**Cracow University of Technology**

Faculty of Mechanical Engineering

**Chair of Production Engineering**

Field of study: Automation and Robotics

Specialization: Information technologies in production processes

FULL-TIME STUDIES

# ENGINEERING THESIS

**Mateusz Bugaj**

Object detection and position estimation from camera image using neural networks

Doctoral advisor:

PhD. Eng. **Piotr Lipiec**

Cracow, AY 2020/21

## Table of Contents

## 1. Goals

This project focuses on designing and training a convolutional neural network model capable of predicting the position in three-dimensional space and rotation in one chosen axis of the 3D model. The first part explains the process of generating training data for supervised learning. The second shows the architecture of the model with a focus on a different aspects of CNN and neural networks in general. The third part shows the evaluation of the performance. There are two different methods for rotation regression compared with each other. One being a naïve approach of describing the angle by number in a range from 0 to 360. The other is based on the hue circle and works by regressing rotation to 3 numbers representing RGB components. Also, one branch of the created network predicts a bounding box for the object on the image. Program for generating dataset was written in Java using Processing-core library. The neural network was created with free and open-source tool TensorFlow Keras.

## 2. Introduction

Neural networks are, in recent years, one of the most rapidly developing field of machine learning and computer science in general. They become unmatched by humans in various tasks. For example in detecting basic characteristics and segmentation of customers to predict their interests and propose adequate advertisements. In banking and finances, NNs replaced traditional statistical techniques being better at future price and exchange rate forecasting or predicting stock performance. In medicine, they are developed to help recognize diseases from body scans, develop a new medicine, and simulate protein folding. There is also big progress in the field of autonomous driving where NN’s task is to detect conditions on the road from various sensors and cameras and based on this data operate a vehicle. After being trained on labeled and/or unlabeled video footage, containing many hours of different road conditions, situations and objects there is hope that it will become better than humans at detecting and predicting next actions of other cars on road, pedestrians on sidewalks, etc. while maintaining full focus at all times. Comprehension of camera footage is crucial in many automation tasks.

Computer vision is a scientific field in which efforts focus on giving computers the ability to understand digital images and videos. In this context, understanding means transforming pixel values from given images into an adequate description that can be understood by humans or another machine. A CV includes i.e. scene reconstruction, event detection, video tracking, object detection, and 3D pose estimation.

Knowledge of the position and rotation of an object relative to the camera’s lens is an important aspect of for example programming pick-and-place robots which can be found in almost all factories or sorting centers. These machines need to have reliable information about whether the object in question is in the field of view and where it is in space. Many applications use a device like ultrasonic distance sensors or LIDARs. They are much more expensive than a regular camera and rarely solve pose estimation problem outside strict and sterile environments with traditional algorithms without neural networks. Pose estimators based on monocular cameras are cheap and flexible. They can detect objects in a changing environment even when their color is not distinct.

## 3. Related Literature

Although artificial neural networks are known since the 1950s, they weren't capable of advanced image processing until the popularization of deep learning and convolutional neural networks (CNNs) [1]. Pose estimation is considered a challenging task and cannot be solved using basic architectures like multilayer perceptron consisting of only dense layers. CNN incorporates convolutional layers which are advised for pattern recognition from images.

PoseCNN is an end-to-end model proposed by Xiang et al. for pose estimation from images [2]. It performs semantic categorization on pixels and predicts object translation. 3D rotation is estimated by regressing convolutional features to a quaternion representation. The problem of symmetric objects where identical observations are obtained for different orientations was addressed by using a custom loss function focusing on matching the 3D shape of an object. Most of the database used for training was consisting of generated synthetic images with randomly placed objects in a scene.

Similarly, Deep-6DPose uses Lie algebra for rotation representation which allows the network to predict each object independently [3].

An example of a model not trained in end-to-end fashion is BB8 in which different NNs are used for object detection and localization and 3D pose prediction [4]. Instead of rotation regression, the authors used position estimation of corners of the 3D bounding box around the object. Symmetry was solved by using angle restrictions and image manipulation.

The specific use case was proposed by Giefer et al. with CNN designed for pose estimation for apples for inspection in logistic centers [5]. Two methods of rotation estimation were considered: quaternions and Lie algebra. The database was created from real images of apples in known orientations. Due to the apple's symmetry, one axis of rotation was omitted. Rotation in two others was recorded in 10-degree intervals. Rotation around one axis was automated with a stepper motor and the other – performed manually. For evaluation of the neural network's performance, 20% of images were taken from the dataset, and the remaining ones acted as a training set. To obtain location and orientation, the authors used two-stage cascaded CNN where each network was task-specific and trained by optimizing one loss function. One part of the model predicts the bounding box of the object and the other estimates the rotation angles. Input is cropped by the pipeline so that the part responsible for predicting orientation, gets only a chunk with the object.

# 4. Neural networks

## 4.1. General Concept

In the most simplistic form, artificial neural networks, are multi-dimensional formulas that include many different parameters and nested functions. Their job is to imitate the neural system of a living organism [10]. The most basic form of a neural network model is called perceptron and can be represented as shown in Figure 1.

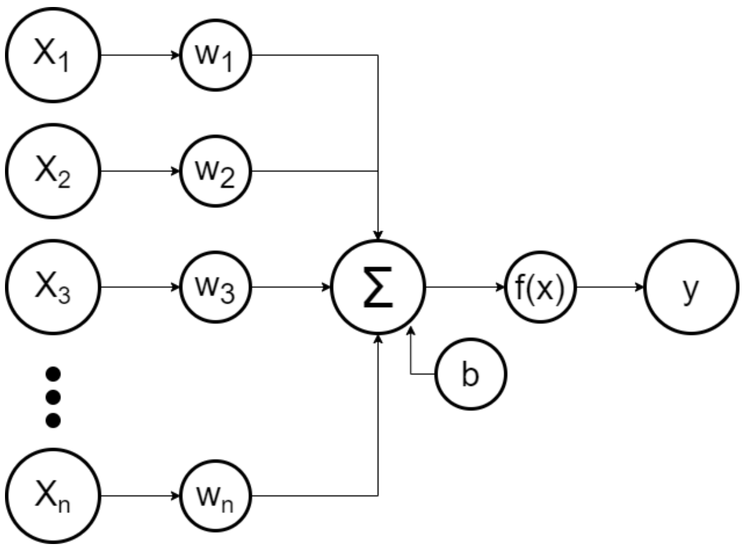


*Figure 1. Perceptron*

Every neuron can have multiple inputs and only one output. These are connections with the rest of the network and work similarly to synapses. Inputs are multiplied by weights and added alongside with bias. Bias allows output to be non-zero in the case when all inputs are equal to zero. The activation function is applied to the sum to introduce non-linearity. Output can be calculated with the following equation:

$$y = f\left(\sum_{i=0}^{n} x_i \cdot w_i + b\right) \tag{1}$$

This architecture greatly limits the capabilities of a neural network to work only in very simple categorization. To map or estimate any linear or non-linear function, there need to be more neurons combined into layers creating multilayer perceptron or Feed Forward Network with a version containing multiple hidden layers called Deep Feed Forward (Figure 2). In this configuration, the output of a neuron is passed as input to each node in the next layer or output of the network. In deep learning, this type is called a "dense" or "fully-connected" layer. It is a network in form of an acyclic graph meaning there are no loops inside and the description will focus mostly on this type as it is the one chosen for the project.

