Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

Tor Magne Kippersund

Exploring Machine Learning Methods for Plastic Classification

Masteroppgave i MTDT
Veileder: Hai Thanh Nguyen
Juli 2020

Tor Magne Kippersund

Exploring Machine Learning Methods for Plastic Classification

Masteroppgave i MTDT
Veileder: Hai Thanh Nguyen
Juli 2020

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for datateknologi og informatikk

I would like to dedicate this thesis to my wonderful family. To my brother and sisters, to my mom and dad, and to my girlfriend. Had it not been for their unyielding support and their belief in my abilities I would not have been able to complete this work.
I wish to thank my supervisor, Hai, who guided me through the project and pointed me in the right direction when I was stuck, this thesis would not be possible without him.
I would also like to thank my friends, and flatmates, from Klostergata 56 for three wonderful years in Trondheim. I will dearly miss our time together.

Summary

In this paper we explore ways in which augmentations can be calculated once and added to a dataset for increased performance, as well as what type of deep learning machine vision method is best suited for plastic classification. Three experiments are conducted to investigate this, training three different machine learning models — InceptionResNetv2, SSD-VGG-300 and Mask-RCNN-resnet101 — using six different datasets. Results show that adding augmented copies of original images to a dataset does not increase model performance significantly, but increases training time proportionally to the amount of images added. Furthermore we find that instance segmentation, in this case using Mask-RCNN, can be recommended, since it shows good enough performance on the small dataset and can categorise more than one item at a time. This does come at the notable expense that instance segmentation has a lot higher cost of development for a dataset and the Mask-RCNN model needed almost 24 times as long to train an equal amount of epochs as The InceptionResNetv2 model.

I denne oppgaven utforsker vi forskjellige måter vi kan forbedre ytelsen til maskinlæringsmodeller ved å utvide datasett med forhåndsprosesserte bilder, samt hvilke dyp lærings-metoder for maskinsyn er best egnet til å klassifisere plastikk. Tre eksperimenter ble utført for å undersøke dette. Vi trente tre forskjellige maskinlærings modeller — InceptionResNetv2, SSD-VGG-300 og Mask-RCNN-resnet101 — ved bruk av seks forskjellige datasett. Resultatene viser at det å legge til transformerte kopier av originale bilder til et datasett ikke gir noen betraktelig økning i modellens ytelse, men øker tiden det tar å trene modellen proporsjonalt med mengden bilder som er lagt til. Videre finner vi at instanssegmentering, i dette tilfellet ved bruk av Mask-RCNN, kan anbefales, siden det viser god nok ytelse på et lite datasett og kan kategorisere mer enn en ting av gangen. Dette kommer med den nevneverdige kostnaden at instanssegmentering har et mye høyere ressurskrav i forbinnelse med å utvikle et datasett og at Mask-RCNN modellen bruker nesten 24 ganger så lang tid på å trene like mange epoker som InceptionResNetV2 modellen.

Table of Contents

List of Tables

List of Figures

Abbreviations

AI	=	Artificial Intelligence
DNN	=	Deep Neural Network
HDPE	=	High Density Polyethylene
IRNV2	=	Inception-ResNetV2
LDPE	=	Low Density Polyethylene
mAP	=	Mean Average Precision
ML	=	Machine Learning
MRCNN	=	Mask RCNN
PET	=	Polyethylene Terephthalate
PP	=	Polypropylene
RCNN	=	Region-based Convolutional Neural Network
RPN	=	Region Proposal Network
SGD	=	Stochastic Gradient Descent
SSD	=	Single Shot Multibox Detector
CPU	=	Central Processing Unit
GPU	=	Graphics Processing Unit

Chapter 1

Introduction

1.1 Problem Description

Plastic is widely used in our modern world. We use it as packaging, as containers and as tools. Plastic is durable and takes a long time to break down. This is one of its main advantages, but when the plastic is thrown away this immediately becomes a challenge. The combination of these two factors have caused plastic to pile up in our landfills and ultimately in natural habitats around the world [4].

This has become a problem and one of the solutions is recycling.

Plastic comes in a lot of different varieties, from films and foils to rigid inert plumbing and containers. These are very different in chemical composition and simply melting everything together is not a practical way of recycling them. A list of resin-codes have therefore been established by the American Chemistry Council and the European commission to identify and separate different types of plastics and recyclables. The main ones for plastic are shown in table 1.1.

These types of plastic must be separated before they can be recycled effectively. Automatically classifying plastic would be an advantage, since it could help especially machines and potentially humans to more effectively separate and sort plastic that could then be recycled and kept out of landfills. Doing this using images is preferred, since it removes the need for special equipment and direct contact or destructive analysis of the plastic. Today automatic classification is often performed by machine learning models based on neural networks, and when classifying images convolutional neural networks perform especially well.

There have been attempts at this with regards to plastics, but there is still a lot of potential for improvement [5] [6].

One of the problems that were faced in these attempts was the amount of data available. Datasets with enough images of the relevant plastics, and with enough variations and context, is currently not available. The research also suggested that object detection could be a possible way forward.

With this in mind the decision was made to explore the problem of small datasets and look into using object detection and instance segmentation to sort plastics. This lead to the following research questions being formulated.

Symbol	Code	Description
♳ 01 PET	#1 PET(E)	Polyethylene terephthalate
♴ 02 PE-HD	#2 PEHD or HDPE	High-density polyethylene
♵ 03 PVC	#3 PVC	Polyvinyl chloride
♶ 04 PE-LD	#4 PELD or LDPE	Low-density polyethylene
♷ 05 PP	#5 PP	Polypropylene
♸ 06 PS	#6 PS	Polystyrene
♹ 07 O	#7 O (OTHER)	All other plastics (reffered to as NA in this project)

Table 1.1: Plastic types and recycling/resin codes.

1.2 Research questions

- **RQ1:** What actions can be performed to increase the quality of a size-limited dataset?
- **RQ2:** How does classic image-classification compare to object detection and instance segmentation with regards to sorting plastic?

1.2.1 Research Question 1

The focus of this question lies in what affects the quality of a dataset, and specifically how you can most efficiently increase the quality of a small dataset through image-transformations.

1.2.2 Research Question 2

This question focuses on how well classic whole image classification performs compared to methodologies such as object detection and instance segmentation, which aim at not only categorizing images, but also locate where in the image the classified object is.

Chapter 2

Theory

2.1 Machine Learning

Machine learning is a branch of AI. It consists of different branches, but in this paper we are mostly concerned with supervised learning[7]. For an in-depth introduction to machine learning fundamentals, see [8].

2.2 Supervised learning

Supervised learning is essentially learning by example[9]. It works by showing an agent an example, letting it categorise or act on it, giving it feedback on whether it was wrong or right, and possibly how wrong or how right it was. In machine learning(ML) this is usually accomplished with the use of deep learning.

2.3 Deep Learning

Deep learning is the concept of using deep neural networks with many hidden layers to approximate a function. The function can be as simple as performing an AND, OR or XOR function on two inputs, but it is more common to use this method for a purpose where the mapping of inputs to outputs is not known beforehand. Deep learning is usually performed with supervised learning, where loss is calculated on the networks output for a given input and then back-propagated through the network to adjust the models weights before being shown the next example. This process is repeated, and with a sufficient number of examples and enough training time the model can hopefully be used to correctly process new input.

2.4 Convolutional Neural Networks

In the world of machine learning today, and specifically machine vision, convolutional neural networks are what performs best. There are many different variants of them, having different depths, different amount of convolutional layers, feedforward parts and such and such, but there are fundamental simillarities between them. A convolutional neural network consists of at least two different parts; one part that extracts features from the image, using image convolutions, pooling and dimensional reduction, and a classifier that gives predictions based on the features extracted.

2.5 Image Classification

Image classification is the process of automatically classifying images into different categories. This can be achieved with deep learning, and is usually performed by convolutional neural networks. Since images often have a lot of pixels it is often impractical to map each of these pixels to a separate input node. Convolutional neural networks therefore use matrix convolutions as their first layers which reduce the spacial resolution of the image and outputs many smaller images to the next layer and so forth until you are left with a many-dimensional, but very small image. Each of these dimensions can be thought of as a ”feature” in the image and a classification layer is often added to the end of this to map different sets of features to different categories of outputs.

For the image classification in our project we will be using a model architecture called InceptionResNetv2[10], which has previously been trained on the ImageNet dataset[11]. **Fig. 2.1** shows the structure of InceptionResNet in diagram form.

