Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering

Bachelor Thesis

Birds on line data processing support

Author of thesis:

Mikheil Maisuradze

Declaration

I declare that I have worked on my bachelor thesis titled "Birds on line data processing support" by myself, and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break the copyrights of any person.

In Prague on 30.11.2020 ____________________________

Acknowledgment

First and foremost I wish to thank my thesis supervisor Ing. Josef Pavlíček, Ph.D., for the time and instructions he dedicated to me. He has been supportive since the days I began working on the bachelor thesis “Birds on line data processing support”. Last but not least I want to thank all who gave me the data and information needed for the thesis.

Birds on line data processing support

Abstract

This bachelor thesis deals with creating a custom dataset for object detection. In the theoretical part, it describes what object detection is. Object detection is explained in various ways: From the human point of view to its latest updates and all the main history leading to modern technology. The theoretical part of the thesis describes one of the leading and fast object detection models YOLO(which can recognize multiple classes of objects in an image). It includes YOLOs history, explanations of how it works, what is needed for YOLO to create a custom database, and the YOLO algorithm performance test. It follows with in-depth descriptions of the used programs and data for this thesis. The first program is LBLImg, which is an image annotation tool that was used for this thesis for labeling and converting given data to the format appropriate for the YOLO model. After the LBLImg thesis describes the programming language Python in which YOLO performs. Next comes information about the data provided by the thesis supervisor Ing. Josef Pavlíček, Ph.D. Data contains the photos from the birds (Tit) nest from the year 2017 and specifications of the motion detection camera used to shoot the images for this thesis.

The practical part ilustrates the process of development of ealier mentioned custom database for the YOLO model to detect birds open mouth and food portion on the image. It also shows step by step how I used earlier mentioned software and data to accomplish the project and test results.

Keywords: Data, Lable, LabelImage, Python, Object Detection, YOLO, Database, COCO database.

Podpora zpracování dat Birds On Line

Abstrakt

Tato bakalářská práce se zabývá s vytvořením vlastního datového souboru pro detekci objektů. V teoretické části popisuje, co je detekce objektů a je vysvětlena různými způsoby: Z pohledu člověka až po nejnovější technologii a historii vedoucí k moderní technologii. V teoretické části práce také popisuje jeden z předních a rychlých modelů detekce objektů YOLO, v souladu s jeho historií, vysvětlením toho, jak to funguje, a co je potřeba, aby YOLO vytvořil vlastní databázi, a test algoritmu YOLO. Následuje hluboký popis použitých programů a dat pro tuto práci. Prvním programem je LBLImg, což je nástroj pro anotaci obrázků, který byl použit pro tuto práci k označení a převodu daných dat do formátu vhodného pro model YOLO. Následuje popis programovací jazyka Python, ve kterem model YOLO je napsana. Dále přicházejí informace o údajích poskytnutých vedoucím práce Ing. Josef Pavlíček, Ph.D .. Data obsahují fotografie z ptačí hnízdo (sýkora) z roku 2017 a specifikace kamery pro detekci pohybu použité k fotografování snímků pro tuto práci.
Praktická část ilustruje postup, jak se zmínit o vlastní databázi pro model YOLO k detekci otevřených úst a potravy ptáků na obrázku. Postupně také krok za krokem ukazuje, jak jsem použil výše uvedený software a data k provedení projektu a výsledků testů.

Klíčová slova: Data, Lable, LabelImage, Python, Detekce objektů , YOLO, Database, COCO database.

Table of content

List of pictures

1 introduction

The topic of the bachelor thesis “Birds on line data processing support” was chosen deliberately. The main reason for choosing this particular work for the author is interest in the automatization of processes that cannot be done by a human, or it will take lots of effort and time to do. In the given topic I touched on the problem of how count bird feeding process. Whith data obtained from the work, we can generate statistics about the frequency of birds feeding the nestlings. This information will help farmers to make a nature-like environment for bird farming, Including when and how often nestlings are fed. We are gathering data about nature's behavior through long observation. With the help of coding in python and visual detectors, I managed to convert this data into useful information. This information will teach software on how to recognize when birds are feeding the nestlings, by comparing frames to the given information. The thesis works only with input photos, which are given to the detector by the user, but with few adjustments, it can also work on the live stream video, which will allow us to observe as many bird nests as cameras are given, gain not only data but already information converted into the numbers.

The work was done in the programming language Python. Python is a dynamic high-level programming language with a wide range of capabilities, the design of which focuses on easy-to-read code. The Python standard library is large and versatile. I chose python for its high speed, highly clear, understandable, and readable syntax. With help of its libraries, we can easily do huge tasks, with a couple of pre-made functions.

YOLO (You only look once, visual detector, which can recognize multiple classes of objects in an image) is highly engaging and useful for today's use. It’s fast with a comparison with other object detection networks.

This work aims to study the possibility of labeling food carried by a bird. The work will be used to teach the appropriate convolutional neural network. This network will then be used to verify the machine's ability to detect food accuracy.

In this work, there will be explained all steps to create a deep neural network that will recognize bird feeding accuracy. Starting from preparing a custom dataset for training YOLO Object Detector to testing the network.

2 Objectives and Methodology

2.1 Objectives

The bachelor thesis is focused on the development of Artificial intelligence with the ability of object recognition using the programming language Python.

Work consists of the following partial objectives – collecting all the data for the thesis photos from inside birds(tits) nest. choosing useful photos from a huge dataset of photos taken by the camera inside of the nest. Labeling photos, where birds are doing the motions we are interested in, when there is food inside the nest and when the nestlings have open mouths. Implementation of network training, for which was used Jupiter notebook an open-source web application, training will generate weight file for our custom dataset used for detecting the motions on photos. And the last but the most important part is the making of a custom dataset for training YOLO object detector and testing, for which we will need all the above preparation. After finishing the training user can give photos to the detector which will give the user the information if on the photo are the above mention motions and how accurate are those matchings.

2.2 Methodology

Solving the bachelor thesis problem will be based on online and book researches to understand how and in which environment the problem should be solved. The next step was obtaining data on which the thesis will be based on. These data are around photos from the bird's nest, which I had to convert into useful information by going through all of them one by one and choosing the right photo which will be useful to label them. After having the useful data, I had to make label them in the thesis adequate format (format: YOLO), labeling was done in the open-source program LabelImg, every label had to be done for every motion, some of the photos have more than one motion I needed to label, so I had to make labels for every motion. The next step was to train the network using the Jupyter notebook and generate a weight file for our custom dataset. After the YOLO network was trained by our custom dataset. After training network is able to recognize motion on photos the user gives to the network and give matching accuracy percentage back to the user.

The outcome of this work can be used to solve real-life problems. A particular YOLO algorithm with our dataset will be used to detect the feeding of nestlings on photos.

3 Theoretical Part

3.1 Object Detection

The human brain contains many mysteries. With the advent of modern electronics, attempts have begun to hardware-software playback of his work. The rapid development and application of the apparatus of artificial neural networks have come over the past half-century. Currently, the task of detecting and classifying objects in images is quite relevant. Software is being developed everywhere for automatic control of vehicles, validation of scanned documents, etc. In other words, algorithms and computer vision programs are being developed.

First of all, the algorithm for detecting objects in images requires high accuracy and speed of determining the location of an object and its classification. To date, algorithms have been proposed that achieve excellent performance according to these criteria. In this paper, we consider some of them.

In 2013, the first version of the R-CNN (Region-based Convolutional Neural Networks) algorithm was introduced based on image segmentation and convolutional neural network methods. Subsequently, these ideas evolved in Fast R-CNN and Faster R-CNN. The Faster R-CNN algorithm is one of the most accurate in its class. Still, in order to achieve high accuracy, it is necessary to sacrifice performance, which in turn leads to the inability to work in real-time

At the end of 2015, the YOLO (You Only Look Once) algorithm was introduced, which allows fast image processing (about 1000 times faster than R-CNN, 100 times faster than Fast R-CNN), but with lower accuracy.

In algorithms such as Faster R-CNN, the definition of objects in images occurs in two stages. The first stage is the results of the work of a deep, fully connected network (Region Proposal Network, hereinafter RPN), the purpose of which is to determine the regions in which the desired objects are supposedly located. The second stage is the use of the Fast R-CNN detector, which searches for objects in the proposed regions. Thus, the RPN selects the areas that Fast R-CNN further checks for objects. However, in this approach, these two artificial neural networks (ANNs) are trained independently of each other.

The YOLO system solves the detection problem as a regression problem. Due to the high speed of image processing, it is suitable for use in real-time systems.

One of the important areas of AI(Artificial Intaligence) is computers sight, known as Computer vision which is the science of computers and software systems that can recognize and understand images and scenes. Computer vision also consists of various aspects, such as image recognition, object detection, image generation, super-resolution images, and much more. Object detection is most likely the deepest and most complex aspect of computer vision because of the sheer number of practical cases.

