

Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering



Bachelor Thesis

Detecting objects using ONNX in ML.NET

Author: Thomas Stvarnik

Supervisor: Ing. Petr Hanzlík, Ph.D.

© 2021 CULS Prague

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Thomas Stvarnik

Informatics

Thesis title

Detecting objects using ONNX in ML.NET

Objectives of thesis

The main goal of this thesis is to develop an application that can be used to detect, recognize and count different objects within an image. Supporting goals are to explain the way a program can detect and count objects in an image and to take advantage of transfer learning and use a pre-trained deep learning ONNX model in ML.NET.

Methodology

The methodology of the thesis is based on analysis of technical and scientific sources focusing on artificial intelligence, machine learning, and object recognition. Based on the synthesis of the knowledge gained, a prototype ONNX model will be constructed and trained, and C# application using ML.NET implemented to detect objects in an image, classify them, and draw bounding boxes around them. The performance of the application will be assessed in terms of detection precision and classification accuracy.

The proposed extent of the thesis

40-50 pages

Keywords

Deep learning, Machine learning, Image recognition, Image classification, TensorFlow, ML.NET

Recommended information sources

CAPELLMAN, Jarred, 2020. Hands-On Machine Learning with ML.NET. Birmingham: Packt Publishing. ISBN 9781789801781.

CYGANEK, Boguslaw, 2013. Object Detection and Recognition in Digital Images: Theory and Practice. Chichester, UK: John Wiley and Sons. ISBN 978-0470976371.

PRICE, Mark J., 2019. C# 8.0 and .NET Core 3.0: Modern Cross-Platform Development: Build applications with C#, .NET Core, Entity Framework Core, ASP.NET Core, and ML.NET using Visual Studio Code. 4. Birmingham: Packt Publishing. ISBN 978-1788478120.

Expected date of thesis defence

2020/21 SS – FEM

The Bachelor Thesis Supervisor

Ing. Petr Hanzlík, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 23. 2. 2021

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 23. 2. 2021

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 10. 03. 2021

Declaration

I declare that I have worked on my bachelor thesis titled " Detecting objects using ONNX in ML.NET" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any person.

In Prague on 15.03.2021

Acknowledgement

I would like to thank my supervisor Ing. Petr Hanzlík Ph.D. for his thoughtful guidance throughout the whole process of writing and doing research, advices and time.

Detecting objects using ONNX and ML.NET

Abstract

The following thesis is focused on modern machine learning technologies, and the development of a software that demonstrates safety features related to the current pandemic for retail businesses in connection with their existing CCTV infrastructure, using some of these technologies. The thesis will also cover the main stages of development, such as implementing a pre trained CNN model, and using the output to determine the current situation at the store, like the density of people or just the overall number of persons present as well as providing a picture with graphical representation of what the software believes to be a human. The result will be a working application that expects two inputs from the user, and then provides output in two forms. A picture with all the persons detected, and written output that will indicate the state of the situation considering the provided data and input.

Keywords - Deep learning, Machine learning, Image recognition, Image classification, TensorFlow, ML.NET

Abstrakt

Tato práce se zaměřuje na moderní strojové učení, a vývoj softwaru demonstrujícího bezpečnostní prvky pro obchody, za pomoci těchto technologií. Bezpečnostní prvky se týkají stávající situace s pandemií, a jsou lehce použitelné ve spojení s existující infrastrukturou bezpečnostních kamer. Práce také popisuje hlavní kroky vývoje, jako je například implementace trénovaného CNN modelu, a využití jeho výstupu k vyhodnocení situace v obchodě, jako například hustota zalidnění, celkový počet přítomných osob, nebo také ke grafickému znázornění kde program zaznamenává osobu. Výsledek práce bude fungující konzolová aplikace co po spuštění očekává od uživatele zadání dvou parametrů, a následně vrátí dva různé výstupy. Obrázky se znázorněnými osobami, a text v konzoli, indikující stav situace v obchodě s ohledem na poskytnuté parametry.

Klíčová slova - Deep learning, Machine learning, Image recognition, Image classification, TensorFlow, ML.NET

Table of content

1	Introduction	12
2	Objectives and Methodology	13
2.1	Objectives	13
2.2	Methodology	13
3	Literature Review	14
3.1	Machine learning	14
3.1.1	Introduction to Machine learning	14
3.1.2	Use cases of machine learning	14
3.1.3	Tasks of Machine Learning	15
3.1.4	Types of Machine Learning	17
3.2	Artificial Neural Networks	19
3.2.1	Introduction to ANNs	19
3.2.2	Perceptrons	20
3.2.3	The learning process	21
3.3	Convolutional Neural Networks (CNN)	23
3.3.1	Introduction	23
3.3.2	Topology of CNN	24
3.3.3	Architecture of CNN	24
3.4	The YOLO CNN Architecture	27
3.4.1	YOLO types	28
3.4.2	Technologies used in YOLO	28
3.5	ML.Net	30
3.6	TensorFlow	30
3.7	ONNX	31
3.7.1	Technical Design	Chyba! Záložka není definována.
3.8	Technology in retail	32
4	Practical Part	34

4.1	Designing the Safe distance in retail algorithm.	34
4.1.1	Definition of intents	34
4.2.	Configuring the workspace	35
4.3	Implementing the Tiny YOLOv2 model	35
4.3.1	About the model	35
4.3.2	Transforming the input	36
4.3.3	Transforming the output (YoloOutputParser)	37
4.4	Filtering bounding boxes	38
4.5	Drawing bounding boxes	38
4.5.1	Processing a single image	39
4.5.2	Implementation	40
4.6	Implementing the public safety features	41
4.6.1	Determining current state of the shops capacities	42
4.6.2	Warning system for possible neglect of social distancing	43
4.7	Performance	44
4.7.1	Testing Classification Accuracy of the model	44
4.7.2	Overall functionality	45
5	Result	47
6	Conclusion	48
7	References	49
8	Appendix	50

List of Pictures

Figure 1	Percepttron input and output.....	Chyba! Záložka není definována.
Figure 2	Detailed perceptron.....	21

Figure 3 Convolution process + Relu + Sub-Sampling.....	24
Figure 4 CNN representation	25
Figure 5 Max pooling.....	26
Figure 6 YOLO CNN Representation	27
Figure 7 Transformation of the input images.....	36
Figure 8 File paths.....	36
Figure 9 Loading pictures and sending them as input to the model	37
Figure 10 Parsing the model output	38
Figure 11 Resizing bounding box.....	40
Figure 12 Drawing the bounding box onto picture.....	40
Figure 13 Calling the DrawBoundingBox function	40
Figure 14 Saving the output image	41
Figure 15 Output photo	41
Figure 16 Console asking for input.....	42
Figure 17 Console output - shop capacities	42
Figure 18 "if" statements to determine box overlap.....	43
Figure 19 Console social distancing warnings	44
Figure 20 Test run - Console.....	46

List of Abbreviations

AI - Artificial Intelligence
ANN - Artificial Neural Network
API - Application Programming Interface

BCE - Binary Cross-Entropy
CNN - Convolutional Neural Network
CCTV - Closed Circuit Television
CCE - Class Classification Error
FPS - Frames Per Second
IoU - Intersection over Union
MAE - Mean Absolute Error
mAP - mean Average Precision
ML - Machine learning
MCE - Mean Square Error
BoF - Bag of Freebies
BoS - Bag of Specials
ONNX - Open Neural Network Exchange
OS - Operating System
ReLU - Rectified Linear Unit
RGB - Red, Green and Blue
YOLO - You Only Look Once - CNN
UI - User Interface
MB – Mega Bytes

1 Introduction

In the past year, covid-19 has been a major influence on every part of people's lives. In an effort to minimize the spreading of the disease, the government decided to enforce certain rules for public spaces, whether it's at the office, a mall or a shop. This has forced a large part of the population to be even more involved with smart devices and different technologies that heavily rely on machine learning of different kinds, without them even realizing. In this thesis, we will focus on a problem that emerged for retail businesses during this time, and that is keeping track of how many people are allowed to be present at any given time in their stores. And ensuring a safe environment for their customers. We have seen many different solutions to this problem. There are stores that have decided to use cards with numbers on them and distributed them to every customer that entered, or they used shopping carts to track the number of customers in their store. In this thesis, we will explore another way to solve this problem, and explain the underlying logic and technologies used for such tasks in today's world.

