RPN-BASED ARCHITECTURE FOR OBJECT DETECTION AND POSE ESTIMATION USING RGB-D DATA

A Thesis
Presented to
The Academic Faculty

By

Rémi Gourdon

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the School of Electrical and Computer Engineering

Georgia Institute of Technology

December 2018

RPN-BASED ARCHITECTURE FOR OBJECT DETECTION AND POSE ESTIMATION USING RGB-D DATA

Approved by:

Dr. Thomas R. Collins, Advisor
School of Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Zsolt Kira
School of Interactive Computing
Georgia Institute of Technology

Dr. Patricio A. Vela
School of Electrical and Computer Engineering
Georgia Institute of Technology

Date Approved: December 6, 2018

ACKNOWLEDGEMENTS

First and foremost, I cannot find words to express my gratitude to Benjamin Joffe for picking me as his research assistant on his deep learning project, for providing me with a setting in which to develop my understanding of the field and for offering me continuous support and valuable insights from my first day on the job down to the writing of this thesis.

I wish to thank Thomas Collins for accepting to be my thesis advisor, for informing the direction of my research and for providing valuable inputs that informed my writing. I am also thankful to him for constituting a reading committee with Zsolt Kira and Patricio Vela, to whom I extend my gratitude.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ADD Average Distance. 26

ANN Artificial Neural Network. 8

API Application Programming Interface. 28, 29

ATAS Aerospace, Transportation and Advanced Systems Laboratory. 1

CAD Computer-Aided Design. 13, 14, 16, 26, 36, 37

CNN Convolutional Neural Network. 8, 9, 16–18

DoF Degrees of Freedom. viii, 3, 5, 8, 17

FPTD Food Processing Technology Division. 1, 27, 30, 37, 40

GTRI Georgia Tech Research Institute. 1

ICP Iterative Closest Point. 11–14, 40

MLP Multi-Layer Perceptron. 8

ReLU Rectified Linear Unit. 9, 22

RoI Region of Interest. 9, 10, 27, 28

ROS Robot Operating System. 31–34

RPN Region Proposal Network. viii, 1, 9, 10, 20, 27, 28, 30, 40

YCB Yale-CMU-Berkeley Dataset. 26, 34

SUMMARY

A 6 Degrees of Freedom (DoF) pose estimation network was developed with the long-term objective to integrate it in an end-to-end solution based on RPNs. The different network versions were implemented using the open-source TensorFlow framework. The integration of the pose estimation head on a RPN trunk was attempted by modifying a TensorFlow implementation of Faster R-CNN, and other options for the end-to-end solution were proposed.

The development of the pose estimation network was then conducted separately and considering different variations of the architecture, including orientation and position estimation using a single or two separate regression heads. This development was inspired by the extensive research in pose estimation from which insights were taken and combined into a fully connected network capable of leveraging depth information for 6D pose estimation.

An automated data collection process was designed to facilitate the recording of pose labels, a time consuming process when handled manually. This acquisition method enabled the collection of an application specific and novel chicken dataset using an industrial robotic arm and multiple commercial off-the-shelf cameras. This data was used to evaluate the pose estimation network and served as an initial confirmation of its applicability in industrial poultry processing.

CHAPTER 1
INTRODUCTION

The pose describes the position and orientation of an object in a particular frame. It is a useful information in a wide range of applications. As such, the problem of pose estimation is today a topic of interest in computer vision research. This work focuses on extending well-known deep learning architectures based on RPNs to provide an estimation of an object's pose in the scene.

Fast-growing robotics applications including autonomous vehicles and robotic manipulators rely on this data to build an understanding of the environment. A preliminary step generally involves the detection of the surrounding objects but this alone is often insufficient to make an informed decision.

Indeed, a robot car needs not only to know whether or not pedestrians or other vehicles are nearby, but also what lane they are positioned in and what is their direction of motion, which can as a first approximation be inferred from their orientation. A robotic arm performing a grasping operation and equipped with accurate information about the pose of the target object and obstacles is able to achieve efficient motion planning.

1.1 Motivation

Although this work takes a path toward a general approach to the pose estimation problem, it is supported by the Food Processing Technology Division (FPTD) of the Aerospace, Transportation and Advanced Systems Laboratory (ATAS) at the Georgia Tech Research Institute (GTRI). For this reason, the research described in the rest of this thesis is primarily aimed at poultry processing applications and specifically at carcass grasping scenarios with robotic arms.

Industrial applications allow the deployment of the robotic systems in environments

where external factors such as lighting or background can be controlled, simplifying the vision task. However, the processing of poultry products introduces additional constraints related to the varying shapes of the carcasses and their deformable nature. While those constraints have a particular impact on the grasping strategy, they also need to be taken into account in the design of the robot visual perception.

This motivates the exploration of options toward a generalized solution which will nonetheless be applicable in poultry processing, as part of a larger robotic solution including perception, control and grasping. The vision system builds upon the advances in the field of deep learning for computer vision, using commercial off-the-shelf hardware coupled with advanced algorithms.

1.2 Outline

Chapter 2 briefly introduces the concepts this work builds upon before Chapter 3 presents other approaches to the pose estimation problem found in literature. Chapter 4 details the core of the work conducted toward the writing of this thesis and Chapter 5 describes the evaluation of the models developed during this project. Finally, Chapter 6 concludes this thesis and presents possible follow up to this work to further refine the results and move closer to a fully integrated end-to-end solution.

CHAPTER 2

FUNDAMENTALS

2.1 3D Pose Estimation Problem

The problem studied in this work consists in estimating the pose of an object in the 3-dimensional space using 2-dimensional color images, eventually supplemented by depth information. The pose of an object is defined with respect to a reference frame, and it has 2 components that fully constrain its 6-DoF.

The position component is represented using a translation vector $\boldsymbol{t} = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}^T$ in $\mathbb{R}^3$. The orientation component is represented by a rotation belonging to the 3D rotation group $SO(3)$. A rotation matrix $\boldsymbol{R}$ is used for this explanation, but the orientation can be stored in multiple forms, as described in Section 2.2.

Although orientation and rotation have slightly different meanings, the former describing a state and the latter a transformation between 2 states, both might be used interchangeably throughout this work, since the rotation could be assumed to be relative to a specific origin. The same reasoning applies to position and translation.

Following computer vision conventions, the camera coordinate frame C is defined with the origin at the camera optical center and the z-axis oriented along the camera optical axis. The reference frame is considered to be the local object frame O. The estimated pose is thus the transformation (translation and rotation) that brings the object from its local frame to the camera frame. The resulting frame, once rotated, translated and projected in 2D using the camera intrinsic matrix, can be visualized as seen in Figure 2.1.

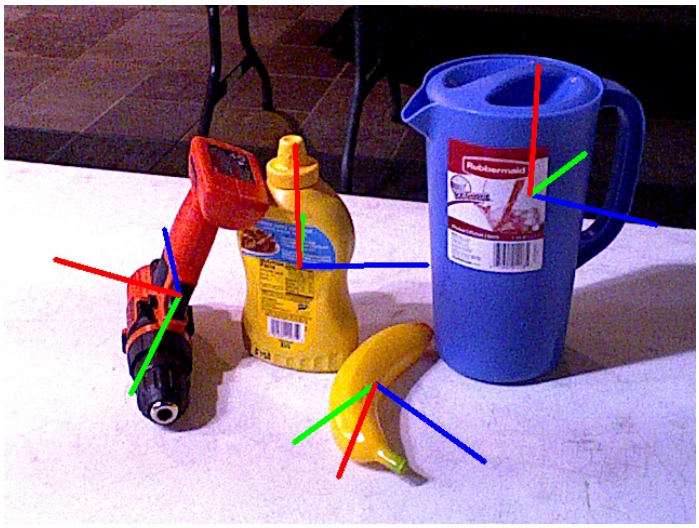

Figure 2.1: Illustration of the local object frames projected in the 2D image space

Let $\boldsymbol{p} = [x\ y\ z]^T$ represent a point, $\boldsymbol{P} = [p_1 \cdots p_m]$ represents a set of points. $\boldsymbol{P}_C$ and $\boldsymbol{P}_O$ represent respectively a set of points in camera frame and a set of points in object frame. The objective is to solve for $\boldsymbol{t}$ and $\boldsymbol{R}$ to satisfy Equation (2.1).

$$\boldsymbol{P}_C = \boldsymbol{R}\boldsymbol{P}_O + \boldsymbol{t} \tag{2.1}$$

2.2 Orientation Representation

The orientation information in the 3-dimensional space can be represented under different forms, with a minimal number of 3 parameters, one for each degree of freedom. Considering that the object's position and orientation correspond to the translation and rotation required to bring the object from its local frame to the camera frame, the orientation is effectively encoded as a rotation.

4 main methods are commonly used to describe 3D rotations: the rotation matrix, the Euler angles, the axis-angle and the quaternion [1]. The choice of one method over the others depends on the application.

2.2.1 Rotation Matrix

The rotation matrix represents the orientation as a 3×3 orthogonal matrix $\boldsymbol{R}$. The opposite rotation that brings the object from camera space to local space is thus $\boldsymbol{R}^{-1} = \boldsymbol{R}^T$. More than a simple representation of the orientation, the set of all 3×3 orthogonal matrices is a direct mapping of the $SO(3)$ 3-dimensional rotation group.

This representation is useful for visualization purposes, because it can directly rotate the points that form an object's model to camera space. However, rotation matrices contain redundant information, with 9 values to parameterize 3-DoF. It thus requires that 6 implicit constraints be enforced to describe a valid 3D orthonormal basis. As such, not all 3×3 matrices are valid rotation matrices, and this can lead to difficulties for estimating the orientation component of the pose by regression, because enforcing those constraints in the network is difficult.

2.2.2 Euler Angles

Euler angles represent the orientation by a succession of 3 rotations, with each being applied on the object's original axes, which change following each rotation (intrinsic rotations). Because any sequence of rotations is valid, the Euler angles representation relies heavily on conventions. One of the most common convention comes from the aerospace field and is referred to as yaw-pitch-roll and is illustrated in Figure 2.2.