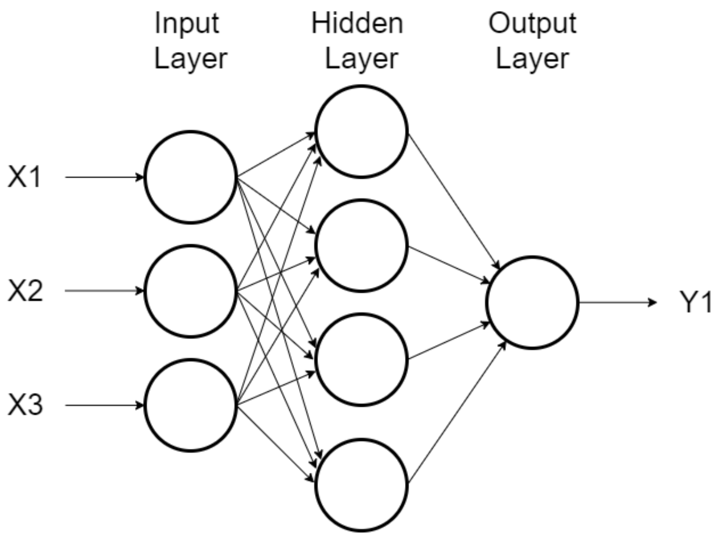


*Figure 2. Multilayer Perceptron*

Many different nonlinear functions can be used as an activation function. These are chosen i.e. based on the range of data present in the network or cost of computation. These days, the ReLU function is the most widely used, because it's easy to compute and works (Figure 3).

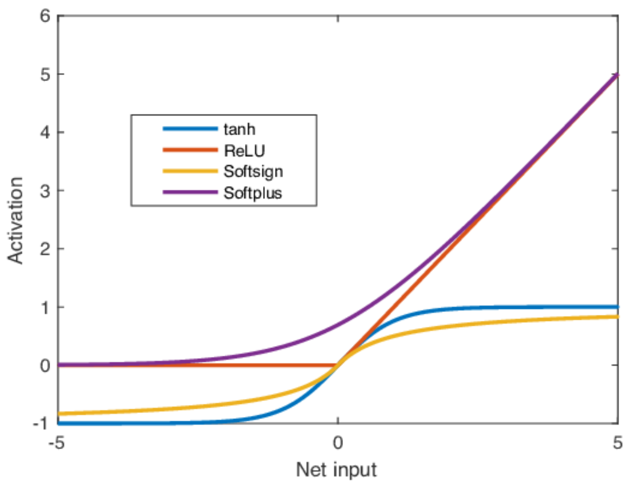


*Figure 3. Most popular activation functions*

In practical implementation, neural networks can be represented as matrix operations or tensors:

$$\begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} = \begin{bmatrix} w_{11} \cdot x_1 + w_{11} \cdot x_1 + \cdots + w_{1n} \cdot x_n \\ w_{21} \cdot x_2 + w_{22} \cdot x_2 + \cdots + w_{2n} \cdot x_n \\ \vdots \\ w_{m1} \cdot x_1 + w_{m2} \cdot x_2 + \cdots + w_{mn} \cdot x_n \end{bmatrix} \xrightarrow{f(x)} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad (2)$$

Where *n* represents the number of inputs and *m* number of nodes in a single layer.

This allows to efficiently compute weights multiplication as dot product and optionally use hardware acceleration with optimized to work with this type of data GPU.

The process of traversing through the network from the input layer to the output layer and calculating values for each node is called forward propagation or forward pass. It outputs the prediction of the network.

## 4.2. Backpropagation

Weights and biases in a neural network are called parameters. Modifying them to better fit the desired function is referred to as training or fitting. The most commonly used algorithm used for training is called backpropagation. In essence, BP computes the gradient or the derivative of the loss function with respect to the parameters of the network using the chain rule. This gives an insight into how different weights influence the value of the error between prediction and ground truth creating a multidimensional error surface (Figure 4).

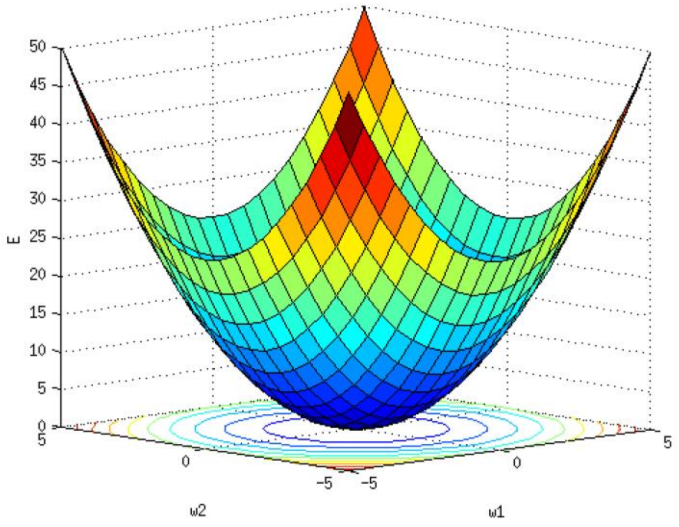


*Figure 4. Error surface of a linear neuron with two input weights [17]*

Finding the lowest point of that surface, or global minimum, is called gradient descent. As can be seen in [6], there are many algorithms based on that concept incorporating different hyperparameters describing and optimizing the way of updating the model:

a) Vanilla / Batch Gradient Descent

Vanilla gradient descent computes the gradient of the loss function L(θ) with respect to the parameters θ for the entire training dataset. Then for every weight it substitutes (or adds depending on the sign of the gradient) portion of its value, specified by the learning rate η:

$$\theta = \theta - \eta\nabla_{\theta}L(\theta) \quad (3)$$

In other words, the learning rate determines the size of the steps we take to reach a minimum. The biggest disadvantage of this approach is calculating the loss function of the whole dataset before updating the model. This makes it very slow and inapplicable for large datasets.

b) Stochastic Gradient Descent

Stochastic gradient descent updates parameters for each training example $x_i$ and label $y_i$:

$$\theta = \theta - \eta\nabla_{\theta}L(\theta\,;\,x_i\,;\,y_i) \quad (4)$$

This allows performing computations on large datasets and online, with adding new samples on-the-fly. It is also much faster, but because of frequent updates, loss function tends to fluctuate more. This is why there is another type called mini-batch gradient descent which performs updates after calculating loss function on *n* samples:

$$\theta = \theta - \eta\nabla_{\theta}L\left(\theta\,;\,x_{i;i+n}\,;\,y_{i;i+n}\right) \quad (5)$$

c) SGD with momentum

Momentum tries to solve the tendency of SGD to stop on the local minimum and not reaching the global minimum (Figure 5).

