Object Detection using YOLOv3

Bachelor Thesis

Written by

Sana Jamal Agha

at

Fachbereich Ingenieur- und Naturwissenschaften
Fachbereich Angewandte Informatik

Gutachter: Prof.Dr. Michael Schenke

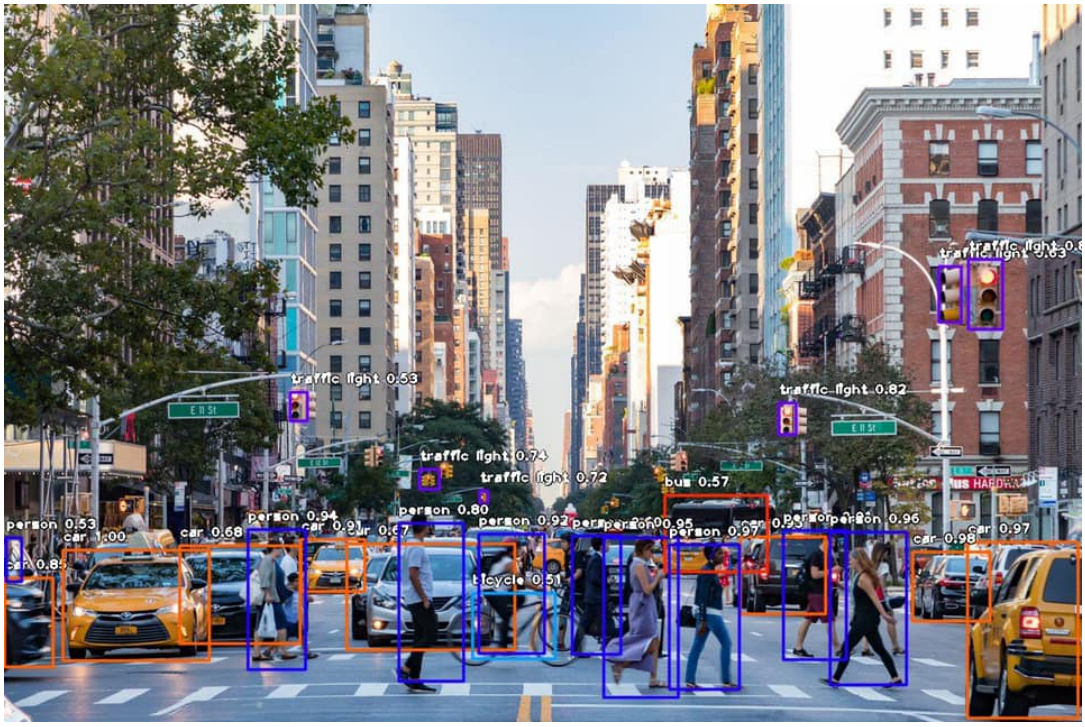
Gutachter: Prof.Dr. Karsten Hartmann

1. Introduction

In the field of computer vision, artificial neural networks for solving tasks have recently become more and more popular. This technology has established itself particularly well in the field of autonomous driving, for example when recognizing street signs. Apart from that, there are other areas of application. This thesis describes how to use deep learning to detect objects in images.

1.1. Goal

The aim of this work is to provide an overview of artificial neural networks and methods for object recognition within an image. Many methods will be thoroughly explained to gain perspective about the best approach to implement the image object detection system. Transfer learning should be used to keep track of the number of images required to train a small network. The results obtained can thus be compared and discussed. Figure(1) shows what the result should look like.



Figure(1) Multiple objects detected in an image

1.2. Thesis structure

In the introductory chapter, the background and the objective of this work are briefly explained. The rest of the work is structured as follows:
Section 2 gives an overview of the theory behind neural networks and object recognition. In addition to a brief description of the most popular deep learning frameworks. Furthermore, section 3 gives an idea of the implemented project and the used methods. In addition to explaining the steps of the implementation

.

2. Basic Fundamentals

This chapter explains the basic theory on which the work is based. This includes a general introduction to artificial neural networks. In Addition to the structure and principle of training. Furthermore, performance problems and limitations are explained as well as the application of methods for

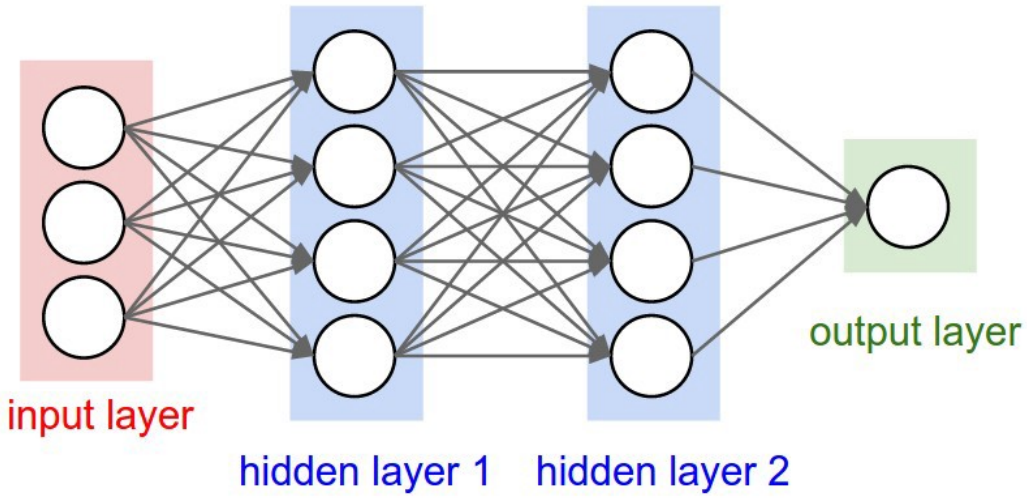
regularization in order to improve the performance of the network. After that, convolutional neural networks (CNN) and its methods for recognizing objects in an image are described. At the end of the chapter, the most popular deep learning frameworks are briefly explained and GPU support is discussed

2.1. Artificial Neural Networks

An artificial neural network is a computer-generated model based on a simplified principle of the brain. Like the human brain, an artificial neural network improves over time. Training data is required for this learning process [1].

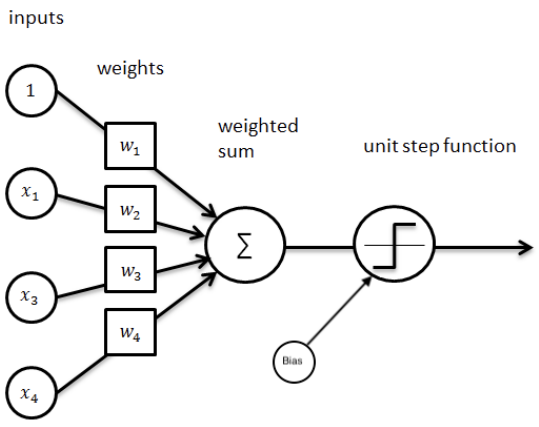
2.1.1. Neurons and network structure

An artificial neural network consists of innumerable neurons that are assigned to different layers within this network. These layers are basically divided into three categories: an input layer, several so-called hidden layers and an output layer. A simple artificial neural network is shown in Figure(2). The network is a three Layer network, the first layer aka. Input layer is not counted in deep learning.



Figure(2) Three Layer Neural Network

The input layer consists of multiple nodes which represents the number of the features in the dataset.
The output layer has a number of nodes which define how many different categories the network can recognize.
As for the layers in between the input and output, they are called the hidden layers and in this layer is where all the work happens. These layers can have different setups in depth and width according to how complicated or simple the dataset is. Also Deeper networks do give better results, but the deeper the network is the higher its computational time is.
The nodes in the network are called Neurons. Each of these nodes represents a value or a feature. The values of the nodes in the input layer correspond to the information that is fed to the network. In the hidden layers, these nodes are linked to one or more nodes from the previous and the following layer by weighted connections. These weightings change during the network learning process. The value that a neuron receives from a connection to a neuron in the previous layer(Input layer). Based on all inputs of a neuron, the value of this node is calculated in the hidden layer. This is usually calculated from the sum of all inputs. The value can be supplemented by a so-called bias (section 2.1.3) or adjusted using an activation function (section 2.1.2). This calculation is shown graphically in Figure(3).



Figure(3) Node from the hidden neural Network Calculation

All The values of the neurons in the Input layer are connected with all the nodes of the first hidden layer. All of the connections have a representative impact “w” in other words they are the weightings of the connection. The input is assigned to the category whose neuron has the highest value [3]. Now to calculate the sum of one node of the first hidden layer we will multiply each one of the input layer node’s value with the corresponding weight and get the weighted sum. Each node on the hidden layer has an Activation function (section 2.1.2) which will determine based on the summarized value if the node is activated or the level of “activity” it shall have. Also Bias value could be added in a positive or negative value to adjust the shift of the activation function.

2.1.2. Activation functions

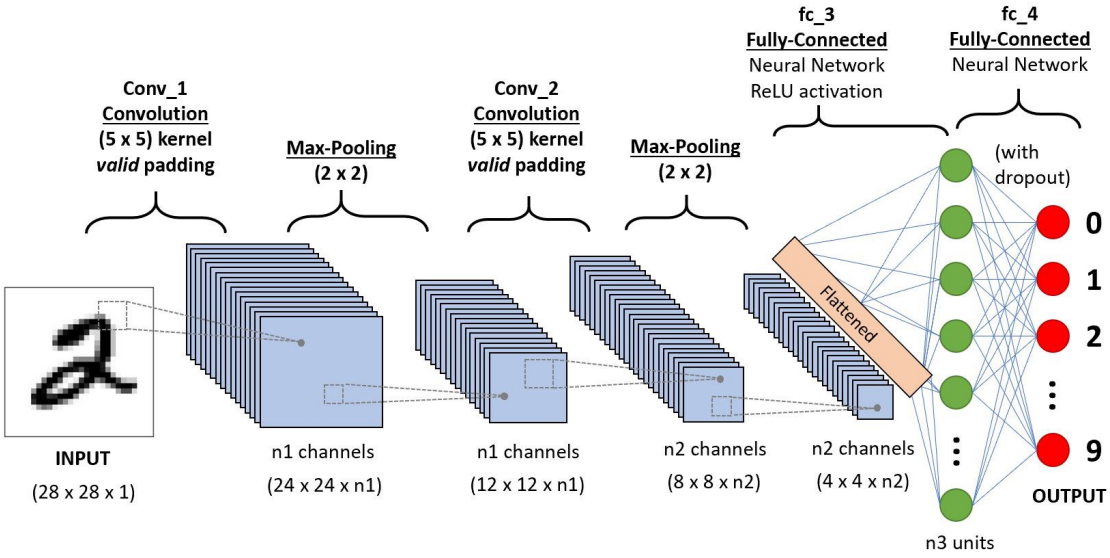
Activation functions are functions to adapt the value of a neuron or to keep it within a certain range of values. These are used to determine how big the influence of the input is on the output. In this way, for example, it can also be avoided that certain neurons achieve a value that is much higher than others and thus have too great an influence on the calculation of values in later layers. In principle, activation functions can be divided into linear and non-linear activation functions. Many different functions can be used for this [3].

2.1.3. Bias

A bias is an optional value that can be added after the calculated value from the inputs when using the activation function. It can be in the negative as well as in the positive value range and can therefore be used to shift the activation function to the right or left. This means that the bias indicates a threshold value from which a neuron is fired depending on the activation function. This value can also change during the learning process [49]

2.2. Image classification and CNN

A special type of artificial neural network is used or required to classify images. This type of neural network is called a convolutional neural network “CNN” and can analyze features in images. This network differs from conventional neural networks mainly through special layers. These layers include convolutional layers (section 2.2.1), fully connected layers (section 2.2.4) and pooling layers (section 2.2.3). The structure of convolutional neural networks is somewhat similar to the general Artificial Neural Network structure, it consists of an input layer, followed by a combination of several convolutional and pooling layers, then one or more fully connected layers and finally an output layer. This output layer is again responsible for the categorization of the input image. A structure of such a CNN is illustrated in Figure(4). Compared to a conventional neural network, a feature map can be seen with the output of a neuron and a kernel of a convolutional layer with a weighted connection. This type of network is invariant to a certain extent with regard to the scaling and translation of features [4].
Figure(4) is a General presentation of a convolutional neural network and how it works. The convolution layers at the beginning of the network are necessary to recognize low-level features such as corners and edges. High-level features at the end of the network are responsible for a precise understanding of the image. The fully connected layers at the end are responsible for classifying the image.

