Fast Object Recognition with MobileNet and YOLO

1st Arzt Johannes
Munich University of Applied Sciences
Munich, Germany
jarzt@hm.edu

2nd Dogan Samed
Munich University of Applied Sciences
Munich, Germany
sdogan@hm.edu

3rd Großbeck Thomas
Munich University of Applied Sciences
Munich, Germany
grossbec@hm.edu

***Abstract*—The following paper describes the implementation of an algorithm for the detection of bees. The implementation is based on *MobileNet* as feature extractor and *YOLO* as detection network. Two separate detection blocks on different image scales greatly improved model capacity. The provided dataset was doubled in size and augmented through random brightness and contrast changes. Random noise was applied as well, increasing model robustness. Generalization was improved by utilizing a small batch size of 8, though even smaller mini-batches did not improve results significantly. Major changes in detection quality could not be observed through different loss coefficients. Proper detection is hindered by large groups of clustered bees, due to the small resolution of the detection grid. Thence, increasing grid size of the final prediction layer is proposed.**

I. Introduction

Object detection has been a cornerstone of computer vision for decades. Albeit many achievements, detection and fine-tuning still remains a challenge for specific niche applications. The following paper describes the implementation of robust and efficient bee-detection, given a small dataset of 2000 labeled images. Subject focus consists of architectural decisions, data preprocessing, hyper-parameter search and model evaluation.

II. Architecture (Samed Dogan)

A. *MobileNetV2*

Network accuracy directly relates to the magnitude of parameters of the given network. However, such gain in accuracy comes at the cost of performance speed at inference. Many deep neural networks demand high computational resources, making the network practicably unusable for mobile setups. In scope of the given task, bee detection should be feasible in real-time on constrained environments. Thus, in order to strike a balance between accuracy and performance, MobileNet is used as feature-extractor and constitutes the backbone of our bee detection network.
As described in [10], memory footprint is reduced by utilizing linear bottleneck layers. The compressed low-dimensional tensor is first expanded onto a high-dimensional space, where depth-wise convolution is applied. The high dimensional tensor is then back-projected to the low-dimensional representation by a linear convolution layer, hence reducing the size of intermediate tensors. Network propagation is further improved by introducing residual connections [4] for a sequence of bottleneck blocks, alleviating the vanishing gradient problem for deep networks. The general structure of a bottleneck block is depicted in figure 1. ReLu activation is capped at six

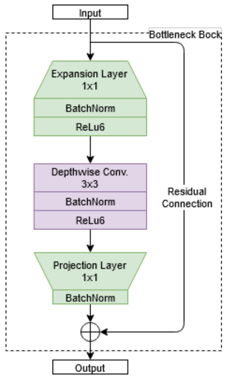

Fig. 1: Sequence of expansion layer, filter layer and projection layer with *c* input and *c'* output channels. Expansion and filter layers omit $c * ex$ channels, where *ex* refers to the expansion factor.

for more robust low-precision computation [10][5]. The input image of size $96x416x3$ is down scaled 32 times, resulting in a $3x13x320$ tensor used for further computation. The detailed architecture is similar to [10] and given in table I.

Input	Layer	c	n	ex	s
96x416x3	BConv2D 3x3	32	1	-	2
48x208x32	BottleneckBlock	16	1	1	1
48x208x16	BottleneckBlock	24	2	6	2
24x104x24	BottleneckBlock	32	3	6	2
12x52x32	BottleneckBlock	64	4	6	2
6x26x64	BottleneckBlock	96	3	6	1
6x26x96	BottleneckBlock	160	3	6	2
3x13x160	BottleneckBlock	320	1	6	1
3x13x320	BConv2D 1x1	1280	1	-	1

Table I: Architecture of the feature extraction network. BConv2D refers to a sequence of convolution, batch-normalization and Relu6 activation. All layers are repeated *n* times and share *c* output channels, whereas the first layer of each sequence uses stride *s*. The expansion factor *ex* is applied to the input, yielding a intermediate tensor of *ex*input* channels.