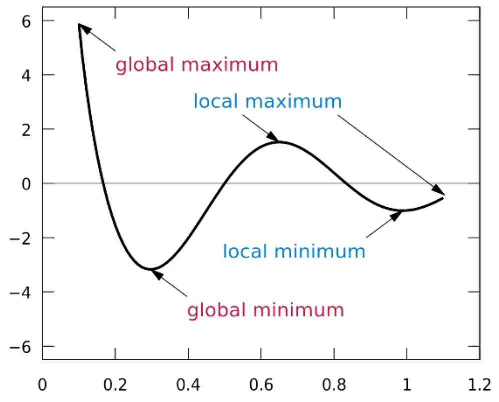


*Figure 5. Global and local minimum [16]*

It takes into account the step of the gradient and helps to accelerate the algorithm in the proper direction with damping oscillations. It does so by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma v_{t-1} + \eta\nabla_{\theta}L(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t \quad (6)$$

d) Adagrad

Adagrad adapts the learning rate for every parameter $\theta_i$ at every time step t, performing larger updates for infrequent and smaller updates for frequent parameters. It performs well on sparse data.

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,i} + \varepsilon}} \nabla_{\theta_t} L(\theta_{t,i}) \tag{7}$$

$G_{t,\,i}$ is a diagonal matrix where each diagonal element *i* is the sum of the squares of the gradients with respect to $\theta_i$ up to time step *t*. ε is usually equal to 1e-8 and ensures no case of division by zero. Because $G_{t,\,i}$ keeps growing during training, the learning rate shrinks to values effectively equal to zero.

e) Adadelta / RMSprop

RMSprop and Adadelta have both been developed independently around the same time and both try to resolve Adagrad's aggressive and monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to fixed-size ω. It stores the sum of gradients recursively defined as the decaying average of all past squared gradients. In this algorithm, the learning rate is eliminated from the learning rule.

f) Adam

Adaptive Moment Estimation (Adam) also computes adaptive learning rates for each parameter but besides storing an exponentially decaying average of past squared gradients, like Adadelta and RMSprop, it also keeps an exponentially decaying average of past gradients, similarly to momentum.

## 4.3. Basic machine learning paradigms

### 4.3.1. Supervised learning

In supervised learning, the algorithm uses samples in form of pair consisting of an input object and desired output value. The goal is to approximate the mapping function so well that when the new input object is introduced, the function outputs an adequate label. This process of iterative prediction and correction is similar to the concept of learning observed in animals or humans.

### 4.3.2. Unsupervised learning

Unsupervised learning or self-organization allows for the modeling of probability densities over unlabeled input data. It is often used in pattern detection. The lack of labels means little to zero human supervision which lowers the cost and potentially increases the size of a dataset. A variant called semi-supervised learning makes use of supervised and unsupervised techniques.

### 4.3.3. Reinforced learning

In reinforced learning, the focus is on finding a balance between exploration and exploitation of acquired knowledge of an intelligent agent. The goal is to take actions that maximize the reward based on observations and interpretation of the environment (Figure 6).

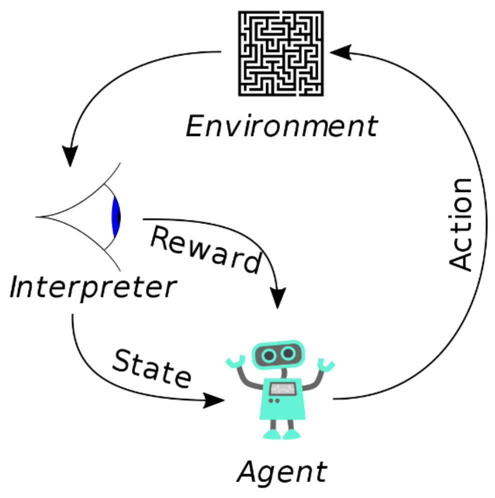
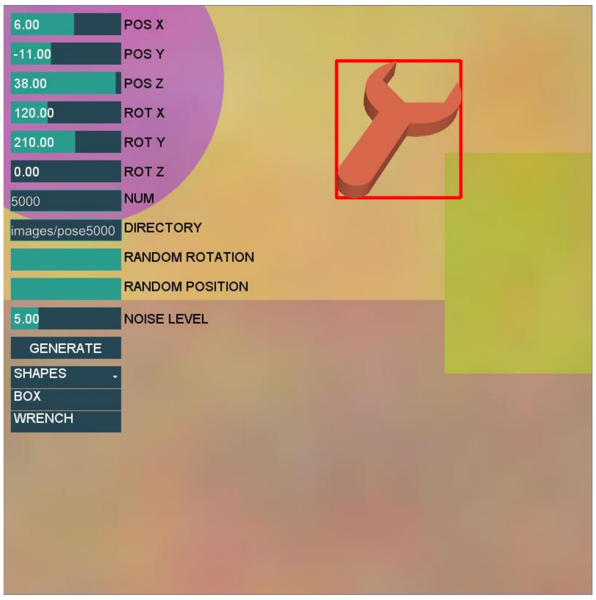


*Figure 6. Graphical interpretation of the reinforcement learning algorithm [15]*

## 5. Method

### 5.1. Dataset generation

Supervised learning requires large sets of labeled data. This is done by using a custom Java application that serves as an artificial environment for collecting annotated images of objects in form of 3D models. Imported models can be manually moved around the scene with sliders to verify results or debug. In the background, random simple shapes are generated to serve as a distraction for the neural network. Furthermore, 2D Perlin noise can be applied at various levels to increase randomization. The number of samples can be selected and the dataset will be automatically generated. Color images with the resolution of 800x800 pixels are being saved in a folder labeled with a unique number alongside a text file containing JSON of every image which information like position, rotation, and bounding box of the object. GUI of the application can be seen in Figure 7.