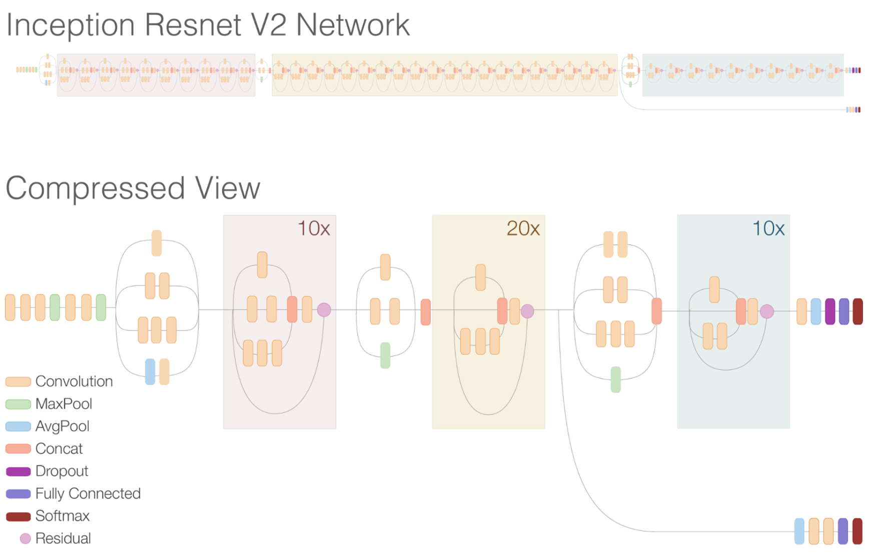


Figure 2.1: Diagram of InceptionResNet Architecture[1]

2.6 Object Detection

Explained in short an object detector in machine learning is a network that takes an image as input and outputs bounding boxes surrounding the objects of interest in the image.

This works by having a Region Proposal Network (an RPN) that automatically gives regions (crops) of an image that might have an object, and a CNN that classifies each of these areas.

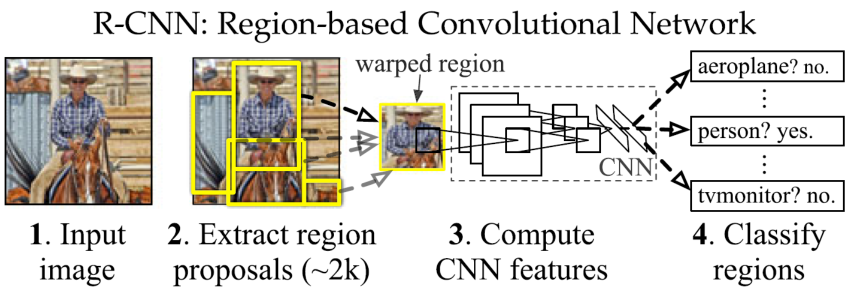


Figure 2.2: How the RCNN model detects objects in an image[2].

2.7 Instance Segmentation

Instance segemntation goes one step further and also predicts a mask, or an outline, of where an object is, resulting in a per-pixel classification, as shown to the right in Figure 3.1.

This is accomplished in the Mask-RCNN model with the use of a separate output for mask prediction and a different algorithm for aligning the regions of interest[12].

As illustrated below in figure 2.3, the image is first passed through a convolutional network, that extracts the features of the image. The last convolutional layer is then passed through the region-proposal network and recombined with the output features to produce the regions of interest that is finally classified to generate a class, bounding box and mask.

2.8 Datasets

A very important factor in how a machine learning model performs is the dataset it has been trained on. The better the dataset the better the performance is generally the case. Exactly what makes a dataset good is a different question, and slightly more complicated to answer. It is generally accepted that size matters. A bigger dataset will usually be better than a smaller one. This does however generally come at the cost of longer training time and of course higher cost of developing the dataset.

A balanced dataset will also often outperform an imbalanced one. Balancing in this term means that there are about the same amount of images or examples for each category.

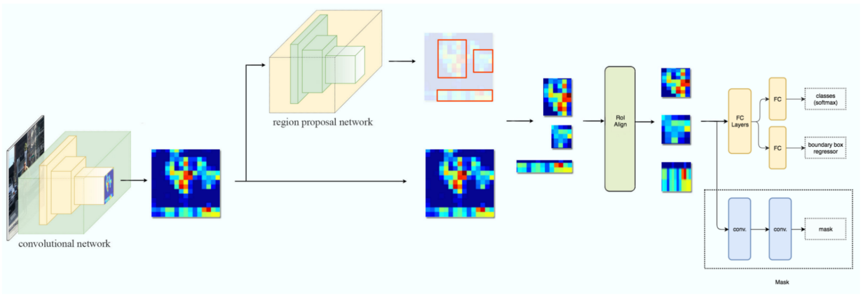


Figure 2.3: Illustration of the structure of the Mask-RCNN[3]

Having a large discrepancy between the amount of examples for each category can produce a model that is scewed in its performance, performing better on the well represented categories and worse at the less represented categories.

There are also ways to minimize the required size of the dataset. Having a large variety in the images helps the model to generalize better, augmentations can be performed on the images to create artificial variety and if there already exist a large and varied dataset for a different purpose than yours it is possible to first use this for training and afterwards substitute it for your own to fine-tune your model. This process is called transfer learning and is explained in more detail below.

There already exist a number of different, quite large datasets such as PASCAL VOC, MS COCO and ImageNet. These are made for different tasks however, so not all can be used for the same type of models. The Pascal VOC datasets contain images with labeled bounding boxes, and is used for object detection. MS COCO consists of segmented images and is used for both semantic segmentation and object detection. ImageNet is by far the largest dataset, but only consists of labeled images, without information of bounding boxes or masks for objects within them, it is therefore used for image classification, and is not suitable for object detection or semantic segmentation.

2.8.1 Data Augmentation

Augmentation is perhaps the easiest way we can expand a small dataset to possibly give our model better performance. This is because it can be done automatically, randomly and cheaply or almost free. An augmented image is essentially an image that has been manipulated, distorted or altered in a way so that it is different from the original. It is not hard to see how this might help a model to better generalise. A picture of a bottle is still recognisable even if there is added noise, if it is slightly larger in the picture, moved to a different part of the picture or blurred slightly. It is also still recognisable if the colors are scewed, as to mimick different lighting conditions or a different colored bottle. This effectively increases the size of the dataset without having to capture new images and label them.

2.9 Transfer Learning

Transfer learning is a very important and effective tactic in machine learning. In short it is the practice of using weights from a previously trained model as a starting point for training a new model. The benefit of this is that models trained on large, varied datasets usually contain weights that generalise quite well, and can therefore give a much better starting point than initialising the model weights with random values. This also allows a much smaller dataset to be used for training a specific model, since we are only fine-tuning an already well-performing model towards a specific goal.

Chapter 3

Project Focus

This project will focus on two major factors that plays into both how well a plastic sorting model will perform as well as what we can use it for. This will be the dataset that the model is trained on and the method the model uses to differentiate the plastic images.

3.1 Datasets

A very important part of machine learning is the dataset that a model trains on. Supervised learning allows the neural nets to learn by example, so what those examples are, and how many of them there are matters a great deal.
There are a few different factors to keep in mind when building a dataset, most notably quality and cost.

3.1.1 Quality

Given the research questions for this thesis a natural question to ask is what constitutes the quality of a dataset with regards to machine learning. Although this is a rather broad question it would be fair to say that increasing the quality of a dataset would be beneficial to training the machine learning model. This benefit can come in different forms, ranging from confidence in the models performance, increased performance or decreased training time.
A good dataset will cover a wider spectrum of what the model is expected to learn. As an example, if a model is supposed to classify images of different breeds of cats, a good dataset would include pictures of as many different cat breeds as you would like to classify. If the dataset contains too few images of one type of cat the model might not have enough material to learn what differentiates that breed from the rest. A dataset can also have enough images, but too few different examples, if for example too many of the examples in a category is the same instance or individual. This might teach the model what separates that individual from the rest, but due to differences between individuals in the same breed

it might learn the wrong identifiers and suffer on new examples. This will essentially lead to overfitting and bad generalisation of the model.

3.1.2 Dataset Creation Cost

Ideally you would like a dataset that is both big and varied. In the real world however you are limited by how much time and resources you are willing to put into acquiring the data.

This cost is also tied to the method you are using to classify your images. Image classification is the cheapest of the three methods discussed in this project. Image classification only concerns itself with whole images, and as long as there is only one category of object per image, they can simply be sorted in a file structure with one folder for each category. This makes labeling the images very fast and easy, since all that is required is moving it to the correct folder.

