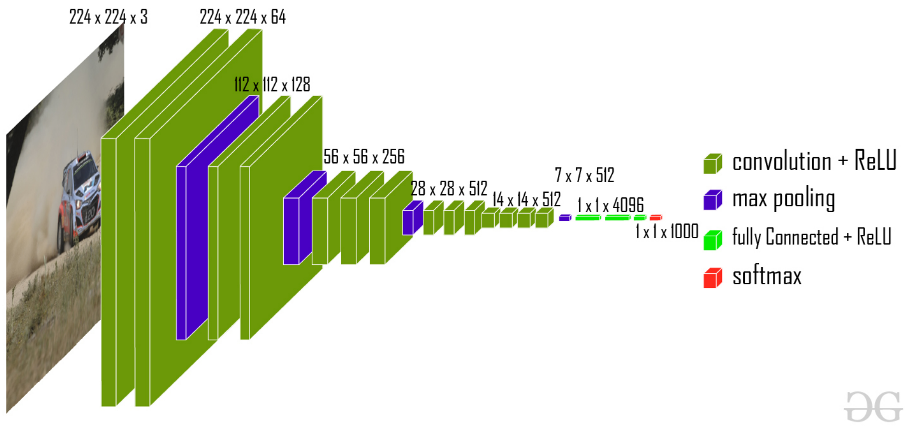


Figure 2.1: The VGG-16 architecture, illustration from GeeksforGeeks[28].

scribes the process of testing the resulting program, and the experiments that were run.

2.3.1 Creating and training an image recognition model

As told in section 2.1, TensorFlow with Keras was used for image recognition. Two different approaches to building a model were used. In general terms, an image recognition model will consist of a deep, artificial neural network with many layers in sequence, and there will be two main parts of the network. Those two parts are the feature extractor and the classifier. It is possible to train an image recognition model on a series of images, and then use only the feature extractor portion of the model. The resulting feature extractor will then detect a series of features generally useful for image classification. It is possible to attach a trained feature extractor to an untrained classifier, and then only train the classifier on new data. This is called transfer learning, and it can be especially useful when training data differs from testing data[29].

Figure 2.1 shows an illustration of the VGG-16 model proposed by Simonyan and Zisserman[30], a very popular model for transfer learning. As the figure shows, this model has an input layer capable of accepting 224×224 RGB images, and can classify 1000 classes. The model consists of a total of 14 convolution layers with 4 max-pooling layers interspersed, connected to 3 dense layers and an output layer. For this project, the pre-trained VGG-16 model was downloaded without the "head", the four last layers responsible for classification, and the weights were frozen. On top of this model, a global max-pooling layer, a dense layer using ReLU, a dropout layer and an output layer using softmax as the activation function for the output layer.

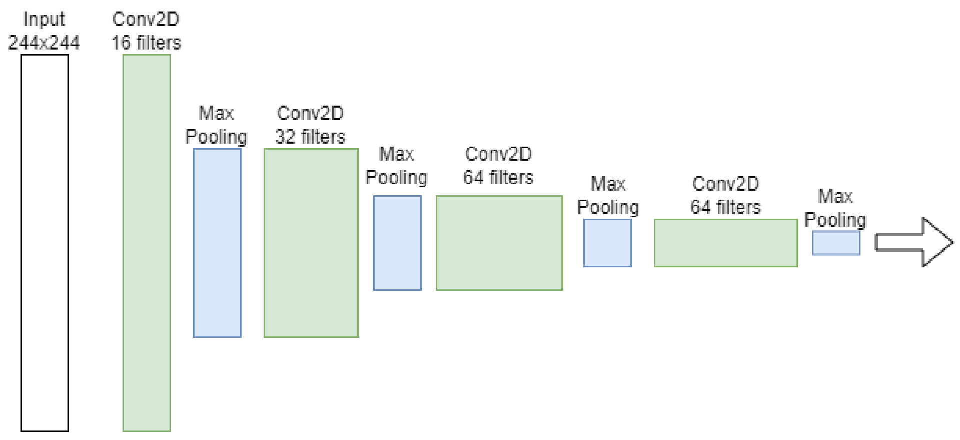


Figure 2.2: Illustration of feature extraction segment of model trained from scratch.

Another approach to creating an image recognition model was also tested. A model was trained completely from scratch, without using transfer learning. The feature extraction layers used for this model are illustrated in Figure 2.2, with four convolution layers, and max pooling reducing the dimensions of each subsequent layer. The number of filters in each convolution layer is increased as the dimensions of the input decreases, to avoid creating too many neurons. As the head of the classifier, four dense layers with dropout in between were used, with a fifth dense layer for the output. For all of the layers except the output layer, the ReLU activation function was used. The output layer used the softmax activation function. This model was adapted from an example shown in one of Prof. Bremdal's lectures[31].

Several models were trained using both of the models above as starting points. They were trained using the images generated by the process described in section 2.3.2, with varied pre-processing techniques and hyperparameters used, as described further in section 2.3.4. The program created for this part of the project allows the user to easily create new models with changed parameters, and it allows the user to save and load those models at a later date.

Using the Keras ImageDataGenerator to split datasets into training and evaluation sets, it is also possible to apply pre-processing to images in real time as they are streamed from a directory in batches. Such pre-processing can also be built into the image recognition model as a pre-processing layer, but for this use case, keeping the pre-processing separate has some benefits. Theoretically, the pre-processing can be completed asynchronously and on otherwise idle cores for better training speeds on CPUs, but this is not immediately relevant nor even noticeable. More relevant for this experiment is that it allows for easy changes to the pre-processing steps without making changes to the model structure.

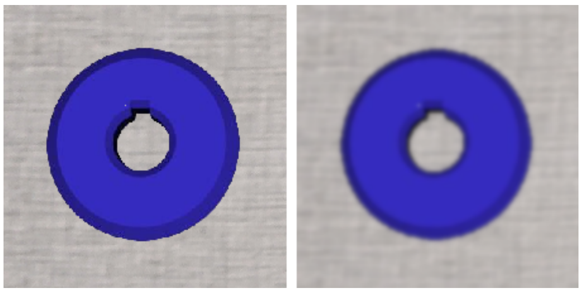


Figure 2.3: Lift turn before and after Gaussian filter pre-processing.

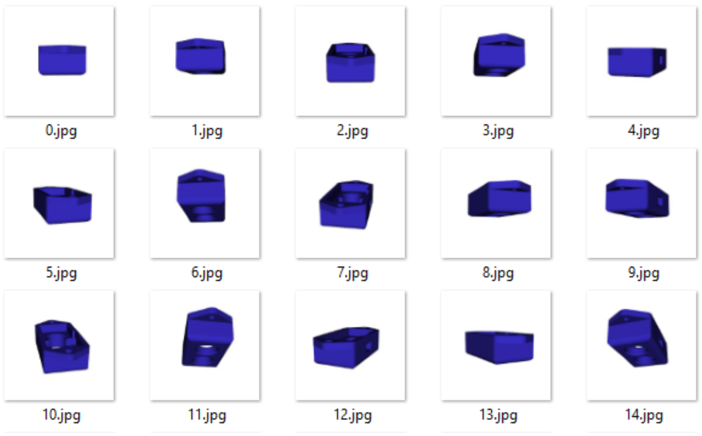


Figure 2.4: “distance_sensor.stl” rendered from different angles.