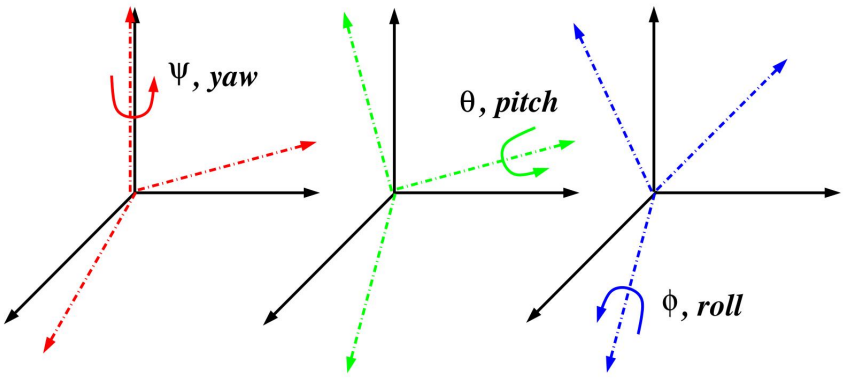

Figure 2.2: Illustration of the yaw-pitch-roll convention for Euler angles [2]

As long as conventions are established correctly, Euler angles are intuitive to use and do not suffer from over-parameterization. However, while any set of 3 Euler angles form a valid rotation, angles are not uniquely described unless the angle values are bounded, typically to $\pm 180°$ for the yaw and roll and $\pm 90°$ for the pitch. Furthermore the Euler angle representation suffers from a problem known as gimbal lock that arises when 2 of the rotation axes align.

2.2.3 Axis-angle

The axis-angle representation encodes the orientation as the combination of a rotation axis defined in the local object space, and a rotation value. The Euler rotation theorem proves proves that it is possible to encode any rotation in this form, which although not as approachable as Euler angles, can still be understood easily.

If the axis of rotation is expressed as its codirectional unit vector $\boldsymbol{n}$, and if θ is the rotation angle, then the axis-angle can be represented as the vector $\boldsymbol{e} = \theta\hat{\boldsymbol{n}}$. The axis-angle can thus be stored in a compact form with 3 values, and the angle of rotation can be recovered easily as $\theta = \|\boldsymbol{e}\|$.

The unit vector $\hat{\boldsymbol{n}}$ and rotation angle θ could be regressed separately in the network. The unit norm constraint is easily enforced using an L2 normalization on the vector component, but the rotation angle, similarly to Euler angles, wraps around $[0, 2\pi[$.

2.2.4 Quaternion

Quaternions can be seen as an extension of the complex numbers in the 4-dimensional space with the following properties on the units: $i^2 = j^2 = k^2 = i \cdot j \cdot k = -1$. The normalized quaternion represents an orientation as a combination of a vector $\boldsymbol{v} = \begin{bmatrix} q_x & q_y & q_z \end{bmatrix}$ and a scalar q_w. This thesis will only make use of normalized quaternions, also called versors, described in Equation (2.2).

$$\boldsymbol{q} = q_w + q_x\boldsymbol{i} + q_y\boldsymbol{j} + q_z\boldsymbol{k} = \begin{bmatrix} q_w & \boldsymbol{v} \end{bmatrix}, \qquad \|q\| = \sqrt{q_w^2 + \|\boldsymbol{v}\|^2} = 1 \tag{2.2}$$

Although quaternions are more difficult to understand than Euler angles or the axis-angle representation, and require an extra real value to store, the unit norm is the only constraint that the network needs to enforce in order to ensure that the estimation will always be a valid rotation. Unit quaternions can also be easily converted to rotation matrices as shown in Equation (2.3).

$$\boldsymbol{R} = \begin{bmatrix} 1 - 2(q_y^2 + q_z^2) & 2(q_x q_y - q_z q_w) & 2(q_x q_z + q_y q_w) \\ 2(q_x q_y + q_z q_w) & 1 - 2(q_x^2 + q_z^2) & 2(q_y q_z - q_x q_w) \\ 2(q_x q_z - q_y q_w) & 2(q_y q_z + q_x q_w) & 1 - 2(q_x^2 + q_y^2) \end{bmatrix} \tag{2.3}$$

Moreover, quaternions have a property called double-cover on the rotation group $SO(3)$ which is not a useful feature for the pose estimation problem but make them appealing in other fields. It means however that any single orientation can be represented by two distinct quaternions, $\boldsymbol{q}$ and $-\boldsymbol{q}$ with $-\boldsymbol{q} = -\begin{bmatrix} q_w & \boldsymbol{v} \end{bmatrix} = \begin{bmatrix} -q_w & -\boldsymbol{v} \end{bmatrix}$. Luckily, this problem can be easily solved by negating any quaternion that displays a negative real part, ensuring that only "positive" quaternions are considered.

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) belong to the larger family of Artificial Neural Networks (ANNs), a set of algorithms that rely on the concept of artificial neuron, whose functioning is loosely inspired by biological neural networks. They have been the topic of increasing interest in computer vision applications in the recent years, and are used in this thesis to perform feature extraction.

The seminal paper on CNNs [3] in the late 90s presented them as a new kind of Multi-

Layer Perceptrons (MLPs). Both CNNs and MLPs are examples of feedforward networks, composed of an input layer, multiple internal layers called hidden layers, and an output layer, through which the data is fed during the forward pass. CNNs were developed to deal with 2-dimensional data, of which images are a typical example. Compared to fully connected MLPs, they have a reduced number of trainable parameters which correspond to the weights of a set of filters. Those filters exploit the spatial correlation of the image information to reuse the same weights across all the image space, with each feature map sharing the same filter weights. Beyond the reduction of the number of DoF in the network, this has the effect of introducing robustness toward scale and shift in the image, a characteristic that is not provided by dense layers that do not preserve the spatial information.

A layer in the CNN takes in the output of the preceding layer and convolves it with each of the learnable filters, effectively transforming an input volume (input image or feature maps from the previous convolutional layer) and transforms it in an output volume, as illustrated in Figure 2.3. The convolution is a common mathematical operation used in image processing that consists in a succession of multiplications between a small matrix (filter) that slides across the 2-dimensional space, and the local image patches. Each sliding filter of a layer forms a feature map, with each unit of the filter connected to a corresponding local unit in the preceding layer. The 3 main characteristics of a convolutional layer are the number of its filters (also called depth), the size of the filters receptive field (the area they are connected to in the previous layer) and the stride of the filters.

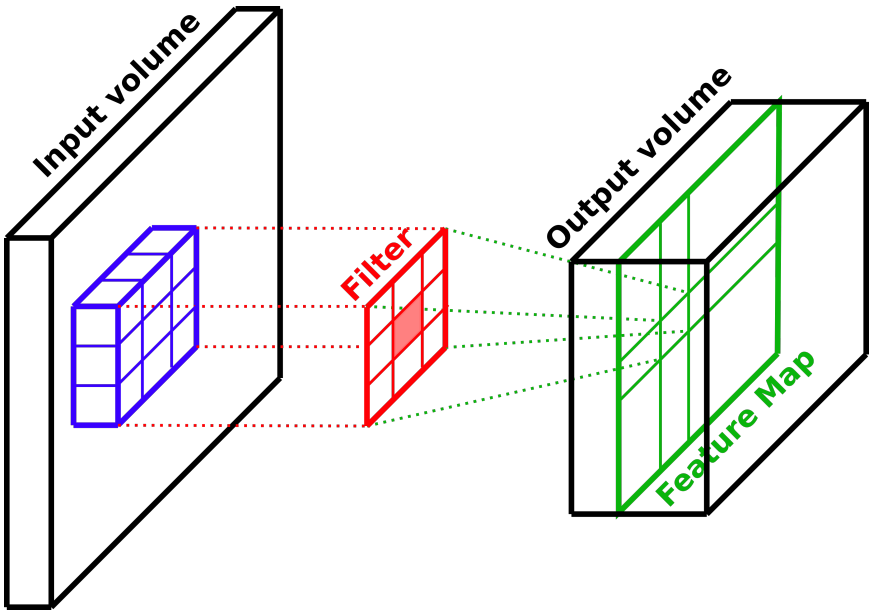

Figure 2.3: Illustration of a convolutional layer

A convolutional layer is followed by a non-linear activation function and eventually a pooling layer used to reduce the size of the feature maps by applying spatial averaging, in effect contributing to regularization. Modern architectures such as AlexNet [4], VGG [5], ResNet [6] and Inception [7] use Rectified Linear Units (ReLUs) [8] as the activation function on all convolutional layers and max-pooling layers [9]. CNNs are typically trained via backpropagation with gradient based optimizers and the batch gradient descent method.

2.4 Region Proposal Networks

RPNs were introduced as an addition to the Fast Region-based CNN (Fast R-CNN) [10], an object detection network that takes in the feature maps from a deep CNN and a set of Region of Interest (RoI) inputs. The Fast R-CNN detector is used to predict classes while in parallel the RPN predicts the location and shape of multiple boxes across the image and provides an estimate of the probability that each box contains or does not contain an object.

These predictions are made by 2 separate fully connected layers, which take as an input a small sliding region of the feature map at the last convolutional layer, as well as a set of

anchor boxes built with different scales and aspect ratios around a central anchor point. This concept coined as "pyramid of anchors" enables predictions for different shapes of boxes but using a single scale sliding window on the feature map [11]. The feature extractor and the RPN constitute the trunk of the network that provide RoI features to the different heads of the network, in this case a bounding box regression head and a classification head.

This approach has been proven successful and has been extended to create Mask R-CNN [12], which adds a parallel head to the network for semantic segmentation (pixel-wise classification) and an evolution of the RoI pooling layer that provides a better alignment of the pooled features with the input image. The head predicts a mask for each class and relies on the classification head introduced in Fast R-CNN to have a single mask contribute to the loss function.

CHAPTER 3
RELATED WORK

Because of the wide range of its applications, pose estimation has been an active topic for research in computer vision. Solutions available on the market have mostly relied on traditional methods such as point cloud registration or feature-based matching. More recently, the democratization of deep learning techniques has opened the doors to new ways to overcome the challenges of those methods.

3.1 Deep Learning Approaches

The research on deep learning methods for pose estimation is very active and extensive. This thesis work and thus this review focus on papers presenting some characteristics of interest:

- color (RGB) or color and depth (RGB-D) data inputs which are available in commercial off-the-shelf sensors ;
- pixel-based approaches rather than voxel-based in order to build upon the extensive list of available object detection and segmentation networks.