B. *YOLO V2.5*

Building on the performance aspect of MobileNet, Yolo proved to be reliable as underlying detection algorithm. Each new version of Yolo introduces new state-of-the-art techniques such as skip connections, residual blocks, multi scale predictions or drop-block regularization. However, with regards to the given dataset and specific detection task, outdated architectures were chosen in favor of simplicity, though still producing viable results.(See V) Therefore, a mixture-model between YoloV2 and V3 is proposed. Multiscale predictions were discarded due to the bees continuous shape and appearance and replaced by three independent detection blocks, respectively. Every detection block consists of a series of 3x3 filter and 1x1 reduction layers, as firstly described in [11]. Detailed detection

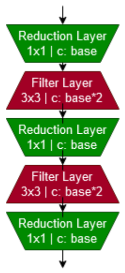

Fig. 2: Detect-block where c refers to the output channel of each layer e.g., DetetcBlock(512) would bear two intermediate tensors of 1024 channels.

architecture is referred to in table II. In order to preserve low-

Input	Layer	c	s
12x52x32	DetectBlock	128	1
12x52x128	BConv2D 3x3	128	4
6x26x96	DetectBlock	256	1
6x26x256	BConv2D 3x3	256	2
3x13x1280	DetectBlock	512	1
3x13x512	BConv2D 3x3	512	1
3x13x896	Conv2D 1x1	n*(5+1)	1

Table II: YoloV2.5 architecture. The ReLu6 activation in the BConv2D layer is replaced by LeakyReLu with $\alpha = 0.1$

level semantic information, the last two intermediate outputs of the feature extraction network are passed forward to each corresponding detection block. Resulting tensors are further down scaled to the same spatial dimension via strided convolutions and subsequently concatenated along their depth dimension in a similar fashion to [6]. Bounding box prediction is performed by a concluding convolution with $n * (5 + classes)$ channels, where n refers to the number of bounding boxes per grid. The complete network architecture is pictured in figure 3.

III. Data Preprocessing

A. Data Augmentation (Thomas Großbeck)

An important part of deep learning is the preparation of the data. Since only 2000 images are available at the beginning, the first step is to expand the data set to achieve a better learning effect. Training with only a few images would lead the model to overfit. For this purpose, the images and corresponding bounding boxes were flipped horizontally, thus effectively doubling the dataset in size. Furthermore, this ensures an even distribution of bee positions across the dataset.

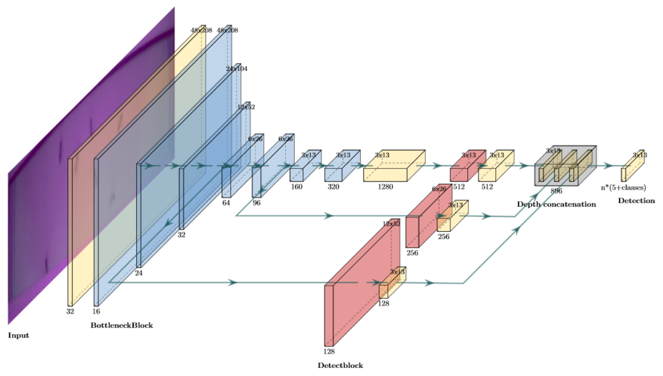

Fig. 3: Final Network architecture consisting of different building blocks distinguished by color. Regular 1x1 and 3x3 convolutions are depicted in yellow, bottleneck layers in blue and detection layers in red. Figure was created with [1]

The dataset is randomly shuffled and divided into training and test data using an 80/20 split. Variety in the training dataset is further increased by randomly changing image brightness and contrast. Also, small noise is added, improving generalization performance and robustness [3, pg. 236-238]. Pixel values are normalized to [-1,1] prior to training.

B. YOLO-Format (Johannes Arzt)

The input image have to be scaled to 96x416px to fit the input format of our YOLO detection network. The output of our YOLO detection network (see in section II-B), is a 3x13 Grid with a depth of nx(5+number of classes), where n is the number of anchors per grid cell. Every anchor is described through:

- X/Y: centroid of the anchor with regards/in regard to the grid cell.
- W/H: height/width of the anchor in regards to the grid cell
- O: object mask, indicating the presence of an object within the respective cell
- C: class label of the respective object

X	Y	W	H	O	C

Fig. 4: YOLO-format for every anchor

The bounding boxes are given in pixel coordinates (top-left corner, width, height) and need to be transformed to grid coordinates. Each grid cell represents a 32x32px large piece of the input image, thus the scale is 32.

$$centerbb_x = left + \frac{width}{2} \tag{1}$$

$$centerbb_y = top + \frac{height}{2} \tag{2}$$

$$index_x = \lfloor \frac{centerbb_x}{scale} \rfloor \tag{3}$$

$$index_y = \lfloor \frac{centerbb_y}{scale} \rfloor \quad (4)$$

The remaining calculations for X and Y are equivalent.