An example of such a pre-processing step is shown in fig. 2.3. A Gaussian low pass filter was tried to reduce the impact of rendering artifacts and overly sharp transitions between model and background not found in photos from the laboratory. A concern when training the image recognition model was the possibility of overfitting the model to the point where it would only recognize synthetic images. To this end, several pre-processing techniques were attempted as described in section 2.3.4.

2.3.2 Automatic generation of datasets

In order to generate sufficiently varied points of view for the rendering of the 3D model, the Fibonacci sphere method was used. This method creates an arbitrary number of equidistant points on a sphere. Using vtkplotlib, a configurable number of images of all provided 3D models are rendered and saved to a folder. Some of the results of this process can be seen in fig. 2.4.

To introduce some entropy to the dataset, and hopefully get more robust models

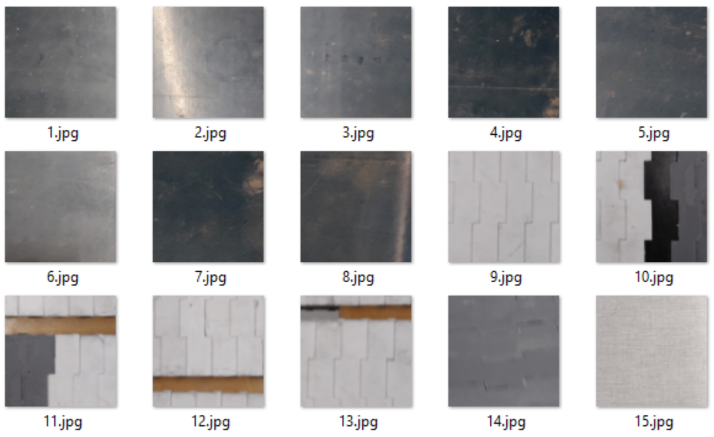


Figure 2.5: Most of the background photographs used for the project.

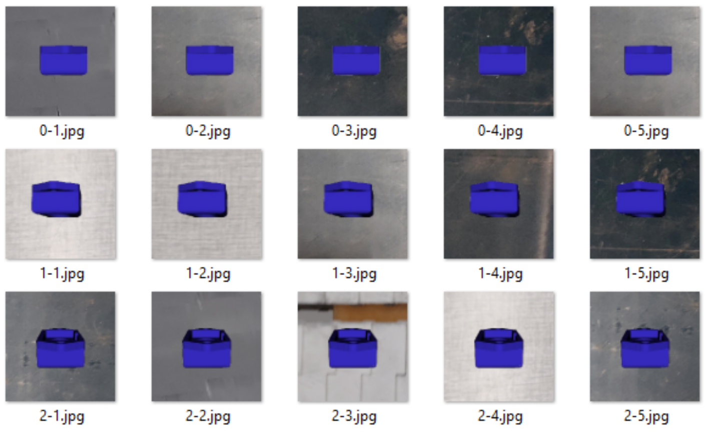


Figure 2.6: The rendered models combined with backgrounds.

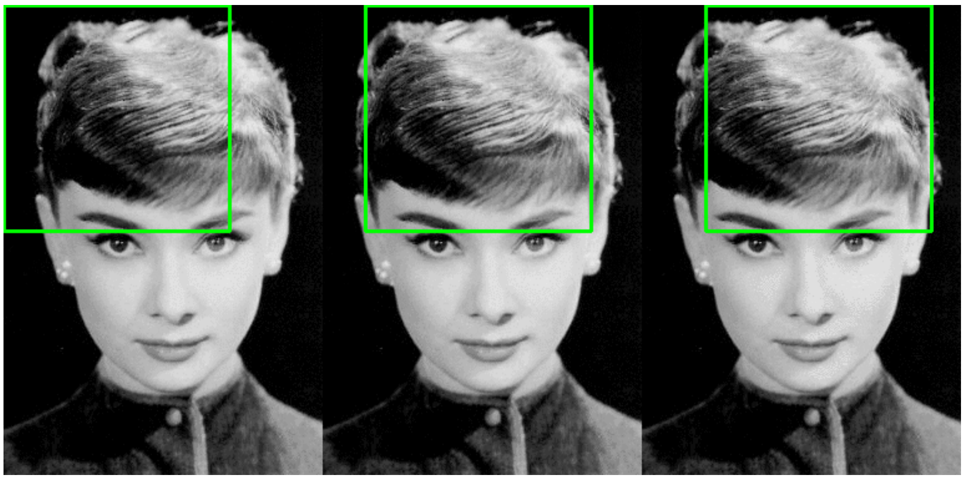


Figure 2.7: A window sliding over an image, adapted from blog-post by Dr. Roesebrock[32].

less prone to overfitting, Python Pillow is used to automatically insert segments of photographs of the different tables and conveyor belts in the laboratory. As indicated by Sarkar et al.[13], the incidental clutter in the background is prone to cause error for image recognition models trained with synthetic datasets, and introducing as much variation as possible in the training is important. Most of the background photos used in this process are shown on fig. 2.5. Each original image is put on top of a number of different, random segments of background and saved as 224×224 images. By default, this yields 200 different viewpoints each repeated across 5 random backgrounds, for a total of 1000 images per class. The result of this process is shown in fig. 2.6.

2.3.3 Object detection

In section 1.4, the YOLO model was described. This model stands out from its competitors with its deviation from what has been considered common practice in object detection; You Only Look Once means that the model proposes regions of interest (ROIS) and classifies them in a single pass. Other object detectors will commonly use a small and fast neural network to propose ROIS, and then send those to the full image recognition model to label. It is also entirely possible to drop the region proposal step of the process and exhaustively search the image using the image recognition model. It is the latter approach that has been implemented for this project.

Typically, objects will appear at different sizes in photographs, as the distance to the camera decides apparent size. With a camera mounted to a robot arm, the distance is considered variable. This means the object detector must

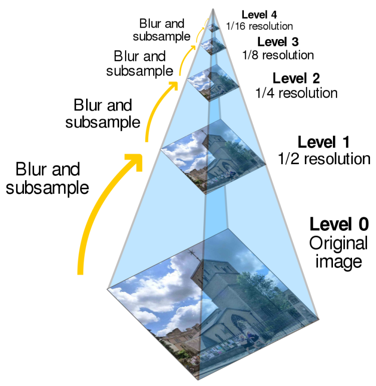


Figure 2.8: "Visual representation of an image pyramid with 5 levels" by Cmglee shared under CC BY-SA 3.0.

be capable of detecting the objects at different sizes. The parts will also be spread around the image, hence the need for object detection. To this end, two techniques will be used in conjunction to generate possible ROIS for the image recognition model to classify; the sliding window approach combined with an image pyramid. This process is further detailed in a guide by Dr. Adrian Rosebrock[32], which was followed in part over the course of developing the image detection program.

The sliding window technique consists of creating a series of cropped selections corresponding to the contents of a window sliding over an image, along the width and height of the image. Such selections can be seen in fig. 2.7. When using sliding windows, a stride must be selected to determine how far the window will move for each step. Large strides will result in fewer ROIS, and thus faster computation, but this will decrease the precision of the object detector.