6.00 POS X
-11.00 POS Y
38.00 POS Z
120.00 ROT X
210.00 ROT Y
0.00 ROT Z
5000 NUM
images/pose5000 DIRECTORY
RANDOM ROTATION
RANDOM POSITION
5.00 NOISE LEVEL

*Figure 7. GUI of image dataset generator*

In this case, 5.000 samples proved to be enough to successfully train and evaluate the model. On every sample, there is always one object of interest, which is a 3D model of a wrench. In the background, the program generates ellipse and rectangle in various positions and sizes. Noise is applied for every component of RGB color. The position of the object is randomized in a way that ensures it is always in the field of view. Rotation changes only for one axis in the full range from 0 to 360 degrees. The object is manipulated in space with dimensions 70x70x40 units with 40 units of depth. The model has 12 units in length.

Generated images undergo global standardization which makes mean values and standard deviation of pixels in all color channels equal to 0 and 1 respectively [7]. This process ensures that input of the neural network is in the form of floating-point numbers in the range from -1 to 1 which is important as high values act negatively on the network's training.

## 5.2. Evaluation metrics

To have a reliable way of measuring performance and different aspects of the network, it is necessary to introduce evaluation metrics. These provide data regarding errors from the regression of position, rotation, and bounding boxes.

### 5.2.1. Position error

Error in estimating position in three-dimensional space is defined as the sum of the absolute difference between prediction and ground truth of each axis. Every metric is calculated as a mean from samples in the batch.

$$E_{pos} = \frac{1}{N}\sum_{i=0}^{N} |x_i - \hat{x}_i| + |y_i - \hat{y}_i| + |z_i - \hat{z}_i| \tag{1}$$

### 5.2.2. Rotation error

To evaluate rotation prediction, a formula based on acc15 was used. It is introduced in [5] as an equivalent of the accuracy metric in the classification network for the regression problem. It is defined as the ratio between a number of predictions with the absolute error being less than 15 degrees and the total number of predicted outputs.

$$E_{rot} = \frac{1}{N}\sum_{i=0}^{N} p_i, \tag{2}$$

$$where\ p_i = \begin{cases} 1, & |r_i - \hat{r}_i| < 15 \\ 0, & |r_i - \hat{r}_i| \geq 15 \end{cases} \tag{3}$$

### 5.2.3. Intersection over union

The inaccuracy of bounding box regression is represented by intersection over union (IoU). The definition for that metric is visualized in Figure 8 as a ratio between the overlapping area of ground truth frame with predicted frame and their area of union.

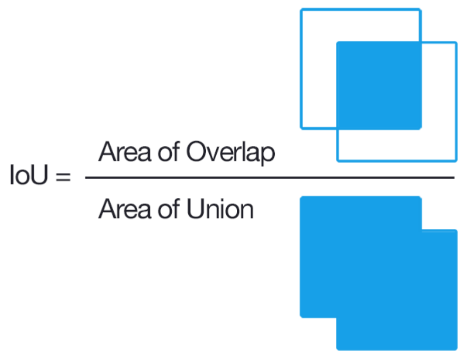


*Figure 8. The definition of IoU in computer vision*

## 5.3. Representing rotation as RGB color

Regression of rotation in degrees means that the neural network will for every given input of pixels, predict a scalar value representing the angle of rotation around the axis for a given object on the image. In this case, the last layer of that particular branch in the model will consist of one node that is expected to output a number in the range from 0 to 360 degrees. Using the backpropagation algorithm, parameters inside NN will be modified to minimize the loss function calculated from the error between the predicted value and given ground truth. The problem with this approach is caused by the lack of noticeable difference between objects rotated by for example 0 and 359 degrees. Prediction of numbers from the beginning of the range with ground truth located on the other end is very likely especially in the early stage of training. High error calculated from that prediction will cause improper and exaggerated modification of parameters which can lead to overfitting or generalization of rotation only in the middle of the spectrum.

In literature, rotation is calculated usually by implementing quaternions or Lie algebra [2] [3] [4]. For the problem of rotation in only one axis, a different method was proposed which is based on regressing angle to RGB color and calculating the angle of rotation from the hue (Figure 9).

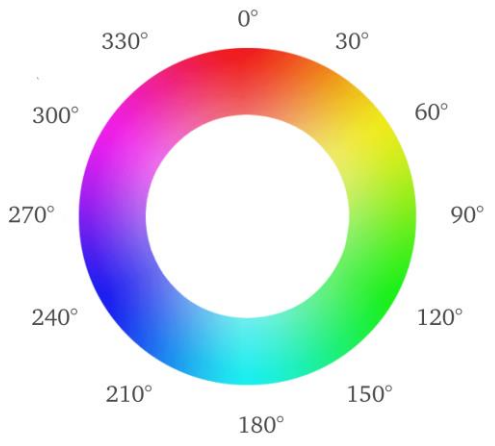


*Figure 9. Hue with corresponding angle values*

This eliminates the mentioned earlier edge case as now rotation is regressed into three scalars translating to values in the range from 0 to 255 (Figure 11). The sum of these numbers creates a continuous line that never makes rough transitions through the range of the spectrum (Figure 10).

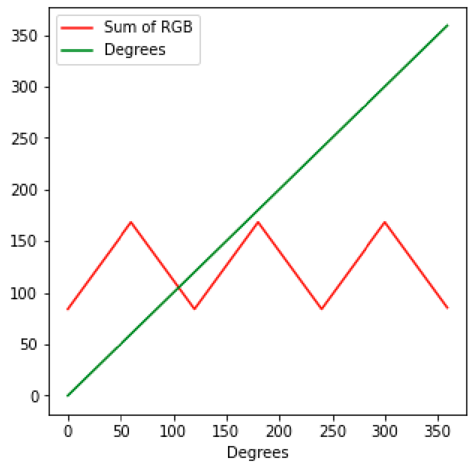


*Figure 10. Relation between sum of RGB components and angle*

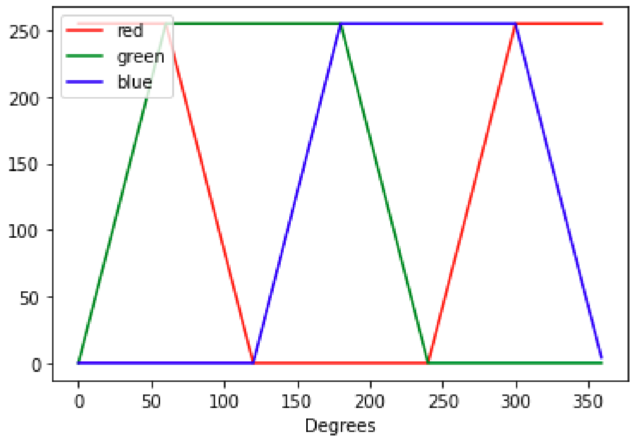


*Figure 11. Values of RGB components*

For given components RGB, the angle can be calculated using the following formula:

$$r = atan2\left(\sqrt{3} \cdot (G - B), 2 \cdot R - G - B\right) \tag{4}$$

To calculate hue from the angle, a simplified algorithm was used that divides a circle into six sections. For each part, there is one component with a value equal to (for this purpose) 0, 255, and one in between. This is how the python implementation looks like:

```
def AngleToRGB(angle):
    X  = 1 - abs(angle/60%2-1)
    if(0 <= angle < 60):
        R = 1
        G = X
        B = 0
    elif (60 <= angle < 120):
        R = X
        G = 1
        B = 0
    elif (120 <= angle < 180):
        R = 0
        G = 1
        B = X
    elif (180 <= angle < 240):
        R = 0
        G = X
        B = 1
    elif (240 <= angle < 300):
        R = X
        G = 0
        B = 1
    elif (300 <= angle < 360):
        R = 1
        G = 0
        B = X
    return [R*255, G*255, B*255]
```

This approach proves to be successful and superior in the context of describing angle as a number which can be seen in Table 1.