$$x = \frac{centerbb_x - index_x \cdot scale}{scale} \quad (5)$$

$$W = \frac{width}{scale} \quad (6)$$

$$H = \frac{height}{scale} \quad (7)$$

The calculated values are set in the grid at $index_x, index_y$ for the anchor that has the best IOU (intersection over union) in regard to W and H, with:

$$IOU = \frac{min(w, anchor_w) * min(h, anchor_h)}{anchor_w * anchor_h + w * h - intersection}. \quad (8)$$

[2] O is set to one and C is set to zero for the best fitting anchor within the grid cell. Output boxes are calculated as described in figure 5, given four output coordinates t_x, t_y, t_w, t_h for each predicted bounding box.

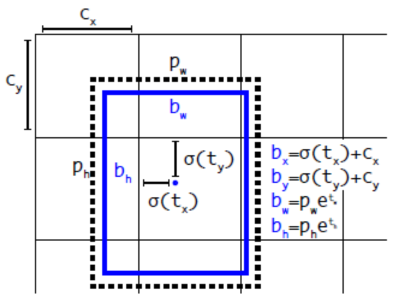

Fig. 5: Bounding box calculation in Yolov2, where c_x, c_y is the offset from the top-left corner of the image and p_w and p_h define the width and height of the used anchor. Image taken from [8]

IV. Evaluation of the Hyperparameters

A. Loss Metric (Thomas Großbeck)

The loss function provides information about the quality of the model. The lower the values here, the better the model fits the data. Model loss can be divided into the three main components [12]:

- localization loss: Considers the position and size of the predicted bounding boxes.
- classification loss: Provides information about the quality of the classification. It is the accuracy of the predicted classes
- confidence loss: "Measures the sensitivity of the detection" [12]. Looks at the number of detected objects and whether objects were falsely detected (false positives).

However, loss functions do not fully reflect detection quality in terms of bounding box accuracy or false predictions. Hence, precision and recall is used as performance metric for detection related hyperparameters. Precision is given as $\frac{t_p}{t_p+f_p}$ and provides the ratio of correct to false predictions, whereas recall is given as $\frac{tp}{t_p+f_n}$ and describes the percentage of correct detections with respect to the ground truth data. Recall and precision are calculated for each test image with an IOU threshold of 0.5. The described metrics are later used in the hyperparameter search to validate the model.

B. Training Hyperparameters (Samed Dogan)

Architectural choices were validated by the ability to overfit for a single image. Earlier versions, containing merely one prediction block, suffered from high-bias. Thus, two more blocks with strided convolutions were added, see II-B. MobileNet layers were initialized with weights pre-trained on the ImageNet dataset and frozen afterwards. After guaranteeing algorithmic consistency and sufficient capacity, the network was trained on a small subset of images, ensuring the ability to converge for a given set of parameters. As a result, learning rates above 10^{-5} and below 10^{-6} were discarded. Because of poor convergence with Adam, RMSProp with $\rho = 0.9$ and $\epsilon = 10^{-7}$ was used instead. This could be due to the added momentum in Adam, which appears to be obstructive for highly curved cost functions. Consistent with [7], we observed a significant drop in model quality for large mini-batches, i.e for batches greater than 16. Model quality is determined through the ability to generalize, that is, correct detection on previously unseen data. Consequently, optimal batch size is assumed to be in the range of 4 to 8, where sizes below 4 are not feasible in view of time efficiency and sizes above 8 may hurt performance. Lastly, the network was trained for a maximum of 400 epochs on the full training set with predetermined hyperparameter ranges. Validation loss is depicted in figure 6. We chose 400 epochs as trade-off between

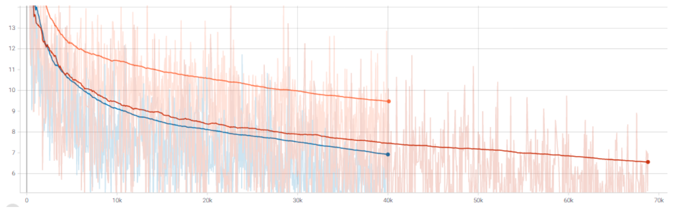

Fig. 6: Validation losses for different batch-sizes b and learning rates lr with $b = 8$, $lr = 10^{-6}$ (orange), $b = 8$, $lr = 10^{-5}$ (blue) and $b = 4$, $lr = 10^{-5}$ (red). Training took approximately 14 hours for $b = 8$ and 26 hours for $b = 4$

training time and accuracy. As seen in 6, loss is decreasing consistently, indicating further room for improvement without the risk of overfitting. Since smaller batchsizes or learning rates did not improve results, $b = 8$, $lr = 10^{-5}$ were chosen as appropriate parameters.

