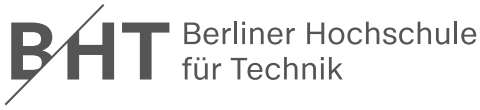

Berliner Hochschule für Technik

Fachbereich II Mathematik - Physik - Chemie

Bachelorarbeit

von

Philipp Zettl

zur Erlangung
des akademischen Grades

Bachelor of Science (B. Sc.)

im Studiengang
Mathematik

Thema:

Machine Learning Methods for the Localization and Classification of Insects in Images

Betreuer:	Prof. Dr. Frank Haußer
Betreuer:	M.Sc. Teodor Chiaburu
Gutachter:	Prof. Dr. Andreas Tewes
Eingereicht:	4. April 2022

Preamble

Hiermit erkläre ich, dass ich die vorliegende Abschlussarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
I hereby declare that I have written this thesis independently and have not used any sources or aids other than those indicated.

Abstract

This thesis has been written in the citizen science project *KInsekt* at the *Berliner Hochschule für Technik*. Its main objective is to investigate different Machine Learning techniques for the localization and classification of insect orders, namely "Coleoptera", "Hymenoptera, Formicidae", "Lepidoptera", "Hemiptera" and "Ordonata", based on image files. The accompanying code repository (`https://gitlab.com/kinsecta/ml/thesisphilipp`) contains software written in Python (version 3.6.9), developed using the libraries `numpy`,
`tensorflow`, `keras`, `keras-tuner` [1] and `scikit-learn` [2]. The code has been written in the attempt to be easily extendable or changeable, to e.g. append the list of available classification classes.
All used algorithms and "random" generated numbers are seeded, using the seed 42.
The Machine Learning model is supposed to run efficiently on a small computer, such as the RaspberryPi, therefore widely used architectures can not simply be used.
This thesis contains a brief description of the Machine Learning pipeline from data collection, and preparation to preprocessing of the data set and finally using the resulting data set to train different models and architectures. At the end, the best models, based on predefined metrics, will be chosen and its performance against state-of-the-art architectures, including YOLO, evaluated. The results of this evaluation will then reveal that custom tailored architectures perform worse on the given task, when compared to SotA architectures.

?abstractname?

Die hier vorliegende Arbeit wurde im Rahmen des Citizen-Science Projekts *KInsecta* an der *Berliner Hochschule für Technik* verfasst. Ihr Hauptziel ist es verschiedene Machine Learning Techniken zur Lokalisierung und Erkennung von Insekten der Ordnung "Coleoptera", "Hymenoptera, Formicidae", "Lepidoptera", Hemipteraund Ordonata, anhand von Bildern zu erkunden. Das begleitende Code-Repository (`https://gitlab.com/kinsecta/ml/thesisphilipp`) enthält Programme geschrieben in Python (Version 3.6.9), welche mit Hilfe der Bibliotheken `numpy` `tensorflow`, `keras`, `keras-tuner` [1] und `scikit-learn` [2] flexibel entwickelt wurden, sodass spätere Anpassungen, beispielsweise die Erweiterung einer neuen Ordnung, möglich ist.
Alle Programme und zufälliggenerierte Zufallszahlen, die für dieses Projekt entwickelt wurden, sind mit einem Seed 42 initialisiert worden.
Das resultierende Machine Learning Modell zielt darauf ab effizient und platzsparend auf einem kleinen Computer, wie ein RaspberryPi, ausgeführt zu werden.
Diese Abschlussarbeit beinhalten kurze Erläuterungen des Ablaufes eines Machine Learning Projekts, von der Beschaffung und Weiterverarbeitung eines Datensatzes zum Training verschiedener Modellarchitekturen. Am Schluss werden die besten Modelle anhand von vorher definierten Metriken gewählt und gegen State-of-the-Art Architekturen, inklusive YOLO, verglichen. Diese Vergleiche werden zeigen, dass State-of-the-Art Architekturen besser, als maßgeschneiderte Architekturen, dafür geeignet sind das hier beschriebene Problem zu lösen.

Contents

1 Intro

Over the last two decades Machine Learning (ML) algorithms have advanced drastically to the extent that they are portable enough to be carried in our pockets or to be executed on a device with similar hardware constraints such as a *RaspberryPi* microcomputer.
Due to increasing amount of interest in the development of faster and computationally better GPUs, in combination with the growing amount of available data, Deep Learning (DL), a field of Machine Learning, started to gain popularity. The groundbreaking 1998 paper by Yann LeCun et al. [3] introduced *LeNet-5*, a Convolutional Neural Network (CNN) architecture, designed for recognition of handwritten check numbers.
More than a decade later the *AlexNet* model [4] won the 2012 ImageNet ILSVRC challenge by a large margin. *AlexNet* is structured similarly to *LeNet-5*, but has much more parameters, which gave it the advantage over other submissions in that year. Hence, *AlexNet* is considered to mark the start of widely spread Deep CNNs in research and industry.
Since then many new CNN architectures got introduced that reached even higher scores in similar competitions, such as the VGG-16 [5] or *MobileNet* [6] models, which will be used here for benchmark comparisons. Deep NNs and Deep CNNs are not limited to solve computer vision tasks, they can be used to solve a wide variety of problems, e.g. solving tasks like speech recognition using audio data, analyzing trends in the stock marked based on tabular data or predicting sentences for emails using textual data.
The following section will describe the motivation behind this paper, give an introduction into the problems that are being solved, briefly review existing work in the field and give an overview about the following sections.

1.1 Motivation

The 2017 paper of Caspar A. Hallman et al. [7] revealed shocking facts about the reduction of biomass over the last decades.
The authors explain that their results of 27 years of research, in lowland areas of west Germany, give them reason to estimate a seasonal decline of 76%, and midsummer decline of 82% in flying insect biomass. Additionally, the researchers point out that this decline is not dependent on the habitat type, however a change in weather or habitat can not explain this overall decline. Furthermore, the paper states that this decline is evident throughout growing season and irrespective of habitat type or landscape configuration, which suggests the involvement of large-scale factors. Unfortunately the researchers were not able to incorporate factors such as pesticide usage, increased usage of fertilizers or frequency of agronomic measures, which may form a plausible cause.
To make people aware of this problem and of insects' role in the environment as well as to start the process of collecting more insights, about this decline in biomass, the *KInsecta* [1] project was formed. *KInsecta* is a citizen science project [2] of BHT in cooperation with the UBZ Listhof [3] and is part of the

[1] KInsecta project website: `https://kinsecta.org/`
[2] Citizen Scientist: a member of the general public who collects and analyses data relating to the natural world, typically as part of a collaborative project with professional scientists.
[3] UBZ Listhof website: `https://listhof-reutlingen.de/`

Figure 1: Coleoptera sample

Figure 2: Hemiptera sample

Figure 3: Lepidoptera sample

Figure 4: Odonata sample

Figure 5: Hymenoptera sample

funding programm "KI-Leuttürme für Umwelt, Klima, Natur und Ressourcen". Its objective is to build a device capable of monitoring insects in a local area, e.g. a garden. The device works in a way that, in contrast to other monitoring techniques, collects data about insects in the area where it is deployed without harming the individual. Instead of killing or collecting insects alive for later analysis, *KInsecta*'s monitoring device uses different sensors, such as a *Wingbeat* [4] sensor to detect wing beat frequencies, and a high resolution camera to collect visual data. The camera module and an experimental setup can be seen in Figure 33, Figure 34 and Figure 35.
The collected data of the *Wingbeat* sensor will eventually get processed by a custom ML classifier. In addition to that, the device misses a computer vision classifier to process the images collected from the camera. This thesis investigates different ML methods to set the ground to build an extendable, stable

[4]The *Wingbeat* sensor is an effort of the *KInsecta* project team, a reference page can be found at `https://kinsecta.org/sensorik-2#wingbeat` (German).

algorithm to detect insects in images.

Detecting and classifying objects in images is not a novel task, solutions to this task have been around for decades. But most techniques require a lot of pre-processing work on the input data, different methods to apply to the images and/or a lot of computational resources such as RAM. Where conventional approaches are too dependent on one task, existing ML Models seem to be too complex and too resource intense. Hence a tailored architecture is desired.

1.2 Problem description

As the title suggests this thesis focuses on different methods to localize and classify insects in image data. The tasks of localization and classification can be solved in different ways using different techniques. The main question this paper will try to answer is, if there is a ML method which solves both tasks in an acceptable manner, while following resource and architecture constraints to allow usage of the resulting program on a *Raspberry Pi*.
The acceptance criteria for such method is called a metric, that is either, in the case of the classification task, the accuracy of correct predictions, or in case of localization, further also called Bounding Box regression (BBReg), a visual comparison of predicted Bounding Boxes (BBs) with their corresponding ground truth, the *GIoU* metric. The exact definition of these metrics can be found in the subsection "Performance metrics".
Generally speaking the objective of this thesis is to use image data, such as the example images Figure 1, Figure 2, Figure 3, Figure 4 and Figure 5 [5] , process them using a ML method to then obtain a class label with an accuracy comparable to results presented in the 2017 iNat data set paper [8]. To do so, two approaches are chosen.
First, in the **two-stage-method**, two dedicated methods process the input data to come to a final conclusion of a class label, as displayed in Figure 6. One method is used to first detect insects in the raw input images, by predicting BB coordinates. After obtaining the coordinates the input image gets cropped to then only contain the patch covered in the BB. The resulting cropped image will then be used as input image for another method that will perform the classification task on it to obtain a class label.
For the **single-stage-method**, methods get explored that can solve both tasks at the same time, either in parallel or by building onto each other as displayed in Figure 7.

1.3 Existing work in the field

As stated in the introduction, since the publication of *AlexNet* a wide range of new computer vision, specifically object detection, architectures got released, including *VGG-16* and *MobileNet*. Both of these architectures were designed with the objective of solving a classification task and generally output extracted

[5]Image Credits: 1 (CC-BY-NC) by user "m3d5" `https://www.inaturalist.org/observations/36275496`, 2 (CC-BY-NC) Observation © Uriah Resheff `https://www.inaturalist.org/observations/13067479`, 3(CC-BY-NC) Observation © Gregory Greene `https://www.inaturalist.org/observations/30220853`, 4 (CC-BY-NC) Observation © orewoet `https://www.inaturalist.org/observations/32606254` and 5 (CC-0) Observation © Wouter Koch

Figure 6: **2-stage-method**, top down: an image gets passed to a method solving the localization task, then the input image gets cropped to the shape of the predicted BB. At the end a second method is used to obtain the class label.

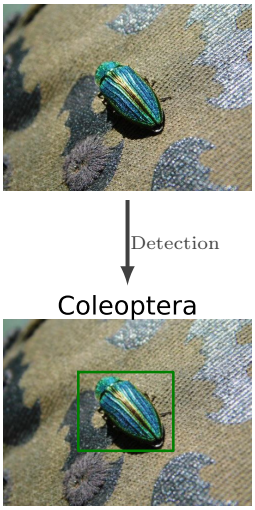

Figure 7: **Single-stage-method**, an image gets processed by a method that outputs a Bounding Box and a classification label prediction.

features from the input images to either predict one out of 1000 (VGG-16) or 200 (MobileNet) classes. Hence, both can be grouped together with methods that solve the tasks individually, using two independent methods or in this case model instances. In 2016 Redmon et al. [9] introduced a novel architecture, *YOLO*, that contains a single Neural Network that predicts Bounding Boxes and class probabilities directly from full images in one evaluation, which is the reason for the name of this architecture, **Y**ou **O**nly **L**ook **O**nce. YOLO essentially subdivides the image into regions and predicts BBs and probabilities for each region. Another architecture is *MobileNet-SSD* from the *MobileNet* [6] paper. An architecture based on the **S**ingle **S**hot **D**etection framework [10], that essentially works the same way as YOLO. Both of these methods are part of the previously mentioned **single-stage-method** approach to solve the detection problem. But most of these industry standard architectures come with a big

disadvantage. To generalize over a wide range of tasks and to generalize big fractions of the training set the models contain a huge number of parameters resulting in high resource requirements to train, maintain and use these models. In the Application section we look deeper into problems of such architectures. To make them available for devices like the *RaspberryPi* a lightweight form can be created, e.g. by pruning the dense model into a sparse model or quantizing its floating point valued weights into integer values [6]. Each of these techniques will trade accuracy for size and computational speed.

1.4 Approaches to solve the problem

There are many different solutions and approaches to both classification and regression tasks. Using dedicated methods for each of the tasks, the 2-stage-method, or using a single method to solve both tasks at once, the single-stage-method, is one of the aspects this thesis analyzes. After briefly reviewing conventional statistics methods such as Linear Regression, or Support Vector Machines (SVMs), the reader is invited in the following sections to read more about modern methods such as Neural Networks (NNs) and Deep Learning (DL) and Convolutional Neural Networks (CNNs) to solve both tasks individually or combined. Whereas conventional methods allow fast solutions by precomputing required parameters, modern techniques require iterative algorithms to slowly adjust parameters until they converge to a solution. This process of fitting parameters until they match a combination that results in satisfying outputs is called the training of an ML algorithm, also called training a model. There are multiple ways of performing the training of a model, that will be discussed in Training Algorithms. Because the data set used here consists of pre-existing class labels, this work focuses on pure *supervised learning* tasks.

1.5 Strategy

The following sections follow the general pipeline of a Machine Learning project, as visualized in Figure 8, and is separated into four theoretical sections followed by two sections describing the application and solution to the given problems. The next three sections will describe and explore the underlying data sets ("Available data sets"), discuss used data processing methods in "Data preprocessing" and discuss different "Machine Learning Methods" to solve both tasks. Section "Application" will focus on the application of previously mentioned ML methods followed by the last section "Conclusion" wrapping up re-

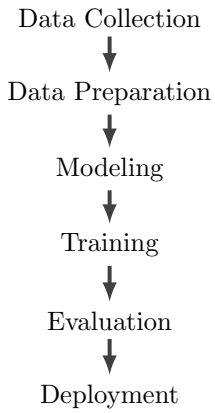

Figure 8: Machine Learning Project Pipeline

[6] Some techniques are described e.g. in the *Tensorflow* documentation `https://www.tensorflow.org/lite/performance/model_optimization`, in addition to that we will look closer into some of the techniques described there in a later section (5.4).

sults and motivating the reader to continue exploring other methods and improving the results displayed here.

The theory and experiments here will only discuss briefly the previously mentioned conventional approaches, as they are considered widely known and appear in most statistics literature and base statistics courses [11–13]. After setting ground and measuring the performance of these methods using different performance measures, so called "Performance metrics", the more modern approach of NNs and CNNs and the combination of them will be explained and explored further and it will be shown that these algorithms and methods solve the task in a more precise and accurate way.

2 Available data sets

For the classification and localization of insects in images, an existing data set was used and extended with Bounding Box annotations. Additionally, a cropped version of this annotated data set is created, that will be used later for the training of a **two-stage-method**. The following section will describe the taxonomy used for the classification task, introduce used data sets and explain what has been done to generate additional labels for the task of Bounding Box prediction.
It should be noted here that all image examples used for illustration in this paper have been published as part of the "iNaturalist" data set under the **C**reative **C**ommons (CC) license, and wherever it was possible to research the origin of the images, reference has been made to the creators.

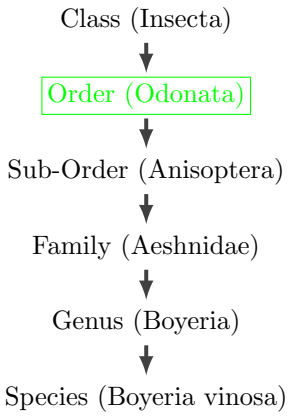

Figure 9: Simplified taxonomic tree of insects. From top to bottom the tree becomes more and more fine-grained.

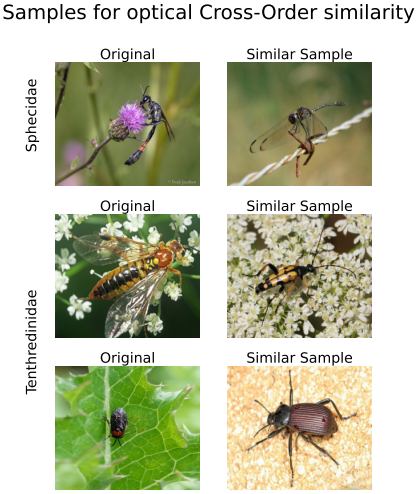

Figure 10: Samples for two families of the order Hymenoptera, compared to members of other orders.

2.1 Taxonomy

In order to classify objects, a general understanding of their taxonomy is required. A taxonomy is a hierarchical scheme used for classification, that originated from categorisation of organisms. Generally, a taxonomy includes description, identification, nomenclature and classification of groups of objects that share the same characteristic features [14]. This taxonomy represents the underlying classes learned by the classification algorithm. A simplified taxonomic tree for insect classification can be found in Figure 9. This paper focuses on the detection of five "Orders" of insects that are "Coleoptera" (beetles, Figure 1), "Lepidoptera" (butterflies, Figure 3), "Hemiptera" (true bugs, Figure 2), "Odonata" (dragon flies, Figure 4) and "Hymenoptera". To simplify the detection of the order "Hymenoptera" the family "Formicidae" (ants, Figure 5) was

selected. This simplification was done, because some Hymenopteras look similar to other orders to the human eye. This would cause issues in the classification task, i.e. the order Hymenoptera contains a wide variety of differently structured and sized insects ranging from ants to wasps as sampled in Figure 10.

2.2 iNaturalist

The iNaturalist Species Classification and Detection Dataset (iNat data set) is a collection of image files in combination with their class labels. It is a CVPR [7] competition data set from 2021 and was originally build in 2017 with the objective to represent the unpredictable distribution of nature in regards to species and abundance [8].

2.2.1 Data set definition

The full 2021 data set consists of approximately 2.8M images of 10,000 different species with annotated class labels [15]. In contrast to the original 2017 data set, the 2021 competition data does not contain annotations for Bounding Boxes. 2.7M of the image samples are combined into the public training and 100,000 into the public validation set. The public training set data set is intended to be used during the training phase of an ML Model, which will be discussed in a later section (4.2). The validation set is used to validate the predictive power of the current Model during this training phase. In total, the data set contains 2,526 individual species of the class "Insecta". It is organized in a directory structure, were directories are named by a unique identifier (UID) for each species and its taxonomic name, e.g. `02420_Animalia_Arthropoda_Insecta_Odonata_Euphaeidae_Euphaea_formosa`. These directories hold the image files. All files are in *JPEG* file format and range from 6,0 kB to 105 kB in file size. The data set was collected based on data available on the website iNaturalist.org [8]. iNaturalist is a citizen science effort, every person can register on the website and publish or edit observations. This means either uploading images or labeling existing images on the platform.
Due to the fact that not every citizen is an expert in biology, specifically in the taxonomy of *all* organisms, it is presumable that the annotation data contains inaccurate species labels, albeit the data set has been reviewed by multiple citizen scientists. Due to the power of the numbers involved, it is nearly impossible to have a 100% accurate set of labels for almost 3M images. By only considering the scale, for every image there is a big chance of choosing a wrong species label. Similar, the probability of selecting a wrong species label for the insects order is considered high, as mentioned in "Taxonomy". But for the purpose of this work, it can be assumed that the order specification is in fact accurate, based on the argument that the species taxonomy is more fine grained than the order taxonomy and therefore harder to apply.
The base data set used in this project is a reduced form of the original *iNat* public training data set that was provided by the team of *KInsecta*. It is prefiltered by the taxonomic class type "Insecta", insects. This base data set is hosted on

[7] Conference on **C**omputer **V**ision and **P**attern **R**ecognition hosted by the Computer Vision Foundation (`https://www.thecvf.com/`
[8] iNaturalist website: `https://www.inaturalist.org/`

a **N**etwork **A**ttached **S**torage (NAS) device that runs in the local network of BHT.

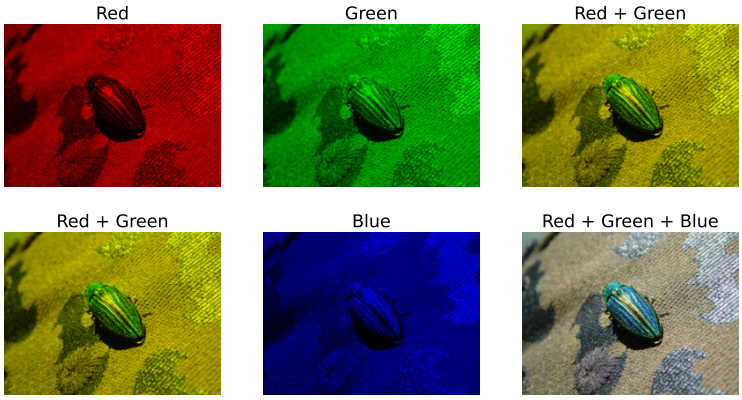

Figure 11: Image channels individually and combined. All channels combined result in the original image.