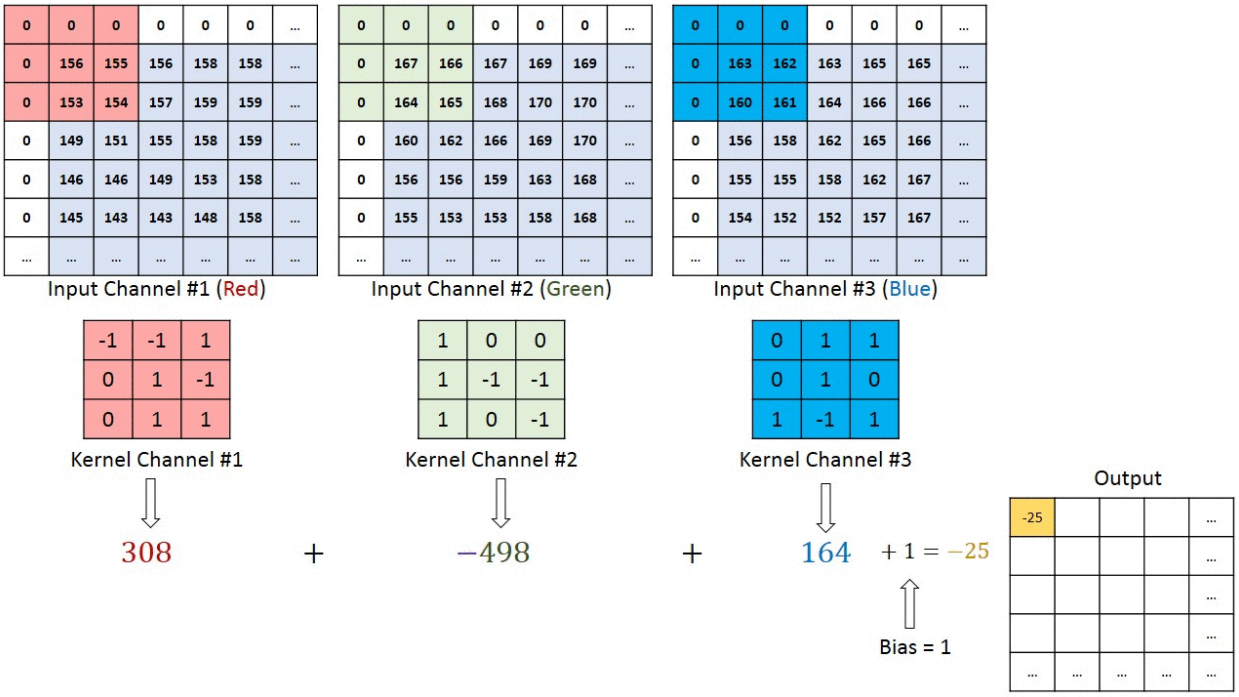


Figure(4) Convolutional Neural Network [5]

2.2.1. Convolutional Layer

A convolutional layer consists of a set of filters, also known as a kernel. In the simplest case, such a filter is a two-dimensional matrix of weightings, provided that each two-dimensional activation map of the input is seen as an independent input. The calculations in a convolutional layer are carried out by moving a kernel over the entire two-dimensional matrix for each input and calculating a value for the output activation map for each position. The horizontal and vertical shift of the filter is called the stride. The output is basically calculated in the same way as with a conventional neuron. For this purpose, the inputs weighted by the filter are totaled and, if necessary, a bias (section 2.1.3) is added. This calculation is shown graphically in Figure(5). An activation function (section 2.1.2) can then be used. Using a kernel means that, depending on the size of the filter and the shift, information is lost at the edge of the input and the output matrix is smaller than the input matrix. Under certain circumstances this can be avoided by padding (section 2.2.2). A convolutional layer can also have several activation maps as output [4].

Figure(5) is a Graphical representation of the functional principle of a convolutional Layer. A new feature map is created by adding up the input areas several times after applying the kernel (filter).

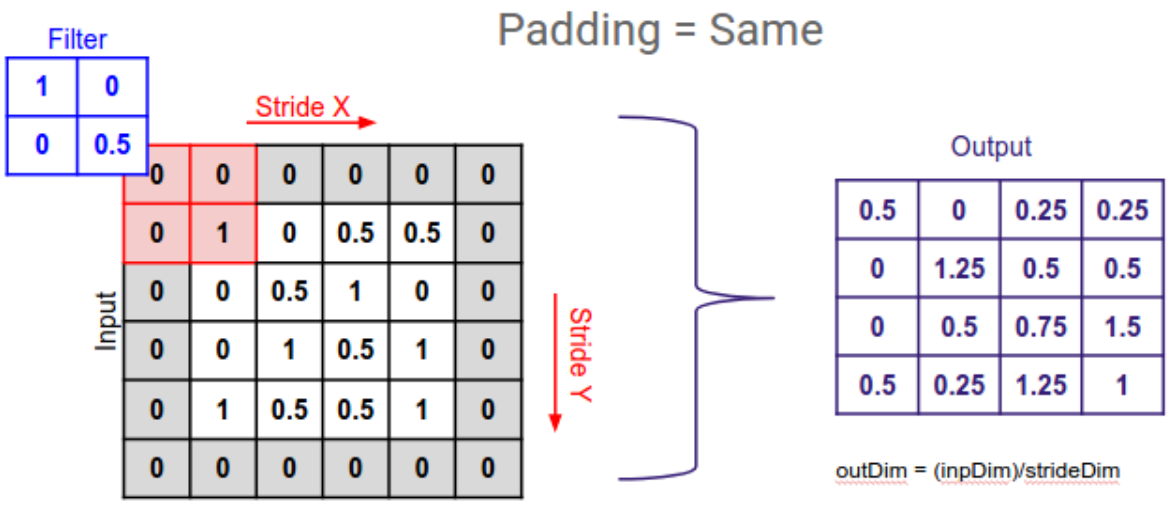


Figure(5) Graphical representation of the functional principle of a convolutional Layer [5].

2.2.2. Padding

Padding is used to avoid loss of information at the edge of the input. For this purpose, the activation map of the input is enlarged by adding additional information to the margin. As a result, a kernel is now also applied centrally to the originally outermost positions of the input. There are various methods that can be used to expand the input information. A widely used method is to expand the input matrix with zero everywhere. This is known as zero padding and can be seen in Figure(6) [6].

Figure(6) is a Graphic representation of the functional principle of zero padding. By adding zeros to the edge of the source map, the feature map created will keep the same size after the kernel is applied.

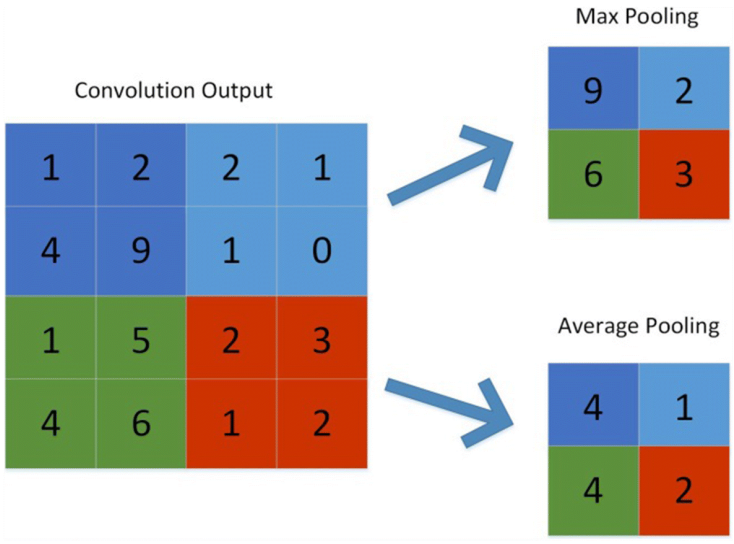


Figure(6) Graphic representation of the functional principle of zero padding [5][6]

2.2.3. Pooling Layer

Pooling layers are used to reduce the size of the activation maps. This should be seen as positive, as it saves computing effort and enables deeper networks. Relatively little information is lost and the advantage is that the network reacts less sensitively to different positions of features. Another advantage is that kernels that are applied to already reduced activation maps cover a larger area of the original image without increasing the kernel and thus the number of parameters. The most common pooling strategies are max pooling and average pooling. Basically, pooling works in a similar way to a convolutional filter. However, no matrix calculation is carried out here, only the largest or the average value from the input range selected by the filter is passed on as output. Figure(7) illustrates pooling using max pooling [4].

Figure(7) is a Graphic representation of the functional principle of Average and Max-Pooling. In Average-Pooling: the average number of each color-coded area is calculated and transferred to the next feature map, where the Max-Pooling: selects the highest number for each color-coded area and transfers it to the next feature map.



Figure(7) Graphic representation of the functional principle of Max-Pooling [9]

2.2.4. Fully Connected Layer

A fully connected layer is a layer in a CNN that does not consist of filters, it consists of neurons that store a single value. This type of layer is located at the end of a CNN. When changing from an activation map to a fully connected layer, each value of the activation map is connected to a neuron in the subsequent fully connected layer. These neurons are connected to neurons from the next layer via weights. The number of neurons in the output layer corresponds to the number of possible image categories [8].

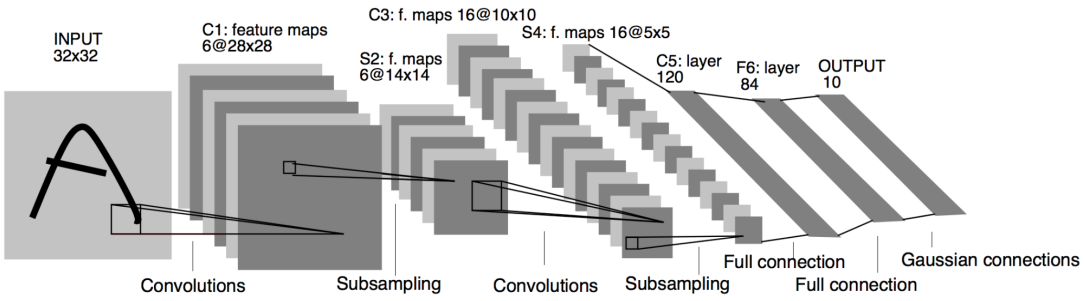
2.2.5. CNN models

A wide variety of convolutional neural networks for classifying images have proven to be the most accurate over the past few years. They differ from one another in the number and arrangement of certain layers and their parameters. Most of these networks participated in and won the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC). ILSVRC is a renowned competition in the field of computer vision. For training, these networks used the ImageNet database, which contains around 14 million images. Success is indicated in the competition with the top 5 test error rate, which allows the conclusion to be drawn about the relative proportion of images to be classified that were not correctly classified by the network under the top 5 most likely categories. The most popular CNNs are briefly explained below [4].

2.2.5.1. LeNet

LeNet is a model developed by LeCun in 1998 to recognize handwritten digits [10]. It consists of seven layers. The seven layers are made up of three convolutional layers, two average pooling layers and two fully connected layers. The input for this model was limited to 32 × 32 pixel gray-scale images. This architecture is shown in Figure(8).

Figure(8) is a Graphic representation of the architecture of the LeNet-5 model.

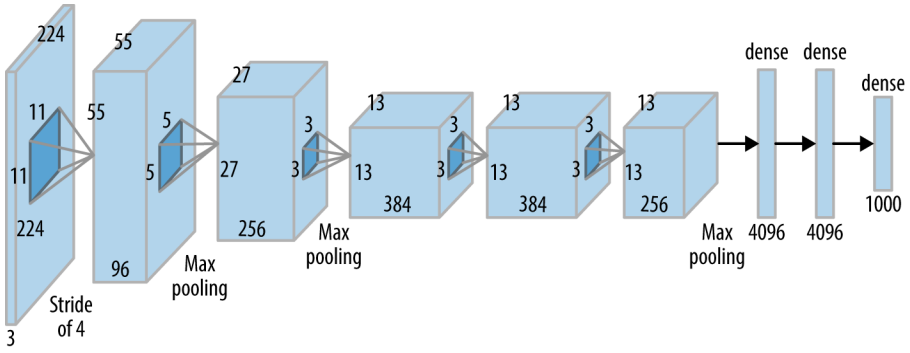


Figure(8) Graphic representation of the architecture of the LeNet-5 model.