An image pyramid consists of a series of images sub-sampled from an original image. An example of this technique using a down-scaling factor of 2 is illustrated in Figure 2.8. Running sliding windows of constant size over several sub-sampled images allows image recognition for a series of different scales, without having to change the size or scale of the output of the sliding window. Like with the stride of the sliding windows, a small down-scaling factor will lead to greater granularity at the cost of more computation. This effect compounds, with exponentially higher memory and computation costs if both the step size and scale factors are decreased. Default settings of a step size of 32 pixels and a down-scaling factor of 1.4 were used for the object detection program.

With a number of ROIS classified and labeled, the result would have a number of partially overlapping bounding boxes for each object detected. This would

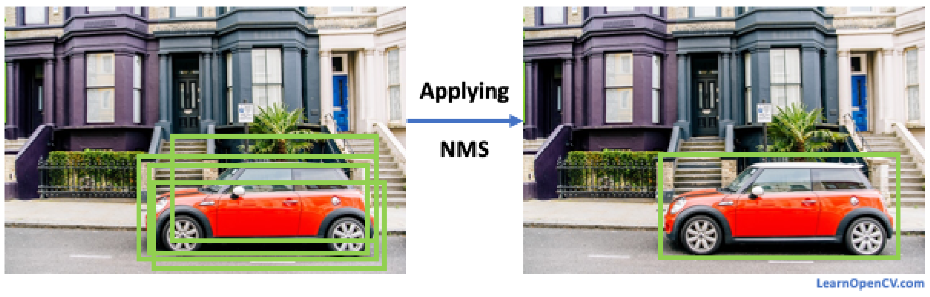


Figure 2.9: NMS illustration from LearnOpenCV[33].

effectively count each part several times, and would make the object detection unusable for its intended purpose. To counteract this, Non Maximum Suppression (NMS) is applied to the bounding boxes. This algorithm only keeps the bounding box with the highest confidence among a set of overlapping boxes. The result of the process is illustrated in fig. 2.9.

2.3.4 Testing and experiments

In order to evaluate the ability of the program to automatically train image recognition models, a series of models were trained and evaluated. For this experiment, the model trained from scratch was primarily used, as it generally proved faster to train than the model based on the VGG-16 feature extractor. Models were trained using four different learning rates, 0.001, 0.0005, 0.0002 and 0.0001. For each of those learning rates, five different modes of image augmentation were applied to the dataset; no augmentation, stretching and shifting the images, applying a Gaussian filter, varying the brightness of the image and applying all of the previous steps. This process was then repeated for datasets containing 1000, 2000 and 10000 images per class. Four different learning rates for five different modes of augmentation for three sizes of training data totals 60 models. A validation split of 0.3 was used. After the 60 models were trained, a model was trained for no and all augments, on a dataset of 10000 images per class, with a learning rate of 0.00001.

For each of the resulting models, the accuracy, validation accuracy, loss and validation loss for each epoch was stored, along with the time expenditure. To save time and avoid overfitting, the training stopped early, and the model and got restored to the weights with lowest validation loss, if five epochs passed without any decrease in validation loss. The models were also tested on segments of photographs from the lab, and the accuracy and loss from the testing was recorded. As repeating this process for the model based on VGG-16 was deemed prohibitively expensive in terms of time and computation, select

VGG-16 based models were trained to compare and contrast to scratch-trained models that seemed promising. No and all augments were tried for learning rates of 0.0005 and 0.0001, on datasets of 1000 and 2000 images per class. As before, the relevant metrics were recorded.

The models so trained were evaluated on a set of 31 photographs spread across the three different classes.

In addition to the quantitative testing of the classifier portion of the program, a more qualitative approach was taken to testing the object detection system. Images from the laboratory and elsewhere were tested, and the results of using the program were noted. Incidents of true and false positives, as well as instances of no detection, were also counted and are displayed in table 3.2.

3
Results and Discussion

This chapter concerns the results of the testing described in section 2.3.4. Section 3.1 lists measurements extracted over the course of the research. Section 3.2 analyses the results, considers implications and considers the goal achievement for the project.

3.1 Results

This section describes the results from the quantitative research regarding the training, validation and testing of different image recognition models. It also relates the experienced had when using the program for object detection. The first subsection shows the scores from a systematic regiment of testing different models, and the training of a selection of interesting models. The second subsection describes the use of the object detection program, and the observed difference when using different image recognition models of object detection.

3.1.1 Image recognition

As described in section 2.3.4, a total of 70 image recognition models were trained on synthetic training data, and tested on images from the lab. Table 3.1 shows the results of this testing. As can be seen, the model based on the VGG-16 feature extractor, with a learning rate of 0.0005 and 1000 images per class, using all image augmentation techniques, had the best accuracy in testing. It is worth noting that all VGG-16 based models performed better than models built and trained from scratch.

Type	Learning Rate	Images/class	Augments	Loss	Accuracy
vgg-16	0.0005	1000	all-augments	0.263783	0.903226
vgg-16	0.0005	1000	default	0.281574	0.83871
vgg-16	0.0001	1000	default	0.285783	0.870968
vgg-16	0.0001	1000	all-augments	0.303629	0.903226
vgg-16	0.0001	2000	all-augments	0.405479	0.806452
vgg-16	0.0005	2000	all-augments	0.511962	0.806452
vgg-16	0.0001	2000	default	0.587432	0.741935
vgg-16	0.0005	2000	default	0.748707	0.677419
og	0.001	1000	all-augments	1.096939	0.322581
og	0.0005	1000	all-augments	1.097786	0.419355
og	0.0005	2000	all-augments	1.097922	0.322581
og	0.0005	2000	shift-zoom	1.097968	0.322581
og	0.001	2000	shift-zoom	1.098037	0.419355
og	0.001	10000	shift-zoom	1.098283	0.419355
og	0.001	2000	default	1.098318	0.419355
og	0.0005	10000	all-augments	1.098366	0.419355
og	0.001	2000	gaussian	1.098374	0.322581
og	0.001	10000	gaussian	1.098385	0.322581
og	0.0005	10000	default	1.098691	0.258065
og	0.001	2000	all-augments	1.098705	0.258065
og	0.001	10000	all-augments	1.098774	0.258065
og	0.001	10000	brightness	1.098858	0.322581
og	0.0005	2000	gaussian	1.099041	0.322581
og	0.0005	10000	shift-zoom	1.099182	0.322581
og	0.001	1000	gaussian	1.099213	0.258065
og	0.0005	1000	shift-zoom	1.099259	0.322581
og	0.001	10000	default	1.099328	0.258065
og	0.001	1000	shift-zoom	1.09988	0.258065
og	0.001	1000	brightness	1.100113	0.258065
og	0.0002	1000	shift-zoom	1.152173	0.548387
og	0.0001	2000	all-augments	1.489207	0.612903
og	0.0001	2000	shift-zoom	1.498179	0.483871
og	0.0001	10000	shift-zoom	1.622983	0.580645
og	0.0005	1000	brightness	1.636843	0.354839
og	0.0002	1000	all-augments	1.646527	0.516129
og	0.0001	10000	brightness	1.663313	0.419355