Object detection is more costly. This method requires that a bounding box is defined around each object in the image, and to keep track of these a separate annotation file is required. This file is usually in xml- or json-format, since they are easy to read and parsed as objects by programs. To increase efficiency these annotation files should be created automatically given the coordinates for each bounding box. This requires specialised software. Even with this in mind marking out each box in each image of a dataset and labeling it is a tedious, repetitive and time-consuming process. For this reason object detection datasets usually consists of fewer images than those used for image classification. The performance effect of this is mitigated somewhat by the fact that each image can contain multiple objects and therefore be more useful for the object detection than a single image is for the classifier.

Most costly of all these is instance segmentation however. This method requires separate labeling files, same as object detection, but the labeling process is more complex, requiring a person to draw the outline of each instance, not only a bounding rectangle. This is therefore even more time-consuming and laborious than object detection and as a result even more costly.

3.1.3 Computational Cost

The third factor is the computational cost. This is usually the downside of having a large dataset and model, that the bigger the dataset, and the more complex the model, the more time it takes to train. This is usually not a problem, since modern hardware greatly increases computing performance, and training can happen offline before a model is deployed into action. The model size is also usually limited by the computational hardware that is available, and it is easy to justify building as large models as the resources available allow, if it achieves better results. If only to better understand the problem of machine vision. Model size becomes more of a problem when a model needs to be deployed on lightweight hardware, such as a mobile phone, or it needs to run fast or even real-time. Cutting down on training time can also be very valuable, especially when developing a new model, since getting results earlier can allow for faster development with more design iterations.

Augmentation is one factor that can affect this. If an image is augmented every time it is fed into a model it will take up computational resources, and depending on image size

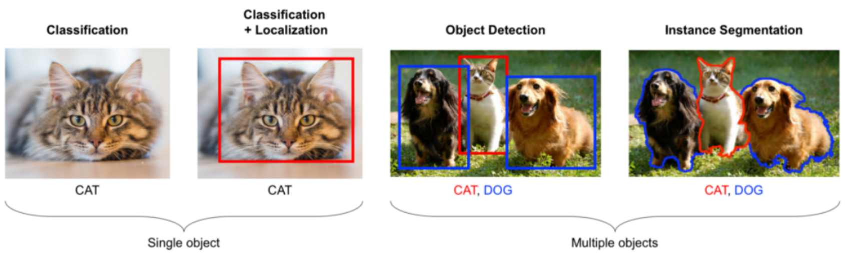


Figure 3.1: Illustration of differences in methods for machine vision

and the amount of filters applied this might be significant.

For this reason a dataset with the necessary augmentations already computed sounds tempting, since the computation would only need to happen once and could be done up front. It could then be skipped for each training session.

This is one of the areas this project seeks to explore, whether augmenting the dataset up-front is a viable option.

3.2 Computer vision methods

We will also be looking at what machine learning method is best suited to categorize plastic. When looking at machine vision there are three main deep learning based approaches; image classification, object detection, and instance segmentation. These three differ in many aspects, but can be thought of as building on top of each other, as illustrated in figure 3.1, with image classification as the foundation, object detection on top and instance segmentation as a more advanced version of object detection, combining semantic segmentation and object detection.

These all have their usecases, and provide a different set of functionality and corresponding value.

Image classification has the advantages that creating a dataset for the purpose is relatively inexpensive, it can be quite accurate and is relatively fast. This method does not however give any information about where in the image the subject is or how many instances there are. If the amount of objects in an image becomes very large it can also confuse the classifier, reducing its accuracy.

This is where object detection comes into play. With object detection you also get a prediction of where in the image the subject is, usually marked by a rectangle(a bounding box) encompassing the object of interest. This can of course be quite useful. Having location information about the subject in the image can both give machines knowledge to act on or give humans confidence that the model is actually looking at what we want it to look at. The method also allows for multiple different objects in a single picture and will draw a bounding box around each object it recognises.

Instance segmentation takes this one step further by not only giving a bounding box around the object of interest, but creating a pixel level mask that outlines the object. This is especially useful if there are multiple objects in an image that partially blocks each other, or if you want to isolate the object from the rest of the image.

3.2.1 Use-case for sorting plastics

If accuracy is good enough then standard image classification would allow us to simply take a picture of a plastic object and get a reliable prediction of what type of plastic it is, and therefor where or how it should be recycled. This could be useful for people, but also for machines if the model was fast enough and images could be taken of each object one at a time.

Object detection could be more valuable, as it would allow us to classify more than one object at once, and also give a machine the positional information needed to separate the plastics, for example on a conveyor-belt. Object detection is also better suited for running on video-streams, which would allow an app to show what type of plastic an object was in real-, or near real-time.

An instance segmentation model would similarly provide even more detailed information for what plastic an object consists of, especially if the object consists of multiple parts.

3.2.2 Choice of Models

We will be looking at classification, object detection and instance segmentation to get a cursory understanding of how well they work for their given plastic sorting problem. There are plenty of architectures to choose from, especially with regards to image classification and object detection. We will be using IRNv2 for Image classification, the SSD-network for object detection and MRCNN for instance segmentation.

The reason for this is that each of these models were state of the art models when they came out, have been around for a while, has readily available weights trained on very large datasets and should give a good indication on whether or not these methods are feasible for plastic sorting.

InceptionResnetv2

InceptionResNetv2 was designed by Szegedy et al. in 2017 [10]. It is the combination of a residual network and the inception network and achieves good results, with reasonable training time. Its structure can be seen in figure 2.1. This project uses the official keras implementation for this model.

SSD-VGG-300

The SingleShot multibox Detector, designed by Liu et al. in 2016[13] is the dedicated object detection model used in this project. The model achieves good accuracy in real-time on both the PASCAL VOC 07- and 12-datasets[13]. The model uses a VGG-16 model as its convolutional network, with an image input size of 300x300px. This project uses a keras-implementation of the model by Pierluigi Ferrari[14].

Mask-RCNN-ResNet101

Mask-RCNN is the instance segmentation model we will be using in this project. It builds upon the previous R-CNN, Fast R-CNN and Faster R-CNN to also predict masks for each detected object[12].The model has a resnet101 model as its backbone and takes input images of size 1024x1024px. The model used in this project is based on the tensorflow and keras implementation made by Matterport[15].

3.3 Comparing the models

These three methodologies all do different things, so comparing them to each other is something that needs to be thought about. All the methods have a way to determine a successful detection, so this could be used as a metric, but this would ignore the benefits object detection or semantic segmentation can give, and the costs they come with in terms of processing power.

Since these methods have such a variation in goals we will mostly be comparing them qualitatively in this project, highlighting how well each perform on their own. We will still strive to give a good representation as to what can be expected from each model, both in terms of accuracy and speed.

The InceptionResNetv2 model will be evaluated based on validation accuracy, while SSD and MRCNN will be judged on mAP(mean Average Precision).

Qualitative judgements will also be made based on how each model predicts classes on completely novel images.

Chapter 4

Technologies

4.1 Development and environments

4.1.1 Programming languages

The programming languages used throughout this project was python 3.6. There are different libraries used in the field of machine learning and deep learning, but there are two main ones: Caffe[16] and Tensorflow[17]. Tensorflow was used for this thesis due to familiarity with the library as well as the python programming language.

4.1.2 Tensorflow

> *"TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications."*[18]

Due to availability of implementations of the different models both tensorflow 2.0 and tensorflow 1.15 was used. The implementations of SSD and MRCNN were run with tensorflow 1.15 due to compatibility and InceptionResNetv2 was run with tensorflow 2.0.

4.1.3 Keras

> *"Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear & actionable error messages. It also has extensive documentation and developer guides."*[19]

The keras versions used was 2.2.4 for the SSD-model, 2.2.5 for the MRCNN model and tensorflows built-in version for the IRNv2 model.

4.1.4 Tensorboard

Tensorboard allowed us to monitor model performance during training, and compare it to other runs at the same time[20]. The reason we used tensorboard was due to its tight integration with tensorflow and keras.

4.1.5 Poetry

Poetry was used to keep track of dependencies for the dataset-creation project. The tool provides requirements-handling and virtual environments to make sure that the project can run smoothly regardless of what machine it is run on. This was very helpful and made working on the project from different machines much easier. This specific tool was chosen due it being currently supported and developed, as well as its similarities to what I had been using earlier.

4.1.6 Anaconda

Python distribution tailored specifically for data science. Contains its own package manager as well as tools to create virtual environments. This helps keep track of dependencies and make sure that different projects work independent of the machines they are run on. For this project a separate environment was used to train and evaluate all three of the different machine learning models. The reason for this was that they were all implemented with slightly different dependencies and separating them allowed changes to be made to one without affecting the others.