2.2.5.2. AlexNet

AlexNet was developed by Krizhevsky, Sutskever and Hinton in 2012 [11] and was the first major breakthrough for neural networks in the field of computer vision. This network model won the ILSVRC in 2012 with a top 5 test error rate of 15.4%. Compared to modern networks, the network consists of a relatively simple architecture with only eight layers. It consists of five convolutional layers, max pooling layers and dropout layers as well as three fully connected layers. The filters in the convolutional layers have a size of 11 × 11, 5 × 5 or 3 × 3. ReLu was used as the activation function (Section 2.1.2). The network was trained to distinguish between 1000 different image categories. This model is shown in Figure(10)

Figure(9) is a Graphical representation of the architecture of the AlexNet [12]

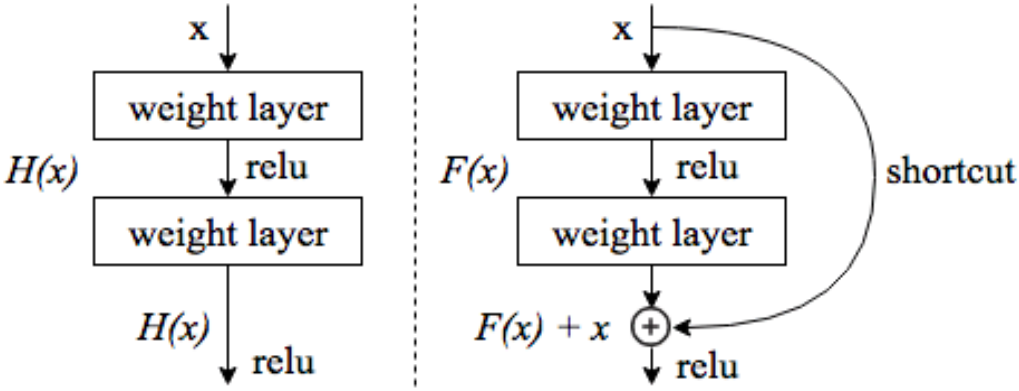


Figure(9) Graphical representation of the architecture of the AlexNet

2.2.5.3. ResNet

ResNet was developed in 2015 by Microsoft Research Asia [38] and consists of a total of 152 layers. The special feature of the model is that inputs from earlier layers are routed backwards via a kind of bypass in the network and thus combined with the output of a later layer. This is shown in Figure(13). This network architecture won first place at the ILSVRC 2015 with a top 5 test error rate of 3.6%.

Figure(10) gives a clear view how the ResNet network provides a shortcut connection

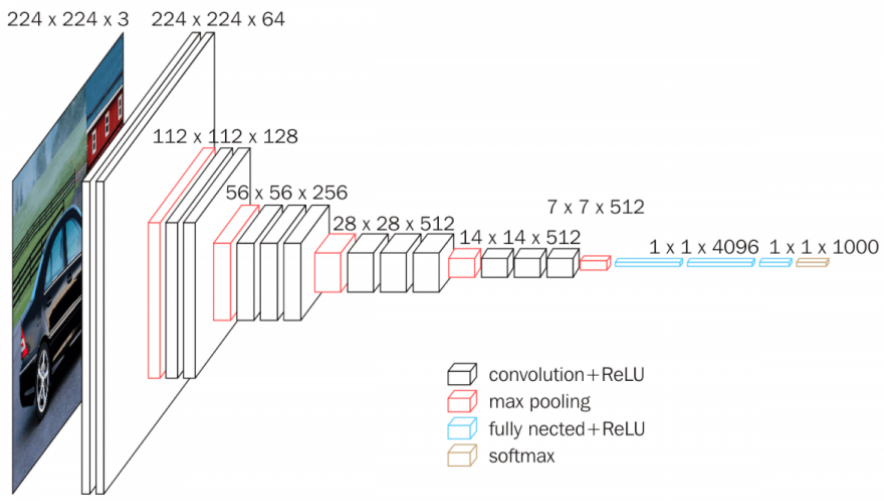


Figure(10) Standard CNN on the left vs ResNet on the right

2.2.5.4. VGGNet

The VGGNet architecture was developed by Simonyan and Zisserman from the University of Oxford in 2014 [13] and consists of 19 layers. It achieved a top 5 test error rate of 7.3% at the ILSVRC. The layers are made up of convolutional layers, max pooling layers and fully connected layers. The filters in the convolutional layers have a uniform size of 3 × 3. By arranging two or more layers with small filters one behind the other, a layer with a larger filter can be reproduced. This has the advantage that the number of parameters is reduced. As a result of the pooling layer, the size of the output activation map is reduced by half in each dimension. However, the number of filters in the subsequent convolutional layer is increased, which leads to a larger number of activation maps.

Figure(11) is a Graphical representation of the hoe th VGGNet architecture is built (14)

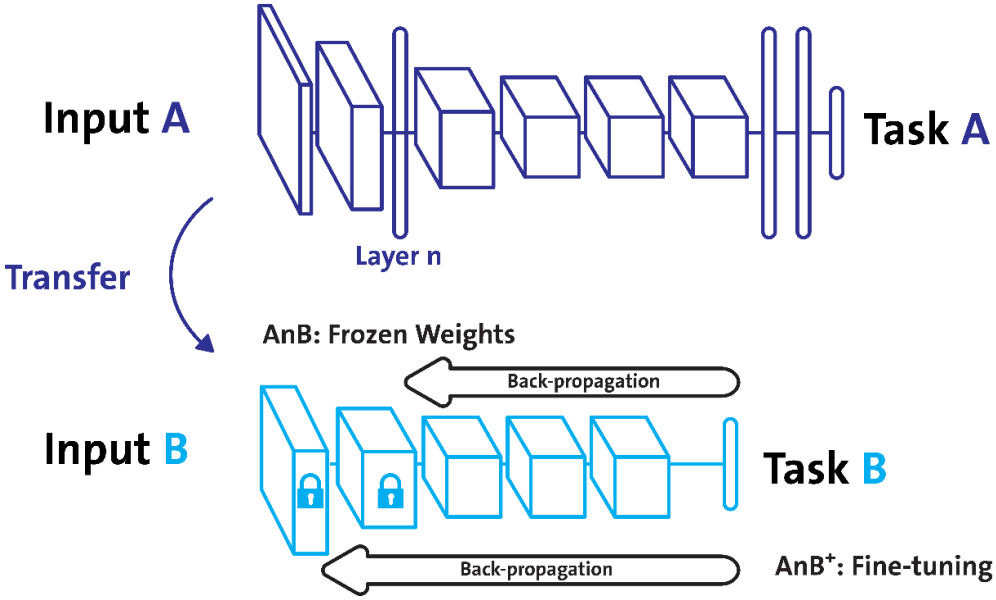


Figure(11) VGGNet Architecture (14)

2.3. Transfer Learning

The data for training a neural network is a very important part of obtaining a useful result. Often the amount of data that the network needs is not large enough. In such a case, transfer learning is used. With transfer learning, the network is trained with

another extensive data set. This model, including the learned parameters and weightings, is used as the basis for the learning process with your own data. To do this, the last layer in the network is removed and replaced by a separate classification layer.

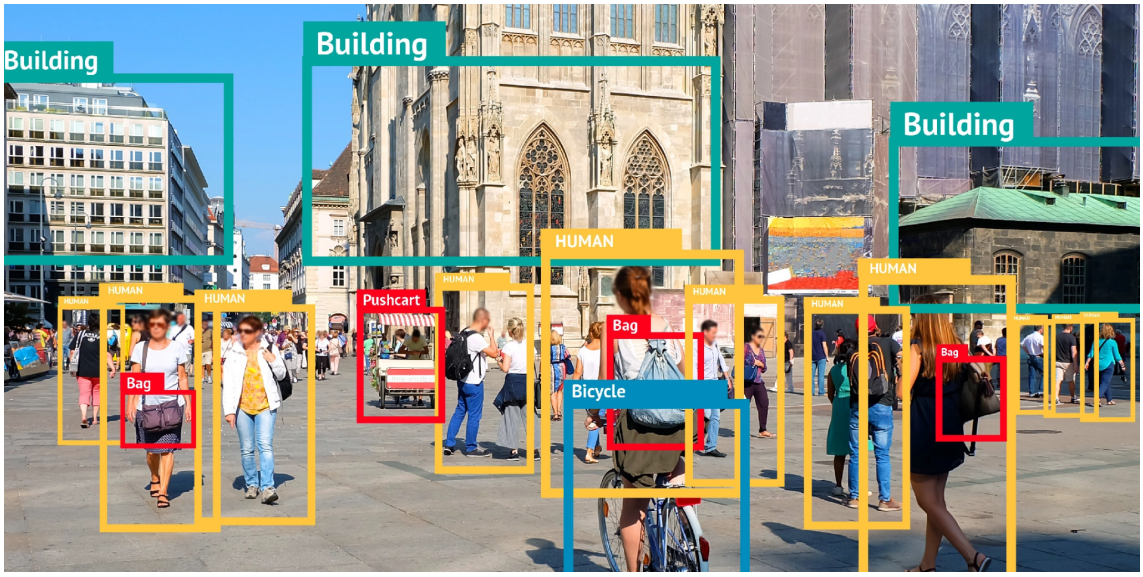


Figure(12): Transfer learning network structure

The weightings in the convolutional layers are partially frozen. This means that their weightings can no longer change when the learning process is repeated. This process is also known as fine-tuning. Since the early layers in the network are responsible for recognizing general features such as edges and corners, these learned features can be used unchanged for your own data set and do not need to be trained again. The further back the layers are located in the network, the more specific the features that are extracted from the layer become. Depending on how much your own data set differs from what was used for basic training, more or less convolutional layers can be frozen. The layers that are responsible for the classification must be retrained [15].

2.4. Object Detection

The detection of objects in an image does not correspond to the classification of an image itself. The difference between object recognition and classification is that the position of the object in the image in object recognition algorithms is also analyzed. This is often shown by means of a bounding box and can be seen in Figure(13).



Figure(13) Recognition and localization of several recognized objects in an image by means of bounding boxes and classification[44]

Furthermore, several objects should be able to be found within an image by means of object recognition. This cannot be solved directly with a CNN, since the number of objects within an image cannot be determined in advance and the output vector therefore has a variable length [50]. An initial, bona fide attempt to solve this problem was here to define different regions of interest in the image and to apply a CNN to each of these regions for classification. The problem with this experiment is that objects can be in different places in the picture, they have a different aspect ratio or can be of different sizes. Therefore, a large number of regions had to be defined in order to cover all possibilities. To solve this problem, methods such as Region-based Convolutional Neural Networks (R-CNNs) or You Only Look Once (YOLO) have been developed [43].

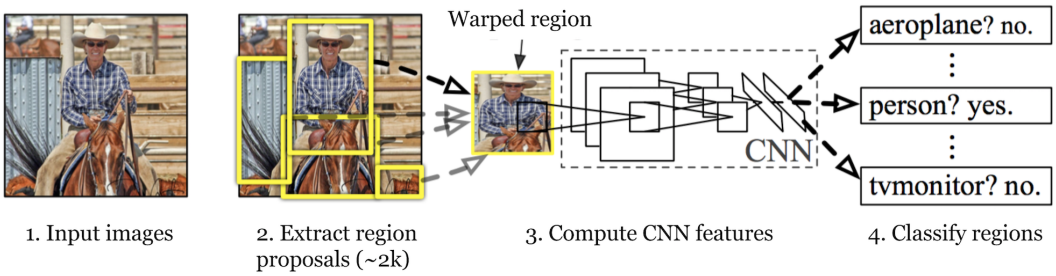
2.4.1. Region-based Convolutional Neural Network

One of the best-known methods for recognizing objects within an image is called a region-based convolutional neural network. Over time, different approaches to solutions have been established, which are described below

2.4.1.1. R-CNN

To solve the problem of the large number of regions required, [16] describes a method that uses a selective search algorithm [52] so that the number of regions of interest can be reduced to

2000. A selective search algorithm works on the principle of performing a sub-segmentation at the beginning, which provides a large number of potential region candidates. A greedy algorithm is then applied to recursively merge similar regions into larger areas. The combined regions are ultimately used to determine the final regions of interest. How a selective search algorithm for object recognition works can be read in detail under [20]. These 2000 regions are transformed to a uniform, square size and analyzed with a CNN which produces a 4096 dimensional feature vector as output. The features extracted from the CNN are fed into a Support Vector Machine (SVM) [51] in order to classify a potential object in one of these regions. In addition to the classification of the object, the position and size of the object in the image is also forecast. This is shown graphically in Figure(14).