2.2.2 Image data format

Image files can vary significantly in file formats, file size and image aspect ratios. The here used images are in JPEG format and therefore have three channels. Let $I \in \mathbb{R}^{h \times w \times 3}$ be an image of height h, width w and with three channels, then a pixel within this image can be described as the vector $\boldsymbol{p} \in C^3$, for `uint8` data format with $C_{\texttt{uint8}} = [0, 255]$. When loaded the values are converted to `float32` with $C_{\texttt{float32}} = [0, 1]$. Each of the elements of $\boldsymbol{p}$ represent a color intensity for the corresponding channel. The three channels are a **R**ed, a **G**reen and a **B**lue channel. To recreate the image the channels are combined in their spacial dimension (Figure 11),

$$I = [R_I, G_I, B_I] \tag{1}$$

where R_I, G_I and B_I are the corresponding color-channel-element-$h \times w \times 1$-matrices with h rows and w columns respectively.
This simplifies working with images drastically, as they can be interpreted and used as a matrix. Due to this format a single pixel can easily be queried using its coordinates in the image $P = I(x, y)$, or e.g. $P_R = R_I(x, y)$ for the corresponding red value of this pixel. In the following, the computer vision (CV) notation of $P = I(y, x); P_R = R_I(y, x)$ will be used. This is a notation simplified for the image data format and is introduced in this context due to the way the data is stored on the computer: as an array, which is accessed by its height component first, e.g. `image[height][width]`. Additionally, the underlying coordinate system is a down-right positive Cartesian coordinate system, ranging from the origin, the top left corner of the image to the lower right corner.
Apart from the information stored in each pixel, a second important information is the height and width of the image. In order to process images further and to feed them into the Machine Learning algorithms mentioned earlier, the images

need to be rescaled into a unified size. The size was fixed here at $224px \times 224px$ [9]. For scaling purposes in this project the bilinear interpolation method is used, which is a two dimensional variant of linear interpolation and the default configuration of most Tensorflow methods [10]. An interpolation method was selected in order to not change the information in the input features. Filling the images with black pixels to maintain the original image aspect ratio, but artificially add more content to it, would add additional information to the input features and would distort the results.

2.3 Collection

As previously mentioned, the base data set $iNat_{\text{train}Insecta}$ is only a subset of the original *iNat*-2021 data set $iNat$. It contains 663,682 image files scattered unbalanced across 2527 insect species. The data set extracted for the experiments X is an evenly distributed subset of the nearly 660,000 insect image files and contains 500 samples per order. A simplified visualization of this set substitution can be found in Figure 12.
To reduce network traffic and prevent issues with connecting to the *NAS* server, some precocious preparations were made, i.e. the data collection pipeline, one of the Python script files in the accommodating repository of this paper, was developed on a very reduced subset, five pictures, of the overall data set. The script allows random selection of n species per order and m samples for each of them by providing the desired parameters accordingly. A storage directory is created as a temporary storage directory for simplified usage during "Manual labeling" and was required to solve the issue of changed file paths during the upload phase into the labeling tool.

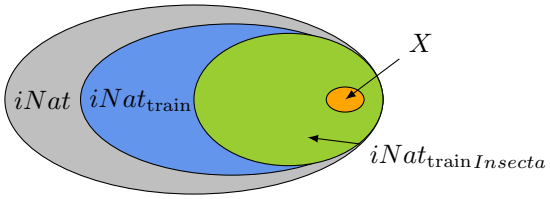

Figure 12: Venn diagram of the used data sets $DS \subset iNat_{\text{train}Insecta} \subset iNat_{\text{train}} \subset iNat$, where X is the data set used in this project.

2.4 Manual labeling

For the localization task of this paper an extended data set containing locations of insects, in each image, is required. As representation for these locations Bounding Boxes (BBs) are used. BBs are rectangular boxes, a subset of pixels of the original image, that contain an object. Each BB is represented as

[9] The scale $224px \times 224px$ was used in the VGG-16 paper and was here adopted to achieve best results with this architecture.
[10] Tensorflow documentation for the image resizing function: `https://www.tensorflow.org/api_docs/python/tf/image/resize`

coordination vector in the form

$$\boldsymbol{c^{(m)}} = \left(y_{\min}^{(m)}, x_{\min}^{(m)}, h^{(m)}, w^{(m)}\right)^T \tag{2}$$

where $y_{\min}^{(m)}$ and $x_{\min}^{(m)}$ are the coordinates of the upper-left corner of the m-th BB in a data set, and $h^{(m)}$ and $w^{(m)}$ height and width respectively. This turned out to be less error prone in further usage [11]. In order to make the annotated BBs available for later use in the *KInsecta* team LabelStudio [16] was used. LabelStudio is a self hosted open source labelling tool managed by *KInsecta* within the BHT. Using LabelStudio the data set of the 2500 pre-selected *iNat* image files was manually labelled. Figure 14 gives an overview on how the interface of Label Studio looks like. The BBs can be exported from LabelStudio in JSON file format and are stored in a dictionary-like fashion, where each image file is linked to the manually created BB annotations. An example for the representation can be seen in "LabelStudio JSON format". In order to get an understanding of the manually generated data a final area distribution for the created BBs can be seen in Figure 13. This will be useful for the following section, in which the data gets preprocessed, augmented and wrapped into a final data set, that is used for the rest of the project.

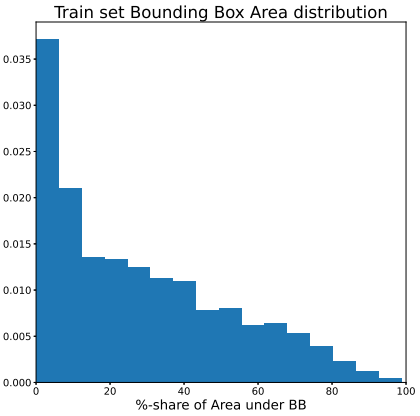

Figure 13: Distribution of share of area under BBs.

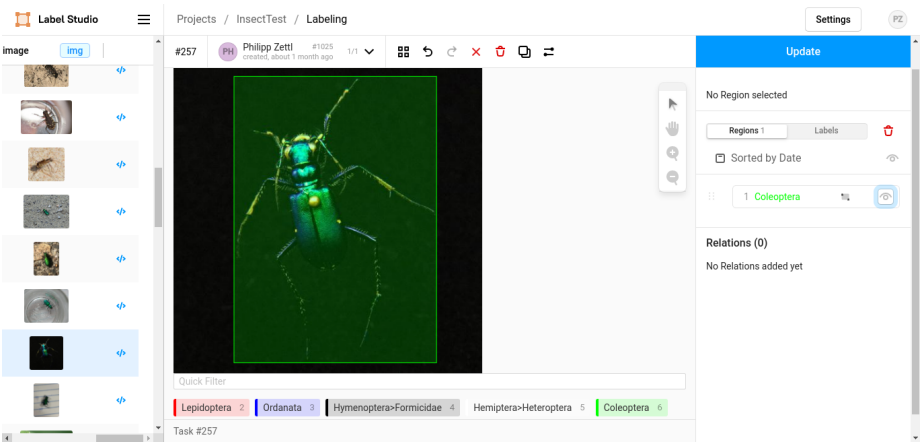

Figure 14: Annotation-View of LabelStudio: Bounding Boxes can be selected using Drag&Drop mechanics; Each BB is labelled with a corresponding order-label.

[11] A more detailed description will follow in the section "Roadblocks".

3 Data preprocessing

After the data is collected and annotated an accommodating data set needs to be formed. The following section will describe techniques to build the core data set, that is used to train, validate and test the ML Methods.

3.1 Data sets

Due to the type of data applied in this paper and to allow a simplified plug and play solution for the preselected image files, a custom data loading pipeline was created. The data set handler loads image data on demand into the RAM instead of loading all available training samples at once. This allows flexibility and reduces the required amount of RAM during the training phase.
Additional to the data loader a simplified storage format was introduced. Again, using the JSON file format, to allow human interaction and simplify debugging tasks, the structure of the storage is described within a file called "`dataset-structure.json`". Each of these files represents the underlying file structure and holds the attributes "train", "test" and "validation" representing data sets, "labels" a list of all available classes to learn from the data set and "created_at" a timestamp when the file was created. The attributes "train", "test" and "validation" represent the three subsets of the whole data set and are used for their corresponding purposes. The "train" set is used to train, or fit the ML models, the "validation" set to validate the predictive power of the method during the training phase. And the "test" set will be processed at the end of the project pipeline to evaluate the predictive power of the resulting model. The distributions of these three sets can be found in Figure 15. For the purpose of this paper three data loading methods were implemented [12], yielding different types of samples. Each loading method scales the images to a provided size, as previously mentioned here we are using $H \times H$ with $H = 224px$. Before processing each of the images gets transformed into an flattened vector $\boldsymbol{x} \in \mathbb{R}^N$ with $N = 224 \cdot 224 \cdot 3 = 150528$. The following subsections will discuss the underlying structure and generation process of these samples for each method respectively.

3.1.1 Two Stage Regression

For the regression tasks, the data set was built out of the original images available in the iNaturalist competition data set as described in "Manual labeling". Apart from manually labeling no further optimizations have been performed on the data set. Each of the M samples $S_r^{(m)}$ in the data set X has the form

$$S_r^{(m)} = (\boldsymbol{x^{(m)}}, \boldsymbol{c^{(m)}}) \tag{3}$$

where $\boldsymbol{x^{(m)}} \in \mathbb{R}^N$ is the m-th input feature vector and $\boldsymbol{c^{(m)}}$ the corresponding ground truth BB. This data set is available using the module's `load_directory_to_regression_dataset` method.

[12] The methods are available in the module `src.data.load_dataset`

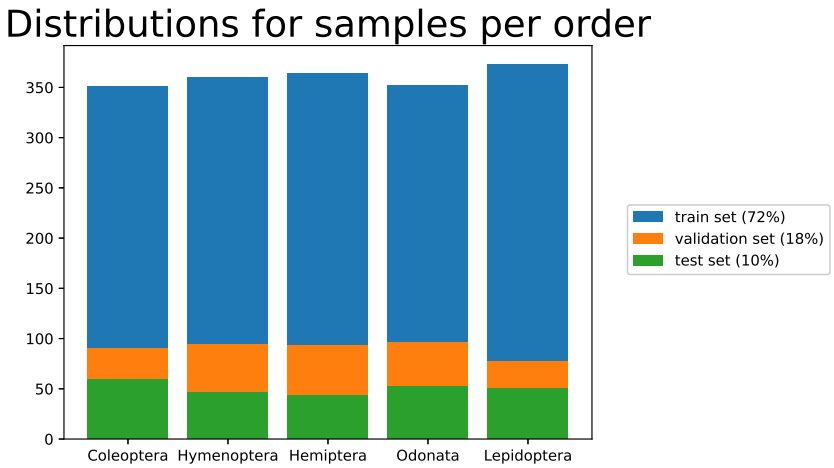

Figure 15: Distributions of training, validation and test set.

3.1.2 Two Stage Classification

The classification method will process full image samples for the **single-stage-method** approach. But when applied using the **two-stage-method** approach the method will only process cropped images during inference time. Hence, a cropped version of the regression data set was generated using the ground truth labels form the Bounding Box regression data set as the mask for cropping.
Let $h : \mathbb{R}^{H\times H\times 3} \times \mathbb{R}^4 \to \mathbb{R}^{H\times H\times 3}$ be the cropping and resizing function

$$h(\boldsymbol{x^{(m)}}, \boldsymbol{c^{(m)}}) = \boldsymbol{x_c^{(m)}} \tag{4}$$

where $\boldsymbol{x_c^{(m)}}$ maintains the size of the input $\boldsymbol{x^{(m)}}$[13]. Then one data set yields elements of the form

$$S_c^{(m)} = (h(\boldsymbol{x^{(m)}}, \boldsymbol{c^{(m)}}), \boldsymbol{y^{(m)}}) = (\boldsymbol{x_c^{(m)}}, \boldsymbol{y^{(m)}}) \tag{5}$$

where $\boldsymbol{y^{(m)}} \in \mathbb{R}^K$ is the ground truth class label of the m-th sample and the other data set with full images

$$S^{(m)} = \left(\boldsymbol{x^{(m)}}, \boldsymbol{y^{(m)}}\right) \tag{6}$$

accordingly. The label vector $\boldsymbol{y^{(m)}}$ is a *one-hot-encoded* vector. This means, to represent class k the vector with $y_k^{(m)} = 1$ and $y_i^{(m)} = 0$ for $i = 1, \ldots, K; i \neq k$ is defined. The classification data set can be generated using the `load_directory_to_classification_dataset` method.

3.1.3 Single Stage Prediction

The previous two methods are only usable when solving tasks using the **two-stage-method** approach. For **single-stage-methods** an additional data set

[13]Note, in the following $\boldsymbol{x^{(m)}}$ is used to represent an image tensor $\boldsymbol{m^{(m)}} \in \mathbb{R}^{H\times H\times 3}$, sometimes it is used for its flattened version $\boldsymbol{x^{(m)}} \in \mathbb{R}^N$.

is required that yields samples with both types of labels

$$S_{\text{2-in-1}}^{(m)} = \left(\boldsymbol{x^{(m)}}, \left(\boldsymbol{y^{(m)}}, \boldsymbol{c^{(m)}}\right)\right) \quad . \tag{7}$$

This is required to evaluate both tasks simultaneously on the same input, while training the model. A data set with these samples is available using the `load_directory_to_two_in_one_dataset` method.

3.1.4 Notation

In the following chapters the notation

$$X = \begin{pmatrix} \boldsymbol{x^{(1)}} \\ \vdots \\ \boldsymbol{x^{(m)}} \\ \vdots \\ \boldsymbol{x^{(M)}} \end{pmatrix}, Y = \begin{pmatrix} \boldsymbol{y^{(1)}} \\ \vdots \\ \boldsymbol{y^{(m)}} \\ \vdots \\ \boldsymbol{y^{(M)}} \end{pmatrix}, \tag{8}$$

will be used. With X the data set consisting of M samples and Y the labels in shapes as described in (3), (6), or (7).

3.2 Augmentation

Data augmentation can help to prevent a model from over-fitting the training data. Over-fitting is one of the obstacles when working on an ML project that involves Neuronal Networks, which will be introduced later in section "Neural Networks (NNs) and Deep Learning (DL)". It is the phenomenon of an algorithm predicting outputs for the training data very well, it fits too well on the training set, but performing poorly whenever it faces unknown data and hence, does not generalize the given task well. The issue of *over-fitting* can be overcome by simplifying the underlying model, cleaning the training data set, or increasing the amount of samples in the train set. Now we will look into increasing the number of samples, later in section 4.8 so called *regularization*-methods get introduced that also help resolving the issue of over-fitting, by simplifying the function described by the model. For this paper, augmentation methods have been implemented from the beginning, as state-of-the-art architectures perform better on bigger data sets, compared to small ones with few samples. All augmentation methods can be summarized by the function P. It is crucial to note that none of the augmentation methods shall change the input format and dimensions. Therefore, let $P : \mathbb{R}^N \times \mathbb{R}^K \to \mathbb{R}^N \times \mathbb{R}^K$ be any augmentation function for the regression sample

$$P\left(\boldsymbol{x^{(m)}}, \boldsymbol{c^{(m)}}\right) = \left(\boldsymbol{x_A^{(m)}}, \boldsymbol{c_A^{(m)}}\right) \tag{9}$$

or

$$P\left(\boldsymbol{x^{(m)}}, \boldsymbol{y^{(m)}}\right) = \left(\boldsymbol{x_A^{(m)}}, \boldsymbol{y_A^{(m)}}\right) \tag{10}$$

respectively for classification samples. Common augmentation techniques for computer vision tasks include geometric transformations, e.g. flipping, stretching and rotating of the original sample, as well as those changing pixel values

by adding noise or performing color value transformations on images. It is important to note here that augmentation should only be applied to the training set. E.g. applying augmentation techniques to the validation set will change the underlying feature distribution and might not represent the expected data during inference time. Hence, the model would converge to an alternate solution based on the augmented data and will most likely not perform on the same quality once it is deployed and used for real data. All the here used augmentation techniques were implemented from scratch and can be found in the `src.data.augmentation` module of the repository. The augmentation is intended to be used through the `DataAugmentationHelper` class, that sequentially yields augmented samples in the same way the previously introduced data loaders do.

3.2.1 Regression

For the regression task, the following augmentation methods have been applied:

1. changing contrast values
2. Rotate by 90, or 270 degrees
3. Flipping
4. Cropping

Each of these techniques can be initialized with a probability value that will be used to determine which method is used while yielding new samples. Whereas 2, 3 and 4 require the underlying BB-coordinates to perform the augmentation and return a new set of BB-coordinates, 1 only alters the input features and leaves the BB labels unchanged. Whenever a method, e.g. rotation, changes logically the shape of the image, the image is resized into its original shape. Examples for all implemented augmentation methods can be seen in Figure 37. To not generate the same augmented samples in each iteration, the methods are called using random numbers out of a uniform distribution. For simplification, the helper class iterates through a list of pre-configured desired operations and compares a new uniform distributed random value with the probability of the method. This can lead to not selecting a method, which is desired so the resulting data set contains the original samples as well. A plot of the distribution for the used seed [14] can be found in Figure 16.

3.2.2 Classification

As previously discussed, the data set for the classification task yields samples with a reduced, cropped, input image. Due to that fact not all augmentation techniques used for the regression data set can be applied to the classification data set, i.e. the images can not be cropped further. This reduces the available augmentation techniques for the classification set by 4 cropping. The resulting distribution for the augmentation techniques used here can be seen in Figure 17.

3.3 Filtering

Using filters on images can improve the quality of prediction of an algorithm as presented in the initial paper of VGG [5]. A preprocessing color correction filter

[14] The seed was initially mentioned in the abstract, for the experiments in this paper the seed 42 has been used to generate "random" numbers in the algorithms.

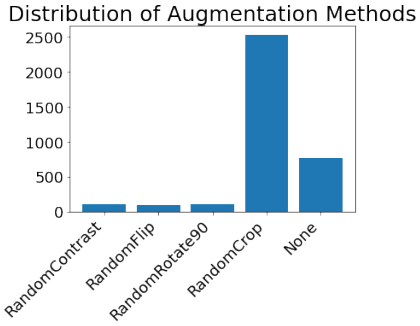

Figure 16: Distributions for the regression data set.

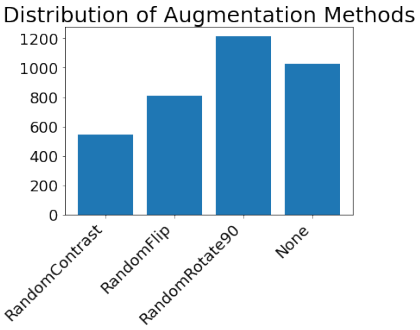

Figure 17: Distributions for the classification task data set.

is used in VGG-16 to potentially overcome the issue of having too dark images in the train set that don't represent the production data that is expected. Hence, when using a pretrained model, in the example of VGG-16 and MobileNet here both are using a base of existing weights [15], it is required to use the provided preprocessing method during inference time. Omitting the method would result in unpredicted input data and the method would respond with poor predictions. Examples for such preprocessing methods are visualized in Figure 18 and Figure 19.

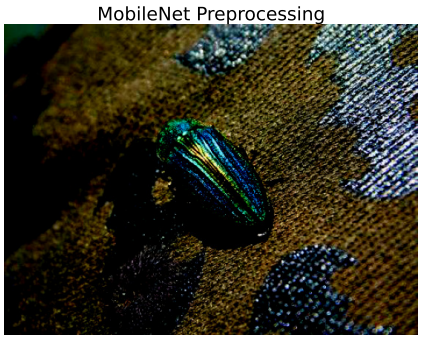

Figure 18: Image pre-processed for MobileNet

Figure 19: Resulting image of preprocessing method for VGG-16

3.4 Dimensionality Reduction

Sometimes the input data can not be resized, cropped or transformed using simple pixel wise operations. And sometimes there are too many input features available to use while fitting the desired method, which can cause issues regarding resource and performance constraints. For this reason a technique called "Dimenionality Reduction" is introduced, that reduces the dimensions of the input data.

[15] In the implementation this is done by providing a string value representing the name of the data set that was used to generate the weights, here `'imagenet'` weights are used (for more see the documentation of Keras-Applications: `https://keras.io/api/applications/`).