2 Objectives and Methodology

2.1 Objectives

The main goal of this thesis is to develop an application that can be used to detect, recognize and count different objects within an image. Supporting goals are to explain the way a program can detect and count objects in an image and to take advantage of transfer learning and use a pre-trained deep learning ONNX model in ML.NET.

2.2 Methodology

The methodology of the thesis is based on analysis of technical and scientific sources focusing on artificial intelligence, machine learning, and object recognition. Based on the synthesis of the knowledge gained, a prototype ONNX model will be constructed and trained, and C# application using ML.NET implemented to detect objects in an image, classify them, and draw bounding boxes around them. The performance of the application will be assessed in terms of detection precision and classification accuracy.

3 Literature Review

3.1 Machine learning

3.1.1 Introduction to Machine learning

Machine Learning is the use and development of a process that is able to improve and adapt, without explicit instructions. The data used and created is not random, it contains some structure that can be used to predict outcomes and probabilities or knowledge. The idea is to use past experiences, which are in the form of structured data, to optimize a performance criterion. There are several fundamental building stones for the methods used in machine learning:

- Statistics - For reaching a conclusion on evidence from the sample data.
- Numerical Algorithms - For optimizing criteria, and manipulation of the models.
- Computer Science - Data structures and programs that solve Machine Learning problems efficiently.

3.1.2 Use cases of machine learning

Machine learning has a wide variety of use cases.

Cars

This is one of the more talked about use cases at this time. Because of machine learning, cars nowadays are oftentimes semi autonomous, or at least use some driving-assistance systems that rely on machine learning, like for example lane assist or active cruise control. Fully autonomous vehicles are definitely the feature of automobiles, and we have machine learning to thank for that.

This could also help make cars safer, and more efficient in the future, as The National Highway Traffic Safety Administration says “The critical reason, which is the last event in the crash causal chain, was assigned to the driver in 94 percent of the crashes.” (National Highway Traffic Safety Administration, 2015)

Medical Diagnosis

“Machine learning has demonstrated truly life-impacting potential in healthcare – particularly in the area of medical diagnosis.” - Hayk Gharagyozyan, (Macadamian, 2019)

With clinical data like for example the DNA sequences, environment of the patients and their symptoms and medical records, machine learning algorithms are able to detect anomalies or patterns, that might indicate a form of cancer or other diseases, and the Doctors can act accordingly. This will hopefully result in more lives saved, with more accurate diagnosis, or more diseases caught in the early stages, where it might be easier to save the patient.

Shopping

With the help of AI, we are able to create a shop that is completely streamlined and autonomous. With a network of cameras and sensors in a retail shop, we can make a scenario happen, where the customer can simply walk into the shop, pick up whatever they need, and leave the shop, without ever standing in line, having to come in contact with a cashier or even a cash register at all. The whole process of charging the customer for the items in their cart can be done automatically with a combination of object detection software using hundreds of cameras, many different sensors like weight and pressure sensors, an account registered to their name and a phone app. The first such shop open to the public has emerged in downtown Seattle in 2018, and it has been set up by Amazon, it is called “Amazon Go”.

3.1.3 Tasks of Machine Learning

The main idea of machine learning is that the system analyzes all the data and makes decisions without or with minimal help from the outside.

We could separate four key tasks, which are :

Regression

is a numerical prediction, the machine analyses all the previous data. For example, it can tell us what are going to be sales this season.

Classification

it assigns the object to already known classes. For example, it can predict if a debtor is going to return money, analyzing his previous history.

Clustering

it is a division of a vast amount of objects into clusters - classes, within one class all the objects have similar characteristics. Division is made by some parametric specification.

Size reducing

it reduces the size of data for better further visualization.

Anomaly detection - searching for exceptional, rare objects which differ from the rest. For example, fraudulent transactions.

Anomaly detection

It is searching for exceptional, rare objects which differ from the rest. For example, fraudulent transactions.

3.1.4 Types of Machine Learning

We can differentiate between 14 types of machine learning, some are algorithms, and some are just techniques used.

Supervised Learning

Supervised learning is the first type of learning problem, where input and output data is given. It is used for regression and classification mentioned above. It is called supervised since the machine is working with a training data set and makes its prediction, corrected/checked by a supervisor.

Unsupervised Learning

The second type of learning problem, unlike supervised, only works with input data. It is called unsupervised because there is no supervisor to correct.

Reinforcement Learning

The third type of learning problem, in this case, works with feedback and must learn from its own mistakes. A great example is a game, where the machine has a task to obtain maximum points so it must find the best way to get it otherwise there is a punishment.

Semi-Supervised Learning

Semi-Supervised Learning is a kind of supervised learning, but here the training set contains a lot of unlabeled data and only a few labeled examples. The goal is to learn not only from labeled examples but also to use unlabeled ones.

Self-Supervised Learning

This type of unsupervised learning with the use of supervised methods, works only with unlabeled data but uses supervised algorithms to solve it.

Multi-Instance Learning

Multi-Instance learning works with a labeled group of examples where it has to understand what makes them belong to the same class, so later it is easier to work with unlabeled examples.

Inductive Learning

Inductive learning is a problem of induction, the machine makes predictions about the future, therefore it has to observe all the previous data given in order to do so.

Deductive Inference

Deductive inference is the opposite of induction, while induction relies on all the data as a piece of evidence, deduction needs to meet all the premises to make a true prediction.

Transductive Learning

Transductive learning operates with specific (training) examples to make a decision about specific (test) examples.

Multi-Task Learning

Multi-Task learning is a type of machine learning where few similar problems must be resolved over one dataset.

Active Learning

“The key idea behind active learning is that a machine learning algorithm can achieve greater accuracy with fewer training labels if it is allowed to choose the data from which it learns. An active learner may pose queries, usually in the form of unlabeled data instances to be labeled by an oracle (e.g., a human annotator).”—
Active Learning Literature Survey, 2009.

Online Learning

This is a type of learning where after or before each prediction the data must be updated since some predictions are made over time for better precision.

Transfer Learning

Transfer Learning is the type of learning when we train our model on one task then move it to the other one similar to it with for example different dataset so it learns and it can be transferred multiple times.

Ensemble Learning

The purpose of Ensemble learning is to get the optimal outcome by combining predictions of multiple models over the same dataset.

3.2 Artificial Neural Networks

3.2.1 Introduction to ANNs

An Artificial Neural Network is a supervised learning system built of a large number of simple elements. We call these neurons or perceptrons.

We divide neural networks into 2 categories.

shallow neural networks and deep neural networks.

Both the shallow and deep networks have an input layer and an output layer, but where they differ is the hidden layers.

A shallow neural network has only one hidden layer, whereas a deep neural network would have 2 or more hidden layers, which process inputs.

Although it has been proven that even shallow networks can tackle complex tasks , deep networks have shown to be more accurate, and are improving accuracy with more hidden layers added. But not indefinitely. The limit of layers seems to be around 9-10 layers. After this the predictive powers start to decline, which is the reason for most models today using between 3-10 neuron layers, it is called the gradient vanishing problem, and it will be addressed in a different part of the thesis. The following section will introduce the basic building block for building and training ANNs.

3.2.2 Perceptrons

A perceptron is a binary classification algorithm modeled to emulate the functionality of a neuron in a human brain. They are the foundation of neural networks and have a simple structure, yet have the ability to learn and solve complex problems.