### 5.4. Implementation details

TensorFlow library served as a framework for building neural network model [8]. It offers multiple levels of abstraction and ways of building, training, and validating. One of them is Keras with its Functional API that can handle models with non-linear topology, shared layers, and multiple outputs which is necessary for purpose of this project [9]. On top of that, it provides detailed and comprehensive documentation. The library was used in Python 3.8.

## 6. Model architecture

### 6.1. Feature extraction

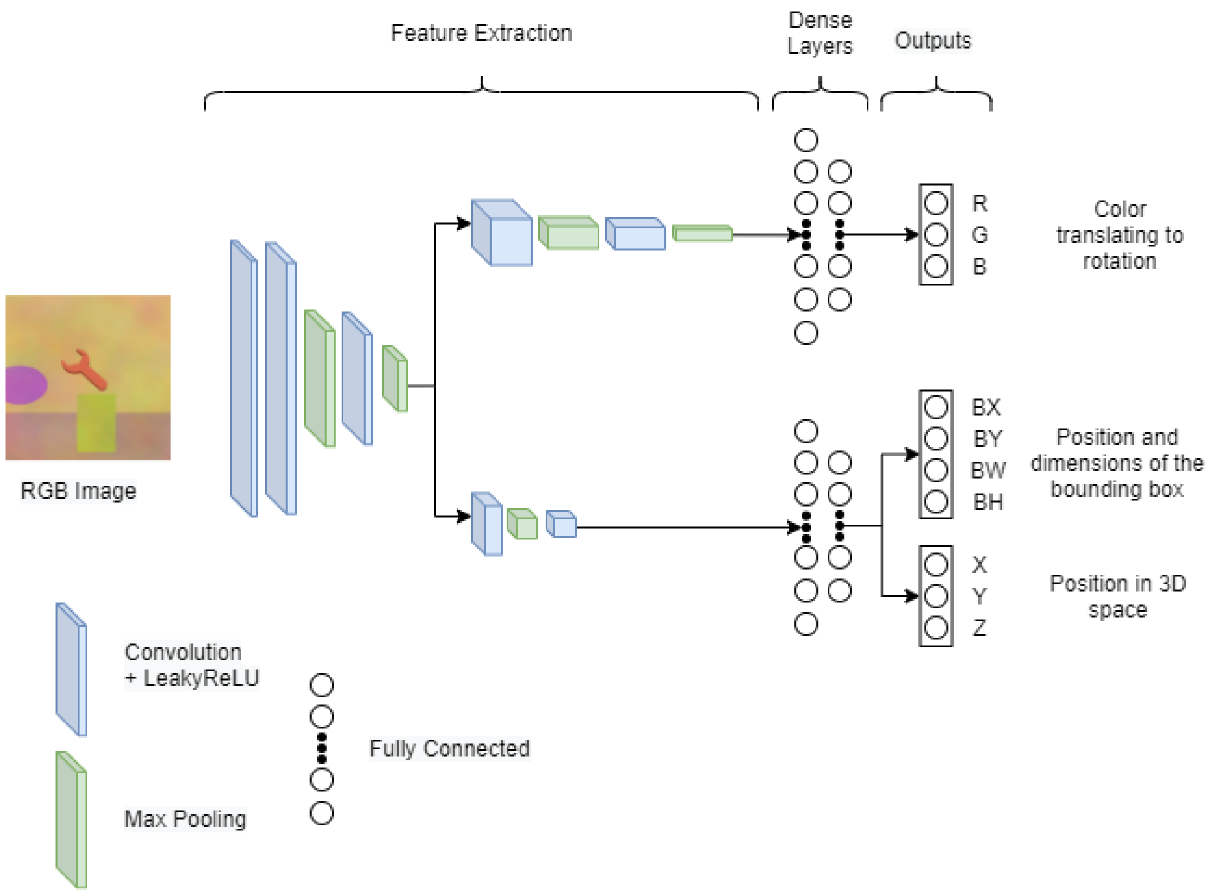


*Figure 12. Architecture diagram*

Figure 12 illustrates the architecture of the network. On the input, there is an RGB image with a resolution of 150x150 pixels. The next part consists of 3 convolutional layers and 2 max-pooling layers performing initial feature extraction.

The goal of this process is to enhance different details of the object on the picture and create a separate matrix of pixels for each. It is done by using a convolution matrix more commonly referred to as the kernel. In convolution layer network calculates a dot product of kernel and pixels from every area of the image. Convolution matrix performs a sliding motion where it moves step by step from side to side of the pixel array and maps generated outputs to the corresponding place. It provides an opportunity to detect and recognize features regardless of their position in the image. A step is called a stride and is usually equal to one pixel but this can be changed to achieve different outputs resolutions. In layer, there is usually more than one kernel with different values of matrix producing filters extracting various features [1]. This is often interconnected with max-pooling layers which generate output by calculating maximum value within pixels in specified areas of every filter. This helps to lower the resolution while maintaining sparse data and its location which lowers the number of parameters in the network. It also improves generalization in dense layers. Example output on various stages of feature

