Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Control Engineering and Information Technology

Object Detection as Multi-Task Learning

Bachelor's Thesis

Author
Nordin Belkacemi

Advisor
Dr. Márton Szemenyei
Tamás Kern

December 8, 2022

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Belkacemi Nordin*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2022. december 8.

Belkacemi Nordin
hallgató

Absztrakt

A multitaszk tanulás révén a gépi tanulásos modellek egyszerre több feladatot is meg tudnak tanulni. A kutatók azt állítják, hogy egy modell javíthatja az egyes feladatokon mért teljesítményét azáltal, hogy más feladatokból származó információkat hasznosít. Multitaszk tanulás esetén általában a szokásosnál több adatra és időre van szükség, ezért érdemes azt megvizsgálni, hogy megmaradnak-e a módszer előnyei, ha több részre osztunk egy feladatot. Ebben a szakdolgozatban pont ezt vizsgáljuk objektum detektáló modelleken.

A YOLO algoritmus tanulmányozása és implementálása után újraterveztem a rendszert úgy, hogy külön-külön támogassa az adathalmaz objektumosztályainak részhalmazain (ún. osztálycsoportjain) történő detektálást. Három modellt hasonlítottam össze: a single task (hagyományos) verziót, a 3 osztálycsoportot használó multitaszk verziót és az osztálycsoportonként egy osztályt használó modellt. A modelleket a KITTI és a PascalVOC adathalmazokon tanítottam/értékeltem ki.

A PascalVOC-on tanított modellek eredményei határozottan alacsony teljesítményt mutattak (1 mAP alatt). Ez valószínűleg korai túlillesztés miatt jelentkezik, amelyet adatmennyiség, illetve adat augmentáció elégtelenség/hiány okoz. Az utóbbi probléma kijavítása után a hátralévő idővel és erőforrásokkal sikerült KITTI-n betanítani a modelleket, amelyek jelentős javulást mutattak, de közülük a single task modell teljesített a legjobban. Azt is meg lehet figyelni, hogy minél több feladatra bontjuk az objektumdetektálást, annál alacsonyabb mAP-t eredményez. A túlillesztés továbbra is probléma, de ez elkerülhető további adat augmentációval, nagyobb mennyiségű tanítási adattal és a hiperparaméterek hangolásával, valamint a tanulási ráta ütemezésével.

Abstract

A recent area of research called multitask learning has presented cases in which neural networks show the ability to learn multiple tasks at once. Researchers proposed that this way, a model can improve performance on individual tasks by leveraging information from other tasks. Multitask learning usually requires more data and time than usual, which is why we may ask ourselves if the benefits of the method still remain if we simply divide one task into several parts. In this thesis project, this idea is applied to object detection.

After studying and implementing the YOLO algorithm, I redesigned the system to support detection on subsets of a dataset's object classes separately instead of all of them at once. Three models were compared: the single task (traditional YOLO) version, the multitask version using 3 class subsets, and the one using the maximum amount of subclasses (one class per subclass). The models were trained and evaluated on the KITTI and PascalVOC datasets.

Results on PascalVOC showed significantly low performance on all trained models (below 1 mAP) most likely due to early overfitting, caused by insufficient data and data augmentation. However, after fixing the latter issue, with the remaining time and resources, I ran trainings on KITTI and the model showed significant improvement, but the single task model performed better than multitask versions. Overfitting still remains an issue, but this can be avoided with even more data augmentation, larger amounts of training data and the tuning of hyperparameters, as well as learning rate scheduling.

Chapter 1

Introduction

Deep learning is a branch of machine learning that has been driving numerous computer vision systems from snapchat filters to driver assistance features. The extensive research in image recognition has led to findings that are used for other tasks such as object detection and semantic segmentation. Early works focused on how convolutional neural networks (made up of convolutional layers) allow for the extraction of relevant features in an image. LeNet [13] is the first example of such a system, which was motivated by the task of recognizing handwritten digits. However, it was not until the capabilities of convolutional layers were fully leveraged, that convolutional neural networks became popular: this happened with the appearance of AlexNet [11], which won the ILSVRC-2012 image recognition competition by a significant margin. After AlexNet, even Google researchers pitched in with GoogleNet and image classifier models were getting better and better each year. A general trend insued; the advancement of computational resources and the growth of the amount of training data, allowed for deeper and deeper neural networks which performed better and better. Without these advances, performance of object detection models would be nowhere near what they are today.

The goal of object detection is to accurately identify and locate objects of interest in an image or video. It has numerous applications in various industries such as self-driving cars, surveillance, and medical imaging. Early methods included running image classifiers on regions of an image, which was computationally demanding in its naive implementations. However, getting region proposals from deep neural networks alleviated some of this burden. Still these detectors were not the fastest. More modern research has proposed one stage detection methods, where there is no need to run image classifier networks on regions, but instead a network would take an image as input and directly output bounding boxes around relevant objects. YOLO was the first method to provide the best combination of speed and performance, allowing for real-time object detection. Another method, CenterNet, with the main advantage of its simplicity allows for real time object detection viewing objects as keypoints. YOLO has many versions (7 at the time of writing) and with each version, researchers push the boundaries of speed and performance with an empirical approach; trying new methods originating from modern research and evaluating them. Similarly, in this thesis, we propose a new approach to object detection, inspired by multitask learning research.

Multitask learning refers to the ability of a neural network to learn more than one task at the same time. Instead of focusing on optimizing a model for just one specific task, it has been shown that information coming from training signals of a related task can help us do even better, increasing the ability of the network to generalize. In object

detection, bounding boxes are drawn and object classes are predicted for each bounding box. Instead of completing this "single task" over a set of object classes, we can divide it up into subtasks, in which detection is done separately on subsets of the original set of object classes. This way, we don't need more training data than we already have, and perhaps, the network could learn how to generalize better. The goal of this thesis is to implement an object detection model capable of detecting objects on multiple groups of classes at the same time, and evaluate/compare the speed and performance of various multitask models relative to the single task variant and to each other.

The proposed solution is to implement the YOLO algorithm in a way that it supports multitask learning. The model has as many detection heads as there are class groups. Anchor box or cluster centroid computation was done with k means clustering on each class group. The training and validation datasets used were PascalVOC and KITTI. After training, mAP and inference time were measured for each model and detections were visualized.

The results showed that with the implementation I have at the moment, multitask learning does not benefit performance. Due to a high confidence loss, with the single task model producing too many detections per image, the more detection heads there are, the more predictions the model outputs, hence the drop in mAP, with the increase of the number of class groups used (number of tasks). The models trained on PascalVOC did not use enough data augmentations, which is why their performance is significantly lower than those trained on KITTI (which used augmentations).

Chapter 2

Related Work

2.1 Deep learning in computer vision

As the prevalence of cameras and image data have grown over the past decades, with today's computational resources, many computer vision techniques have been (and are still being) developed. A high number of these techniques are driven by research done in the field of artificial intelligence, namely Deep Learning. While the core ideas behind artificial neural networks have been around since the 50s and were further studied throughout the following decades, the sufficient advances in hardware only happened in the 2000s. The following section gives insight on several important concepts and results in the domains of image classification all the way to state-of-the-art object detection models.

2.2 Neural Networks

Artificial Neural Networks are a mathematical model inspired by how the brain works. Signals (e.g. pixels of an image) are recieved as input, and just like the neurons in the brain are activated through synapses, neurons in an neural network are connected by weights. A higher weight means that the source neuron's signal will be transmitted with greater magnitude and vice versa. In a typical neural network, neurons are arranged in layers, and each neuron's activation depends on the weighted sum of a previous layer's neurons. More specifically, the weighted sum of neurons is fed into a non linear activation function (like the sigmoid function[1] or ReLU [16]).

This construct has been proven to belong to a class of universal approximators, as stated by the title of a paper [8] from 1989. Functions are input-output mappings; an input could be an image, and the output could be the class of the image (e.g. bird image). The latter example is a virtually infinite input-output mapping, but we can help neural networks recognize patterns: Just like the human brain can infer through memory, so can neural networks in a sense. It is possible to train/teach a network what output it should produce to given inputs through gradient descent. A cost function (or loss function) quantifies the magnitude of the error in the neural network's output compared to a target (what the output should have been). Gradient descent is a method that minimizes loss, by iteratively updating the parameters of a model (in our case the weights of a neural network). The update rule makes use of the gradient of the loss w.r.t. model parameters, which is by definition the direction of greatest change of the loss.

[1] $\sigma(x) = 1/(1 + \exp(-x))$

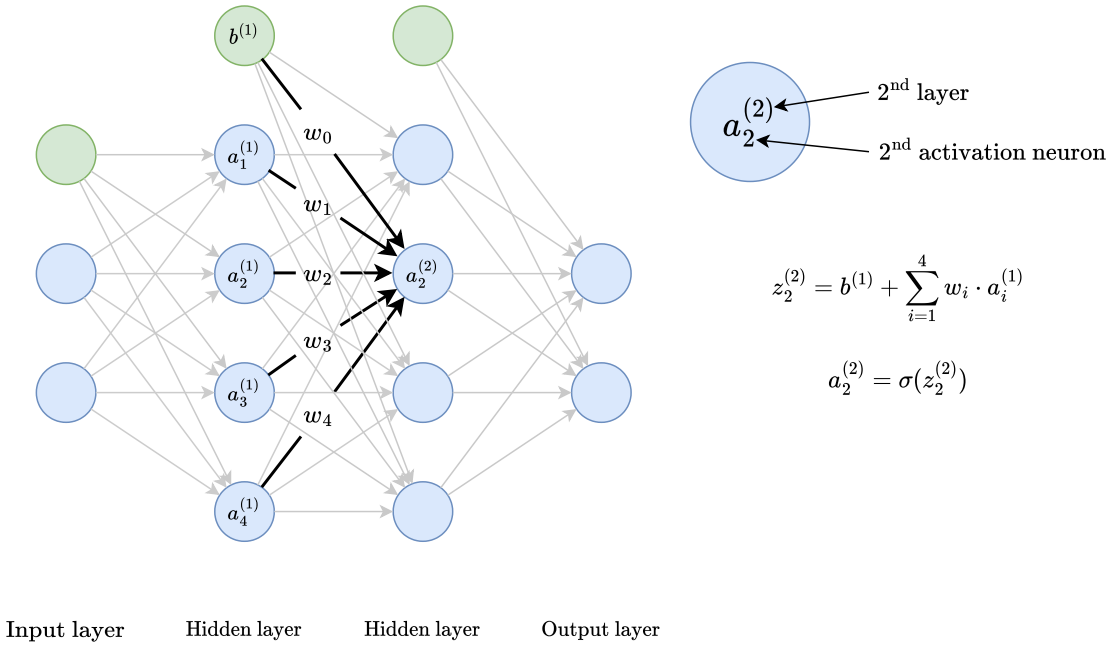

Figure 2.1: Illustration of a neural network and how its neurons' activations are computed: z is used to denote the weighted sum of the previous layer's activations (in blue) and bias (in green), and a is the activation of a neuron, which is obtained by computing the sigmoid of z. As mentioned previously, the activation function does not have to be the sigmoid in every neural net (it could be ReLU or other functions).

Taking a step back from neural networks, it can be helpful to consider the example of linear regression. Neural networks are function approximators and so is a linear equation that fits to data points (see figure 2.2).

On figure 2.2, the method of gradient descent for linear regression is described. It is important to note, that the plot of the loss function and the update rule is simplified for simplicity/visualization purposes: the θ_0 parameter is *fixed*, when in a more accurate description it would not be, the loss function would have to be plotted against both θ_0 and θ_1, resulting in the loss function being a paraboloid like surface and the gradient being a 2 dimensional vector. The idea is still valid here, and the parameteters of a model $h_\theta(x)$ are iteratively being updated to minimize loss.

In neural networks the parameters of the model are its weights and biases. Due to the more complex nature of the model, the most efficient way to calulate gradients of loss w.r.t. weights is through backpropagation. The method makes use of the chain rule, because each layer is computed from previous layer's activations, so in order to compute the gradient w.r.t. weights from one layer, we must first compute the gradient w.r.t. weights from the succeeding layer. In other words, from the last layer to the first, we propagate the gradient: this is where the backpropagation algorithm got its name from. Rumelhart et al. have shown that this procedure is relatively simple and paralellizable.

Based on our understanding of the computation of the gradient in gradient descent so far, it involves using all of the data points available. This is called batch gradient descent, and when considering the large amounts of data used to train a network, it comes with

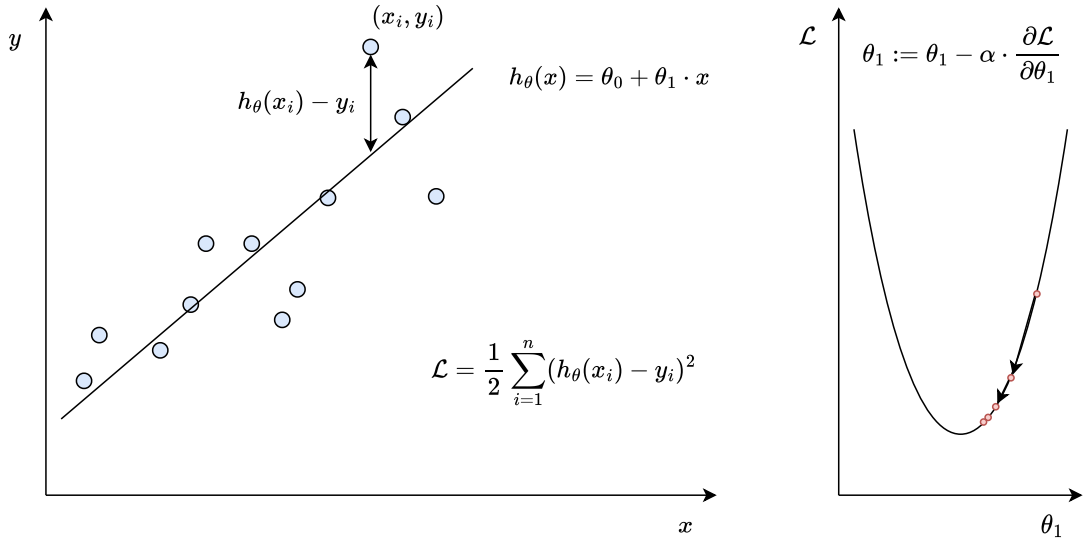

Figure 2.2: Illustration of linear regression. The goal is to fit a line to data points (x_i, y_i). On the left, A model $h_\theta(x)$ approximates the best fit, and its "errors" come from the distance between data points and the line. The loss $\mathcal{L}$ function is therefore written as the sum of squared errors. On the right, a plot of the loss function vs the parameter θ_1 *at a given value of* θ_0 is shown. On the plot the red points indicate the different $(\theta_1, \mathcal{L})$ values that may occur in iterations of gradient descent. The points are connected by arrows showing one step of gradient descent. The equation at the top of the plot on the right is the update rule, i.e. the rule followed when taking one step, in which α is a hyperparameter (has to be adjusted to get best results)

a significant computational burden. Luckily, it turns out that using small portions of the shuffled dataset, rather than all of the datapoints, we can achieve similar results. This is called stochastic gradient descent, and has many advantages over batch gradient descent, namely better computational efficiency, and better/faster convergence.

The concepts mentioned above are all central to how today's artifical intelligence systems work; implementations with typically large network architectures and large amounts of training data fall into the branch of Deep Learning, a highly influential family of machine learning methods that have achieved astounding performance in various fields, including speech recognition, natural language processing computer vision and many others.

2.3 Image classification and Convolutional Neural Networks

Not long ago, researchers throught that programming a computer to be able to differentiate between a dog and a cat is impossible. However nowadays, convolutional neural network architectures are capable of doing this with nearly perfect accuracy. Although the subject of this thesis is not image classification, most of the methods used to make image classification models learn better, are also largely present in object detection models.

2.3.1 Convolution

Convolution refers to 2 dimensional convolution using kernels, which are weights in a small grid that form a connection between an input grid of the same size as the kernel and one scalar output value. Applying a kernel to a grid requires repeatedly taking the weighted sum of a kernel sized portion of the input grid in a sliding window fashion. The computation is shown on figure 2.3.