Object detection refers to the ability of a computer and software systems to find objects in an image/scene and identify each object. Object detection is widely used for face detection, vehicle detection, pedestrian counting, web images, security systems, and driverless cars. There are many ways to use object detection technology, as well as many areas to explore. As with any other computer technology, a wide range of amazing and creative applications for object detection technology will certainly come from programmers and software developers.

Using modern methods for detecting objects in applications and systems, as well as creating new applications based on these methods, is not a direct task. Early implementations of object detection technology included the use of classic algorithms, for example, from theses supported by OpenCV, a popular computer vision library. However, these classic algorithms were not able to provide sufficient performance to work in different conditions.

The breakthrough and rapid implementation of an in-depth study in 2012 led to the emergence of modern and high-precision algorithms and methods for detecting objects, such as R-CNN, Fast-RCNN, Faster-RCNN, RetinaNet, and yet fast and high-precision SSD and YOLO. The use of these methods and algorithms based on deep learning, which is also based on machine learning, requires a good understanding of the mathematical and deep learning frameworks. There are millions of expert programmers and software developers who want to integrate and create new products that use object detection technology.

3.2 YOLO (You Only Look Once)

The most significant advantage of the YOLO model, in fact, is reflected in the name - You Only Look Once. This model superimposes a grid on the image, dividing it into cells. Each cell tries to predict the coordinates of the detection zone with confidence estimates for these fields and the probability of classes. Then the confidence score for each detection zone is multiplied by the probability of the class to get the final score.

It is an advanced real-time object detection system. On the official website, you can find SSD300, SSD500, YOLOv2, and Tiny YOLO, who have been trained with two different data sets: VOC 2007 + 2012 and COCO. You can find even more configuration options and datasets for machine learning on the Internet (for example, YOLO9k). Thanks to the inclusive range of options available, you can choose the version that is most suitable for your needs. For example, Tiny YOLO is the most "compact" option that can work quickly, even on smartphones or Raspberry Pi. We liked the last option, and we used it in our project.

YOLO has a strict input data array size of 608x608 pixels. We needed some kind of interface that can accept any image, normalize it and feed it into a neural network. And we have developed this interface. For normalization, it uses TensorFlow, which works much faster than other solutions we tested (native Python, NumPy, OpenCV).

An example of this can be seen in Fig. 2, gives the tensor size **S * S * (B * % + C)**, which is enough to build the boundaries of the detected objects. Here **S**- is the grid dimension, **C**- is the number of boundaries used in the estimation for each cell, is the number of classes that the neural network is able to recognize.

1. The image is superimposed with a grid dimension **S * S**. Each grid cell corresponds to a vector of dimension **5 * B + C**, where the number 5 determines the number of indicators of each border. Used indicators: **x, y** - coordinates of the center of the border inside the cell (take values from 0 to 1 relative to the size of the cell); **w, h** - width and height of the found border (have values from 0 to 1 with respect to the width and height of the original image, respectively); **c** - the probability that the boundary is correctly defined. The first **5 * B** values of the vector corresponding to the cell characterize precisely these parameters. The remaining **C** values in this vector show the probabilities that the center of the object is in this cell. At the moment, these probabilities are not tied to

boundaries; to find the class probabilities for each of the boundaries, it is necessary to multiply the boundary **c** characteristic by these vector values **C**, as shown in Figure 1.

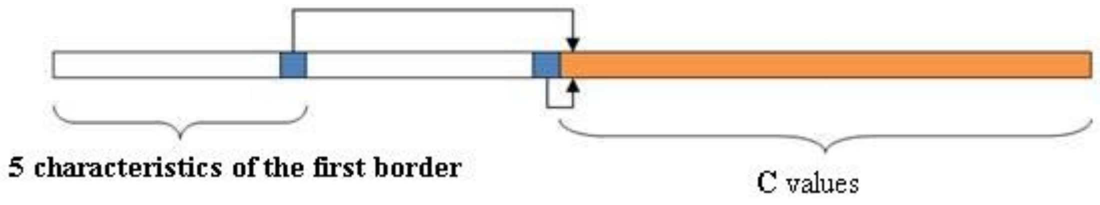


Figure 1Multiplication of boundary c by vector values C

2. As a result, we obtain boundaries **S * S * B** with class probabilities. In order to get the final recognition, you must do the following:

2.1. A class is fixed, for example, a “dog” for which there is a vector with a probability value of the class “dog” with each of the **S * S * B** boundaries.

2.2. We reset the values for those boundaries for which the value is less than the threshold (set in advance).

2.3. Sort the vector in descending order.

2.4. We apply the algorithm NMS (Non maximal suppression). It works according to the following principle: at the input, a vector of S * S * B probabilities of the “dog” class is supplied for all boundaries. The maximum value is selected since the vector is sorted, this element is in the first position, and then this boundary is compared with the boundaries to the right, which have a class probability of more than zero. The comparison takes place according to the intersection area: if the intersection area is greater than 0.5, then for the border, it is less likely that this probability is reset. By this principle, we compare the remaining boundaries. And then, we fix the next boundary with nonzero probability and carry out similar comparison operations. Thus, all boundaries for the dog class are considered,

2.5. Next, the next class is taken, and similar operations of comparison and zeroing are carried out. After a similar procedure, sparse vectors are obtained with probabilities for each class for each boundary found.

It remains only to decide what boundaries should be applied to the original image. The selection method is quite simple: each boundary is considered, the maximum probability value is taken by classes and, if it is greater than zero, then the boundary is applied to the original image. Otherwise, the next one is skipped and considered.

In the classic case of applying the YOLO algorithm, a grid is built in size, two boundaries are constructed for each cell, and the network is trained in 20 classes. A three-channel image of 448x448 size is fed to the input of a neural network. This tensor is passed through a modified GoogLeNet network (the first 20 layers), at the output of which we have a set of feature maps with a spatial dimension of 14x14x1024. Next, we use a set of convolutions with ReLU (rectified linear unit), which is an activation function of the form:

$$\varphi(x) = \begin{cases} x, & \text{For} \quad x \geq 0, \\ 0.1 * x, & \text{For} \quad x < 0. \end{cases}$$

Figure 2 rectified linear unit functuon

As a result, we get a tensor of size 7x7x1024. At the next stage, this set is passed through a fully-connected ReLU layer of dimension 4096x1 and then through a fully-connected layer, at the output of which we have a vector of 1470 elements, which is transformed into a 7x7x30 tensor. Further, for this tensor, we use the modifications described above.

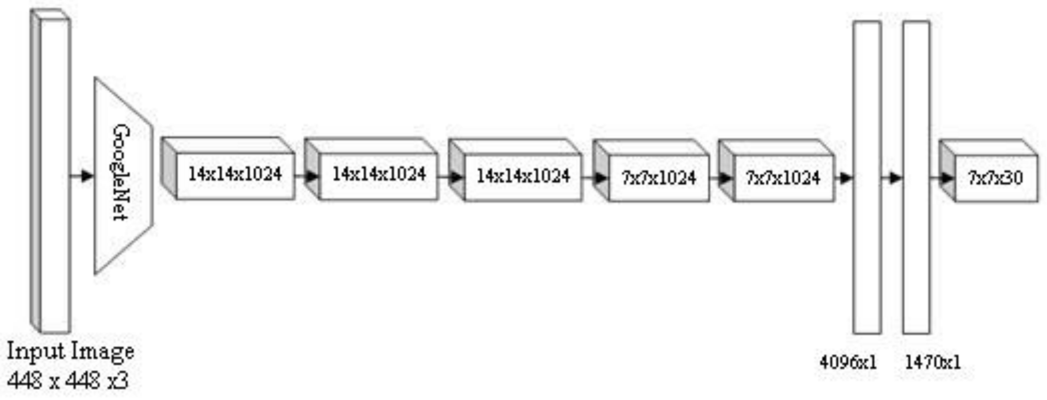


Figure 3 Input Image Transformation

3.2.1 Features of the YOLO v2 algorithm

Just six months after the publication of YOLO, an improved version was introduced, in which recognition of about 9000 categories was implemented (while the first version in real-time could recognize about 200 classes). Significant improvements include

Batch normalization. It leads to a significant improvement in convergence, eliminates the need for other forms of regularization. The increase in average recognition accuracy was 2%.
High Resolution Classifier. Replacing the GoogLeNet framework with ImageNet made it possible to fine-tune the resulting network for discovery. The increase in average recognition accuracy was 4%.
Dimensional Clusters and Direct Layout Prediction. In the first version of the neural network, the estimate of the intersection area, which works well only with boundaries of comparable sizes, was used as a metric. To solve this problem, as a metric that would work equally well with borders of different sizes, it was proposed:

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

$$\text{IOU}(\text{box}, \text{centroid}) = \frac{S_{\text{box}} \cap S_{\text{centroid}}}{S_{\text{box}} \cup S_{\text{centroid}}}$$

Figure 4 Dimensional Clusters and Direct Layout Prediction

That allowed us to increase the accuracy by about 5%.
In total, all these innovations made it possible to increase the accuracy of the neural network from 63.4% to 76.9%.

