Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

A Performance Evaluation of Pixel-wise Voting Networks for Object Pose Estimation

Master Candidate

Lorenzo Mantovan

Student ID 2006673

Supervisor

Prof. Alberto Pretto

University of Padova

Academic Year
2022/2023

To my family
and friends.

Abstract

Computer vision is an interdisciplinary field of study that studies algorithms and techniques to enable computers to recognise subjects and extract useful information within an image or other visual input. In other words, it aims to make machines capable of reconstructing a context around an image, giving it real meaning. Among the most important tasks in this area is the 6 degree of freedom pose estimation, i.e. the detection of the pose (translation and rotation) of an object given an input image. This thesis employs PVNet, one of the newest and best-known methods in the literature, to perform several tests: the effectiveness of introducing the DSAC module, the influence of the Pnp type on performance, the validity of using synthetic datasets and the search for an effective strategy for generating them, the dependence of the network on a quantity of real images in the dataset during the training set, and the search for optimal parameters for the score and loss functions. PVNet variants were trained using the LINEMOD dataset. The experiments showed that:

- the best configuration turns out to be the one with DSAC and with EPnP;
- the more the synthetic dataset generation strategy produces varied data close to reality, the more effective it is;
- the network turns out to be very dependent on real data in the training phase;
- the right calibration of the parameters of the DSAC module and the loss function can make the network achieve very good results.

Sommario

La Computer Vision è un campo di studi interdisciplinare che studia algoritmi e tecniche per permettere ai computer di riconoscere i soggetti e di estrarre informazioni utili allinterno di unimmagine o di altri input visivi. In altre parole, essa mira a rendere le macchine capaci di ricostruire un contesto intorno allimmagine, dandole un vero e proprio significato. Tra i compiti più importanti in questo ambito troviamo la stima della posa a 6 gradi di libertà, ovvero lindividuazione della posa (traslazione e rotazione) di un oggetto data unimmagine in input. Questa tesi impiega PVNet, uno tra i metodi più recenti e noti in letteratura, per eseguire diversi test: lefficacia dellintroduzione del modulo DSAC, linfluenza della tipologia del Pnp nelle performances, la validità dellutilizzo di dataset sintetici e la ricerca di una strategia efficacie per la loro generazione, la dipendenza della rete a una quantità di immagini reali nel dataset durante il training set e la ricerca dei parametri ottimali per le funzioni di score e di perdita. Le varianti di PVNet sono state addestrate utilizzando il dataset LINEMOD. Dalle sperimentazioni è emerso che:

- la configurazione migliore risulta essere quella con DSAC e con EPnP;
- più la strategia di generazione di dataset sintetici realizza dati vari e vicini alla realtà più risulta essere efficace;
- la rete risulta essere molto dipendente dai dati reali in fase di training;
- la giusta taratura dei parametri del modulo DSAC e della loss function può far raggiungere alla rete dei risultati molto buoni.