Although Poetry(4.1.5) worked great for managing the dependencies of the main project, anaconda ended up being easier to use for keeping each ML model in a separate environment from each other.

4.1.7 Jupyter Notebook

Jupyter notebook is a popular development environment in datascience and prototyping. It allows for interactive testing and writing of code, allowing for each part of a program to be executed by itself, without having to run the entire program each time, load datasets again and again, or download and compile models continuously.

For this reason the SSD model was trained and evaluated using jupyter notebooks adapted for the purpose of this thesis.

Due to familiarity with the InceptionResNetv2 and the training process was made into a more modular script and configural script, granting the ability more seamlessly automate the testing process.

The MRCNN was trained and tested similarly to the InceptionResNetv2 model, but with less strict automation capability. This was due in part to limited time for such development and less need for it, as the overhead of manual testing decreased in comparison to the time each test took.

4.1.8 Visual Studio Code

Visual studio code is Microsoft's free code editor. It is lightweight compared to a lot of other IDEs, like Visual Studio and Pycharm, but also powerful and feature-rich enough. This IDE was chosen out of personal preference, and does not impact the results shown in this thesis.

4.2 Labeling tools

When labeling images for object detection or semantic segmentation having a dedicated program for the task becomes important for efficiency and accuracy. When using a classic model such as IRNv2 that takes an image and only decides what category it belongs to, you can quite easily label the training-data by simply placing each category of image in a different folder. This is easy and works quite well, as such keras and tensorflow have functionality to use this type of labeling structure.

This is not the case with object detection and segmentation, since these techniques not only gives a prediction of what category of object is in the picture, but where in the picture it is. As such these techniques also require label files for each image file. These files are often either XML-files or JSON-files for newer datasets. These files contain information with regards to what image the label is for, what objects are in the image and coordinates for the shapes that surround them, whether that is bounding boxes for object detection or polygons for segmentation. These files are not impossible, but impractical to create manually and are designed to be created and read by machines. The machines can sadly not label these for us, since this is the problem that is being solved in the first place. So the solution is simply to create a program that lets a user see an image and place rectangles or other shapes around the objects of interest in the image, that the program can then turn into a valid XML- or JSON-file. This is not a hard problem, but usually not a set of functionality found in regular image editing software. Luckily there already exist several options to choose from that does exactly what we want.

4.2.1 LabelMe

The choice for annotation software fell on LabelMe for this project. LabelMe is a free tool developed by Kentaro Wada and is available on github at time of writing[21]. After installing LabelMe the process of labeling images began. This was a slow and repetitive, but not difficult process. The software was easy to use, but as discovered later in the project, rotated the images according to their exif-information without transforming the annotations in the same way.

This ended up causing a problem as some of the image masks and bounding boxes were rotated 90 degrees with regards to the images when the models used them for training.

This was solved by rotating the images again according to their exif-tags in the dataset-generation process, but it is worth to keep in mind for the future use of this software.

4.3 Development

For the sake of consistency across the tests we needed a way to create and augmented versions of the dataset for each of the different models used in this paper. Although this took some time to complete it was worth it, as new datasets could be generated automatically and consistently.

4.3.1 Dataset augmentor

A framework with this goal was constructed using python to automatically create new augmented datasets from pre-labeled datasets. The framework was made to be modular, readable, extensible and easy to use.

The framework takes different functions and applies them to a dataset and either create a new dataset with copies of the old, or apply them in-place on the dataset. It was made to either apply the transformations all at once, to make a new copy of the original image for each transformation or sequentially, creating copies of the previously augmented images as well.

Although the framework was quite successful, the speed of the program is acknowledged to be sub-optimal. It currently lacks support for multi-threading and since it is written entirely in python there is overhead that can be significantly reduced if optimisation by for example refactoring parts into cython and compiling it to C. For this specific reason, and out of necessity, libraries such as pillow, numpy and opencv were used to manipulate the images. Even with this precaution many of the transformations could take several seconds to perform per image if the image size was above 3000x4000px. For this reason the option of reducing image size to the target input of the model was added, which reduced the time to generate datasets dramatically, especially for the IRNv2 and SSD models which take 299x299px and 300x300px images respectively.

As such the speed of the framework was not a problem for this project, since datasets only needed to be generated a few times, and the amount of images in the dataset was rather small.

Chapter 5

Experiments

5.1 Evaluation methods

Due to the differences between the models, not all metrics were available for all of the models. For the IRNv2 classification model validation accuracy and loss were used as metrics to monitor the models performance. For the SSD object detection-model only train and validation loss were used for realtime monitoring, while mAP was intended to be used for evaluation of the models performance. For the the MRCNN instance segmentation model accuracy and loss as well as bounding box loss, class loss and mask loss were monitored with tensorboard. To evaluate the model mAP was calculated on both the training and validation datasets.

5.2 Experiment setup

All experiments were performed on a desktop computer outfitted with an 8-core *Ryzen 3700x* CPU, 16GB RAM, and an *Nvidia RTX 2060 Super* GPU with 8GB video memory. All models were implemented in keras and tensorflow and run within an Anaconda virtual environment to isolate their dependencies. The training was performed using CUDA on the GPU to speed up the process.

Three large experiments were conducted with the intent of seeing how different models performed on different datasets. The models are trained to separate 6 different types of plastic, namely HDPE, LDPE, NA(Other), PET, PP and PS. The object detection and segmentation models also included a class for background.

Since part of the goal was finding guidelines for which augmentations could be used to expand small datasets, the original dataset was quite small, only containing 504 pictures. These images were then multiplied according to how many augmentations were applied and in what way.

5.3 Datasets

The original dataset consisted of 504 images of plastic items from the six different classes 5.2.
These pictures were manually segmented using the LabelMe software to produce label-files in *.json* format for each of the images. Bounding boxes were then calculated and *xml*-versions of the labels were created. This resulted in a dataset that could be used for each of the three networks.

Four new augmented datasets were then created with this as the origin. The transformations used to create these were xy-shifting, blurring, hue-shifting as well as adding noise. Examples can be seen in figure 5.1. Paramters for each of the transformations are described in table 5.1.

Transformation	Parameters	Description
XY-Shifted	Horizontal range: [-0.20, 0.20] Vertical range: [-0.20, 0.20] Mode: Mirror	The image was shifted a random fraction of the width and height of the image determined by the input ranges. Empty space was filled by mirroring the original image.
Blurred	Gaussian blur Radius: 3	The image was blurred with a gaussian blur of radius 3.
Hue-shifted	Hue range: [-180,180]	The image was converted to HSL format and the hue channel was shifted by a random amount between -180 and 180.
Noisy	Gaussian noise Mean: 0 Standard deviation: 0.1	Gaussian noise was added to the image with a mean of 0 and standard deviation of 0.1.

Table 5.1: Transformations that was used and the parameters they were used with.

The images were shifted in the x and y direction, to ensure that the model did not depend on the object being centered to detect it. This was not a large concern since all the models used in this project are able to recognise features in the entire image. Blurring the images was also done to ensure the model was not dependent on super sharp images, possibly allowing it to categorize based on other parameters, such as shape, and hopefully to allow classification from lower quality photographs. A radius of 3 was determined to be a good value for the gaussian blur, since the resulting blur was significant, but not too much to not recognise the shape of the object. Shifting the hue was done to ensure the model were not dependent on the color of the object, seeing as all the different plastics can have all sorts of colors. This could also simulate the image being taken under different lighting conditions. Adding noise to the image was also done to make sure the models were robust on lower quality images, and to induce randomness into the image.