og	0.0002	2000	all-augments	1.774786	0.322581
og	0.0001	1000	shift-zoom	1.802011	0.451613
og	0.0002	10000	all-augments	2.082379	0.612903
og	0.0005	10000	brightness	2.115772	0.322581
og	0.0001	1000	all-augments	2.155455	0.483871
og	0.0002	2000	shift-zoom	2.202826	0.322581
og	0.0001	10000	all-augments	2.218802	0.548387
og	0.0002	1000	brightness	2.240292	0.387097
og	0.0002	10000	shift-zoom	2.699522	0.516129
og	0.0001	10000	default	2.919375	0.419355
og	0.0005	1000	default	3.05364	0.258065
og	0.0005	1000	gaussian	3.5083	0.419355
og	0.0001	1000	brightness	3.532643	0.290323
og	0.00001	10000	all-augments	3.561959	0.612903
og	0.0002	1000	default	3.751132	0.354839
og	0.0002	2000	default	4.041717	0.290323
og	0.001	1000	default	4.070347	0.258065
og	0.0001	1000	default	4.104784	0.225806
og	0.0002	1000	gaussian	4.118333	0.419355
og	0.0002	10000	brightness	4.805454	0.322581
og	0.0001	1000	gaussian	4.882947	0.354839
og	0.0001	2000	default	5.578257	0.290323
og	0.0002	2000	gaussian	6.06869	0.290323
og	0.0002	2000	brightness	6.085357	0.225806
og	0.0005	2000	default	6.284452	0.258065
og	0.0001	2000	gaussian	6.618521	0.225806
og	0.0002	10000	default	7.06943	0.387097
og	0.0001	2000	brightness	7.287948	0.096774
og	0.00001	10000	default	7.913941	0.322581
og	0.0005	2000	brightness	8.612436	0.258065
og	0.0001	10000	gaussian	8.847723	0.451613
og	0.0002	10000	gaussian	9.466131	0.483871
og	0.0005	10000	gaussian	9.545554	0.387097
og	0.001	2000	brightness	12.307479	0.290323

Table 3.1: Complete test results in order of increasing loss.

Figure 3.1 shows the training of the best candidate model produced. Compare to fig. 3.2, a model with comparable accuracy and loss. The first model was trained over 21 epochs for a total of 4098 seconds, while the same image preprocessing and dataset with a lower learning rate resulted in a model trained

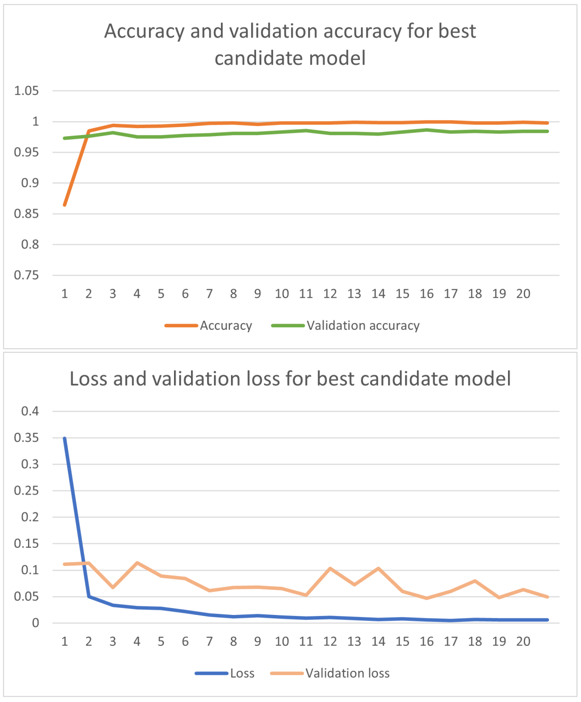


Figure 3.1: Graphs showing training of winning candidate model.

over 13 epochs for a total of 2245 seconds.

Figure 3.3 shows the training of a model from scratch, with a dataset of 10,000 images per class, a learning rate of 0.00001 and no pre-processing. The validation loss and accuracy arrived upon in training is similar to those shown in fig. 3.1 and fig. 3.2. When testing this model on real photographs, the accuracy was 0.32 and the loss was 7.9.

For the 8 VGG-16 based models trained, the average time to complete training was 59.5 minutes. When using 1000 images per class, the average training time was 57 minutes, while for 2000 images per class, the training time was 62 minutes on average. For the models trained from scratch, those trained with 1000 images per class took an average of 6.3 minutes, and those trained on 2000 images took 13.9 minutes to train on average. The average time to complete training overall was 10.1 minutes.

3.1.2 Object detection

When using the object detector, an image is rendered into a series of regions to be classified over the course of 0.5 to 0.6 seconds on the computer used

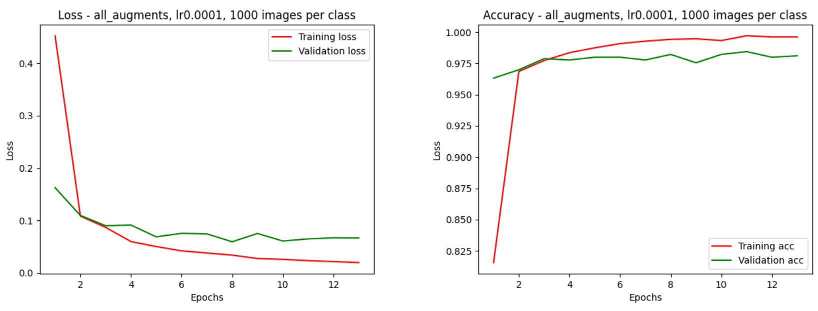


Figure 3.2: Model with comparable accuracy but higher loss.

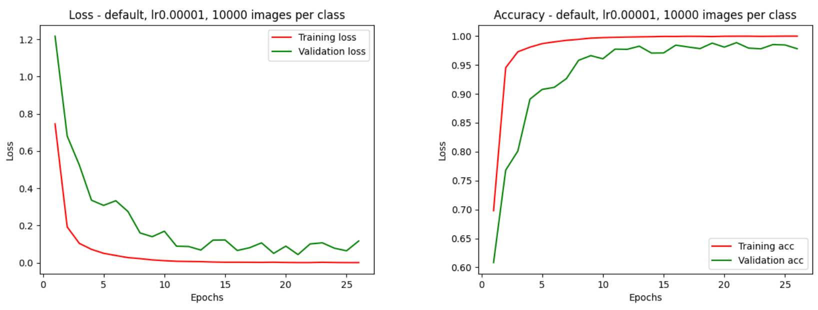


Figure 3.3: Training of model with loss of 7.9 and accuracy of 0.32 in tests.

for testing. The ROIS are then classified. Using an image recognition model built from scratch, classifying 2343 such regions usually takes 5 to 7 seconds. Using a VGG-16 based model for object detection, classifying those ROIS takes around 138 to 140 seconds. The VGG-16 based model takes 20 times longer to classify the ROIS.

Figure 3.4 shows the use of the object detection program with an image recognition model built and trained from scratch with only Gaussian filtering.

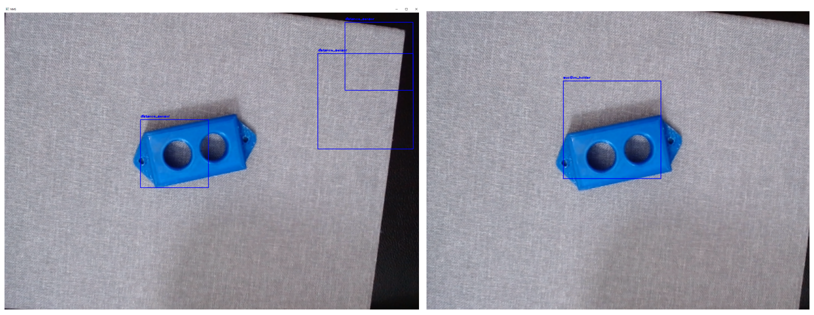


Figure 3.4: Object detection with original, inaccurate model.