The perceptron can take in multiple inputs, and then provides an output in binary form or a value between 0 and 1, this indicates whether the given inputs represented by a vector of numbers belong to a specific class, or what the confidence level is.

This way, each one of them can make simple decisions, and send them on to other neurons. They are organized into interconnected layers, and together this neural network with enough computing power and training samples, can emulate almost any function and answer pretty much any question.

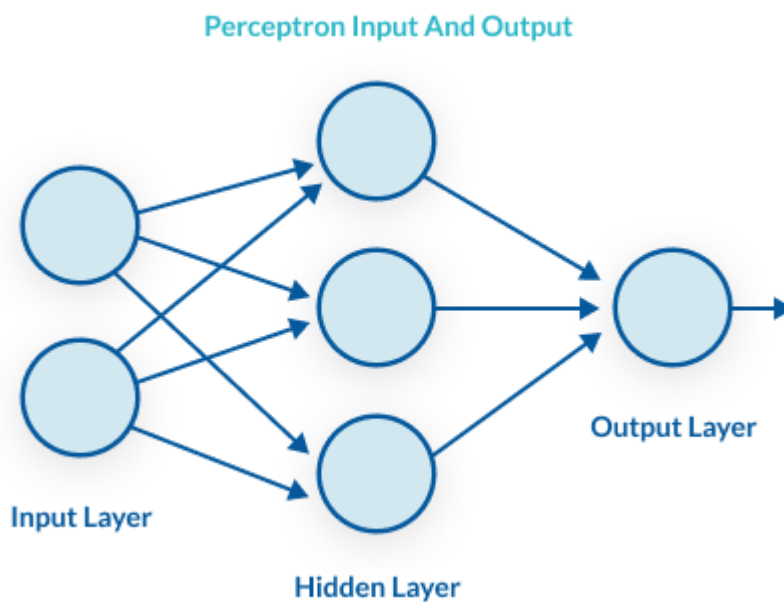


Figure 1 Perceptron input and output (missinglin)

When we add these perceptrons into multiple layers, we call them multilayer perceptrons, and they are very similar to an actual modern neural network. At this point we are missing activation functions, training algorithms(backpropagation) and some advanced architectures, one of which we will address later in greater detail.

for example:

- Recurrent Neural Networks
- Convolutional Neural Networks

- Generative Adversarial Networks

3.2.3 The learning process

The perceptrons take inputs in the input layer, and usually also a constant in the form of a One which is used to move the output function of each perceptron in different directions on the number graph, by changing its weight.

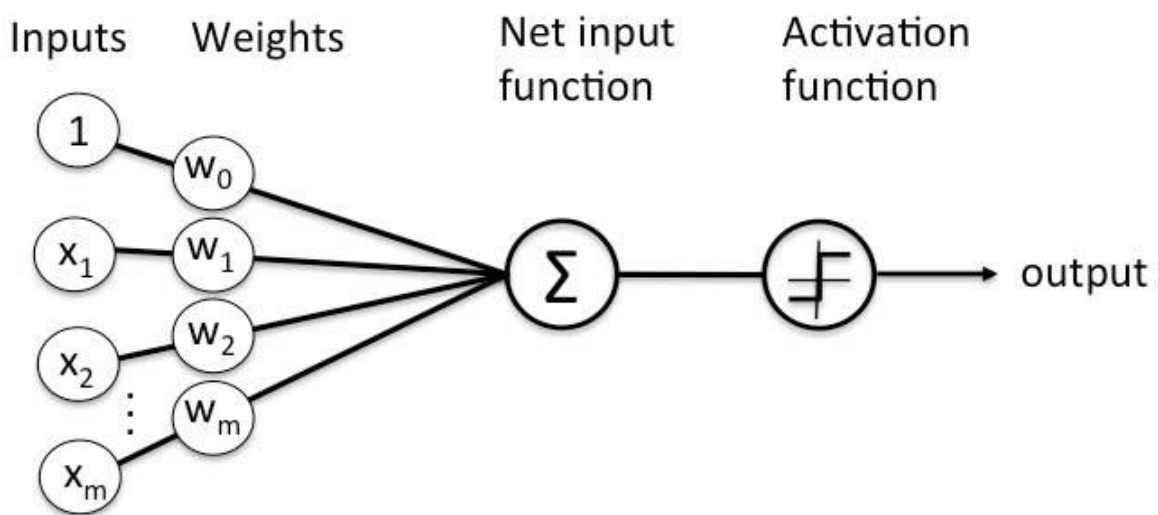


Figure 2 Detailed perceptron (simplelearn)

These inputs are then multiplied by their weight, which is the relative importance to the classification decision, and summed. This sum is then fed into the activation function, which returns the output.

Loss Functions

The output of the loss function is a number. The higher the number, the further away from the correct answer the model is.

There are two main categories of loss functions, and they are both useful in different situations – classification loss functions and regression loss functions.

Classification loss functions are useful for models that are trying to categorize an image for example to a specific group or class from predetermined possibilities.

Here are some examples:

- Hinge Loss Function - The model is penalized when it returns a wrong prediction(-1) or a right prediction without certainty(0).
- Multi Class Classification Function (CCE) - perfect use are multi-class classifications. But it has a requirement - there must be the same number of the output nodes as classes. After the activation stage (SoftMax) the outcome will be a probability that lies in between 0 and 1.
- Binary Crossentropy Function (BCE) - this function is used for binary classifications, for example when we need to divide objects into two classes. When it passes the activation stage (Sigmoid) the output will be either 1 or 0.

Regression loss functions are used when the outcome of the model is a prediction of a continuous value.

Here are some examples:

- Mean Square Error Function (MSE) - as the name suggests it is the mean of squared differences between real and predicted values.
- Quantile Loss Function - this function predicts an interval or a range of predictions instead of only one point prediction.
- Mean Absolute Error Function (MAE) - it is calculated as a sum of absolute differences between the target and predicted value.

Backpropagation

The purpose of backpropagation algorithms is to adjust the weights for perceptron inputs in such a way that the result is as close as possible to the known correct result. Meaning getting the loss function value to as small of a value as possible.

The gradient vanishing problem

With the gradient vanishing problem, we can explain in further detail why we are seeing performance dips after adding more than 9-10 layers to neural networks.

The logic behind this problem is that with every layer added, the amount that the weights of these layers are being manipulated by, is dependent on the partial derivative of the error function. This means that in some cases, the amount that the weights get changed by in the lower layers is vanishingly small, which makes the evolution of the neural network stagnant or extremely slow.

3.3 Convolutional Neural Networks (CNN)

3.3.1 Introduction

Convolutional neural networks are a group of architectures that are very effective in recognizing patterns. This is why they are extensively used for all kinds of computer vision.

The way they work is they use a three dimensional structure with three sets of neurons, each neuron is used for one of the three colors in the basic RGB color model. We can think of this as a filter, that can for example find the edge of an object. It divides the whole picture into many smaller squares, and goes over them one by one, detecting different edges to then send them to a given neuron in the next layer, unlike a standard ANN, which would send the output to all the neurons in the next layer. In each layer we are creating a “feature map”, with probabilities, that the given feature may belong to a desired class, and perform “pooling” which reduces the complexity of the layer output, while maintaining the most important information for the next layer.

A CNN can repeat this process of “convolution” and “pooling” many times, until the features are at the right level of granularity to identify the class of the image, or the objects in the image.

CNN is a key technology in Deep learning since all the best results today are received using it.

CNN is very successful due to its concept and 2-dimensional topology. Considering the size of it, this network has got a relatively small number of parameters that need to be set, unlike its ancestors.