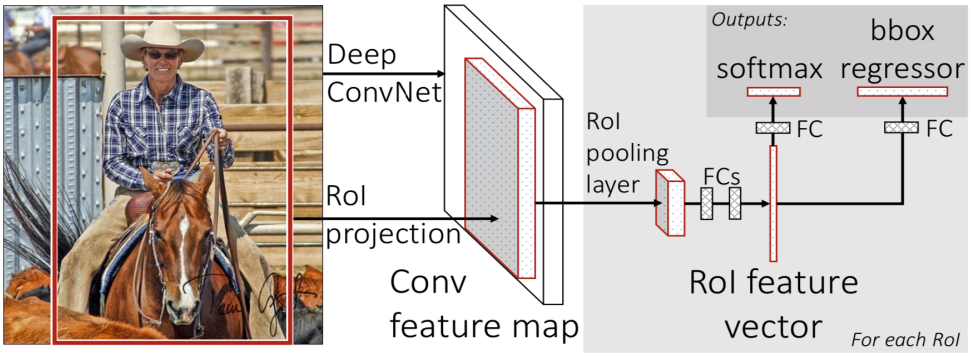
Figure(14) Famous example selective search, extracting Region proposal and applying classification

However, the solution method still has some immature approaches. It takes a very long time to train the network because 2000 different regions have to be classified per image. As a result, the time required to analyze a test image is correspondingly long. Furthermore, the use of a selective search algorithm to determine the regions of interest is not an optimal solution, since this is a permanently defined algorithm and

therefore cannot improve over time, which leads to poor region candidates.

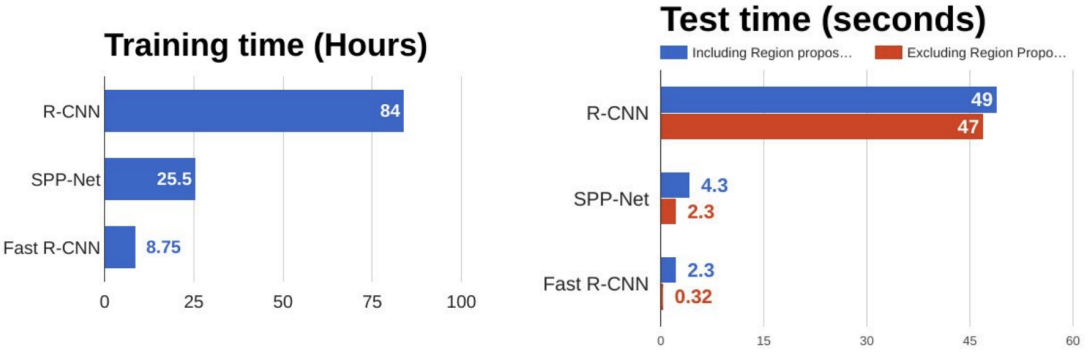
2.4.1.2. Fast R-CNN

The same author at R-CNN was able to solve some of these issues and was able to come up with a faster approach. He called this approach Fast R-CNN [18].



Figure(15) Applying CNN and using ROI pooling layer to convert

Basically, the approach is pretty similar. Instead of analyzing each individual region using a CNN, this method uses a CNN to analyze the entire image, which saves an enormous amount of time. After applying CNN to analyse the entire image ROI pooling layer is used to convert the feature maps of ROI obtained from this, the regions of interest are identified with a selective search algorithm and transformed into squares with a fixed size. In this way, they can be converted into a fully connected layer. Finally, a softmax layer is applied to classify the regions. Furthermore, the position and size of the object in the image is determined with a BoundingBoxRegressor. This architecture is shown in Figure(15)



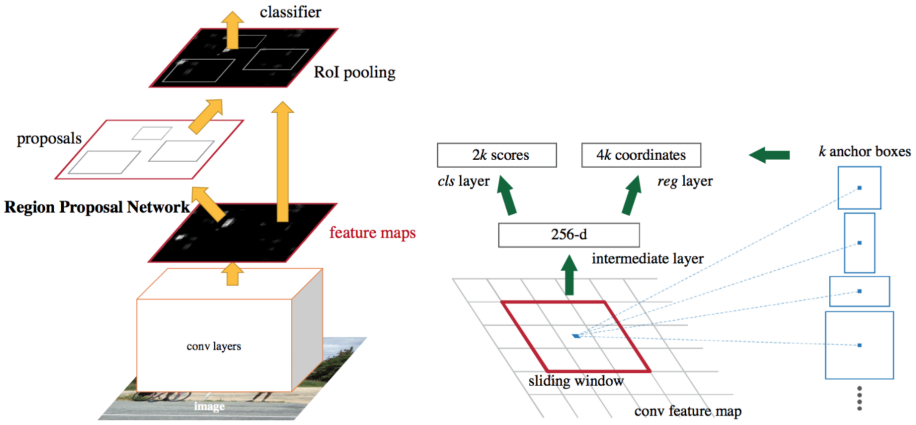
Figure(16) R-CNN vs Fast R-CNN performance

In Figure(16) you can see that Fast R-CNN is significantly faster than R-CNN. The values here were taken from [19] and are intended to indicate the relative time difference, since the absolute time required always depends on the type of task and the system used. With the time required for testing, it can be seen that a large part of the computation time of the algorithm is required for the region prediction. This means that the prediction of the regions of interest by the selective search algorithm is the weak point of the Fast R-CNN algorithm in terms of performance and that there is still room for improvement.

2.4.1.3. Faster R-CNN

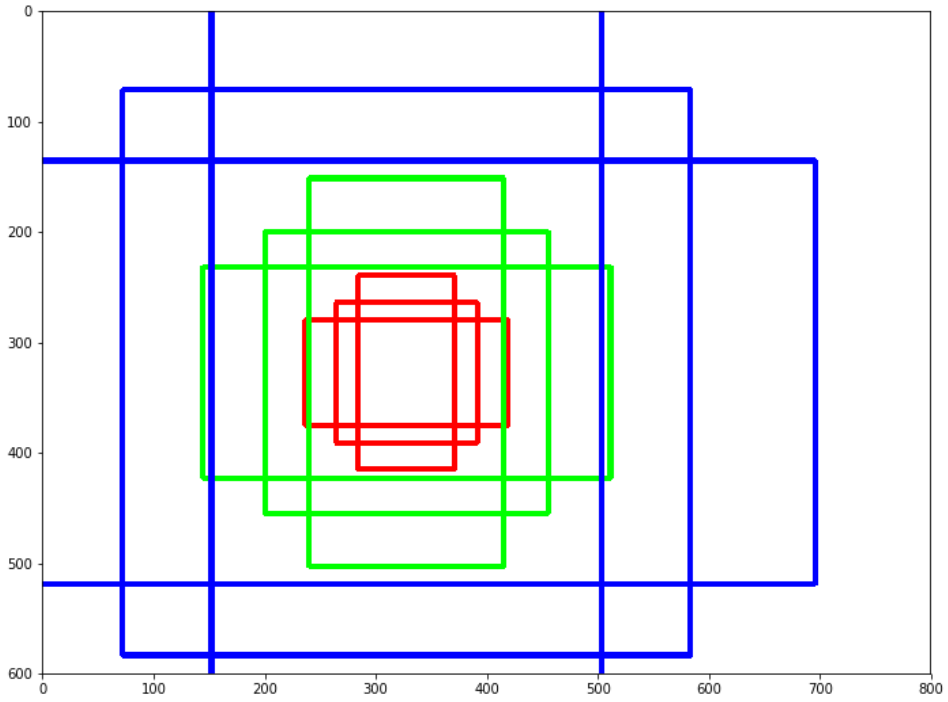
The performance problem in determining the regions of interest is solved with Faster R-CNN [21]. As with Fast R-CNN, the image as a whole is analyzed using a CNN. In order to suggest regions, a selective search algorithm is not used, as in the methods described above. Instead, an object recognition algorithm is introduced in which the network independently identifies the suggested regions through learning. A separate, so-called Region Proposal Network is used for this purpose. This brings an enormous time advantage and has the consequence that the entire process takes about the same amount of time as Fast

R-CNN without specifying the region proposals.



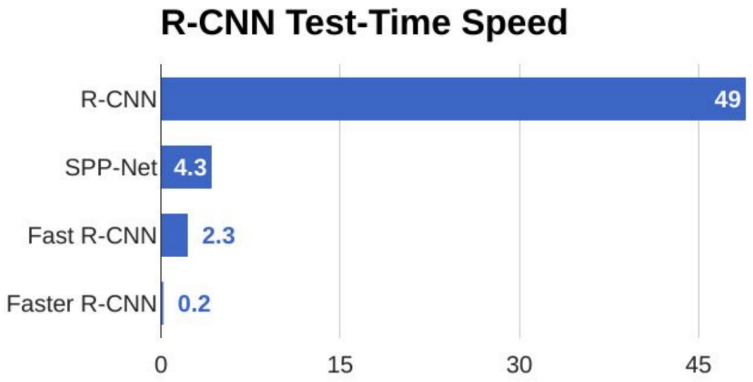
Figure(17) Faster R-CNN Model and anchor boxes

The predicted, suggested regions are then reshaped with a RoI pooling layer to classify the image within the suggested region and predict the offset values for the bounding boxes. The most important idea in Faster R-CNN is the idea of anchor boxes which will be mentioned later in the paper again (section 3.2.1.3). Anchor boxes provide a predefined set of bounding boxes which consists of width and height and predict bounding box that are relevant to the bounding box instead of predicting the bounding box which is relevant to the image , these bounding boxes may have different size and ratios as it is shown in Figure(18) below. In summary anchor boxes allows YOLO to predict multiple bounding boxes per grid instead of having just one bounding box



Figure(18) Different sizes and ratios of Anchor boxes

The architecture of Faster R-CNN and a demonstration of the anchor boxes are shown graphically in Figure(17).



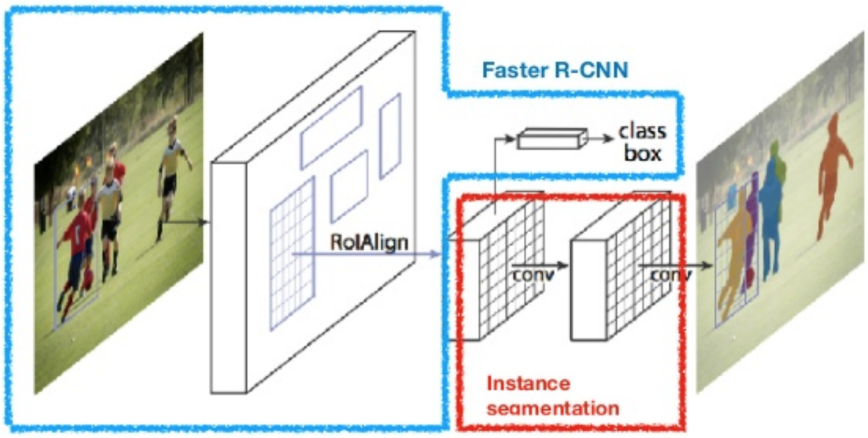
Figure(19) Fast**er** R-CNN Test Time Speed

In Figure(19) A test speed comparison between R-CNN, Fast R-CNN and Fast**er** R-CNN including suggestions for regions. Faster R-CNN was able to dramatically reduce the time it takes

to propose regions compared to Fast R-CNN. The values here were taken also from [19.]

2.4.1.4. Mask R-CNN

Mask R-CNN [22] describes a method that functionally extends Faster R-CNN. In addition to the existing branches, which determine the classification as well as the position and size of the object, another branch is added here. This branch is responsible for predicting segmentation masks for the individual regions of interest. For this purpose, a segmentation mask is predicted at the pixel level by means of a fully connected network. The additional calculation only leads to a minimally greater computational effort.



Figure(20) Mask R-CNN architecture[41]

Since Faster R-CNN is basically not designed to analyze at the pixel level, which is most evident in the pooling operations, a quantization-free layer that persists the exact position data was introduced. The system architecture can be seen in Figure(20).

Figure(21) shows an example image of what a segmentation mask looks like using mask R-CNN.

Figure(21) Segmentation mask using Mask R-CNN [40]

3. Project

3.1. Concept

The main concept of this project is to find out the best method to detect and recognize objects with a neural network. After intensive research it was decided to use YOLOv3 since every frame of the picture could have a different size, content quality and background noises. In (section 3.2.1) the YOLO method will be explained. To train the chosen method around 1000 Images will be collected and processed. All of the Images will be obtained from open source websites and the process of the training will be explained later on.

3.2. Used Methods

For the practical implementation, it was decided to focus on one object recognition method. The choice is made with the main focus on YOLOv3. YOLOv3 relies on the fastest possible analysis of an image

and is one of the leading recognition methods in this area. It was decided to use a self-contained implementation for YOLO, which is based on the original realization of the inventor. Alternatively, there would also be implementations in various frameworks.