Let $R : \mathbb{R}^{N_i} \to \mathbb{R}^{N_o}$ be a reduction function with N_i the number of input features and N_o the number of output features [16]. For this reason a more sophisticated decision algorithm was applied, **P**rincipal **C**omponent **A**nalysis (PCA). PCA essentially extracts the most correlating components out of the training images, the principal components, and stores them in a format where the first component is the most correlating one and the last component is the component with the least correlation to others.[17] Let PCA_N be the PCA containing the N most correlating components of the training data set, then the function $R : \mathbb{R}^N \to \mathbb{R}^N, R\left(\boldsymbol{x^{(m)}}\right) \mapsto \boldsymbol{x}_R^{(\boldsymbol{m})}$ describes the transformation of an image onto these components, where $\boldsymbol{x}_R^{(\boldsymbol{m})}$ is the reduced form of $\boldsymbol{x^{(m)}}$. This gives a great advantage for some of the conventional methods, due to the fact that it is not required to consist of 150528 parameters. Unfortunately this results in a changed image that can not be displayed in a proper way, nor processed by a CNN-architecture, this will be discussed in the upcoming chapter. The following section will give an introduction to algorithms and architectures used to solve the tasks of this paper.

[16] Later, in other transformations will be no differentiation between N_i and N_o, all inputs are considered to be in the size N where N depends on the transforming method.

[17] For more information, theory and background see [13, 17].

4 Machine Learning Methods

Machine Learning Methods is a very broad term that over the last years, with the growing popularity of Neural Networks, got mixed with the more general term Artificial Intelligence. The chapter will address the theory behind the "Modeling", "Training" and "Evaluation" steps in the Machine Learning Project Pipeline (Figure 8). Starting with a brief introduction on conventional ML methods such as Linear Regression and Support Vector Machines (SVMs), then introducing algorithms that got very popular over the last few decades, such as Artificial Neural Networks and Convolutional Neural Networks (CNNs).
After discussing the "Modeling" aspect an outlook on the "Training" and "Evaluation" of these models based on "Losses" and "Performance metrics" is given. Then finishing with a short description on a method of finding right parameters for the selected models called "Hyper Parameter Optimization". The content of this chapter is a summary of the detailed explanations found in Aurélien Géron's "Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow" [13], combined with other books [18], lectures and paper resources.

4.1 Tasks

Using ML a wide range of tasks can be tackled, starting from predicting house prices in a specific area to detecting cancer based on blood test values.
As stated in "Data sets", the N components of the feature vector $\boldsymbol{x}^{(m)}$ used here for all tasks are pixel values from images. All of these tasks can be described using a function T.
Let T be the task function $T : \mathbb{R}^N \to \mathbb{R}^K, \boldsymbol{x^{(m)}} \mapsto \hat{\boldsymbol{y}}^{(\boldsymbol{m})}$, with K the number of output values of the prediction $\hat{\boldsymbol{y}}^{(\boldsymbol{m})}$ for the m-th input $\boldsymbol{x}^{(\boldsymbol{m})}$. A pixel $\boldsymbol{p}$ represents a color vector $\boldsymbol{p} = (p_R, p_G, p_B)^T$, with $p_R, p_G, p_B \in \{0, 1, \ldots, 255\}$. Images are two dimensional, they have a width w and a height h and most of the time their values differ between images. For all ML methods the input values need to be scaled to the same dimensions, as stated in "Image data format".

4.1.1 Classification

Classification is the task of predicting class labels $\hat{\boldsymbol{y}}^{(\boldsymbol{m})} \in \mathbb{R}^K$ for a given input $\boldsymbol{x}^{(\boldsymbol{m})} \in \mathbb{R}^N$. The classification task can be subdivided into binary, multilabel [18] and multioutput classification. Binary classification focuses on detecting the probability of a given input belonging to a class or not and multioutput classification predicts probabilities, with a total probability of 100%, for a predefined set of classes. Multilabel classification, on the other hand, predicts independent probabilities for multiple potential outputs, in contrast to multioutput the resulting probabilities do not necessarily sum up to 1.0. Here, multioutput classification is applied and described further.
As introduced, multioutput classification is the task of predicting a class out of K, here $K = 5$, given classes for an given input $\boldsymbol{x}^{(m)}$. Let T_c be the multioutput classification task function

$$T_c : \mathbb{R}^N \to [0, 1]^5, \boldsymbol{x^{(m)}} \mapsto \hat{\boldsymbol{y}}^{(\boldsymbol{m})} \tag{11}$$

with $\hat{\boldsymbol{y}}_k^{(\boldsymbol{m})}$ the predicted probability of class k for the m-th input $\boldsymbol{x}^{(\boldsymbol{m})}$.

[18] Multilabel classification is sometimes also called *multinomial* classification.

4.1.2 Regression

Regression, on the other hand, maps a given input onto one or many real numbers. As in "Manual labeling" introduced, the regression task of predicting BB coordinates requires an output vector $\boldsymbol{c^{(m)}} = \left(y_{\min}^{(m)}, x_{\min}^{(m)}, h^{(m)}, w^{(m)}\right)^T$, hence, a multi-output regression.
Therefore, let T_r be the regression task function predicting values in shape (2)

$$T_r : \mathbb{R}^N \to \mathbb{R}^4, \boldsymbol{x^{(m)}} \mapsto \boldsymbol{c^{(m)}} \quad . \tag{12}$$

4.2 Training Algorithms

Commonly used across the Machine Learning scene are supervised algorithms for the problem of localization and classification of objects in image data. Apart from supervised learning algorithms there are also unsupervised, reinforcement and semi-supervised learning algorithms. Supervised learning algorithms get trained on a stream of training samples, as in "Data sets" introduced. Here, the samples have the shapes (3), (6) and (7). During the training phase, the algorithm will compare the true values of each sample to the result of the prediction through the model for a given sample to its true value. This will be described in detail in "Losses".
Unsupervised learning algorithms on the other hand do not get trained based on sample pairs $S^{(m)}$, they get trained on just the input values $\boldsymbol{x^{(m)}}$.
Algorithms for **semi-supervised learning** rely on a mixture of labelled and unlabelled training data. A common example would be a group of people that is recurrently visible in images. In order to train the algorithm to detect specific people within different images, the annotator needs to provide a single labelled sample and the algorithm is capable of clustering the following images together in which those people appear.
Reinforcement learning algorithms on the other hand work completely differently to the (semi-/un-) supervised algorithms. For this kind of algorithms, an agent explores a given environment and learns policies by gaining rewards or penalties for actions. Along the way, the agent will eventually find the *cheapest* or *most rewarding* path of actions through the environment.
In the following, supervised learning algorithms will be discussed in detail, and a general overview given on how to train different kinds of supervised learning algorithms that are able to generalize the given task of localization and classification of insects in images.

4.3 Linear Regression

Linear Regression (LinReg) is one of the standard supervised learning methods most often employed in classification and regression tasks, here LinReg will be applied to predict approximated Bounding Boxes $\hat{\boldsymbol{y}}^{(\boldsymbol{m})} \in \mathbb{R}^4$ for an input $\boldsymbol{x^{(m)}} \in \mathbb{R}^N$ and therefore solve a mutli output regression task.
LinReg finds a linear function that models the training set. To perform a prediction of the k-th BB-coordinate of the m-th input – $\hat{y}_k^{(m)}$ – using a Linear Regression Model the weighted sum of the input values $\boldsymbol{x^{(m)}}$ needs to be com-

puted and added to a bias term $\omega_{0,k}$

$$\hat{y}_k^{(m)} = \omega_{0,k} + \omega_{1,k} \cdot x_1^{(m)} + \ldots + \omega_{N,k} \cdot x_N^{(m)} \tag{13}$$

with $x_n^{(m)}$ the n-th component of $\boldsymbol{x}^{(m)} \in \mathbb{R}^N$, $\omega_{0,k}$ the bias of the k-th output and $\omega_{n,k}$ for $n = 1, \ldots, N$ the corresponding weights. In vectorized form

$$\hat{y}_k^{(m)} = \left\langle \boldsymbol{\omega_k}, \boldsymbol{x^{(m)}} \right\rangle + \omega_{0,k} \tag{14}$$

with weights $\boldsymbol{\omega_k} = (\omega_{1,k}, \omega_{2,k}, \ldots, \omega_{N,k})^T$, $\boldsymbol{x^{(m)}} = \left(x_1^{(m)}, x_2^{(m)}, \ldots, x_N^{(m)}\right)^T$ and $\omega_{0,k}$ the bias.
To perform multivariate predictions with K outputs $Y \in \mathbb{R}^{M \times K}$ the form

$$Y = X \cdot W^{19} \tag{15}$$

can be used, with $X \in \mathbb{R}^{M \times (N+1)}$ the data set with 1's in the first column and $W \in \mathbb{R}^{(N+1) \times K}$ a matrix containing 1's in the first row and the elements of the column vectors $\boldsymbol{\omega_k}$, where each value represents the weights for one of the K outputs. For instance, multiplying X with the 3rd and 4th column in W would compute the predictions of the BB-height and width respectively for all instances in X. Note that LinReg, as well as the following "Support Vector Machines" and "Neural Networks" require an input vector $\boldsymbol{x^{(m)}} \in \mathbb{R}^N, N = H \cdot H \cdot 3$, hence the input features need to be converted into a vector. To measure how well a model generalizes on the training data so called loss functions are introduced. For LinReg this is MSE. MSE is a metric, that computes an averaged l_2-norm[20] over the data set X with M samples. To calculate the MSE for the predictions $\hat{Y} \in \mathbb{R}^{M \times K}$

$$\text{MSE}(Y, \hat{Y}) = \frac{1}{M} \sum_{m=1}^{M} \left\| \boldsymbol{y^{(m)}} - \boldsymbol{\hat{y}^{(m)}} \right\|_2^2 \tag{16}$$

with $\boldsymbol{y^{(m)}} \in \mathbb{R}^K$ the m-th ground truth BB and $\boldsymbol{\hat{y}^{(m)}}$ corresponding prediction by the model. It is the objective of a model to find a set of approximated weights $\hat{\boldsymbol{\omega}}$ that minimizes $\text{L}(Y, \hat{Y})$, for LinReg $L = \text{MSE}$.
An adjusted form of LinReg is Ridge Regression, by Hoerl and Kennard [19]. Ridge Regression was introduced to solve the task of multi class prediction, but can be used for regression tasks as well. The difference between Ridge and Linear Regression lies in the regularization parameter α that is added to the loss function computed to calculate the weights for the model

$$L(Y, \hat{Y}) = \text{MSE}(Y, \hat{Y}) + \frac{1}{2} \alpha \sum_{k=1}^{K} \sum_{n=1}^{N} (\omega_{n,k})^2 \tag{17}$$

where $\omega_{n,k}$ is the n-th weight of the k-th output. Note that $\omega_{0,k}$ the bias term is **not** part of the regularization penalty. The factor $\frac{1}{2}$ is introduced to simplify the gradient, that is later computed, by cancelling out the multiplication

[19] This system of linear equations can not be solved directly, but it can be solved using the method of least squares that results in $\hat{Y}$, an approximation.
[20] The l_2-norm is also be known as Euclidean norm or p_2-norm.

by 2 through the exponent. Another regularized version of LinReg is Lasso-Regression [13], it applies a different regularization term, which is also weighted by the parameter α

$$L(Y, \hat{Y}) = \text{MSE}(Y, \hat{Y}) + \alpha \sum_{k=1}^{K} \sum_{n=1}^{N} |\omega_{n,k}| \quad . \tag{18}$$

In this paper Ridge Reg is used for classification, to perform multi-class classification a **O**ne **v**ersus **R**est (OvR) method is employed to combine 5 individual RidgeReg models into a multi-class regressor.

4.4 Support Vector Machines (SVMs)

Another supervised learning method are SVMs, they can be applied for classification and regression. Other than Linear Regression, SVMs find a boundary, the *decision boundary* (DB), between the data to separate it, instead of trying to find a function that represents the data. The goal of an SVM is to find the hyper plane that generates the widest margin that does not contain objects of the training set. It does so by minimizing the length of the normal vector pointing to the closest of those objects. These closest objects are called *Support Vectors*. Figure 20) visualizes this for a binary classification example. The left plot shows three decision boundaries of three possible linear classifiers, whilst the purple and red line seem to separate the data, they most likely will fail to classify new samples because they are too close to the training samples. The green dotted line represents a bad classifier that does not separate the classes at all. On the right side the solid line represents the DB of a SVM classifier, that separates the classes and maintains the widest margin, visualized as the area between the DB and the dotted lines.
A prediction using a linear SVM can be performed using the decision function

$$\hat{\boldsymbol{y}}^{(\boldsymbol{m})} = \begin{cases} 1 & \text{if } \left\langle \boldsymbol{\omega_k}, \boldsymbol{x}^{(\boldsymbol{m})} \right\rangle + \omega_{0,k} > 0 \\ 0 & \text{else} \end{cases} \quad . \tag{19}$$

here $\boldsymbol{\omega_k}$ is the normal vector to the k-th hyper plane and $\omega_{0,k}$ the corresponding bias. Describing SVMs further goes beyond the scope of this paper, for more details and how SVMs work see "Chapter 5: SVMs" in [13]. Here multiple linear SVMs are applied to solve the task of multioutput classification, again using a OvR method.

4.5 Gradient Descent (GD)

To solve the optimization problem and find a set of weights $\hat{\boldsymbol{\omega}}$ that minimizes the loss L the one-step-algorithm of Gradient Descent can be used. GD is an optimization algorithm that can solve many different tasks, but its main objective is to gradually search for the global minimum of a function's landscape, although it often lands in a local minimum. It does so by taking small steps, configured by the *learning rate* η, and computing the *loss* of the underlying model in each iteration step until it converges to a solution that is satisfying according

[22] From [13] Figure 5-1, Page 154.

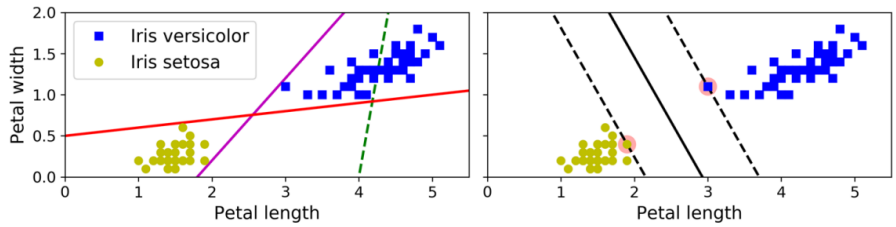

Figure 20: [22]Maximum margin classification for two classes. On the left side three potential decision boundaries. One that separates the data badly (green), and two (purple + red) which divide the data into two classes, but very strictly. The right side shows the DB of an SVM with its margin to the support vectors (pink).

to Performance metrics, or when reaching the maximum number allowed iterations that were configured. This happens to be working very well with *convex* functions, like MSE. This implies that the function only has a global minimum that GD will guarantee to approximate. To perform a GD-step, first the partial derivative of the loss function with respect to each model weight ω_j

$$\nabla_{\boldsymbol{\omega}}\text{MSE} = \begin{pmatrix} \frac{\partial}{\partial\omega_1}\text{MSE} \\ \frac{\partial}{\partial\omega_2}\text{MSE} \\ \vdots \\ \frac{\partial}{\partial\omega_J}\text{MSE} \end{pmatrix} \tag{20}$$

is computed, with $\boldsymbol{\omega}$ the current weights and $J = (N+1) \cdot K$ the total number of weights. Then the weights updated accordingly

$$\boldsymbol{\omega}^{(i+1)} = \boldsymbol{\omega}^{(i)} - \eta\nabla_{\boldsymbol{\omega}}\text{MSE} \tag{21}$$

here are $\boldsymbol{\omega}^{(i)} \in \mathbb{R}^J$ the current weights. But GD has two main issues: selecting a too small learning rate η will take a very long time to converge, and using a very big value for η might cause convergence or divergence because of overshooting the minimum.

4.6 Stochastic GD (SGD)

As mentioned, GD requires to compute the gradients of all weights for all training samples, which is not very fast, nor resource saving. An improvement to GD is the *Batched* GD. Instead of calculating the models loss based on the full training set X with its M elements, the GD step is performed on a random subset of the training set, called a *mini-batch*. This decreases the number of gradients to compute, but decreases the accuracy in each step. Another improvement to this is SGD. Unlike GD, SGD chooses a random sample in each iteration step to compute the gradients. This allows increasing the training data set without an increase in time spent computing those gradients, because of, the random nature of SGD, but unfortunately it requires more iterations. Just like SGD, there are many improvements to the original GD algorithm,

such as **ADAM** [20], adaptive moment estimation, a more sophisticated improvement that increases the convergence time of SGD. It does so by leveraging the momentum in the gradients to increase or decrease the current learning rate η_i in iteration i [21].

4.7 Neural Networks (NNs) and Deep Learning (DL)

Artificial Neural Networks, short ANNs, are another method of ML to solve a wide range of tasks. They originated around the early 1940s from the Neuron algorithm by Warren McCulloch and Walter Pitts [22]. Since that publication, many improvements to the original algorithm were proposed, including Frank Rosenblatts *Perceptron* algorithm. Perceptrons are based on TLUs (threshold logic units). Again, to achieve multivariate predictions each output $\hat{y}_k^{(m)}$ receives a TLU. Let $\text{TLU} : \mathbb{R}^N \to (0, 1)$ be the formula describing the k-th TLU

$$\text{TLU}\left(\boldsymbol{x^{(m)}}\right) = f\left(\left\langle \boldsymbol{\omega_k}, \boldsymbol{x^{(m)}} \right\rangle + \omega_{0,k}\right) \hat{y}_k^{(m)} \tag{22}$$

with the dot-product $\left\langle \boldsymbol{\omega_k}, \boldsymbol{x^{(m)}} \right\rangle$, computed from $\boldsymbol{\omega_k}$ the k-th weight vector and the input sample $\boldsymbol{x^{(i)}}$, the bias term $\omega_{0,k}$ and f a step function. The step function f is also called *Activation Function* and could be, in the case of binary classification, a function like

$$f\left(\boldsymbol{x^{(m)}}\right) = \begin{cases} 1 & \text{if } \left\langle \boldsymbol{\omega_k}, \boldsymbol{x^{(m)}} \right\rangle + \omega_{0,k} > 0 \\ 0 & \text{else} \end{cases} \tag{23}$$

A Perceptron consists of one, or many, TLUs that are combined into a *Layer*. A layer is a collection of TLUs that are not connected to each other, but each input of a Layer is connected to one TLU. As Figure 21 shows, the weighted sum of the input data $\boldsymbol{x^{(m)}}$ gets fed into a Activation Function that results in the Perceptron's output. The vector of input values is also referred to as *input Neurons*. Multiple Perceptrons can be connected in a series where each follow-

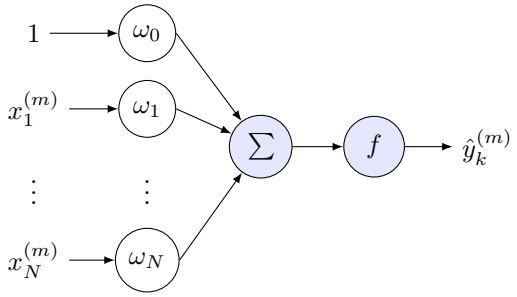

Figure 21: Visualization of a Perceptron with $N + 1$ inputs computing their weighted sum with the current weights $\boldsymbol{\omega}$ and outputting the result of the activation function f.

ing Layer of Neurons is fully connected to the Neurons of the previous Layer This is called short a *fully-connected Layer* or *Dense Layer*. Every layer, apart from the *Input-Neurons* and the last Layer, the Output-Layer, is called a *hidden* Layer. Multiple Layers of Perceptrons are called a **M**ulti-**L**ayer-**P**erceptron

(MLP). Apart from the Perceptron introduced here, different Neuronal architectures exist where Neurons connect in different ways to achieve different results. Because of the way the Neurons connect between each other, one or many connected Layers are called a Neural Network (NN). An NN with *many* [23] Layers is called a deep Neural Network. The parameters defining an NN architecture, such as number of Neurons in each layer, the number of layers, or the step-function f used in each of the layers, are called Hyper-Parameters. One part of developing an NN architecture is finding the right Hyper-Parameters ("HPO"). This will be discussed at the end of this chapter. Due to the fact that the connections in NNs grow with each of the layers in the network, the complexity in the partial derivatives grows. A different method, other than calculating the gradients of each training sample in a GD-step (21), is required that uses allows automatic computation of these gradients.

4.7.1 Backpropagation

Backpropagation (BP) is an iterative algorithm that is used to calculate the gradients during a GD-step for a Neural Network when solving a supervised learning task. Backpropagation is a version of GD designed specifically for optimization problems solved via Deep NNs. In one forward and one backward pass BP uses numerical differentiation to compute all gradients required to update the weights. The forward pass consists of performing a prediction for a sample and calculating the corresponding loss of it. Then in the backward pass the BP propagates the gradients back through the network while performing the GD-step (21). After adjusting all weights accordingly through the network the process is repeated until the algorithm converges.[24]

4.7.2 Activation Function

Different tasks demand different activation functions, apart from the previously introduced activation function (23). In this paper, two activation functions are implemented: ReLU and Softmax, but there are many different others[25].
ReLU
Let $\text{ReLU} : \mathbb{R}^N \to [0, \infty]$ be the ReLU-Activation Function

$$f\left(\boldsymbol{x}^{(m)}\right) = \max\left(0, \boldsymbol{x}^{(m)}\right) \tag{24}$$

Then ReLU is scale invariant and compared to the logistic function, also called "Sigmoid"

$$f\left(\boldsymbol{x}^{(m)}\right) = \frac{1}{1 + \mathrm{e}^{-\boldsymbol{x}^{(m)}}} \tag{25}$$

faster in converging for random weights because it turns off neurons with negative weights, due to its codomain. This becomes clearer through visualization, see Figure 22.