The picture below is a visual representation of how the layers work:

Convolution Process + ReLU + Sub-Sampling

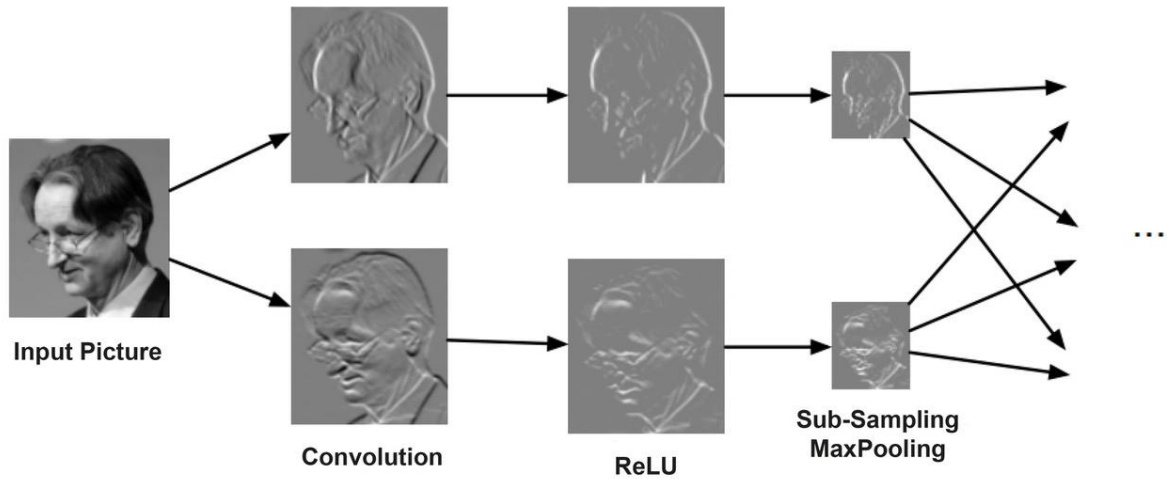


Figure 3 Convolution process + Relu + Sub-Sampling (Habr, 2018)

3.3.2 Topology of CNN

We can distinguish following steps that affect the choice of topology:

- set a task to solve (prediction, classification, detection)
- set limitations (precision of an answer, speed)
- set inputs (type such as image or sound, size, RGB format)

3.3.3 Architecture of CNN

CNN has input, hidden, and output layers. Usually all the calculations such as dot product and multiplication take place in the hidden layer. After follows pooling, activation, fully connected layer, etc.

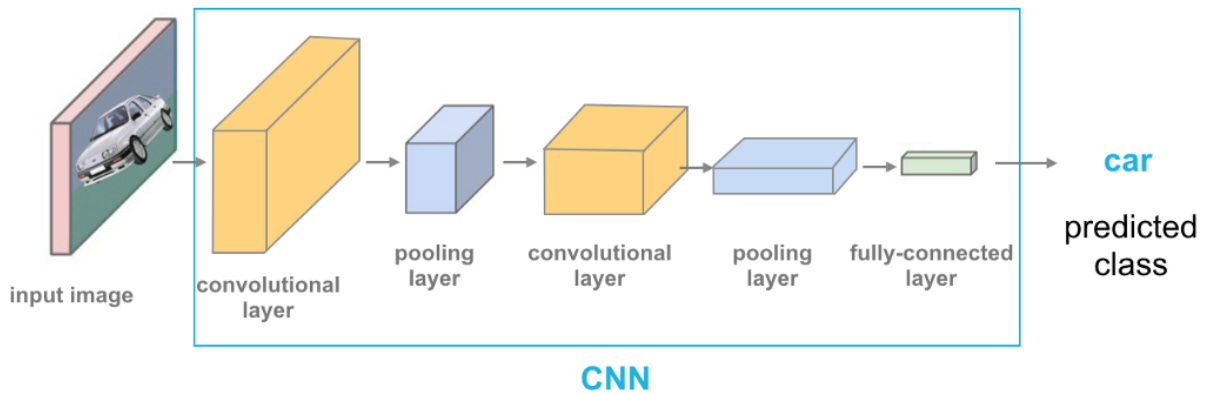


Figure 4 CNN representation (CAMACHO, 2018)

We can distinguish the following types of layers:

Input Layer

Works with input data, which are colorful images. Size requirements must be followed, otherwise, if it's too big - the speed limitation will break so it will not do a prediction in the previously set time, or if it's too small - the network might not catch the key characteristics such as face details.

The picture is split into three channels red, green and blue(RGB model).

Convolutional layer

The convolutional layer is considered to be the main block of CNN. This layer's parameters consist of filters. The quantity of them depends on the precision of recognition, but the complexity will change as well.

Filters go through all the parts of an image and detect particular characteristics, for example, if it was trained on multiple faces - it will detect features such as eyes, mouth, nose, etc. If the size of the filter is too small then it might not detect some features, if it is too big it might create too many connections between neurons.

Pooling layer (Sub-sampling)

The purpose of this layer is to reduce the size of the previous one. On the convolutional level were already found some characteristics, so now we don't need such a detailed image anymore. Using MaxPooling(most common) we reduce the size. Also, some details are not needed anymore, so it is good not to overlearn.

Pooling helps to increase computation power.

- MaxPooling - method where from each region covered by a filter we pick the highest value.

Example:

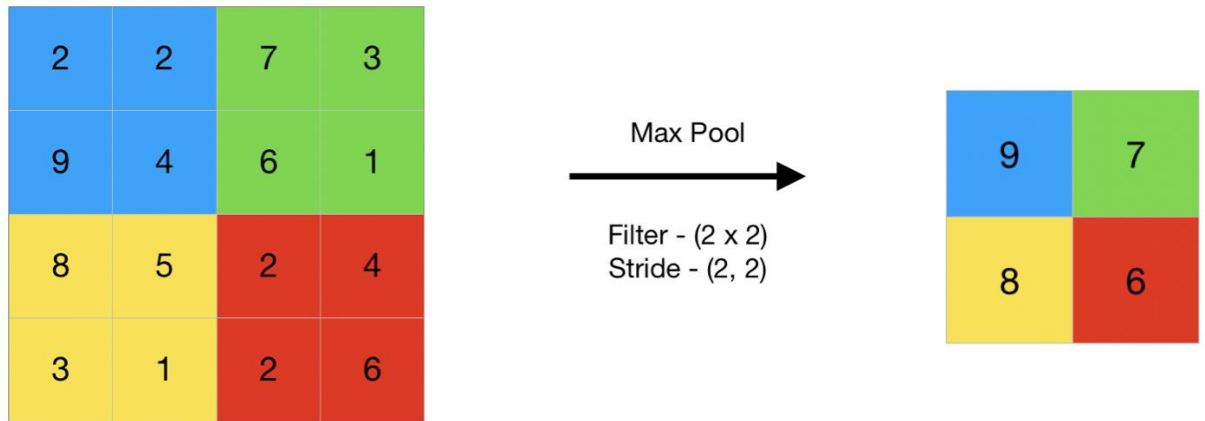


Figure 5 Max pooling (Geeks for geeks, 2019)

Fully connected layers

The purpose of the Fully connected layer is to compile the data from previous layers to form the final output

Activation layer

After every convolution and pooling layer goes activation. There if conditions are not met it returns 0.

Types of activation functions:

- ReLu - is good because it is fast to learn and does not enhance or worsen the gradient. It shows the input straight away if it is positive, otherwise output will be zero.
- The TanH function - simple calculation, strongly -1, 0 and 1. Disadvantage - enhances/worsen gradient.
- The sigmoid function - value interval is from 0 to 1. Has a vanishing gradient problem - input is a very high or low value, so it is hard to make a prediction.

Loss layer

The Loss layer is connected to all the neurons of the previous one. The quantity of neurons is equal to the number of classes we divide it into, for example, two - face and non-face. The Loss layer is also the last layer of CNN.