3.2.1. YOLO – You Only Look Once

You Only Look Once (YOLO) is a widely used object detection method that was developed by Redmon et al. [37] and has since been further developed in several steps. You see object recognition as a regression problem in which objects are determined from the pixels of the input image with the help of a neural network. A recognized object is described by a bounding box, which clearly defines its position and size. Furthermore, as many class probabilities as there are possible classes in the respective application are determined for each recognized object. A class probability value is determined for each class - the sum of these values is one. A recognized object is assigned to the class with the highest class probability.

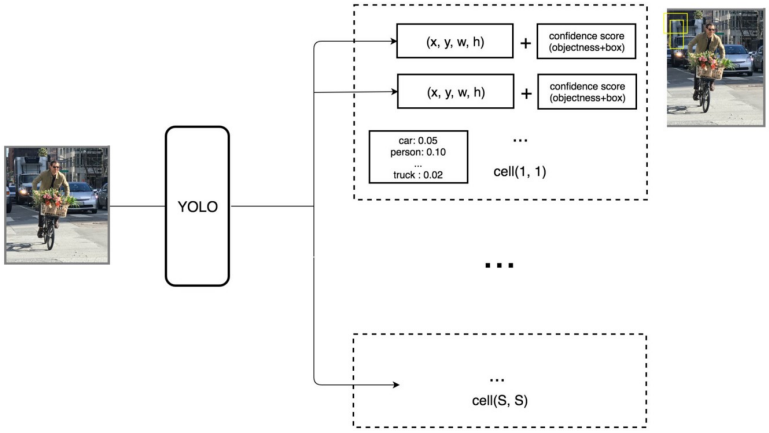
3.2.1.1. Theoretical background for YOLOv1

In YOLO [23] the image is divided into a grid with a size of S x S. The cell of the grid field in which the center of an object falls is also responsible for the recognition of this object. For this purpose, bounding boxes (B) are predicted for each cell. Each of these bounding boxes gives a statement of five values. These are the X position, the Y position, the width (W) and the height (H) of the box and a confidence value that indicates the probability that the box contains an object, regardless of what it is. The X and Y positions indicate the center of the box relative to the grid boundary. The width and height of the box are given relative to the entire image. In addition, class probabilities (C) are predicted for each grid field. In summary in the YOLOv1 each bounding box is described by coordinates (x, y, w, h) and a confidence score. The Confidence score shows how accurate the

bounding box is or how likely that the bounding box contains an object. Formally, the confidence score of a bounding box is thus defined as follows:

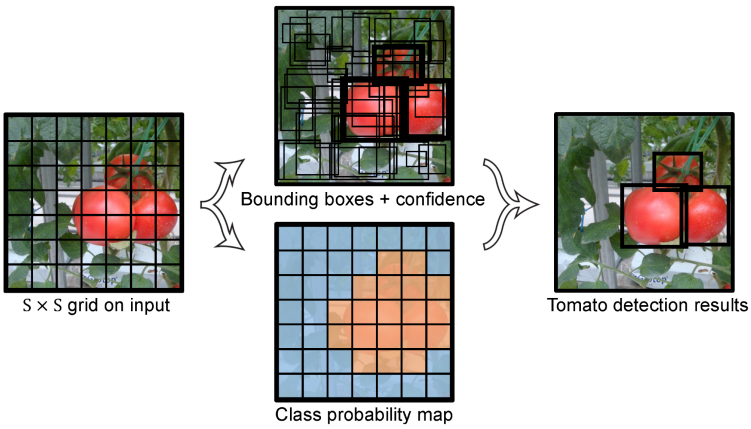
$$\text{Confidence} = \text{pr(Object)} \times IoU_{pred}^{truth}$$

Pr(Object) is the Objectness Score, which determines how likely it is that the box contains an object.
IoU_{pred}^{truth} determines how accurate the size dimensions and position of the box are compared to the dimensions of the contained object. An overview of the parameters is shown graphically in Figure(22).



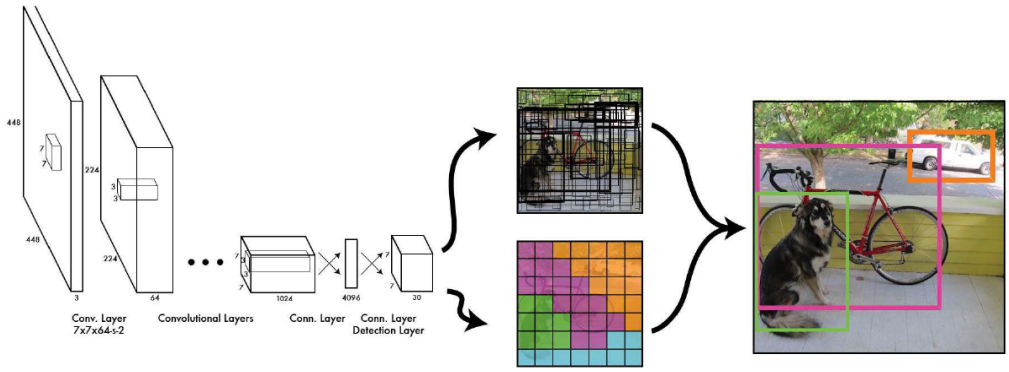
Figure(22) Bounding box Elements [42]

In Figure(23) the image on the left shows the division into a grid field. The Image above shows the predicted bounding boxes, whereby the line width is to be interpreted as a confidence value.



Figure(23) YOLO model detection

In the image below displays the class that was most likely found for a cell. The result can be seen in the picture on the right. Here only the bounding boxes for which the confidence value exceeds a certain threshold value are displayed and combined with the class forecast of the associated cell. This means that a maximum of exactly one object can be recognized per cell, regardless of how many bounding boxes are generated per cell. This has the disadvantage that an image with many small objects cannot be recognized correctly. Most of them are ignored here. The network model consists of 24 convolutional layers, followed by two fully connected layers. 1 × 1 filters are used to reduce the number of parameters. The exact architecture can be seen in Figure(24). YOLO achieved an average of 63.4% at 45 fps. Alternatively, there is another implementation of the neural network. This is called Fast YOLO (YOLOv2) and consists of nine instead of 24 convolutional layers. Fast YOLO achieved a rate of 52.7% at 155 fps.



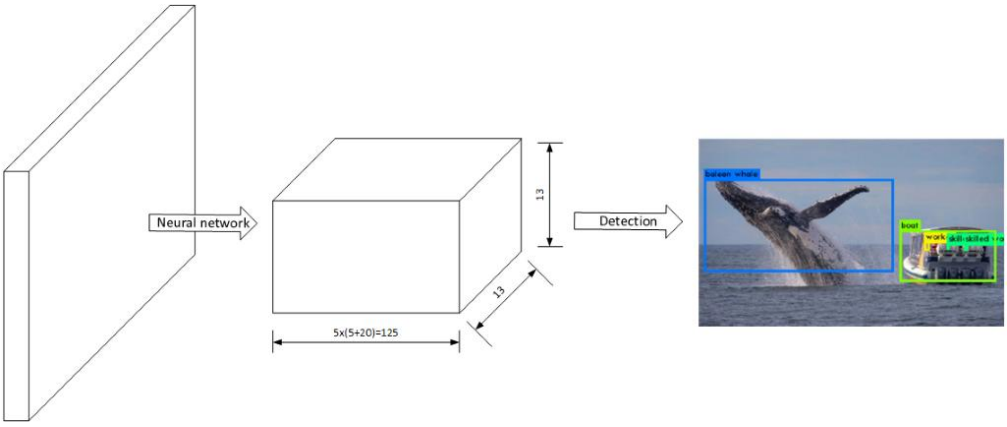
Figure(24) YOLOv1 Network pipeline

3.2.1.2. Limitations of YOLOv1

YOLOv1 has a limitation in predicting nearby objects as each grid cell is limited to predict two boxes, which leads to limitation in the number of classes. Each grid may have only one class. Furthermore YOLOv1 has a high localization error and can detect up to 49 objects in an image.

3.2.1.3. Theoretical background for YOLOv2

After YOLOv1 was released and the limitations were documented, researchers continued to work on improving the object detector. While comparing YOLOv1 to other two stage approaches such as Fast R-CNN and Faster R-CNN. It was noticed, as mentioned above, the occurrence of many localization errors in addition to the low recall. For this reason, Redmon et al. created YOLOv2 [24][53] which is an improved version of YOLOv1. YOLOv2 draws its improvements on the Faster R-CNN Idea. Redmon et al. introduced in his paper the usage of **anchor boxes** (section 2.4.1.3) and the **batch normalization** in YOLOv2, as a solution for the poor prediction ability of the bounding boxes, especially for multiple targets in a grid.



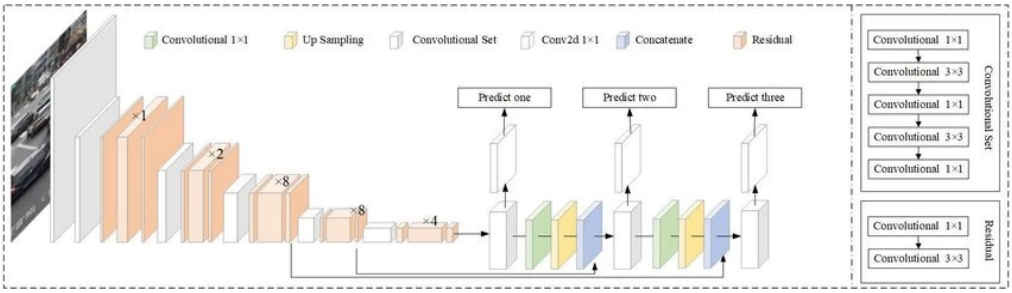
Figure(25) YOLOv2 model

Let's talk shortly about the idea of Batch Normalization. Of Course normalization is a well known data preprocessing tool which generalizes the model well. It ensures that the numerical data is scaled unifiedly without distorting its shape. As for the batch normalization; it is adding an extra layer which does normalization and standardization processes to the imputed data in the deep neural network. This layer is inserted after a full CNN layer. Other than these two steps in general the network structure hasn't changed much from YOLOv1.

3.2.1.4. YOLOv3

YOLOv3 [25] has some minor improvements compared to its predecessor. One of the main improvements in YOLOv3 is that it can detect two objects that are close to each other, where this isn't possible in YOLOv1 & v2. In addition, YOLOv3 provides a prediction for the objectness score by using logic regression. So now Darknet-53, which consists of 53 layers, is used as the network. As in the ResNet architecture (section 2.5.5.3), certain layer outputs are forwarded backwards in the network. Further 53 layers are added on it, which makes the total of the stacked layers in the YOLOv3 106 layers. The architecture of the network can be seen in Figure(). Furthermore, instead of the mean

square error, binary cross entropy is used as a cost function in this network. The implementation is freely accessible online5.