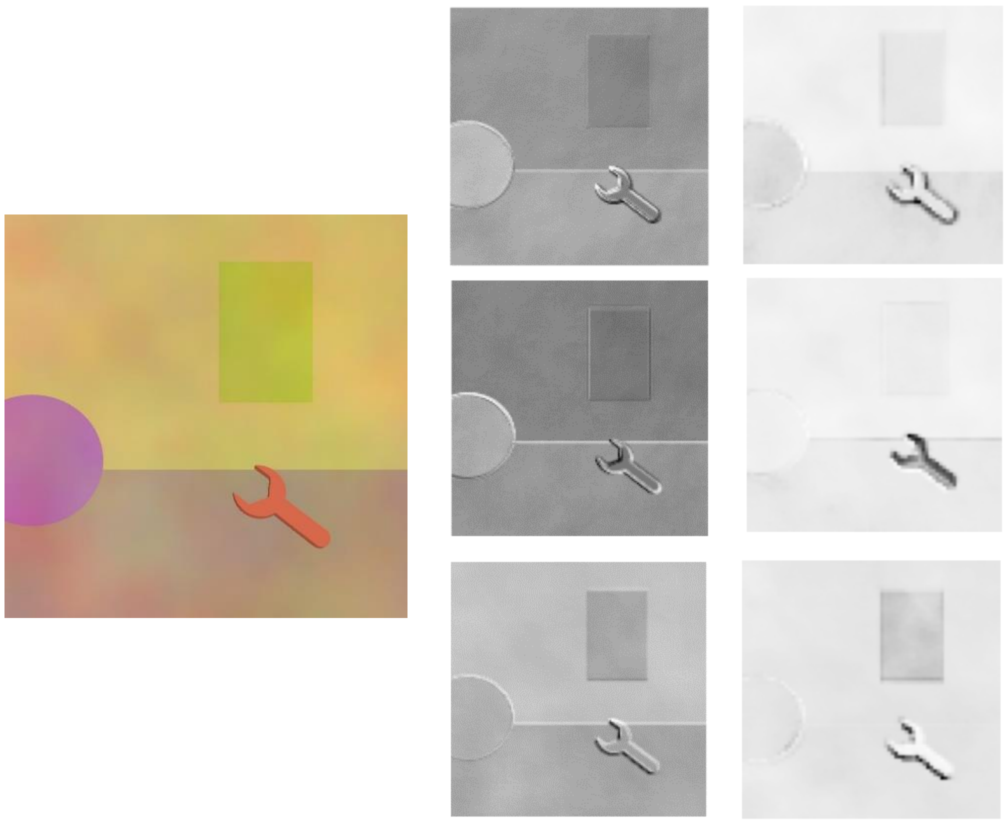


*Figure 13. Feature extraction at various stages*

extraction can be seen in Figure 13. It is worth mentioning how different edges are extracted by filters and how objects get highlighted in the background. Also, the resolution of output gets lower each time. This part is the backbone of the network since the extracted features are used by network branches.

### 6.2. Branches

The top branch consists of 2 convolutional layers, 2 max-pooling layers, and 2 fully connected (dense) layers and outputs 3 values describing RGB color which can be translated into the angle as mentioned earlier. The bottom branch has 2 conv. layers, one max-pooling layer, and two dense layers. The output is divided into two sections that describe the location of the top left corner of the bounding box on the image and its width and height. The other outputs position of the object in space relative to the camera.

### 6.3. Activation functions

A model usually has two types of activation functions; applied in hidden layers and output layers. Because values for each pixel after global standardization are in the range from -1 to 1, widely used ReLU would cause loss of data in convolutional layers and dead weights as was observed during development. For this reason, a modified ReLU function called LeakyReLU was used defined as:

$$f(x) = \begin{cases} x, & x > 0 \\ 0.01x, & x \leq 0 \end{cases} \tag{5}$$

### 6.4. Loss function

The total loss function can be described as the sum of losses across all outputs.

$$L = L_{rot} + L_{pos} + L_{frame} \tag{6}$$

For every individual output, the loss is the mean absolute error (MSE), defined as:

$$MSE = \frac{1}{N}\sum_{i=0}^{N}(y_i - \hat{y}_i)^2 \tag{7}$$

Where $y_i$ is the ground truth label and $\hat{y}_i$ its predicted value.

### 6.5. Training process

Here, 90% of the dataset was used for training, and the remaining 10% for validation. The online machine learning method adopted in the process means that data was available in sequential order and the network's parameters were updated after each step. Each batch consisted of 32 samples as it is generally recommended for this type of task.

#### 6.5.1. Optimizers comparison

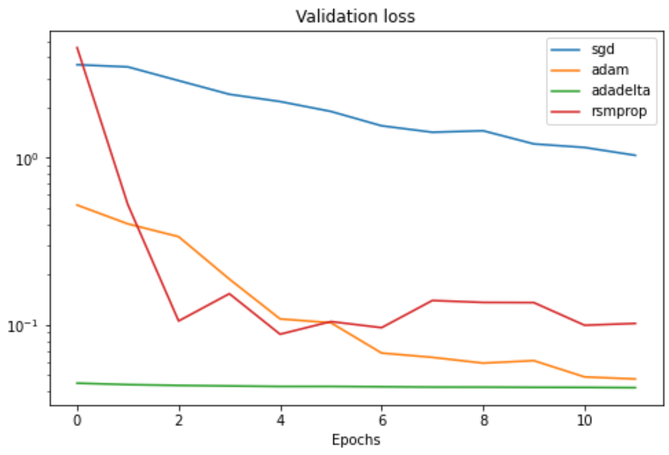


*Figure 14. Validation loss of chosen optimizers*

Few popular optimizers were chosen and compared in terms of performance (Figure 14). Adadelta can be observed to perform remarkably well on this type of task. This is confirmation of results achieved by researchers in [5] who also chose this algorithm after comparing it with others in a similar task. It significantly outperforms the initially proposed Adam even after the first epoch.

### 6.5.2. Overfitting

Overfitting can occur during training in which the model instead of generalizing data, learns samples from the training dataset. This renders it unable to correctly perform prediction on different, not previously seen data. To avoid it, the noise was introduced in the background of the image to make samples more distinct from each other. Also, there is a relatively low number of nodes in dense layers which helps to prevent "memorizing" data. The training was stopped when no improvement was being made. Graph representing loss shows that function for training and validation has different magnitude but noticeably similar shape (Figure 15). This indicated that overfitting was successfully avoided.

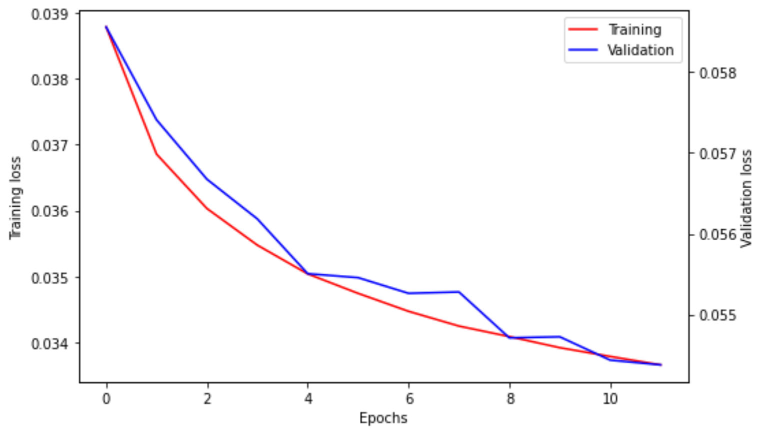


*Figure 15. Loss values for Adadelta*

## 7. Results

### 7.1. Comparison of two rotation angle notations

Results were obtained for both rotation angle represented by a number in the range from 0 to 360 and by RGB color with validation dataset. In both cases, the model was optimized using the Adadelta algorithm. Table 1 shows the comparison of the ACC accuracies. It is clear that using degrees for rotation regression is not applicable for this task with an average rotation error equal to 24.5 degrees. Almost half of the predictions are wrong by more than 20 degrees which can be caused by the mentioned earlier problem with training. Only 14.25 percent of predictions, or one in every seven, have less than 5 degrees of error. The RGB method shows acceptable results with more than 90% of predictions having less than 15 degrees of error. More than half of prediction have error lower than 5 degrees.