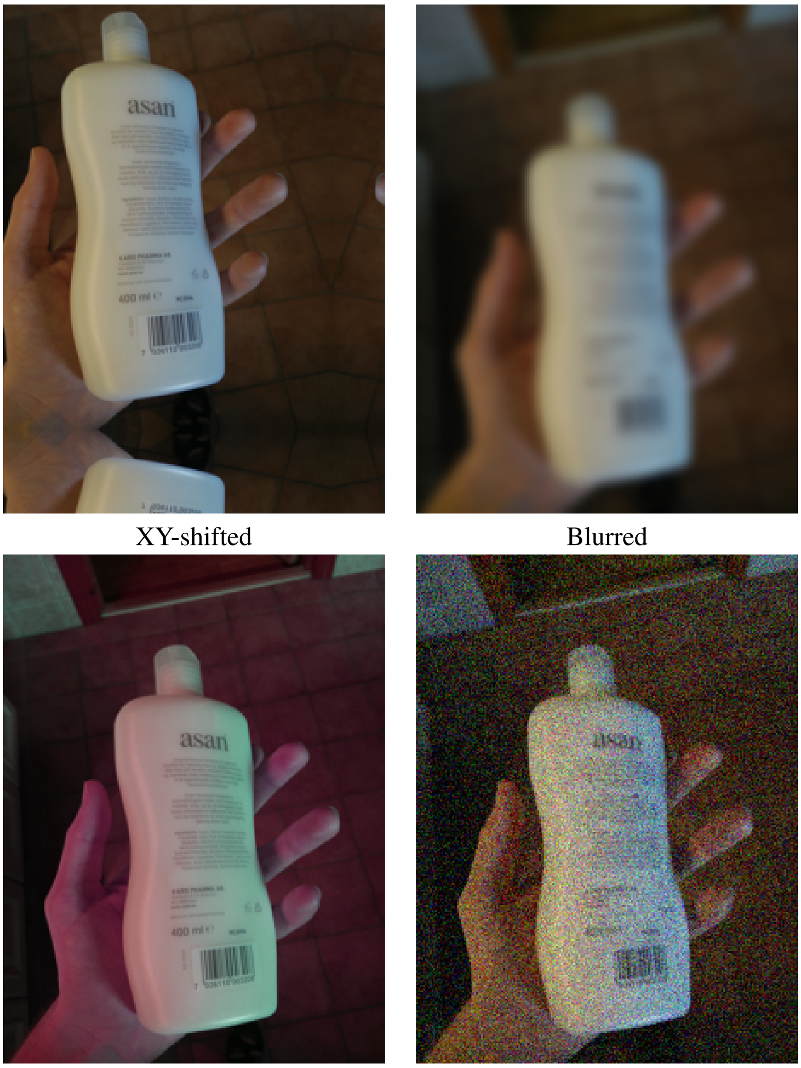


Figure 5.1: Examples of augmented images

The resulting four datasets were 2x the size of the original and contained the normal images as well as a version of the image with the respective augmentation applied.

Of course, if either of these augmentations increased the models performance it would of course be interesting to see what would happen if they were applied together. For this reason two additional datasets were made. One with blur-, hue-shift- and noise-transformations applied to the original, the other with blur and hue-shift applied to the original and noise applied to the the resulting set.

The first dataset was created a 4x-size dataset, by applying the augmentations to the originals. It contained the original- as well as a blurry version, a hue-shifted version and a noisy version of the images.

The other one created a 6x-size dataset, by first applying the blur- and the hue-shift transformations to the original, creating a 3x size dataset, then doubling the size by creating noisy versions of the 3x-dataset. This order of applying the transformations was decided on because of the fact that blurring the noised images would somewhat negate the effect of the noise, and shifting the hue too many times seemed unnecessary.

All the datasets were scaled down to the max input size for each of the models. Images for IRNv2 was scaled to 299x299, images for SSD to 300x300 and the images for the MRCNN model was scaled to 1024x1024 The resulting datasets(table 5.2) were used to train the models.

Splits of 10% were taken from the end of the datasets to be used for validation.

Dataset name	**# Train images**	**# Validation Images**	**Description**
Normal	453	51	Unaugmented, original images
XY_shift	906	102	Images randomly shifted in x- and y-direction as well as originals
Noisy	906	102	Images with randomly added gaussian noise as well as originals
Blurry	906	102	Images with an applied gaussian blur as well as originals
Hue_shifted	906	102	Images with their hue-channel randomly shifted as well as originals
Blur_hue_noise	1812	204	Images where blur, hue-shift and noise had been applied separately to the originals, as well as the originals
Noisy_ blurhueshift	2718	306	Original images, blurred and hue-shifted versions, as well as copies of these with applied noise.

Table 5.2: Descriptions of the 7 different datasets used in this project

5.4 Experiment 1 - InceptionResNetv2

Previous work had have already shown some promise for this model, with regards to classifying plastic [5]. In this project we will not be going in depth on what can be done to improve this models results specifically, but rather look at how it benefits from different augmentations to the training dataset and how it performs compared to object detection and instance segmentation. The model was therefore trained on all the 7 different datasets — the original as well as the augmented sets — and the resulting loss and accuracy values were compared.

5.4.1 Hypothesis

We expect to see an increase in accuracy and a decrease in loss on models trained on augmented datasets, in comparison to the model trained on the unaugmented dataset. We would also expect that models trained on datasets combining several augmentations would outperform the ones with only a single augmentation.

5.4.2 Setup

The models was trained with these hyperparameters:

- model: InceptionResNetV2
- optimizer: SGD
- epochs: 60
- batch size: 16
- leaning rate: 0.01
- validation split: 0.2
- loss_function: categorical_crossentropy
- Datasets: as described in table 5.2

5.4.3 Results

The results were shockingly good, with an improvement from the un-augmented dataset giving an accuracy below 60% to the best augmented dataset giving close to 93%. Notably the model with 1800 images was the one that performed the best with a validation accuracy of 93%. At first glance this would seem to indicate that stacking augmentations ontop of each other might not be necessary to achieve good results, and that simply adding them one by one to the original dataset might be sufficient or even better.

This did however seem too good to be true. Although it would be expected to see an improvement for the models, the results shown in figure 5.2 was far outside expectations. A theory for this vast improvement in accuracy and loss is that the test and validation data are being mixed together and artificially increasing validation accuracy. Since the augmented images are not too different from the originals it would stand to reason that if the model got better at identifying an image it would become better at identifying an augmented version of the image and vice versa. Furthermore since the training and validation splits were at first picked at random from each dataset it would seem likely that the validation set could end up containing versions of an image that also existed in the training set. To be certain that this was the case another round of tests were initiated. This

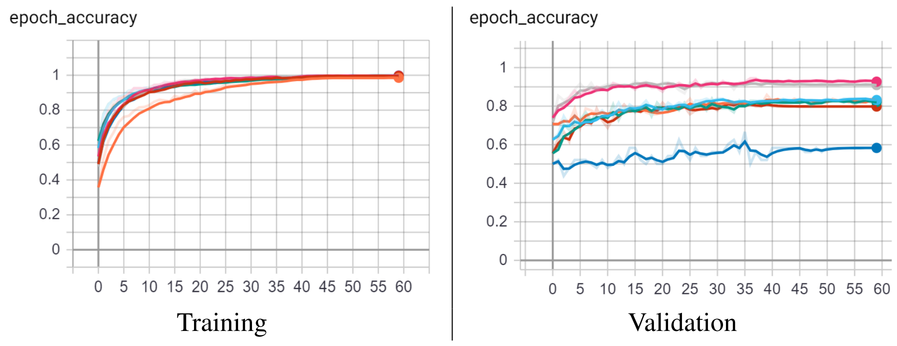


Figure 5.2: Training and validation accuracy of first models trained. Validation set had leaked into training set with more augmentations, and the apparent increase in accuracy can be seen in the pink and grey lines of the validation data.

involved creating a separate validation set that all the models would use, regardless of what augmentations had been applied to their training dataset. Due to a lack of images in the PS and NA classes, some images were removed from the training-set and put into the new validation set. New images were then gathered for the other classes, resulting in a training set containing 489 images and a validation set containing 115 images. The augmented datasets were then re-generated from the new training-set and new models were trained. This time all models were validated on the same, unaugmented testset.

The results clearly showed that there was a difference. All models performed similarly, with none of the augmented datasets significantly outperforming the others. Even more, none of the models trained on augmented datasets outperformed the original unaugmented one, on either loss or accuracy, as seen in table 5.3.

Dataset	Best validation accuracy	Best validation loss	End of training validation loss	Training time 60 epochs
Unaugmented	0.600	1.282	1.633	8m
XY-Shifted	0.574	1.160	2.157	13m
Blurred	0.600	1.425	2.303	13m
Hue-shifted	0.591	1.503	2.196	13m
Noisy	0.557	1.490	2.268	13m
Blur_hue_noise	0.583	1.443	2.180	22m
Noisy_BlurHueShift	0.565	1.461	2.015	31m

Table 5.3: Validation accuracy, loss and time to complete for each model.