Table 3.2: Detection over a series of images.

	True positive	False positive	No detection
Distance sensor	0	37	2
Lift turn	6	12	0
Suction holder	6	14	0

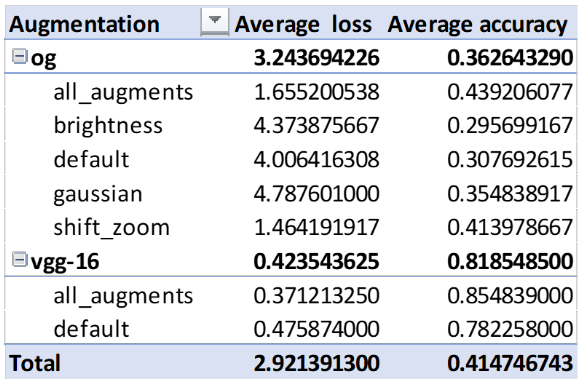
In testing, this model had an accuracy of around 0.5. As can be seen in the image, the object detection program was capable of finding a distance sensor holder, but also falsely labeled the distance sensor holder as a suction holder. Image recognition models trained from scratch failed to detect lift turns, yielding neither false nor true positives.

Figure 3.5 shows the usage of the winning candidate from the previous experiment for object detection. The images show false positives for all classes, even with a detection threshold at a confidence of 95%. Using this model to run object detection on a series of images, true positives, false positives and instances of no detection were counted for the different classes. The results are shown in table 3.2.

3.2 Discussion

In this section, the results shown in section 3.1 are considered and analysed. The degree of goal achievement for the project is also discussed.

Table 3.3: Average loss and accuracy by model and pre-processing.



Augmentation	Average loss	Average accuracy
og	**3.243694226**	**0.362643290**
all_augments	1.655200538	0.439206077
brightness	4.373875667	0.295699167
default	4.006416308	0.307692615
gaussian	4.787601000	0.354838917
shift_zoom	1.464191917	0.413978667
vgg-16	**0.423543625**	**0.818548500**
all_augments	0.371213250	0.854839000
default	0.475874000	0.782258000
Total	**2.921391300**	**0.414746743**

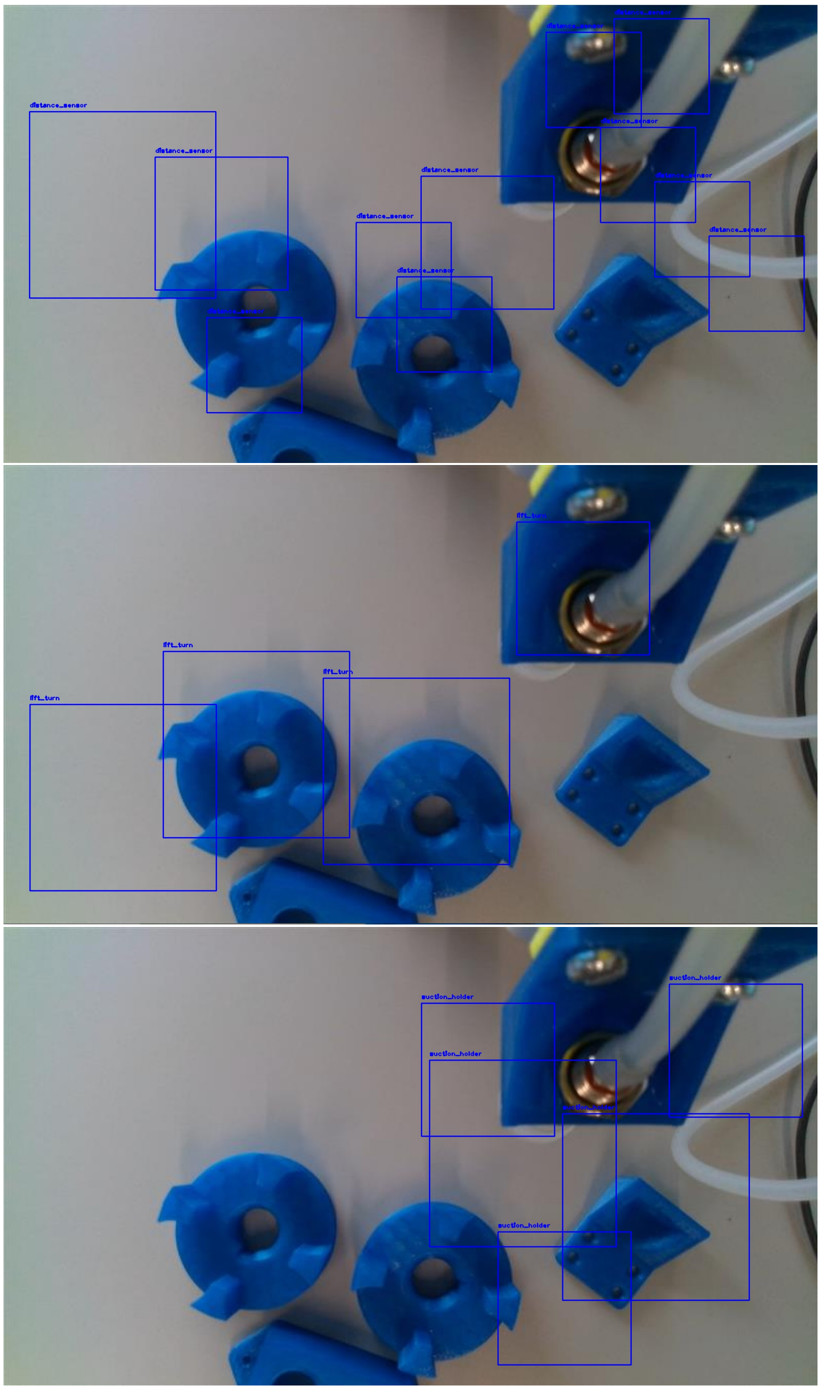


Figure 3.5: Object detection using the winning candidate.

3.2.1 Analysis

In Table 3.3 can be seen the average loss and accuracy of the models tested, by architecture and image augmentation steps. By looking at this average, we discover that varying the brightness and introducing Gaussian filtering will not necessarily improve the loss or accuracy for a model. It can be inferred that shifting and stretching the training images is important for training of a model from scratch. Looking at the VGG-16 based models, however, we can notice that this effect is lessened here. From table 3.1 we can see that for VGG-16 based models, including the image augmentation when training the model will generally improve loss and accuracy. Unlike with the untrained model, this factor does not seem more important than the size of the dataset and the learning rate. This could suggest that the VGG-16 feature extractor is more capable of extracting features recognizable in photographs.

If we compare the training of the best image recognition model, as seen in fig. 3.1, with that of a select model from near the bottom of the list, as shown in fig. 3.3, a problem with the image recognition models trained from scratch becomes apparent. When evaluating the models on automatically generated data, those two models are similar in evaluation loss and accuracy. Without real example images, it is effectively impossible to discover which image recognition model is best suited for purpose. The models with feature extraction layers trained on synthetic datasets seem incapable of making accurate predictions, no matter how promising the training seemed. From this, a possible hypothesis is that using transfer learning is a necessary step to recognizing parts from rendered images. It is also possible that better rendering techniques can sidestep the need for transfer learning.