*Table 1. Comparison of accuracies*

| | Acc 20 [%] | Acc 15 [%] | Acc 10 [%] | Acc 5 [%] | Avg [°] |
|---|---|---|---|---|---|
| Degrees | 56.01 | 45.54 | 29.17 | 14.25 | 24.50 |
| RGB | 95.98 | 93.24 | 85.58 | 56.56 | 5.12 |

### 7.2. Rotation

Graph representing probability error with given value can be seen in Figure 16. It shows that the network performs generally well and often predicts rotation without an error. On average there is 5.12 degrees of difference between estimation and given ground truth.

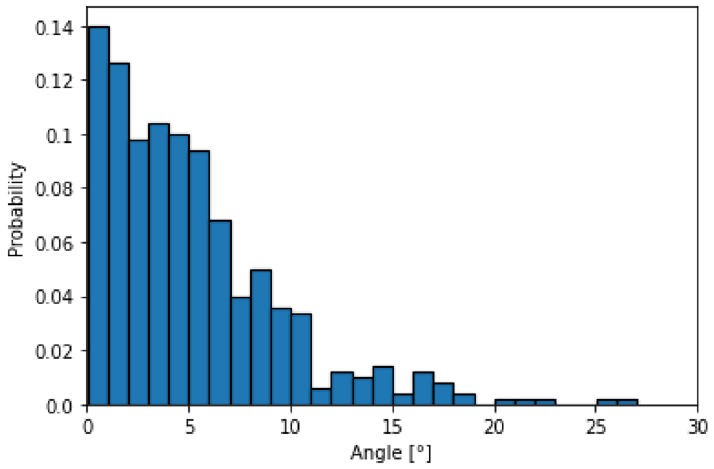


*Figure 16. Probability graph of rotation error*

The average error can be visualized by taking the object's model as portrayed by the network and overlaying it with a prediction. Displacement error is omitted for purpose of clarity (Figure 17).

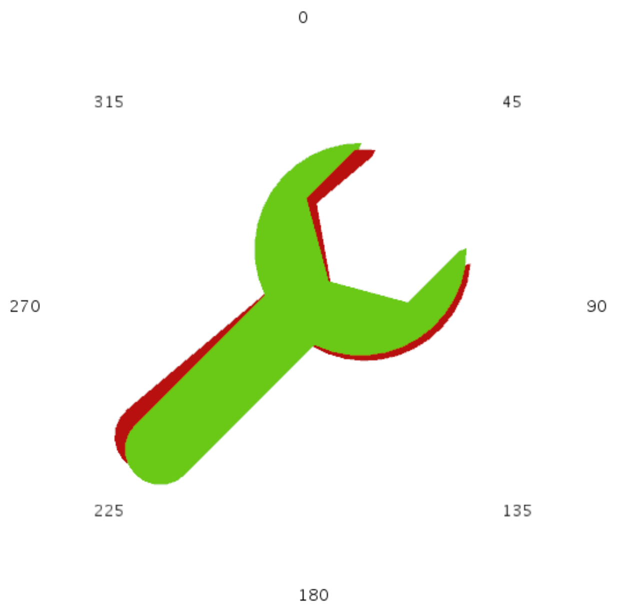


*Figure 17. Ground truth (green) with average rotation prediction error (red)*

## 7.3. Position

The mean error in estimating position for each axis is represented in Figure 18. It is worth noticing that the value of 1 unit equals only 8.3% of the length of the object.

It is clear that the network shows worse results for predicting distance from the camera. Figure 19 represents the relation between the ground truth of distance and estimation error. As expected, on average smaller value occurs for closer objects. This proposes a solution in which the prediction is taking place every time the camera gets closer to further correct the error. Visualization of average error can be seen in Figure 20.