3.2.2 YOLO algorithm performance test

The YOLO v2 algorithm was tested on the same sample of images using CPU (processors) and GPU (graphics cards). GPU calculations were performed using CUDA technology [9]. On average, it took 8.97 s to recognize one image using the CPU, and only 0.92 s when using the GPU.

A significant increase in performance when using the GPU is associated with a high degree of parallelism of the source code of the YOLO v2 algorithm. Graphic cards allow you to quickly process the given blocks of images and carry out their classification.

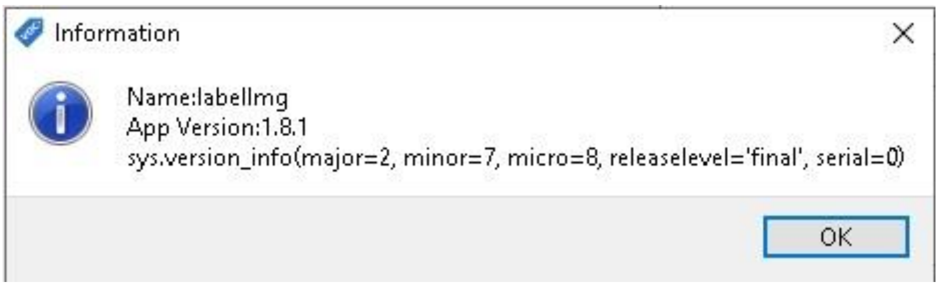
3.3 Used Programs and Data

3.3.1 LBLImg

LabelImg is a graphical image annotation tool. It uses Qt for its graphical interface, LBLImg is written in Python and. LabelImg can generate labeling in YOLO format.

The main reason I used LBLImg is that it supports the YOLO format. For this project, I needed to label more than 800 Photos, and LBLImg was the bast choice to choose because of its easiness. In the practical part of the thesis, it will be furtherly explained how LBLImg was used, and we can see why it was the best choice.

For this thesis I used LBLImg 1.8.1 version



```
Name:labelImg
App Version:1.8.1
sys.version_info(major=2, minor=7, micro=8, releaselevel='final', serial=0)
```

Figure 5 labelImg Information

LBLImg has some predefined classes as well. It was not used for this project, but those classes are extremely helpful for other projects.

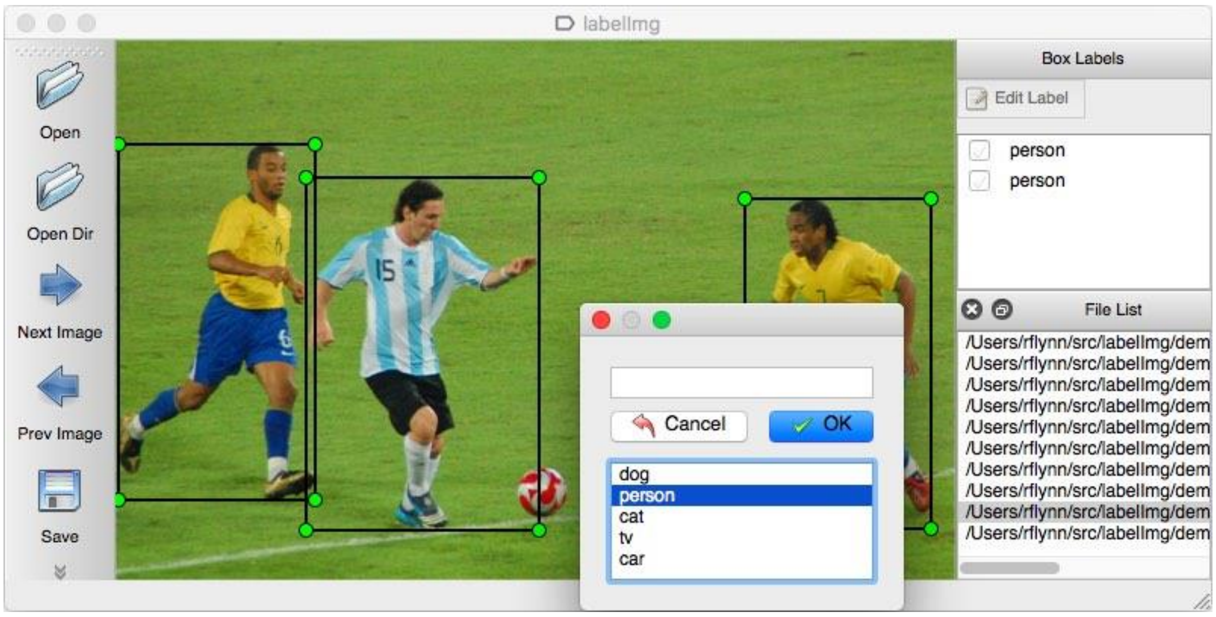
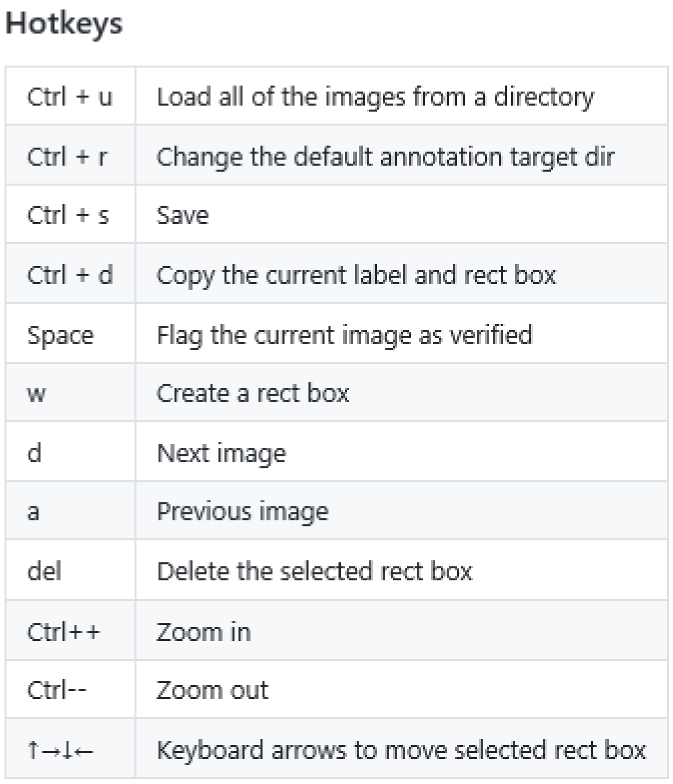


Figure 6 labelImage Visualization (github, n.d.)



Hotkeys

Ctrl + u	Load all of the images from a directory
Ctrl + r	Change the default annotation target dir
Ctrl + s	Save
Ctrl + d	Copy the current label and rect box
Space	Flag the current image as verified
w	Create a rect box
d	Next image
a	Previous image
del	Delete the selected rect box
Ctrl++	Zoom in
Ctrl--	Zoom out
↑→↓←	Keyboard arrows to move selected rect box

Figure 7 Hotkeys for lablImg (github, n.d.)

Verify Image:

Pressing space can flag the image as verified. The green background will appear after verification. This verification is used when the user wants to create a dataset automatically. Then the user can go through all the photos and flag them instead of annotating pictures.

Difficult:

When an object is clearly visible but difficult to recognize without substantial use of context, the difficult field is set to 1. It means that the object has been explained as „Difficult“.

3.3.2 Python

Figure 8 Python Logo (www.python.org, 1990)

Python is an interpreted, object-oriented, high-level programming language with dynamic typing, automatic memory management, and convenient high-level data structures such as dictionaries (hash tables), lists, tuples.

Supports classes, modules, exception handling, and multi-threaded computing. Python has an expressive and straightforward syntax. The language supports several programming paradigms: structural, object-oriented, functional, and aspect-oriented.

Python was developed at the end of 1989. Guido van Rossum (Guido van Rossum) during the Christmas holidays, when his research laboratory was closed, and he simply had nowhere to go. He borrowed many of the programming tools inherent in other languages.

The name of the language did not come from the name of the reptile family. The author named the language after the famous British comedy television show of the 1970s, Monty Python's Flying Circus.

Unlike other programming languages, Python is not only distributed entirely free of charge, but it also has absolutely no restrictions in terms of use.

No one restricts the commercial use of software products written in this language without any royalties. Programmers are also free to upgrade the language without notifying the author.

Version 1.0

Python 1.0 appeared in January 1994. The main new features included in this release were functional programming tools: lambda calculus, map, filter, and list folding.

Van Rossum claimed that "Python acquired lambda, reduce (), filter (), and map () thanks to a Lisp lover who lacked them, and he provided patches that implement these functions."

The latest version released by Van Rossum while working at the Center for Mathematics and Computer Science was Python 1.2. Since 1995, Van Rossum has continued to work on Python at the National Research Initiatives Corporation in Reston, Virginia, where several versions of the language have been released.

By version 1.4, Python included many new features, among which the most notable were the named parameters borrowed from Modula-3 and the built-in support for complex numbers. Also, in 1.4, a simple form of data hiding using name mangling appeared.