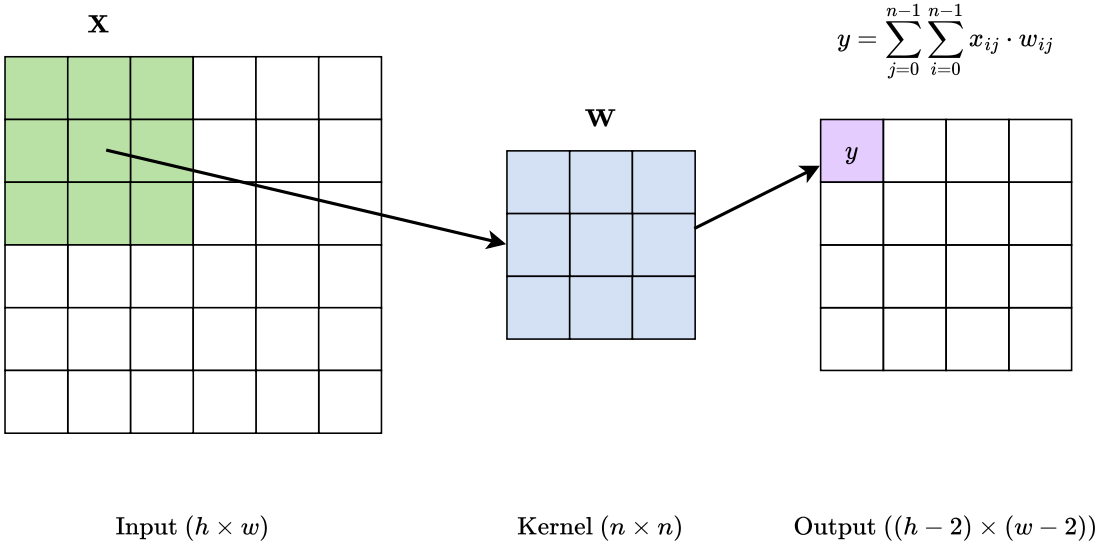

Figure 2.3: Illustration of the way one output grid cell's value is computed. On this diagram, y is the top left element of the output. The cell to the right of it is a similar weighted sum with the same kernel weights, but applied to a portion slid to the right by one grid cell.

A Convolutional layer of a neural network takes several channels as an input, (each channel being a grid), convolves them with separate kernels and adds up the result to get an output channel. This is shown on figure 2.4. A convolutional layer can have more than one output channel: in that case, ch_{out} set of kernels are used, where ch_{out} is the number of output channels. This is shown on figure 2.5

We call networks that are made up of or contain convolutional layers Convolutional Neural Networks, or CNNs. The kernels' of the conv layers' weights are what consitute the weights of a convolutional neural network.

2.3.2 Pooling

As we will see, convolutional layers extract "features" (important information) from an input image, but output feature maps of convolutional layers can be sensitive to the location of the features on the input image. A way to fix this issue is to downscale the image, reducing this sensitivity.

Two common ways to do this are max pooling and average pooling. As suggested by the name, max-pooling takes the maximum of values in portions of the image as shown on figure 2.6, while average pooling takes the average for each portion, as shown on 2.7

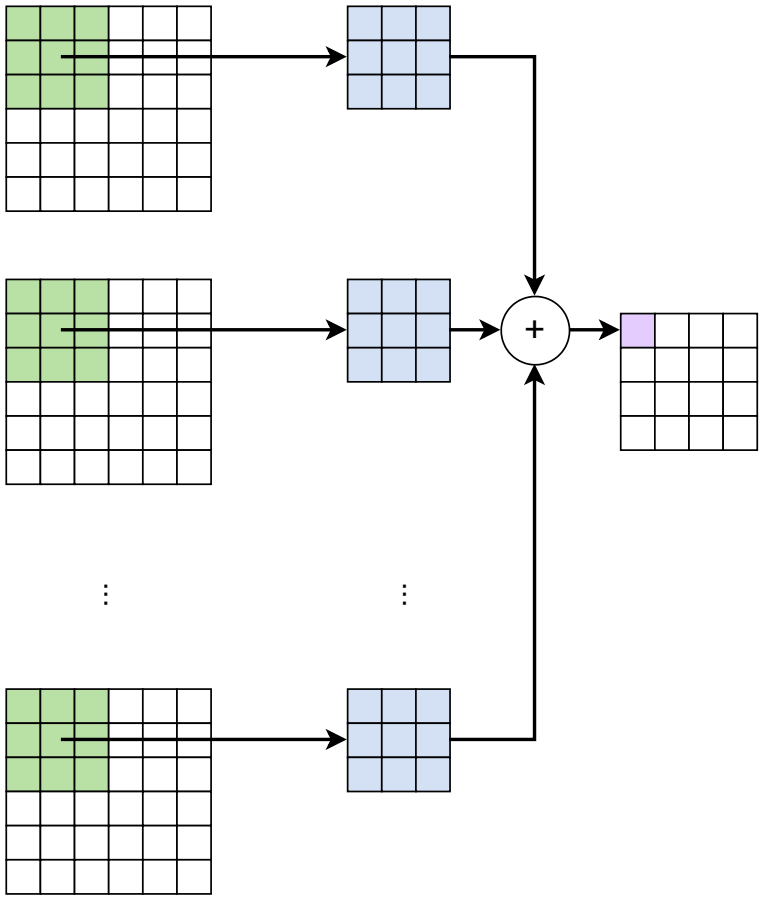

Figure 2.4: Illustration of the way one output channel is computed when passing through a convolutional layer

2.3.3 LeNet

The work of Yann Le Cun et al. on LeNet [13] focused on comparing different methods applied to handwritten character recognition. They have shown that convolutional neural networks outperform other techniques:

Convolutional layers of a network extract meaningful and low-dimensional features that we can work on by taking adavantage of the spatial correlations of the pixels on an image. On figure 2.8 it is shown that the features' (or feature maps') dimensionality decreases with depth. During training, each convolutional layer (operations that produce the feature maps) learns what information needs to be extracted, the abstraction level of which can vary from layer to layer i.e. first layers pick up on edges of a digit, later layers recognize patterns like loops. Similarly, when classifying an image of an animal, different levels of abstraction may be the shape of a horse or its edges vs the features in its nose/eyes/ears. On figure 2.8 the network's size pales in comparison to today's architectures used for deep learning in computer vision.

2.3.4 AlexNet

Convolutional neural nets' (CNNs) rise to fame was largely contributed to by AlexNet [11], as it popularized, *deeper* neural nets (more hidden layers). With machine learning models, the more data they learn from, the better their ability to generalize what they

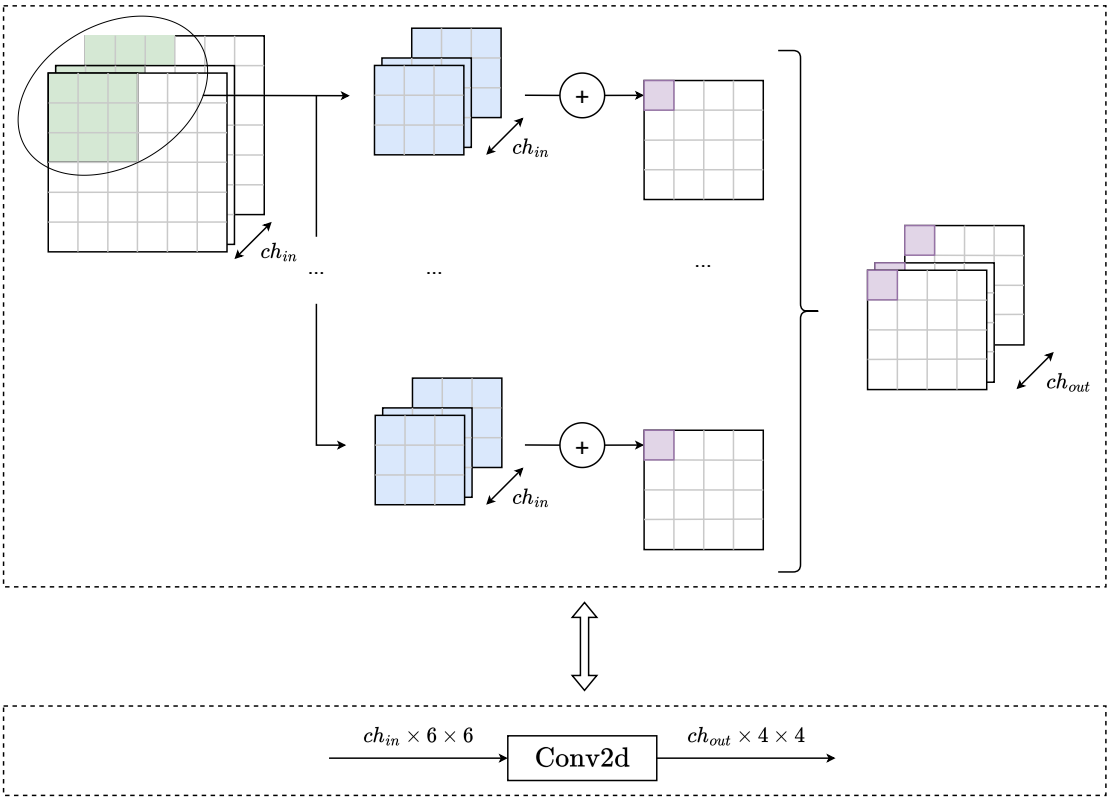

Figure 2.5: Illustration of what a convolutional layer with ch_{in} input channels and ch_{out} output channels does. The layer consists of ch_{out} sets of kernels and each set of kernels contains ch_{in} kernels. The way one output channel is computed is shown in figure 2.4

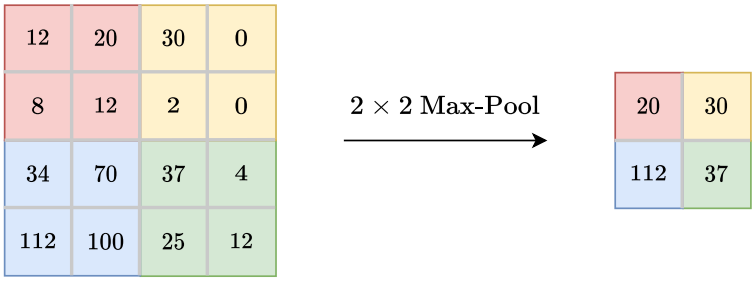

Figure 2.6: Illustration of max pooling

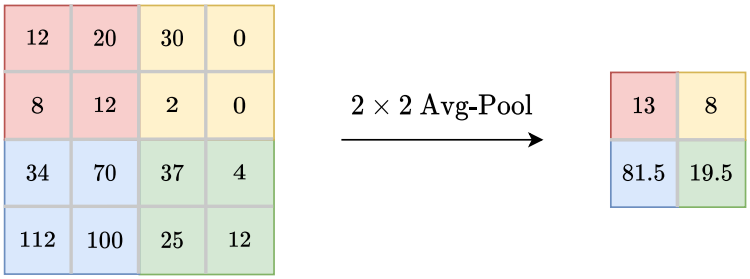

Figure 2.7: Illustration of average pooling

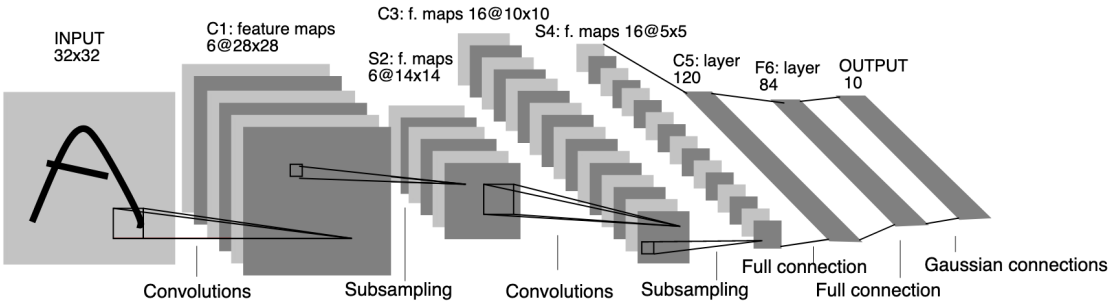

Figure 2.8: Architecture of LeNet-5, a CNN. Each plane is a feature map [13]

have learned, leading to increased performance. However Alex Krizhevsky et al. stated that the immense complexity of the image classification task could not be covered even by a dataset with millions of images over 1000 categories, therefore the model would "need lots of prior knowledge to compensate for all the data [they didn't] have" [11]. CNNs were exactly what they needed for this so using Convolutional layers, the proposed key components were the following:

- More depth in the architecture
- The use of ReLU as the activation function, which allows for the model to train faster compared to when using the sigmoid function.
- A regularization technique[2] called Dropout. From a representational point of view, this technique cuts out some neurons, forcing the model to learn more robust features.

The model won first place in the ILSVRC-2012 competition, beating the competition by a significant margin.

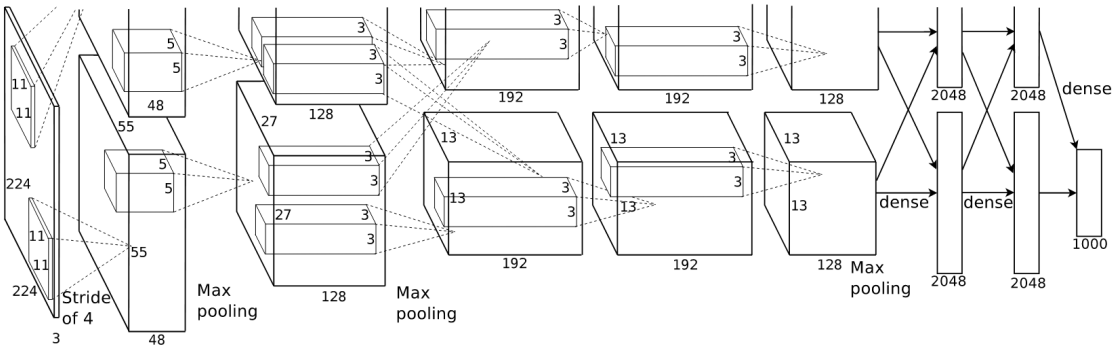

Figure 2.9: An illustration of the architecture used in AlexNet [11]

Several other big milestones can be mentioned after AlexNet. One of them is VGGNet, which explored the idea of make the network even deeper, using small, 3 by 3 convolutions. Another is GoogleNet, the success of which was powered by their *inception modules*. The

[2]If models are trained for too long, it can learn knowledge that is too specific to the training data, which results in a drop in performance on validation data. This is called overfitting and it can be mitigated by regularization

models mentioned above were introduced in the era when deeper networks resulted in better performance. However, pushing depth can't always be the solution and researchers came to realize that other options are needed. This is where ideas like ResNet [7] and DensetNet came into play.

2.4 Deeper neural networks with ResNet

Experiments with deeper and deeper networks have shown what is called the degradation problem: Increasing depth does indeed increase performance/accuracy, but this accuracy saturates at a point and if further layers are added, it degrades. The proposed solution to the problem is residual learning:

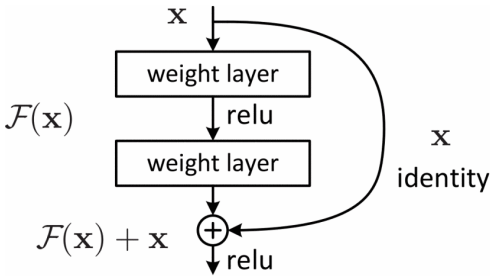

Figure 2.10: Residual learning: A building block [7]

He et al. state that it should be possible to construct a deeper network by adding layers to a model that already has good performance. In theory, if the added layers simply learn the identity function, they don't change the part of the network that learned to perform well. However, fitting a zero mapping by a stack of nonlinear layers is easier than fitting an identity mapping. Therefore, the idea is to include a "default identity mapping", i.e. a shortcut connection to the output; If the final desired mapping $\mathcal{F}(x) + x$ is close to the identity function, then the residual $\mathcal{F}(x)$ has to fit a mapping close to zero, which as mentioned before is an easier task. Compared to previous models, where depth was 19 layers, it was found that the deeper the model (up to 8x deeper), with the building blocks shown, training error decreased. These Deep residual networks won 1st places in the ILSVRC and COCO 2015 competitions and are still used to this day, not only for image classification, but also segmentation and object detection.