3.4 The YOLO CNN Architecture

(You only look once, real time object detection)

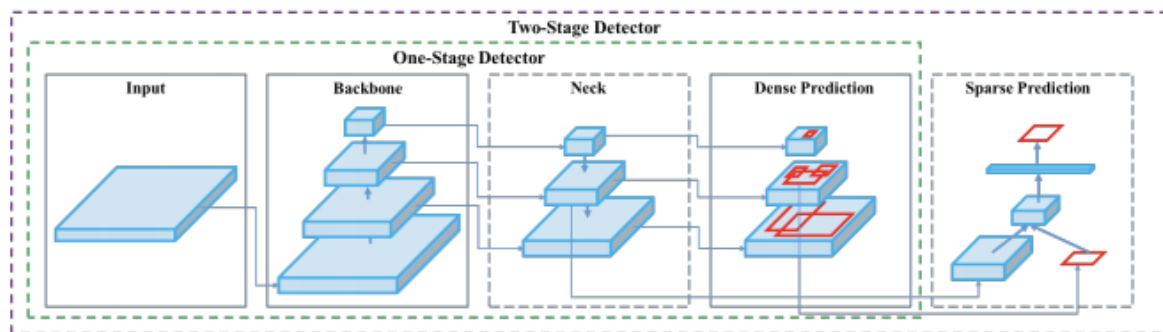


Figure 6 YOLO CNN Representation (RUGERY, 2020)

The YOLO model was the first one to connect bounding boxes with labels.

The anatomy of the YOLO CNN Architecture is following:

- **Backbone**

The backbone consists of mostly convolutional layers. It extracts and forms picture features. This step also mostly affects object detection performance. Data augmentation is also an important part of the backbone. The purpose of it is to maximize the variability of images in order to improve the training process.

- **Neck**

The role of the Neck is to mix and combine image features from different layers and to pass it further for prediction.

- **Head**

In the Head, the final prediction is performed. It picks the right bounding box with an object and shows the most fittable layer.

3.4.1 YOLO types

CNN YOLO gives an opportunity to identify and localize objects using one model.

There are a couple of versions

- **tiny-YOLO** (9 convolutional layers)
- **YOLOv2** (22 convolutional layers)
- **YOLO v3** is in TOP-5 of all CNN's available when we talk about accuracy. For detection, it implies an optimized architecture Darknet-3, which contains 53 convolutional layers and 23 connections.

It is good to use to detect objects on images with a resolution of 4k since this version has it all - good precision and a reasonable computing time.

- **YOLO v4** - is a great improvement over YOLO v3, mAP increased by 10% and FPS by 12%. Although training cost increased as well but not by as much that we would need to reconsider after we compare it to the upgraded performance. What's new is a bag of freebies(BoF), a bag is specials (BoS) and CSPDarknet53 in the backbone.
- **YOLO v5** - compared to all the previous versions is different since it is a PyTorch implementation while the other ones were from Darknet, also it is way faster and lighter than its ancestors with a weight of only 27 MB. What is new is that it has mosaic data augmentation and self-learning bounding box anchors.
- **PP-YOLO** - is based on Parallel Distributed Deep Learning (PaddlePaddle) which is an open-source deep learning platform. This version has implemented a ResNet backbone which makes a decent improvement in performance.

3.4.2 Technologies used in YOLO

Bag of Freebies

BoF - these methods are intended to increase the variability in input set, such as:

- MixUp -linear combination of images;
- Photometric distortion -creates new versions of the picture by changing settings like brightness, hue, saturation, etc.;
- Geometric distortion - cropping, mirroring, rotating image, etc.;
- Focal loss - pictures with an extreme imbalance between foreground and background;
- CutMix - cuts a part of one picture and puts in another one;
- IoU loss - calculates the difference in size and place of given and real bounding boxes;
- Label smoothing - there is no 100% true prediction otherwise the machine might simply memorize but not learn, so here the maximum value is set to 0.9.

Bag of Specials

BoS - new is a Mish activation. It helped to eliminate the dying ReLu problem described before.

CSP Darknet 53

Its architecture was taken from the DenseNet. Its purpose is to take the previous input and link it with the current one before moving it to the next layer, eliminate the vanishing gradient problem as well as reduce the number of parameters.

Mosaic data augmentation

YOLO v5 processes data online and passes it through a data loader. There are three kinds of data augmentation such as scaling, mosaic data augmentation and color space adjustments.

Mosaic data augmentation is basically a combination of four images into four tiles. It is extremely useful when we need to resolve the problem of detecting small objects.

Self-learning bounding box anchors

YOLO v 5 self-learning bounding box anchors work on the custom training data sets which is very helpful for users to work with since their datasets can differ from normal distributions.

3.5 ML.Net

ML.Net is a free machine learning library/software for C# and F# developers. There are quite a few cases where it will come in handy such as image classification, object detection, regression, different types of prediction, sentiment analysis, recommendations, and more. It makes the whole development process a lot easier and faster.

ML.Net is supported by various OSs like Linux, Microsoft, and macOS. Also, it is easy to use since it is possible to use all the libraries and knowledge in the package so all types of developers can use it. Another great bonus is that it is easy to integrate it into other .Net apps.

ML.Net is an extensible platform so it is possible to collaborate with other ML frameworks such as TensorFlow, ONNX or Infer.NET to widen its use cases.

3.6 TensorFlow

TensorFlow is an open-source AI library, an end-to-end platform used to build and implement ML models and multi-layered neural networks.

The main purposes of it are classification, perception, prediction, discovery and understanding.

It uses a high-level Keras API to create and train models.

We can differentiate between following types:

- **DistBelief** - is a machine learning system based on deep learning neural networks, after some time it was upgraded and then became the first

TensorFlow. Later, some new features were implemented such as backpropagation which led to an increase in accuracy.

- **TensorFlow 1.0** - with the second generation it became more versatile. Now it can run on different OSs.
- **TensorFlow Lite** - was created specifically for mobile development. There is also an addition which is a TensorFlow Lite Micro for microprocessors.
- **Tensorflow 2.0** - is the latest major version. Changes were significant - all the old libraries were removed, models became cross-platform, and finally - the scheme has changed from automatic to "Define-by-run".

3.7 ONNX

ONNX stands for an Open Neural Network Exchange is a community open-source AI ecosystem. It can be used with various frameworks and tools so each developer can pick what is best for them. ONNX is a set of blocks for machine and deep learning models. It is designed to maximize the performance of hardware.

"ONNX provides a definition of an extensible computation graph model, as well as definitions of built-in operators and standard data types.

Each computation dataflow graph is structured as a list of nodes that form an acyclic graph. Nodes have one or more inputs and one or more outputs. Each node is a call to an operator. The graph also has metadata to help document its purpose, author, etc.

Operators are implemented externally to the graph, but the set of built-in operators are portable across frameworks. Every framework supporting ONNX will provide implementations of these operators on the applicable data types." (Open Neural Network Exchange, 2019)

3.8 Technology in retail

The retail business, just like any other field, has been adapting and getting better and more efficient with technology since the 19th century.

The Cash Register

The Cash register, invented in 1883, is helping retail shops to keep track of their inventory, cash flow, and helps prevent cashiers to pocket money. With the data from the cash register, shops can also better analyze what customers are buying, and structure their business accordingly.

The charge card (credit card)

The first credit cards emerged in the 1940s. This new way of spending money without having to withdraw it, has a very positive effect on the retail business, as people tend to spend more with their charge cards, as opposed to when they go shopping with cash on hand.

Electronic Article Surveillance

First developed in the 1960s, this easy to implement method of loss prevention has proven very effective over time. Estimates range from 60 to 80% of thefts being prevented.

Barcodes

Barcodes, invented in 1974, have helped shops to keep even better track of their inventories and streamline the process of checking out for all parties involved.