[23] The definition of *deep* NNs is very vague, in literature it is referred to as *many* Layers [13].
[24] Youtube video of Grant Sanderson on the BP topic: "What is Backpropagation really doing?" `https://youtu.be/Ilg3gGewQ5U`.
[25] Other activation functions available in Tensorflow can be found at `https://www.tensorflow.org/api_docs/python/tf/keras/activations`.

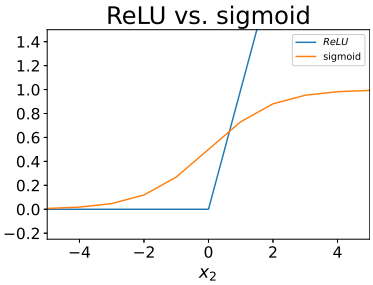

Figure 22: Visualized comparison of ReLU (blue) and Sigmoid (orange). Sigmoid is evenly balanced between positive and negative values, whereas ReLU sets all values $f\left(\boldsymbol{x^{(m)}}\right) < 0 \Rightarrow f\left(\boldsymbol{x^{(m)}}\right) = 0$.

Softmax

The previously mentioned *Sigmoid* is used for binary classification with NNs. The Softmax activation function allows for the classification task the usage of N neurons in the output layer to compute probabilities of N classes. This is done by first computing the score $s_k(\boldsymbol{x^{(m)}})$ of the input for each class k, then estimating the probability for each class by applying the Softmax function. Let $f : \mathbb{R}^N \to [0,1]^K$ with $B = [0,1]$ be the Softmax, or unit, activation

$$f_k\left(\boldsymbol{x^{(m)}}\right) = \frac{\mathrm{e}^{s_k(\boldsymbol{x^{(m)}})}}{\sum_{j=1}^{K} \mathrm{e}^{s_j(\boldsymbol{x^{(m)}})}} \tag{26}$$

where s represents the scoring function

$$s_k(\boldsymbol{x^{(m)}}) = \left\langle \boldsymbol{\omega_k}, \boldsymbol{x^{(m)}} \right\rangle + \omega_{0,k} \tag{27}$$

with $\boldsymbol{\omega_k}$ the k-th row-vector of W and $\omega_{0,k}$ the k-th bias. To then extract the class with the highest probability, the argmax method processes the output.

4.7.3 Regularization

To overcome the problem of overfitting the training data and improve convergence speed or overall generalization, Regularization methods can be applied to the NN.

Kernel Regularization

One idea is to apply Kernel Regularization. Similar to the improvement of Ridge Regression over Linear Regression, the Kernel Regularization is essentially a mechanism to apply penalties to the weights of the Layer. This causes the underlying model to have fewer degrees of freedom and therefore it gets harder for it to overfit the training data. Using the l_2-regularization method for kernels in each training step the regularization term

$$L_r = \frac{\alpha}{2} \cdot \sum_{j=1}^{J} \omega_j^2 \tag{28}$$

gets calculated to then be added to the overall, and final, loss of the model for the iteration step.

Dropout

Another regularization method is Dropout. Dropout is the method of randomly *turning Neurons off*, which means, literally, to set the weight of several Neurons to $\omega_i = 0$, during training. This allows the model to be more robust during inference time and it proves to yield a more generalized solution [23].

Batch Normalization (BN)

BN was introduced in the a paper by Sergey Ioffe and Christian Szegedy [24]. It allows to configure much higher learning rates and be less careful about initialization. It also acts as a regularizer, in some cases eliminating the need for Dropout. Let $BN : \mathbb{R}^{B\times N} \to \mathbb{R}^{B\times N}$ be the BN transformation, with B the batch size and N the number of input features

$$\hat{x}_n^{(m)} = \frac{x_n^{(m)} - \mu^{(m)}}{\sqrt{\sigma_n^2 + \epsilon}} \tag{29}$$

with

$$\mu^{(m)} = \frac{1}{B}\sum_{n=1}^{B} x_n^{(m)}, \quad \sigma_n^2 = \frac{1}{B}\sum_{n=1}^{B}\left(x_n^{(m)} - \mu^{(m)}\right)^2 \tag{30}$$

where $\mu^{(m)}$ is the mean over the B element batch and σ_n^2 the variance in the n-th feature.

Early Stopping

A different way of regularization of the iterative learning algorithm is to stop the training process once the predictive power of the model reaches a minimum accepted value threshold.

Learning Rate Schedule

Another regularization method that is not directly applied to the training samples is called a learning rate schedule. It adjusts the learning rate during the training process while following predefined rules. Learning Rate Schedulers can be split into the groups of

1. Power scheduling
2. Exponential scheduling
3. Piecewise constant scheduling
4. Performance scheduling
5. 1cycle scheduling

For this paper (4) was selected. It measures the validation error of the model every s steps and reduces the learning rate η by a factor of λ when the error stops decreasing.

4.8 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks, short CNNs, are one type of architecture of feed forward NNs. In contrast to ANNs, CNNs are capable of processing spatial data more suitably. Due to this fact, CNNs are very popular for CV tasks. They consist of layers, just like regular NN architectures, but with a twist. Instead of connecting the input in each layer with neurons, CNNs connect the

input neurons with *feature maps* using *receptive fields*, also called kernel filter masks. Let $Y^{(l)} \in \mathbb{R}^{h^{(l)} \times w^{(l)}}$ be a receptive field were $h^{(l)}$ is the height and $w^{(l)}$ the width of the l-th kernel in a layer. To generate the feature map for a conv. layer, $Y^{(l)}$ slides over regions of pixels, which is why it is called a sliding window, extracting features. The step size used for sliding is called *stride* and has a height and width parameter. Additionally, a padding can be added to the input, which increases its size, but allows the kernel to process the borders as well. The kernel size and the stride are both Hyperparameters. Let a kernel in a conv. layer be described as $C^{(l)} : \mathbb{R}^{w_i^{(l)} \times h_i^{(l)}} \to \mathbb{R}^{w_o^{(l)} \times h_o^{(l)}}$ with $w_i^{(l)}$ and $h_i^{(l)}$ the width and height components of the input and $w_o^{(l)}, h_o^{(l)}$ and d_o number of kernels, width height and depth components of the output, respectively. Figure 23 visualizes this process, called a convolution [26]. To use

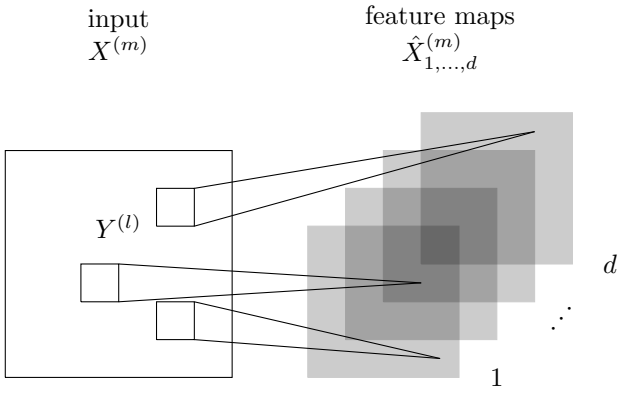

Figure 23: Visualization of the functionality of a conv. layer. For each output feature map a kernel slides over the input to extract the features.

CNNs to solve a specific task, such as classification or object detection, a fully connected multi layer Perceptron architecture processes the extracted features from the CNN to solve a specific task. In this paper this MLP is called a task-solving-head. The architectures implemented here and task-solving-heads are described further in "Chosen Model Architectures". A more detailed description and explanation of how the algorithm behind CNNs works can be found in [25]. Also, the parameters to configure a CNN, e.g. the kernel size, the number of conv. layers, a regularization for the kernels or a dropout, are Hyper Parameters. Let $C^{(l)} : \mathbb{R}^{H \times H} \times \mathbb{R}^{h \times w} \to \mathbb{R}^{(H-h) \times (H-w)}$ be the function of the l-th kernel in a layer that computes a convolution for a 2D input $X^{(m)} \in \mathbb{R}^{H \times H}$ [27]. Then the full image $\hat{X}$ can be computed looping over each of the input elements $X_{i,j}$ by iterating with row index $i = 1, \dots, H - h$ and column index $j = 1, \dots, H - w$

[26] A great explanation on how convolution works can be found in a video by the University of Nottingham (Computerphile) called "How Blurs & Filters Work - Computerphile", or the MIT lecture of Grant Sanderson "Convolutions in image processing".

[27] Note, this input has a single channel, like a feature map, or a single color channel in an RGB image.

and calculating

$$\hat{X}_{i,j}^{(m)} = \sum_{s=1}^{w} \sum_{r=1}^{h} X_{i+r,j+s}^{(m)} \cdot Y_{r,s}^{(l)} \tag{31}$$

here $X^{(m)}$ represents the m-th input matrix and $Y^{(l)}$ the l-th kernel of a layer. This results e.g. in the first feature map of Figure 23.

Pooling

Pooling can be used to reduce the shape, or the number of extracted feature maps of a CNN. They can either reduce the height, with or depth components h, w, d, or reduce the number of extracted feature maps, which is called global pooling.

Let $Pool : \mathbb{R}^{k \times h_i \times w_i \times d_i} \to \mathbb{R}^{k \times h_o \times w_o \times d_o}$ be a function with k the number of feature maps, h_i, w_i, d_i the dimensions of each input feature map and h_o, w_o, d_o the dimensions of the output feature maps respectively.

Global pooling combines neighboured feature maps and computes the average or max values accordingly.

Let $Pool_G : \mathbb{R}^{n_i \times h \times w \times d} \to \mathbb{R}^{n_o \times h \times w \times d}$ be a global pooling function with n_i the number of input feature maps, n_o the number of output feature masks and h, w, d as previously given.

To create a Deep CNN, different combinations of conv., pooling, BN layers, or layers with activation functions such as ReLU contain, can be stacked onto each.

4.9 Losses

Loss functions are a class of functions which map an input with one or many values to a real number.

Let L be the loss function given as $L(Y, \hat{Y}) : \mathbb{R}^{M \times K} \times \mathbb{R}^{M \times K} \to \mathbb{R}$ with the predicted labels $\hat{Y}$ and the ground truth values Y.

During the training process, an optimization algorithm is used to adjust the model parameters in a way that minimizes the output of the loss function. It does that by calculating the losses of current predictions in each step and uses their outputs to update the model parameters. This makes the model eventually converge to a set of parameters, which map given input values X onto the expected output Y so that $\min L(Y, \hat{Y})$. The following descriptions contain functions that calculate the loss of in a single prediction.

4.9.1 Classification

Logistic Loss (Cross-Entropy)

For the classification task the most commonly applied function is the Logistic Loss, or Cross-Entropy [13]. The Cross-Entropy between two probability vector collections Y the true label and $\hat{Y}$ the prediction, for K classes, can be computed as

$$CE(Y, \hat{Y}) = -\sum_{m=1}^{M} \left\langle \boldsymbol{y^{(m)}}, \log \hat{\boldsymbol{y}}^{(\boldsymbol{m})} \right\rangle \tag{32}$$

This loss measures how distinguishable two probability are from each other. The negative sum ensures that the loss decreases, when the distributions are

closer to each other.

Categorical Cross-Entropy

Categorical CE, or softmax loss, is a combination of CE(32) and (26). For Categorical CE it is required that the activation function of the layer prior to the output neurons is Softmax. Here $\hat{\boldsymbol{y}}^{(\boldsymbol{m})}$ and $\boldsymbol{y}^{(m)} \in \{0,1\}^K$ are *one-hot* encoded, as described with (5). Other than this requirement Categorical-CE (32) is CE, the process is displayed in Figure 24.

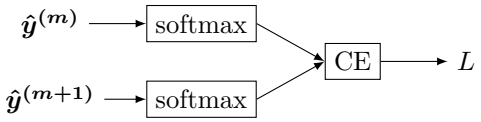

Figure 24: Visualization of Categorical-Cross-Entropy Loss. For each prediction softmax is computed, then the CE-loss over all predictions.

4.9.2 Regression

The Loss functions to solve the regression task here can be split into a group of functions that compute distance based values, i.e. comparing vector values, and a group of functions that compares two dimensional objects, Bounding Boxes. The former group consists of metrics using the commonly known l_1- and l_2-norm, the latter is based on **I**ntersection **O**ver **U**nion, a geometric metric.

Mean Average Error (MAE)

MAE is a metric using the l_1-norm, it is computing its mean average, and given by

$$\mathrm{MAE}\left(Y, \hat{Y}\right) = \frac{1}{N} \sum_{m=1}^{M} \left\| \boldsymbol{y}^{(\boldsymbol{m})} - \hat{\boldsymbol{y}}^{(\boldsymbol{m})} \right\|_1 \quad . \tag{33}$$

Mean Square Error (MSE)

As in "Linear Regression" introduced MSE (16) is another metric for regression.

Generally speaking the loss functions MAE and MSE are widely used regression-losses for tasks using tabular data, such as predicting house prices or detecting cancerous blood samples. Because they compute distances between vectors MAE and MSE are not intuitively the best choice for BB regression. The distance between vectors can be the same for two different vectors due to the definition of l_1 and l_2 norm. To encode the location as well as the object within the predicted BB vector $\hat{\boldsymbol{c}}^{(\boldsymbol{m})}$ a different loss function is required.

Intersection over Union (IoU)

Intersection over Union, also called the Jaccard-Index, is one of the most popular evaluation metrics and losses applied in object detection. Other than previously introduced MAE, the IoU does not encode distances between vectors, but rather encodes the shape properties of the objects in comparison [26]. Here it will encode the location, height and width of Bounding Boxes. As its name suggests, IoU is computed from the area of intersection for each BBox prediction $\hat{\boldsymbol{c}}^{(\boldsymbol{m})}$ with respect to its true Bounding Box values $\boldsymbol{c}^{(\boldsymbol{m})}$ as well as the union of these. Let $\boldsymbol{c}^{(\boldsymbol{m})} = \left(x_{\min}^{(m)}, y_{\min}^{(m)}, x_{\max}^{(m)}, y_{\max}^{(m)}\right)$ be the ground truth coordinates of

a Bounding Box and $\hat{\boldsymbol{c}}^{(\boldsymbol{m})} = \left(\hat{x}_{\min}^{(m)}, \hat{y}_{\min}^{(m)}, \hat{x}_{\max}^{(m)}, \hat{y}_{\max}^{(m)}\right)$ predicted coordinates by the regressor. Then, the intersecting area, as seen in Figure 25, can be computed from $w_I^{(m)} = \min(x_{\max}^{(m)}, \hat{x}_{\max}^{(m)}) - \max(x_{\min}^{(m)}, \hat{x}_{\min}^{(m)})$, the width of the intersection area, and $h_I^{(m)} = \min(y_{\max}^{(m)}, \hat{y}_{\max}^{(m)}) - \max(y_{\min}^{(m)}, \hat{y}_{\min}^{(m)})$, the respective height.

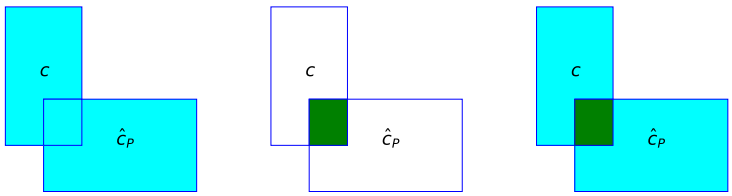

Figure 25: Left: $A_I^{(m)}$ (green) in combination with $A_U^{(m)}$ (cyan and green). Center: The intersection area $A_I^{(m)}$. Right: The union area $A_U^{(m)}$ for two BBs.

$$A_I^{(m)} = h_I^{(m)} \cdot w_I^{(m)} \tag{34}$$

Using $A_I^{(m)}$, the area of union, $A_U^{(m)}$ can be computed, visualized in Figure 25 (center). Let $h_c^{(m)} = x_{\max}^{(m)} - x_{\min}^{(m)}$ be the height of the ground truth, $w_c^{(m)} = y_{\max}^{(m)} - y_{\min}^{(m)}$ the corresponding width and $h_{\hat{c}}^{(m)} = \hat{x}_{\max}^{(m)} - \hat{x}_{\min}^{(m)}$ the height of the predicted Bounding Box with $w_{\hat{c}}^{(m)} = \hat{y}_{\max}^{(m)} - \hat{y}_{\min}^{(m)}$. Then, the union area can be computed by subtracting the intersection area $A_I^{(m)}$ from the sum of the areas within the BBs

$$A_U^{(m)} = (h_c^{(m)} \cdot w_c^{(m)}) + (h_{\hat{c}}^{(m)} \cdot w_{\hat{c}}^{(m)}) - A_I^{(m)} \quad . \tag{35}$$

To compute the IoU, $A_I^{(m)}$ is devided by $A_U^{(m)}$

$$\text{IoU}(\boldsymbol{c}^{(m)}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}) = \frac{A_I^{(m)}}{A_U^{(m)}} \quad . \tag{36}$$

This concept can be easier described using images. As visualized in Figure 25, the function computes the ratio of intersecting area over the union area.
The IoU can be used as a loss function by simply calculating $L_{\text{IoU}}^{(m)}\left(\boldsymbol{c}^{(\boldsymbol{m})}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}\right) = 1 - \text{IoU}\left(\boldsymbol{c}^{(\boldsymbol{m})}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}\right)$. This is also called the Jaccard distance, measures the dissimilarity in the objects and can be a target of minimization, whereas IoU is 1 when $\hat{\boldsymbol{c}}^{(\boldsymbol{m})} = \boldsymbol{c}^{(m)}$.
To calculate the IoU_B, the IoU over a batch of predictions, the average is computed

$$\text{IoU}_B\left(\boldsymbol{c}^{(\boldsymbol{m})}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}\right) = \frac{1}{M} \sum_{m=1}^{M} \text{IoU}\left(\boldsymbol{c}^{(\boldsymbol{m})}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}\right) \tag{37}$$

Generalized Intersection over Union (GIoU)

In the 2019 paper by Rezatofighi et al. [26] the researchers found out that using a modified version of the regular *IoU* for axis aligned 2D bounding boxes – so called *Generalized Intersection over Union (GIoU)* – improves predictions drastically.
The GIoU is calculated by subtracting the share of non overlapping area

$$\frac{A_C^{(m)} - A_U^{(m)}}{A_C^{(m)}} \tag{38}$$

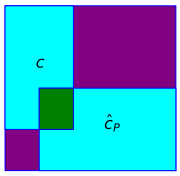

Figure 26: A_I (green), A_U (cyan and green) and A_C (purple + cyan + green)

with $A_C^{(m)}$ the area of the smallest convex hull or shape around the two Bounding Boxes. Let $\bar{\boldsymbol{c}}^{(\boldsymbol{m})} = \left(\bar{x}_{\min}^{(m)}, \bar{y}_{\min}^{(m)}, \bar{x}_{\max}^{(m)}, \bar{y}_{\max}^{(m)}\right)$ be to coordinates of the convex hull with

$$\begin{aligned}
\bar{x}_{\min}^{(m)} &= \min(x_{\min}^{(m)}, \hat{x}_{\min}^{(m)}) \\
\bar{y}_{\min}^{(m)} &= \min(y_{\min}^{(m)}, \hat{y}_{\min}^{(m)}) \\
\bar{x}_{\max}^{(m)} &= \max(x_{\max}^{(m)}, \hat{x}_{\max}^{(m)}) \\
\bar{y}_{\max}^{(m)} &= \max(y_{\max}^{(m)}, \hat{y}_{\max}^{(m)})
\end{aligned}$$

and $A_C^{(m)} = (\bar{x}_{\max}^{(m)} - \bar{x}_{\min}^{(m)}) \cdot (\bar{y}_{\max}^{(m)} - \bar{y}_{\min}^{(m)})$. In combination with the IoU

$$\text{GIoU}\left(\boldsymbol{c}^{(\boldsymbol{m})}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}\right) = \text{IoU}\left(\boldsymbol{c}^{(\boldsymbol{m})}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}\right) - \frac{A_C^{(m)} - A_U^{(m)}}{A_C^{(m)}} \quad . \tag{39}$$

A visualization of these areas can be seen in Figure 26. The corresponding loss function is

$$L_{\text{GIoU}}\left(\boldsymbol{c}^{(\boldsymbol{m})}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}\right) = 1 - \text{GIoU}\left(\boldsymbol{c}^{(\boldsymbol{m})}, \hat{\boldsymbol{c}}^{(\boldsymbol{m})}\right) \quad . \tag{40}$$

In the same way, like the IoU, is GIoU a similarity measure. Additionally, it is capable calculate this measure for objects without overlap.