5.4.4 Discussion

This was quite interesting, as it goes against our prior belief that augmenting the dataset, and thus increasing its size, would improve the models performance. What's more is that the models quickly started to increase their loss, giving indications of overfitting faster on

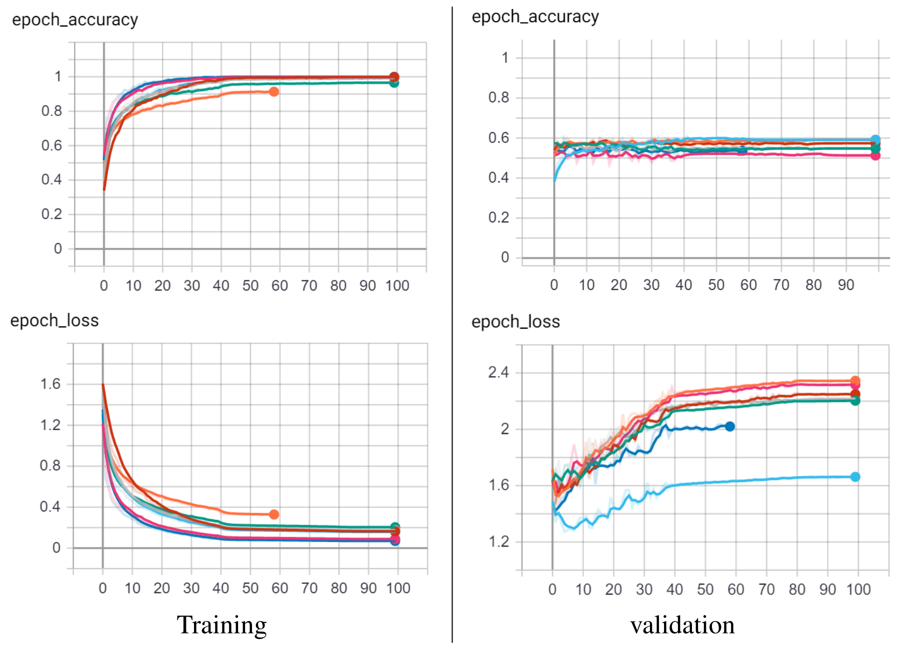


Figure 5.3: Training and validation accuracy and loss of models run on the datasets with strictly separated validation set.

the augmented than on the original dataset.

This hints at a problem with this way of augmenting data.

Since none of the augmentations were very intrusive or destructive, and since they are not changing from epoch to epoch, it might be that the augmented images essentially acts as copies of the original images. This could explain why the model seemed to react quicker, but end up in a similar steady state as with the original dataset. It might be that the result of extending the dataset this way is that the model essentially trains two epochs in one. This would result in overfitting after fewer epochs if the dataset was too small, but perhaps the same amount of time. With the dataset having twice the original amount of images each epoch could would take twice as long, but make the model learn more and so could end up with similar performance at similar training time.

A way to avoid this would be to add random augmentations to the images at runtime. This would give a greater range of variation, especially as training time increases. The downside of this is that performing each of these augmentations in runtime can add a significant computational load and make training go slower. This could still be worth it, if enough resources are available for training.

5.5 Experiment 2 - SSD

For the next experiment we looked at the SSD model. We used the same original 504 images, but having learned from the last experiment declared a specific subset of them to be the validation set, to make sure that there would not be any bleed-over between the sets when augmenting them.

We used the same augmentations as before with the exception of the xy-shift, since this did not seem to give much value. The validation set was also augmented, but separately to see if there could be an improvement in testing accuracy from this. This gave us 6 of the 7 datasets from table5.2. The datasets for this model was in the PASCAL VOC format.

5.5.1 Hypothesis

We expect to see a decrease in loss, as we use more augmented datasets, if augmenting the images has a positive effect. If not we expect to see similar results to experiment 1 in 5.4.

5.5.2 Setup

The model we used for this experiment was the SSD-300 model as described by Liu et al. [13] and implemented by Pierluigi Ferrari[14].
The model has been adjusted for this usecase by reducing the output classes from 21 to 6. Hyperparameters were as follows:

- model: SSD-300
- optimizer: Adam
- epochs: 60
- batch size: 16
- leaning rate: 0.001
- validation split: 0.1
- loss_function: SSDLoss with neg_pos_ratio=3 and alpha=0.1
- Dataset: Described in table 5.2

5.5.3 Results

We saw improvement from augmentation to augmentation in relation to the unaugmented set, the largest relative improvement came from the Blur_hue_noise dataset. The loss was however still quite high, and what we saw when closely examining predictions and comparing them to the ground truths was that some of the ground truths were flipped 90 degrees from what they should be, and thus were not alligned with the image.

The cause of this was found to be that the software used to label the images, LableMe4.2.1, had rotated images according to their exif-tags when displaying them to the user, but not rotated the labeling files accordingly when they were saved.

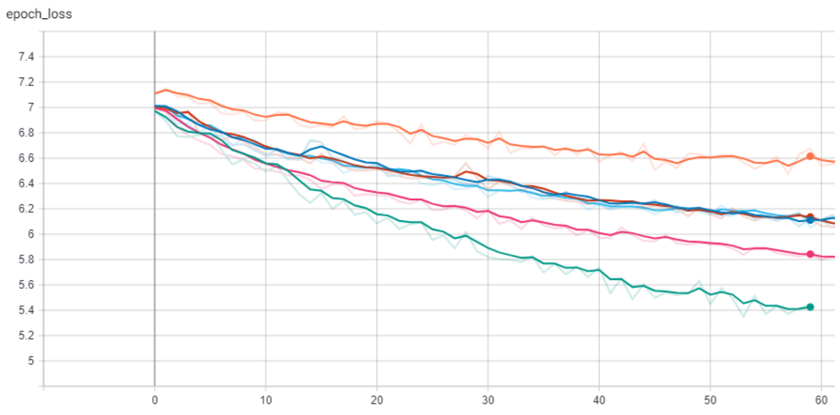


Figure 5.4: Training loss of models trained on compromised dataset as well as corrected(bottom green).

This was rectified by algorithmically rotating the images to their correct orientation in the dataset-generation process. The datasets were re-generated, this time with all labels correctly oriented and the model was trained on the datasets again.

What was immediately clear was that this error had been destructive to the performance of the model, as this time the unaugmented dataset got better results than the best models from the first round, as shown in fig. 5.4.

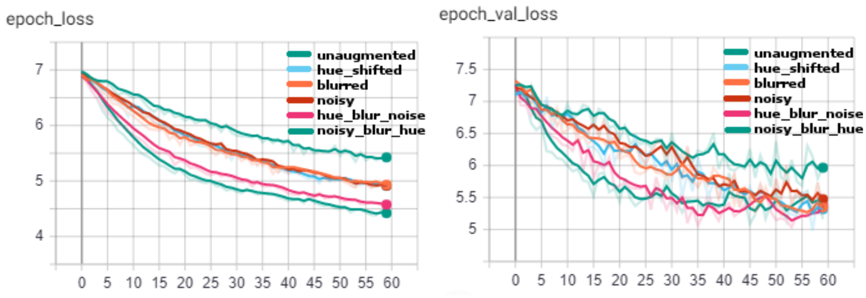


Figure 5.5: Training and validation loss of models trained on corrected dataset.

We also found both the blurred, noised and hue-shifted datasets outperform the unaugmented, and the combinatory sets outperform these. The loss did still seem a bit high, and the validation loss seemed to jump erratically up and down with time. At the end of training all the augmented models had very similar loss, see table 5.4. Interestingly the model trained on the hue-shifted dataset had the best loss of all the models, although the model trained on the Noisy_BlurHueShift set ended with the best score.

What we saw when we later loaded the models in inference mode was that it would rarely give an actual prediction on the test images. For this reason we were not able to get an actual measurment for the models mAP.

Dataset	Best training loss	Best validation loss	End of training validation loss	Training time 60 epochs
Unaugmented	5.445	5.681	5.983	33m
Blurred	4.889	5.233	5.226	1h 9m
Hue-shifted	4.877	4.984	5.274	1h 9m
Noisy	4.898	5.327	5.327	1h 9m
Blur_hue_noise	4.548	5.035	5.535	2h 16m
Noisy_BlurHueShift	4.404	5.085	5.164	3h 24m

Table 5.4: Best training and validation losses for each trained model, as well as training time

5.5.4 Discussion

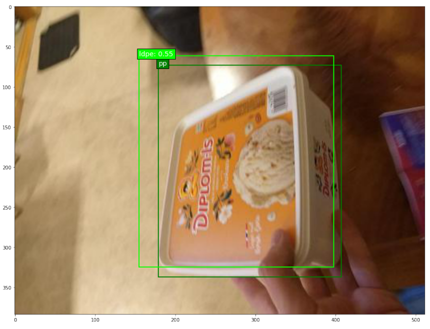
The results of this experiment once again shows how important the quality of a dataset is. Having bad data in the dataset is not only providing zero value, but actually hurts the training of the model. This is not surprising, and nor is it the intent of the experiment to show this, but it is good to see that our intuitions hold true.