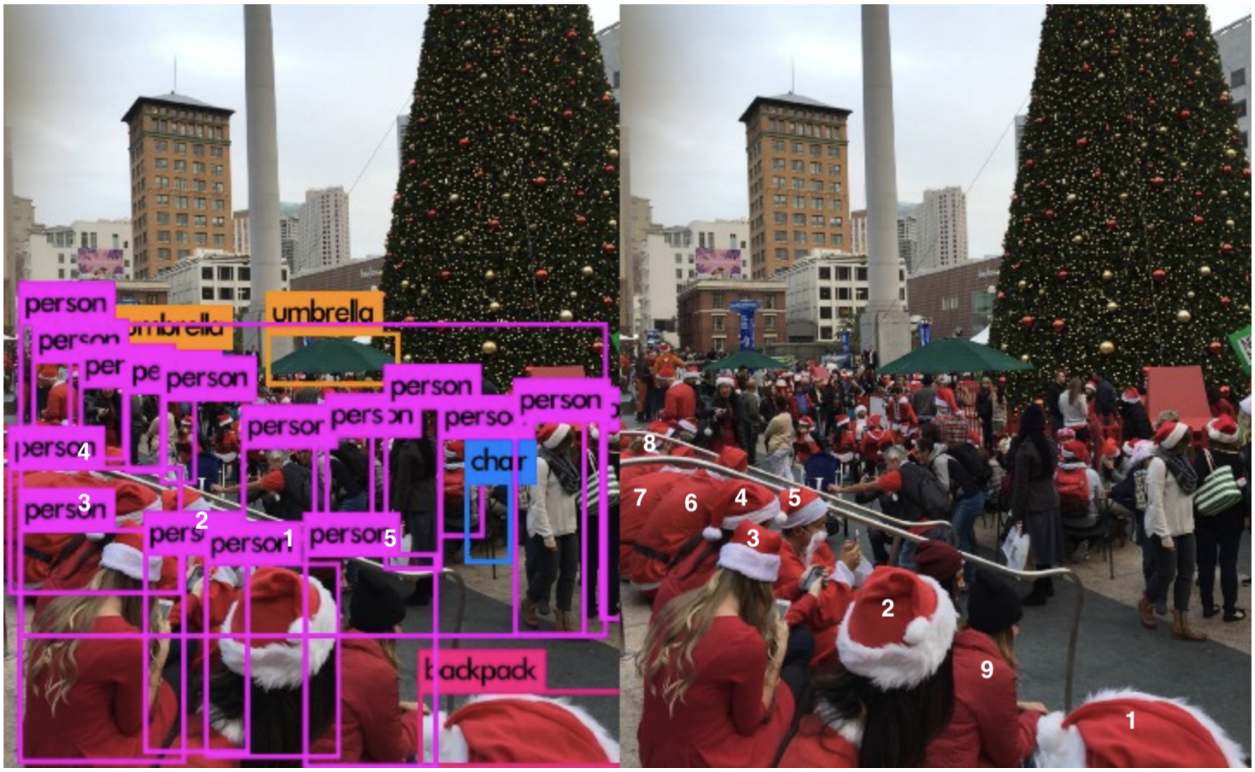


Figure(26) YOLOv3 network Architecture

3.2.2. Limitation of YOLO

YOLO can only predict a limited number of bounding boxes per grid cell, 2 in the original research report. And while this number can be increased, only one class prediction can be made per cell, which limits detection when multiple objects appear in a single grid cell. Hence, it fights with limited groups of small objects like flocks of birds or several small objects of different classes.

In Figure(27) you can see that there are nine people sitting down in the lower left corner of the picture (numbered in white); only 5 people were detected by YOLO. This leads to the conclusion that YOLO could miss objects that are too close.



Figure(27) Non accurate number of person due to the closeness of [42]

4. Implementation of the Project

4.1. Programming language

The most popular programming language for data science projects according to dasca (Data Science Council of America)[55] is Python as 66% of data scientists use python daily and 84% of these Data scientists use Python as their primary programming language. The reason behind python's popularity among data scientists is that it allows the developers to write programs for machine learning tasks with ease. In addition to that it is possible to develop libraries and tools from scratch. Furthermore it offers a variety of ready to use machine learning libraries, add-ons models and frameworks which simplify the development process and time[54].Therefore Python programming language will be used to develop this project. In addition

to Python some HTML and CSS has been used to make a simple webpage where pictures could be uploaded and processed.
One of the limitations of implementing YOLOv3 is that it could only be developed on a linux system. Furthermore, in order to use YOLOv3, OpenCV12 must also be available on the system. Therefore the Python programming language, OpenCV12 and a computer with a Linux operating system will be used for the project. More tools will also be used to be able to self train the used dataset(section 4.3).

4.2. Deep Learning Frameworks used in the project

To develop the Object detection with YOLOv3 project the following popular machine learning platform and technologies has been used

4.2.1. TensorFlow

TensorFlow is an open source machine learning (ML) platform that offers various levels of abstraction. Thus, this framework is recommended for both beginners and experienced users who want more control over the network. The official high-level Keras API can be used to create a neural network. For larger ML tasks, the Distribution Strategy API can be used, which is designed to distribute the training to different hardware devices without having to change the model itself. Furthermore, TensorFlow is language and platform independent. So it is possible to use it for desktop, mobile, web and cloud applications. TensorFlow supports programming languages such as Python, C ++ and R. The framework also has a tool called TensorBoard, which offers an effective representation of network modeling and performance. There is extensive documentation and the largest community in the field of deep learning [30].

4.2.2. Qt5

Qt is a framework with which (GUI) applications can be developed for different platforms. Together with Python3, Qt5

can be used in particular to create applications with a graphical user interface.

4.2.3. Keras

Keras is a high-level API for neural networks in Python that can be used in combination with three different backends. These include TensorFlow , CNTK and Theano. With this framework, the focus of development is on a simple interface and on enabling rapid test execution. Keras supports both convolutional networks and recurrent networks and a combination of the two. The CPU or GPU can also be used for execution. The greatest advantages of Keras are the ease of use, the modularity and the simple expandability with new modules. In order to be able to use the full range of Keras, additional software or additional Python modules must be downloaded and installed. This is described in detail in the documentation [31].

4.2.4. PyTorch

PyTorch is an open source deep learning platform that ensures a seamless transition from prototype development to product development. In order to optimize the performance, the training can be divided into several systems. PyTorch can be used both in Python and in a C ++ runtime environment. Models can be exported here in the Open Neural Network Exchange (ONNX) format and thus used by other platforms that support this standard. Furthermore, PyTorch is supported by cloud platforms, which enables training with a large amount of data [32].

4.2.5. Numpy

The abbreviation NumPy is an acronym for “Numeric Python”. It is a program library that extends the programming language with functions for numerical calculations and mathematical routines. Efficient computing with large matrices and arrays is possible with the help of Python. This particular program is used because all big data

and deep learning applications programmed with Python use NumPy's mathematical functions, as the Python programming language is not optimized for numerical calculations.

4.2.6. Pip

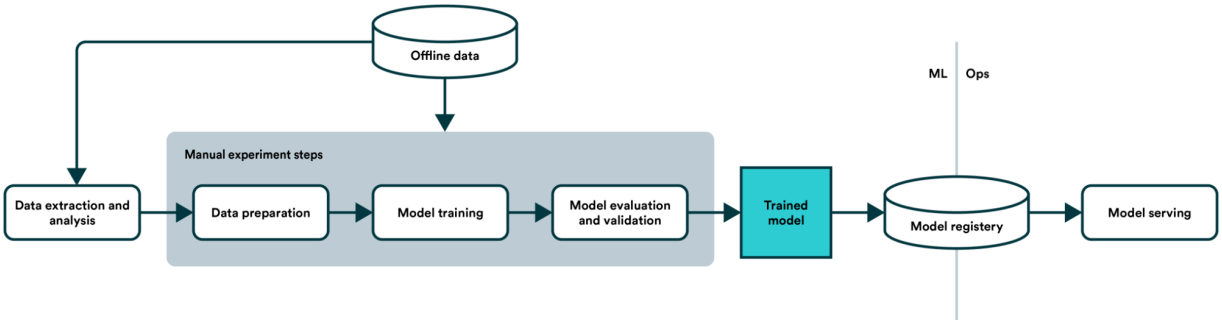
As for **Pip:** it is an acronym that stands for “Preferred Installer Program”. This command line tool allows the OS to install, reinstall, or uninstall Python Package Indexes with one simple and straightforward command: “Pip”.The command pip could also be followed by a number for example “pip3” which stands for the python version being used.

4.3. Training the Data

The powers of the object detection dataset and models have enorm potential, by training the computers to detect what every pixel represents can help in the detecting process of animal pictures to improve the used technologies in the healthcare department. To be able to reach the futuristic technological potentials of object detection and deep learning we must train our datasets in an efficient way. An important factor to choose the best method is that In order to be able to use YOLOv3 with a self trained dataset, the images must be labeled with bounding boxes. Therefore Labelimg tool[56] was used to train the dataset for this project.

Definition:

Labelimg: is a graphical image annotation tool which uses bounding boxes to label objects in images



Figure(28) Dataset preprocessing steps

In Figure(28) the process to train a new data set consists of four main steps which are:

1. Raw data gathering
2. Data Preparation
3. Model training
4. Model evaluation and validation

4.3.1. Raw data gathering

In order to build the object detection website using YOLOv3 technology, a training dataset must be done.
First of all I have used a collection of different already existing pictures of different objects and creatures (i.e. animals, vehicles, and items which could be found in nature..etc). I have used some of the free datasets from different online sources; such as Data.gov, EU Open Data Portal, UCI machine learning repository and Kaggel.

4.3.2. Data Preparation

To prepare the data two aspects must be considered, first of all the collected data (images) are RGB and have a jpg and png type. The second aspect is that the images must have four bounding box variables (x,y min and x,y max). Most importantly in this step the dataset (1400 images) will be split into two categories.

Category A: has around 70% of the data volume (1000) and will be used to train the machine learning Model in the next step of the process. **Category B:** has about 30% of the data volume and will be used to evaluate the model. The dataset has 10 classes which are: Person, Dogs, Cats, Elephant, horse, sheep, Koala, cow, vehicles (Car, Bus, Truck, Motor, Bicycle), traffic light.
Next step in data preparation is labeling the images using Labelimg tool as mentioned above. Labelimg supports YOLO text and VOC XML. VOC XML format will be used during the labeling of the dataset for this project. This format is more universal than YOLO xml and it could be converted with ease.

Definition:
VOC: An annotation format which was developed for the Visual Object Challenge, later on it gained popularity as a common interchange format for object detection labels.

Labelimg: will save the objects as an xml format after drawing bounding boxes in the images automatically around the object in the foto.

For this step a github tutorial was followed [57] the only difference is that I have used another labeling tool [56].

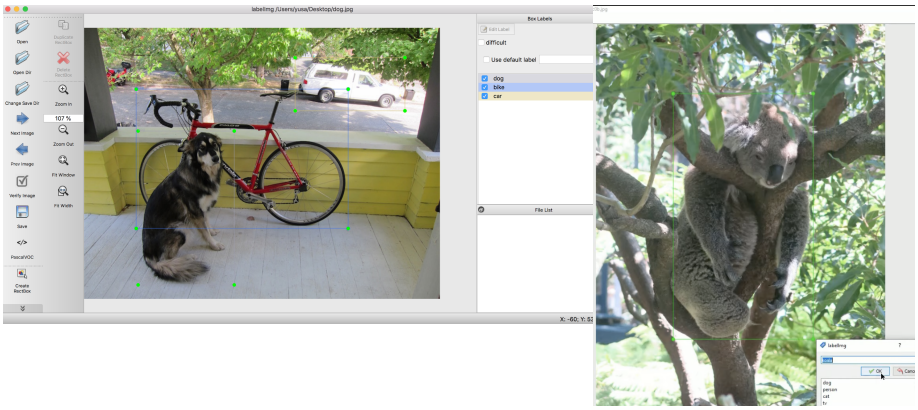
4.3.2.1. Labeling the dataset steps:

1. Install qt5 and labelimg on the linux system via the terminal. The process is shown in Figure(29)

```
sudo apt-get install pyqt5-dev-tools
sudo pip3 install -r requirements/requirements-linux-python3.txt
make qt5py3
python3 labelImg.py
```

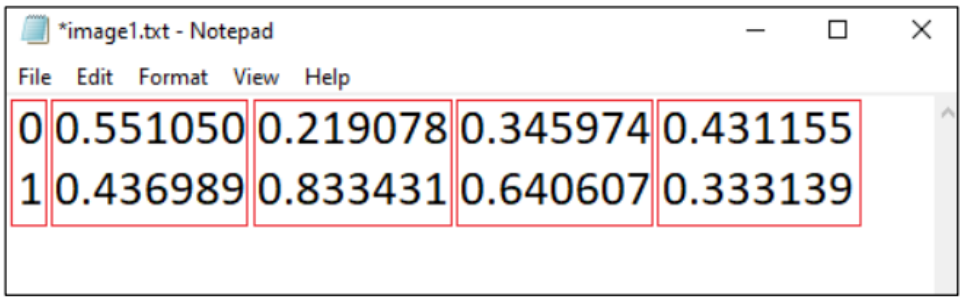
Figure(29) qt5 and Labelmg installation on Linux sys

2. Predefine the classes that shall be used in a “data/predefined_classes.txt”
3. Add a folder of the collected raw data into labelimg and open it via open dir
4. Start creating Rectangular Boxes (Bounding boxes) and annotate them. Repeat for every object in the image as shown in Figure(30)



Figure(30): LabelImg while annotating

5. After annotating all the images the folder shall look like the Figure(31) below. The folder has all the previously collected data plus a file for each annotated image holding the same name as its corresponding image (.txt file). The file will contain a series of numbers