BeOpen Version

In 2000, the core of the Python development team moved to BeOpen.com, forming the BeOpen PythonLab team. Python 2.0 was the only release of BeOpen.com. After him, Van Rossum and the rest of the PythonLab developers joined Digital Creations.

Version 2.0

Python 2.0 introduces list inclusion — a function borrowed from the functional programming languages SETL and Haskell.

The syntax in Python for this construct is very similar to Haskell, except that Haskell preferred to use punctuation characters, and in Python, keywords. Also, in Python 2.0, a garbage collection system with support for circular references has been added.

Starting with the alpha release of Python 2.1, all code, technical documentation, and specifications belong to the non-profit organization Python Software Foundation (PSF), created in 2001 based on the Apache Software Foundation.

The release included a change in the language specification that supports nested scope, as in languages with a static (lexical) scope.

In Python 2.2, there was a combination of Python base types and user-created classes in one hierarchy. This has made Python a fully object-oriented language.

Version 3.0

Python 3.0 (called "Python 3000" or "Py3K") was designed to address fundamental flaws in the language. These changes could not be made, provided that full backward compatibility with the 2.x version was maintained, so a significant version number change was required.

The guiding principle behind the development of Python 3 was: "reducing duplicate functionality by eliminating obsolete ways to do this." Python 3.0 was released on 03.12. 2008.

Python 3.8.2 and 3.9.0a4 are still in development. Python 3.9.0a4 beta version will be available in 18.05.2020.

What do they write in Python?

- System utilities.
- Web sites (Django, Flask, Pyramid, Tornado, TurboGears).
- Applications for scientific calculations (NumPy, SciPy).
- Prototypes.
- Desktop applications (Tkinter, PyQt, wxPython).
- Games (Pygame).
- Mobile applications (Kivy).

Where is Python used?

- Google uses Python in its search engine.
- Companies like Intel, Cisco, Hewlett-Packard, Seagate, Qualcomm, and IBM use Python to test hardware.
- The YouTube Video Sharing Service is slowly implemented in Python.

- NSA uses Python to encrypt and analyze intelligence.
- JPMorgan Chase, UBS, Getco, and Citadel are using Python to predict the financial market.
- The popular program BitTorrent for sharing files in peer-to-peer networks is written in Python.
- Google's popular App Engine web framework uses Python as an application language

Python syntax

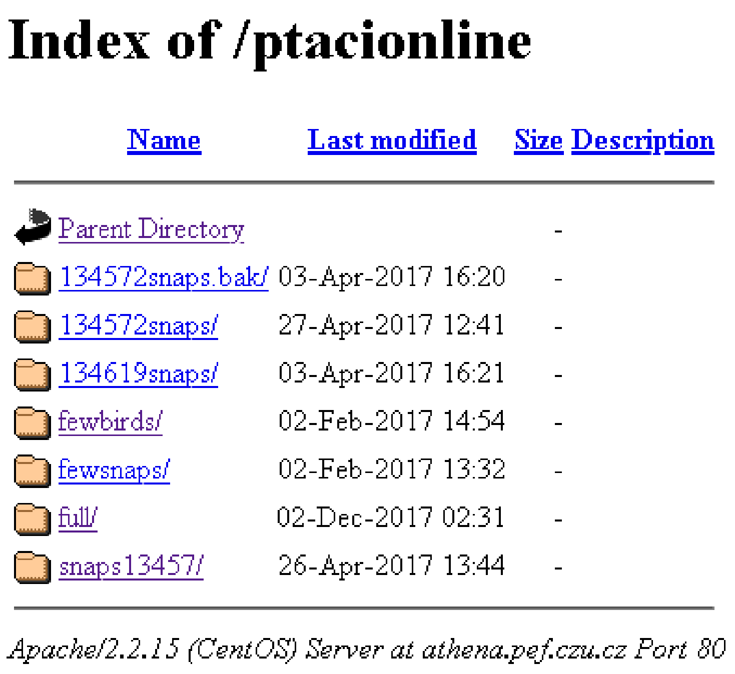
The syntax of the Python language, like the language itself, is very simple. It does not contain complex non-intuitive constructions. Therefore it is quite simple to learn.

Python syntax guidelines:

- The end of the line is the end of the statement (no semicolon is required).
- Nested instructions are combined in blocks according to the indentation size. The indentation can be any, the main thing is that the indentation is the same within one nested block.
    - Nested instructions in Python are written according to the same pattern when the main statement ends with a colon followed by a nested block of code, usually indented below the line of the main statement.

3.4 Data used

From my thesis supervisor, Ing. Josef Pavlíček, Ph.D. I gained acces to the Apache/2.2.15 (CentOS) Server at of the food-bearing photos on: http://athena.pef.czu.cz/ptacionline/ .



Index of /ptacionline

Name	Last modified	Size	Description
Parent Directory		-	
134572snaps.bak/	03-Apr-2017 16:20	-	
134572snaps/	27-Apr-2017 12:41	-	
134619snaps/	03-Apr-2017 16:21	-	
fewbirds/	02-Feb-2017 14:54	-	
fewsnaps/	02-Feb-2017 13:32	-	
full/	02-Dec-2017 02:31	-	
snaps13457/	26-Apr-2017 13:44	-	

Apache/2.2.15 (CentOS) Server at athena.pef.czu.cz Port 80

Figure 9Visualization of http://athena.pef.czu.cz/ptacionline/ . (czu, n.d.)

On this server, there is a database of photos from inside Parus's nest from the year 2016. There is approximately 500 terabyte of data.

Photos represent snaps from the motion capture camera which was installed in the Tit's nest. On the photos, there are two grown Tit birds and six young birds. Hatching is also captured by a motion camera.

Hardware specifications used to capture the photoes :

Microphone in the MIC slot. "HMU0603C-65" - Standalone cable microphone. No microphone.

Event trigger in the BARRIER slot. "IRBAR" - Standard U-shaped infrared light barrier. Auxiliary hardware in the IOP0, IOP1, and IOP2 slots. "TS" - Temperature sensor. Only one piece can be used. "LTS" - Light and temperature sensor. Only one piece can be used.

Video cameras in the CAM0 and CAM1 slots. If both cameras are used offline for triggered event recording, CAM0 records prior to CAM1. "UI-1541M" - Monochrome

camera by IDS, 1280x1024 px. Offline performance only. "UI-1641C" - Color camera by IDS, 1280x1024 px. Offline performance only. "UVC" - Generic USB Video Class camera. Online/offline. No night vision camera, there was no need to have to take photos in night mode, because given birds do not feed at night.
Radiofrequency identification reader in the RFID slot. "ELB149C5M" - 125 kHz RFID reader.
Wifi modem assembled on the SQM4-VF6 module. "AR4100" - Qualcomm Atheros AR4100 SIP (SQM4-VF6-W module).
On-board temperature sensor. "MCP9804" - 0.25-degree temperature sensor.

Sleep (power-saving) mode settings. When the system is in this mode, the event trigger, RFID reader, and offline cameras are powered off, disallowing to record any video. Up to three intervals applicable - separate the values by commas. Start time of the sleep mode. Is from 18:00. The end time of the sleep mode: 04:00
Automatic records upload to the remote data server. Up to three intervals applicable - separate the values by commas. The start and end times of data upload are from 22:00 till 04:00. It was necessary to upload every day the photos to the server. Because of the huge amount of data. It is impossible to locally save all data into the camera.

4 Practical Part

In the practical part, will explained all the main practical steps in order to complete the work. The first step, the most important part was is defining and analyzing the problem of the work Birds On Line, with help of my supervisor we did this part with excellence and we got the smart solution for the right problems. The solution was to create artificial intelligence and teach it how to recognize whether birds are feeding the nestlings or not, on the photos user will give to it. This solution combines two separate knowledge for the artificial intelligence, first, it has to recognize if there is food on the photo. Second, it also has to understand if nestlings have open mouths. If these two conditions are met we can declare that birds are feeding the nestlings.

To complete the above-mentioned solution I had to start from the beginning. The beginning was to get photos of the nest with nestlings. Those photos were provided by the Thesis supervisor: Ing. Josef Pavlíček, Ph.D. who gave me access to the *http://athena.pef.czu.cz/ptacionline/*. On this server is located huge data of photos, taken inside of Tits nest. From this huge data, I have to gather around 800 photos on which there are clearly visible food inside the nest and nestlings' open mouth. It has to be this amount because camera captured the hatching of nestlings and their growth as well, so nestlings do not look similar in every photo, and for the artificial intelligence newly hatched nestlings open mouth and open mouth of the grown nestling are different photos. For this purpose, I had to go through almost all the data and gather 800 right photos for the next step, which is to label those photos.

To train the Artificial intelligence I had to convert those actions(open mouth and food in the nest) in camera captured photos into the numbers understandable for our neural network called YOLO(you only look once). And for the purpose to convert actions into numbers I had to label the exact action on every photo and converted it to the YOLO format. For this purpose, I used a graphical image annotation tool LabelImg, which can convert labels into the YOLO format.