Another problem that Resnet can solve is the vanishing gradient problem (even though this wasn't the primary goal of the paper). Roughly speaking, the vanishing gradient occurs when elements of the gradient become very small, and due to the use of the chain rule, multiplying more and more small numbers between 0 and 1 together results in number close to zero, i.e. the gradient "vanishes". This multiplication of elements involves multiplying activations of neurons across layers, so the deeper the neural network, the more likely it is that this will occurs. With ResNets, if the activations at the output of the stack of non linear layers of a residual block were to become very small, the skip connections allow the original input to flow directly to the output, thereby helping avoid the problem.

2.5 Object Detection

A difficult, but important task in computer vision is that of object detection, which is the focus of this thesis. Instead of classifying an image into dog/cat and other categories, we need to draw bounding boxes around all objects in an image, while classifying each object into a category. While many traditional methods outside of the domain of machine learning exist, their performance is not nearly as good as today's solutions powered by deep learning. Object detection models fall into two categories: Two-stage and One-Stage detectors.

Two-stage object detection: As stated in the name, detection happens in two steps.

1. Get region proposals of the regions that likely contain an object (Regions of interest or ROIs).
2. Run image classification on the ROIs.

One-stage object detection: Detections are directly the output of the model, there is no intermediate step.

A naive approach to two-stage detectors is to use the sliding window method, only considering a small portion or region of the image at a time, then running a CNN on that small region (The CNN has been trained on closely cropped objects) to classify the region into an object class. The sliding window method may be repeated for several window sizes:

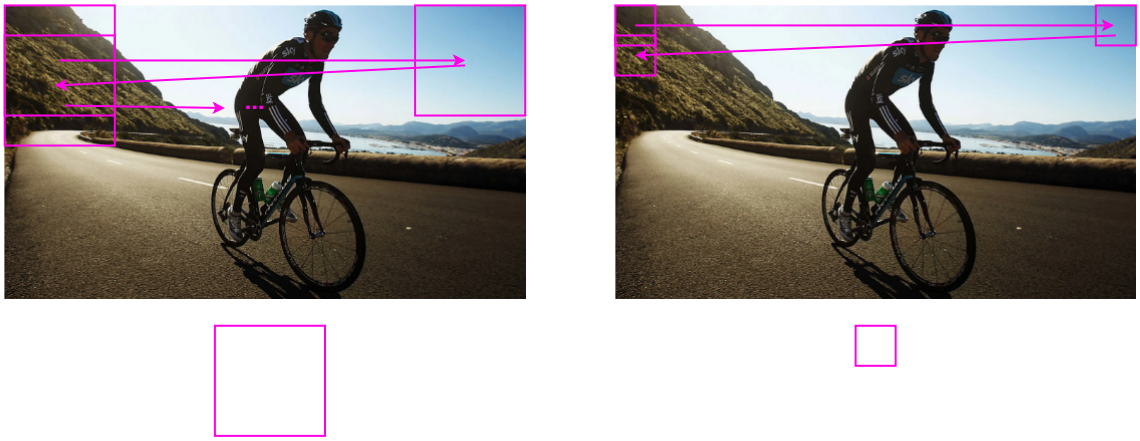

Figure 2.11: Illustration of the sliding window method

The method suffers from multiple drawbacks. The first glaring issue is the computational inefficiency: a CNN has to be run on every region for every single sliding window size. The stride by which the sliding window moves across the image can be increased, to reduce the amount of regions we need to run the CNN on, but this may cause the window to potentially "miss" some objects, thereby hurting performance. The second issue is the varying aspect ratio of objects, which could be remedied by an adequate selection of window sizes (not just squares).

2.5.1 RCNN

In contrast to the method above, RCNN [6] deviates from the naive sliding window approach, and instead obtains regions from the output of a segmentation algorithm. Seg-

mentation produces an image in which each pixel is assigned to a category, e.g. road, background, sky etc.

Figure 2.12: Illustration of a Segmented image

From the segmented image 2.12, it is possible to better select regions that may contain objects as shown on 2.13:

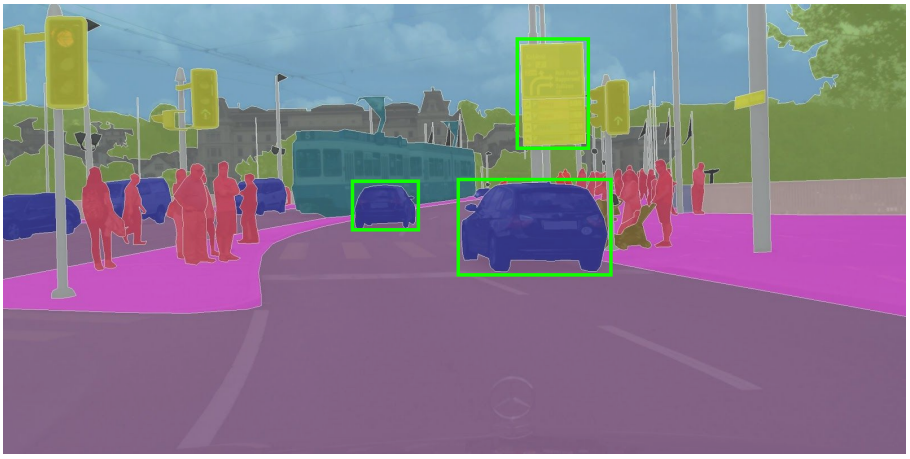

Figure 2.13: A few region proposals on a segmented image

In RCNN, around 2000 region proposals are obtained, and classification is run using CNN for feature extraction. This may get rid of unwanted regions to run a CNN on, but it still has a relatively high computational cost. Further resarch with this method was aimed at increased speed; the methods are called Fast RCNN [5] and Faster RCNN [20].

2.5.2 Overfeat

Figure 2.11 showed how the sliding window method works, and Sermanet et al. have proposed a method that implements it "convolutionally". With the slidig window method, for each window sized portion of the image, we get a classification output, which is a vector with length identical to the number of classes/categories. The way this would look in the classifier Network is a few fully connected layers at the end with the output being the classification vector. We can substitute these by conv layers with a kernel size of 1×1 and get indentically sized features. For a particular window, we know the amount

of classification outputs we need; it depends on the stride by which the window is moved across the image. Assuming the image is square, this is $n \times n$. The convolutional method that [21] proposes produces as many $n \times n$ output grids as there are object classes, which can be done with the conv layers of kernel size 1×1 mentioned previously. The drawback of this method is the poor accuracy of the position of bounding boxes.

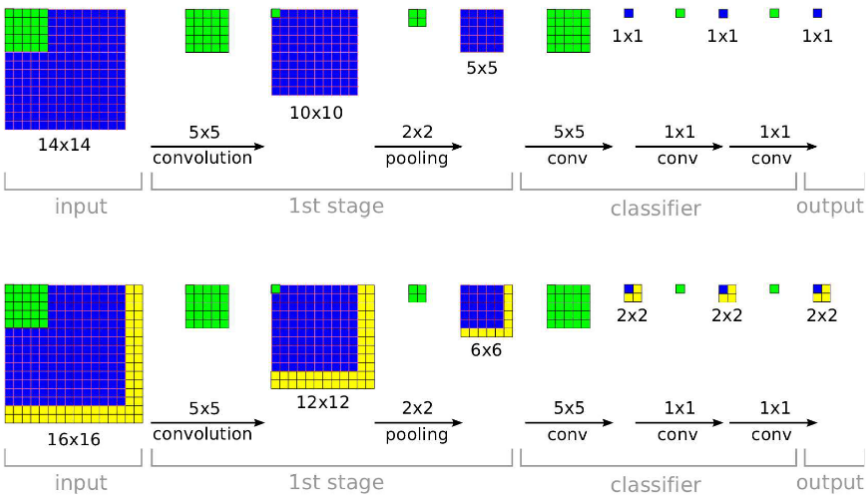

Figure 2.14: Illustration [21] of the convolutional implementation of the sliding windows method

2.5.3 YOLO

In 2015, Joseph Redmon et al. came up with a revolutionary new object detection algorithm called YOLO [19] (You Only Look Once), which unlike other methods of iteratively looking at proposals, only has to look at an image once (hence the name) to detect objects accurately. The key feature of YOLO is its speed, since among real time object detection algorithms it was the first of its kind. A small sized version of its neural network processed images at 155 frames per second, with twice the mean average precision of other real time systems.

The solution first divides the input into an $S \times S$ grid and each cell is responsible for predicting an object if the object's center falls into that cell. A cell's prediction consists of bounding boxes' x, y coordinates and w, h dimensions, prediction confidence, which in the paper is formally defined as $\Pr(\text{Object}) * \text{IOU}_{\text{pred}}^{\text{truth}}$ and conditional class probabilities, $\Pr(\text{Class}_i|\text{Object})$. The output is represented as a $S \times S \times (B * 5 + C)$ tensor. In the paper, they used $S = 7$ and $B = 2$, while training and evaluating on the 2007 PascalVOC dataset.

Their network is inspired by GoogleNet [23], has 24 convolutional and 2 fully connected layers:

The input is a 448×448 image and in the output, predictions of bounding boxes contain numbers bounded between 0 and 1: w and h are normalized by the image width and height and x and y are also normalized coordinates within the grid cell responsible for the box's prediction. There can be multiple bounding boxes predicted in each cell; this is done with

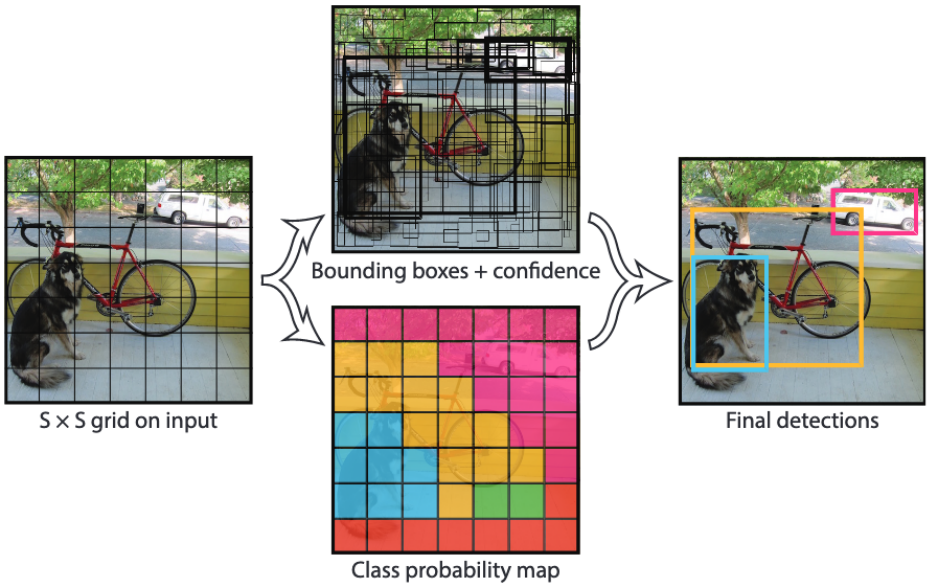

Figure 2.15: Illustration [19] of the outputs involved in detection with YOLO

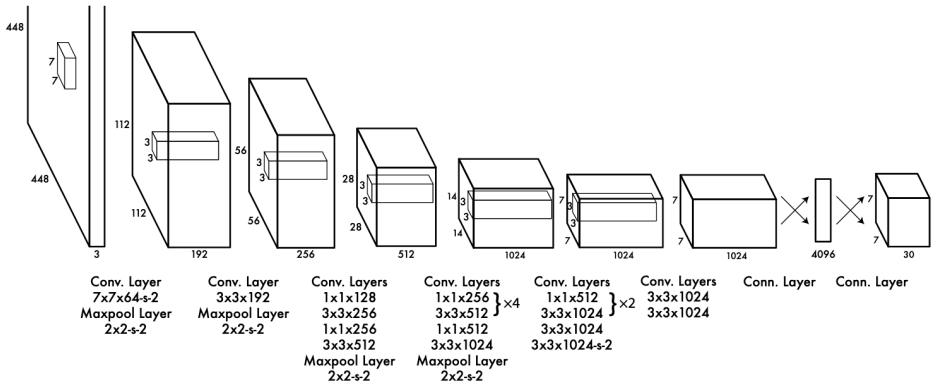

Figure 2.16: Illustration of the original YOLO network architecture [19]

the use of "predictors". A prediction is assigned to the anchor box that has the highest IoU with the ground truth box. The loss function to optimize is the following:

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] \\
&+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2] \\
&+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
\tag{2.1}
$$

The function is separated into 4 parts: The sum squared errors the following:

- x and y coordinates
- $\sqrt{w}$ and $\sqrt{h}$. Note, that square root is taken to attempt to balance the loss of smaller and larger boxes
- confidence values
- class probabilities

$\mathbb{1}_i^{obj}$ denotes if object appears in cell i and $\mathbb{1}_{ij}^{obj}$ denotes that the j^{th} bounding box predictor in cell i is "responsible" for that prediction. The reasoning behind using sum squared error loss is the ease of optimization, but the inherent flaw is that it does not directly aim to increase average precision. Another limitation is the need to explicitly balance the loss terms (λ_{coord}, λ_{noobj}). Typically, many grid cells do not contain objects, meaning that the confidence loss will dominate over other components like localization loss and push the cells' confidence to 0. To solve the issue, values of $\lambda_{coord} = 5$ and $\lambda_{noobj} = 0.5$ were used.

YOLO is fast and accurate but has some limitations: Strong constraints are put on bounding boxes, because only 2 boxes can be present in one grid cell, and only one class can be predicted for them. Unusual aspect ratios are not well handled by the network. Finally, they state that the loss function treats small and large box errors the same, which can decrease performance, as errors in large boxes do not affect IoU as much as the ones in small boxes.

2.5.4 YOLOv2

From the authors of the original YOLO paper [19], a new and improved YOLO method was introduced a year later and was entitled YOLO9000: Better, Faster, Stronger [17].

First, they made their networks *better* by

- Adding batch normalization [9], thereby adding stability to the network while training and reducing the need for various other regularization techniques: it allowed the authors to remove dropout without overfitting.
- A higher resolution classifier, pretraining it on double the original resolution, making the object detection network (whose input is this doubled resolution) perform better on the 448×448 resolution.
- Adding anchor boxes: Instead of having two predictors in one grid cell with bounding boxes of the same grid cell constrained to one class, prediction of class and objectness is done for every anchor box.
- Dimension clusters: Anchor boxes are cluster centroids of width and height. They describe a method in which they run k means clustering on box dimensions; with this method, instead of hand picked anchor boxes we have a better representation of the data, making the task easier to learn by some degree. The results of k means clustering can be seen on figure 2.17
- Multi scale training. While training the network, the image input size is changed allowing the network to learn more/better from images at different resolutions. The resolutions range from 320×320 to 608×608.