4.10 Performance metrics

As in previous sections briefly described metrics are a group of functions, similar to Losses, that estimate the current predictive power of a ML Method. Like in the case of the loss, it is desired that the fitted function optimizes these metrics. Some metrics, like the *Accuracy* of right predictions or *GIoU* need to be maximized, some metrics, such as the *RMSE* get minimized, but all metrics measure or monitor the performance of the model during and after training. The following section will introduce metric functions implemented here, and will address some of their trade-offs.

4.10.1 Classification

Confusion Matrix
For the classification task a widely used metric is the Confusion Matrix. The

Confusion Matrix contains rows for actual classes and columns for predicted classes for each of the available output classes. The cells contain the number of occurrences of each pair of row and column value.
As an example, consider the results of binary classification for the classes "Coleoptera" and "not Coleoptera". If the model predicts "Coleoptera" for images containing "Coleoptera" the result is called a True Positive (TP) prediction, a predicted label of "not Coleoptera" for the same image is called a False Positive (FP). For images of class "not Coleoptera", on the other hand, predictions of "not Coleoptera" are called True Negatives (TN) and "Coleoptera" False Negatives (FN). Then, the resulting parameterised confusion matrix would look as following.

	Coleoptera	not Coleoptera
Coleoptera	TP	FN
not Coleoptera	FP	TN

Albeit this example is using binary classification, the confusion matrix is not limited to two classes and can be computed for multi-label classification, as well. The values in the resulting confusion matrix can be leveraged to calculate other metrics, such as the total number of correct predictions, or the total number of false predictions. These values, in turn, can be applied to calculate other metrics such as Precision, Recall and the Accuracy.

Precision and Recall

To measure the Precision of a classifier the formula is given by

$$precision = \frac{TP}{TP + FP} \quad . \tag{41}$$

According to this formula, the Precision computes how many predicted positive labels are indeed positive, but it ignores False Negative predictions. In some cases it is important to consider these False Positive predictions, to do so the Precision is used in combination with the Recall. The Recall is given by

$$recall = \frac{TP}{TP + FN} \tag{42}$$

Also called the sensitivity or *True Positive rate*, it is the ratio of positive instances that are correctly detected by the classifier. Recall is a measure of how many of the truly positive labels were labeled positive by the classifier.

F_1

Sometimes it is desired to combine Precision and Recall in one metric, this is called the F_1 *score*. The F_1 score is the harmonic mean of precision and recall

$$F_1 = \frac{2}{\frac{1}{precision} + \frac{1}{recall}} = 2 \cdot \frac{precision \cdot recall}{precision + recall} = \frac{TP}{TP + \frac{FN+FP}{2}} \tag{43}$$

Unfortunately, due to the fact that this metric is a combination of precision and recall it is impossible to optimize it to maximize both values, this is called the *precision-recall tradeoff*.

Accuracy

The Accuracy can be computed by dividing the number of correct predictions by the total number of predictions

$$ACC = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FP} + \mathrm{FN}} \tag{44}$$

4.10.2 Regression

The standard metric applied for regression tasks is the MSE (16) [13], or its easier to interpret the *R*oot Mean Squared Error (RMSE).
Root Mean Squared Error (RMSE)
RMSE has the form

$$\text{RMSE}\left(Y, \hat{Y}\right) = \sqrt{\frac{1}{M}\sum_{m=1}^{M}\left\|\boldsymbol{y^{(m)}} - \boldsymbol{\hat{y}^{(m)}}\right\|_2^2} \tag{45}$$

and is considered to be easier to interpret for humans than the MSE due to its scale in codomain.
(G)IoU
As in "Regression Losses" mentioned, IoU-Loss and $GIoU$-Loss originated from the metrics IoU and $GIoU$. They can be easily calculated back to the original form

$$\text{GIoU}(Y, \hat{Y}) = 1 - L_{\text{GIoU}}(Y, \hat{Y}) \tag{46}$$

$$\text{IoU}(Y, \hat{Y}) = 1 - L_{\text{IoU}}(Y, \hat{Y}) \tag{47}$$

For the regression tasks in this paper, the GIoU in combination with the RMSE is computed, to ensure own implementations of wrappers for the GIoU calculation are correct.

4.11 Training Strategies

Amongst the supervised training algorithms, we can differentiate the training strategies for the problem of localization and classification of insects into two groups. Whereas one group of strategies trains two separate models ("2-Stage-Model-Strategies") to solve the detection task, the other uses a 1-stage approach ("Single-Stage-Model-Strategy") where one model is used to solve the task. In supervised learning and for all of the following strategies the objective is equal: a loss function gets optimized until it reaches a satisfying threshold from which on the predictions are considered accurate enough or within the required boundaries, as described before in the previous section "Losses".

4.11.1 2-Stage-Model-Strategies

The former group, further called 2-Stage-Model-Strategies, includes independent training and sequential training training. All 2-Stage-Strategies search for solutions of two different optimization problems.
Let $g : \mathbb{R}^N \to \mathbb{R}^4$ be the bounding box predictor, $f : \mathbb{R}^N \to \mathbb{R}$ be the classifier and h the cropping and resizing function (4).
Therefore, we calculate at inference for the m-th input

$$\boldsymbol{\hat{y}^{(m)}} = \hat{f}\left(h\left(\boldsymbol{x^{(m)}}, \hat{g}\left(\boldsymbol{x^{(m)}}\right)\right)\right) \tag{48}$$

Independent Training: When training f and g independently, we solve the optimization problems for each model separately. The optimization task for the

BB regressor is

$$\hat{g} = \underset{g}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} L_{\text{bbox}} \left(g \left(\boldsymbol{x}^{(\boldsymbol{m})}, \boldsymbol{c}^{(\boldsymbol{m})} \right) \right) \tag{49}$$

with $L_{\text{bbox}} = 1 - \text{GIoU}$ the L_{GIoU}, as described in "Regression". Both models get trained on individual data sets, the Regressor on (3) and the Classifier on (6) respectively.

And the minimization of the classification-loss

$$\hat{f} = \underset{f}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} L_y \left(f \left(h \left(\boldsymbol{x}^{(\boldsymbol{m})}, \boldsymbol{c}^{(\boldsymbol{m})} \right) \right), \boldsymbol{y}^{(\boldsymbol{m})} \right) \quad . \tag{50}$$

The two models get trained independently, each on a data sets containing labels of the related shapes (6) and (3), to be later combined into a independent two-stage-model.

Sequential Training: For Sequential Model training, the same optimization for the regression-loss $\hat{g}$ is used as previously described in the independent training approach (49). For the classification task, the loss

$$\hat{f} = \underset{f}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} L_y \left(f \left(h \left(\boldsymbol{x}^{(\boldsymbol{m})}, \hat{g} \left(\boldsymbol{x}^{(\boldsymbol{m})} \right) \right), \boldsymbol{y}^{(\boldsymbol{m})} \right) \right) \tag{51}$$

gets optimized. Where $\hat{g}\left(\boldsymbol{x}^{(\boldsymbol{m})}\right)$ is the predicted BBox coordinates from the already fitted regressor. This strategy requires finding a fitting solver for the regression task first, then training the classification method upon a training set generated by the regressor.

4.11.2 Single-Stage-Model-Strategy

To solve the classification and localization task in a single pass through the network, a different approach is chosen. Let $f : \mathbb{R}^N \to \mathbb{R}^4 \times \mathbb{R}$ be the object detector (visualized in Figure 27). Then the loss calculated for the network is given by

$$\hat{f} = \underset{f}{\operatorname{argmin}} \frac{1}{M} \sum_{m=1}^{M} \left[L_y \left(f_2(\boldsymbol{x}^{(\boldsymbol{m})}), \boldsymbol{y}^{(\boldsymbol{m})} \right) + \lambda L_{\text{bbox}} \left(f_1 \left(\boldsymbol{x}^{(\boldsymbol{m})} \right), \boldsymbol{c}^{(\boldsymbol{m})} \right) \right] \tag{52}$$

Where f is again the classifier, g the bounding box regressor and $\boldsymbol{y}^{(\boldsymbol{m})}$ and $\boldsymbol{c}^{(\boldsymbol{m})}$ the ground truth labels yielded from a data set with samples in the shape (7). This architecture does not crop the image during inference time, which can be a potential speed improvement. Using the parameter λ the losses can be weighted differently. Sometimes the losses of the outputs differ in their codomain. This can be solved using the weight parameter λ to scale L_{bbox} accordingly. The here used configuration is based on equal weighted losses, GIoU and (24) share almost the same codomain, so no further weights were introduced and $\lambda = 1$ fixed set.

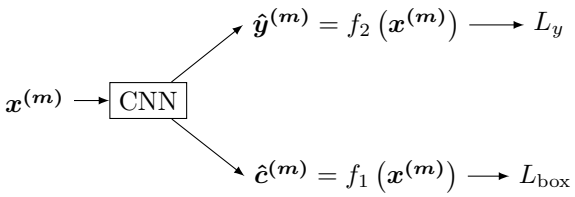

Figure 27: single-stage-model: The input gets processed by a single CNN, the resulting features are used by a Classification- and Regression-Head.

4.12 Model Optimization

4.12.1 Hyper Parameters

Hyper Parameters (HPs) are parameters used to configure the model prior to the training phase. Commonly used HPs are the learning rate of the optimization algorithm, regularization parameters for underlying layers of the model or the batch size used to feed the training data into the model.

4.12.2 Hyper Parameter Optimization

For the task of Hyper Parameter Optimization (HPO), several optimization techniques and algorithms are available. Amongst broad random parameter initialization, so called Random Search, more sophisticated algorithms were build to find the best parameter combination for the model to solve its task with the minimal loss. Let $L(y, \hat{y})$ be the loss function measuring the model's predictive power, then it is desired to find $\hat{\boldsymbol{\theta}}$ the vector of hyper parameters with minimal loss

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L_{\boldsymbol{\theta}}(Y, \hat{Y}) \tag{53}$$

with $\boldsymbol{\theta}$ the vector of hyper parameters.
Grid search is the most naive HPO method, in which a grid of HP combinations gets created and evaluated on each step. Unfortunately, this method naively evaluates all points in this equidistant grid, without any restrictions. In order to speed up the HPO process and use a more sophisticated method evaluating these HPs Bayesian Optimization [27] was used.
For this the implementation of `keras-tuner`'s [1] `BayesianOptimization` was integrated.
`BayesianOptimization` allows configuration of an objective `objective` that the optimizer optimizes, `num_initial_points` the number of starting points in the Bayes parameter space, `max_trials` the number of maximum total trials and `seed` a seed value to replicate the experiments. For this paper the validation accuracy was optimized for classifiers and the validation loss (L_{GIoU}) for regressors. Each of the HPO iterations used 12 initial points and performed a total of 36 trials. For all experiments the seed of 42 was applied.

The optimized HPs per task can be found in Figure 7 and Figure 8. Another method to optimize the HPs is cross-validation. This method was not applied in this paper, due to the fact that it was not possible to implement it for the case of 2-Stage-Model-Strategies, because the classifier receives the pretrained weights of the regressor.

5 Application

The following section will describe the implemented and applied techniques and model architectures, will display results of the experiments done here, describe the validation process of the final architectures and give a brief introduction about the roadblocks that were faced while implementing the experiments.

5.1 Chosen Model Architectures

To solve the tasks of classification and Bounding Box regression individually and combined several methods were chosen to train and compare in benchmark tests.

5.1.1 Conventional Methods

All experiments for conventional methods were performed on PCA-reduced data sets, as described in "Dimensionality Reduction", for this purpose the data set gets transformed onto the first 100 and 400 PCs of the training set.

Ridge Regression

The Ridge Regression model as introduced in "Linear Regression" is a single output predictor. To convert it to a multi-output Ridge Regression the implementation of `MultiOutputRegressor` [28] was used. `RidgeReg1` the first model that gets as input images transformed onto the 100 first PCs and `RidgeReg2` the second model using 400 first PCs of the regression data set that yields elements of shape (3) accordingly.

SVMs

For the classification task conventional linear SVMs were applied. Unlike the RidgeReg model, SVMs as implemented in footnote 28 can be applied for multi-class classification tasks directly without an additional wrapper method. Again, 100 (`SVM_1`) and 400 (`SVM_2`) extracted PCs of the cropped data set (6) have been used to solve the classification task.

5.1.2 CNN-Backbones

As stated in Convolutional Neural Networks (CNNs) the here implemented CNN-architectures are purely used for the task of extracting features from images, so called Backbones. Additional to these Backbones a dedicated "Task-solving-Head" was added according to the underlying training strategy. The architecture of the task solving heads will be described in the upcoming subsection.

INet

$INet$ is a custom CNN architecture implemented. Several iterations of experiments using different approaches led to the architecture used in this paper, a graph of the architecture can be found in Figure 28. The parameters used to create this architecture were selected according to best practices in the field, such as the pyramid structured conv. layers, but are also part of initial experiments while searching for an architecture.

State of the Art (SotA) Architectures

[28] scikit-learn overview of multiclass and multioutput algorithms: `https://scikit-learn.org/stable/modules/multiclass.html`

As discussed, this paper compares the results from previously mentioned methods, to state of the art architectures *VGG-16*, *MobileNet* and *YOLO*. Whereas VGG and MobileNet are *classical* CNN architecture, YOLO differs in many aspects. Explaining each of the architectures in detail will go beyond the scope of this paper. To get a full overview and read more about the details in the implementations of these architectures, t reader is encouraged to look into the papers [5,6,9], as well as looking into surveys [28] that have been published over the last years, or the Tensorflow "models" project. The implementations of VGG-16 and MobileNet applied here are part of the `keras`[29] library and can be imported easily. Both models were used with pretrained weights. Those weights have been trained on the `'imagenet'` data set. For the YOLO architecture the YOLOv5 [30] was applied.

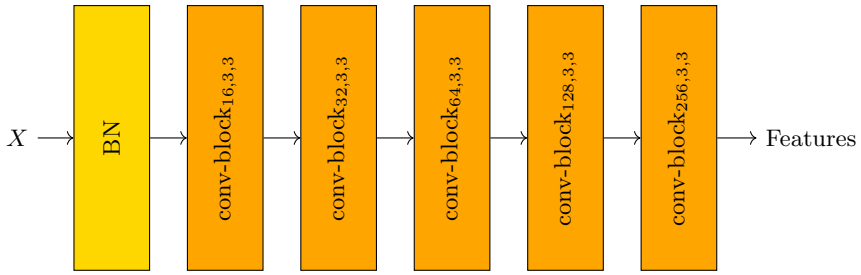

Figure 28: CNN Backbone feature extractor architecture. The input gets normalized using batch normalization, then gets processed by a series of convolutional blocks. Each of the blocks consists of a conv-layer[32], BN, ReLU and global-max-pooling layer and is using 3×3-filter masks. The i-th conv-block$_i$ has 2^{4+i} filter masks, for $i = 0, \ldots, 4$.

5.1.3 Task-solving-Head

To solve tasks based on extracted features the Backbone models from the previous section were extended by task solving heads. For the purpose of this paper a global-max-pooling layer was added right after the feature extractor, then a Dropout layer with factor 0.5, followed by a fully connected dense layer with 128 neurons and ending with a dense block with the number of output neurons that the task requires. The classification-Head consists of $K = 5$ output neurons and for regression $K = 4$, according to the descriptions in "Tasks". For the purpose of this paper a variety of different head architectures were explored until a satisfying combination of parameters was found. A visualization including the parameters can be seen in Figure 29. The regularization factor α of the output dense-block is part of the HPO.

[29] `https://keras.io/`

[30] YOLOv5 github source: `https://github.com/ultralytics/yolov5`

[32] conv-layer: Convolutional Layer with activation function ReLU and parameters n, h, w for number, height and width of the underlying filter masks.

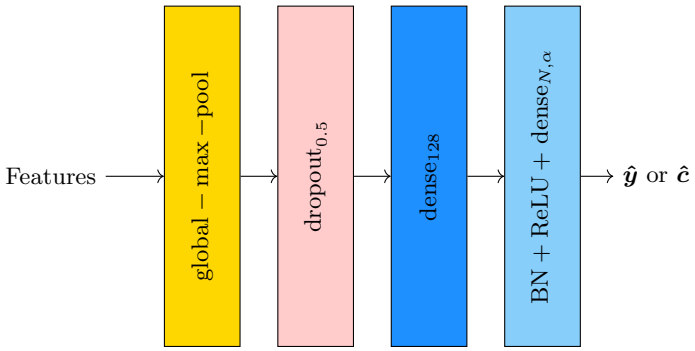

Figure 29: Visualization of task solving head. From left to right: Features get passed into a global max pooling layer, followed by a dropout layer with factor 0.5. Then the reduced feature set gets passed into a fully-conntected dense layer with 128 neurons. After the dense layer a final a dense block with N neurons and the regularization factor α calculates the final prediction.

5.2 Training results

For the experiments a small set of different feature extractors were trained and optimized using the Gaussian-Optimization as described in "Hyper Parameter Optimization". Each of the experiments was executed one the original data sets as described in "Data sets", and an augmented version of them. The HPs were selected according to Table 7 and Table 8. To compare the training results for the classification task the measures of Accuracy (44) and F1 (43) were selected. For the regression task GIoU and RMSE. This and the following sections describe the most promising results, a full overview of all training results, including plots of loss function progressions, or the used HPs and their final sets, can be found in the code repository wiki [33]. Here discussed results in combination with visualizations can be found in the "Appendix" section at the end of the document.
All architectures, except YOLOv5 [34] have been trained on a dedicated server, provided by BHT, using a "NVIDIA Quadro RTX 8000" GPU, if not explicitly marked differently.

5.2.1 Terminology

The following sections will display assembled methods, also called solvers, and the results of training and validation experiments of these methods.
While describing these results three terms are used that will be summarized here briefly.

- 2-stage-method: A solver, using two of the methods described in "Machine Learning Methods".

[33] Results listed at `https://gitlab.com/kinsecta/ml/thesisphilipp/-/wikis/home`, allows looking at results individually.
[34] YOLOv5 is implemented in pytorch and requires a TF version `>=2.4`. Therefore the YOLO model was trained on Google Colab using a GPU runtime environment (`https://colab.research.google.com/`).

  - Independent: This solver performs classification predictions on full images independently from the BB predictions.
  - Sequential: This solver performs classification predictions on cropped images based on BB predictions from the applied regressor.

- 1-stage-method: A solver, predicting both label types simultaneously based on full images.

5.2.2 Conventional Methods

Initially the conventional methods LinReg and SVM were fitted onto the individual training sets with samples in form (3) and (6) using a OvR method.

Method	**GIoU**	**RMSE**	**Accuracy**	**F1**
PCA_{100}				
Ind.	0.3481	21.2985	0.4806	0.4725
Seq.	0.3481	21.2985	0.284	0.2685
PCA_{400}				
Ind.	**0.3525**	**20.9326**	**0.5339**	**0.5200**
Seq.	0.3525	20.9326	0.3040	0.2704

Table 1: Training results on the raw validation set for conventional methods fitted independently (Ind.), using the training set with shapes (3) and (5), and sequentially (Seq.), using the training set with shapes (5) based on the predictions done with the LinReg model. One set of methods was fitted using the reduction method PCA_{100}, the other PCA_{400} as described in "Dimensionality Reduction"[36].

Looking at the results presented in Table 1 reveals that the conventional methods seem to generalize fine on the training data for the regression task. The results for classification are mediocre to bad. Once connected in the sequential (Seq.) approach the metrics for classification drop even more to a low of 30% accuracy. Another interesting learning is the comparison between the results obtained using PCA_{100} and those using PCA_{400}. Visualizations of predictions can be found in Figure 30, confusion matrices for visual validation are available in Figure 38. In these plots it becomes more obvious how bad the classification predictions are. As introduced in "Performance metrics" the confusion matrix is desired to contain 1.0 on its diagonal, here it is more like a random distribution. These visualisations match the results of the metrics as listed in Table 1.

These results proof the initial assumption, the conventional methods will not perform in a satisfying manner for the here given tasks. Therefore, no further experiments were performed using conventional methods.

5.2.3 Two-Stage-Methods

For the non conventional two-stage-methods each CNN-model was trained independently, on the original, raw, training data sets as well as on an augmented version of them. For the regression task the data set yields samples of form (3),

[36]Note that for the regression task the results are equal for the Independent and the Sequential Two-Stage-Strategies.