After labeling I had to get ready to train my custom object detector, for which I had to make several files train.txt, test.txt and classes.names, detailed information about the files I will explain below in this work. Briefly, those files are names and the location of the output of the previous step(labeling images and conversion into the YOLO format) gathered together. This step is needed to train our custom dataset and get the file named weights with help of a darknet detector. Weights file is the combination every before step into one file. This is the "brain" of the artificially intelligent.

With the weights file, I can proceed to the last step. Which was creating the python code. Code has to be able to get photos from the user (the user has to save files to the given location) go through our pre-trained weights and detect the actions we taught our network. After detection code gives back photos to the user with squares around the found action. On these squares is the information about which action they represent.

Every step with further details will be explained in the practical part

4.1 Gathering data and Labeling them

From Thesis supervisor: Ing. Josef Pavlíček, Ph.D. I got access to the server *http://athena.pef.czu.cz/ptacionline/* where is all the photos taken inside birds nest by motion camera. Approximately 500 terabytes of photos are gathered on this server. Because of the huge number of photos taken by a motion camera and their size, processing the data was highly time-consuming. After gathering the information-carrying photos from all the data, it was necessary to label the predefined actions on all of the photos, which are nestlings' open mouths and appearance of food on photos. Last and the most significant part of the gathering and labeling data was to convert them into a useful format, which is the YOLO format.

Gathering data

The first and one of the most significant steps was to understand what kind of data was useful for this project.

With help from my supervisor: Ing. Josef Pavlíček, Ph.D. I decided to choose two types of photos: 1. When food appears in a photo, 2. When nestling has an open mouth.

We chose those two labels because those two actions indicate that birds are feeding the nestlings. These two actions combined give one action, which detection we need to teach to Artificial Intelligence.

EXAMPLES:

1. When grown birds have food inside the nest. On the photo, it should clearly be visible that the bird has a portion of food in its beak.

Example:

GOOD:

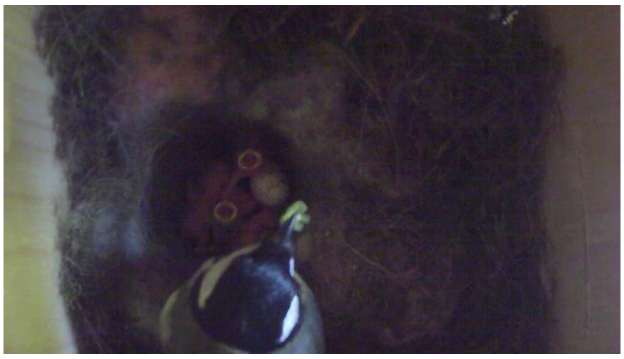


Figure 10 example of food inside

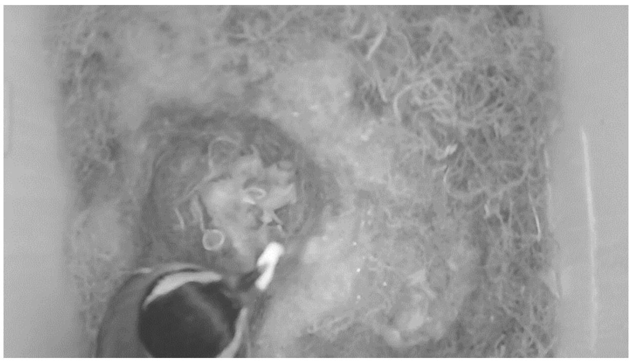


Figure 11 example of food inside

BAD

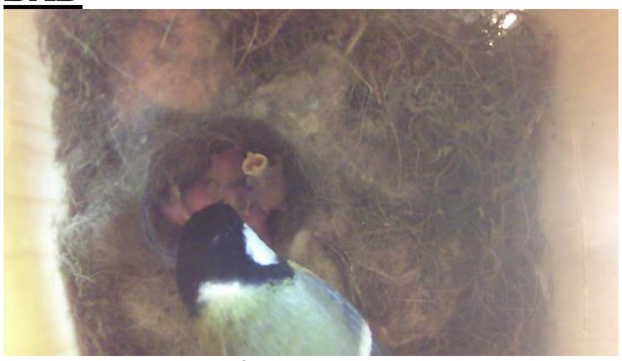


Figure 12 Bad example food inside

2. Action when nestling has an open mouth on a photo.

Example:

GOOD

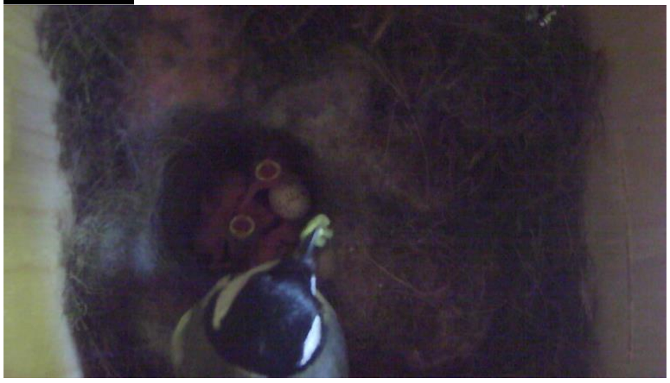


Figure 13example open mouth

<u>BAD</u>

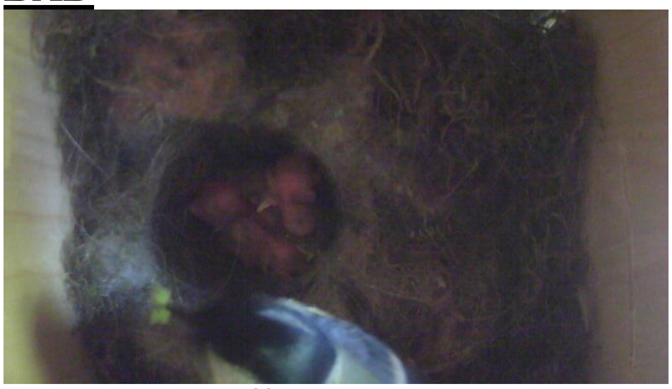


Figure 14 Bad example of food inside

4.2 Labeling Images

After I gathered the correct database of photos, I had to find two ‘actions’: open mouth and food appearance in photos and convert them into machine understandable code to proceed with teaching artificial intelligence how to recognize those actions in a user given photos. After finding them I had to label them and save them into the correct format.

For the labeling purpose, I needed software that would be able to do all the above mentioned, after consulting those needs with my Thesis supervisor: Ing. Josef Pavlíček, Ph.D. he gave me a solution as a graphical image annotation tool LBLImg. Which is user friendly and highly fast to work with. I had to go through its annotations to work with it properly.

I named two labels(classes): open_mouth and food_inside and proceed with the work.
In this phase, I had to go through all the gathered photos which were around 800 pieces, square the right action for the project and label them accordingly what actions those were. After processing each photo I had to save one by one into the YOLO format which is a .txt-extension where is written an object number and coordinates on the photo(an example is given below).
The output of the labeling photos is those .txt extensions with YOLO format for our actions open_mouth and food_inside.

Steps for labeling the photos:

LBLImg is highly easy to use, with knowing what to expect from the software. After realizing what was needed for the project I used software according to those needs.

After opening the software we see that it has a graphical interface that is highly user-friendly. Every button has discretion and icon so users can work fast by checking specifications only with peripheral sight.

Below is an illustration of how I used this program for the thesis.

STEPS:

1. Step is to choose the directory where the photos are stored and the directory where the LBLImg should save the YOLO format .txt files. For choosing the direction where photos are stored we only have to click on the icon with Open Dir and we will see the new window (depends on what OS user is using) and from this window, we can choose the directory. The same way is chosen directory where to save YOLO format labels(in .txt extension).

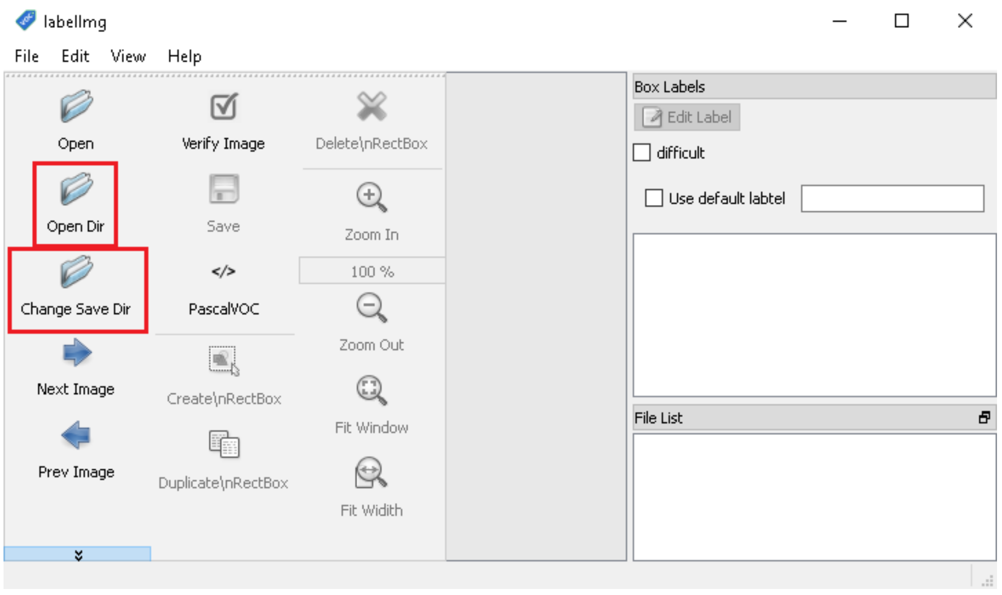


Figure 15 practical visualization LBLImg

2. The most important part is to choose the right format for the project. For the thesis, I needed to choose the YOLO format.
Choosing a format is by clicking the format button, which does not indicate that it is the format, only the format name, after clicking on the format name it changes to a different format and we need to click on it as long as the YOLO format does not appear(as shown on the screen below).