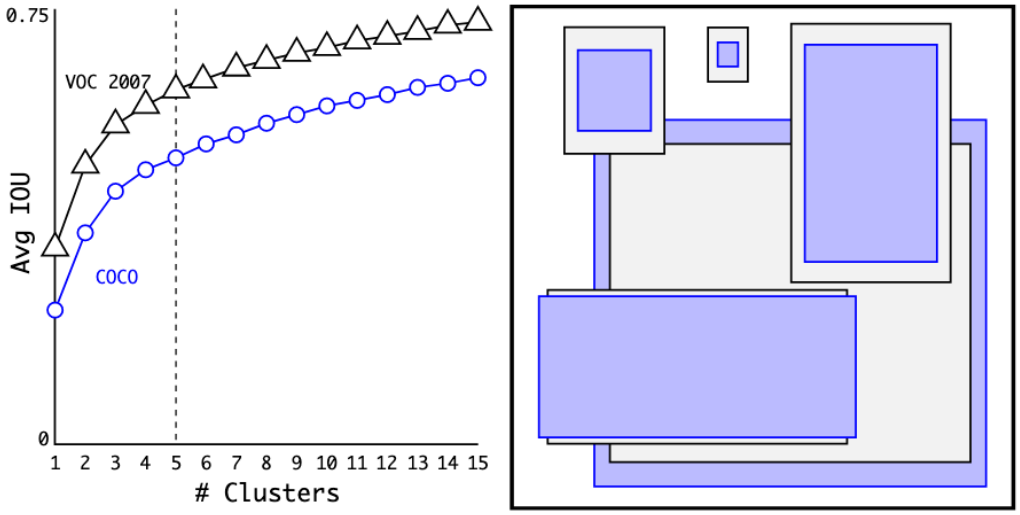

Figure 2.17: An illustration of the results of k means clustering. This figure is from [17]

The predictions of bounding boxes are made using the formula shown on figure 2.18

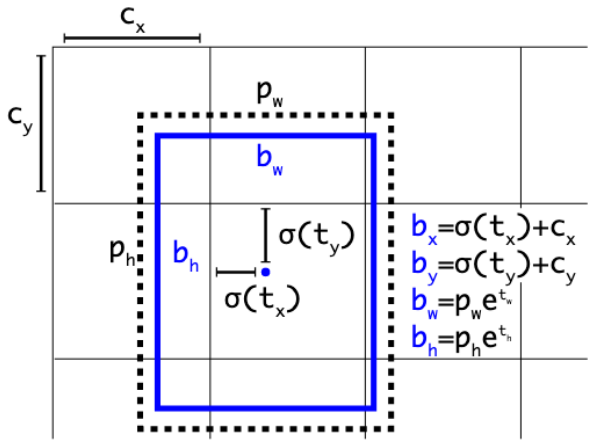

Figure 2.18: An illustration of how bounding box dimensions (b_x, b_y, b_w, b_h) are calculated from the network's outputs (t_x, t_y, t_w, t_h) [17]

They also made the method *faster* by Using Darknet 19, a new classification model, using 19 convolutional layers and 5 maxpooling layers. See figure 2.19.

2.5.5 YOLOv3

YOLOv3 [18] is an incremental improvement over YOLOv2, with the most relevant changes being:

- Loss function modifications: On figure 2.18, the equations by which bounding box coordinates and dimensions are computed contain the model's raw predictions

Type	Filters	Size/Stride	Output
Convolutional	32	3×3	224×224
Maxpool		$2 \times 2/2$	112×112
Convolutional	64	3×3	112×112
Maxpool		$2 \times 2/2$	56×56
Convolutional	128	3×3	56×56
Convolutional	64	1×1	56×56
Convolutional	128	3×3	56×56
Maxpool		$2 \times 2/2$	28×28
Convolutional	256	3×3	28×28
Convolutional	128	1×1	28×28
Convolutional	256	3×3	28×28
Maxpool		$2 \times 2/2$	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Convolutional	256	1×1	14×14
Convolutional	512	3×3	14×14
Maxpool		$2 \times 2/2$	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	512	1×1	7×7
Convolutional	1024	3×3	7×7
Convolutional	1000	1×1	7×7
Avgpool		Global	1000
Softmax			

Figure 2.19: The Darknet 19 model used in YOLOv2. This figure is from [17]

t_x, t_y, t_w, t_h. Sum squared error losses are computed for these values, while targets can be obtained by inverting the equations. Logistic regression is used for objectness.

- No softmax is used in class predictions. It was found that it is not necessary for good performance, so they used binary cross entropy loss, which helps with cases where an object can have overlapping classes (e.g. Woman and Person)

- Predictions across scales: Several convolutional layers were added to their base feature extractor, the last of which predicts a $N \times N \times (5 + n_c)$ where $N \times N$ denotes the size of the grid, and n_c the number of classes. A method of upsampling a previous layer's feature map and concatenating it to another one from earlier in the network is used, allowing for more meaningful semantic information and finer grained detail to be obtained. The network predicts three $N \times N \times (5 + n_c)$ tensors, corresponding to small, medium and large scaled object detection outputs (the grid is finer for small predictions and coarser for large ones).

- New feature extractor: The Darknet 53 architecture is a large improvement over its predecessor, Darknet 19. Inspired by ResNets [7], helped make the neural network deeper and perform better.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 2.20: The Darknet 53 model used in YOLOv3. This figure is from [18]

2.5.6 YOLOv4

The novelties contained in YOLOv4 mostly originate from experimentation with the large amounts of new features from modern object detection research, that are said to improve the perfomance of CNNs. Some of the following are combined (while ran on different feature extractors): Weighted-Residual-Connections [22] (WRC), Cross-Stage-Partial-connections [25] (CSP), Cross mini-Batch Normalization [26] (CmBN), Self-adversarial-training (SAT) and Mish-activation [15], Mosaic data augmentation, DropBlock regularization [4], and CIoU loss [27].

2.5.7 CenterNet

While YOLO is an anchor-based [3] method, other anchor-free methods were proposed in the recent years with similar performance: CornerNet [12] and CenterNet [28]. The CenterNet method proposes to predict object center heatmaps, offsets and object size.

Given an input image $I \in R^{W \times H \times 3}$, a keypoint heatmap $\hat{Y}_{x,y,c} \in [0,1]^{\frac{W}{4} \times \frac{H}{4} \times C}$ is produced at a lower resolution (dividing width and height by 4) where for a given category, the heatmap at coordinates (x, y) is 1 a detected keypoint and 0 is background. In the target, for each ground truth keypoint, the heatmap contains a peak at the keypoint's coordinate (at lower resolution) and around it are values that follow a 2 dimensional gaus-

[3] Anchor boxes are a set of predefined bounding boxes of a certain height and width. Object detection methods that make use of bounding boxes to identify which aspect ratio needs to best be fitted by an object is characterized by the term "anchor-based method"

sian distribution, with an object size adaptive standard deviation. Since the heatmap is at a lower resolution than the input image, a correction is made by predicting offsets (same as offsets within grid cells). The loss regarding keypoints is a penalty reduced pixel-wise logistic regression with focal loss, and the offset is trained with an L1 loss.

Using this ideas mentioned above, the Objects by Points method predicts 2D bounding boxes around objects the following way:

- prediction of the center point of objects: $\hat{Y}_{xyc} \in [0,1]^{\frac{W}{4} \times \frac{H}{4} \times C}$. This is trained using pixel-wise logistic regression with focal loss.
- Prediction of local offsets: $\hat{O} \in R^{\frac{W}{4} \times \frac{H}{4} \times 2}$. Scale is not normalized raw pixel coordinates are used directly. This is trained with L1 loss
- The size of objects: $\hat{S} \in R^{\frac{W}{4} \times \frac{H}{4} \times 2}$ Note: they use one size prediction per category to save computational burden. This is also trained with L1 loss.

2.5.8 Non maximum suppression

Non maximum suppression is a post processing step in object detectors that removes redundant detections:

Figure 2.21: An illustration of non max suppression

2.5.9 Comparison between YOLO and CenterNet

YOLOv5 [10] is a family of object detection models that incorporate new ideas with modern research, but no scientific paper has been published yet, due to the ongoing nature of the project. For this thesis, I only considered one-stage detectors, namely YOLO (all versions) and CenterNet, so I have decided to collect the speed and performance of both methods in table 2.1

The table shows that the YOLO method provides greater detection speed as well as higher mAP on the COCO dataset. More resources are also available on the internet that explain YOLO in depth, and many implementations are avilable on open source platforms. However, CenterNet is an anchor free method that is much simpler. Its downside may be reduced speed and performance compared to YOLO, but implementation might be more straightforward.

Model	mAP COCO	inference time (ms)	FPS
YOLOv5n (nano)	45.7	**6.3**	**158**
YOLOv5s (small)	56.8	6.4	156
YOLOv5m (medium)	64.1	8.2	121
YOLOv5l (large)	67.3	10.1	99
YOLOv5x (x-large)	**68.9**	12.1	82
CenterNet-DLA	39.2/41.6	35.7	28
CenterNet-HG	42.1/45.1	128.2	7.8

Table 2.1: A comparison of YOLO models with CenterNet models. mAP in CenterNet models is given as single-scale / multi-scale testing (YOLO mAP numbers are single-scale tested results)

Chapter 3

Specification and Design

3.1 Specification

Typically, multi-task learning involves a network learning two related tasks at once, which in most cases requires separate data for each task. However, in this thesis, we get around this requirement, by formulating multi-task learning as follows: A regular method (like YOLO or CenterNet) needs to output bounding boxes with class and over separate subgroups of object classes that are in the dataset. For example, consider a dataset that contains dogs, cats, fish, bears, foxes, giraffes. Dividing these classes into subgroups, we could get dogs and cats in one group, fish and bears in another and foxes and giraffes in the last. Since it appears that Multi task learning improves the ablity of a network to generalize, the objective is to show wether the same can be said for this specific case/form of multi-task learning.

Using datasets that contain object detection data, and don't have a large amount of classes, the job is to train and compare the performance of regular object detection (single task OD) to several multi-task variants, yielding the following three cases to be studied:

- All classes together (single task OD)
- Subgroups of classes (multitask OD 1)
- One subgroup for each class (multitask OD 2)

3.2 Design

The chosen method is YOLO, because I had some experience with the algorithm from the previous semester and because compared to CenterNet, YOLO's speed and performance is superior. My decision was also influenced by the face that there are many implementations of the various versions of YOLO, which could help me experiment with different techniques.

Before going into details on the implmentation of multitask object detection, a thourough explanation of the "single task" object detection must be given.

3.2.1 Architecture

The network architecture is taken from the YOLOv5 github repository [10] and corresponds to their "small" sized network. Their architectures are similar when it comes to

the building blocks. By changing depth (number of layers) and width (feature size), they provide 5 different model sizes each with different speeds and highest mAP as shown on table 2.1. Depth is changed by changing the parameter N, which appears on figures 3.1 and 3.2 and determines the amount of successive bottleneck blocks the C3 module contains. Another parameter that changes in different YOLOv5 architectures is the number of features across the network. If the first ConvBNSiLU block's output channel number is doubled, so is that of the next ConvBNSiLU, as well as the following C3 block's hidden channel number and output channel number etc. In YOLOv5, these parameter changes were inspired by [24] and were motivated by the improvement of efficiency and speed, thereby making their models more accessible to the public.

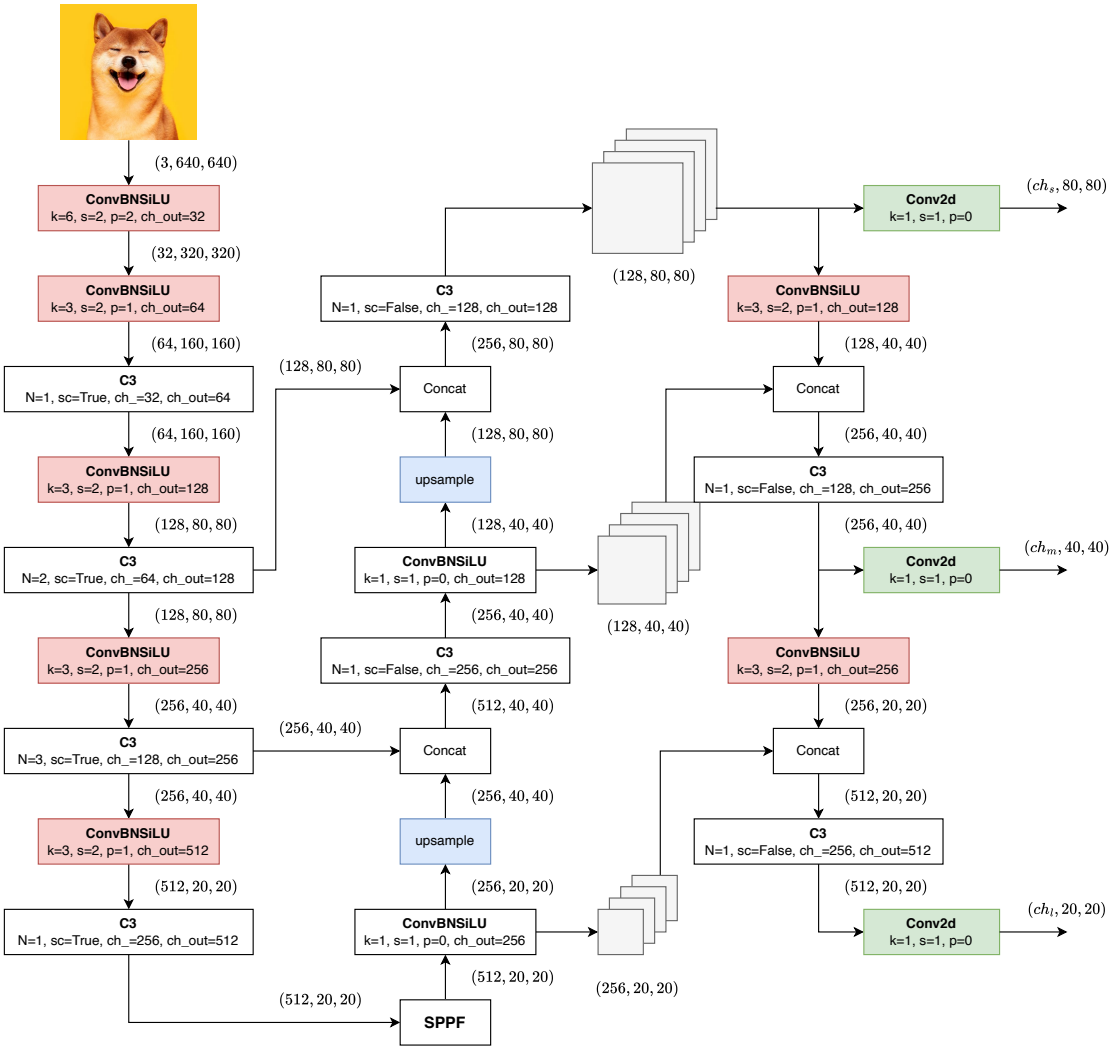

Figure 3.1: Architecture of YOLOv5s. Figure inspired by the Architecture summary in [10]. The ConvBNSiLU block is comprised of a convolutional layer, followed by a BatchNorm, followed by a SiLU activation function. Each ConvBNSiLU block is annotated with its convolutional layer's parameters: Kernel size (k), stride (s), padding (p) and the number of its output channels (ch_{out}).

A building block that appears frequently in the network is the C3 block, which is a slightly modified Cross Stage Partial connection or CSP block [25] (used in YOLOv4 and onwards). Cross stage partial connections allow for more lightweight models to perform better compared to other deep neural networks' "lightweight" versions. They provide richer gradient combination while reducing the amount of computation, by partitioning feature maps of a base layer into two parts and then merging them in a cross staged hierarchy.

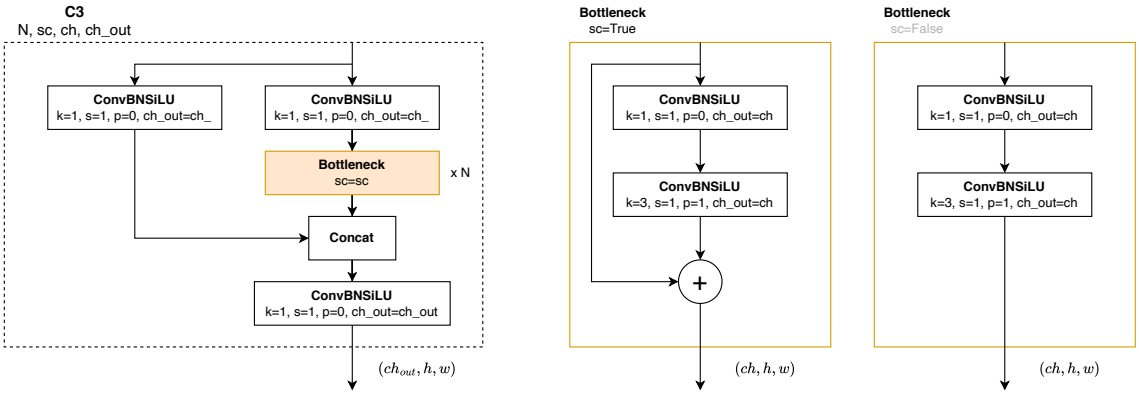

Figure 3.2: The C3 module, a building block of YOLOv5 networks

Another custom block that Glenn Jocher, owner of the YOLOv5 github repository has implemented is a modified spatial pyramid pooling (SPP) block. It is claimed that the new SPPF block shown on figure 3.3 is up to twice as fast while being mathematically identical to SPP, which is also detailed in the Architecture summary section of [10]. Instead of having max pooling layers with kernel size 5, 9 and 13 in parallel, we have kernel sizes 5, 5, and 5 in series. The reduction in FLOPs [1] originates from the fact that the maxpooling layers have to compute output on already downscaled inputs.