The approach taken in [13] is directly applied to robotics and integrates both the segmentation step using SegNet [14] and the pose estimation using Iterative Closest Point (ICP) and Kalman filtering to maintain the pose estimate across frames. Although the data input is RGB-D, segmentation is done on the color channels only, and point clouds are cropped using the pixel-wise segmentation output of SegNet. The semantic segmentation and point-cloud cropping proposed in this work seems to be an interesting way to leverage efficient segmentation networks to extract object point clouds while avoiding a 3D segmentation. However, although the proposed multi-hypothesis registration method provides

a coarse estimate of the pose, the complete solution relies on the ICP, thus requiring a model attached to each example.

The method used in [15] takes a local approach to the pose estimation problem. A codebook of scale-independent descriptors is regressed from a training set of synthetic RGB-D images, using a Convolutional Auto-Encoder (CAE). It compresses the data from the randomly sampled patches into a small feature space, where each descriptor contains the features and their attached local votes (the pose of the object it was sampled from). At inference, the input RGB-D image is sampled randomly and a feature is computed for each sample using the CAE. A k-nearest neighbors algorithm is used to find the closest matches in the codebook and votes are applied based on the distance to them. One of the main interest of this approach is that the CAE can leverage training sets of synthetic images and thus does not necessarily require a huge set of application-specific images to train on.

The solution presented in [16] assumes a bounding boxed RGB input from which it regresses the camera pose with respect to the object. A pre-trained and beheaded classification network is used to extract features from the images. Multiple pose networks are then used (one per class) to regress the orientation component of the object's pose (3 DoF), in a quaternion or axis-angle representation. They are composed of a series of fully connected layers with specific non-linearities to model the output space constraints. The paper introduces a geodesic loss function that applies on the rotation matrices space. It is used to fine-tune the overall network after an initial training of the pose networks using a standard mean-squared error. While this paper provides an elegant way to regress the pose of an object, it is limited to the rotation estimation.

[17] recently presented a completely different approach to the problem revolving around an augmented autoencoder used to encode solely in-plane rotations and form a codebook on a dataset of synthetic object views and using domain randomization to allow the network to generalize to real sensor data. At test time, the matching orientation is recovered by performing a k-nearest neighbors search and the position component is recovered by

estimating the distance geometrically and eventually refining it using ICP. Although being able to train this model without the need to collect a dataset and accurate pose labels is interesting, this approach does not appear easily applicable in this project because Computer-Aided Design (CAD) models cannot be accurately used to form a codebook for deformable objects such as poultry.

3.2 Other Approaches

3.2.1 Point Cloud Registration

Registration algorithms align the 3D information obtained from the scene (called reading) with a known model of the object (called reference) by finding the geometric transformation necessary to go from one point cloud to the other.

One of the earliest algorithm developed to this end is the ICP [18], [19]. It matches pairs of points and minimizes the distance between the 2 point clouds. Although it provides a solution to the registration problem, the number of points is a bottleneck as its complexity is in $\mathcal{O}(M, N)$ with M and N the number of points in the reading and reference point clouds. Furthermore, it does not guaranty the convergence to the optimal minimum and because it works at the local scale, obtaining satisfying results requires a proper initialization (overlapping point clouds) that make it practical only in setups where a coarse pose estimate is available. As such, the ICP is often used to refine the results of a preliminary estimate. Despite its drawbacks, ICP has been the topic of numerous improvements over the past 25 years, making it an effective solution in a number of use cases as outlined in a recent review [20].

Another group of registration algorithms working at the global point cloud scale provide a solution to the initial estimate problem. The 4-Points Congruent Sets (4PCS) algorithm [21] relies on the invariance of certain geometries under a rigid transformation to find matching groups of 4 points between two previously unaligned point clouds. While the original algorithm has a quadratic complexity in the number of points, recent improve-

ments taking the form of the Super-4PCS algorithm [22] allow for linear time registration. These global methods are typically used to get an initial registration that can be further refined using ICP.

This family of algorithms is model-based and thus requires each object to be attached a 3D CAD model to perform the registration upon. This does not fit well in an end-to-end setup where the ideal input is a single RGB-D image. What's more, applying this type of algorithm requires a proper segmentation of the target object point cloud in the input data, which is typically done in the color space rather than directly on the point cloud when using calibrated RGB-D sensors. Although this is another problem that has seen many solutions using regular computer vision methods, ranging from simple color-based clustering to graph partitioning, reliable results in non-industrial setup, where the environment cannot be easily controlled, are difficult to obtain. In general, the performance of these methods relies heavily on tweaking parameters to obtain good results on a specific benchmark, but this leads to difficulties when applying them in the real world.

3.2.2 Feature Matching

A different set of algorithms that have been used for pose estimation are feature-based. One of the most widely known is LINE-MOD [23], which was specifically developed to make use of RGB-D data, from which it computes modalities (gradients, normals) both on the color channels and the depth channel, to define local features. The most salient features extracted from a set of reference images are combined to form templates to be matched against the images at run time using a nearest neighbor search. Although LINE-MOD can indeed be used to get an estimation of an object's pose and performs well on textureless objects without the need for a large training dataset, it is slow and suffers from false positive and has a maximum orientation accuracy that depends on the number of views used to form the codebook, as explained by the same authors [24]. Feature matching approaches are not easily transferable to real world scenarios, because variance in the objects, environment

and lighting tend to have a big impact on their performance. Moreover, their application to chicken pose estimation is made even more difficult by the fact that a chicken carcass is a non-rigid and deformable object.

CHAPTER 4
SOLUTION APPROACH

A step-by-step approach toward solving the 3D pose estimation problem is taken. It builds upon existing feature extractors and object detection architectures with the end goal of providing an end-to-end solution for detection and pose estimation. The implementation of the networks is done purely in Python and using the open-source TensorFlow framework. This work assumes that color and optionally depth information are available at train and inference time. CAD models of the objects of interest are also assumed to be available, for visualization purposes only.

4.1 Feature Extraction

As a first stage in the deep learning architecture, features are extracted using a CNN. The objective is to use color information and provide the possibility to incorporate depth information. Available models are pre-trained on the ImageNet database, consisting of millions of images of thousands of object categories. The variety of objects used during the initial training provides their internal layers with enough generalization to be applicable on other sets of objects. The typical approach is then to take a network developed and pre-trained on a classification task and to repurpose it by beheading the network and keeping only the feature extractor and its weights. Although the task at hand is not longer a classification task, the features extracted by the CNN are nevertheless relevant to the pose estimation task.

In the recent years some networks have brought important advances and dominated the ImageNet classification challenge which has been run yearly since 2010 and is based on a subset of the ImageNet database containing objects in a thousand classes. The VGG architecture [5] is used for an initial implementation of the orientation regression described in

[16]. While VGG has a straightforward architecture and smaller receptive fields than previous networks, effectively reducing the number of DoF in the network, its medium-sized version VGG-16 still contains 138 million trainable parameters. As the project moves forward the ResNet architecture is chosen because it provides a deeper network while reducing both the size of the network and the training time by introducing the residual units [6].

In order to incorporate the depth information in the network, a fourth channel is initially added at the input of the feature extractor and the weights for the green channel are copied. At a later step in the project, an approach consisting in combining 2 parallel feature extractors is considered, one for color data and the other for depth. Similarly to the work presented in [25] for fusing temporal and spatial CNNs for action recognition, the goal is to fuse the output of a color feature extractor and a depth feature extractor. The feature maps of the ResNet network are taken after the 4th residual block and fused as described in Equation (4.1) and illustrated in Figure 4.1, with $\boldsymbol{a}$ and $\boldsymbol{b}$ respectively the color and depth feature vectors, α, β, δ and γ the trainable parameters, and K the desired output depth.

$$c_k = \left(\sum_{i=1}^{M} \alpha_{ki} a_i + \gamma_k \right) \odot \left(\sum_{j=1}^{N} \beta_{kj} b_j + \delta_k \right) \tag{4.1}$$

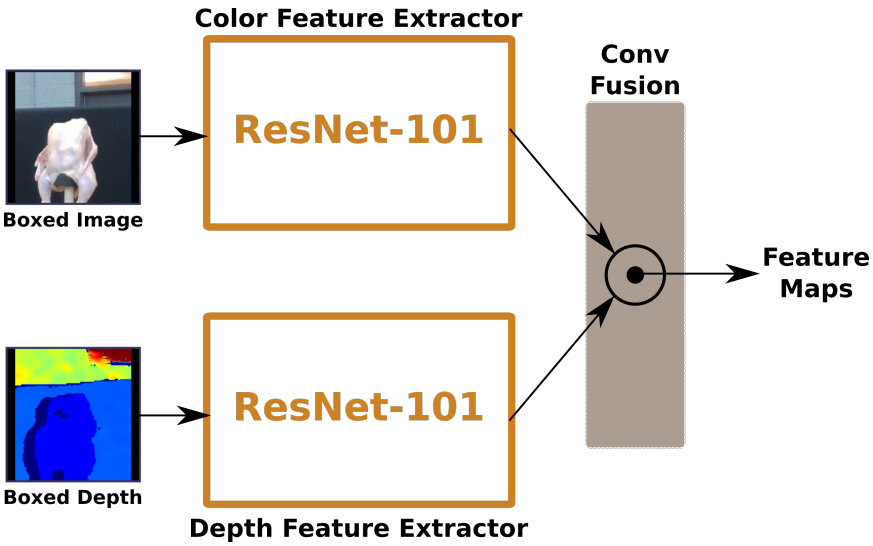

Figure 4.1: Fused Resnet feature extraction networks

Even with a dedicated depth feature extractor, simply adding a parallel CNN and replicating the depth channel across the 2 extra channels does not take full advantage of the color extraction capabilities of the network. The idea is to map the 1-channel depth image to a 3-channel color image using the Jet colormap described in [26], which maps grayscale values in $[0, 255]$ to red, green and blue channels also in $[0, 255]$, where small values are mapped to blue, mid-range values to green and high values to red. The authors have shown that this method of depth colorization performs similarly if not better than the popular HHA method that encodes on 3 channels the height above ground, horizontal disparity and angle between the surface normal and the gravity [27]. The HHA method requires heavy preprocessing of the depth information, whereas the colorization approach only performs a normalization of the data prior to the mapping.