Another consideration for the practicality of an image recognition model is the time spent training the model, and the speed at which a model can make predictions. Training a VGG-16 based model took between 5 and 10 times as long as training the model made from scratch. This is not a big problem if the user has a general idea of which learning rate and size of dataset will provide sufficient, but a systematic search can be very computationally expensive when using transfer learning. As described in section 3.1.2, making a prediction for a single took upwards of 60 milliseconds when using a VGG-16 based model, compared to 3 milliseconds for the simpler model. Improving the speed could be a software engineering problem, a matter of more efficient operations and careful handling of memory. It is also possible the program is hitting a hard limit of the architecture of the image recognition model. Section 4.1 suggests ways to ameliorate the issue if it is indeed the case of the latter.

From section 3.1.2, and as seen in fig. 3.4 and fig. 3.5, we can see that even with an image recognition model providing better than 90% accuracy, the object detection yields a number of false positives, and struggles with detecting the depth sensor holder. False positives could be a result of the output layer of the image recognition model using the *softmax* activation function. Using this activation function, the confidence for all the different classes will always sum up to 1, in a given prediction. This is useful for classification, where the model must distinguish between objects of different classes, but it can lead to an inability to correctly identify a negative match. Possible solutions to this problem are mentioned in section 4.1.

3.2.2 Goal achievement

When considering the degree of achievement reached over the course of this project, the requirements laid out in section 1.5 must be considered. The subsections enumerate the requirements for three separate parts of the project. Section 1.5.1 concerns the training of an image recognition model, and section 1.5.2 concerns the generation of training data for said image recognition model. These two components of the project directly correspond to the objective of the MSc as described in section 1.2. The subsections 1.5.3 and 1.5.4 then concern the implementation of object detection and the testing of the program to examine the viability and practicality of the system.

The requirements described in section 1.5.1 have been fulfilled. The installation and usage of the program is described in appendix B, and this program can create, compile and train an image recognition model using two different network structures with variable hyper-parameters and pre-processing steps. The models so trained are deep, convolutional neural networks. Models with varied weights and structures can be saved and restored easily, and the program enables evaluation and testing of the image recognition model. As described in section 3.1.1 and reflected on in section 3.2.1, an image recognition model with an accuracy of over 90% has been trained, which indicates the viability of training models on synthetic data.

The program has a class called “ImageGenerator” used for generating synthetic training data. Provided a folder of STL files and a series of background images, this class can load STL files and render them on top of varied and random backgrounds, from any number of angles. This process is described in appendix B.2. While there are possible changes and improvements to this process described in section 4.1, the outcomes achieved means the goal of creating a basis for training and testing using synthetic images has been fulfilled. With this, both of the objectives described in section 1.2 have been fulfilled.

This thesis examines the viability of using the methods described above to achieve object detection. To this end, a rudimentary object detection system based on the trained image recognition model was implemented. Section 3.1.2 shows that the object detection achieved by the end of the project was not sufficiently precise to steer a robot arm by, and the object detection was too slow to be entirely practical for this purpose. It is nevertheless my belief, based on the achievements over the course of the project, that the methodology for generating datasets and training image recognition models has been proven as viable and practical.

The work in this thesis represents a step towards further automation of automatic picking of 3D printed and machined objects. It forms a base for studies into this novel application of several tried and tested technologies. Based on the results achieved, object detection quick and precise enough to pick up parts should be possible with further refinements to the process. A number of such refinements and ideas for further development are proposed in section 4.1.

4 Further Development and Conclusion

4.1 Further development

Over the course of the project, a number of ideas for further work sprung to mind; features outside the original scope of the project, different approaches to parts of the project and ways portions of the project could be improved upon. With the benefit of hindsight, some of the decisions made early in the project can also be considered in that light. Some of the areas of further development could even form the basis of future Master's theses.

Upon experiencing the significant delay between a web camera taking an image and the object detection program yielding a result, different possible approaches to bringing the process closer to real-time were considered in turn. One possible approach to expediting the object detection process could be to discard the current machine learning solution entirely, instead utilizing a network made end-to-end for simultaneously detecting and classifying objects in an image. An example of such a solution is YOLO[11]. Such a change would also require re-engineering the synthetic image generator, and instead of small images with the object of interest in the center, a whole scene with scattered parts would have to be synthesized, paired with suitable labels and bounding boxes. A future attempt at fulfilling the goals of the fundamental project could try such an approach, and in part base the project on the code and discoveries made in this project.

With the current approach taken to object detection, the activation function used in the output layer of the image classifier can be prone to false positives. Without re-engineering the entirety of the project as described in the paragraph above, adding a specialized and fast neural network for region proposal could drastically reduce the incidence of false positives. Depending on the exact capabilities of the region proposal network, the object detection could also become much faster using such an approach. Future work exploring this possibility could more easily expand on the methods described in this thesis, as it would require the integration of a new, specialized component instead of a complete re-engineering of the project.

Using standard methods and formats for taking pictures and storing them, each pixel of an image has three associated values; red, green and blue. Those three values correspond with the third dimension of the input layer used by the neural networks. The cameras used during this process have the capability of providing depth information as well. Future work could concern an attempt to utilize depth data for object detection. With rendering of 3D models, synthesizing depth data should be possible. A region proposal system could also be implemented on the assumption that all protrusions from an otherwise flat background are possible ROIS.

As became apparent while sifting through metrics on different combinations of parameters for training image recognition models, deciding on which model to use for object detection could be a challenge by itself. A possible future improvement to the program would be an algorithm capable of automatically searching out the best model candidates based on training metrics and automated testing. Automating this would go a long way towards the eventual full automation of the system.

The lack of positive identification of a lift turn in real-world test images illustrated a problem for the object detection system using the model trained from scratch. This is a common problem associated with pose-invariant classification. A coin laying on the ground is rarely seen from any other facing than head (or tail) on. The lift turn faced the same problem. When laid on a table, the appearance of the part matched only a small percentage of the views spread evenly in a sphere around the 3D model. Future work on the automatic generation of training data could experiment with using rudimentary physics to develop a more accurate model of how the different parts will appear. Physics based rendering with subsurface scattering and other methods of approaching photo-realism could also yield interesting results.

4.2 Conclusion

This thesis has examined the viability of training an image recognition model solely on synthetic training data generated by rendering 3D models on different backgrounds. It also examined the viability and practicality of using such a model for object detection in order to pick printed parts off an assembly line.

A program was created that can generate training images from STL files and photographs of different backgrounds. The program can also train and store a number of image recognition models using those training images, and the program can retrieve and use a saved image recognition model for object detection.

A total of 72 image recognition models were trained using two different network structures and a series of different parameters. Those models were then evaluated on 31 images of the different parts, taken from the lab. The best image recognition model was used for object detection, and the practicality of the program was considered.

This thesis suggests a successful methodology for automatic generation of training data, and for the creation of custom image recognition models based on this. In doing so, it confirms the viability of the use of synthetic training data for computer vision for automation in robotic manufacture, and provides a basis for future work in the field. It also suggests a method of using such a model for object detection, and possible refinements and improvements that can be attempted in future work.