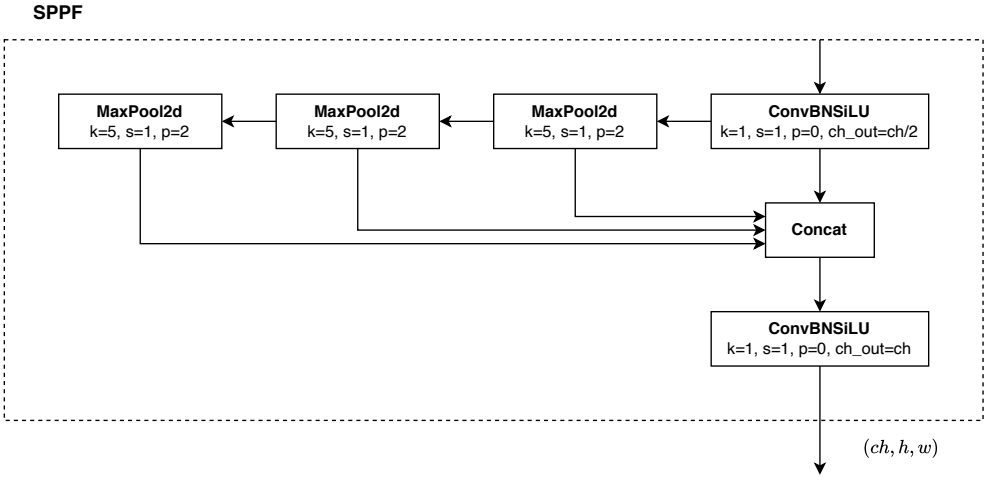

Figure 3.3: Illustration of the SPPF block, used in YOLOv5

The framework I have chosen to implement the model is PyTorch, because it is well documented and has a large community that I have found helpful on multiple occasions.

In figure 3.1, the green Convolutional layers in the last column of network blocks correspond to PyTorch's Conv2d module, which implements what is described in section 2.3.1.

[1] Floating point operations per second

Three of these blocks are present at the end of the network, each of which are responsible for producing detection outputs at a particular scale. The amount of grid cells in such a prediction output puts a spatial constraint on objects i.e. how close they can be to each other, therefore, the finer the grid (more grid cells), the smaller the detected objects are and the converse is true for a coarser grid (less grid cells). Accordingly, the detection outputs' dimensions are $(ch_s, 80, 80)$ for small, $(ch_m, 40, 40)$ for medium and $(ch_l, 20, 20)$ for large detections as shown on figure 3.1.

3.2.2 Detections

As mentioned previously, the last Convolutional layers produce the detection output. This output allows for the computation of bounding boxes, condfidence and object class.

YOLO, as an anchor based method, predicts bounding boxes with a confidence score and classification for each grid cell *and* for each anchor on a given scale. The predictions are computed from the network outputs, $t_x, t_y, t_w, t_h, t_{conf} \in \mathbb{R}$ and $t_{cls} \in \mathbb{R}^{n_c}$ (a one hot vector for classification) with the following equations:

$$\begin{aligned} b_x &= \sigma(t_x) + c_x \\ b_y &= \sigma(t_y) + c_y \\ b_w &= p_w e^{t_w} \\ b_h &= p_h e^{t_h} \\ b_{conf} &= \sigma(t_{conf}) \\ b_{cls} &= \arg\min_i(\sigma(t_{cls,i})) \quad \forall i \in n_c \end{aligned} \tag{3.1}$$

σ is the sigmoid or logistic activation function, c_x and c_y are grid coordinate offsets, p_w and p_h are prior (or anchor box) width and height values and n_c is the number of object classes.

3.2.3 Loss function

3.2.3.1 Assignment of a ground truth object to an anchor

Let's say we have 9 anchors in total for a dataset, with each of the anchors given as 2 element arrays encoding width and height [w, h]: [10, 13], [16, 30], [33, 23], [30,61], [62,45], [59, 119], [116, 90], [156, 198], [373, 326] (These anchors are just examples, they need to be computed for a particular dataset). We will assign each anchor to a scale by anchor box area, more specifically, since we have 9 anchors and 3 scales, each scale gets 3 anchor boxes. Let us consider the small object scale; the three anchor boxes we are working with are [10,13], [16, 30], [33,23] and a small ground truth object needs an anchor assigned to it. The metric we use to find the best fit is Intersection over Union (see figure 3.4)

The ground truth object is assigned to the anchor with which it has the best IoU as shown on figure 3.5

In the context of the two other scales, medium and large, the ground truth object in this example is not assigned to any anchor (because it does not fit medium or large anchors). The way this is implemented is by taking the IoU of a ground truth box with all 9 anchors, and only performing the assignment on the appropriate scale.

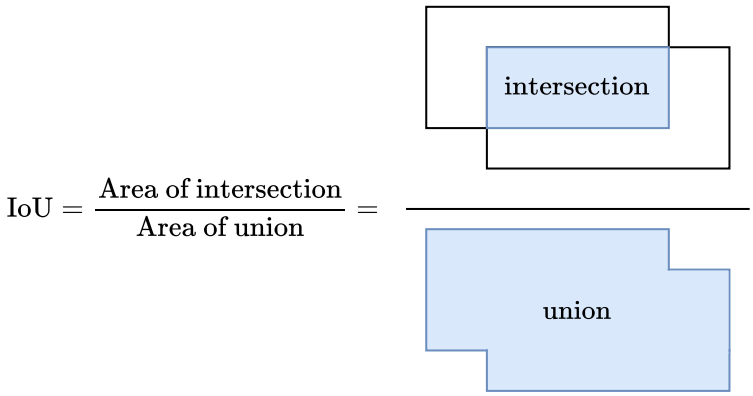

Figure 3.4: Illustration of IoU metric

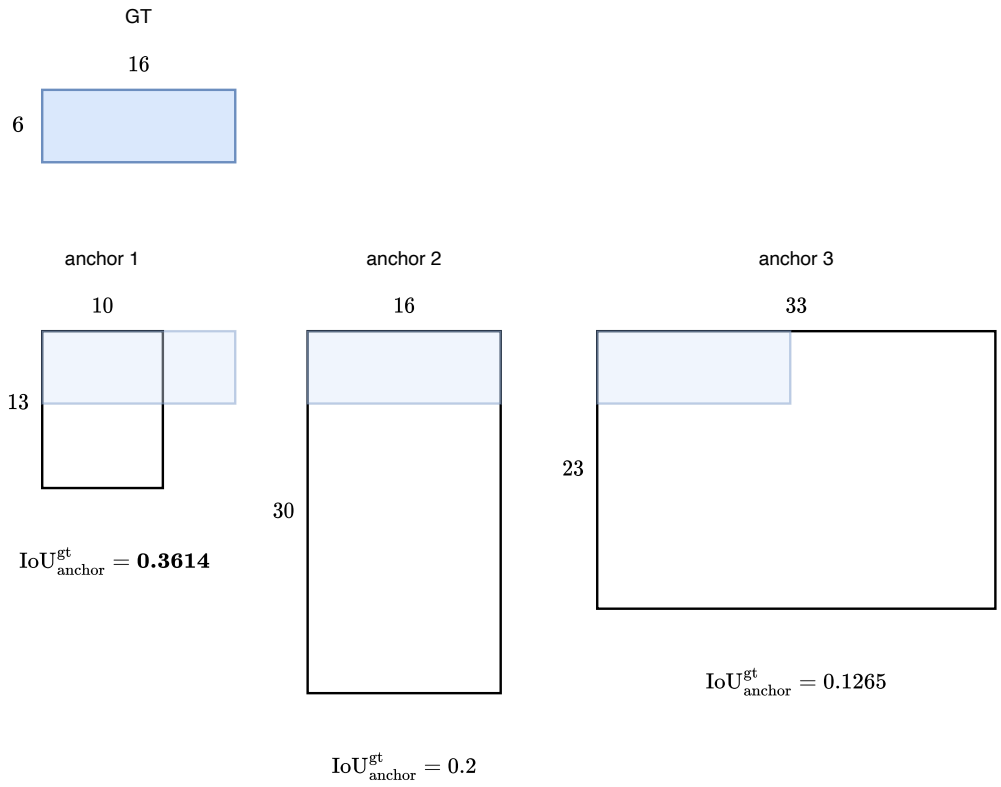

Figure 3.5: Illustration of how a ground truth object is assigned to an anchor. In this example, it is assigned to the firs anchor, because that is where the IoU is the highest.

3.2.3.2 Masking

Now that ground truth to anchor assignment has been explained, the loss function can be discussed. Remembering the loss function used in the original YOLO paper, detailed in subsection 2.5.3 and given by equation 2.5.3, $\mathbb{1}_{ij}^{obj}$ denotes that the j^{th} bounding box predictor in cell i is "responsible" for that prediction. Let this term be called the object mask, with the j-th predictor referring to the j^{th} anchor in our case (saying that the j^{th} anchor is "responsible" for a prediction means that the ground truth corresponding to the prediction is assigned to the j^{th} anchor). The i^{th} cell still refers to a particular grid cell, which can be determined by calculating a ground truth object's x and y grid coordinates.

If we still consider the small object detector at the end of the network, assuming the same 9 anchors as before, its output is a tensor of shape $(3 \cdot (5 + n_c), 80, 80)$. In the loss function, before doing any computation, this tensor is reshaped to $(3, 5 + n_c, 80, 80)$ as shown on figure 3.6, allowing us to apply the object mask to the elements that we wish to compute the loss on. To summarize, the mask is a tensor of shape $(3, n_g, n_g)$ that is only 1 for elemets at all (a, i_x, i_y) indices where a ground truth object has been assigned to anchor a and grid cell (i_x, i_y). All other elements of the mask are 0.

Before getting to the loss function, it is important to note that the total number of anchors does not have to be 9, meaning that at any given scale, the number of anchors is not always 3.

3.2.4 Loss

Definition 1. Let $p \in \mathbb{R}^n$ be a prediction, and $y \in \mathbb{R}^n$ a target. The mean squared error loss is the following:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (p_i - y_i)^2$$

Definition 2. Let $p \in \mathbb{R}^n$ be a prediction, and $y \in \mathbb{R}^n$ a target. The binary cross entropy loss is the following:

$$\text{BCE} = \frac{1}{n} \sum_{i=1}^{n} y_i \log(p_i) + (1 - y_i) \log(1 - p_i)$$

The loss function is the weighted sum of the following four components:

- MSE_{xy}: Mean squared error loss of object masked x and y predictions, where the predictions are $\sigma(t_x)$ and $\sigma(t_y)$, corresponding to the terms that appear in equation 3.2.2
- MSE_{wh}: Mean squared error loss of object masked w and h predictions, where predictions are the network outputs t_y, t_w.
- BCE_{conf}: The weighted mean of
  - the Binary cross entropy loss of object masked confidence predictions (weight $= \lambda_{obj} = 1$)
  - the Binary cross entropy loss of "no object" masked (the opposite of the object mask) confidence predictions (weight $= \lambda_{obj} = 2$)

  where the predictions are $\sigma(t_{conf})$
- BCE_{cls}: Binary cross entropy loss of object masked classification predictions, where the predictions are the elements of $\sigma(t_{cls})$

To summarize, with $\mathcal{L}_{scale}$ denoting the total loss of a detection output at a particular scale

$$\mathcal{L}_{scale} = \text{MSE}_{xy} + \text{MSE}_{wh} + \text{BCE}_{conf} + \text{BCE}_{cls}$$

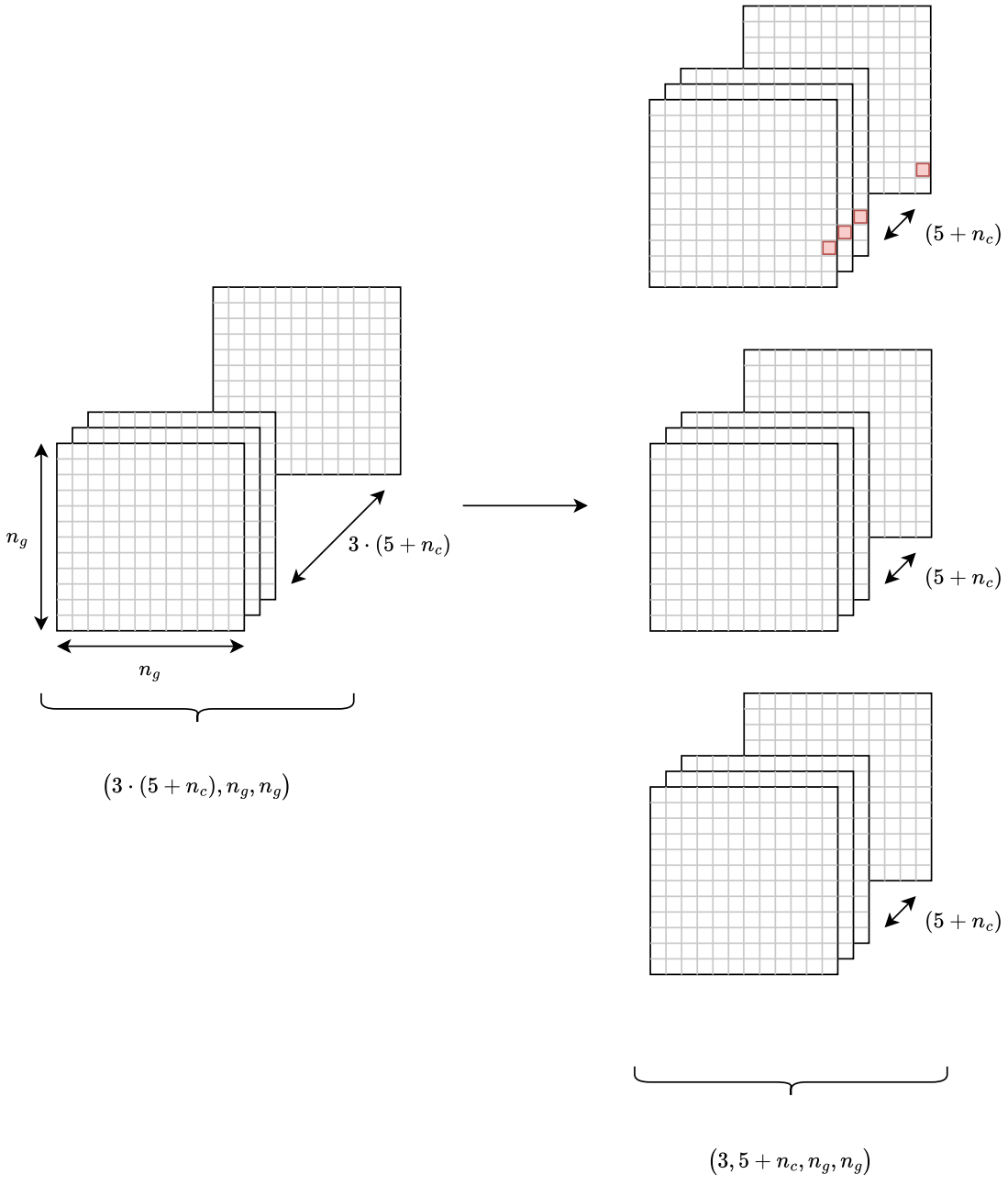

Figure 3.6: Illustration of the original network output (left) and the output separated by anchor (right), assuming 3 anchors. If a ground truth object has been assigned to the first anchor and the grid cell 2 cells above the bottom left corner, the cells highlighted in red are the $5 + n_c$ output numbers needed to compute the bounding box, condidence and class of the object.