Because the input depth information range and unit depend on the sensor and on the application, the colorization does not assume any particulars and the normalization step is parameterized with a maximum threshold above which the depth values are clipped. This threshold value is determined as the 99th percentile of the original depth value, across the entire dataset, above which points are considered to be spurious or in the background. Once the depth is normalized as floating point values in $[0, 1]$ the mapping is then applied as described in Equation (4.2). The mean value for each colorized depth channel is computed across the dataset and subtracted from the images at run time, similarly to what is done for color images from which the ImageNet RGB mean values are subtracted.

$$\begin{aligned} r &= min(4 \times d - 1.5, -4 \times d + 4.5) \\ g &= min(4 \times d - 0.5, -4 \times d + 3.5) \\ b &= min(4 \times d + 0.5, -4 \times d + 2.5) \end{aligned} \tag{4.2}$$

An example of color image and its associated colorized depth is visible in Figure 4.2. Blue corresponds to the closest objects and red to the farthest, and it is noticeable that the

object's edges constitute salient features in the depth image.

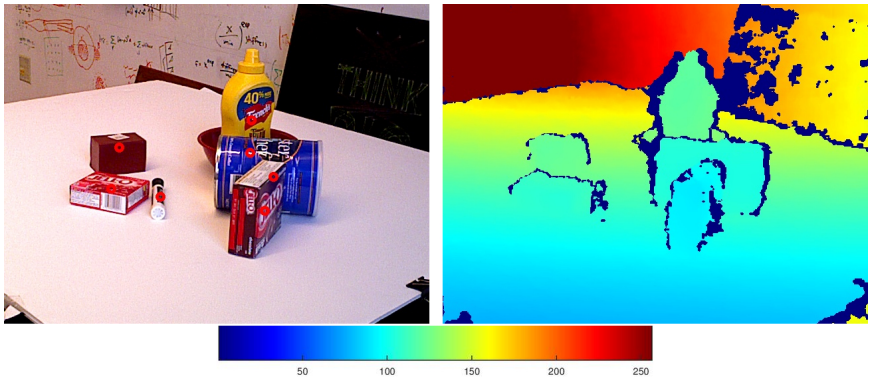

Figure 4.2: Color and jet mapped depth images

The input image and depth are resized to 224 pixels in height and width, with padding to preserve their aspect ratio. Augmentation is applied randomly to the input images, with rotations in $\{0, 90, 180, 270\}$ degrees and scale in $\{1.0, 1.1, 1.2, 1.3, 1.4, 1.5\}$. This augmentation step is illustrated in Figure 4.3.

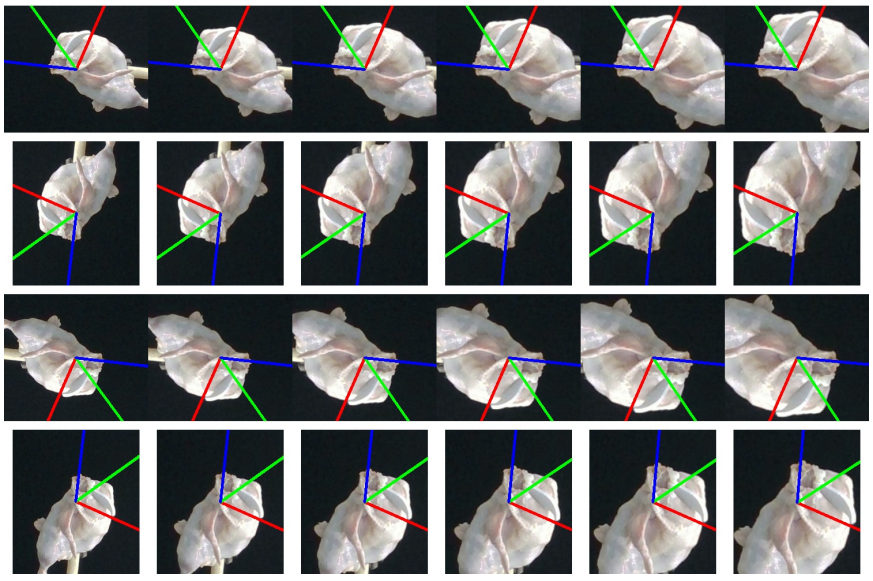

Figure 4.3: Example of augmentation on a chicken color image

4.2 Pose Estimation Network

The objective of this work is to ultimately perform a full 6-D pose regression. As an initial step, the goal is set to develop a network to perform a pure orientation regression, while translation estimation is studied at a later stage.

4.2.1 Orientation Estimation

As described in Section 2.2, the quaternion representation seems best suited for this application and has been successfully used in the past [15], [28]–[30].

Similarly to the approach taken in [16], at an initial stage, the focus is put solely on the pose estimation while the classification and bounding box regression tasks are considered done outside the network. This approach is consistent with the goal of incorporating the pose estimation in an end-to-end solution with a RPN, presented in Section 4.3. The feature extractor input thus consists in bounding boxed images of the objects and the corresponding class.

The pose estimation network consists in a set of fully connected layers that take as input the feature maps from the feature extractor. Several variations of the network using a single or 2 inner fully connected layers have been compared, with different number of neurons. The estimation is class specific and inspired by the work presented in [16], but the approach is different. Whereas the solution presented in the paper proposes a completely separate pose regression branch per class, the pose network here outputs a vector of $4 \times K$, one quaternion regressed for each class, as illustrated in Figure 4.4. This approach is similar to the one taken for semantic segmentation in Mask R-CNN [12], where the pose head regresses K masks using a fully convolutional network.

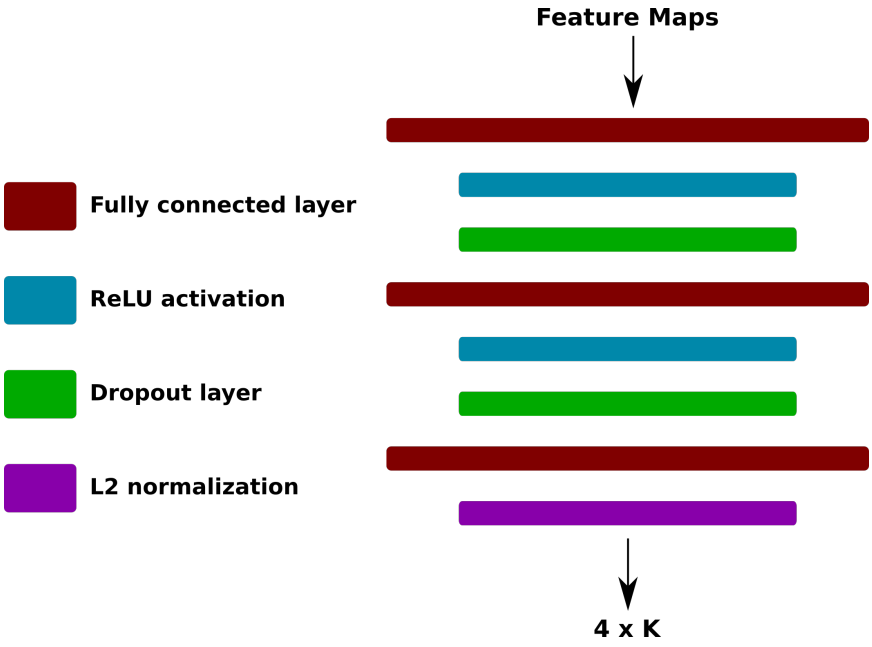

Figure 4.4: Orientation network

Several loss functions are considered for quaternion regression. Because normalized quaternions lie on the unit sphere, a geodesic distance is primarily used as the error function (Equation (4.3)). It represents the shortest arc between 2 unit quaternions and can serve as the loss function, as successfully done in [16] and explained in more details in [31]. Because of rounding errors in the L2 normalization, the dot product of the 2 normalized quaternions may end up exceeding 1, but the arccosine is restricted to $[-1, 1]$. To prevent spurious NaN values appearing in the loss, the dot product is clipped. Alternatively, a more numerically stable log loss (Equation (4.4)) is also considered.

$$\mathcal{L}_q = 2cos^{-1}(|\hat{\boldsymbol{q}} \cdot \boldsymbol{q}|) \tag{4.3}$$

$$\mathcal{L}_q = log(1 + 10^{-4} - |\hat{\boldsymbol{q}} \cdot \boldsymbol{q}|) \tag{4.4}$$

ReLU activations are used on the inner fully connected layers and an L2 normalization (Equation (4.5)) is added with $\epsilon = 10^{-12}$ to enforce the versor's unit norm constraint and

thus ensure that the estimation is always a valid rotation. Experimentations with a leaky ReLU are conducted, to avoid the death of neurons when instabilities in geodesic distance cause the gradients to explode. The leak consists in a small gradient introduced for negative activations, effectively offering a path for the neuron to recover.

$$\hat{\boldsymbol{q}} = \frac{\boldsymbol{y}}{max(\sqrt{\sum_{i=1}^{4} y_i^2}, \epsilon)} \tag{4.5}$$

Based on standard guidelines for deep learning and experimental results, several other modifications are added in the implementation, and can be enabled or disabled in order to compare network performances. Batch normalization is applied in order to reduce training time by normalizing the outputs of each internal fully connected layer across the mini-batch, with $\epsilon = 10^{-3}$, as shown in Equation (4.6). Batch normalization is applied before the activation function, as suggested in the original paper [32]. The bias term in the dense layer is omitted because the batch normalization applies a shift β that will be learned. Furthermore, the scaling is not performed by batch normalization but by the following linear activation, and thus Equation (4.7) shows only the shift.

$$\bar{\boldsymbol{z}} = \frac{\boldsymbol{z} - \mu}{\sqrt{\sigma^2 + \epsilon}} \tag{4.6}$$

$$BN(\boldsymbol{z}) = \bar{\boldsymbol{z}} + \beta \tag{4.7}$$

Despite the small regularization effect introduced by the batch normalization which adds a small batch noise, dropout [33] is also used to reduce overfitting by randomly dropping neurons with a probability $p = 0.5$, reduced to $p = 0.1$ if batch normalization is activated, as suggested in [32].