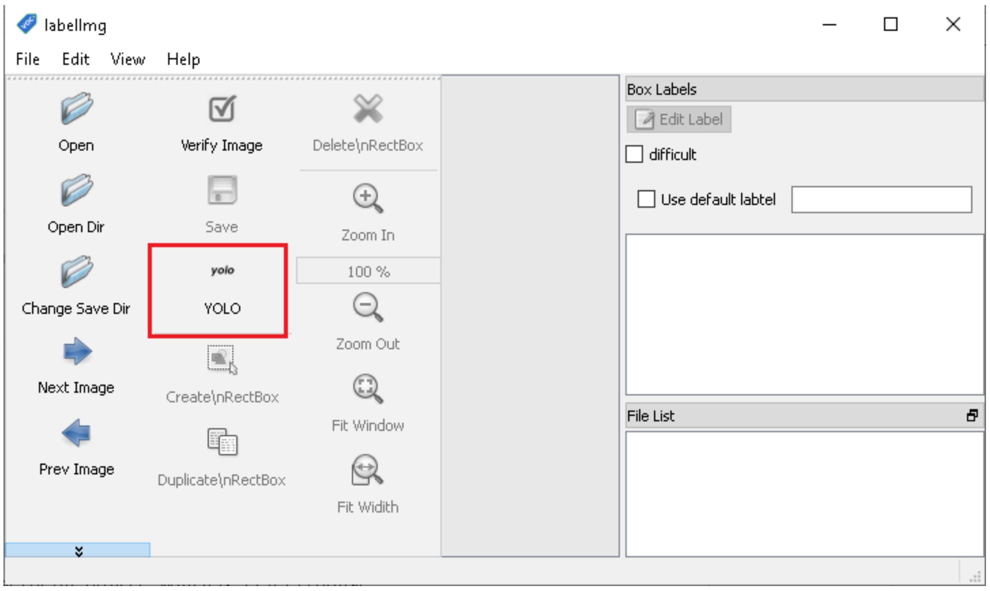


Figure 16 Practical visualization LBLImg

3. Step. After setting up all the necessary settings I opened the dictionary by clicking the open icon and chose the first photo. The chosen photo will display in the software and I already can start labeling actions on the chosen photo. After done labeling the current photo I changed the photo by clicking on the next image which has right direction arrow.

How I labeled images:

1. Have to find an exact area which we want to label. This is the area where the action is displayed.
2. Then click on Create/nRectBox, which allows us to select the chosen area.
3. I selected the action area by clicking and dragging the cursor.
4. In this step, we have to give the name of the label we defined(we have two labels: open_mouth and food_inside) .

5. After selecting all areas and made the labeling of them, I have to save the YOLO file, by clicking on the save button so that LBLImg and without choosing directory(because I have chosen the save directory in the first step) save it to the chosen directory.

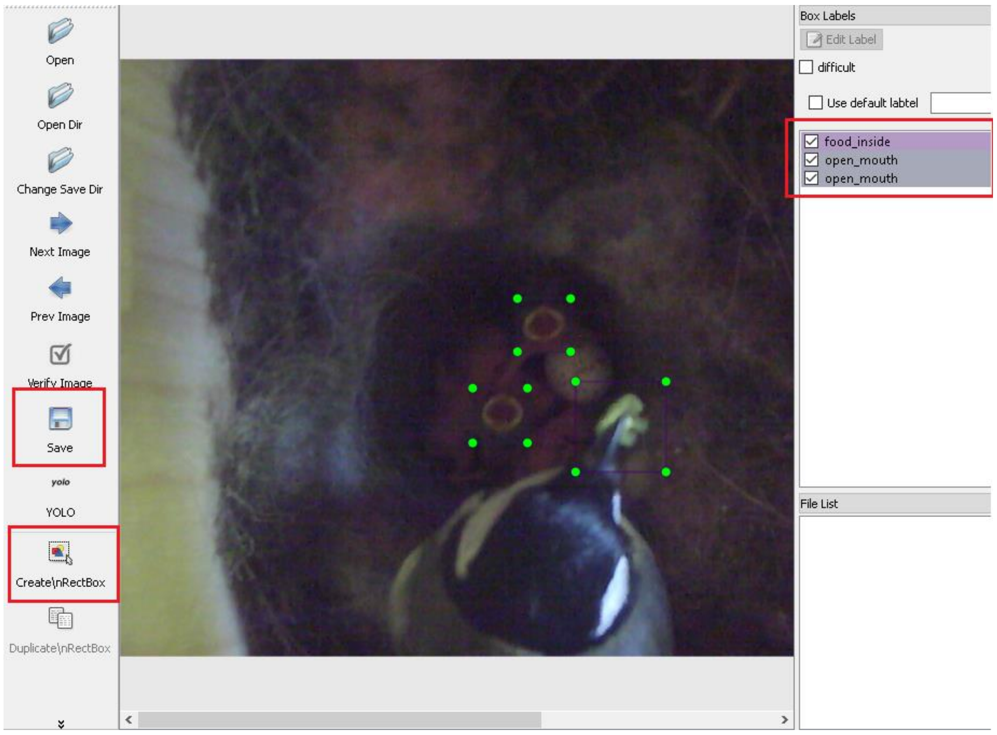


Figure 17Practicat visualization LBLImg

4. After saving the labels in YOLO format, LBLImg will generate a .txt file which will be used to tech algorithm.

.txt-file for each photo-file in the chosen directory and with the same name as the photo. With .txt format, and put in to .txt file object number and object coordinates on chosen photo, for each action in new line: <object-class> <x> <y> <width> <height>.

Explnation of line:

- <object-class> - is integer number of object from **0 to (classes-16)(**in my project there are 2 classes but LBLImg itself had 14 its own cases(label names), so custom lables got the class numbers 15 and 16**)**
- <x> <y> <width> <height> - float values relative to width and height of image, it is in between 0.0 to 1.0.

- Example: <x> = <absolute_x> / <image_width> or <height> = <absolute_height> / <image_height>
- We have to know that <x> and <y> - are not the top left corner but the **center of the rectangle**.

For example for photo1.jpg you will be created photo.txt containing:

```
15 0.470703 0.614583 0.085156 0.151389
16 0.357031 0.595833 0.051562 0.091667
16 0.398438 0.444444 0.050000 0.088889
```

5 Train a deep neural network YOLO to recognize food

After all the previous steps I could start training the model using the photos and labels I created earlier. For model training, I had to implement this data into architecture.
This architecture I use is proposed by Joseph Redmon and is called YOLO. This is an abbreviation for You Only Look Once, which in English means "You only watch once." That is, the digital image data passes through the neural network only once. Due to this, the predictive model is productive and analyzes up to 60 frames per second.

Even though I trained the model on my own data set, it was beneficial not to train it from scratch, but to use the “transfer of training". That is, use as a starting point the weight of another, already trained model. So I have used the ready architecture YOLO and changes adopted it to my use. This means changing the training batch, change the number of filters accordingly to my dataset, and created two necessary files **classes.names.txt**(contains names of classes(labels)) and **obj.data**. (includes parameters for training) and save those two files in the location ./darknet/cfg/.

classes.names.txt example:

```
open_mouth
food_inside
```

Obj.data file includes all the parameters for training. Trai.txt and test.txt files are combination of all YOLO format output of the labeling photos. I have put 95% of it in train.txt nad rest 5% into the test.txt which is created to test the above mentioned 95%.

obj.data example:

```
classes= 2

train = \Desktop\ALL DATA + LBL\train.txt

valid = \Desktop\ALL DATA + LBL\test.txt

names = \Desktop\ALL DATA + LBL\classes.names

backup = backup
```

Because training, which itself implies a weight file, needs huge memory and graphics processing unit. For this reason, I had to use Jupyter Notebook to train the model and generate a custom weight file.
For training itself, I used Darknet which is an opensource neural network framework that allows CPU and GPU computation. I installed the darknet. And checked it correctness.
After all the necessary files and installations I continued with training. After darknet installation, I only need to save all necessary files into the location **./darknet/cfg/.**
To start training the deep neural network I executed the darknet by command line.
Line: `./darknet detector train cfg/obj.data cfg/yolov3-tiny.cfg darknet53.conv.74`

As long as the weight file is huge, it was saving by iteration of 100 till the 900 and after 900, every10 000.
The result of the training is a weight file that will bet the brain of my project. After running the python code and giving it the photo, the python script will go through this huge file and will determine what it sees.