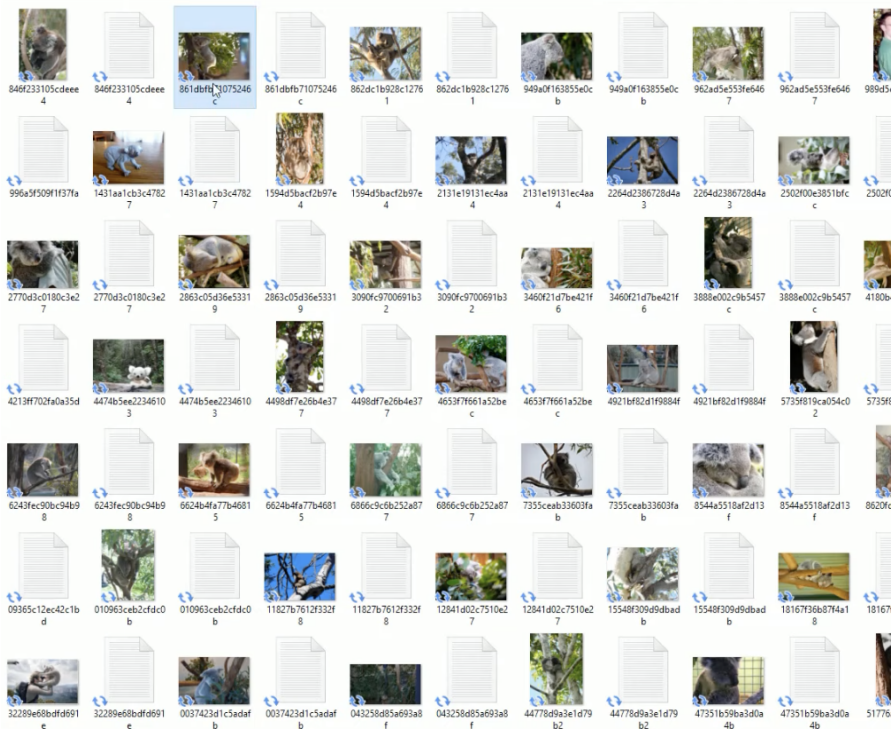
Figure(31) Annotation txt file of a labeled image

For example: (15 0.392578 0.504395 0.566406 0.791992) which represents (ObjectID, Center_X position, Center_Y position, width, height).

ObjectID: Represents the object's category or class. The number of the objectID is determined in the class.txt file.

Center_X position, Center_Y Position: The X, Y center point of the bounding box. Both values are normalized to range between 0 and 1.

Width,Height: Represent the width and height of the bounding box.



Figure(32) Annotated images with annotation txt file

4.3.3. Model Training and Testing

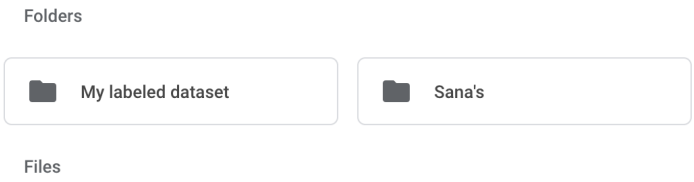
In this step the dataset was trained by feeding the trained_data.txt (Category A) of the dataset into the pretrained YOLOv3 Model. The Model which is used for this project is

trained and tested using **google colab**; with google colab is an environment which allows you to code large datasets in ML and DL completely on the cloud.

4.3.3.1. Model Training and testing Implementation

As mentioned above, the model training and testing implementation will be conducted in google colab. The Implementation took the following steps:

1. Upload the prepared dataset which contains the labeled images to a new folder in the google drive account.



Figure(33): Real time image from my gdrive

2. Import all the important libraries which will be used for the training of the model like (os, Tensorflow, Tensorflow.keras.layers, numpy) and instead of writing my own architecture I tried using the previously used mobile net architecture; after testing this library I have found out through research that it does perform well (Tensorflow.keras.applications.mobilenet_v2). In addition to that I have used a library of call backs for measuring the validation accuracy value. Lastly, to mount the folder of the labeled dataset which is uploaded on google drive I have also added the gdrive library.

Figure(34) Imported libraries/dependencies in google collab

3. Mount the folder from google drive to get the folder through a verification code, then unzip the folder to retrieve all the data. This process can take up some time to finish especially in this project as the dataset was larger than average and multiple data classes have been trained.

4. Use two different data generators one for training and another for validation/Testing as shown in the code Below.

Figure(34) The data generators code in google collab

5. Define Model by using the already defined model Architecture via Mobile net [60]. This architecture model has predefined arguments which could be changed according to the needs of the model.In the two code blocks below you can see the difference between the original Mobile net architecture implementation and the changes which have been made to fit my dataset training needs.

```
#Original Model
tf.keras.applications.mobilenet_v2.MobileNetV2(
    input_shape= None, alpha=1.0, include_top=True,
weights='imagenet',
    input_tensor=None, pooling=None, classes=1000,
    classifier_activation='softmax', **kwargs
)
```

Input shape specifies the input image resolution and for this model training I am using input shape of (224,224,3). Alpha represents the number of filters in each layer which will remain by default. As for the weights file, ImageNet weights shall be used.

```
#Modified Model
tf.keras.applications.mobilenet_v2.MobileNetV2(
    input_shape= (224,224,3), alpha=1.0, include_top=False,
weights='imagenet',
    input_tensor=None, pooling=None, classes=1000,
    classifier_activation='softmax')
```

6. Last step in the dataset training is to clone, configure and compile the dataset with the model.

The model takes some time to train all the dataset ca. 6 hours.

7. During the model testing step the performance of the newly trained dataset will be measured for example in the following code block

4.4. Implementation of project

4.5. System Requirements

This Project runs on an Ubuntu or macOS based computer system. The most crucial requirement is Python version 3.7.6, furthermore the following dependencies are also required:

1. Install Tensor flow, ImageAI,OpenCV, Numpy
2. Use the trained Object detection Model
3. Django v.1.9.8 to build a Web app for uploading and processing the objects in the images

4.6. System Implementation and steps

4.6.1. Backend Implementation (Object detection)

The final stage in the project is to create a backend system using python 3.7.6 and some dependencies mentioned in the section above. During the Implementation phase I have followed a simple tutorial which explains the YOLOv3 object detection implementation phase[59]. The implementation of the backend system (object detection) is done within one class. The class is called “Detection”. An instance of the class wasYOLOv3 Model type was set. The path of the previously trained dataset using google colab is set and loaded. And the Network path is also defined Once the code is running, The function which is incharge of detection was called and parsed in the path of the input and output image. Furthermore, an Iteration over the returned result is performed in order to return the class name of the object as well as the confidence percentage. In this Implementation a python based library called Imageai is used. Imageai produces a detection configuration file which supports

detecting objects in Images by parsing the extracted parameters into the object detection function. Afterwards the uploaded Images are saved In a folder inside of the python project called “Media”. The uploaded image will be processed through the YOLOv3 Model and will be sorted into their corresponding classes after being processed by the MyData trained model. Hence each detected object will be given a name of its category (class) in addition to the confidence rate. The confidence rate will be written in percentage (1-100). After the Object detection process ends, the Processed Image(result) will be saved also in the “media” folder under its uploaded name + the word “detected” to be able to distinguish the original image from the object detection result. In addition to that, two paths for both the input and the result image will be created.

Remark(line 6&7): Tensorflow takes control of the GPUs. Hence I needed to rewrite the environment in order to make GPU(s) visible for the data process.

```
1  from imageai.Detection import ObjectDetection
2  import os
3
4  class Detection():
5      def detect(image_name):
6          os.environ["CUDA_DEVICE_ORDER"] = "PCI_BUS_ID"
7          os.environ["CUDA_VISIBLE_DEVICES"] = ""
8          image_path = "media/"+image_name
9          detected_url = "media/detected-"+image_name
10         execution_path = os.getcwd()
11         detector = ObjectDetection()
12         detector.setModelTypeAsYOLOv3()
13         detector.setModelPath( os.path.join(execution_path , "MyData-yolov3.h5"))
14         detector.loadModel()
15         print("reached here so far")
16         print(os.path.join(execution_path , image_path))
17         detections = detector.detectObjectsFromImage(input_image=os.path.join(execution_path ,
               image_path),
18                 output_image_path=os.path.join(execution_path , detected_url), input_type="file",
                       output_type="file")
19         return "/"+image_path, "/"+detected_url
20 # for eachObject in detections:
21 #     print(eachObject["name"] , " : " , eachObject["percentage_probability"] )
22
```

Figure(35) The YOLOv3 implementation

4.6.2. Web Application Implementation

The web app is developed using Django.

Django: is a full stack framework written in Python that enables the rapid development of web applications. One of its great advantages is reusable components and integration with multiple plugins.

```python
from django.shortcuts import render, redirect
from django.conf import settings
from django.core.files.storage import FileSystemStorage

from uploads.core.models import Document
from uploads.core.forms import DocumentForm
from .detection import Detection

def home(request):
    documents = Document.objects.all()
    return render(request, 'core/home.html', { 'documents': documents })


def detect(request):
    if request.method == 'POST' and request.FILES['myfile']:
        myfile = request.FILES['myfile']
        fs = FileSystemStorage()
        filename = fs.save(myfile.name, myfile)
        uploaded_file_url = fs.url(filename)
        original_url, detected_url = Detection.detect(filename)
        return render(request, 'core/detect.html', {
            'original_file_url': original_url,
            'uploaded_file_url': detected_url
        })
    return render(request, 'core/detect.html')
```

Figure(36) Image Detector website Implementation

The Website is a simple frontend which contains three different views (pages).

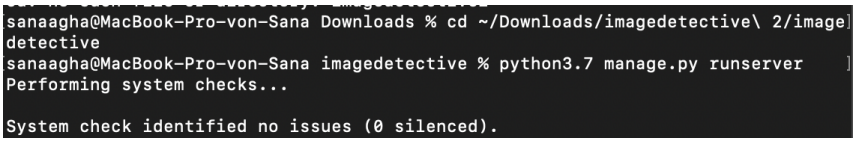
- **The Home page:** Contains the name of the project “Image Detective”. A start Button, which allows the user to upload an image to detect its objects. Furthermore four static Images which show some of the results of the tool.
- **Upload Image Page:** This page contains an upload button called “SELECT IMAGE” and a “Submit” button to confirm and start the upload and object detection processes.
- **Result Page:** Shows two Images. The image on the left hand side shows the uploaded image before the object detection processing with a link path above it to, which allows the user to see the Image in a bigger (Original size). The Right handside Shows the processed image which contains the detected objects plus a bounding box with its corresponding class and confidence rate. Also on top of this image a path is provided which allows the user to see the image more closely.

4.6.3. End Result

The end result of the implementation will be demonstrated with images in this section. In addition to this the full code is downloadable from Github[61] with a txt file of all the needed dependency versions.

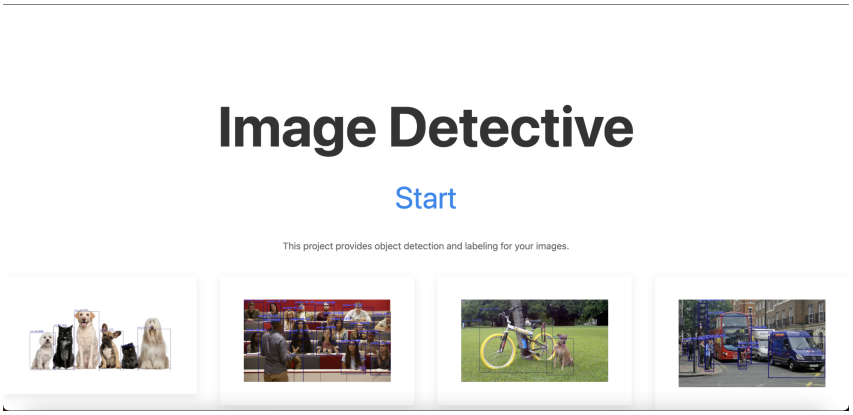
4.6.3.1. Step by step demonstration of the Image detective project

1. To start the Image Detector website, a command “python3.7 manage.py runserver'' should be performed in the terminal. This command could only be run inside of the correct directory (Project Folder). Furthermore Type the localhost IP “127.0.0.1:8000” in the browser (preferably Chrome).



