Machine Learning Project Documentation

Team Members

Tobias Lukasewitz, 4659023 (inf20060@lehre.dhbw-stuttgart.de)
Niklas Elsässer, 4842156 (inf20184@lehre.dhbw-stuttgart.de)
Jannik Peplau, 1995581 (inf20017@lehre.dhbw-stuttgart.de)

Abstract

Goal of this project is the creation of a machine learning algorithm that can be provided with an image of unsorted coins in the EURO currency and classify the coins of different value, as well as to visualize the results.

Introduction

Machine learning algorithms are able to process large amounts of data, learn from set data and improve their recognition performance over time. With these advantages, Computer Vision is ideal to recognize patterns and features in images for tasks like detecting coins in images. In a situation where manually counting coins would be time-consuming, for example a vending machine, Computer Vision is capable to detect set coins more accurate and efficient than traditional methods.

To build a Programm which recognizes coins in Pictures, it is neccesary to have a Database to train the Neural Net on. To achieve the needed amount of pictures mulitple pictures of Various Euro Coins, from 50 Cent, 1 and 2 Euro were taken.

Documentation

Collecting Data

As a first step, data needs to be gathered. In the example of this machine learning algorithm, the data consists out of self-made photos of different coins and different values of the EURO currency. In order to reduce the time the model takes to fully train, this algorithm will only be able to distinguish 2 EURO coins, 1 EURO coins and 50 CENT coins.

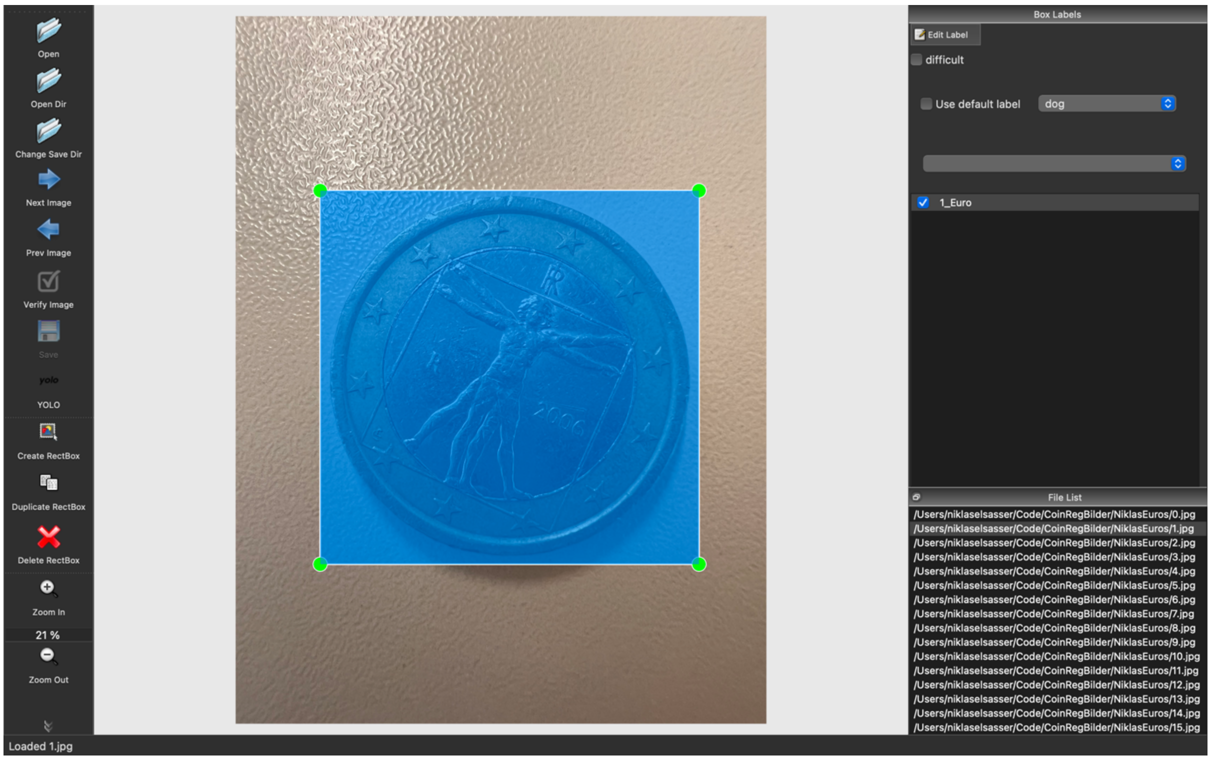
When taking the images, it is important to either automatically or manually downsize the picture quality to save on training time. In most cases this will have little to no impact on the actual accuracy, but it will speed up the training process by a lot.

Labeling

In the next step, the gathered data needs to be labeled, so that the model can learn the coins. An example of software used to label such data is LabelImg, as used in this algorithm.