CCTV

Security cameras made their way into retail shops in the 1970s. With the cameras monitoring what is happening in the shop, people are even further discouraged from stealing products or cash from the cash registers.

Member cards and loyalty programs

This practice has its roots in the 17th century. The retailer creates a way to reward repeat customers. This encourages the customers to spend more at the same place, and in return could save the customer some money. More recently, retailers have chosen the form of an account accessible via mobile app, and a physical member card to go along with it. This method has not only the before mentioned benefits, but it also allows the retailer to keep track of what products the person buys, when they buy them, and how they buy them. With proper analysis of this data, the retailers can personalize the experience and maximize efficiency.

4 Practical Part

The practical part of this thesis focuses on creating a program that will be able to help retail shops keep track of the number of customers present at their store, as well as creating a warning system for possible overcrowding of certain areas, and place bounding boxes around the detected people, while using the stores existing CCTV infrastructure as input.

4.1 Designing the Safe distance in retail algorithm.

The algorithm should:

- Ask for the parameters needed to determine the safe number of people present in the specific shop.
- Count the number of people present in the shop.
- Determine if there is a possibly crowded area.
- Provide feedback to the user according to the findings.
- Draw bounding boxes around the detected persons.

4.1.1 Definition of intents

This program is supposed to help keep track of customers in retail spaces, with respect to the current pandemic and the restrictions and safety guidelines that come with that. So the first step should be determining the size of the shop, and how many people per m² are allowed according to the recent restrictions. The program should ask the user for this information.

The program will then take pictures from different parts of the shop and look for persons in the picture using a pre trained tiny YOLOv2 model, which will be counted up, and compared to the calculated maximum occupancy of the shop. If the capacity is exceeded, the user should get a warning.

At the same time, the program is drawing bounding boxes around the persons found in the pictures, and if they should overlap at some point, the program will see this as a possibly crowded area and will warn the user about this situation. This might help with optimizing the flow of customers, or just keep track of the situation in the store.

4.2. Configuring the workspace

The initial draft of the program was designed as a C# Console Application, in Visual Studio 2019,

The next thing I need is to install the Microsoft.ML NuGet packages, Microsoft.ML.ImageAnalytics, Microsoft.ML.OnnxTransformer and Microsoft.ML.OnnxRuntime. After installing all these packages, the environment is ready to use.

4.3 Implementing the Tiny YOLOv2 model

To be able to use the YOLO model, we will need to use the following “using” statements in a file so that we can use all the libraries needed to execute this code.

- using Microsoft.ML;
- using ObjectDetection.YoloParser;
- using ObjectDetection.DataStructures;

The model itself is stored in the assets/Model Folder.

4.3.1 About the model

The model is a standard pre-trained neural network for real-time object detection, which we are using for a specific use case. The model itself is trained to recognize 20 different classes of objects, including things like boats, bicycles, cars and different kinds of animals, as well as persons, which is what we are looking for.

4.3.2 Transforming the input

The model expects an input of a picture with the following dimensions: 416x416, this means that when we load the model, we need to resize the images, and apply the ONNX model after that. For this and other transformations we are using MIContext from the Microsoft.ML imported functions, with parsed values relevant to our model, like for example the image height and width (Microsoft docs tutorial, 2020).

```
var pipeline = mlContext.Transforms.LoadImages(outputColumnName: "image", imageFolder: "", inputColumnName: nameof(ImageNetData.ImagePath))
    .Append(mlContext.Transforms.ResizeImages(outputColumnName: "image", imageWidth: ImageNetSettings.imageWidth,
    imageHeight: ImageNetSettings.imageHeight, inputColumnName: "image"))
    .Append(mlContext.Transforms.ExtractPixels(outputColumnName: "image"))
    .Append(mlContext.Transforms.ApplyOnnxModel(modelFile: modelLocation, outputColumnNames: new[]
    { TinyYoloModelSettings.ModelOutput }, inputColumnNames: new[] { TinyYoloModelSettings.ModelInput }));
```

Figure 7 Transformation of the input images (own)

Implementation

Providing images as an input one by one would be tedious. To automate the process, we need to specify the source of the pictures. For this we create another folder in /assets called "images".

To have fast and easy access to the files in the project, we create variables with the relevant file paths.

```
var assetsRelativePath = @"../../../../../assets";
string assetsPath = GetAbsolutePath(assetsRelativePath);
var modelFilePath = Path.Combine(assetsPath, "Model", "TinyYolo2_model.onnx");
var imagesFolder = Path.Combine(assetsPath, "images");
var outputFolder = Path.Combine(assetsPath, "images", "output");
```

Figure 8 File paths (own)

So that when we want to feed the pictures in the given file to the model, we can simply use the ImageNetData.ReadFromFile function from our imported libraries and load all the pictures from the file into a variable.

The loaded pictures are ready to be used by our prepared functions that will transform the input (the pictures) and run in through the Yolo model.

```

// Load Data
IEnumerable<ImageNetData> images = ImageNetData.ReadFile(imagesFolder);
IDataView imageDataView = mlContext.Data.LoadFromEnumerable(images);

// Create instance of model scorer
var modelScorer = new OnnxModelScorer(imagesFolder, modelFilePath, mlContext);

// Use model to score data
IEnumerable<float[]> probabilities = modelScorer.Score(imageDataView);

```

Figure 9 Loading pictures and sending them as input to the model (own)

4.3.3 Transforming the output (YoloOutputParser)

To transform the data into a more convenient form, we create the YoloOutputParser class. This parser class includes all the different functions needed to extract the information we want from the model output.

The main function's first task is to take the model output, and extract four main pieces of information from each box. Because of the model output being in a 125x13x13 tensor format that gets flattened into an array (Microsoft docs tutorial, 2020), we need to use a function that calculates the offset to access the correct data for each box. Here is the data that we are interested in:

- **Bounding box dimensions** - those are simply extracted into a new object.
- **Confidence** - We use the sigmoid function to get a simple confidence score from 0-1 for each box, which indicates how confident the model is that it has found an object. We multiply that value by the result of the softmax function for the class that was found to be most likely the correct one.
- **Label** - The name of the class that was found to be the most likely result.
- **Color** - Red is used for all the boxes, but it is set to be transparent if the confidence score is below 50%

This information is extracted for each box, and saved into an object, which makes it easy to work with going further.

Implementation

To implement the output transformation, we need to send the probabilities that we received from the model to the YoloOutputParser

```
var boundingBoxes =
  probabilities
    .Select(probability => parser.ParseOutputs(probability))
    .Select(boxes => parser.FilterBoundingBoxes(boxes, 100, .5F));
```

Figure 10 Parsing the model output (own)

4.4 Filtering bounding boxes

The first filtering occurs already in the initial output transformation, where any box that has a simple confidence score (before the multiplication with the most likely class probability) below 0.3, meaning any box that has less than 30% confidence that there is an object present, is filtered out.

The filter bounding boxes function

Because the model will likely find the same object multiple times, with different bounding boxes, we have to filter the boxes in such a way, that each object is accompanied with only one bounding box.

The function looks at all the boxes that we have, and removes the boxes that are overlapping by more than 50% and have a lower confidence score, or are not “persons”, since the model is able to recognize 20 different objects, but for this program we are only interested in the persons detected.

4.5 Drawing bounding boxes

By drawing bounding boxes around the detected persons, we can use the output to be displayed in a UI to provide a more complete picture of what is actually happening at the given store. Another use case is to determine the actual accuracy of the

program, as we can see exactly where the AI thinks that the person is, and we can compare it to reality with our own eyes.

To enable the program to access the prebuilt drawing tools, we need to include the following “using” statements.

- using System.Drawing;
- using System.Drawing.Drawing2D;

4.5.1 Processing a single image

Our function receives

- image file location
- image name
- bounding boxes for the image
- output file location

Our first step would be loading our target image from the file path, and saving the pictures dimensions into variables.