Since we have three prediction scales, the total loss is

$$\mathcal{L} = \mathcal{L}_s + \mathcal{L}_m + \mathcal{L}_l$$

3.2.5 Multitask architecture

The single task architecture is shown on figure 3.1, but this can be simplified, before we consider its multitask form.

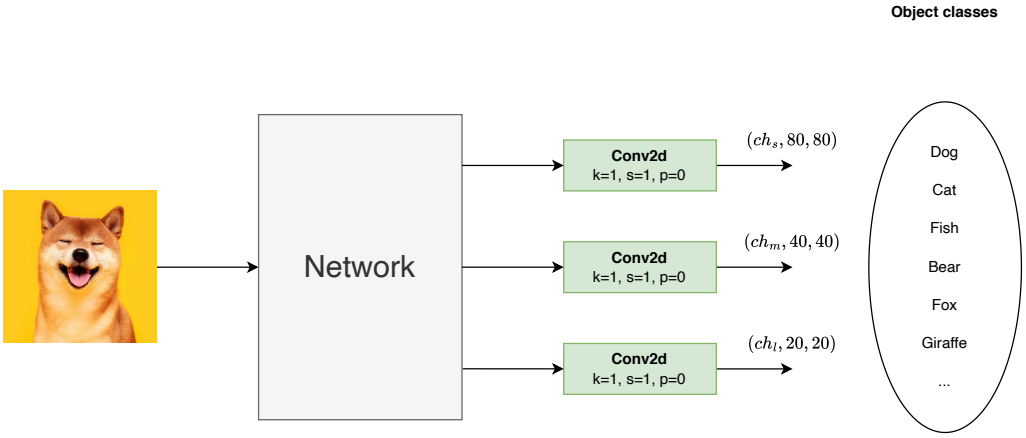

Figure 3.7: Simlified illustration of the base, or single task YOLO architecture. Object detection is performed on all classes of a dataset.

Since the tasks are divided into object detection over different subgroups of classes, the multi task network has detectors for each class group we define. The resulting architecture is depicted on the figure below:

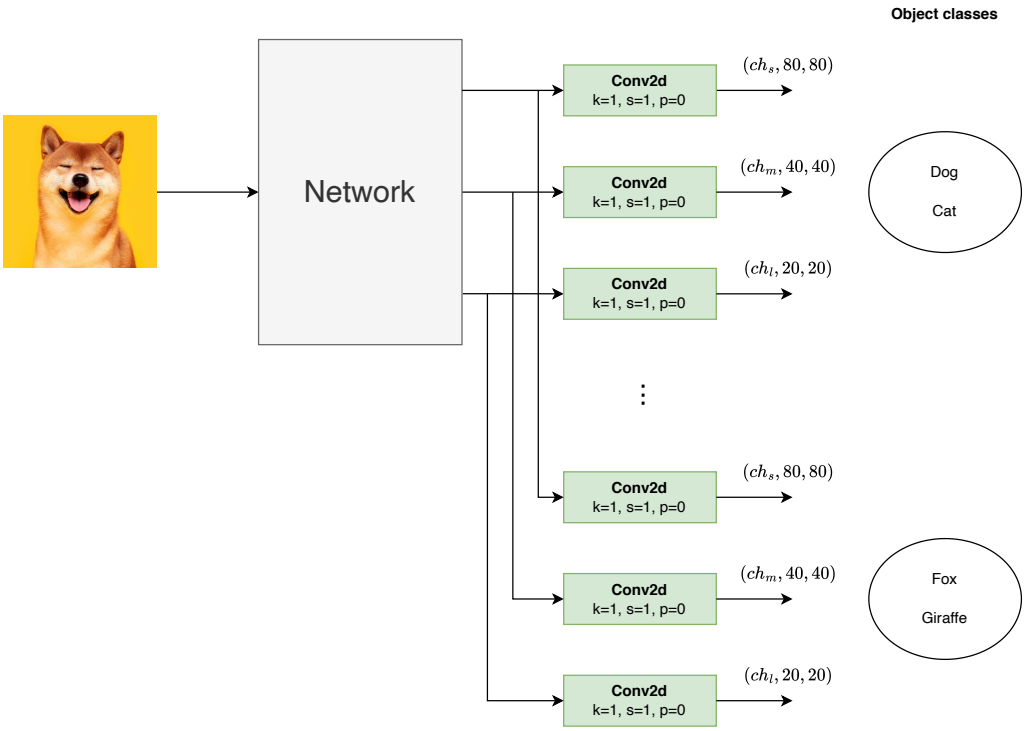

Figure 3.8: Illustration of the multi task YOLO architecture. Object classes of the dataset are divided into subgroups and detection is performed on subgroups separately.

The loss function in this case is the sum of losses for each task:

$$\mathcal{L}_{total} = \sum_{g \in groups} \mathcal{L}_g$$

3.2.6 Network input

Images of a dataset are not always square. The method I have chosen to remedy this issue is to add top and bottom or right and left padding to the image depending on wether width is greater than height or vice versa:

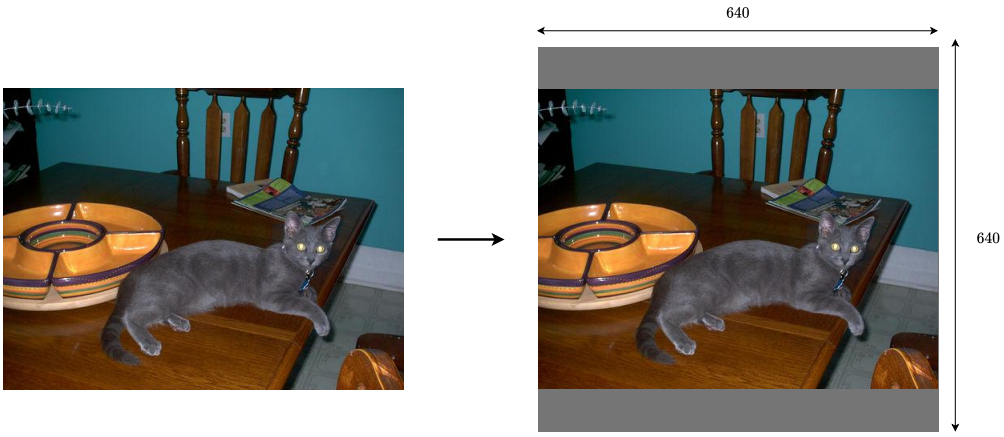

Figure 3.9: Padding an image to make it a square. *Note: on this figure, we assume that the image on the left has a width of 640, but images have different resolutions, so the padding process described above must be followed by a resizing step.*

To be able to calculate loss, we need the ground truth (or label) bounding boxes to be adjusted to the square padded image. The format of all labels are in the so called YOLO format, which is (c_x, c_y, w, h), where (c_x, c_y) are the coordinates of the center of the box and (w, h) are the width and height of the box. c_x, c_y, w, and h are all expressed relative to the width and height of the image (true pixel coordinates divided by true pixel dimensions of the image). The adjustment involves changing the x coordinate and the width, w of the bounding box or it's y coordinate and h, depending on wether the width of the image is greater than its height or vice versa. For example, in figure 3.9 c_y and h need to be changed for label bounding boxes.

3.2.7 Anchor boxes

What we call anchor boxes were called cluster centroids in YOLOv2 [17]. This is because the authors of the paper decided to collect and record the width and height of all objects that occur in the entire dataset, and run k-means clustering on these data points with the following distance metric:

$$d(\text{box}, \text{centroid}) = 1 - \text{IoU}(\text{box}, \text{centroid})$$

The reason for this is that if we were to use euclidean distance, larger boxes would generate more error than smaller ones. In addition, ground truth and anchor box matching is done based on the IoU metric, not euclidean distance, so it is only natural to create clusters of boxes that have high average IoU.

Considering that with multi task object detection, groups of object classes are treated separately, I have decided to do the same in the anchor selection process. I ran k-means clustering on objects belonging to different groups separately.

For one class group and 6 different k values ($k = 3, 4, 5, 6, 7, 8, 9$), I repeated the clustering 5 times for each value of k, measuring the mean IoU and recording the cluster centroids. The mean IoU for a given choice of k is defined as

$$\frac{1}{k}\sum_{i=1}^{k}\left(\frac{1}{n_i}\sum_{j=1}^{n_i}\text{IoU}(\text{box}_j, \text{centroid}_i)\right)$$

n_i is the number of bounding boxes in the i^{th} cluster and centroid_i is the centroid of the i^{th} cluster. box_j is the j^{th} bounding box label among the boxes that belong to the i^{th} cluster.

As a result, we have 5 different (mean IoU, centroids) pairs for each value of k. The final step is taking the pair with the highest mean IoU of the five and leaving the rest out. This allows us to create a plot similar to the shown in figure 2.17. One of the plots for a subgroup of a dataset that I used is shown below:

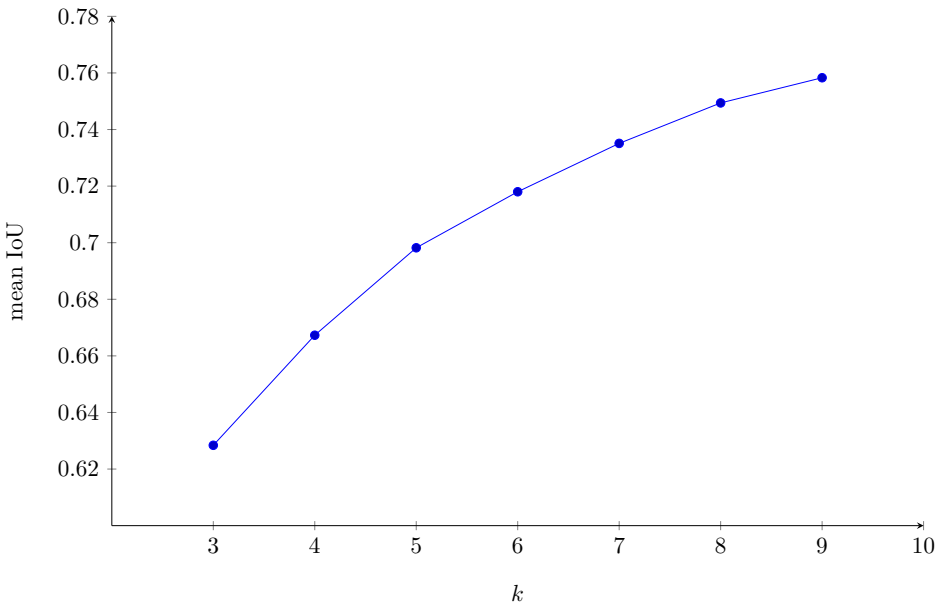

Figure 3.10: Mean IoU vs k for the person class group in the KITTI dataset (see results section for more details)

Selecting the value of k (determining the number of clusters) is done with the elbow method, that is, selecting the elbow of the curve. A clear example can be seen on figure

2.17. On figure 3.10, 5 can be a good selection, but so can 8 or 9 since if I could have run clustering for higher values of k like shown on 2.17, a "clearer" elbow would have presented itself. However, running clustering for higher values of k would have required too much computation and time.

3.2.8 Data Augmentation

In cases when there is not enough training data for a model to perform well on a validation dataset, we can artificially create more data by randomly modifying images and labels. These modifications are called data augmentation, which can prevent models from overfitting. The augmentation techniques I have used are simple, yet effective:

The first is horizontal flipping:

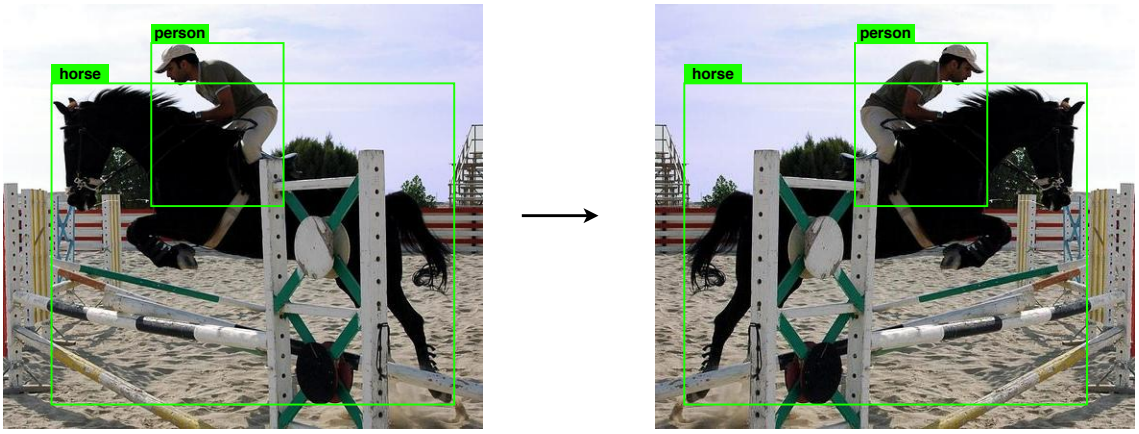

Figure 3.11: Illustration of Horizontal flipping. Labels change with this augmentation method.

The second is Gaussian blur:

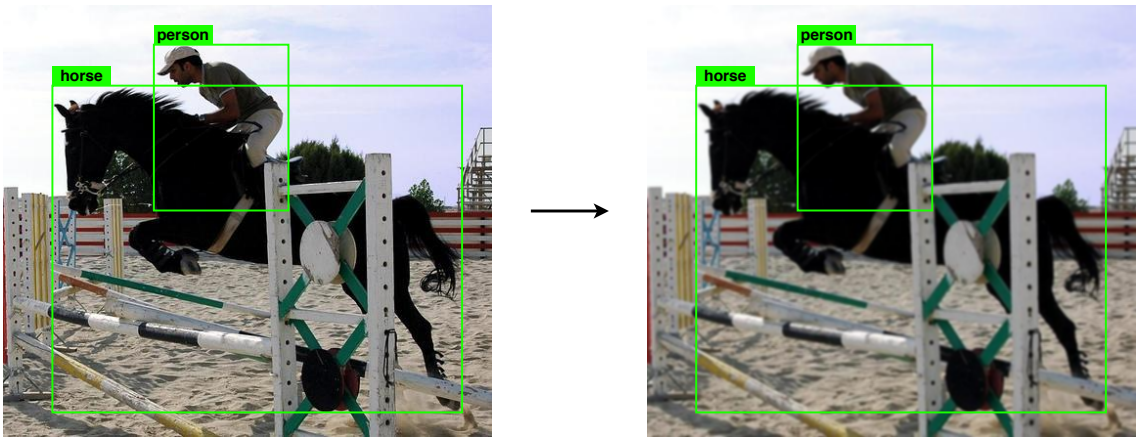

Figure 3.12: Illustration of Gaussian blur with radius 2. Labels do not change with this augmentation method.

The third is Grayscale:

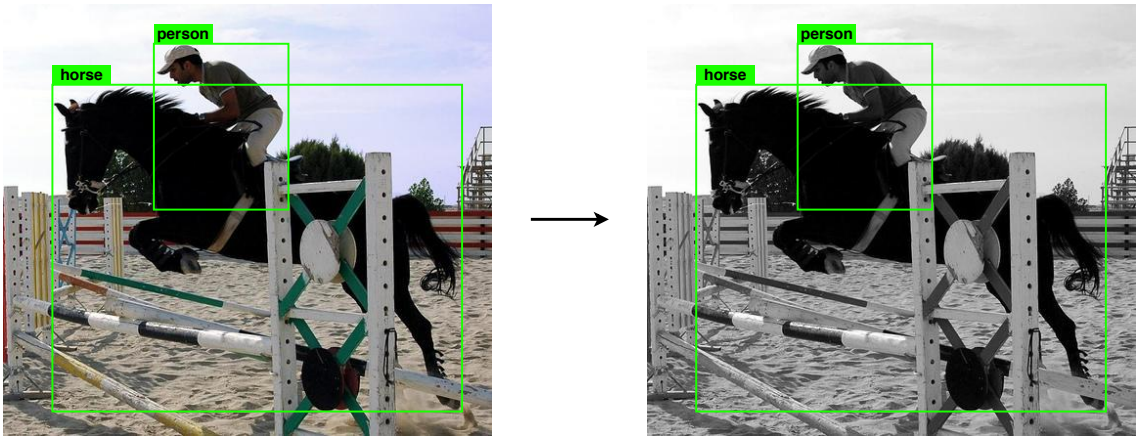

Figure 3.13: Illustration of Grayscale with radius 2. Labels do not change with this augmentation method.