On every image, the area that contains the coin is first highlighted, then a specific name is given to the image. This area is also referred to as a bounding box.

The following image shows the process of labelling the image of a 1 EURO coin. First, the area of the coin is highlighted to allow for the most accurate recognition of the oject. In the next step, a label is assigned to this specific highlighted region, as it is possible to label multiple objets within a single image. This process is repeated for the entirety of the dataset which is to be used for the training process.



☑ 1_Euro

```
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/0.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/1.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/2.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/3.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/4.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/5.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/6.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/7.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/8.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/9.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/10.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/11.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/12.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/13.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/14.jpg
/Users/niklaselsasser/Code/CoinRegBilder/NiklasEuros/15.jpg
```

Loaded 1.jpg

Dividing Data into Train- and Test-Set

In order to train the algorithm, the gathered data needs to be split into two sets. All data within these two sets needs to be manually labelled.

The training set is used to train the algorithm to recognize the different types of labeled objects, in this case the different EURO coins.

The test set consists out of previously unseen data, new to the algorithm. It too is labeled, so that it can be evaluated, wether the algorithm did actually classify the objects in it correctly.

The two sets are split 80/20, 80 percent of the labeled data goes into the train set, 20 percent of the data goes into the test set. This allows for accurate training with enough data, but at the same time ensures, that enough labeled data can be used to validate the performance of the algorithm. For the most accurate results it needs to be assured that the different types of objects are equally spread over the two sets. If a certain object only appears within either the training- or the testset, the algorithm cannot work properly.

Training of the model
For the training process the darkweb software is used. This software requires a predetermined config file, which sets the basic outline parameters of the training process. Also, a file containing the weights needs to be provided. The training process itself then generates a new custom weights file to be used later, which due to it’s size is not contained in the repository.
The training process is based on the following tutorial: https://youtu.be/hTCmL3S4Obw

Testing the model
After the model has been trained using the training set, it can now be tested with the test set. The data in this set, as previously mentioned, differs from the data in the train set to assure that the results are not forged through overlap of data.

Evaluating the model
To evaluate the accuracy and usability of the newly trained model, a new python notebook is created. This notebook utilizes pythons OpenCV library create a Convolutional Neural Network from the configuration and weights file from the training process. Using these files, it can then analyze an image passed along to it and recognize the objects that were trained using the labelled data.
The algorithm is based on the following tutorial: https://youtu.be/zm9h4mYymk0

Developed Code
Data
The Data, images of Coins in 4K Resolution, was used to train the Algorithm after all the Images were manually labeled with the LabelImg-Tool. The images were saved in a separate Folder and the labeling process linked a *.txt* file with the coresponding properties to each picture. Since the folder with the labeled images is too large for GitHub this Repository only holds a example folder named *YoloEuroImages*.

The labeled images got split into train and test files with each file holding the path to the coresponding *.txt* file. Also in the folder is the *classes.names* file, converted from the classes.txt file, which holds the class names like: 50_Cent, 1_Euro and 2_Euro.

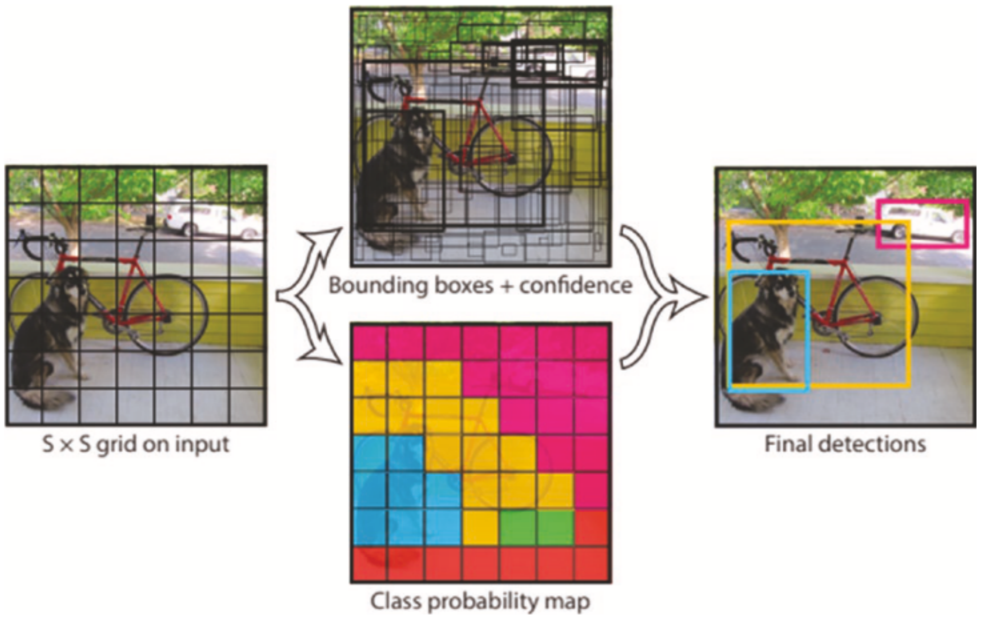
The file *labeled_data.data* holds the amount of classes as well as the path to the train, valid and name files, while also specifying where the trained weights get saved to.