The weight file, Which will be used to run the YOLO detection with my own dataset.

6 Using custom trained database in YOLO

This is the final stage of the project. I used already existing python code and rewrite it for my thesis project. The code itself uses an input photo which the user will save to the given directory and the weight file directory. Code should open the photo and analyze it with the help of the weight file where is the all information needed for the Artificial Intelligent to determine whether or not on the given photo are the actions we predefined at the beginning of the project. Which are nestlings open mouths and food appearance in the nest. After analyzing the photo code will give the user the same photo to the pre-defined directory only with the colored squares with the label names on it accordingly to what those squares are bounding. Examples will be shown below.

I created photo input, and photo output directories and wrote them in the code for the little above then amateur user. Users will have to go to the given directory put photo and run the script. The user will be notified that the script has finished and the output photo will appear in the directory for output photos.

Those are important parts of the code(for code itself please see the appendix):

I used the necessary **python packages** for the project: NumPy, argparse, time, cv2, and os. After importing the packages it is important to create the **argument parse** and **load the customer class labels** our model was trained on.

For a better experience, I chose colors to represent each label. So in the next part I **initialized a list of colors** to represent each class, open_mouth is red and food_inside is yellow.

The main part is to give brains for our model, so in the next part wrote a path to the weights file which was generated from YOLO training in the previous step and model configuration. and load our YOLO object detector trained on our dataset.

After all the preparations we can write input code for photos the user needs to test.

Then we are creating code for boxes with a name and color to create a blob from the input images and pass of the YOLO object detector, giving users our bounding boxes with all information needed.

7 Perform the test and evaluate the result

Before starting generating my own dataset I had to control all the steps by already created weights to be sure that I was going on the right path. It was part of my training for this project because I was a beginner in Artificial Intelligence training and in Python programming language.
Tests had to be performed after the project's every step because everything from the beginning had to be the correct to create the weight file and after its creation, I had to control the python script for every bug and misspells of the words. The Crucial was to chose also packages for the python. After all the preparation tests I finally could perform the conclusion test for the project.

First I tested if the project works, checked every line of the code, and performed the test. First I have put the photo which I wanted to test in the directory for the input photos and after I ran the code. After a couple of tries and debugging the first test performed as expected. I got the output of the same photo with the named squares bounding the area with the same action as the square name was. So Test was accepted and the project was finished.
To test for my project from the user's view and determine what kind of knowledge is needed for the user to have I performed a test on 2 participants. The first Participant failed because of the lack of knowledge of the python language and the second performed well. So the output of the test was that the user has to have some experience in the python language to perform the object detector.

The test was performed on 50 photos. On every photo, the object detector detected the actions and gave the imputed photo back with the information (square bounding the action, with the name of the according label of the action).

Output photos:

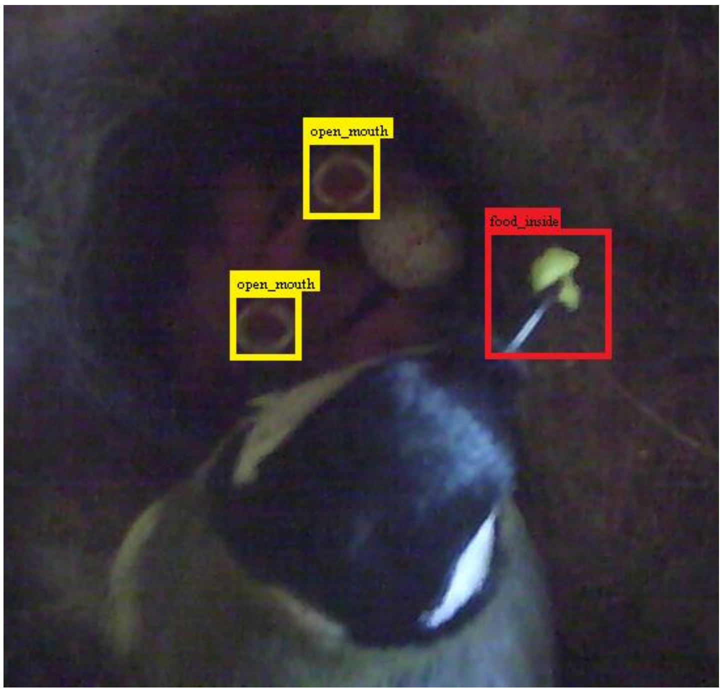


Figure 18 YOLO test output

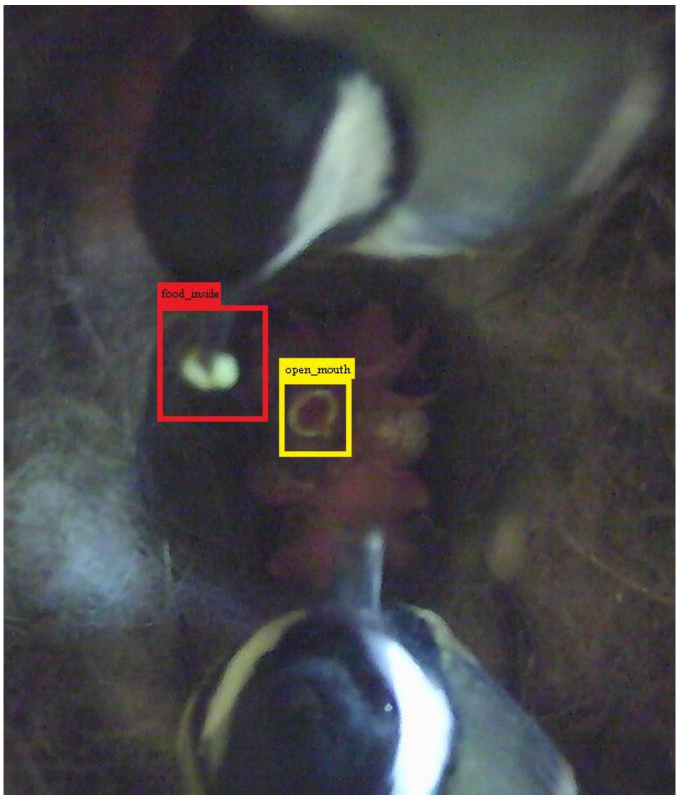


Figure 19YOLO test output

8 Conclusion

This section compiles the work done by me and gives a deeper understanding of the research purposes, its results, and meanings of those results for future researches.
In conclusion will be briefly given the main findings from my work, methods I used to complete the thesis, results, and problems connected with the completion of the thesis. The conclusion is concluded by analyzing how and why this work will help me in my future researches.

Before starting the main points I would like to thank my thesis supervisor Ing. Josef Pavlíček, Ph.D. who gave me the opportunity to work on this interesting topic, for the consultations and notes, which he gave me to complete the thesis.

8.1 Findings

I chose this topic to emerge my knowledge in Python language coding and for the understanding of how artificial intelligence works.
The main purpose of my work was to understand how to work with artificial intelligence, how to teach it to certain functions, and gain the needed output from it. In my case, I had to teach AI how to see what actions are shown in the photo. To teach Artificial intelligence to see the action I got the data, in this case, photos from the tits nest.
Working with artificial intelligence is fragile, because of its sensitivity. Every step should be beforehand analyzed and controlled to reach the goal of the project.
One of the best findings for me was working in the python language. This was the first time for me to code in Python language and it left a huge impact on me, with its simplicity and power of its packages. Because of it, I'm continuing my development in this language and already got a huge step up from the beginning of my work with it.

8.2 Methods Used

To complete the thesis I used the detection method called YOLO (You Only Look Once). This method is a single shot detector which means it only looks once on the given shoot and the response is rapid. It is the fastest method I found to work with, it even makes real-

time inference possible, so it was the best method to use for thesis and for my future development as an IT specialist. YOLO itself uses Darknet-53 as the feature extractor.
Data preparation was a huge part of the thesis. I got access to the photos taken inside of the tits nest with 6 nestlings. In the preparation part mainly work was done manually. I had to manually go through the photos and choose the ones suitable for the thesis, I had to collect around 800 photos where was shown exact actions I wanted Artificial intelligence to detect.
After I got the collection of the photos I had to label them into the YOLO format, for this purpose I used LabelImg. LabelImg is a tool for graphical image annotation. It has a user-friendly interface. All work in LabelImg should have to be done manually. It means that I had to go through all 800 chosen photos and on every single photo. On every photo, I Selected the above-mentioned actions on the photo and name them accordingly. Output was the weight file which is the brain of the python code(the sight understanding for the machine).
The weight file was used together with the python code to detect actions on the given photos. Photos are given to the code by a user. The user has to save photos to the directory from which python code takes input photos and after running the python program, the output photo will be saved into the output directory.