Now the program has everything it needs to proceed to draw the bounding boxes. For this we will loop over all the bounding boxes, and draw them one by one (Microsoft docs tutorial, 2020).

Bounding box dimensions

For every bounding box we need to get the dimensions and the X, Y locations, so that we know where to place the bounding box.

The bounding boxes, however, have been created from the transformed version of the picture which has 416x416 dimensions, but our goal is to apply them to the original picture, this means we need to do some adjustments.

```
// Resize To Image
x = (uint)originalImageWidth * x / OnnxModelScorer.ImageNetSettings.imageWidth;
y = (uint)originalImageHeight * y / OnnxModelScorer.ImageNetSettings.imageHeight;
width = (uint)originalImageWidth * width / OnnxModelScorer.ImageNetSettings.imageWidth;
height = (uint)originalImageHeight * height / OnnxModelScorer.ImageNetSettings.imageHeight;
```

Figure 11 Resizing bounding box (own)

Drawing on the image

With the bounding boxes set to the correct size, we can proceed to applying them to the picture.

The program uses the built-in Pen object, that takes an input in form of a color, and thickness to determine the style of the drawing.

The color we have saved in the Box object itself, and the thickness is hardcoded in our case.

By default, the Pen object will draw a straight line, but when the confidence level for the box is less than 0.5 (50%), the Pen pattern is set to change to create a Dashed line, to draw less attention to potential false positives.

With the following command, we draw the bounding box around our object.

```
// Draw bounding box on image
thumbnailGraphic.DrawRectangle(pen, x, y, width, height);
```

Figure 12 Drawing the bounding box onto picture (own)

4.5.2 Implementation

To implement the functionality with the rest of the algorithm, we will call the DrawBoundingBox function inside the Main function within the loop for all the images, with the required inputs.

```
IssueWarningForOverlap(overlaps_severity_level, imageFileName);
DrawBoundingBox(imagesFolder, outputFolder, imageFileName, detectedObjects);
}
```

Figure 13 Calling the DrawBoundingBox function (own)

Storing the result

We will store the final version of the image in the specified output folder with the following code.

```
if (!Directory.Exists(outputImageLocation))
{
    Directory.CreateDirectory(outputImageLocation);
}

image.Save(Path.Combine(outputImageLocation, imageName));
```

Figure 14 Saving the output image (own)

The result will look something like this.



Figure 15 Output photo (own)

4.6 Implementing the public safety features

Our program will have two main public safety features, and that would be determining

- If the amount of people present at the store is within the current safety limits
- If the people at the store are at a reasonable distance from each other

4.6.1 Determining current state of the shops capacities

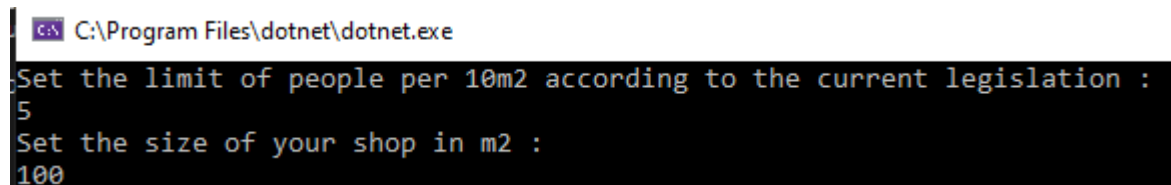
This part of the program keeps track of the amount of people present at the store, and determines if it satisfies the legal limit.

The allowed amount of people present

The first step to finding out if the store is within the legal limit, is getting the necessary information to determine the allowed capacity for the given store.

I created a function that asks the user for input to determine two values:

- The currently allowed amount of people per 10m²
- The store size



```
C:\Program Files\dotnet\dotnet.exe
Set the limit of people per 10m2 according to the current legislation :
5
Set the size of your shop in m2 :
100
```

Figure 16 Console asking for input (own)

With this information, we can simply calculate what the store's legal capacity is.

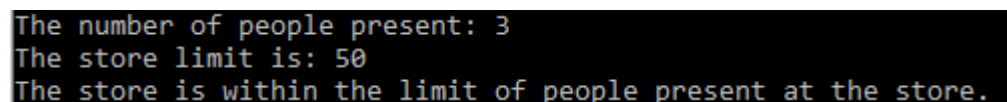
Amount of people present

This step is implemented after we run the model to score our input data (the pictures from the store cameras), extract the relevant data by using the YoloOutputParser, and the FilerBoundingBoxes function.

We count the number of bounding boxes, and that should be the number of people present at the store.

Result

The program then compares the number of people present to the allowed amount of people, and lets the user know what the current situation at the store is.



```
The number of people present: 3
The store limit is: 50
The store is within the limit of people present at the store.
```

Figure 17 Console output - shop capacities (own)

4.6.2 Warning system for possible neglect of social distancing

This part of the program is designed to warn the shop owners for possibly overcrowded spaces in their store.

Detecting close proximity

When people are close to each other, their bounding boxes are likely to overlap, so I decided to use this as a way to warn the shop owner of the possible health hazards in their shop.

To do so, we take all the bounding boxes for a given picture, and compare every box to all the other boxes.

```
if (box_to_compare == box)
{
    continue;
}
if (box_rect.Intersects(box_to_compare_rect))
{
    box_rect.Intersect(box_to_compare_rect);
    if (!box_rect.IsEmpty)
    {
        number_of_overlaps++;
    }
}
```

Figure 18 "if" statements to determine box overlap (own)

The program counts the amount of overlaps, which we then divide by 2 to get the actual number of overlaps, since each intersection is counted from the perspective of the other bounding box as well.

Determining severity

To determine if the situation is significant enough to issue a warning, I've implemented a level system

- Level 0 - Less than 2 overlaps, results in no warning
- Level 1 - 2 or more overlaps
- Level 2 - Can be reached if Level 1 is satisfied, and one of the following is true:
 - The intersections per person ratio have to be greater than 0.7

- The number of people is greater than the limit of people divided by the number of cameras.
- Level 3 - Level 1 has to be satisfied, and both conditions in Level 2 have to be satisfied.

All the levels will trigger a warning with the camera/picture name attached, except Level 0.

```
Level 1 severity social distancing warning: isle_nr_1.jpg  
Level 2 severity social distancing warning: isle_nr_10.jpg  
Level 3 severity social distancing warning: isle_nr_11.jpg  
Level 2 severity social distancing warning: isle_nr_12.jpg
```

Figure 19 Console social distancing warnings (own)

4.7 Performance

With the software complete, we can run the program, providing it with sample images to determine the accuracy of the classification and the precision of the bounding box, as well as if the other features work as intended.

4.7.1 Testing Classification Accuracy of the model

To test the classification accuracy, we will compare pictures with a total amount of 40 people. And because a big part of the current restrictions focuses on wearing masks in indoor places, I have decided to compare two different samples.

Half of the people will be wearing masks, and the other half will be without masks.

Results

Without mask - Out of the 20 people without masks, the program has detected 16, which gives us a success rate of 80%.

Mask - For the 20 people with masks, results were the following - 16 were recognized, leaving us with a 80% success rate.

The test, although the sample size was small, did not show any difference or indication, that wearing a mask would negatively impact the recognition accuracy.

I also detected 11 false positives or very inaccurate bounding boxes, in total. The model is therefore not especially accurate, but for the purposes of this thesis and proof of concept, it is sufficient.