Figure(37) Private computer’s Terminal while running localhost

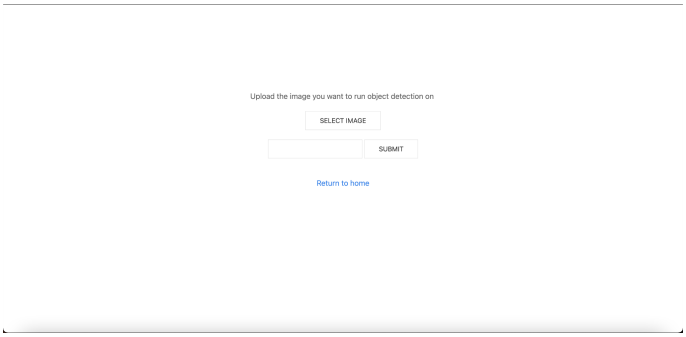
2. The Home page of the Image Detective Project:
   Here you can see some images detected by the image detective Project



Figure(38) Image Detective Home page UI

3. Uploading an Image:

In this step I have picked an Image of different types of dogs and cats to upload to the website

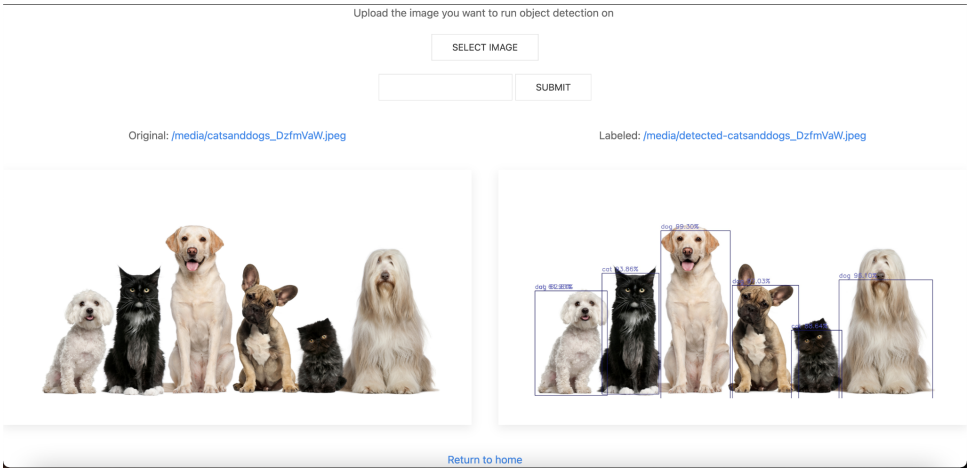


Figure(39) Image Detector Upload page UI

I have chosen the following image to upload inorder to test the website

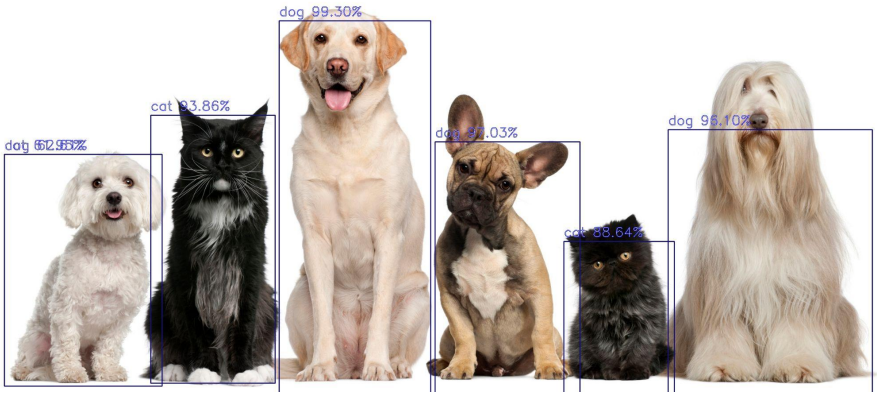
Figure(40) A randomly chosen Image

4. Result:
   After processing time the website shows both the original image (on the left hand side) and the detected objects (on the right hand side). Furthermore the confidence rate (in %) is also shown next to the object name



Figure(41) Image Detector, Object detection final result

Here is a close up picture of the result:
You can clearly see that the first dog from the left has overlapping classes. Of Course this type of mistake is expected from a newly trained dataset module. To achieve better accuracy in the future, one should train more data to the model.



Figure(42) Close up of the final result Image

5. Conclusion

Deep learning is very suitable for object recognition problems. There is no need for complicated, specific algorithms that would be different for each object. Instead, the same project structure can be used all the time. Only the data sets for training the network and some configuration parameters have to be selected individually. There are a number of frameworks for implementing deep learning based object detection solutions, the most popular and widely used one is TensorFlow. This approach was used during this thesis. In regards to training and testing a dataset for a Model, given there is only a small amount of data available. The network is trained with another, larger data set or initialized with weights that have already been trained. About 1400 images were labeled for the detection of advertisements, which were divided into a training data set and a validation data set in a ratio of about 70 to 30.

References

[1] Shubham Panchal. Artificial Neural Networks — Mapping the Human Brain. Medium. März 2018. url: https://medium.com/predict/artificial-neural-networks-mapping-the-human-brain-2e0bd4a93160 (Shubham #) (Shubham 1) (page. 3)

[2] Yann LeCun, Yoshua Bengio und Geoffrey Hinton. „Deep Learning". Nature 521.7553 (LeCu 2)(Mai 2015), S. 436–444 (page. 4-7).

[3] Unsupervised Feature Learning and Deep Learning Tutorial. url: http://ufldl.stanford . edu / tutorial / supervised / MultiLayerNeuralNetworks/ (Unsupervised Feature Learning and Deep Learning Tutorial 3) (page 4, 5).

[4] Stanford University. CS231n Convolutional Neural Networks for Visual Recognition. https://cs231n.github.io/convolutional-networks/ (CS231n Convolutional Neural Networks for Visual Recognition 4)(page. 13–16).

[5] https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3b (Saha) (Shubham 5)d2b1164a53

[6] Ayeshmantha Perera. What is Padding in CNN's. Medium. (page. 14, 15).

[7] https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[8] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology.

[9] Jiang, X., Lu, M. & Wang, SH. An eight-layer convolutional neural network with stochastic pooling, batch normalization and dropout for fingerspelling recognition of Chinese sign language

[10] Yann LeCun u. a. „Gradient-Based Learning Applied to Document Recognition". Proceedings of the IEEE 86 (Nov. 1998).

[11] Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton. „ImageNet Classification with Deep Convolutional Neural Networks". In: Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1. NIPS'12. Lake Tahoe, NV, USA: Curran Associates Inc., Dez. 2012. (PAGE. 16).

[12] https://duchesnay.github.io/pystatsml/deep_learning/dl_cnn_cifar10_pytorch.html

[13] Karen Simonyan und Andrew Zisserman. „Very Deep Convolutional Networks for Large-Scale Image Recognition". In: Proceedings of the 3rd International Conference on Learning Representations. ICLR'15. San Diego, CA, USA, Mai 2015

[14] https://neurohive.io/en/popular-networks/vgg16/

[15] Jason Yosinski u. a. „How Transferable Are Features in Deep Neural Networks?" In: Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2. NIPS'14. Montreal, Canada: Curran Associates, Inc., Dez. 2014,

[16] https://madhuramiah.medium.com/deep-learning-using-resnets-for-transfer-learning-d7f4799fa863

[17] Ross Girshick u. a. „Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: Proceedings of the 27th IEEE Conference on Computer Vision and Pattern Recognition. CVPR'14. Columbus, OH, USA: IEEE

[18] Ross B. Girshick. „Fast R-CNN". In: Proceedings of the 15th IEEE International Conference on Computer Vision. ICCV'15. Santiago, Chile: IEEE, Dez. 2015

[19] Fei-Fei Li, Justin Johnson und Serena Yeung. Detection and Segmentation. Mai 2017. url: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

[20] Liu Wei, Lu Runge und Liu Xiaolei. „Traffic sign detection and recognition via transfer learning". In: Proceedings of the 30th Chinese Control And Decision Conference. CCDC'18. Shenyang, China: IEEE

[21] Shaoqing Ren u. a. „Faster R-CNN: Towards Real-time Object Detection with Region Proposal Networks". In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1. NIPS'15. Montreal, Canada: Curran Associates, Inc., Jan. 2015

[22] Kaiming He u. a. „Mask R-CNN". In: Proceedings of the 16th IEEE International Conference on Computer Vision. ICCV'17. Venice, Italy: IEEE, Okt. 2017

[23] Joseph Redmon u. a. „You Only Look Once: Unified, Real-Time Object Detection". In: Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition. CVPR'16. Las Vegas, NV, USA: IEEE, Juni 2016

[24] Joseph Redmon und Ali Farhadi. „YOLO9000: Better, Faster, Stronger". In: Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition. CVPR'17. Honolulu, HI, USA: IEEE, Juli 2017

[25] Joseph Redmon und Ali Farhadi. YOLOv3: An Incremental Improvement. Techn. Ber. Apr. 2018

[26] Jonathan Huang u. a. „Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors". In: Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition. CVPR'17. Honolulu, HI, USA: IEEE, Juli 2017

[27] Wei Liu u. a. „SSD: Single Shot MultiBox Detector". In: Proceedings of the 14th European Conference on Computer Vision. ECCV'16. Cham, Germany: Springer International Publishing, 2016

[28] Tsung-Yi Lin u. a. „Focal Loss for Dense Object Detection". In: Proceedings of the 16th IEEE International Conference on Computer Vision. ICCV'17. Venice, Italy: IEEE, Okt. 2017

[29] Jonathan Huang u. a. „Speed/Accuracy Trade-Offs for Modern Convolutional Object Detectors". In: Proceedings of the 30th IEEE Conference on Computer Vision and Pattern Recognition. CVPR'17. Honolulu, HI, USA: IEEE, Juli 2017

[30] TensorFlow. url: https://www.tensorflow.org/

[31] Keras. url: https://keras.io/

[32] PyTorch. url: https://www.pytorch.org

[33] Caffe. url: http://caffe.berkeleyvision.org/

[34] Theano. url: http://deeplearning.net/software/theano/

[35] Microsoft Cognitive Toolkit (CNTK) - Documentation. url: https://docs.microsoft.com/en-us/cognitive-toolkit/

[36] Microsoft Cognitive Toolkit. url: https://www.microsoft.com/en-us/cognitive-to olkit/

[37] Christian Szegedy u. a. „Going deeper with convolutions". In: Proceedings of the 28th IEEE Conference on Computer Vision and Pattern Recognition. CVPR'15. Boston, MA, USA: IEEE, Juni 2015

[38] Kaiming He u. a. „Deep Residual Learning for Image Recognition". In: Proceedings of the 29th IEEE Conference on Computer Vision and Pattern Recognition. CVPR'16. Las Vegas, NV, USA: IEEE, Juni 2016

[39] Jenssen, Robert. „Intelligent Monitoring and Inspection of Power Line Components Powered by UAV and Deep Learning ".IEEE, Juni 2019

[40] Segmentaion mask using Mask R-CNN https://blog.paperspace.com/mask-r-cnn-in-tensorflow-2-0/

[41] https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html

[42] https://jonathan-hui.medium.com/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

[43] Rohith Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. Towards Data Science. Juli 2018. url: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571 365e

[44] https://bitmovin.com/object-detection/

[45] https://archive.ics.uci.edu/ml/datasets.php

[46] https://www.kaggle.com/datasets

[47] Data.gov

[48] https://data.europa.eu/data/datasets?locale=en&minScoring=0

[49] Jaron Collis. Glossary of Deep Learning: Bias. Medium. Apr. 2017. url: https://medium.com/deeper-learning/glossary-of-deep-learning-bias-cf49d9c895e2

[50] Rohith Gandhi. R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms. Towards Data Science. Juli 2018. url: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571

[51] Rohith Gandhi. Support Vector Machine — Introduction to Machine Learning Algorithms. Towards Data Science. 7. Juni 2018. url: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

[52] Uijlings, J.R.R., van de Sande, K.E.A., Gevers, T. et al. Selective Search for Object Recognition. Int J Comput Vis 104, 154–171 (2013). url: https://doi.org/10.1007/s11263-013-0620-5

[53] Joseph Redmon und Ali Farhadi. YOLO9000: Better, faster, stronger

[54] https://www.netguru.com/blog/python-machine-learning

[55] https://www.dasca.org/world-of-big-data/article/top-6-programming-languages-for-data-science-in-2021

[56] https://github.com/tzutalin/labelImg

[57] https://github.com/AntonMu/TrainYourOwnYOLO

[58] https://colab.research.google.com/github/hardik0/Deep-Learning-with-GoogleColab/blob/master/Darknet_YOLOv3_Guns_Detection.ipynb

[59] https://github.com/OlafenwaMoses/ImageAI

[60] https://www.tensorflow.org/api_docs/python/tf/keras/applications/mobilenet_v2/MobileNetV2

[61] https://github.com/Sana93JA/ImageDetective