4.2.2 Position Estimation

Regressing the translation representing the object's orientation in the 3-dimensional camera space appears to be the most elegant solution. However, this translation is difficult to describe and constrain in a way that can be enforced in the network, because the values on each axis can vary greatly depending on the application and the field of view of the camera. Indeed images do not convey the full information on the 3-dimensional camera space, as opposed to colored point clouds. However, standard convolutional neural networks cannot be used directly on 3D point clouds and 2D color and depth images are thus used in most pose estimation approaches.

Furthermore, because this work assumes the availability of color and depth information, as well as the camera intrinsic matrix, it is possible, for scenarios where there is no occlusion, to reduce the problem to an estimation of the 2-dimensional translation in the image space. The translation values are normalized to the $[0, 1]$ range with the origin at the top left corner of the image. It is then possible to recover the 3-dimensional translation by remapping the estimated 2D coordinates in pixel coordinates and subsequently using the camera intrinsic matrix and depth value at those coordinates to uniquely define the point in the 3D camera space. This function is described in Equation (4.8), where (t_x, t_y) are the estimated position coordinates in pixels, $\boldsymbol{D}$ the depth image, $\boldsymbol{K}$ the camera intrinsics, (c_x, c_y) and (f_x, f_y) are respectively the coordinates of the principal point and focal lengths of the camera. Alternatively, if occlusions are an issue, geometrical estimation of the distance can be used, such as the projective distance estimation in [17].

$$(t_x, t_y, \boldsymbol{D}, \boldsymbol{K}) \rightarrow \begin{bmatrix} \frac{(t_x - K_{1,3}) \times D_{t_y,t_x}}{K_{1,1}} \\ \frac{(t_y - K_{2,3}) \times D_{t_y,t_x}}{K_{2,2}} \\ D_{t_y,t_x} \end{bmatrix} = \begin{bmatrix} \frac{(t_x - c_x) \times D_{t_y,t_x}}{f_x} \\ \frac{(t_y - c_y) \times D_{t_y,t_x}}{f_y} \\ D_{t_y,t_x} \end{bmatrix} = \begin{bmatrix} t'_x \\ t'_y \\ t'_z \end{bmatrix} \tag{4.8}$$

Several options are then studied to perform a complete 6D pose estimation combining 3D orientation and 2D position regressions. The last fully connected layer of the existing

orientation estimator can be extended to have an output of $6 \times K$, with the regressed vector defined as $[q\ t]$. This is similar to the approach taken in [34] for regressing a complete 7-dimensional vector containing 3D translation and quaternion for camera relocalization applications. In that case, the output vector is split after the hyperbolic tangent activation in order to apply the L2 normalization solely on the quaternion. This solution is illustrated in Figure 4.5. The first fully connected layer always has a size of 2048, while the second fully connected layer, if used, has a size of 512.

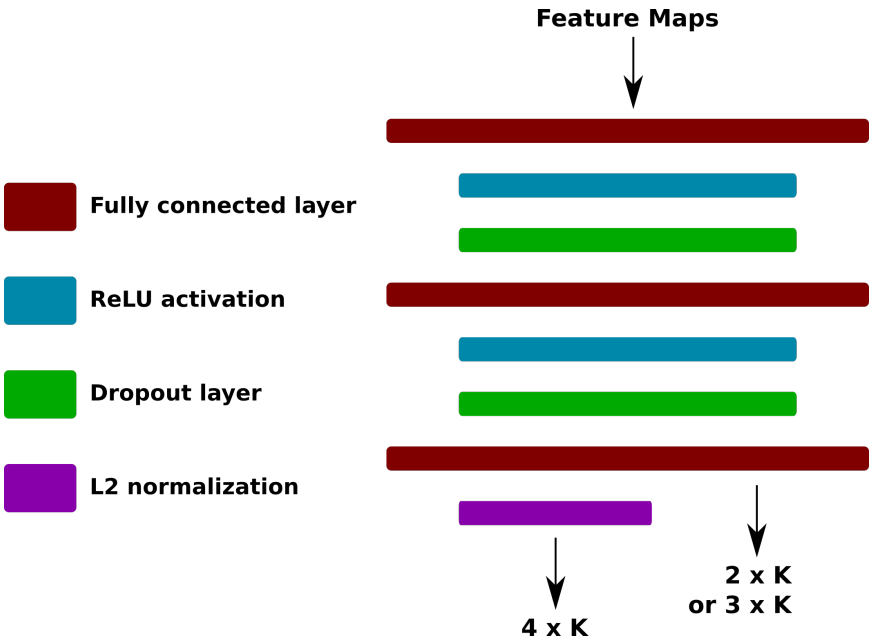

Figure 4.5: Single branch pose estimation network

A second option is to have a separate set of fully connected layers for translation regression. The layer structure and hyperparameters are the same as the quaternion regression branch.

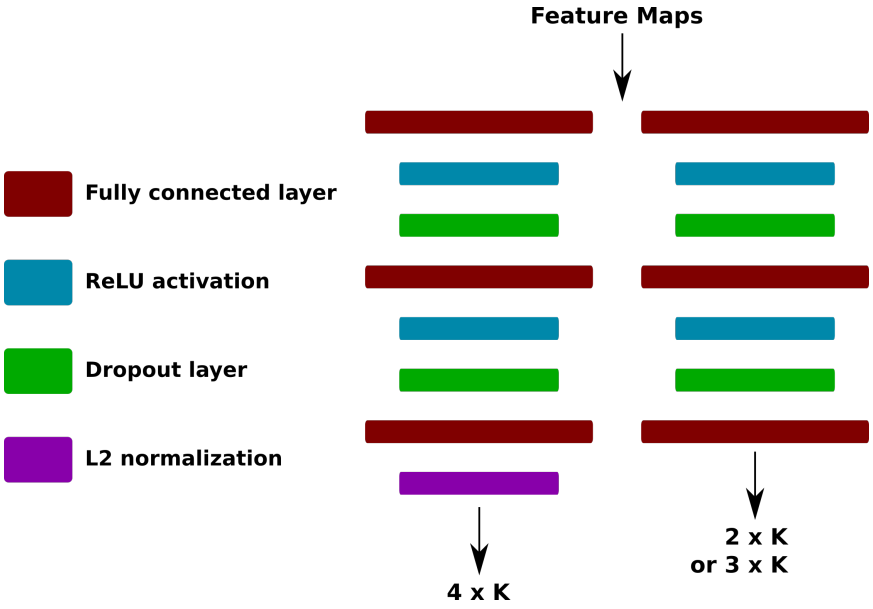

Figure 4.6: Split branches pose estimation network

A simple Euclidean distance (Equation (4.9)) is used as the loss and error function for translation regression. The overall loss (Equation (4.10)), combining quaternion and translation losses, is defined with an additional γ coefficient, as suggested in [34]. This is necessary because the losses are not expressed in the same units or scales and thus need to be balanced. $\gamma = 1$ if the geodesic loss is used, and $\gamma = 10^{-2}$ if the log loss is chosen.

$$\mathcal{L}_t = \left\| \hat{\boldsymbol{t}} - \boldsymbol{t} \right\|_2 = \sqrt{(\hat{t_x} - t_x)^2 + (\hat{t_y} - t_y)^2} \tag{4.9}$$

$$\mathcal{L} = \gamma \mathcal{L}_q + \mathcal{L}_t \tag{4.10}$$

4.2.3 Evaluation Metrics

During the initial phase when only the orientation component is estimated, the metric is limited to the mean geodesic distance and the accuracy is measured based on 2 thresholds: the distance is considered best if under 14 degrees, good if between 14 and 34 degrees.

Later, the Average Distance (ADD) metric is used. Presented in [24] and recently used in [30] as a benchmark for the Yale-CMU-Berkeley Dataset (YCB), it is defined as the

average distance between the points of a CAD model rotated by the groundtruth pose on one side, and the estimated pose on the other. For symmetrical objects, where a one-to-one point matching is ambiguous, the metric computes the average distance between the closest points. An estimated pose is considered accurate if the distance is below a threshold, set to 10% of the CAD model diameter in [30].

The quaternions are transformed into rotation matrices $\boldsymbol{R}$ and $\hat{\boldsymbol{R}}$ following Equation (2.3), and it is applied with the translation vector on the set $\mathcal{M}$ of points, before computing the appropriate average distance, described in Equation (4.11) and Equation (4.12).

$$ADD = \frac{1}{|\mathcal{M}|} \sum_{p \in \mathcal{M}} \left\| (\hat{\boldsymbol{R}}\boldsymbol{p} + \hat{\boldsymbol{t}}) - (\boldsymbol{R}\boldsymbol{p} + \boldsymbol{t}) \right\| \tag{4.11}$$

$$ADD - S = \frac{1}{|\mathcal{M}|} \sum_{p_1 \in \mathcal{M}} \min_{p_2 \in \mathcal{M}} \left\| (\hat{\boldsymbol{R}}\boldsymbol{p_1} + \hat{\boldsymbol{t}}) - (\boldsymbol{R}\boldsymbol{p_2} + \boldsymbol{t}) \right\| \tag{4.12}$$

4.3 End-to-end Solution

The overall objective of the project conducted at FPTD is to provide an end-to-end solution for object detection, classification and pose estimation. The option chosen early on is to extend the well known RPN architectures, presented in Section 2.4, and specifically the Faster R-CNN model.

The objective is to add a third head for pose estimation, positioned at a second stage and thus exploiting both the features from the RoI pooling layer and the estimation from the classification head, to select the appropriate pose estimate at the output. This approach is similar to the one taken in [12] to append a semantic segmentation head to the Faster R-CNN RPN trunk, as illustrated in Figure 4.7. Mask R-CNN also replaces the RoI pooling layer introduced in Faster R-CNN [11] to allow single pass computation on multiple RoIs with the RoI align layer. This new pooling layer does not apply any quantization when computing the RoI corners and uses bi-linear interpolation during RoI sampling to produce

more accurate max pooled features.