All augmentations are applied randomly with probability p to images during training.

Chapter 4

Experimental results

As mentioned in the specification section (3.1), I set out to train and study 3 different cases:

- All classes together (single task OD)
- Subgroups of classes (multitask OD 1)
- One subgroup for each class (multitask OD 2)

4.1 Datasets

Two different datasets were used: PascalVOC [2] and KITTI [3].

4.1.1 PascalVOC

A standard benchmark in object detection, its latest 2012 challenge dataset [1] contains 5717 training images and 5823 validation images of various sizes. Images and objects are nearly equally distributed by class across the training and validation sets, see table 4.1.

For this dataset, the class groupings used in Multitask OD 1 are the following:

Multitask learning has been characterized as a form of inductive transfer, which can help introduce an inductive bias to the model. Inductive bias is the collection of assumptions that a model makes when seeing new input; the better these assumptions, the better the model's ability to generalize. Training signals from auxiliary tasks can help provide this inductive bias, thereby improving the ability to generalize. In our case of dividing one task up into several, the motivation is to see if the model can learn something on more specific tasks (inductive bias gained from expertise in the task) that it could not have by simply learning the one undivided original task. The first method used, shown on table 4.2 attempts to divide object classes into categories, in the hope of making invidual detection heads "experts" in detecting objects of those categories. The second method used in Multitask OD 2, goes to the extreme, and divides the full set of object classes into its individual elements.

	train		val	
	Images	Objects	Images	Objects
Aeroplane	327	432	343	433
Bicycle	268	353	284	358
Bird	395	560	370	559
Boat	260	426	248	424
Bottle	365	629	341	630
Bus	213	292	208	301
Car	590	1013	571	1004
Cat	539	605	541	612
Chair	566	1178	553	1176
Cow	151	290	152	298
Diningtable	269	304	269	305
Dog	632	756	654	759
Horse	237	350	245	360
Motorbike	265	357	261	356
Person	1994	4194	2093	4372
Pottedplant	269	484	258	489
Sheep	171	400	154	413
Sofa	257	281	250	285
Train	273	313	271	315
Tvmonitor	290	392	285	392

Table 4.1: Table of the distribution of the number of objects by class and dataset in PascalVOC

Vehicles	Animals	House objects
Aeroplane	Bird	Bottle
Bicycle	Cat	Chair
Boat	Cow	Diningtable
Bus	Dog	Pottedplant
Car	Horse	Sofa
Motorbike	Person	Tvmonitor
Train	Sheep	

Table 4.2: Table of class groups. Each group has a name: Vehicles, Animals and House Objects.

4.2 KITTI

A dataset specifically meant for autonomous vehicle development, it is not slpit into a training and validation set like PascalVOC, but I have created a custom random 80/20 train/val split (meaning that 80% of the provided images are used as training data, and the remaining 20% were used for validation). The images are of different driving scenarios and the object classes are:

- Car
- Cyclist

- DontCare
- Misc
- Pedestrian
- Person_sitting
- Tram
- Truck
- Van

Note, the object class with the name "DontCare". These are annotations of objects that should be ignored (i.e. objects that are too far away). In typical scenarios, wether the network predicts an object for that class at a given grid cell or not, the loss should neither punish or reward the network. In my approach, for simplicity's sake, I have ignored DontCare labels altogether. This means that for example, if a don't care object appears on a validation image, and the network predicts a car at that location, the loss function will take it into account (i.e. the network will be punished for it). This is not a desirable effect, but since we just want to compare our own models to each other, and not achieve the highest mAP scores across the board, it is not a significant issue.

For this dataset, the class groupings for multitask OD 1 are the following:

Vehicle	Person	Other
Car	Pedestrian	Misc
Truck	Cyclist	
Van	Person_sitting	
Tram		

Table 4.3: Table of class groups. Each group has a name: Vehicles, Animals and House Objects.

The reasoning behind this subdivision of classes is the same as the one provided for the PascalVOC dataset.

4.3 Metrics

Measuring the performance of an object detector is far from trivial. Before introducing mAP, we must define terms like true positive, true negative, false positive, false negative, precision and recall:

Definition 3. True positive (TP): A true positive is an outcome where the model correctly predicts an object. ▪

Definition 4. True negative (TN): A true negative is an outcome where the model does not predict an object where there should not be one. ▪

Definition 5. False positive (FP): A false positive is an outcome where the model predicts an object where there should not be one. ▪

Definition 6. False negative (FN): A false negative is an outcome where the model does not predict an object where there should be one. ▪

Definition 7. Precision: The number of true positives over the sum of the number of true positives and that of false positives. It quantifies how many objects were predicted correctly out of all predictions.

$$\text{Precision} = \frac{TP}{TP + FP}$$

Definition 8. Recall: The number of true positives over the sum of the number of true positives and that of false negatives. It quantifies how many objects were predicted correctly out correct out of all ground truth objects

$$\text{Recall} = \frac{TP}{TP + FP}$$

One extreme is when the network predicts a very large amount of objects, including most of the ground truth objects. In this case recall is high (close to 1) but precision is low (close to 0). The other extreme to consider is when the network only predicts a few objects, not enough to cover all ground truth objects, but those few predictions are correct. This is the case when precision is high, but recall is low. The perfect object detector predicts only the ground truth objects (not more and not less). To measure the performance of an object detector, we have to consider precision and recall jointly, following the method described below.

Definition 9. mAP: mean average precision (mAP) is the mean of the average precision values measured for each object class separately. Letting AP_i be the average precision of the i^{th} class, and n_c the number of classes, mAP is given by the following equation:

$$\text{mAP} = \frac{1}{n_c} \sum_{i=1}^{n_c} \text{AP}_i$$

4.3.1 AP metric

Let us consider one validation epoch, in which we have a list of prediction results over all images of the validation dataset. One prediction result consists of a confidence value, and a correctness value, which is true if the prediction is a true positive sample, and false otherwise. Over the validation epoch, we count the number of ground truth objects of the relevant class, which is denoted by n_{gt}. First, we filter out all predictions with confidence below 0.5. We then sort all the prediction results by confidence value in descending order and include them in a table such as the one shown in table 4.4

After having computed all precision recall value pairs for all data points (prediction results), we can plot the so called precision recall curve. The plot below does not reflect the values in table 4.4.

Confidence	Correctness	$\sum$Correctness	Precision	Recall
0.99	true	1	1.0	0.01
0.98	false	1	0.5	0.01
0.98	true	2	0.67	0.02
0.97	true	3	0.75	0.03
0.96	false	3	0.6	0.03
0.92	false	3	0.5	0.03
...	...	...	...	...

Table 4.4: the i^{th} row of the $\sum$ Correctness column is the culumative sum all rows of index $\leq i$ with true Correctness. The i^{th} element of the Precision column is $(\sum \text{Correctness})_i / i$ and the i^{th} element of the Recall column is $(\sum \text{Correctness})_i / n_{gt}$. In this example, it is assumed that $n_{gt} = 1000$. The values of the last 3 columns can be filled out row by row from the first row to the last.

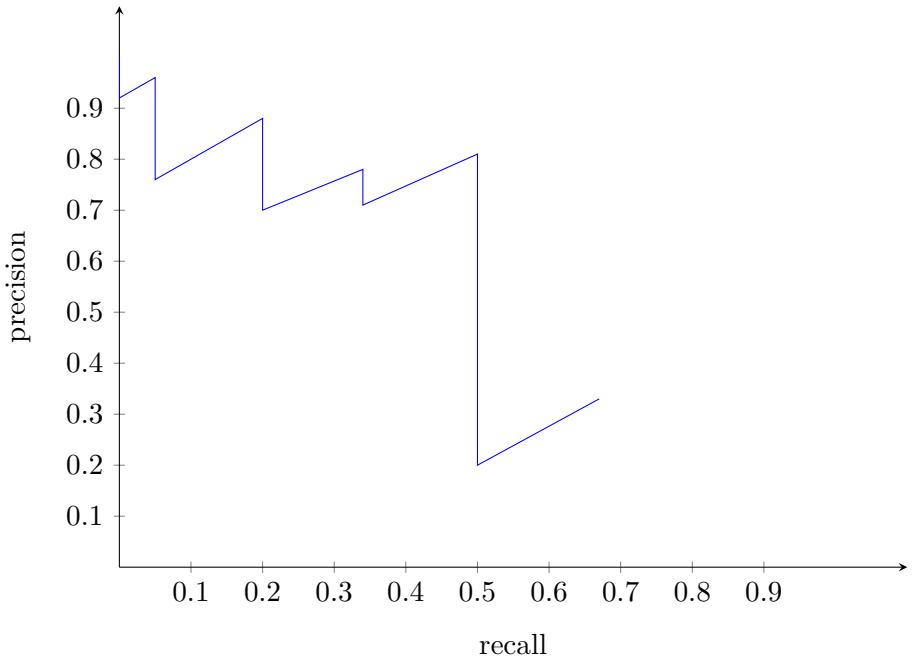

Figure 4.1: A possible precision recall curve

The interpolation method described in PascalVOC [2] states "The precision at each recall level r is interpolated by taking the maximum precision measured for a method for which the corresponding recall exceeds r:

$$p_{interp}(r) = \max_{\widetilde{r}:\widetilde{r} \geq r} p(\widetilde{r})$$

where $p(\widetilde{r})$ is the measured precision at recall $\widetilde{r}$." In other words, we need to make the precision recall curve monotonically decrease. Graphically, this interpolation is shown on figure 4.2.

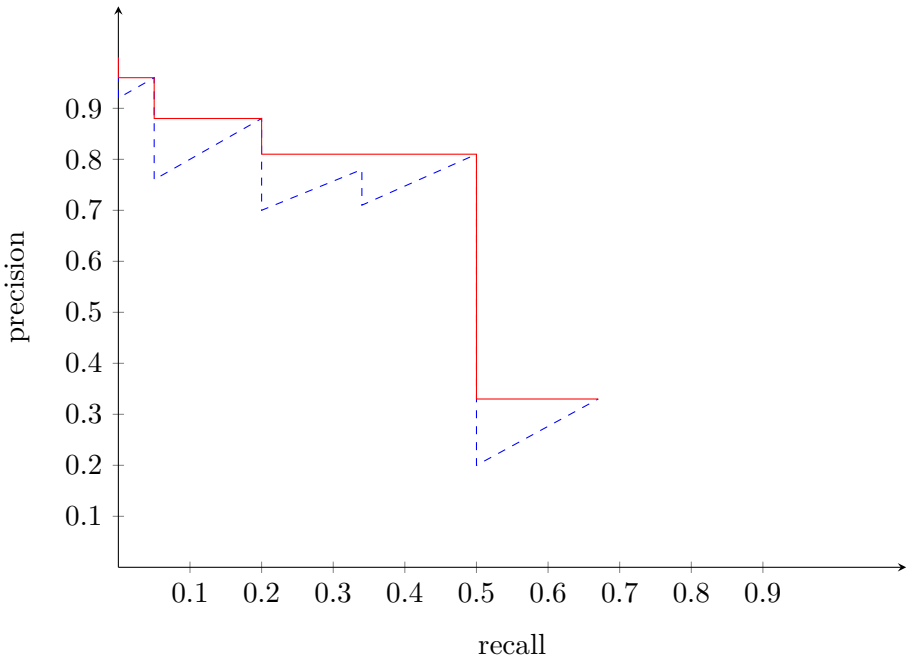

Figure 4.2: Precision recall curve with interpolation (red solid line) vs actual precision recall curve (blue dashed line)

Finally, the AP (average precision) metric is defined as the area under the interpolated precision recall curve as shown on figure 4.3.

4.4 Experiments

4.4.1 Methodology

I have attempted to train all 3 variants of the model (single task, multitask OD 1, and multitask OD 2) using the following parameters:

Using Tensorboard for logging, I plotted the Training loss, the validation loss and mAP in each epoch, and visualized heatmaps for predictions' confidence scores as well as detections on object images and precision recall curves for each object class.

4.4.2 Results

During training, the network struggled to learn confidence correctly: In all cases, there were too many detections, due to confidence values being above the detection confidence threshold in grid cells where there are no ground truth objects. A visual representation of this phenomenon happening in a trained single task network can be see on the figure below 4.4.

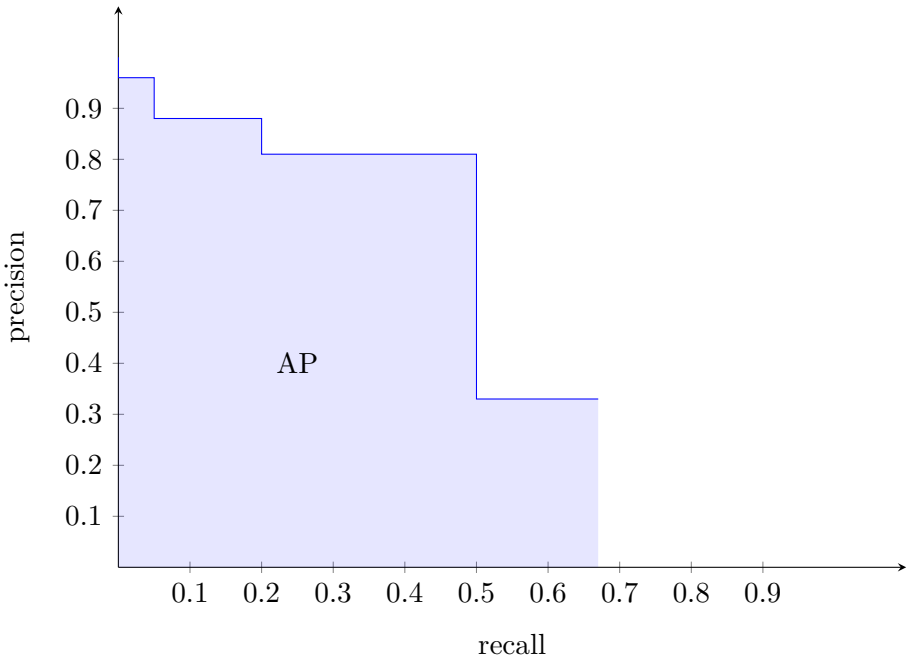

Figure 4.3: Illustraion of AP as the area under the precision recall curve

hyperparameters	
number of epochs	200
learning rate	0.001
batch size	32
optimizer	Adam optimizer with weight decay=1e-4
p_{hflip}	0.5
$p_{gaussianblur}$	0.5
$p_{grayscale}$	0.5

Table 4.5: Hyperparameters used during training

model	n_{groups}	mAP	t_{inf} (ms)
Single task OD	1	**31.6**	**51**
Multitask OD 1	3	11.4	67
Multitask OD 2	8	4.1	94

Table 4.6: Results on the KITTI dataset. n_{groups} denotes the number of class groups used in a given model. t_{inf} denotes the inference time.

The effect described above was amplified in multitask models, since the described error was present across all class groups, and was accumulated when combining their output. This claim is supported by the large drop in mAP as the number of class groups is increased.

The reason that this error was present is most likely due to insufficient training data and augmentations and/or poorly tuned hyperparameters (λ_{obj}, λ_{noobj}, learning rate etc.), because the error was accompanied by overfitting: The observed training loss decreased while the validation loss increased, see figure 4.5.