Bibliography

[1] H. Arnarson, "Reconfigurable Manufacturing System." https://youtu.be/idA-TmYP45c, 2021. Accessed: 2022-04-30.

[2] L. Huang, "CNC History: The Origination and evolution of CNC Machining 0." https://www.rapiddirect.com/blog/cnc-history/, 2021. Accessed: 2022-05-14.

[3] T. S. Huang, "Computer Vision: Evolution And Promise," *1996 CERN School of Computing*, no. 19, pp. 21–25, 1996.

[4] Google, "Google Lens." https://lens.google/, 2022. Accessed: 2022-01-27.

[5] J. Cohen, "Computer Vision Applications in Self-Driving Cars." https://www.thinkautonomous.ai/blog/?p=computer-vision-applications-in-self-driving-cars, 2019. Accessed: 2022-04-30.

[6] Z.-H. Zhou, *Machine Learning*. Springer, Singapore, 2021.

[7] J. Schaeffer and R. Lake, "Solving the game of checkers," *Games of no chance*, vol. 29, pp. 119–133, 1996.

[8] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[9] T. Kohonen, "An introduction to neural computing," *Neural Networks*, vol. 1, no. 1, 1988.

[10] M. Jogin, Mohana, M. S. Madhulika, G. D. Divya, R. K. Meghana, and S. Apoorva, "Feature Extraction using Convolution Neural Networks (CNN) and Deep Learning," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pp. 2319–2323, 2018.

[11] H. Bandyopadhyay, "YOLO: Real-Time Object Detection Explained." https://www.v7labs.com/blog/yolo-object-detection, 2022. Accessed: 2022-01-17.

[12] J. Sullivan, "Machine learning LEGO image recognition: Using virtual data and YOLOv3," *Towards Data Science*, 2020. Accessed; 2022-04-30.

[13] K. Sarkar, K. Varanasi, and D. Stricker, "Trained 3D models for CNN based object recognition," in *Proceedings of the 12th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5* (A. P. Cláudio, D. Bechmann, P. Richard, T. Yamaguchi, L. Linsen, A. Telea, F. Imai, and A. Tremeau, eds.), 2017.

[14] B. Heisele, G. Kim, and A. J. Meyer, "Object Recognition with 3D Models," tech. rep., Honda Research Institute, Cambridge, USA, 2009.

[15] R. Kumar, S. Lal, S. Kumar, and P. Chand, "Object detection and recognition for a pick and place Robot," in *Asia-Pacific World Congress on Computer Science and Engineering*, pp. 1–7, 2014.

[16] TensorFlow, "TensorFlow." = https://www.tensorflow.org/„ 2022. Accessed: 2022-04-30.

[17] "Keras: the Python deep learning API." https://keras.io/, 2022. Accessed: 2022-04-30.

[18] A. Clark and F. Lundh, "Python Pillow." https://python-pillow.org/, 2022. Accessed: 2022-04-30.

[19] B. Woodsend, "Welcome to vtkplotlib's documentation! – vtkplotlib 1.4.1 documentation." https://vtkplotlib.readthedocs.io/, 2021. Accessed: 2022-04-30.

[20] OpenCV team, "About - OpenCV." https://opencv.org/about/, 2022. Accessed: 2022-05-02.

[21] PyTorch, "PyTorch." https://pytorch.org/, 2022. Accessed: 2022-05-03.

[22] Oracle, "What is Java and why do I need it?." https://java.com/en/download/help/whatis_java.html, 2022. Accessed: 2022-05-03.

[23] The Mathworks, Inc, "Help Center for MATLAB, Simulink and other MathWorks products." https://se.mathworks.com/help/index.html?s_tid=CRUX_lftnav, 2022. Accessed: 2022-05-03.

[24] T. Peters, "PEP 20 – The Zen off Python | peps.python.org." `https://peps.python.org/pep-0020/`, 2004. Accessed: 2022-05-03.

[25] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems." `https://www.tensorflow.org/`, 2015. Software available from tensorflow.org.

[26] A. Paszke, S. Gross, S. Chintala, and G. Chanan, "pytorch/pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration." `https://github.com/pytorch/pytorch`, 2022. Accessed: 2022-04-30.

[27] Kitware, "VTK - The Visualization Toolkit." `https://vtk.org/`, 2022. Accessed: 2022-01-27.

[28] pawangfg, "VGG-16 | CNN model - GeeksforGeeks." `https://www.geeksforgeeks.org/vgg-16-cnn-model/`, 2020. Accessed: 2022-05-03.

[29] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big Data*, vol. 3, no. 9, 2016.

[30] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition." `https://arxiv.org/abs/1409.1556`, 2014.

[31] B. A. Bremdal, "Convolutional Networks." `https://uit.instructure.com/courses/22308/files/1500060?module_item_id=504809`, 2021. Power Point presentation from lecture in course DTE-3606, Accessed: 2022-05-11.

[32] A. Rosebrock, "Turning any CNN image classifier into an object detector with Keras, TensorFlow, and OpenCV." `https://pyimagesearch.com/2020/06/22/turning-any-cnn-image-classifier-into-an-object-detector-with-keras-tensorflow-and-opencv/`, 2020. Accessed: 2022-04-30.

[33] J. Prakash, "Non Maximum Suppression: Theory and Implementation in PyTorch." `https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/`, 2021. Accessed; 2022-05-03.

A Task Description

Included in this appendix is the original task description for this thesis. The following pages contain the document in its entirety, as received at the start of the project period.

Faculty of Engineering Science and Technology
Department of Computer Science and Computational Engineering
UiT - The Arctic University of Norway

Automatic generation of custom image recognition models

Magnus Fredheim Hanssen

Thesis for Master of Science in Technology / Sivilingeniør

In the machine laboratory in Narvik at DIE, there has been built a reconfigurable manufacturing system (RMS). The system has different platforms such as a 3D printer that can atomically print parts and robot arms that can pick and place objects. The system can reconfigure itself automatically and the goal is to get it 100% automated when producing parts. A video demonstrating how the system works can be found at: https://youtu.be/idA-TmYP45c .

One of the tasks that needs to be automated is to have the robot arm pick and place the parts coming from the 3D printer. To do this, we need custom image recognition models so that the robot can automatically recognize the 3D printed parts. This will rely on machine learning models for image recognition. Due to the fact that such parts can be unique and without any presidents suited for training machine learning algorithms for image classification synthetic data need to be established. Creating such can be a dull, tedious and time-consuming process. However, 3D prints are driven by 3D model of the parts. The primary challenge is to use such as the basis for annotated images that can be used for training and testing of image recognition models. The new idea is to use a 3D model (CAD model) and atomically generate images from the model, which can be used to create a custom image recognition model of various parts. The 3D model can be rotated and different backgrounds can be inserted to produce a sufficient set of perspectives for robust and resilient training.

Hence the objective of the MSc can be formulated:

1. Create an image recognition system based on state-of-the-art machine learning models that can be used for the purpose described above

2. Create a basis for training and testing by means of synthetic images using 3D models generated by the CAD system used to design and produce the parts to be recognized and classified.

The student should spend the first part of this project to formulate the precise problem and suggest possible solution strategies to determine how to recognize and classify a part and instructions. The initial part of the project should probe into state-of-the-art and solicit related work on which the planned effort should build on.