Figure 30: Sequential predictions for the validation set, performed with conventional methods. Left predictions from a model using PCA_{100}, right using PCA_{400}. For both cases some of the class predictions are wrong. Size and location of the BBs seem to converge to a mean value.

as described in "Data sets". These first HPO experiments with *raw* data sets delivered unsatisfying results, as Table 9 suggests. But also the metrics, e.g. MobileNet GIoU = 0.0525 and RMSE = 24.2184, are unsatisfying. Looking closer at the validation samples in Figure 39 hints that the model on the left side seems to be predicting almost equally located and shaped boxes.
To overcome this problem the augmentation techniques described in "Regression Augmentation" were applied. This leveraged the resulting augmented training data set to $2 \cdot M = 3600$ samples. Using this augmented training set, the results listed under "Augmented" (Table 9) were achieved. Unfortunately, the initial assumption was not right, that SotA architectures perform better on bigger data sets. E.g. VGG-16 had trouble converging to any solution for the augmented training set. This example can be seen in Figure 31.

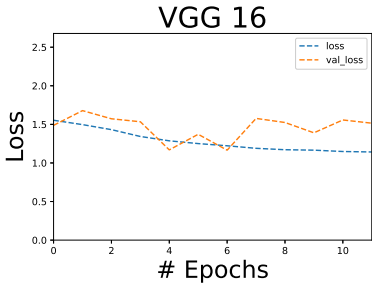

Figure 31: Regression loss function of VGG-16 trained on the raw training data set. Although the training loss `loss` decreases, the validation loss `val_loss` seems to bounce up and down. This model seems to overfit the training data.

Due to the fact that the results of MobileNet were somewhat satisfying for the task applied here, no further investigations using other backbones were done. Generally speaking INet and MobileNet seem to perform similar when trained

on the raw data set, while VGG-16 ends up in unsatisfying results for both training sets. The table (Table 9) and the visual components (Figure 40) reveal MobileNet as the best performing backbone in combination with an augmented training data set.
For the classification task the cropped data set (5) was selected first. Similar like for regression, this training set was augmented, as well. The final results can be found in Table 10, which lists the final metrics for the raw training set under "Raw", the augmented ones under "Augmented". For visual error analysis confusion matrices are used, these confusion matrix can be seen in Figure 41, Figure 42 and Figure 43. Already for the INet backbone trained on the raw training set it becomes clear that these methods seem to perform better than the previously discussed conventional methods. A clear diagonal can be seen. The final, and best, backbone for classification is MobileNet, as the table lists. The related confusion matrix (Figure 42, left) gives arguments for this. The diagonal contains only values ≥ 0.79 and contains less miss classifications, unlike e.g. INat trained on the augmented training set (Figure 41, right).
Hence, MobileNet was selected as a backbone for further experiments.

To combine two independent CNN architectures, to solve the object detection task, an additional HPO was performed on a training data set that yields samples with the original images (6). The results of this experiment can as well be found in Table 10, listed as MobileNet with "Uncropped, Raw" in the "Data set" column. The resulting confusion matrix can be seen in Figure 44 (left).
A last classifier was trained on a training set generated out of the predictions $\hat{\boldsymbol{c}}^{(\boldsymbol{m})}$ done by the best BB regressor, here MobileNet, trained on the raw train set. Finally, the 2-stage-models were assembled based on the best models resulting from previously described tests. Table 2 contains the results for these experiments. The final configuration for the independent approach, as well as for the sequential approach is MobileNet as backbone for both models, regression and classification.

Method	GIoU	RMSE	Accuracy	F1
Independent	**0.5602**	**17.0061**	**0.8511**	**0.8462**
Sequential	0.5602	17.0061	0.5666	0.5661

Table 2: Results of experiments with Two-Stage-Methods based on the training set with shapes (7). The Regression metrics GIoU and RMSE are equal in both cases, because both architectures use the same backbone for regression. Both methods are assembled using MobileNet as backbones for the BB regressor and classifier.

5.2.4 Single-Stage-Methods

For the approach of Single-Stage-Methods, again, three models were assembled, one based on INet, one on MobileNet and one with the VGG-16 as backbone. The results of the HPOs can be found in Table 11 (Figure 47). Additional to the HPs from regression and classification, as listed in Table 7 and Table 8 accordingly, the dropout was configured to be part of the HPO. For previous HPOs this was omitted because the models, that save tasks individually, seemed to not

Method	GIoU	RMSE	Accuracy	F1	Inf. time $[s]$
Independent	0.5638	17.3479	0.916	0.9168	0.6013
Sequential	0.5540	18.026241	0.5200	0.5285	0.7869
Single-Stage	0.3733	19.6725	0.92	0.9197	0.2195
YOLOv5	**0.694285**	**18.78450**	**0.848**	**0.8420**	**0.1951**

Table 3: Validation test results, based on the test set yielding samples in the shape (7), for assembled methods. The "Inf. time" value is the average inference time of 5 random samples[39].

be impacted by a change in dropout, this was the result of initial experiments, while developing the architectures. This was not clear from the beginning for the single-stage-methods, hence it was introduced as HP with $d \in [0, 1]$ the dropout factor. The loss weight was set to $\lambda = 1$, because it is desired to weight both tasks equally. Additional to the manually implemented architectures previously mentioned YOLOv5 was trained using the default configurations [37].

5.3 Validation of trained models on the test set

To validate the models the performance was measured using the, in "Notation" mentioned, test set. These experiments, or tests, are considered representational tests for the models, to see how well they perform on real-life data. For this purpose the best assembled models from the previous experiments were selected and tested. The results can be found in the table (3). For the independent method the combination MobileNet (regression) trained on the augmented data set and MobileNet (classification) trained on the full images was selected. The sequential solver contains the same regression model, but uses for classification the MobileNet backbone trained on a training set of predictions $\boldsymbol{c^{(m)}}$ as discussed earlier. As an additional validation value, the average time of inference in seconds, was calculated for each architecture based on five random selected samples. Predictions and confusion matrices can be found for two-stage-methods in Table 45 (independent), and in Table 46 (sequential). The related plots for the single-stage-methods can be found in Table 11. Previous training tests allowed the assumption that YOLOv5 outperforms all other methods. This becomes more visual by looking at sample predictions from YOLO (32). Not only is YOLO capable of performing better, according to metrics (3), but additionally YOLO is capable of predicting multiple BBs and class labels within the same image (Figure 32, right). Compare to all other methods, this is an outstanding improvement. Hence, it is inevitable to select YOLO for further investigations. Unfortunately, when comparing the results in Table 3 with Table 10 it becomes evident that something is wrong for the 2-stage independent model, mostly because the accuracy seems to differ a lot. This should be a target of further investigation, as well.

[37] The model was trained based on description from a blog post by Ayoosh Kathuria [29]. Further things, such as exporting to TFLite were executed based on descriptions in the code repository (`https://github.com/ultralytics/yolov5`)

[39] These tests have been performed in a private NVIDIA GeForce RTX 2060.

Figure 32: Predictions performed by YOLOv5. On the left side are single label predictions, here with the BB and label having the highest confidence. The right side contains predictions for one or multiple objects.

5.4 Hosting on a Raspberry Pi

This subsection will focus on implementation details which were required to perform to be able to host and use the final models on a RaspberryPi minicomputer. To reduce the size of a Tensorflow/Keras model several techniques can be applied [40].

5.4.1 Weight Pruning

Weight pruning gradually zeros out model weights during the training process to achieve model sparcity. Sparse models are easier to compress, and can skip the zeros during inference for latency improvements [41].

5.4.2 Quantization aware training

Quantization aware training emulates inference-time quantization, creating a model that downstream tools will use to produce actually quantized models. The quantized models use lower-precision (e.g. 8-bit instead of 32-bit float), leading to benefits during deployment [42].

5.4.3 Post training Quantization

Post training quantization is another conversion technique that can reduce model size while also improving CPU latency, with little degradation in accuracy. It allows to train models using floating point values, then convert those weights into TensorFlow Lite format. [43]

[40] A detailed guide can be found in the Tensorflow documentation.
[41] TF reference: `https://www.tensorflow.org/model_optimization/guide/pruning`
[42] TF reference: `https://www.tensorflow.org/model_optimization/guide/quantization/training`
[43] TF reference: `https://www.tensorflow.org/model_optimization/guide/quantization/post_training`

5.4.4 Weight Clustering

Weight clustering is a reduction technique that reduces the number of unique weight values in a model. It first groups the weights of each layer into N clusters, then shares the cluster's centroid value for all the weights belonging to the cluster [44].

Each of these methods can be applied separately or in combination. The models assembled for this paper were, as previously mentioned, trained on a server with a "NVIDIA Quadro RTX 8000" GPU and the target is to preserve the calculated weights as far as possible. Hence, quantization aware training (5.4.2) can not be applied. In the experiments it turned out that the performance of BB regression decreased drastically if a method other than weight clustering is applied. This is potentially caused by the fact that e.g. Quantization (5.4.3) that converts weights from `float32` into `int8` which trades off precision for size. The final configuration can be seen in Table 4.
Visual representations of predictions and confusion matrices can be seen in Figure 48 (two-stage, independent), Figure 49 (two-stage, sequential) and Figure 50 (single-stage).

Architecture	**Method**	**Initial file size**	**Final file size**
Two-Stage			
Independent			
Regressor	(5.4.4.)	40,6 MB	3,6 MB
Classifier	(5.4.3.)	40,6 MB	13,3 MB
Sequential			
Regressor	(5.4.4.)	40,6 MB	13,3 MB
Classifier	(5.4.3.)	14,8 MB	3,6 MB
Single-Stage			
Model	(5.4.4.)	42,2 MB	13.9 MB
YOLOv5	-	13.7 MB	14.2 MB

Table 4: Executed model optimization for architectures to allow faster inference on a Raspberry Pi. Whilst own implementations seem to decrease drastically in size, especially the classifiers, YOLOv5 reduces its size by only a bit more than 1 MB.

The optimization method for YOLOv5 is the interal configuration from the code repository that are available through one of the accommodating scripts, called `export.py`.

After converting the models into TF Lite format, the previously described validation tests were performed again on, this time on a Raspberry Pi using the reduced model weights. The results of these tests can be found in Table 5. Again, YOLOv5 outperforms the remaining methods. But it comes with a big trade-off. Whilst the predictive performance is great, the inference time increases by a factor if 10, compared to the custom single-stage method. This speed decrease is caused by the validation process of YOLO, that validates the set of many BB proposals extracted by the model, to then return the most

[44] TF reference: `https://www.tensorflow.org/model_optimization/guide/clustering`

Method	GIoU	RMSE	Accuracy	F1	Inf. time $[s]$
Independent	0.5540	18.02641	0.9200	0.9210	1.3140
Sequential	0.5540	18.0262	0.5200	0.5285	2.2799
Single-Stage	0.1833	21.7866	0.6080	0.5804	0.7272
YOLOv5	**0.6847**	**22.7424**	**0.82**	**0.8091**	**2.084**

Table 5: Test results for final TF Lite versions, based on the test set that yields objects in shape (7).

promising prediction.
Under the restriction to only use one of the models implemented and analyzed thoroughly, by this paper, the decision for the final model would be using the independent method. Not only the sequential method performs worse for the regression task, also the inference time increased is lower. This is most likely due to the fact that the independent method does not need to crop the image, prior to classification. The speed decrease, compared to the single-stage method, is drastic, by a factor of 3 and might be due to the fact that the independent method is processing the image twice, instead of once like the single-stage method. Over all, it seems that the classification task is easier to solve for the here analyzed CNN architectures.

5.5 Roadblocks

The following section will give a brief overview of main problems faced while working on this project to allow the reader to learn from mistakes done along the road.

5.5.1 Data sets and Augmentation

The here used data sets are highly influenced by the Dataset class provided in the Tensorflow API and are built using the generator approach. Data sets generated using generators are iterable, sequential data sets that allow lazy loading of the underlying samples. Instead of preallocating the full amount of memory required for the data set it loads elements on demand. This can either be single samples or batches of samples. For the purpose of **two-stage-models** and **single-stage-models** a total of three data set preperation helpers were created. One for the Regression task, one for the Classification task and one data set that contains combined outputs. To follow the generator idiom the here implemented data augmentation pipeline works in a similar way. Instead of storing all augmented samples in advance, to then load samples during training, the DataAugmentationHelper class wraps around the data set and creates a new generator data set that yields augmented samples. This allows simple replacement of augmentation methods in the helper class. And also reduces resource requirements in the training phase of the, mostly, magnitudes bigger augmented data set.

5.5.2 GIoU-Loss

One of the major obstacles to overcome was understanding Tensorflow-AddOn's implementation [45] of the GIoU-Loss. Initially a convention of

$$\boldsymbol{c^{(m)}} = \left(x_{\min}^{(m)}, y_{\min}^{(m)}, x_{\max}^{(m)}, y_{\max}^{(m)}\right)$$

for the BB coordinate labels was used to apply the supervised learning technique. This turned out to make the CNN architectures converge to solutions in which the model predicts BBs with either height or width component equal to zero. To solve this, the previously introduced format of $\boldsymbol{c^{(m)}} = \left(y_{\min}^{(m)}, x_{\min}^{(m)}, h^{(m)}, w^{(m)}\right)^T$ (2) was used to enforce learning of the aspect ratios of the BB rather than two independent coordinates. Unfortunately Tensorflows implementation of GIoU uses the format $\boldsymbol{c^{(m)}} = \left(y_{\min}^{(m)}, x_{\min}^{(m)}, y_{\max}^{(m)}, x_{\max}^{(m)}\right)^T$ hence a wrapper [46] was built for the conversion

$$\left(y_{\min}^{(m)}, x_{\min}^{(m)}, h^{(m)}, w^{(m)}\right)^T \rightarrow \left(y_{\min}^{(m)}, x_{\min}^{(m)}, y_{\max}^{(m)}, x_{\max}^{(m)}\right)^T \quad .$$

5.5.3 Tensorflow 2.4 vs. Tensorflow 2.8

The accommodating code repository was developed on a machine running Tensorflow v2.8, without the option of down-grading to a version prior to 2.6, but tests and experiments were executed on a machine running Tensorflow v2.4. This caused a big headache while implementing several building blocks for this paper, such as all of the dedicated Model classes in `src/models` and data sets in `src/data`. Thankfully the developers of Tensorflow maintain backward compatibility between package versions, which allows usage of code developed under 2.4 to run under version 2.6. Hence, after back-porting the code it was ensured to be executable under the newer version.

5.5.4 Dedicated Object Detection Architectures

Apart from CNNs, that are introduced in this paper, more complex architectures such as YOLO [9], or MobileNet-SSD [6] were presented over the last decades, these architectures seem to perform very well on object detection tasks. Due to their complexity in their architecture it was not possible considering the time constraints of this paper, to test fine-tuning or training these architectures. The likelihood is very high that these architectures solve the task better, and I would suggest the reader and my successor at *KInsecta* to look into them and compare results with the benchmarks presented here. Unfortunately, due to hardware and time constraints, it was not possible to display results of the originally trained model further. This is due to the fact that YOLOv5 is implemented in PyTorch rather than Tensorflow.

[45] Tensorflow-AddOn's documentation for GIoULoss: `https://www.tensorflow.org/addons/api_docs/python/tfa/losses/GIoULoss`

[46] This wrapper can be found in `src/losses/giou_loss.py`

6 Conclusion

To conclude, the here introduced CNN architecture has its limitations and existing state-of-the-art classifiers like MobileNet perform predictions with higher accuracy. But YOLOv5 seems to be a good candidate for further investigations, experiments and potentially to be the model that is later applied in *KInsecta*'s monitoring device.

First, it should be said that it was extremely fascinating and enjoyable to research this paper. To put it into the words of my generation: "I had the blast of my life!".
But the overall task given for this paper has been daunting for two reasons. Mainly, the three month time constraint limiting this thesis was a very tight schedule, considering that knowledge about most techniques used here was required to be acquired. Secondly, the object detection task itself was very challenging. On one hand, due to the complexity of CNNs and the abstraction through code, using Tensorflow. On the other, because it took a long time to achieve the here described results[47]. The results in Application are a great building block for my processor at *KInsecta*, and give me enough confidence to claim that the task will eventually be solved in a satisfying manner.

6.1 General ideas and prospects for similar problems

For ML projects in general, it is advised to get a good understanding of the problem and the available data for it. It seemed to be one of the most crucial parts of the ML project pipeline, introduced in the beginning of this paper (Figure 8). When enough knowledge about the raw input data is collected, augmentation techniques should be assembled to increase the amount of samples in the training set. After understanding and increasing the data, it is also recommend to use SotA (state-of-the-art) architectures first. These architectures exist for a reason, they generalize well on many different data sets. The chance of picking a problem that can not be solved in a satisfying way, using these models, seems from the here obtained results, unlikely. Additionally, great predictions with SotA architectures are a good motivation and can reduce the time spent exploring into a specific direction, rather than trying out a different ones.
If the objective is still to create a custom architecture from ground up, the reader is recommended to research different techniques and architectures thoroughly prior to performing first implementation attempts. This can also resolve frustration and might lead into finding comparable results from other researchers.
By now, the reader should be motivated to look deeper into YOLO, especially YOLOv5.

6.2 Outlook

As this paper demonstrated regular CNN architectures have their limitations. Compared to referenced papers and performance measures of SotA architectures give reason to believe that the resulting model is not yet the perfect candidate to solve the here described task. Due to time constraints of this thesis several approaches have not been tested or analyzed further. Albeit, the here assembled architectures do not reach good performances, the outlook that is given by the

[47] Some of the final results were computed in the last week prior to submission.

results of a naively trained YOLOv5 is outstanding. Compared to the other Backbone-Head architectures, the multi object detection by YOLO is a task none of the other ones can currently perform. Furthermore, it is impossible to ever achieve multi object detection, with the current architecture of these models, because each of them is predicting one and only one BB per image. Hence, the next steps should be to perform HPO for YOLOv5. Another open question that seeks an answer is the slow inference time of YOLO, this should be target of an investigation, as well.

7 Acknowledgement

This thesis was supported by my great colleagues at *KInsecta*, that welcomed me with open arms from the beginning and allowed me to take over the here described research. I would like to express my appreciation to Prof. Dr. Frank Haußer and M. Sc. Teodor Chiaburu for the helpful discussions and support during this research. Additionally, I would like to deeply thank my fiance Roberta, that has been under a lot of pressure and was forced to undergo my mood swings, over the last three months.

Abbreviation	Definition
ML	Machine Learning
LinReg	Linear Regression
RidgeReg	Ridge Regression
GD	Gradient Descent
OvR	One versus Rest
MSE	Mean Squared Error
DL	Deep Learning
NN	Neural Network
CNN	Convolutional Neural Network
conv. layer	Convolutional Layer
CV	Computer Vision
BB	Bounding Box
BBReg	Bounding Box Regression
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAE	Mean Average Error
HPO	Hyper Parameter Optimization

Table 6: Table of abbreviations with definitions

8 Appendix


```
{
    "id": 1585,
    "annotations": [
      {
        "id": 1520,
        "completed_by": 1,
        "result": [
          {
            "original_width": 500,
            "original_height": 384,
            "image_rotation": 0,
            "value": {
              "x": 1.1999999999999997,
              "y": 1.0416666666666665,
              "width": 97.59999999999995,
              "height": 97.13541666666664,
              "rotation": 0,
              "rectanglelabels": [
                "Lepidoptera"
              ]
            },
            "id": "G5K3NFjz3X",
            "from_name": "label",
            "to_name": "image",
            "type": "rectanglelabels",
            "origin": "manual"
          }
        ],
        "was_cancelled": false,
        "ground_truth": false,
        "created_at": "2022-02-12T11:27:48.299000Z",
        "updated_at": "2022-02-12T18:01:40.688292Z",
        "lead_time": 14.013,
        "prediction": {},
        "result_count": 0,
        "task": 1585,
        "parent_prediction": null,
        "parent_annotation": null
      }
    ],
    "drafts": [],
    "predictions": [],
    "data": {
      "image": "IMAGE_NAME"
    },
    "meta": {},
    "created_at": "2022-02-11T20:01:40.340238Z",
    "updated_at": "2022-02-12T18:01:40.721007Z",
    "project": 2
  }
```


Listing 1: LabelStudio JSON format.