The real result of the test is more interesting. Once again we see that the augmented models used fewer epochs to reduce loss,to complete each epoch. We still see the augmented models end up with a similar loss-value, which is still very high, but the unagumented model now performs worse than the rest, in contrast to experiment 5.4 where we actually saw the unaugmented model have lower loss, and slightly better accuracy than the augmented versions.

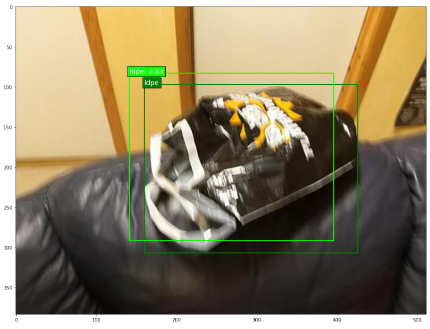
It is not immediately clear why this is the case, but one theory as to why the loss is so high might be the way that the classifying weights were sampled to give only 6, instead of 21 active classes.

The high loss also explains the lack of predictions, and the lack of quality of predictions, that the model was able to output. Some examples are displayed below in figure 5.6, with examples of predictions from the validation set.

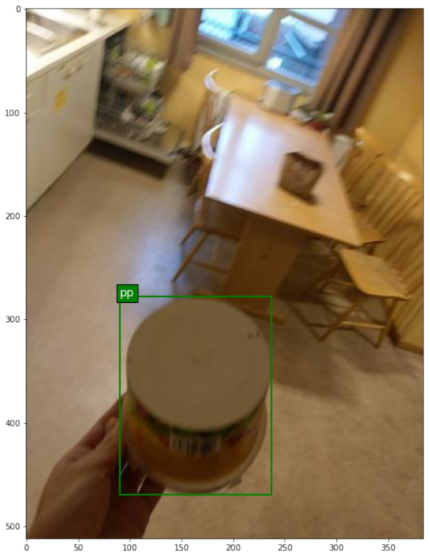
It seems safe to say that this model needs a larger dataset than the one we had available to achieve the desired level of performance.



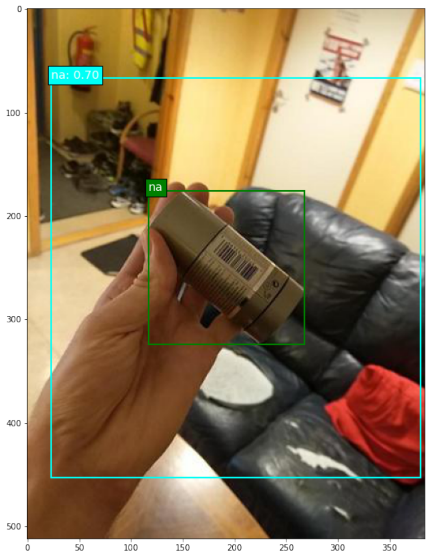
Wrong class, good bounding box



Correct class, good bounding box



No prediction



Correct class, bad bounding box

Figure 5.6: A sample of cherry picked predictions given by the SSD models. The model failed to give a prediction on most of the test-images.

5.6 Experiment 3 - MRCNN

For the last experiment of this project we looked into instance segmentation with the mrcnn model. What this model should allow us to do is get both a bounding box as well as a mask of the plastic item, highlighting where in the image the plastic item is, as well as what type it is.

We used the same 6 types of datasets for this experiment as was used in 5.5, but regenerated in the format used by the mrcnn model, the MS COCO format.

Having learned from the previous experiments we do not expect to see much improvement by adding the augmentations, but will still perform the experiment to make this clear.

5.6.1 Hypothesis

We would expect not to gain much in terms of performance by adding the augmented images, but expect to see the model train on fewer epochs.

5.6.2 Setup

This experiment was performed with Matterports implementation[15] of the Mask-RCNN model described in by He et al. in [12]. All parameters not listed below were left at their default settings, but the number of classes was set to 6 + 1 (plastic types + background). The experiment was run once for each of the datasets, all starting with the same weights. These were the weights pretrained on the COCO database.

Hyperparameters were as follows:

- model: MRCNN
- backbone: resnet101
- epochs: 30
- gpu count: 1
- images per gpu: 1
- leaning rate: $epochs < 20 : 0.001$; $epochs >= 20 : 0.0001$
- validation split: 0.1
- Dataset: sets 1,3,4,5,6 and 7 described in 5.2

5.6.3 Results

We saw mostly the same results as in 5.5, with none of the models outperforming the one trained on unaugmented images, although they all seemed to train faster and decreased their training loss faster. The loss was still significantly lower with all datasets on this model in comparison to both the image classification models and the object detection models in 5.4 and 5.5 respectively. The MRCNN came out with a low-pointcoming out at a lowpoint of 0.55 for validated, on epoch 15 of training with the standard unagumented dataset.

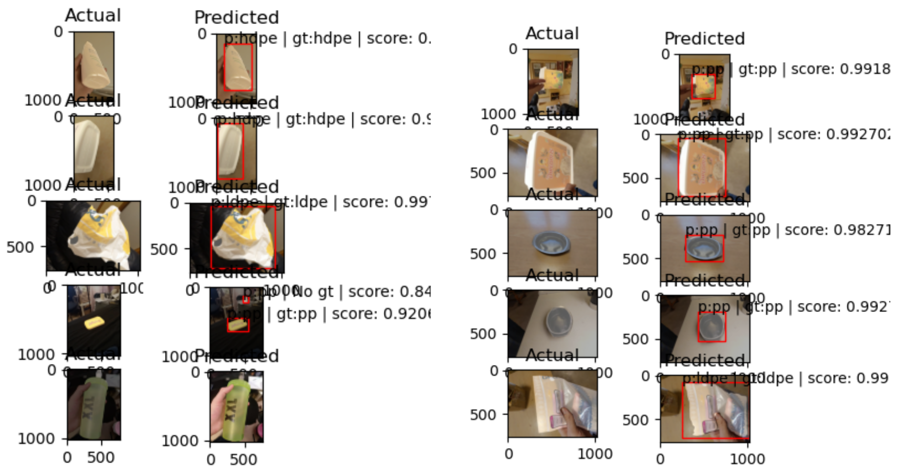
Changing the learning rate to decrease in the latter part of training also smoothed out the loss-curves and seemingly improved the models training.

Dataset	Training mAP	Validation mAP	Training time 30 epochs
Unaugmented	0.425	0.608	1h 42m
Blurred	0.469	0.314	3h 28m
Hue-shifted	0.482	0.343	3h 28m
Noisy	0.497	0.275	3h 28m
Blur_hue_noise	0.501	0.294	6h 51m
Noisy_BlurHueShift	0.509	0.324	13h 6m

Table 5.5: Values for training and validation mAPs, as well as total time trained

MAP also varied greatly for the models trained on the different datasets, results shown in table 5.5. Although the mAP might seem low these results are actually promising, when considering the small dataset and that the original paper achieved an AP of 37.1 and an AP_{50} of 60.0[12].

We also compared ground truths to predictions given by the model, for both the test and validation set, as shown in 5.7.

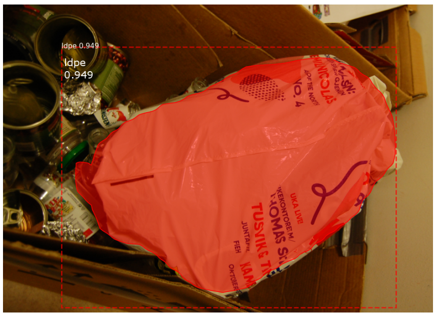

Actual
Predicted
p:hdpe | gt:hdpe | score: 0.
p:ldpe | gt:ldpe | score: 0.99
No gt | score: 0.84
gt:pp | score: 0.920
p:pp | gt:pp | score: 0.9918
p:pp | gt:pp | score: 0.99270
p:pp | gt:pp | score: 0.98271
p:pp | gt:pp | score: 0.992
p:ldpe | gt:ldpe | score: 0.99

Training set Validation set

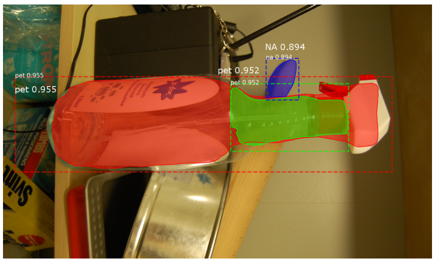
Figure 5.7: Comparisons of ground truths with predictions given by the MRCNN model on training and validation sets

5.6.4 Discussion

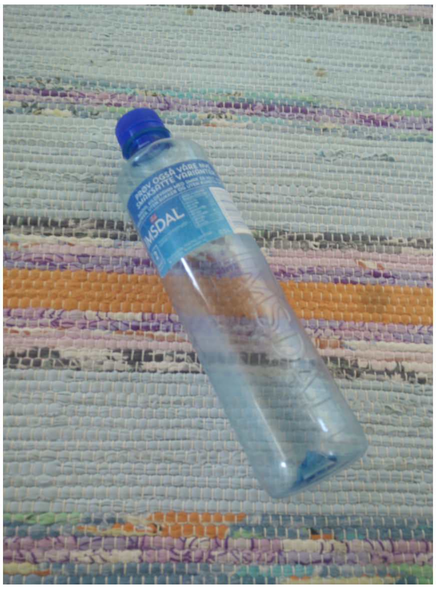
The results from this experiments showed promise for the use of instance segmentation for plastic sorting. The masks generated was very good, although the class predictions were not accurate enough yet, as seen in 5.8. This would be expected to improve with a larger and better dataset. Training times was the only major factor that suffered in comparison to the other models, with the MRCNN needing roughly 24 times longer to train than the IRNv2 model and roughly 6 times more time than the SSD model. This might have something to do with the size of images each model uses, with IRNV2 and SSD using 299x299px, 300x300px images each in comparison to the MRCNN model using 1024x1024px images. This size increase increases loading time, but both the SSD and MRCNN model are more computationally heavy to run than IRNv2.