Model Architecture
Yolo (You Only Look Once) is the model architecture chosen for the Coin Detection Project because it is quite easy to use, it allows to detect multiple objects objets and predicts the bounding boxes and probabilities of classes of objects at the same time. [1]

The detection Process works as follows[1]:

1. Dividing the image into *SxS* grids.
2. Extraction of features like, edges, textures and patterns

3. The image is divided into a grid of cells, where each cell makes predictions of a set of bounding boxes, and class probabilities.
4. Calculating the confidence score as follows:
Confidence score = Probability of objectness x IOU between the predicted box and the ground truth.
If the box contains no object, the confidence is zero.
5. A probability is calculated for each bounding box, indicating the presence or absence of objects within the bounding box
6. To eliminate redundant predictions, non-max suppression algorithm removes bounding boxes that have a high overlap with other boxes
7. The formula for th class spezific confidence goes as follows: **Class confidence score = Pr(Classi|Object) x Pr(Object) x IOU between prediction and ground truth.**
Pr(Classi|Object) represents the probability of a class given the object within the grid cell.[1]



The general YOLO architecture is inspired by the GoogLeNet image classification architecture, consisting of 24 convolutional layers with maximum pooling followed by two connected layers.
The used YOLOv3 architecture has the following improvements:
- 53 layers
- faster detection time[1]

Hyperparameter
YOLOv3 has the following hyperparameters:
- **Learning rate**, controls the step size of the gradient descent algorithm. A high learning rate on one hand might cause the model to converge quickly, or worse cause the model to converge to a suboptimal solution. While on another hand a low learning rate may cause the model to converge slowly or preferably also allow the model to converge to a more optimal solution. The learning rate was set to 0.001 which is the default because it offers a good balance between speed and stability.[1, 2]

- **Batch size**, determines the number of training samples used in each iteration during the training process. A larger batch size can increase the stability of the training process or worse increases the training time. A batchsize of 6000 was chosen because its recommended to multiply the class-size by 2000.[1, 2]
- **Number of training iterations**: The number of training iterations determine the number of times the training process will iterate over the entire dataset. More iteration can lead to better performance but also increase training time and computation cost. The amount of training iterations was set to around 6000 to achieve a high level of precision.
- **Weight decay**, is a regularization technique that helps the model avoid overfitting by adding a penalty term to the loss function. The chosen weight decay of 0.0005 prevents overfitting.[1, 2]
- **Number of anchor boxes**, determines the number of predefined boxes for each cell of the grid. This parameter affects the model's ability to detect objects of different shapes and sizes. The number of anchor boxes was by default set to 10. [1, 2]

Cost Function

YOLOv3 uses a multi-task loss function that combines information about the location of the objects and the classification scores. The cost function is defined as a sum of two main components:

1. Localization Loss, measures the difference between the predicted bounding box coordinates (center coordinates, width, and height) and the actual bounding box coordinates of the object.
2. Classification Loss, measures the difference between the predicted class scores (i.e. the probability that the object belongs to a specific class) and the actual class scores.

Usage

In order to use the model to classify coins on new images, the code from the classification.ipynb can be used.

Using opencv2, an image can manually be read in and is processed by the CNN. The resulting image will show boxes drawn around the detected coins, along with the name of the coin and a certainty of correctness.

Difficulties

During the training, some difficulties of unknown and undeterminable cause have occured. Even thogugh the training process finished successfully, and the test process validated the results, the generated .weights-files cannot be used to detect coins from the jupyter notebook. This issue, unfortunately, was unresolvable, therefore the functionality of the *ObjectDetection.ipynb* jupyter notebook is impaired.

Results and Conclusion

The used images with a resolution of 4K contradicted an efficient training. Although a premium GPU was used to train the algorithm it took more than 2 hours to finish.
With a class-size of only 3 it was possible to achieve respectable results with the test files.

Literature

[1]Ansari, Shamshad [Verfasser/in]. (2020). Building Computer Vision Applications Using Artificial Neural Networks: With Step-by-Step Examples in OpenCV and TensorFlow with Python.

[2] Zhou, Jun & Tian, Yichen & Yuan, Chao & Yin, Kai & Guang, Yang & Wen, Meiping. (2019). Improved UAV Opium Poppy Detection Using an Updated YOLOv3 Model. Sensors. 19. 4851. 10.3390/s19224851.