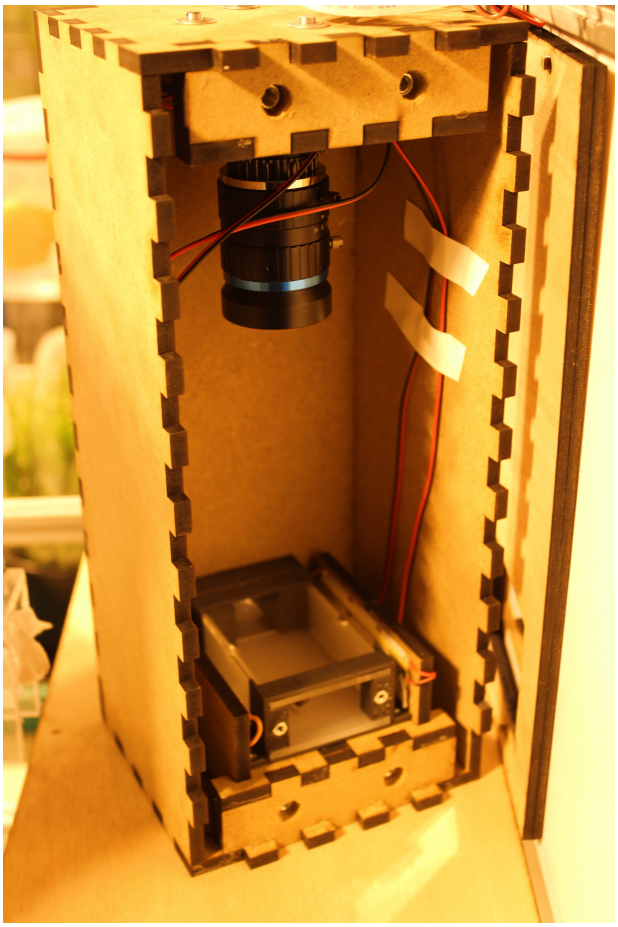

Figure 33: Camera module of the *KInsecta* device. High resolution camera lens on top and a light source triggered by a light barrier on bottom.

Parameter	**Options**
Learning Rate	$\{1^{-4}, 5^{-4}, 1^{-3}, 5^{-3}, 1^{-2}\}$
Regualrization Factor	$\{1^{-4}, 5^{-4}, 1^{-3}, 5^{-3}, 1^{-2}\}$

Table 7: Available parameters for regression HPO.

Parameter	**Options**
Learning Rate	$\{1^{-4}, 5^{-4}, 1^{-3}, 5^{-3}, 1^{-2}\}$
Regualrization Factor	$\{1^{-4}, 5^{-4}, 1^{-3}, 5^{-3}, 1^{-2}\}$
Frozen Layers	{0: All, 1: half, 2: None}

Table 8: Available parameters for HPO of classification models.

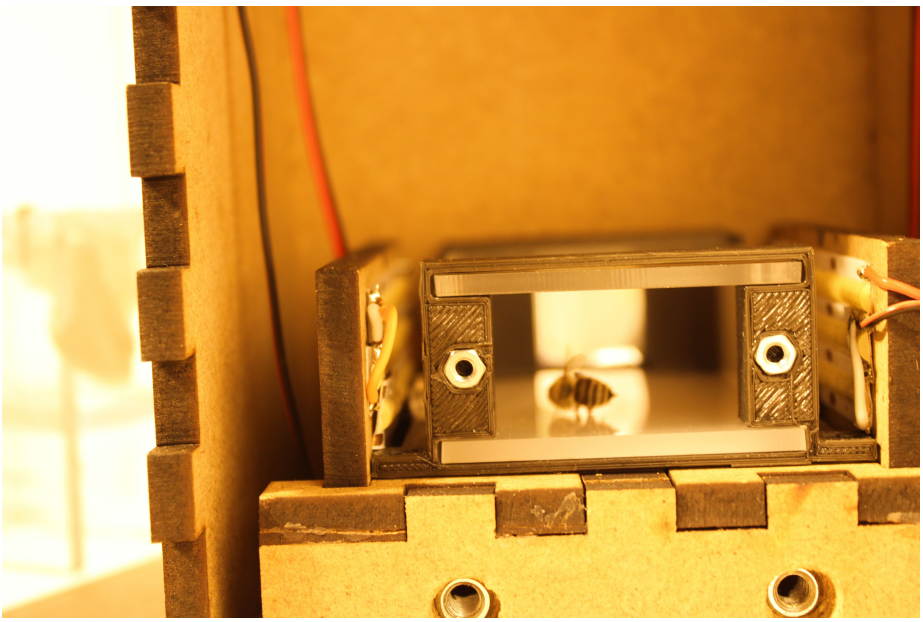

Figure 34: Look through the passage an insect needs to walk through to get recorded.

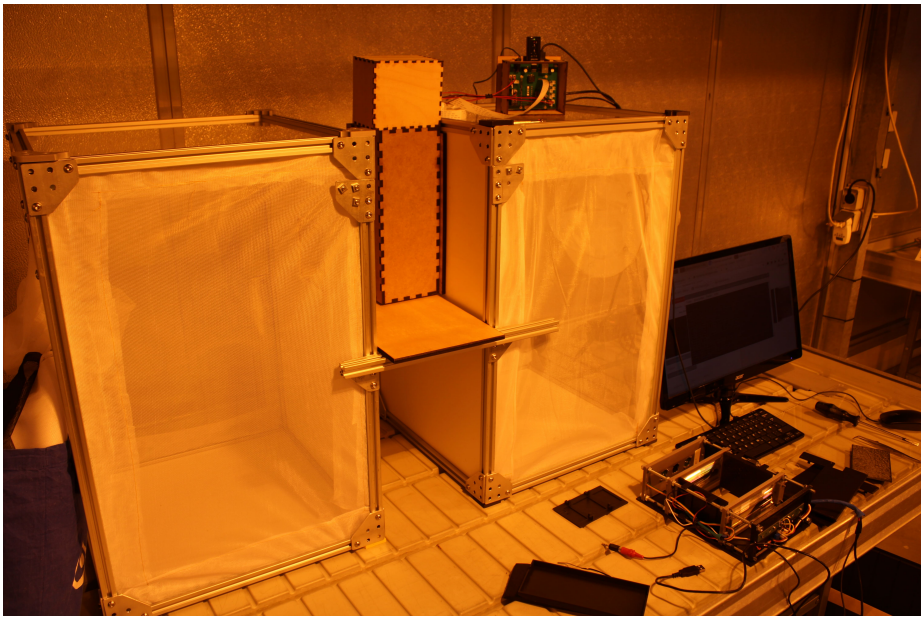

Figure 35: An experimental setup of the *KInsecta* device. Two containers, which will later contain specific insect species, connected with a passage leading through different setups for different sensors (wooden boxes). On the table lies the detached *WindBeat* sensor. Next to it a monitor and keyboard are connected to the *RaspberryPi* on top of the right box.

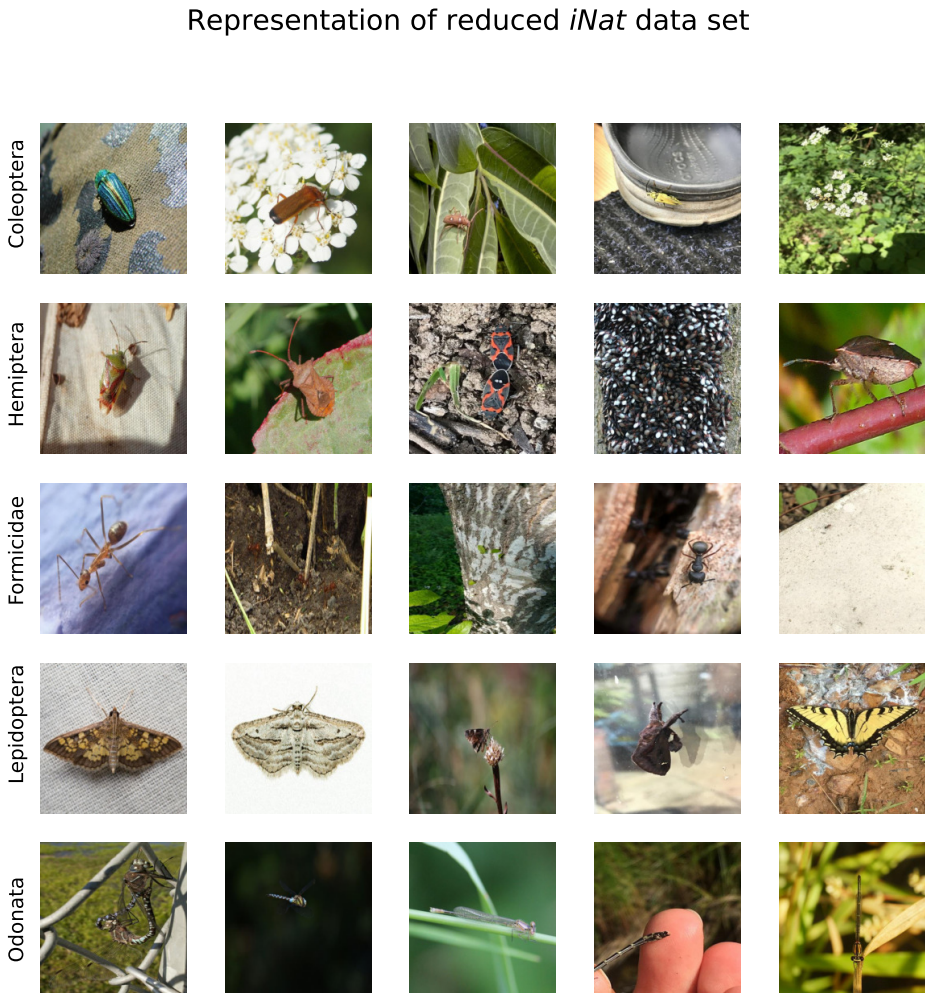

Figure 36: Representation of the here used data set displaying the five insect orders. It is easy to notice that all vary in quality as well as focus and coverage of important surface areas. Especially Hemiptera and Formicidae are difficult for Localization of individual insects due to the fact that they mostly appear in herds of multiple insects.

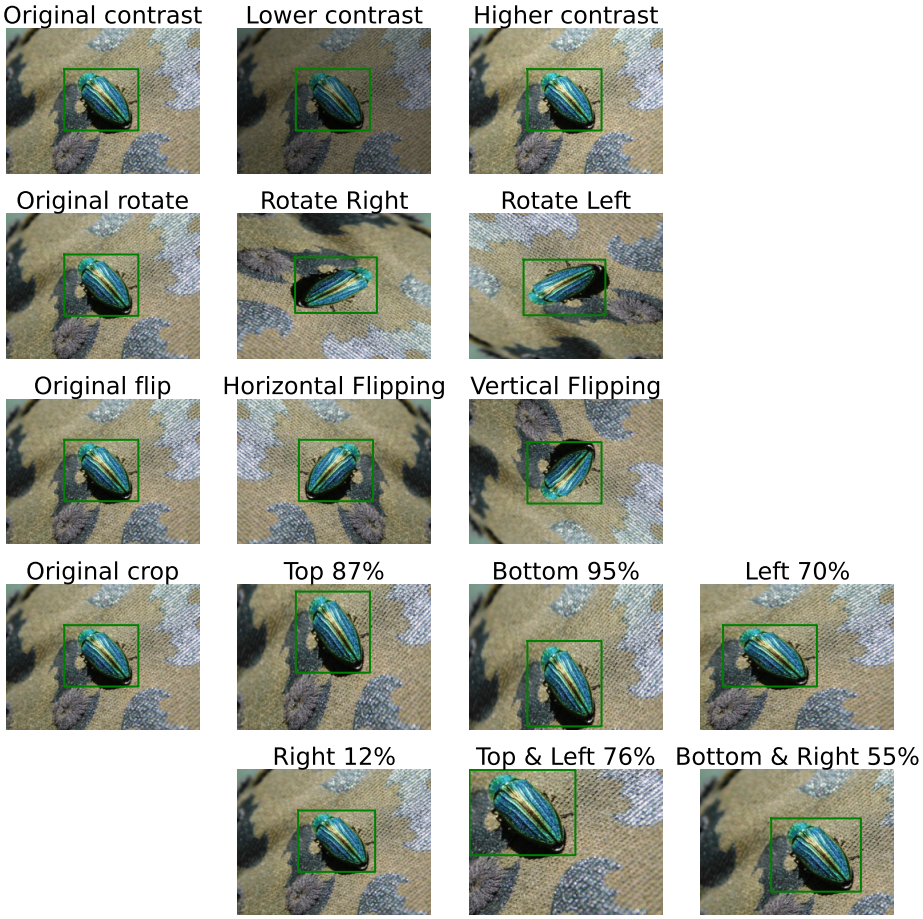

Figure 37: Visualization of different augmentation methods, from top down as listed in "Augmentation".

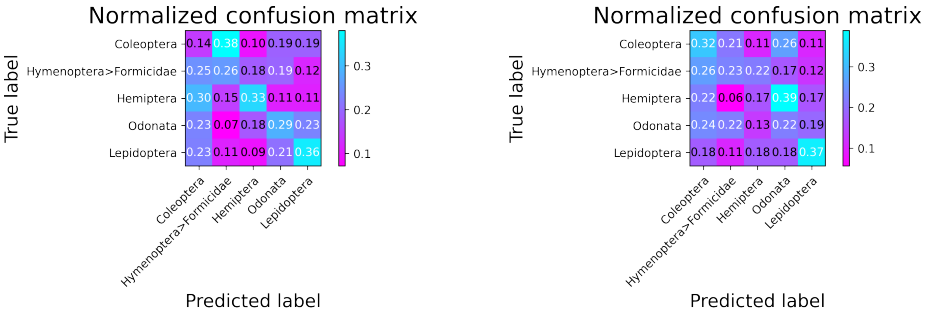

Figure 38: Confusion matrices for conventional methods based on the validation set. Left predictions performed with a model using PCA_{100}, right PCA_{400}. No distinctive classification for both cases. The predictions seem randomly distributed. Most cells contain values ≈ 0.2, which is the same value as a random guess.

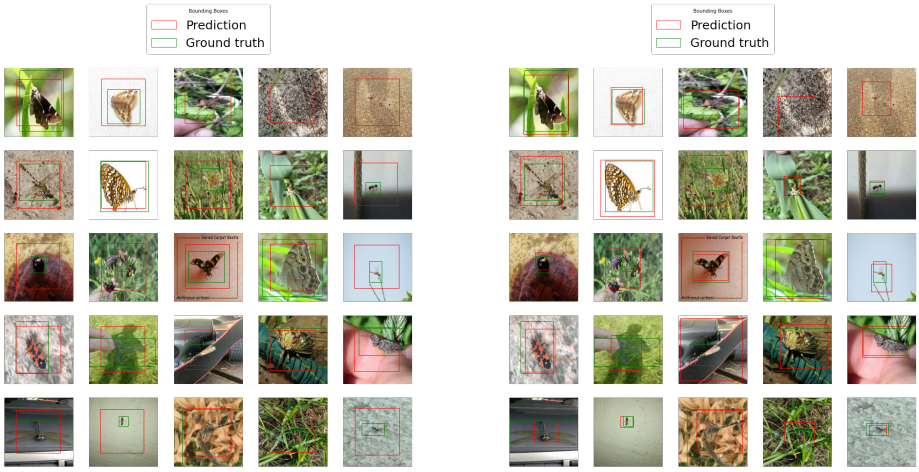

Figure 39: BB predictions on validation set of final models after HPO, using MobileNet as backbone. Left contains predictions from a model trained on the raw training data set. The predictions seem to be bound to a fixed location. Right, predictions performed with a MobileNet backbone optimized while training on the augmented training data set. Leading to a higher variance in BB locations and dimensions.

Backbone	**Data set**	**GIoU**	**RMSE**	**Learning Rate**	**α**	**HPO Time**
INet	Raw	0.3838	25.9176	0.005	0.0005	2h 42m
MobileNet	Raw	0.3877	25.3727	0.01	0.0001	7h 30m
VGG-16	Raw	0.0525	24.2184	0.001	0.01	11h 20m
INet	Augmented	0.3670	26.3442	0.01	0.01	11h 15m
MobileNet	**Augmented**	**0.5602**	**17.0061**	**0.0005**	**0.0001**	**19h 15m**
VGG-16	Augmented	−0.4000	40.7438	0.0001	0.001	22h

Table 9: HPO results for BBReg trained on a raw data set with sample of shape (3) ("Raw") and an augmented version of it ("Augmented").

Backbone	**Data set**	**Accuracy**	**F1**	**Learning Rate**	**α**	**Frozen Blocks**	**HPO Time**
INet	Raw	0.6422	0.7793	0.005	0.0001	0	2h 30
MobileNet	**Raw**	**0.8511**	**0.8462**	**0.01**	**0.001**	**1**	**6h 50m**
VGG-16	Raw	0.8467	0.8439	0.001	0.005	1	11h
INet	Augmented	0.6422	0.7793	0.005	0.001	2	15h 20m
MobileNet	Augmented	0.8333	0.826	0.005	0.0001	2	9h 40m
VGG-16	Augmented	0.8467	0.8439	0.005	0.0001	2	18h
MobileNet	Uncropped, Raw	0.7844	0.7793	0.0005	0.0001	0	2h 25m
MobileNet	Predicted, Raw	0.5289	0.5097	0.0005	0.0001	0	2h 30m

Table 10: HPO results for the classification task. The models were trained on training sets from the original image dataset ("Uncropped"), a cropped version of it ("Raw"), an augmented version of the cropped data set ("Augmented") and on a data set created using the BB predictions of a model. The values are based on performance in the validation part of the overall data sets.

Backbone	**Data set**	**GIoU**	**RMSE**	**Accuracy**	**F1**	**Learning rate**	α	**Dropout**	**HPO Time**
INet	Raw	0.3566	24.2111	0.4844	0.4625	0.003836	0.0001	0.617276	3h
MobileNet	Raw	−0.4751	39.1935	0.7311	0.7793	0.0001	0.0001	0.2352	4h 40m
VGG-16	Raw	−0.5731	39.1935	0.7311	0.7793	0.0045	0.0055	0.1000	6h 30m
INet	Augmented	0.3042	24.2111	0.4844	0.4625	0.0023	0.0022	0.4775	7h 10m
MobileNet	Augmented	0.3604	20.5186	0.7555	0.7496	0.0005	0.0001	0.6069	11h
VGG-16	Augmented	−0.4624	38.7296	0.78	0.7779	0.0001	0.0001	0.3221	21h

Backbone	**Data set**	**GIoU**	**RMSE**	**Accuracy**	**F1**	**Learning rate**	α	**Dropout**	**HPO Time**
YOLOv5	Augmented	0.6943	18.7845	0.848	0.8420	0.0100	–	–	–

Table 11: Single-Stage-Method HPO results. The upper table displays the best results for here implemented methods. All models have been trained using the original data set yielding samples in shapes (7) as well as an augmented version of it. Afterwards each model has been tested on the validation set. The lower table lists the performance of YOLOv5, here the default implementation was used that applies augmentation techniques automatically to the input training data set.

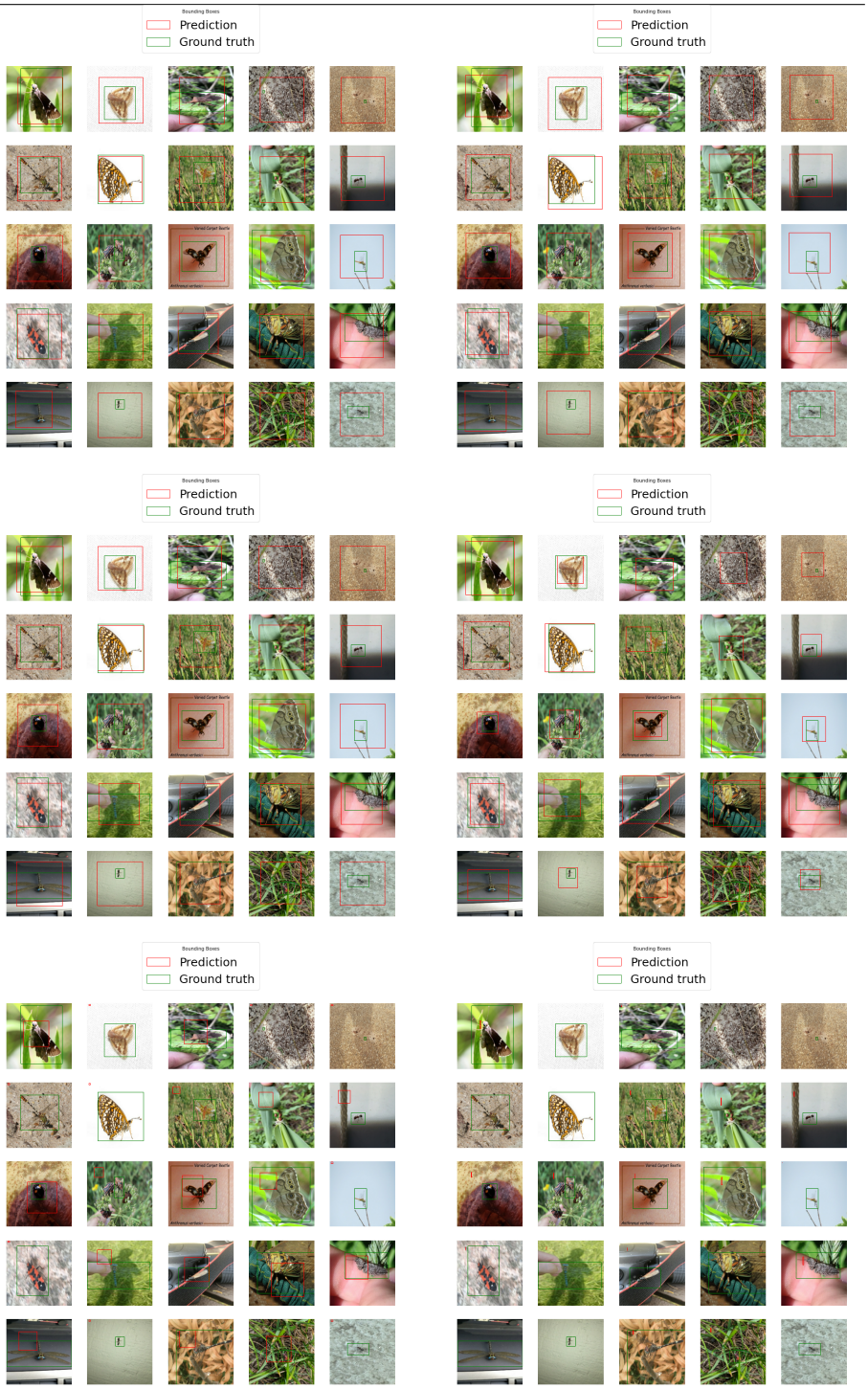

Figure 40: Predictions of models solving the regression task. All predictions from the left column are from models trained on the original data set. The predictions on the right side are done by models trained on an augmented version of it. Starting from top, the first row contains predictions of INet based models, the second MobileNet, and the third by models based on VGG-16 as backbone.