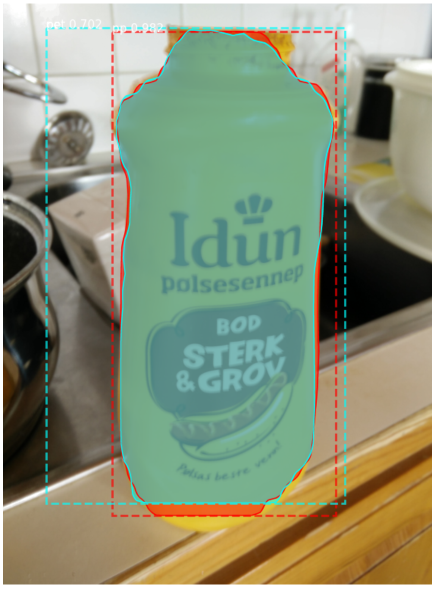
Good mask and correct label



Right class on bottle, but mistakes wood for NA-plastic



No prediction from the model



Both right and wrong classes on top of each other, good masks

Figure 5.8: Examples of predictions given by the MRCNN model on novel plastic items

Chapter 6

Discussion and Conclusion

The results we got from this project was unexpected. Augmenting datasets can be a way to increase the robustness of a dataset, but what this project shows is that the data must not be static, meaning it is augmented once and added to the dataset. Although there might be a performance hit to reprocess an image each time it is given to the model, this might still be a better way than to perform the computations ahead of time. Adding augmented copies of images back into the dataset seems to only make the model overfit faster, and lose accuracy over time.

It is also important to note that even though you might save time for each image that is not reprocessed, adding images back into the dataset increases the size of the dataset, which can quickly lead to long training times. To underline this our MRCNN model trained on the unaugmented dataset took 1 hour 42 minutes to complete 30 epochs on our machine, whilst the model trained on the Noisy_blurHueShift dataset took more than six times as long, 13 hours and 2 minutes, and still performed significantly worse on classifying new items.

To summarize it seems that instance segmentation has potential for the problem of sorting plastics. Given the small size of our dataset it showed quite good results, and with a larger dataset it could very well perform better and be more informative than the other models tested in this project.

Chapter 7

Future Work

7.1 Dataset

The dataset seems to be the biggest problem for sorting plastics with deep learning. As such one of the key factors to better performance seems to lie in getting a bigger, more varied dataset. Suggestions for this might be crowdsourcing image collection or perhaps a collaboration with a recycling plant to gather images of a large variety of plastics.

7.2 Image Classification

The best quality that image classification brings to the table in this context is that datasets are easier to produce, and the model runs faster than the two other models. This does however come at the expense of only being able to classify one object at a time without locating the object in the image. It also seems that a lot more images are required for it to perform at a level necessary to give meaningful value. If given a big and varied enough dataset this approach might still work, and could still be useful if object location is not required. Although we do not view this as the best approach to sorting plastics it might be useful to test alongside other projects as datasets get larger.

7.3 Object detection

For object detection to work for this task the same principle applies as above; more data is needed. The model did show signs of success and did correctly identify several objects, but so far the performance leaves much to be desired. This is very likely to improve with a larger dataset. It might also be a good idea to test out different models to see if there is performance to be gained in both speed and accuracy. Although the model used in this article was state of the art in 2016 much has happened in the field of object detection and a newer model might give even better results.

7.4 Instance segmentation

Although this was the method that took the most resources to train and create a dataset for, it is also arguably the best performing of the three. This fact that the model showed promise in this project even on a quite small dataset is very good news for further development. It seems quite likely that even better performance could be achieved with a larger dataset. Our recommendation for such a dataset is to get as sharp photos as possible, of as many different individual plastic items as possible, to give the model the best possible chance of success. Another reason that we recommend building a larger dataset for instance segmentation is that such a dataset could also be used for both standard object detection and image classification in cases with only one object per image. This means that even though an instance segmented dataset is more resource intensive to build it is also more versatile and can provide value to a wider range of problems.

The method used in this paper also uses larger images as input, which might be useful when needing to distinguish smaller details in the surfaces of plastic items.

For these reasons we recommend pursuing instance segmentation in the quest to automatically sort plastic by machine vision.

Bibliography

[1] Inceptionresnet architecture, accessed: 2019-12-18.
URL `https://1.bp.blogspot.com/-O7AznVGY9js/V8cV_wKKsMI/AAAAAAAABKQ/maO7n2w3dT4Pkcmk7wgGqiSX5FUW2sfZgCLcB/s1600/image00.png`

[2] R. Girshick, J. Donahue, T. Darrell, J. Malik, Region-based convolutional networks for accurate object detection and segmentation, IEEE transactions on pattern analysis and machine intelligence 38 (1) (2015) 142–158.

[3] Instance segmentation with mask r-cnn, accessed: 2020-07-11.
URL `https://medium.com/@jonathan_hui/image-segmentation-with-mask-r-cnn-ebe6d793272`

[4] R. C. Thompson, S. H. Swan, C. J. Moore, F. S. Vom Saal, Our plastic age (2009).

[5] F. Bakken, Master thesis, master thesis at NTNU spring 2019 (2019).

[6] T. Kippersund, Computer science specialisation project, specialisation project at NTNU fall 2019 (2019).

[7] L. P. Kaelbling, M. L. Littman, A. W. Moore, Reinforcement learning: A survey, Journal of artificial intelligence research 4 (1996) 237–285.

[8] T. Hastie, R. Tibshirani, J. Friedman, The elements of statistical learning: data mining, inference, and prediction, Springer Science & Business Media, 2009.

[9] P. Lison, An introduction to machine learning, Language Technology Group (LTG), 1 35 (2015).

[10] C. Szegedy, S. Ioffe, V. Vanhoucke, A. A. Alemi, Inception-v4, inception-resnet and the impact of residual connections on learning, in: Thirty-First AAAI Conference on Artificial Intelligence, 2017.

[11] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, L. Fei-Fei, ImageNet Large Scale Visual

Recognition Challenge, International Journal of Computer Vision (IJCV) 115 (3) (2015) 211–252. doi:10.1007/s11263-015-0816-y.

[12] K. He, G. Gkioxari, P. Dollár, R. Girshick, Mask r-cnn, in: Proceedings of the IEEE international conference on computer vision, 2017, pp. 2961–2969.

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, A. C. Berg, Ssd: Single shot multibox detector, in: European conference on computer vision, Springer, 2016, pp. 21–37.

[14] P. Ferrari, A keras port of single shot multibox detector, https://github.com/pierluigiferrari/ssd_keras (2018).

[15] W. Abdulla, Mask r-cnn for object detection and instance segmentation on keras and tensorflow, https://github.com/matterport/Mask_RCNN (2017).

[16] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: Convolutional architecture for fast feature embedding, arXiv preprint arXiv:1408.5093 (2014).

[17] 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, X. Zheng, TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow.org (2015). URL http://tensorflow.org/

[18] Tensorflow core 2.0, https://www.tensorflow.org/api_docs/python/tf, accessed: 2019-12-15.

[19] Keras, https://keras.io, accessed: 2019-12-15.

[20] Tensorboard, https://www.tensorflow.org/tensorboard/, accessed: 2020-06-30.

[21] K. Wada, labelme: Image Polygonal Annotation with Python, https://github.com/wkentaro/labelme, accessed: 2020-07-11 (2016).