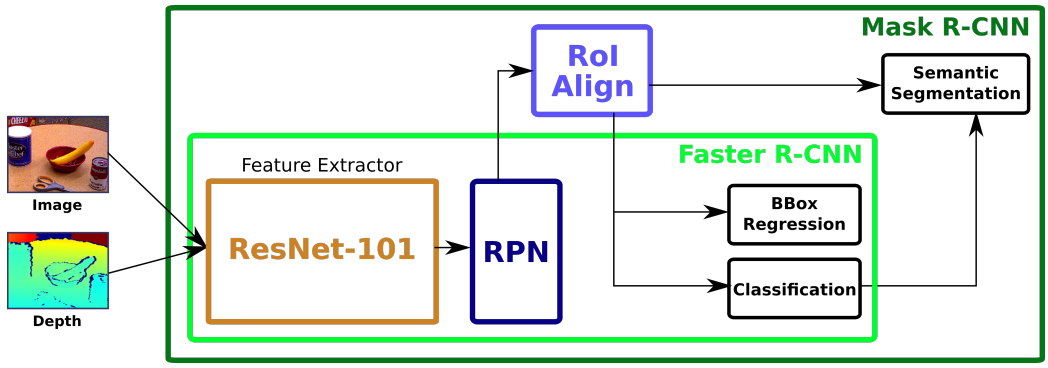

Figure 4.7: Mask R-CNN semantic segmentation head on Faster R-CNN trunk

While the RoI align layer aims at improving the accuracy on the semantic segmentation task, the authors show that it also leads to a significant improvement on the bounding box regression task. The integration of the pose head can thus be done using either a Faster R-CNN or a Mask R-CNN trunk, for which the only difference is the RoI pooling layer. To simplify the block diagram, the pose estimation integration is illustrated with a Faster R-CNN trunk in Figure 4.8. A ResNet feature extractor is used in this diagram to draw a connection with what was done in Figure 4.1, but other feature extractors could be used, as explained in Section 4.1.

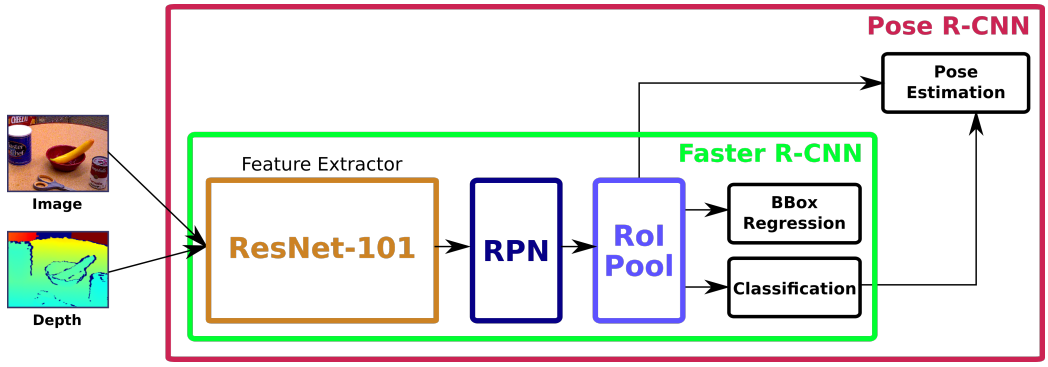

Figure 4.8: RPN with pose estimation head on Faster R-CNN trunk

The initial implementation is an attempt to modify the TensorFlow Object Detection Application Programming Interface (API), available under an Apache license in TensorFlow's research models [35], and add the pose estimation head to the Faster R-CNN trunk.

The trunk is defined as a meta architecture in the API and also provides RoI align and semantic segmentation features of Mask R-CNN. These have been omitted in Figure 4.9 for simplification.

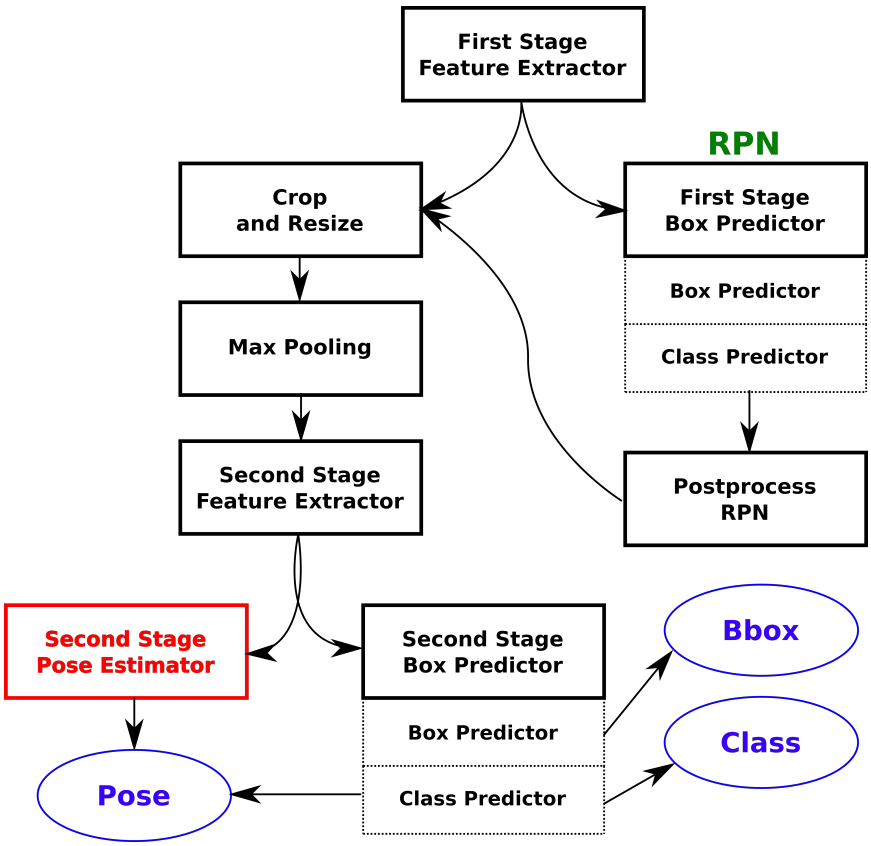

Figure 4.9: Illustration of TensorFlow's Faster R-CNN meta architecture

The repository is forked and the meta architecture is modified in depth because the API does not have a readily available method to extend the existing meta architectures with an additional head. The new groundtruth pose labels must be carried through the input pipeline and made accessible to the pose head, and additional parameters have to be included in the configuration files, which implies modifications to the protocol buffers that provide the configuration definitions, and generally to the entire codebase. The complexity of the architecture and the fact that training and evaluation loops are handled through the higher level TensorFlow Slim library makes debugging more difficult. After numerous attempts, the choice is made at this point to pursue the development of the pose estimation

network separately, as described in Section 4.2. This makes sense considering that the classification and bounding box regression are considered solved problems, and the focus is thus put specifically on the pose estimation task.

Essentially, once the pose estimation network is operational on its own, appending the head to a Faster R-CNN trunk consists in adding the pose regression layers to an existing implementation and registering the new loss functions and metrics. The features are no longer extracted by a feature extractor dedicated to pose estimation, as shown in Section 4.2, but by the feature extractor common to all regression heads. The pooled features are taken at the output of the RPN and the classification estimate provides the information necessary to select the appropriate pose estimate for the loss computation, similarly to what is done by Mask R-CNN and as shown in Figure 4.8.

Before integrating the end-to-end solution as described above, and if computer resources are available, an instance of Faster R-CNN can be loaded in memory along with an instance of the ResNet and pose estimation network. The bounding box prediction is used to crop the input image before feeding it to the ResNet feature extractor which performs pose estimation in conditions similar to what is described in Section 4.2. While inefficient and unsatisfactory as a final solution, this provides an intermediate step after the development of the pose estimation network and before taking on the end-to-end training of the new Pose R-CNN architecture.

4.4 Robotic Application

The project conducted at the FPTD, while aiming in the long term at solving the general problem of pose estimation for industrial applications, has a specific focus on poultry processing. However, no public dataset is available to train the network and thus the objective is set to gather an initial chicken dataset.

The biggest challenge in collecting a dataset suiting the requirements of this project is to get accurate pose information. Although hand labeling the pictures after the acquisition is

an option, this is time consuming and accuracy is limited by human error. The deformable nature and varying appearance of the chickens in this case increases the difficulty of the task.

The availability in the division of a Universal Robots UR5 robotic arm and its control stack with inverse kinematics as well as multiple Intel RealSense cameras provide a way to design a method to acquire automatically color and depth data and their associated poses at no extra cost. Indeed, the robot arm's odometry data and forward kinematics gives an accurate pose information. Furthermore, multiple cameras allow to capture different viewpoints for a single robot pose, augmenting the dataset and enabling future experimentation of multiview pose estimation.

A support for the chicken carcass, visible in Figure 4.10 is mounted on the end effector joint of the robot using a rigid plate (in red). An ArUco marker [36] stand is installed on the support (in yellow) and a cone is attached perpendicularly to the end effector rotation axis (in green).

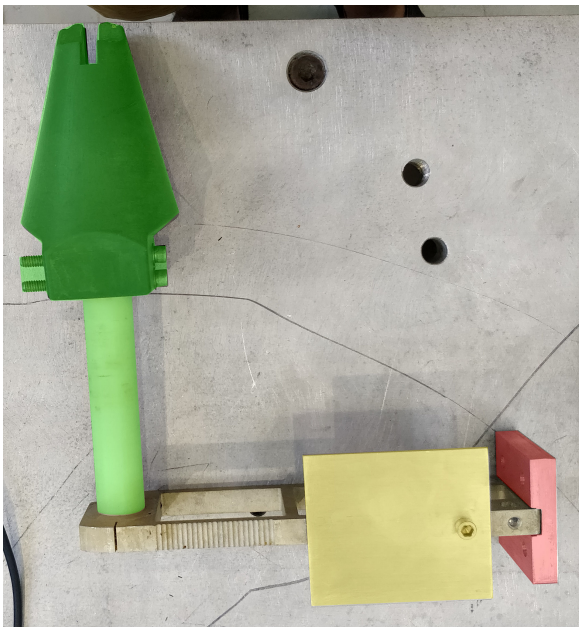

Figure 4.10: Chicken carcass support

A set of ROS nodes is developed to wrap the whole acquisition system. The first phase

consists in calibrating the cameras with the ArUco marker. The detection node is run separately on each camera and the transform from the fiducial frame to the camera optical frame is published using the tf library [37]. Static transforms from the robot's end effector joint to the center of the cone and to the center of the ArUco marker are measured directly on the chicken support (Figure 4.10) and published. This allows to construct the tf tree shown simplified in Figure 4.11, effectively allowing to recover the chicken pose in the camera frame.