Here is an example of such a situation:

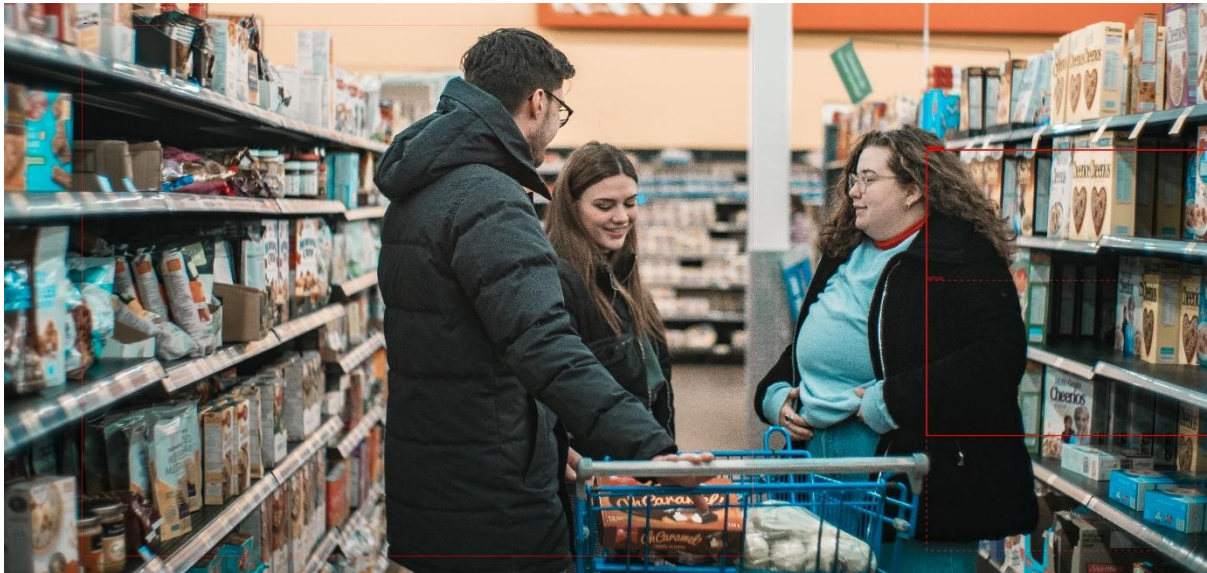


Figure 20 Failed recognition (own)

4.7.2 Overall functionality

A test run of the program shows that all the implementations work, and the program does not crash at any point in the process. The warning messages and the final function call report works, and the pictures with bounding boxes are being saved into the supposed file.

Here is a picture of the console after a test run.

```
Set the limit of people per 10m2 according to the current legislation :
5
Set the size of your shop in m2 :
100
Warning: ImageScaler was a removed experimental ops. In the future, we may
your model as soon as possible.
Level 2 severity social distancing warning: isle_nr_1.jpg
Level 1 severity social distancing warning: isle_nr_10.jpg
Level 1 severity social distancing warning: isle_nr_11.jpg
Level 2 severity social distancing warning: isle_nr_12.jpg
Level 2 severity social distancing warning: isle_nr_13.jpg
=====
The number of people present: 40
The store limit is: 50
The store is within the limit of people present at the store.
=====
```

Figure 21 Test run – Console (own)

We can see that we have set the limit of people per 10m2 to 5, and the size of the shop to 100.

The program has correctly informed us at the end of the function run, that the store limit is 50, and that the store is within the limit of people allowed to be there.

We have also received a couple warnings, which indicate that on these pictures, there was a higher concentration of people in one place.

5 Result

The result of this bachelor thesis is a working program intended and designed to demonstrate what a possible object detection use case could look like in a retail store setting. The software can take input from a user to determine the store's capacity, and detect people in pictures from said store and compare the result to the pre-set requirements, as well as detect possible overcrowded areas of the store, issue a warning, and save pictures with the detected people highlighted.

The source code to this software is attached to the thesis. It consists of a folder that stores the pre-trained model and the pictures for the input, as well as the pictures from the output.

The main "Program.cs" file is used to run the function and is responsible for most of the main functionality. Other important functions are stored in the YoloParser Folder. The most important functions however are specifically in the "yoloOutputParser.cs" file.

For the development of this software, we mainly worked with the C# programming language to make use of the output from the tinyYOLOv2, which has been made possible by the very helpful ML.NET libraries and the ONNX.

While building the features for this software, no major issues have occurred, however the accuracy of the pretrained model is not the best, and when two similar objects are close to each other, the way we filter the boxes can very easily make the program delete a box that could be intended for the second object. But the core functionalities can be applied to any other, preferably better model that can detect humans in pictures.

6 Conclusion

This thesis was meant to describe the technologies related to ONNX, YOLO and ML.NET, as well as creating a program that is using object detection, and describe the key stages of development. The literature review describes the underlying and related technologies, as well as presents some existing issues and hurdles. As the last part it provides a look at the history of technologies used in the retail business, as that is the area that the practical part is designed for. For the practical part the thesis focuses on the implementation and use of a pre trained YOLO model, and the use of its capabilities for different safety features inspired by and intended for the current situation with the pandemic. Each feature is described and the intended purpose explained. The last part of the practical part focuses on testing the accuracy of the pre trained model.

The use of machine learning and object detection software has become a lot easier over time, with the introduction of multiple libraries and technologies that have simplified the development of such software. This means that today we are able to implement such technologies with relative ease, and that there is a lot of documentation freely available, so we can expect that we will be seeing such technologies in many more places than now.

7 References

National Highway Traffic Safety Administration: *TRAFFIC SAFETY*

FACTS [online]. 2015 [cit. 2021-03-3]. Dostupné z:

<https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115>.

Open Neural Network Exchange: *The Linux foundation* [online]. 2019 [cit. 2021-02-17]. Dostupné z: <https://onnx.ai/>

GHARAGYOZIAN, Hayk. *MAcadamian: A Practical Application of Machine Learning In Medicine* [online]. 2019 [cit. 2021-02-1]. Dostupné z:

<https://www.macadamian.com/learn/a-practical-application-of-machine-learning-in-medicine/>

Microsoft Docs Tutorial: *Detect objects using ONNX in ML.NET* [online]. 2020 [cit. 2020-10-1]. Dostupné z: <https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/object-detection-onnx>

Missinglink: *Perceptron input and output* [online]. [cit. 2021-2-10]. Dostupné z:

<https://missinglink.ai/guides/neural-network-concepts/perceptrons-and-multi-layer-perceptrons-the-artificial-neuron-at-the-core-of-deep-learning/>

Unsplash: *Man in store with camera* [online]. [cit. 2021-3-10]. Dostupné z:

<https://unsplash.com/s/photos/>

SimpleLearn: *Detailed perceptron* [online]. [cit. 2021-3-1]. Dostupné z:

<https://www.simplilearn.com/what-is-perceptron-tutorial>

Habr: *Convolution process + Relu + Sub-Sampling* [online]. 2018 [cit. 2021-2-11].

Dostupné z: <https://habr.com/en/post/348000/>

CAMACHO, Cezanne. *Github: CNN representation* [online]. 2018 [cit. 2021-2-23].

Dostupné z: https://cezanne.github.io/Convolutional_Neural_Networks/

Geeks for geeks: *Max pooling* [online]. 2019 [cit. 2021-2-23]. Dostupné z:

<https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/>

RUGERY, Pierric. *BecomingHuman: YOLO CNN Representation* [online]. 2020 [cit. 2021-2-28]. Dostupné z: <https://becominghuman.ai/explaining-yolov4-a-one-stage-detector-cdac0826cbd7>

8 Appendix

Files attached to the thesis:

A Folder named App that includes -

- assets – a folder with the pre trained model and the folder for input and output (images)
- DataStructures – folder with the code to specify the location of the input files (images)
- YoloParser – A folder with files for the Yolo parser scripts and bounding box filter.
- OnnxModelScorer.cs – script that handles some model settings and transformation of data.
- Program.cs – main script of the program with functions for drawing the bounding box, the security features and other helping functions
- Bin, boj, ObjectDetection.csproj are files and folders created by Visual Studio.