C. Detection Hyperparameters (Thomas Großbeck)

Previously chosen anchors boxes of size (0.53, 0.375) and (0.5, 0.4) resulted in small bounding boxes, incapable of fully enclosing each bee. As a result, larger rectangular anchor boxes were used instead. Anchor sizes (1,2) and (3,4) proved to be more suitable, given the bees appearance. The number of anchor boxes and their size was not changed during our measurements. Due to the fact that many cells do not contain a bee, the scores will be close to zero. This can lead to the gradients for cells with bees becoming overpowered, so that the model no longer converges. To counteract this, the coefficients *coeff_bb*, *coeff_obj* and *coeff_nonObj* are introduced to count for this in the loss function. The three values thus determine how strongly the prediction of the bounding boxes and the object mask are weighted in the loss. [9]

To determine the influence of these variables, the metrics described above can be used. A training run with the values suggested in the paper [9] serves as a basis for comparison. Here, *coeff_obj* and *coeff_bb* are each set to 5, whereas *coeff_NonObj* is set to 0.5. We also have tested *coeff_nonObj*=2 and *coeff_obj*=1. Even tough small changes could be observed

results with batch 8 and iou 0.5				
coeff_bb	*coeff_obj*	*coeff_nonObj*	precision	recall
5	5	0.5	61.25 %	48.02 %
5	5	2	62.52 %	44.46 %
5	1	2	61.03 %	48.06 %

Table III: Average precision and recall on all test images given different loss coefficients

concerning recall, coherence between small parameter changes and better performance can not be assumed. Due to extensive training times, evaluation data remains sparse.

V. Conclusion (Johannes Arzt)

Detection is reasonably stable and precise for images containing a small amount of bees, regardless of image quality.

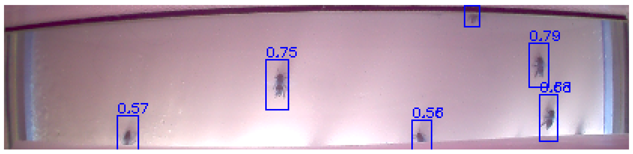

Fig. 7: Successful detection of bees

However, the network has poor detection when many bees are grouped at one point in the image, as seen in figure 8. The

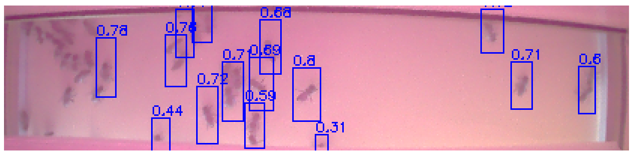

Fig. 8: Bad detection of grouped bees

ground truth 3x13 grid merely contains two anchors per cell, consequently reducing large clusters to a maximum of two big bounding boxes. As a consequence, the network is learning with conflicting ground truth data. This can be counterfeited by increasing the number of available anchors boxes as well as grid size. On the downside, such changes greatly increase network parameters, rendering training on small datasets more difficult. Reduction of parameters could be further achieved by using bottleneck layers in detection blocks with depthwise convolution instead. In summary, we can say that the detection hyperparameter do not have a very big influence on the average precision (see in Table III).

References

[1] URL: https://github.com/HarisIqbal88/PlotNeuralNet (visited on 04/02/2021).

[2] URL: https://github.com/jmpap/YOLOV2-Tensorflow-2.0/blob/master/Yolo_V2_tf_2.ipynb (visited on 03/02/2021).

[3] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[4] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

[5] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861. URL: http://arxiv.org/abs/1704.04861.

[6] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993. URL: http://arxiv.org/abs/1608.06993.

[7] Nitish Shirish Keskar et al. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.

[8] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[9] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[10] Mark Sandler et al. "Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation". In: *CoRR* abs/1801.04381 (2018). arXiv: 1801.04381. URL: http://arxiv.org/abs/1801.04381.

[11] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842. URL: http://arxiv.org/abs/1409.4842.

[12] Zixuan Zhang. *YOLO_Explained*. URL: https://github.com/zzxvictor/YOLO_Explained (visited on 02/02/2021).