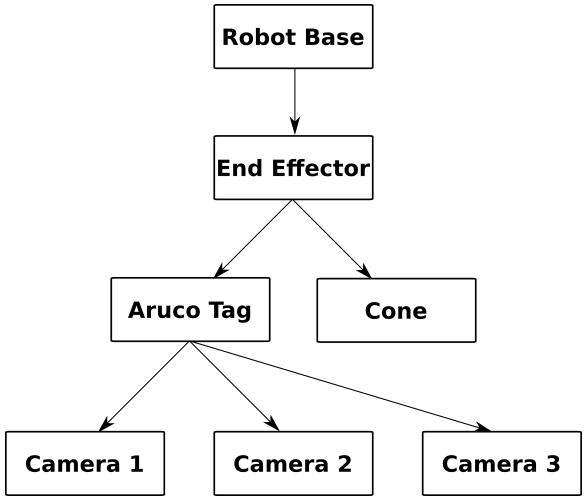

Figure 4.11: Simplified ROS tf tree

The final acquisition setup visible in Figure 4.12 comprises a UR5 arm (in red) with a cone mounted as its end effector to support the chicken carcass (in green) and a set of 3 RealSense cameras mounted on articulated arms with camera brackets (in blue).

Figure 4.12: Chicken dataset acquisition setup

A ROS service takes in a file containing a set of randomly generated poses around a predefined point located in the field of view of the cameras. A request to the service is translated in a command through the UR5 driver and the service responds once the target is reached, triggering the acquisition. A node running for each camera collects the color and depth images and the object to camera transform. During the initial frame recording, the intrinsic matrix for each camera and the camera to camera transforms are also saved. A total of 11 birds is used for the acquisition with roughly a 100 pose per chicken for a total of 1,200 robot poses or 3,600 available examples. An example of the 3 views for one of the robot pose is shown in Figure 4.13.

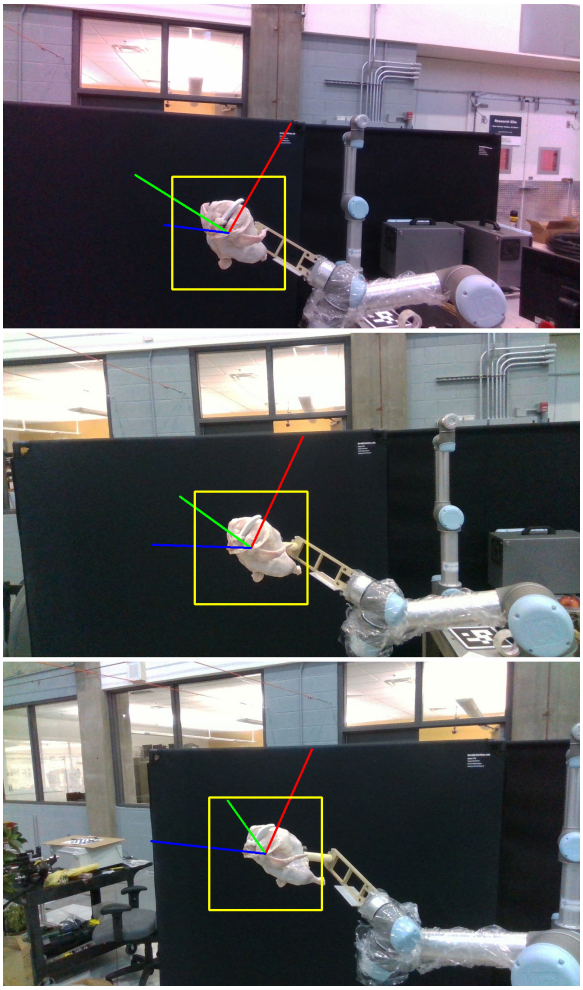

Figure 4.13: Chicken acquisition example

After the acquisition, the bounding boxes are generated for each image by projecting the recorded 3D position to the image plane and taking a simple square around it with a side of 300 pixels. Although not directly useful to the pose estimation, all the color pointclouds are generated. The alignment of the data is checked by republishing the camera to camera transforms in ROS and visualizing the 3 pointclouds in Rviz [38].

While this setup is limited to recording the pose of a single object, and to do so in a very specific setup, with the robot arm visible in the images, it enables the rapid collection of precise data with multiple cameras, and could be adapted to other classes of objects. Another and more versatile variation, bearing some similarities to the approach taken in [30] to collect YCB, would see the camera mounted on the arm and taking successively

different random views of the target object or group of objects. By recording an initial pose of the objects in the camera frame using fiducial markers or manually after the acquisition, and by tracking the camera motion, an accurate dataset could be collected.

CHAPTER 5
EVALUATION

The evaluation is conducted on the application dataset collected using the method described in Section 4.4. The pose estimation network is evaluated on its own, using a ResNet feature extractor, or a pair of identical feature extractors with a multiplicative fusion layer when depth input is enabled, as explained in Section 4.1.

Because the input data consists in bounding boxed images, and the bounding box labels in the data are located exactly at the object's center, a noise sampled uniformly in the ± 30 pixel range is applied on each of the coordinates of the box corners before resizing and padding the images to the 224 pixel size. This randomization is performed once on the entire dataset, before starting the training. As a result, a variety of aspect ratios are introduced and the objects are no longer perfectly centered, which would have been trivial for the network to learn, and they are no longer always entirely visible in the image frame.

2 different train and validation splits are used for evaluation. In the first split method, data on all birds is merged and the 3 views of the same object's pose are included in the same part of the split, with 70 % of the data in training and 30 % in validation. In the second method, the data is split by bird, with the data for 8 birds in training, and 3 remaining birds in validation. The initial trainings are conducted using the first split method, but for comparison some results on the second method are reported for reference in italic in Table 5.1 and Table 5.2.

The 2 evaluation metrics described in Section 4.2 are used to evaluate the different versions of the network. The geodesic distance and euclidean distance metric, for which the results are reported in Table 5.1 is a custom metric that is useful during the development of the network because it provides a separate estimate of the performance on orientation and position estimation. Orientation estimation accuracy is given in 3 bins for great predictions

(under 14 degrees), good predictions (between 14 and 34 degrees) and bad predictions (above 34 degrees).

The ADD gives a measure of the distance between the projected points in meters, and it is generally considered that a distance under 10 % of the object's diameter corresponds to an estimation that looks visually correct. However, while this metric is used throughout the literature, it is designed for non-deformable objects, which the chickens are not. As such, no single CAD model can be used as a perfect substitute to all birds, and unless a 3D scan of each chicken is available, which is not the case for this initial dataset, the ADD results given here are imperfect. They are nonetheless reported in Table 5.2 as they provide an idea of the impact of the orientation and position estimation interaction on the overall quality of the pose estimation. In order to compute the average distance, the estimated image space translation (u, v) is used to compute the 3D translation as described in Equation (4.8) using the depth data.

The models are named based on the number of fully connected layers, whether they use the split (2-branch) or combined (1-branch) network structure described in Section 4.2, and if the depth information is used or not.

Table 5.1: Results on the geodesic distance and euclidean distance metric (italics show results for bird split)

Model	Mean d_t (px)	Mean d_q (deg)	Great	Good	Bad
2 FC, 2 BR + D	8.51	10.41	0.82	0.17	0.01
2 FC, 2 BR	7.83	11.29	0.80	0.16	0.03
2 FC, 2 BR	*7.90*	*11.98*	*0.73*	*0.26*	*0.01*
3 FC, 2 BR + D	10.57	**8.61**	**0.89**	0.10	0.01
3 FC, 2 BR	**7.59**	11.57	0.80	0.17	0.03
2 FC, 1 BR + D	9.64	9.56	0.88	0.11	0.01
2 FC, 1 BR	11.88	10.13	0.88	0.10	0.02
3 FC, 1 BR + D	9.58	9.84	0.82	0.17	0.01
3 FC, 1 BR	9.38	12.68	0.79	0.16	0.05

The results in Table 5.1 clearly show that the depth information is useful to the orientation estimation, with a consistent improvement across the different variations but that it also appears to have a negative impact on the translation estimation, which might be explained

Table 5.2: Results on the average distance (ADD) metric

Model	Mean (cm)	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}	M_{15}
2 FC, 2 BR + D	5.96	0.71	0.75	0.80	0.83	0.86	0.88	0.89	0.90
2 FC, 2 BR	6.37	0.72	0.76	0.79	0.83	0.85	0.87	0.88	0.89
2 FC, 2 BR	*6.12*	*0.70*	*0.75*	*0.79*	*0.83*	*0.85*	*0.87*	*0.88*	*0.90*
3 FC, 2 BR + D	5.79	0.59	0.68	0.75	0.79	0.83	0.86	0.88	0.89
3 FC, 2 BR	5.81	0.71	0.77	**0.81**	0.85	0.87	0.89	0.90	0.91
2 FC, 1 BR + D	**5.38**	0.65	0.72	0.77	0.81	0.83	0.85	0.88	0.89
2 FC, 1 BR	6.08	0.49	0.59	0.68	0.74	0.78	0.83	0.86	0.88
3 FC, 1 BR + D	6.27	0.65	0.71	0.76	0.80	0.83	0.84	0.85	0.87
3 FC, 1 BR	6.06	0.65	0.72	0.77	0.81	0.83	0.85	0.87	0.88

by a slight misalignment of the color and depth information. As a result, Table 5.2 suggests that the interactions between the poorer translation estimation and improved orientation estimation do not lead to an increase in matching at 10 % of the model's size.

Overall, it appears that the 2-branch network with 3 fully connected layers and without depth performs best. Further study is necessary in order to improve the depth and color fusion, which could realistically lead to improved pose estimation, and a better approach regarding the chicken CAD model will need to be put in place in order to obtain more meaningful ADD metric results. An example of the pose estimates obtained with this network on the validation set is visible in Figure 5.1.