Dates

Date of distributing the task:	<10.01.2021>
Date for submission (deadline):	<16.05.2021>

Contact information

Candidate	Magnus Fredheim Hanssen Mha430@post.uit.no
Supervisor at IDBI	Bernt Bremdal bernt.bremdal@uit.no
Co-supervisor at IE	Halldor Arnarson halldor.arnarson@uit.no
Co-supervisor at IDBI	Shayan Dadman shayan.dadman@uit.no

General information

This master thesis should include:

- ❄ Preliminary work/literature study related to actual topic
  - A state-of-the-art investigation
  - An analysis of requirement specifications, definitions, design requirements, given standards or norms, guidelines and practical experience etc.
  - Description concerning limitations and size of the task/project
  - Estimated time schedule for the project/ thesis
- ❄ Selection & investigation of actual materials
- ❄ Development (creating a model or model concept)
- ❄ Experimental work (planned in the preliminary work/literature study part)
- ❄ Suggestion for future work/development

Preliminary work/literature study

After the task description has been distributed to the candidate a preliminary study should be completed within 3 weeks. It should include bullet points 1 and 2 in "The work shall include", and a plan of the progress. The preliminary study may be submitted as a separate report or "natural" incorporated in the main thesis report. A plan of progress and a deviation report (gap report) can be added as an appendix to the thesis.

In any case the preliminary study report/part must be accepted by the supervisor before the student can continue with the rest of the master thesis. In the evaluation of this thesis, emphasis will be placed on the thorough documentation of the work performed.

Reporting requirements

The thesis should be submitted as a research report and could include the following parts; Abstract, Introduction, Material & Methods, Results & Discussion, Conclusions, Acknowledgements, Bibliography, References and Appendices. Choices should be well documented with evidence, references, or logical arguments.

The candidate should in this thesis strive to make the report survey-able, testable, accessible, well written, and documented.

Materials which are developed during the project (thesis) such as software / source code or physical equipment are considered to be a part of this paper (thesis). Documentation for correct use of such information should be added, as far as possible, to this paper (thesis).

The text for this task should be added as an appendix to the report (thesis).

General project requirements

If the tasks or the problems are performed in close cooperation with an external company, the candidate should follow the guidelines or other directives given by the management of the company.

The candidate does not have the authority to enter or access external companies' information system, production equipment or likewise. If such should be necessary for solving the task in a satisfactory way a detailed permission should be given by the management in the company before any action are made.

Any travel cost, printing and phone cost must be covered by the candidate themselves, if and only if, this is not covered by an agreement between the candidate and the management in the enterprises.

If the candidate enters some unexpected problems or challenges during the work with the tasks and these will cause changes to the work plan, it should be addressed to the supervisor at the UiT or the person which is responsible, without any delay in time.

Submission requirements

This thesis should result in a final report with an electronic copy of the report including appendices and necessary software, source code, simulations and calculations. The final report with its appendices will be the basis for the evaluation and grading of the thesis. The report with all materials should be delivered according to the current faculty regulation. If there is an external company that needs a copy of the thesis, the candidate must arrange this. A standard front page, which can be found on the UiT internet site, should be used. Otherwise, refer to the "General guidelines for thesis" and the subject description for master thesis.

The supervisor(s) should receive a copy of the the thesis prior to submission of the final report. The final report with its appendices should be submitted no later than the decided final date.

B Installation and Use of Program

The code used for the project is contained in a zip-archive attached to the thesis. The program consists of a series of Python files, image files and STL files. This appendix describes the installation of packages required to run the program, and a quick guide to using the program for classification and image recognition.

B.1 Installation

After unpacking the archive containing the project, one should either navigate to the resulting directory in a console window, or open up the folder as a project in an Integrated Development Environment (IDE) of choice. In the root of this directory, the file "requirements.txt" lists the packages required for the project. This installation guide assumes that Python 3.9 with pip is installed.

B.1.1 Creating a virtual environment

When installing Python packages for a project, it is recommended to use a virtual environment. To create a virtual environment with a Python 3.9

installation, "`python -m venv c:/path/to/virtual/environment`" can be written into the console if the only python version installed is Python 3.9 and the path to the executable has been added to the `PATH` variable. Otherwise, "python" should be swapped with the path to the executable of a Python 3.9 installation. In order to use the virtual environment instead of the default Python installation, the environment needs to be activated. To do this, the script "`.\venv\Scripts\activate`" should be run in the console. The name of the virtual environment in parentheses should now be displayed to the left of the current directory in the console window to indicate that the virtual environment is active. Packages installed from this window will now be installed into the virtual environment, and Python scripts will be executed using the virtual environment and packages installed therein. There are other tools available for creating virtual environments, but this guide assumes that such advanced users need no further instruction in setting up virtual environments.

B.1.2 Installing requirements

"requirements.txt" contains a list of all required packages. These packages are all available from the Python Package Index, and can be installed with pip, a tool included with Python installations after version 3.4, using the following command:

```
pip install -r requirements.txt
```

If this command is run from the root directory of the project, and with the virtual environment activated as described in Subsection B.1.2, all the packages required to run the project should be installed.

B.2 User Guide

In order to quickly and easily train an image recognition model, running the script "main.py" in the project directory will create images from the included STL files, train an image recognition model based on the VGG-16 feature extractor, and save this model as "default" in the "saved_models" directory. To customize the number of images generated, as well as details of the model trained, the user can either make changes to "main.py" before running it, or they can open a Python thread in the console (by giving the prompt "python") and create an *ImageGenerator* and *ObjectClassifier* object in the console, changing data members before generating images and training a model as shown in the script.

The *ImageGenerator* constructor takes as a parameter the number of viewpoints to generate images from for each 3D model. The number of images per class then becomes this number of view points multiplied by “bg_mult”, 5 by default, which determines how many times each view is rendered on top of a random background. The color used to render the model can be changed by setting “color” to a different hexadecimal string.

The *ObjectClassifier* constructor takes a string as a parameter, using this string as the name of the model, and defaulting to “default” if none is provided. The methods used to save the model, the training history of the model, and the corresponding graphs, all use this name in the respective file names unless another name is provided. The script “log_train_classifier.py” demonstrates how the optimization algorithm and image augmentation steps of an image recognition model can be changed, and how a number of different models can be trained and metrics from the training logged. The script “test_classifier.py" demonstrates how a series of models with certain prefixes were tested and the results logged. The code therein demonstrates how the user can similarly evaluate a model.

To use an image recognition model for object detection, the script “object_detector.py” can be run. Unlike previous scripts, however, this script must be run with additional options. The program requires one of two options to be provided; “`python object_detector.py -wc [camera port number]`” can be run to run object detection using the “default” model from previous steps on an image from a web camera on the specified port. Web cameras connected to a computer are numbered from 0, so debugging includes incrementing this number if several cameras are attached to the computer.

When the program is used with a web camera, the user is prompted to write “Q” to quit, and hitting the enter key with any other input will result in an image being taken and for the loop to continue. Alternately, the user can provide the option for running image detection on an image file, with “`-i path/to/image.png`”. The optional arguments can be listed by running the script with the “`-h`” or “`-help`” option. Enabling extra visualization is not recommended, as it is an option left over from debugging, and will show thousands of images in sequence.

The images “grumpy_cat.jpg” and “test_image.jpg” are included to provide example images with and without any of the 3D printed parts.