8.3 Results and Problems

As result oh the thesis I got fully functioning program which detects nestlings open mouth and appearance of the food inside the nest. If those two actions will be detected on the same given photo we can 95% say that birds are feeding the nestlings.
Results are far more than just this project, this code and methods can be used to make much bigger projects, for example, we can implement the same weight file to the python script which will detect actions on real-time streaming videos.
The problem that consumed most of the thesis preparation time was the manual work done by me to complete the program. T to find the necessary photos and labeling them was the hardest and routine work that had to be done to gain a custom dataset for deep learning.
In the beginning, the problem also was an understanding of python, because this was the first time I ever worked in this language, but this problem slowly but strongly faded away.

The problem also was the device speed because to create a weight file, I had to use a Jupyter Notebook to overcome this problem.

8.4 Future Research

My future research is deeply connected with the python language, I am already working on the automatization of data processing, which itself combines the spread package of the python which is the connection of the server or database or simple os to the google spreadsheets. This connection can be used in various ways, imput data can be from any data source thanks to the Python packages, as well as the output.

Also, I am planning to deeper my knowledge of teaching artificial intelligence. Not only in the way this thesis was done but in the data processing way as well. I plan to teach AI how to process data, how to react to certain changes in the input data, and give output for the human user in a preferred way.

Direct output from the thesis is to continue research on image recognition. On software which can be uploaded on the cloud and can be distributed to the smart cameras by the internet. This way software can detect in real-time, not only birds feeding but anything that will be taught to the AI

I see this project as one of the key points in my studies at the university. Because its implements in itself teaching a machine how to be like a human and see things. People use their eyesight every day to recognize familiar faces, notice obstacles in their path, and broaden their horizons. Today we live in the era when machines also can be taught the say functions, for example, self-driving cars, etc. The future of IT is in machine learning and machine communications between each other.

Computer vision describes the process where it’s using an artificial intelligence algorithm that can identify and process images (photographs, video, etc.), and since the computer “understands” the content creates the corresponding analysis results afterward. In particular, computer vision can classify, identify, verify, and detect objects. Machine learning technologies have promoted the development of computer vision, in particular, the iterative process of learning neural networks and significant leaps in computing power, data storage, and high-quality but inexpensive input devices.

Computer vision, thinking, and speed of communication is one of the most sought after areas at this stage in the development of global digital computer technology. It is required in many areas, even in the areas we have never thought the machine could do good work. But with the fast-growing new technology, which is getting smaller and faster every day, we can overcome every obstacle and make a smart intelligent machine that will benefit mankind.

9 Reference

Angelova, J. R. (2015). Real-time grasp detection using convolutional neural networks. Seattle, WA, USA.

Davies, E. R. (2017). *Computer Vision: Principles, Algorithms, Applications, Learning (ISBN-13: 978-0128092842).*

Forsyth, P. (2011). *Computer Vision: A Modern Approach (2nd Edition) (ISBN - 9789332550117).*

Hartley, R. (2004). *Multiple View Geometry in Computer Vision .*

Johnson J., K. A. (13.05.2018). *Convolutional Neural Networks for VisualRecognition.* Načteno z http://cs231n.github.io/convolutional-networks/

Joseph Redmon, A. F. (2017). *YOLO9000: Better, Faster, Stronger*. Načteno z https://pjreddie.com/darknet/yolo/.

Mahendran, V. (2019). *Custom object training and detection with YOLOv3, Darknet and OpenCV.* Načteno z https://blog.francium.tech/custom-object-training-and-detection-with-yolov3-darknet-and-opencv-41542f2ff44e

Minichino, J. H. (2020). *Learning OpenCV 4 Computer Vision with Python 3.*

Ponnusamy, A. (2019). Načteno z Preparing Custom Dataset for Training YOLO Object Detector: https://www.arunponnusamy.com/preparing-custom-dataset-for-training-yolo-object-detector.html

Prince, S. J. (2012). *Computer Vision: Models, Learning, and Inference (ISBN-13: 978-1107011793).*

Redmon J., D. S. (2015). You Only Look Once: Unified, Real-Time Object Detection // Computer Vision and Pattern Recognition (ISBN: 978-1-4673-8851-1).

Solem, J. E. (2012). *Programming Computer Vision with Python: Tools and algorithms for analyzing images (ISBN-13: 978-1449316549).*

Szeliski, R. (2010). *Computer Vision: Algorithms and Applications (Texts in Computer Science) (ISBN-13: 978-1848829343).*

Tomassini, A. T. (2001). *Soft Computing: Integrating Evolutionary, Neural and Fuzzy Systems (ISBN 978-3-662-04335-6).*

Vaidya, B. (2018). *Hands-On GPU-Accelerated Computer Vision with OpenCV and CUDA: Effective techniques for processing complex image data in real time using GPUs (ISBN-13: 978-1789348293).*

10 Apendix

10.1 Python code for custom trained database in YOLO

```
# import the necessary packages
import numpy as np
import argparse
import time
import cv2
import os

# create the argument parse, parse the arguments
ap = argparse.ArgumentParser()
```

```
ap.add_argument("-i", "--image", required=True,
        help="path to input photo")
ap.add_argument("-y", "--yolo", required=True,
        help="base path to YOLO")
ap.add_argument("-c", "—confidence ", type=float, default=0.5,
        help="min. probability to filter weak detections")
ap.add_argument("-t", "--threshold", type=float, default=0.3,
        help="threshold when applyong non-maxima suppression")
args = vars(ap.parse_args())

# load the custom class labels our YOLO model was trained on
labelsPath = os.path.sep.join([args["yolo"], "classes.names"])
LABELS = open(labelsPath).read().strip().split("\n")

# initialize a list of colors to represent each possible class label
np.random.seed(42)
COLORS = np.random.randint(0, 255, size=(len(LABELS), 3),
        dtype="uint8")

# the paths to the YOLO weights and model configuration
weightsPath = os.path.sep.join([args["yolo"], "yolov3.weights"])
configPath = os.path.sep.join([args["yolo"], "yolov3.cfg"])

# load our YOLO object detector trained on custom dataset (2 classes)
print("loading YOLO")
net = cv2.dnn.readNetFromDarknet(configPath, weightsPath)

# load our input image
image = cv2.imread(args["photo"])
(H, W) = image.shape[:2]

# define only the output layer name from YOLO.
ln = net.getLayerNames()
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]

# create a blob from the input images and pass of the YOLO object detector, giving user
our bounding boxes.
blob = cv2.dnn.blobFromImage(image, 1 / 255.0, (416, 416),
        swapRB=True, crop=False)
net.setInput(blob)
start = time.time()
layerOutputs = net.forward(ln)
end = time.time()

# timing data on YOLO
print("[INFO] YOLO took {:.6f} seconds".format(end - start))

# initialize lists of detected bounding boxes, confidences, and class IDs, respectively.
boxes = []
```

```
confidences = []
classIDs = []

# loop over each of the layer outputs
for output in layerOutputs:
	# loop over each of the detections
	for detection in output:
		# get the class ID and confidence of
		# the current object detection
		scores = detection[5:]
		classID = np.argmax(scores)
		confidence = scores[classID]

		# filter out weak predictions by ensuring the detected
		# probability is greater than the minimum probability
		if confidence > args["confidence"]:
			# scale the bounding box coordinates back relative to the
			# size of the image, keeping in mind that YOLO actually
			# returns the center (x, y)-coordinates of the bounding
			# box followed by the boxes' width and height
			box = detection[0:4] * np.array([W, H, W, H])
			(centerX, centerY, width, height) = box.astype("int")

			# use the center (x, y)-coordinates to derive the top and
			# and left corner of the bounding box
			x = int(centerX - (width / 2))
			y = int(centerY - (height / 2))

			# update our list of bounding box coordinates, confidences,
			# and class IDs
			boxes.append([x, y, int(width), int(height)])
			confidences.append(float(confidence))
			classIDs.append(classID)

# apply non-maxima suppression to suppress weak, overlapping bounding
# boxes
idxs = cv2.dnn.NMSBoxes(boxes, confidences, args["confidence"],
	args["threshold"])

# ensure at least one detection exists
if len(idxs) > 0:
	# loop over the indexes we keep
	for i in idxs.flatten():
		# extract the bounding box coordinates
		(x, y) = (boxes[i][0], boxes[i][1])
		(w, h) = (boxes[i][2], boxes[i][3])

		# draw a bounding box and label on the image
		color = [int(c) for c in COLORS[classIDs[i]]]
```

```
                    cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
                    text = "{}: {:.4f}".format(LABELS[classIDs[i]], confidences[i])
                    cv2.putText(image, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
                              0.5, color, 2)

# show the output image
cv2.imshow("Image", image)
cv2.waitKey(0)
```