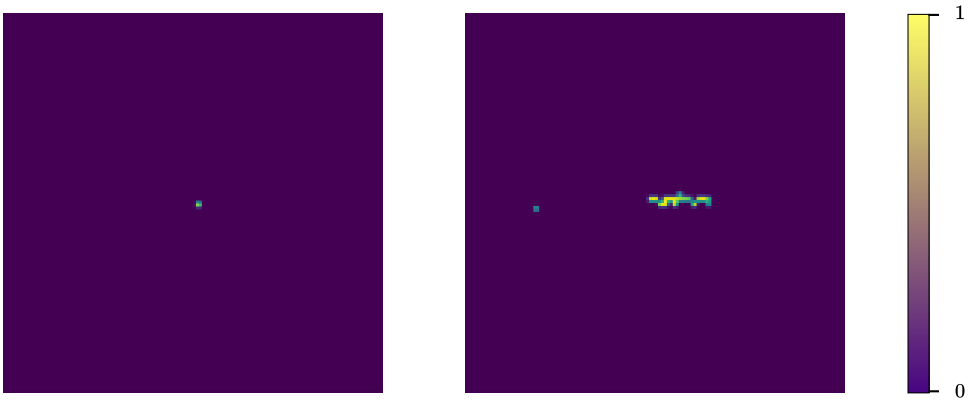

Figure 4.4: Confidence heatmap of the target (left) and that of the prediction (right). Values on the predicted heatmap are scattered more than the on the target.

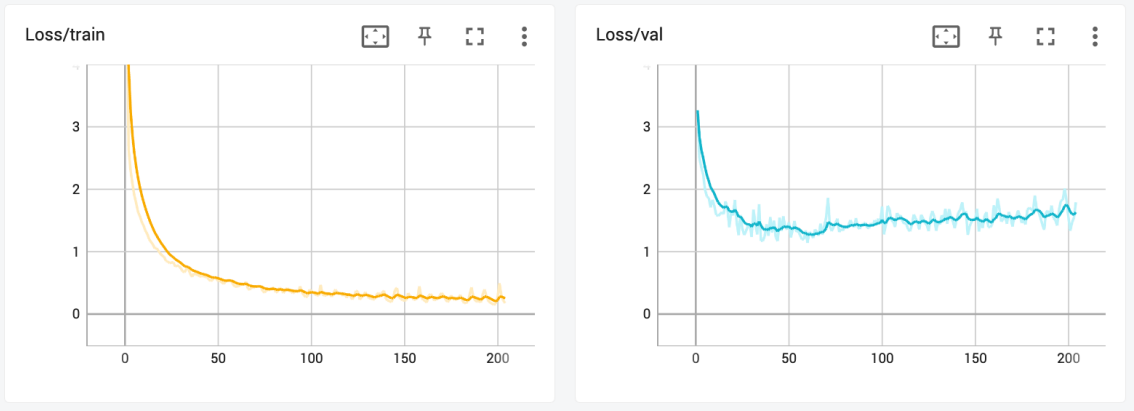

Figure 4.5: Training and validation loss of multitask OD 1 training on KITTI over 200 epochs.

In addition, losses and mAP showed a higher degree of fluctuation in later epochs (between epochs 100 and 200), indicating that the learning rate may have been too high. A learning rate scheduler may have solved this issue.

All training runs ran on the PascalVOC dataset suffered from extreme overfitting and very low mAP. In these runs I had only included the horizontal flip as a data augmentation technique, which partly explains the issue. Models trained on KITTI showed that same symptoms, but with better mAP; this prompted me to evaluate the effects of adding gaussian blur and greyscale as data augmentation. Once I had succeeded in reducing overfitting with these techniques, due to time and computational resource constraints, I could not run all the trainings on PascalVOC.

In the absence of a test set for KITTI, detections were visualized on the validation dataset. The best result, as indicated by table 4.6 was produced by the single task model. Some detections on the first batch of the evaluation are shown below:

Table 4.6 only contains the results obtained for models trained on KITTI, that have been run using all the hyperparameters given in table 4.5. As mentioned previously, before these training runs, several other PascalVOC trainings have been attempted, but these did not run for 200 epochs, as overfitting was more significant. Nonetheless, all results,

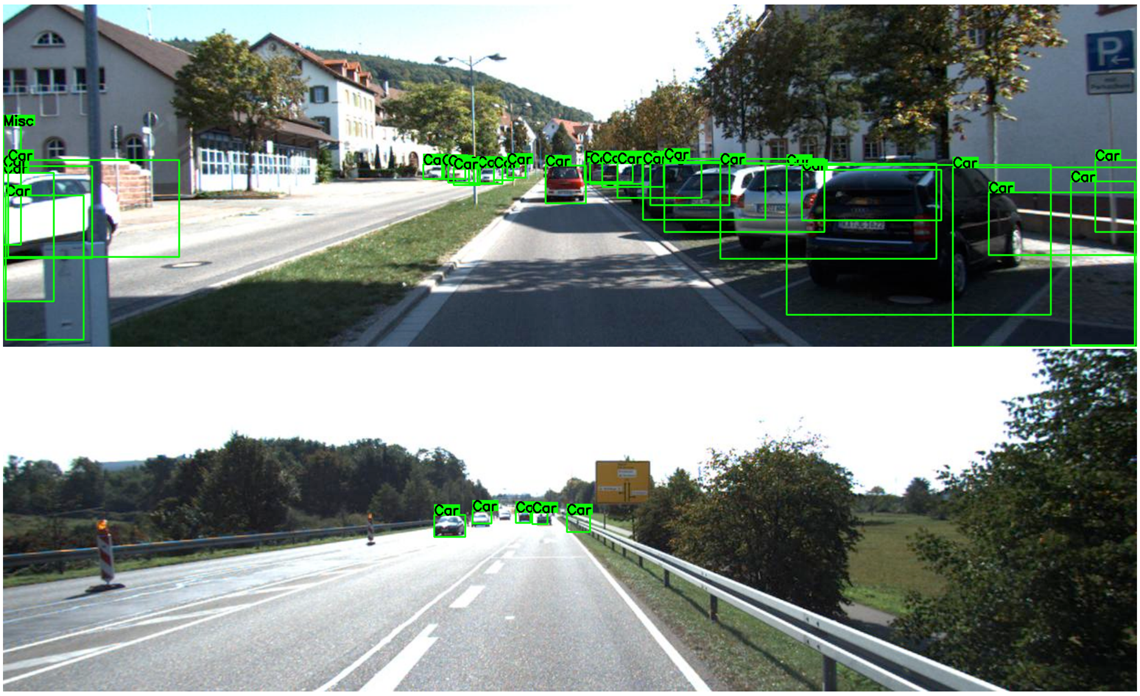

Figure 4.6: Detections produced by the single task model

along with the number of training epochs and augmentation usage, are included in table 4.7

dataset	n_{groups}	mAP	t_{inf} (ms)	n_{epochs}	augmentations
KITTI	1	**31.6**	**51**	200	1, 2, 3
	3	11.4	67	200	1, 2, 3
	8	4.1	94	200	1, 2, 3
PascalVOC	1	0.9	53	72	1
	3	0.2	68	123	1
	20	0.03	100	55	1

Table 4.7: All results summarized. n_{groups} denotes the number of class groups used in a given model. t_{inf} denotes the inference time. n_{epochs} denotes the number of epochs that the model was trained for and in the augmentation column, a list of integers is used to refer to the types of augmentations used during training: 1 refers to horizontal flipping, 2 refers to gaussian blur and 3 refers to grayscale

The issues with PascalVOC trained models led to detections like the ones seen on figure 4.7

The traning and validation loss during the training of the 3 class group multitask model (trained on PascalVOC) is shown on figure 4.8

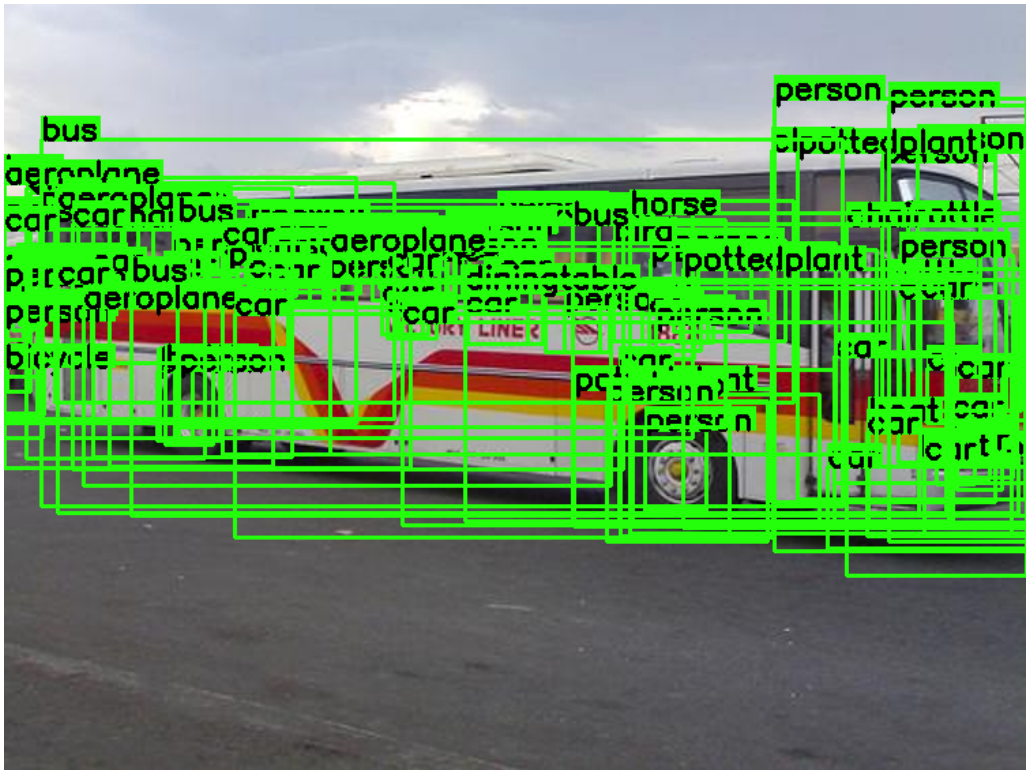

Figure 4.7: Detections produced by a poorly trained, highly overfitted model (trained on PascalVOC). Note the high amount of false positive predictions due to high confidence loss

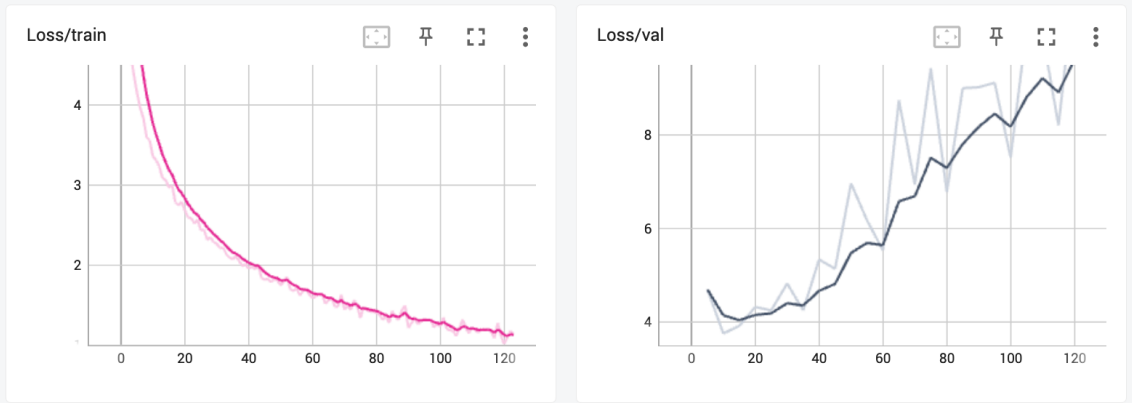

Figure 4.8: The traning loss and validation loss during the training of the 3 class group multitask model (trained on PascalVOC).

Chapter 5

Conclusion

To summarize, I have implemented the YOLO algorithm and adapted it to support multitask learning. The separate tasks were the detection of objects over subsets/groups of a dataset's object classes. The first step was to find a good architecture for the part of the network that all tasks share. This ended up being YOLOv5s, because issues such as overfitting arose when using larger models. The second step, was divide classes into subgroups. One version of the multitask model was a category wise distribution: both datasets that I used, KITTI and PascalVOC had classes that could be split into three categories (see tables 4.2 and 4.3). The third step was to perform k means clustering on all objects of the training dataset per subgroup to find the best anchor boxes. Finally the last step was to train the models with data augmentations randomly with a probability of 0.5 applied on the training input. These augmentations were horizontal flipping, applying a Gaussian blur and applying a Greyscale filter.

Two separate sets of results were obtained. One for models trained on PascalVOC and another for models trained on KITTI. PascalVOC trained models were trained with horizontal flipping augmentation only, which is most likely why they overfit very early and performed poorly. However, KITTI trained models showed higher mAP scores across the board, because gaussian blur and greyscale augmentations were applied to the input when training, allowing to reduce overfitting. Of the latter models, the single task version had the highest mAP by far, and it was observed, that the more tasks we divide "the single task" into, the lower the measured mAP.

With the data available, we can only state with certainty, that the confidence score errors present in the single task detection model's output are also present in each task specific output in multitask models. This causes the overall confidence error in the combined output of multitask models to be greater than that of the single task variant. A solution to this issue is to reduce overfitting even more to allow the models to train for longer periods, and reduce their confidence error. One way to achieve this is to use a larger dataset, like MS COCO [14]. Another way is to use more/better data augmentation techniques, like Cutmix or Mosaic (used in YOLOv5). Another issue that I have observed was the noise in the recorded losses and mAP over later epochs during training. This is was most likely due to the learning rate being too high. An easy way to solve this is to use a learning rate scheduler, e.g. divide the learning rate by 10 every 50 epochs, allowing more stable convergence.

Overall, with the issues of overfitting and unstable loss and mAP, it is not considered fair to compare model performances. However, based on the results of this thesis project,

dividing a single task into separate tasks has not shown any benefits when it comes to object detection performance.

Acknowledgements

I don't think I have ever had an advisor who, as far as I remember, never took more than a day to reply to my messages. Thank you for your invaluable help and overall friendliness, Dr. Márton Szemenyei.

I would also like to thank my Continental advisor, Tamás Kern for his constant support with my internship as well as his helpful tips on visualization, without which the debugging of my models would not have been possible.

Finally, I would like to thank Moni Róbert and everyone involved at Continental and the TMIT faculty for providing me with the necessary hardware to run my trainings.

Bibliography

[1] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html, .

[2] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (VOC) challenge. 88(2):303–338, . DOI: 10.1007/s11263-009-0275-4.

[3] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[4] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Dropblock: A regularization method for convolutional networks.

[5] Ross Girshick. Fast r-cnn.

[6] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. DOI: 10.1109/cvpr.2016.90.

[8] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. 2(5):359–366. DOI: 10.1016/0893-6080(89)90020-8.

[9] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[10] Glenn Jocher. YOLOv5 by Ultralytics, 5 2020. URL https://github.com/ultralytics/yolov5.

[11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. 60(6):84–90. DOI: 10.1145/3065386.

[12] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. 86(11):2278–2324. DOI: 10.1109/5.726791.

[14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context.

[15] Diganta Misra. Mish: A self regularized non-monotonic activation function.

[16] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML 2010*, pages 807–814, 2010.

[17] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. .

[18] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. .

[19] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks.

[21] Pierre Sermanet, David Eigen, Xiang Zhang, Michael Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks.

[22] Falong Shen and Gang Zeng. Weighted residuals for very deep networks.

[23] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions.

[24] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection.

[25] Chien-Yao Wang, Hong-Yuan Mark Liao, I-Hau Yeh, Yueh-Hua Wu, Ping-Yang Chen, and Jun-Wei Hsieh. Cspnet: A new backbone that can enhance learning capability of cnn.

[26] Zhuliang Yao, Yue Cao, Shuxin Zheng, Gao Huang, and Stephen Lin. Cross-iteration batch normalization. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. DOI: 10.1109/cvpr46437.2021.01215.

[27] Zhaohui Zheng, Ping Wang, Wei Liu, Jinze Li, Rongguang Ye, and Dongwei Ren. Distance-iou loss: Faster and better learning for bounding box regression.

[28] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points.