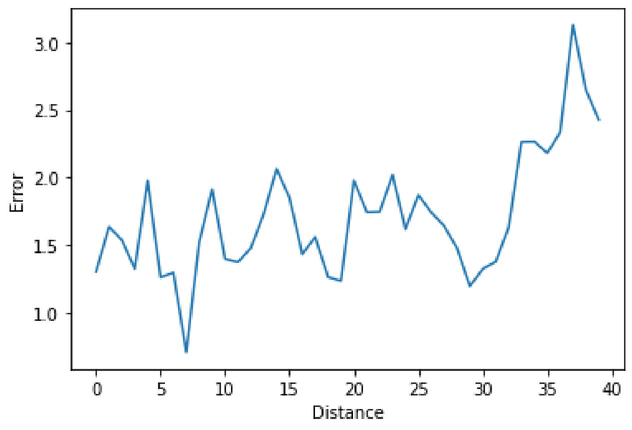


*Figure 19. Error in position estimation on Z axis relative to distance*

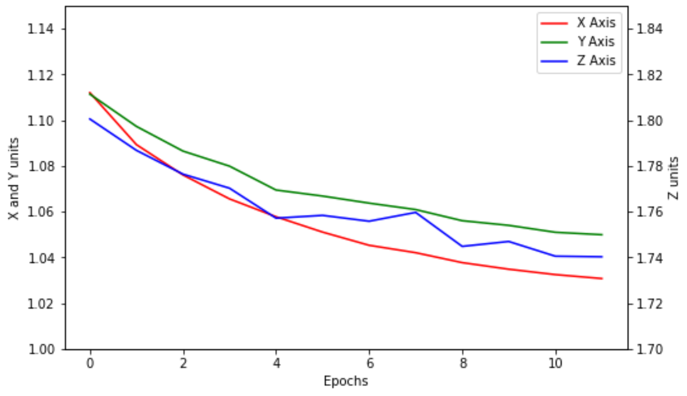


*Figure 18. Average error per axis*

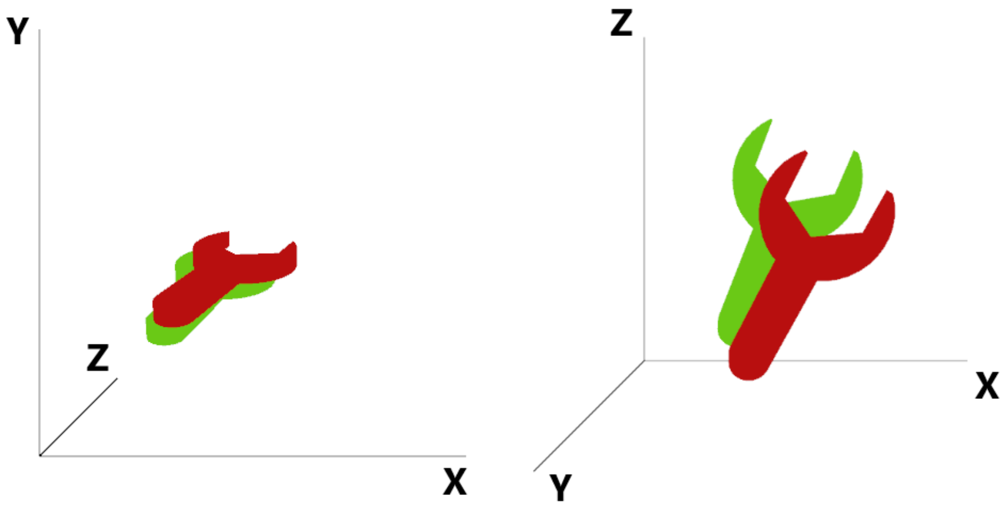


*Figure 20. Ground truth (green) with average position prediction (red)*

## 7.4. Bounding box

For branch predicting bounding box of the object, the IoU is on average equal to 0.69 with a value higher than 0.5 considered as “good” by various sources like for example article on pyimagesearch.com [11] (Figure 21). The probability graph resembles the Gaussian distribution (Figure 22).

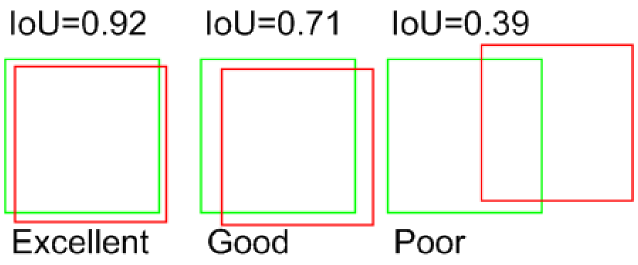


*Figure 21. IoU values comparison [14]*

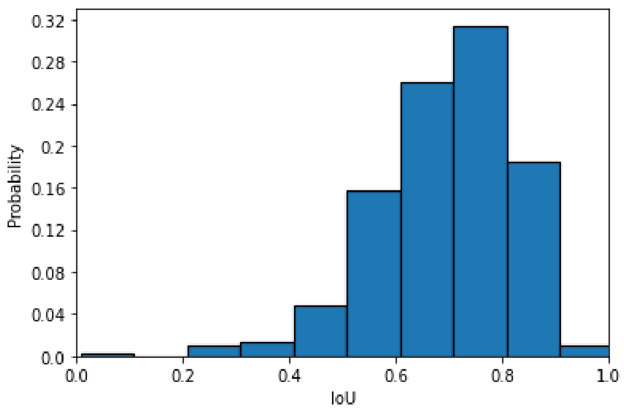


*Figure 22. Probability graph of IoU value*

# 8. Conclusion

In this engineering thesis, a convolutional neural network model was designed and evaluated. Its task is to estimate a rotation of a 3D model in space based on a monocular color image. Additionally, the model predicts a bounding box for the object on the picture. It was built using the new open-source library TensorFlow Keras in Python 3.8. The model was trained with supervised learning with data generated in a custom tool build in Java. Two methods were compared for describing rotation for regression purposes. One representing a naïve approach by using numbers in the range from 0 to 360 degrees. The other was based on a hue circle expressing saturation of color with RGB components. The comparison showed that the latter is indeed a much better solution with the network showing good evaluation results for estimating the angle of rotation in one axis. Few different optimizers were tested to achieve optimal performance. It was clear that the Adadelta algorithm is the most appropriate for this kind of task. Different metrics for position showed that the NN scores on average worse for estimating the distance from the camera than position on a plane perpendicular to the camera lens. IoU calculated for the bounding box proved that estimation is sufficient with the average value being described as good by various sources. This technique can be used for example in sorting centers or factories by pick-and-place robots to detect and position specified objects.

## References

[1] S. ALBAWI, T. A. MOHAMMED and S. AL-ZAWI, "Understanding of a Convolutional Neural Network," *IEEE,* 2017.

[2] Y. Xiang, V. Narayanan, D. Fox and T. Schmidt, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," *arXiv:1711.00199.*

[3] T.-T. Do, M. Cai, T. Pham and I. Reid, "Deep-6DPose: Recovering 6D object pose from a single RGB image," *arXiv:1802.10367,* 2018.

[4] M. Rad and V. Lepetit, "BB8: A scalable, accurate, robust to partial occlusion method for predicting the 3D poses of challenging objects without using depth," *arXiv:1703.10896,* 2018.

[5] L. A. Giefer, J. . D. A. Castellanos, M. M. Babr and M. Freitag, "Deep learning-based pose estimation of apples for inspection in logistic centers using single-perspective imaging," *MDPI,* 2019.

[6] H. Kinsley and D. Kukieła, Neural Network from Scratch in Python, 2020.

[7] S. Ruder, "An overview of gradient descent optimization algorithms," 2017.

[8] "machinelearningmastery.com/how-to-manually-scale-image-pixel-data-for-deep-learning," [Online]. [Accessed 26 12 2020].

[9] "tensorflow.org/about," [Online]. [Accessed 26 12 2020].

[10] "keras.io/guides/functional_api/," [Online]. [Accessed 26 12 2020].

[11] "pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection," [Online]. [Accessed 29 12 2020].

[12] M. Zeiler, "ADADELTA: An adaptive learning rate method," no. arXiv:1212.5701, 2012.

[13] D. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," no. arXiv:1412.6980, 2017.

[14] "http://www.interstellarengine.com/ai/Intersection-Over-Union.html," [Online]. [Accessed 17 01 2021].

[15] "en.wikipedia.org/wiki/Reinforcement_learning#/media/File:Reinforcement_learning_diagram.svg," [Online]. [Accessed 17 01 2021].

[16] "en.wikipedia.org/wiki/Backpropagation#/media/File:Extrema_example.svg," [Online]. [Accessed 17 01 2021].

[17] "en.wikipedia.org/wiki/File:Error_surface_of_a_linear_neuron_with_two_input_weights.png," [Online]. [Accessed 17 01 2021].