In a scenario where the robot arm is equipped with a gripper, which is the objective of the project conducted at FPTD, the quality of the predictions informs the choice of the gripping strategy. Whereas a finger-based gripper requires a very accurate orientation and position prediction, which this architecture does not achieve, it is currently not considered as the option of choice for chicken carcass grasping. Indeed, the deformable nature of the chicken carcass and its slippery surface make finger-based grippers difficult to use. Suction-based grippers and soft-robotic approaches are the most likely candidates in this project and generally do not have as strong requirements on the pose estimation in order to perform a successful grasp on the object. As such, although this has to be confirmed at a later stage in the project, the results described here are promising, particularly on

orientation estimation, with under 10 degrees average distance with the ground truth in validation. Initial inference testing using a different bird moved by hand shows interesting results on the orientation estimate, as shown on the point clouds in Figure 5.2.

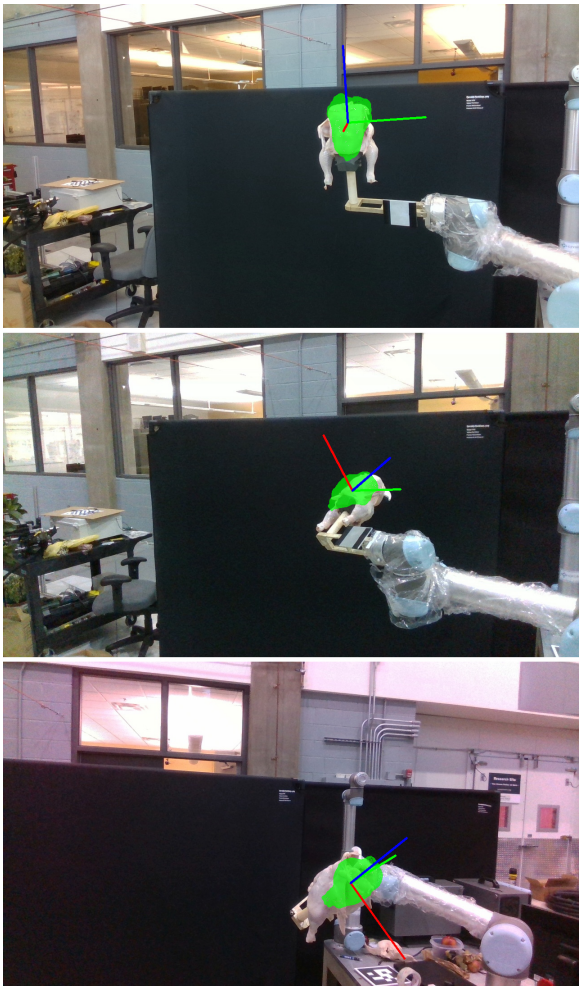

Figure 5.1: Chicken results on validation set

Figure 5.2: Chicken inference test

CHAPTER 6
CONCLUSION AND FUTURE WORK

This work presents a 6D pose regression architecture based on standard feature extractors and building upon previous research in pose estimation. It explores different network structures that allow for the fusion of depth information from commercial off-the-shelf sensors. It focuses on the pose estimation head development, but the integration as an end-to-end solution using a RPN trunk is initiated and a path toward this goal is described for future developments.

It also proposes an approach for the automated collection of a multi-camera color and depth dataset with pose labels, and demonstrates its application for the acquisition of chicken data. This novel dataset is used to evaluate the network as part of an ongoing research effort on poultry processing conducted at FPTD. The multi-view data opens the way to further research on pose estimation using multiple cameras, as a mechanism to refine the pose estimation where ICP cannot be used because of the deformable nature of the objects.

REFERENCES

[1] F. Dunn and I. Parberry, *3D Math Primer for Graphics and Game Development*, English, 2 edition. Boca Raton, FL: A K Peters/CRC Press, Nov. 2011, ISBN: 978-1-56881-723-1.

[2] H. Alemi Ardakani and T. Bridges, *Review of the 3-2-1 Euler Angles: A yaw–pitch–roll sequence*, English, 2010.

[3] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, ISSN: 0018-9219. DOI: 10.1109/5.726791.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2012, pp. 1097–1105.

[5] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556 [cs]*, Sep. 2014. arXiv: 1409.1556 [cs].

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385 [cs]*, Dec. 2015. arXiv: 1512.03385 [cs].

[7] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," *arXiv:1409.4842 [cs]*, Sep. 2014. arXiv: 1409.4842 [cs].

[8] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," en, p. 8,

[9] J. Nagi, F. Ducatelle, G. A. D. Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, Nov. 2011, pp. 342–347. DOI: 10.1109/ICSIPA.2011.6144164.

[10] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1440–1448.

[11] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arXiv:1506.01497 [cs]*, Jun. 2015. arXiv: 1506.01497 [cs].

[12] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," *arXiv:1703.06870 [cs]*, Mar. 2017. arXiv: 1703.06870 [cs].

[13] J. M. Wong, V. Kee, T. Le, S. Wagner, G.-L. Mariottini, A. Schneider, L. Hamilton, R. Chipalkatty, M. Hebert, D. M. S. Johnson, J. Wu, B. Zhou, and A. Torralba, "SegICP: Integrated Deep Semantic Segmentation and Pose Estimation," *arXiv:1703.01661 [cs]*, Mar. 2017. arXiv: 1703.01661 [cs].

[14] V. Badrinarayanan, A. Handa, and R. Cipolla, "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Robust Semantic Pixel-Wise Labelling," *arXiv:1505.07293 [cs]*, May 2015. arXiv: 1505.07293 [cs].

[15] W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab, "Deep Learning of Local RGB-D Patches for 3D Object Detection and 6D Pose Estimation," *arXiv:1607.06038 [cs]*, Jul. 2016. arXiv: 1607.06038 [cs].

[16] S. Mahendran, H. Ali, and R. Vidal, "3D Pose Regression using Convolutional Neural Networks," *arXiv:1708.05628 [cs]*, Aug. 2017. arXiv: 1708.05628 [cs].

[17] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, "Implicit 3D Orientation Learning for 6D Object Detection from RGB Images," en, in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., ser. Lecture Notes in Computer Science, Springer International Publishing, 2018, pp. 712–729, ISBN: 978-3-030-01231-1.

[18] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, Feb. 1992, ISSN: 0162-8828. DOI: 10.1109/34.121791.

[19] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, Range Image Understanding, vol. 10, no. 3, pp. 145–155, Apr. 1992, ISSN: 0262-8856. DOI: 10.1016/0262-8856(92)90066-C.

[20] F. Pomerleau, F. Colas, and R. Siegwart, "A Review of Point Cloud Registration Algorithms for Mobile Robotics," English, *Foundations and Trends® in Robotics*, vol. 4, no. 1, pp. 1–104, May 2015, ISSN: 1935-8253, 1935-8261. DOI: 10.1561/2300000035.

[21] D. Aiger, N. J. Mitra, and D. Cohen-Or, "4pointss Congruent Sets for Robust Pairwise Surface Registration," in *ACM SIGGRAPH 2008 Papers*, ser. SIGGRAPH '08,

New York, NY, USA: ACM, 2008, 85:1–85:10, ISBN: 978-1-4503-0112-1. DOI: 10.1145/1399504.1360684.

[22] N. Mellado, N. Mitra, and D. Aiger, "Super4PCS: Fast Global Pointcloud Registration via Smart Indexing," *Computer Graphics Forum*, vol. 33, no. 5, 2014. DOI: 10.1111/cgf.12446.

[23] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit, "Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes," in *2011 International Conference on Computer Vision*, Nov. 2011, pp. 858–865. DOI: 10.1109/ICCV.2011.6126326.

[24] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab, "Model Based Training, Detection and Pose Estimation of Texture-Less 3D Objects in Heavily Cluttered Scenes," en, in *Computer Vision – ACCV 2012*, K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, Eds., ser. Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 548–562, ISBN: 978-3-642-37331-2.

[25] E. Park, X. Han, T. L. Berg, and A. C. Berg, "Combining multiple sources of knowledge in deep CNNs for action recognition," in *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Mar. 2016, pp. 1–8. DOI: 10.1109/WACV.2016.7477589.

[26] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust RGB-D object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 681–687. DOI: 10.1109/IROS.2015.7353446.

[27] S. Gupta, R. Girshick, P. Arbeláez, and J. Malik, "Learning Rich Features from RGB-D Images for Object Detection and Segmentation," en, *arXiv:1407.5736 [cs]*, Jul. 2014. arXiv: 1407.5736 [cs].

[28] Y. Li, G. Wang, X. Ji, Y. Xiang, and D. Fox, "DeepIM: Deep Iterative Matching for 6D Pose Estimation," *arXiv:1804.00175 [cs]*, Mar. 2018. arXiv: 1804.00175 [cs].

[29] A. Kendall and R. Cipolla, "Geometric Loss Functions for Camera Pose Regression with Deep Learning," IEEE, Jul. 2017, pp. 6555–6564, ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.694.

[30] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes," *arXiv:1711.00199 [cs]*, Nov. 2017. arXiv: 1711.00199 [cs].

[31] D. Q. Huynh, "Metrics for 3D Rotations: Comparison and Analysis," en, *Journal of Mathematical Imaging and Vision*, vol. 35, no. 2, pp. 155–164, Oct. 2009, ISSN: 0924-9907, 1573-7683. DOI: 10.1007/s10851-009-0161-2.

[32] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," en, *arXiv:1502.03167 [cs]*, Feb. 2015. arXiv: 1502.03167 [cs].

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," en, p. 30,

[34] A. Kendall, M. Grimes, and R. Cipolla, "PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2938–2946.

[35] J. Huang, V. Rathod, R. Votel, D. Chow, C. Sun, M. Zhu, A. Fathi, and Z. Lu, *TensorFlow Object Detection API*, Google, Nov. 2018.

[36] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and Vision Computing*, vol. 76, pp. 38–47, Aug. 2018, ISSN: 0262-8856. DOI: 10.1016/j.imavis.2018.05.004.

[37] T. Foote, "Tf: The transform library," in *2013 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, ser. Open-Source Software workshop, 2013, pp. 1–6. DOI: 10.1109/TePRA.2013.6556373.

[38] J. Vaughan, *ROS 3D Robot Visualizer.* ROS, Nov. 2018.