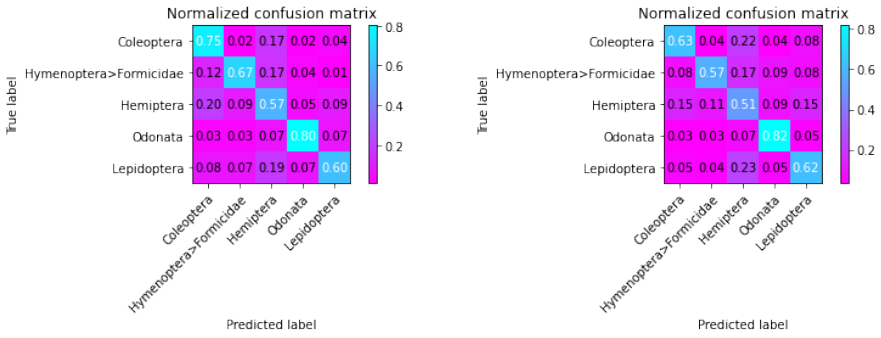

Figure 41: Confusion matrix of predictions on the validation set for the model based on the INet backbone. Left trained on the cropped training ((5)) on the right side trained on the augmented version. The model trained on the raw data set seem to generalize better, compared to its counterpart trained on the augmented set.

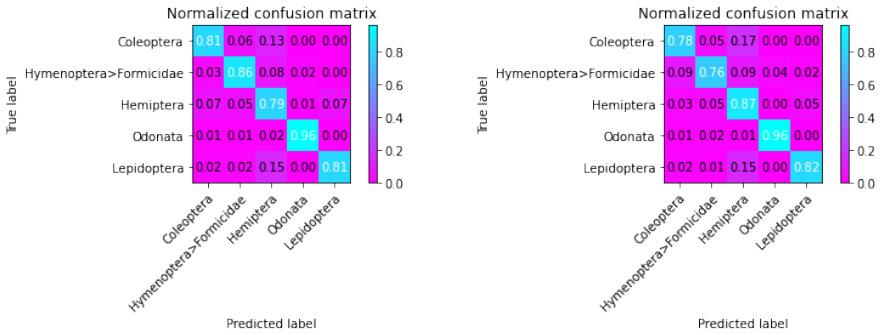

Figure 42: CMs for MobileNet classifier trained on the raw training set (left) and on the augmented version of it (right). Both models perform more or less equally on the validation set, but the overall predictive confidence of the left model seems to be higher than the right one.

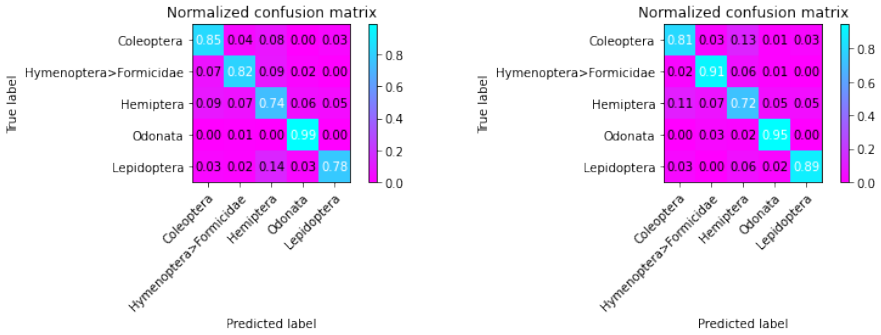

Figure 43: Confusion matrices for VGG-16 trained on a raw (left) and an augmented (right) training set. Other than previous models, e.g. INet, VGG-16 seems to generalize better on augmented data.

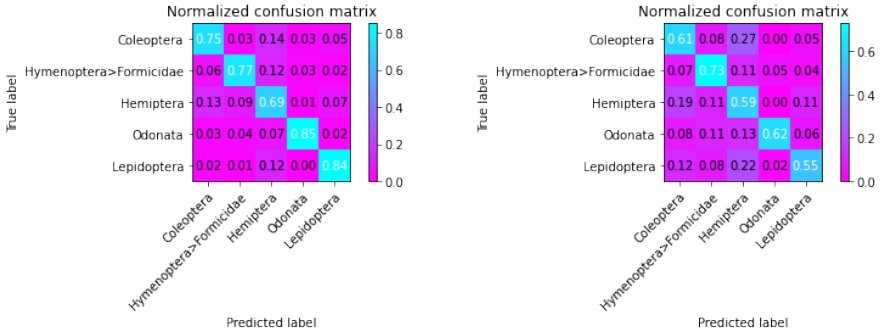

Figure 44: Confusion matrices for classification using a model having MobileNet as backbone and performing a specific task. Left the model was trained on uncropped images (6) and will be used for the independent method approach. On the right side the model later used for the sequential method, trained on a training set generated using predictions of the best BBReg model.

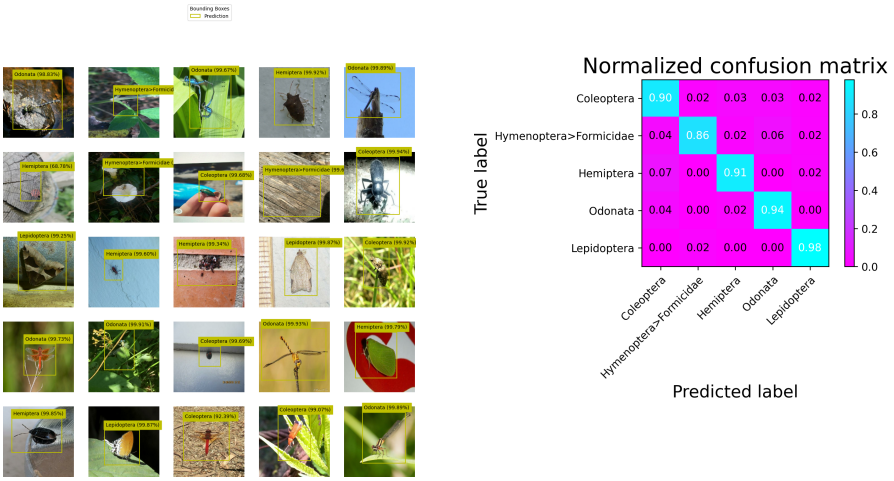

Figure 45: Validation for independent two-stage-method. Left, predictions done by the solver on the test set, on the right side the related confusion matrix. The method seems to generalize well for the classification task, as the CM hints. The BBs generated by the method are good as well, but for some cases (row: 2, column: 3) the predictions are not very accurate.

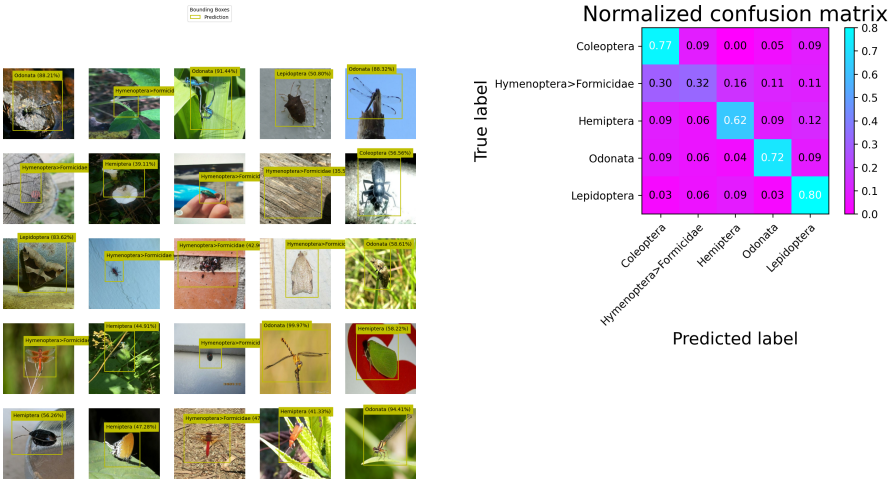

Figure 46: Sequential two-stage-method validation. The BBs are the same as in Figure 45. The confusion matrix differs. This is due to the fact that a new model was trained for the classification, but the regression task was solved by the same model. It seems that the sequential solver has trouble classifying Formicidae (ants).

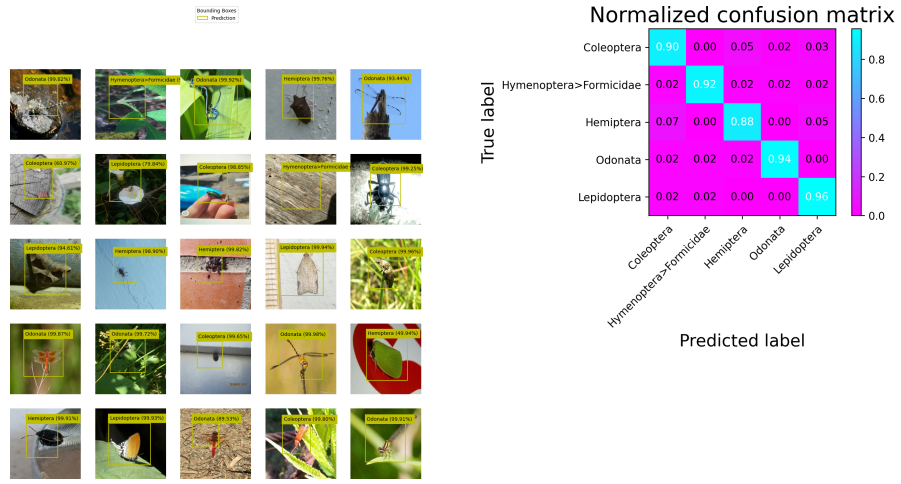

Figure 47: Evaluation of the single-stage-method, performed on the test set with samples of shape (7). The predictions on the left side show a problem, that was previously mentioned, the BBs seem to be fixed in one location. The results of the confusion matrix, compared to previous CMs (Figure 45 and Figure 46), more stable for all classes.

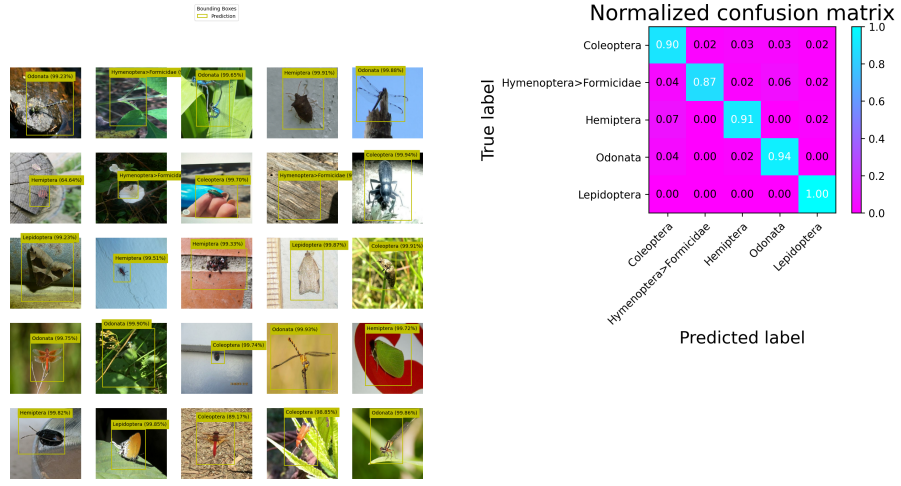

Figure 48: Validation results for TF Lite version of independent two-stage-method, using the test set. A decrease in quality for BB predictions, compared to its original model Figure 45, was predictable due to the conversion to TF Lite. But the classification task does visually seem not to be solved in a less accurate way, this assumption is strengthened by the results displayed in Figure 5.

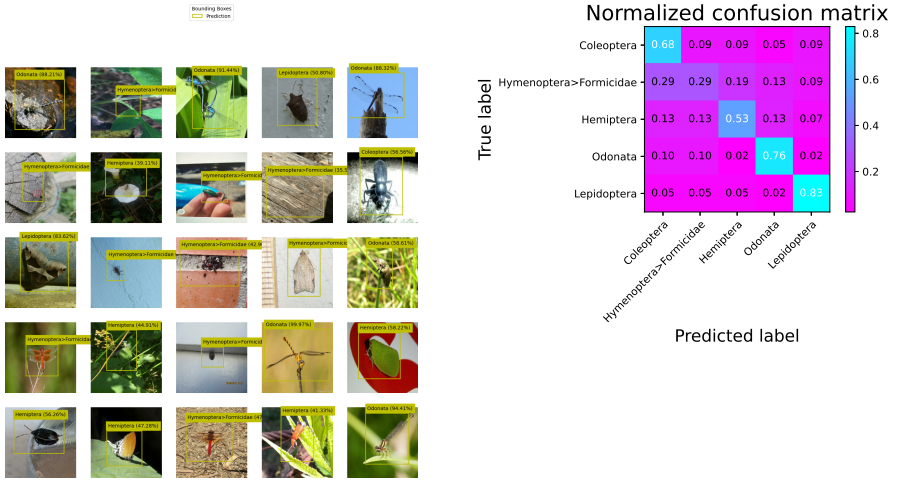

Figure 49: Results of validation for the sequential method converted to TF Lite applied to the test set. As in Figure 48 stated, the BB predictions decreased in quality. Unfortunately in this case the predictive decreased as well for the classification task, as the CM visualizes, e.g. the class Hemiptera lost $\approx 10\%$ in accuracy.

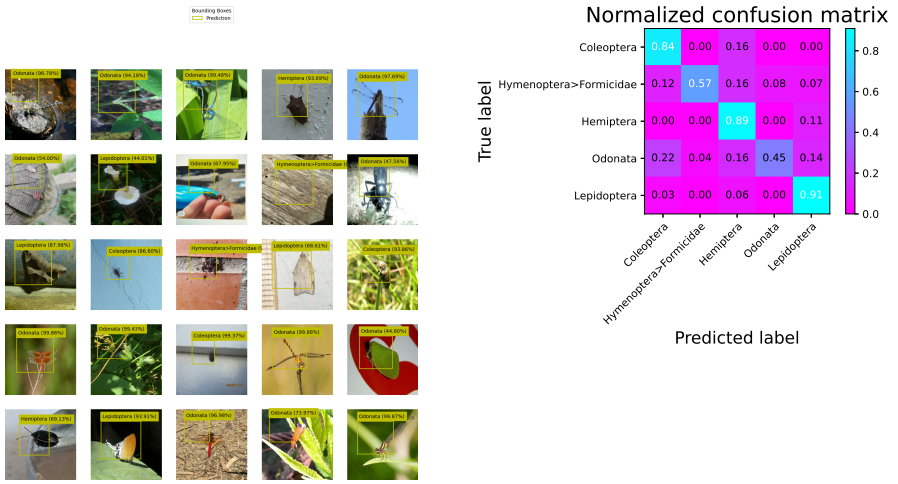

Figure 50: Validation of the TF Lite version of the single-stage-method, based on the test set. The previously badly located (Figure 11) decreased further in quality. Unfortunately, in this case also the classification quality decreases as listed in Figure 5. Whilst the original model performed well for the classification task, after converting to a TF Lite format the accuracy dropped drastically for all classes.

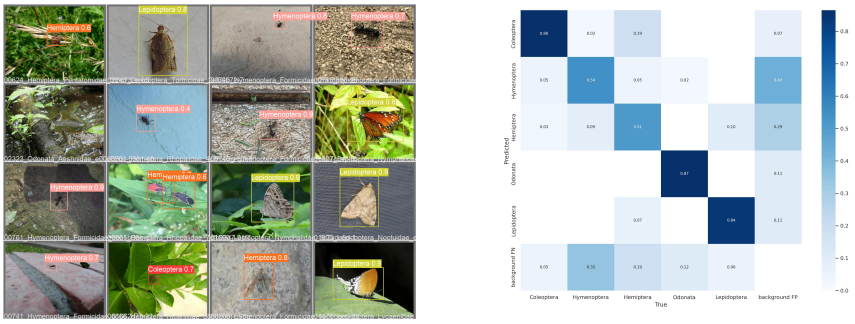

Figure 51: Validation results for YOLOv5 converted to TF Lite. Left the previously displayed (Figure 32) predictions on the right side the related CM.

References

[1] T. O'Malley, E. Bursztein, J. Long, F. Chollet, H. Jin, L. Invernizzi, *et al.*, "Kerastuner." `https://github.com/keras-team/keras-tuner`, 2019.

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] Y. Lecun, P. Haffner, L. Bottou, and Y. Bengio, "Object recognition with gradient-based learning," in *Contour and Grouping in Computer Vision*, Springer, 1999.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (Red Hook, NY, USA), p. 1097–1105, Curran Associates Inc., 2012.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015.

[6] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017.

[7] C. A. Hallmann, M. Sorg, E. Jongejans, H. Siepel, N. Hofland, H. Schwan, W. Stenmans, A. Müller, H. Sumser, T. Hörren, D. Goulson, and H. de Kroon, "More than 75 percent decline over 27 years in total flying insect biomass in protected areas," *PLOS ONE*, vol. 12, pp. 1–21, 10 2017.

[8] G. V. Horn, O. M. Aodha, Y. Song, A. Shepard, H. Adam, P. Perona, and S. J. Belongie, "The inaturalist challenge 2017 dataset," *CoRR*, vol. abs/1707.06642, 2017.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016.

[10] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C. Fu, and A. C. Berg, "SSD: single shot multibox detector," *CoRR*, vol. abs/1512.02325, 2015.

[11] M. J. K. Mr. Andrew Barron, "Yale university – fall 1997-98, statistics 101-103, introduction to statistics: Linear regression." `http://www.stat.yale.edu/Courses/1997-98/101/linreg.htm`, 1997.

[12] scikit learn, "scikit-learn examples: Linear regression example." `https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html`, 2022.

[13] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, Second Edition*. 1005 Gravenstein Highway North, Sebastopol, CA 95472: O'Reilly Media, Inc, 2019.

[14] H. Enghoff, "What is taxonomy? – an overview with myriapodological examples," *SOIL ORGANISMS*, vol. 81, pp. 441–451, 2009.

[15] `https://github.com/visipedia/`, "inaturalist 2021 competition description." `https://github.com/visipedia/inat_comp/blob/4ea5638e6f51c0a46c866305d6e0b640d40609d7/2021/README.md`, 2021.

[16] M. Tkachenko, M. Malyuk, N. Shevchenko, A. Holmanyuk, and N. Liubimov, "Label Studio: Data labeling software," 2020-2021. Open source software available from https://github.com/heartexlabs/label-studio.

[17] J. Shlens, "A tutorial on Principal Component Analysis," 2005.

[18] S. Raschka, *Machine Learning mit Python*. mitp Verlags GmbH, 2015.

[19] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 12, no. 1, pp. 55–67, 1970.

[20] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *arXiv:1412.6980 [cs]*, Jan. 2017. arXiv: 1412.6980.

[21] S. Ruder, "An overview of gradient descent optimization algorithms," 2016.

[22] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, Dec 1943.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[24] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015.

[25] Y. B. Yann LeCun, "Convolutional networks for image,s speech, and time-series." `http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf`, 1997.

[26] S. H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. D. Reid, and S. Savarese, "Generalized intersection over union: A metric and A loss for bounding box regression," *CoRR*, vol. abs/1902.09630, 2019.

[27] P. R. Garnett, "Cse 515t: Bayesian methods in machine learning – spring 2015, lecture 12: Bayesian optimization." `https://www.cse.wustl.edu/~garnett/cse515t/spring_2015/files/lecture_notes/12.pdf`, 2015.

[28] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019.

[29] A. Kathuria, "How to Train YOLO v5 on a Custom Dataset," 2021.