Contents

List of Figures

List of Tables

List of Acronyms

6DoF Six Degree of freedom

AI Artificial Intelligence

ANN Artificial Neural Network

BoB Bags of Boundaries

CNN Convolutional Neural Network

CFT Cascade Forest Template

CV Computer Vision

DL Deep Learning

DRN Dilated Residual Network

DPOD Dense Pose Object Detector

DR Domain Randomization

DSAC Differentiable Random Sample Consensus

EPnP Efficient Perspective-n-Point problem

FC Fully Connected Layer

FPS Farthest Point Sampling Algorithm

HTP Hierarchical Pose Trees

HSL Hue, Saturation and Lightness

KPD Keypoint Detector

KVN Keypoints Voting Network

LIDAR Laser Imaging Detection and Ranging

LMA Levenberg-Marquardt Algorithm

MCC Loss Mask Coordinate-Confidence Loss

ML Machine Learning

NN Neural Network

OK-POSE Object Keypoint-based Pose Estimation

P3P Perspective-3-Point problem

PCOF Perspectively Cumulated Orientation Feature

PnP Perspective-n-Point problem

PVNet Pixel-wise Voting Network

RANSAC Random Sample Consensus

ReLu Rectified Linear Unit

ResNet Residual Network

RGB Red, Green and Blue

RGB-D Red, Green and Blue with Depth

RMSE Root-Mean-Square Error

RoI Region of Interest

RPN Region Proposal Network

SITE Scale-Invariant Translation Estimation

YOLO You Only Look Once

1

Introduction

Computer Vision (CV) is a field of artificial intelligence that allows computers and systems to be able to extract information from images, videos and other visual input and react by performing actions or formulating signals based on this information. If Artificial Intelligence (AI) allows computers to "think", computer vision allows them to "see", observe and understand: in fact, it allows machines to be able to distinguish objects, their distance, whether they are moving and whether there is something wrong with an image. One of the main tasks in computer vision is *object pose estimation*, i.e. the detection and estimation of the position and orientation of an object: this task is fundamental in the fields of robotics, autonomous driving and augmented reality. It is a task that is particularly challenging due to the loss of depth when projecting a real-world 3D scene onto a 2D plane (this will be explained in detail in one of the following chapters). In recent years, thanks to the use of neural networks in pose estimation approaches, considerable progress has been made and very good results have been obtained even when using RGB images as input. This work considers *PVNet* [1], one of the most important pose estimation methods in the literature, and aims to carry out experiments in order to verify:

- if the addition of Differentiable Random Sample Consensus (DSAC) module [2] can improve the system;
- whether changing the Perspective-n-Point problem (PnP) [3] type affects performance and how much;
- how good the strategies for creating synthetic datasets can be;

- how much does a small amount of real images in the training dataset affect the performance;
- what are the most correct parameters for the loss and scoring functions of the system.

The tests described above were evaluated on the LINEMOD dataset [4], which is regarded as one of the standard benchmarks for estimating the 6D pose of an object. It is a dataset consisting of 15 different 3D objects without textures, but only the "cat" and "duck" objects were used in these experiments. To the real images of each object, synthetic images were placed side by side in order to have a good amount of data during training. The entire work is structured into chapters as follows:

- **chapter 2**: all the basics useful for understanding the tools used during the experiments are presented. From an initial overview of the theory of Image Formation, we then move on to an overview of the architecture of neural networks and finally the main estimation algorithms useful for solving the pose estimation problem;
- **chapter 3**: an overview of the methods used for pose estimation similar to PVNet, the network used in these experiments, is given; furthermore, they are categorised according to the approach used to solve the problem;
- **chapter 4**: the structure of the Neural Network (NN) used and the module added are explained in detail; finally, the modifications made to the code in order to be able to carry out the experiments are set out;
- **chapter 5**: the datasets and metrics used in the experiments are described, followed by a discussion of the results;
- **chapter 6**: conclusions and ideas on how to structure possible future experiments.

2

Basic Notions

Before getting to the heart of this thesis and the experiments carried out, it is necessary to recall some theoretical concepts of computer vision and neural networks. In this chapter, they will be set out:

- concepts and mathematical models of projective geometry underlying the representation of object poses in images
- basic concepts on the architectures and functioning of neural networks that have been used in experiments;
- estimation algorithms and their parameter evaluation process from model definition and input data.

2.1 Image formation background

The process of image formation consists of projecting, both geometrically and optically, points from the three-dimensional (3D) world into positions in the plane of a two-dimensional (2D) image. This process is divided into two parts:

1. in the first stage, depending on the camera model used in the imaging process, the geometry of the image is constructed; the projections of the points of the imaged scene in the image plane are then determined
2. in the second stage, the radiometry of the image is constructed: for each point projected onto the image, the brightness is measured as a function of illumination and surface properties.

The following paragraphs will recall concepts related to image geometry.

2.1.1 The pinhole camera model

When humans observe what is around them, the eye projects the 3D world onto a flat photosensitive surface inside the eye called the retina. Luminance, on the other hand, is measured by the rods and cones, which are photosensitive cells on the surface of the retina. Together, the two measurements constitute the human vision of the real world. In computer vision, the camera is a fundamentally important tool: it is the mechanism by which it is possible to "record" the world around us through photographs. The human eye and the camera are both based on the same optical principle: the *pinhole camera model*.

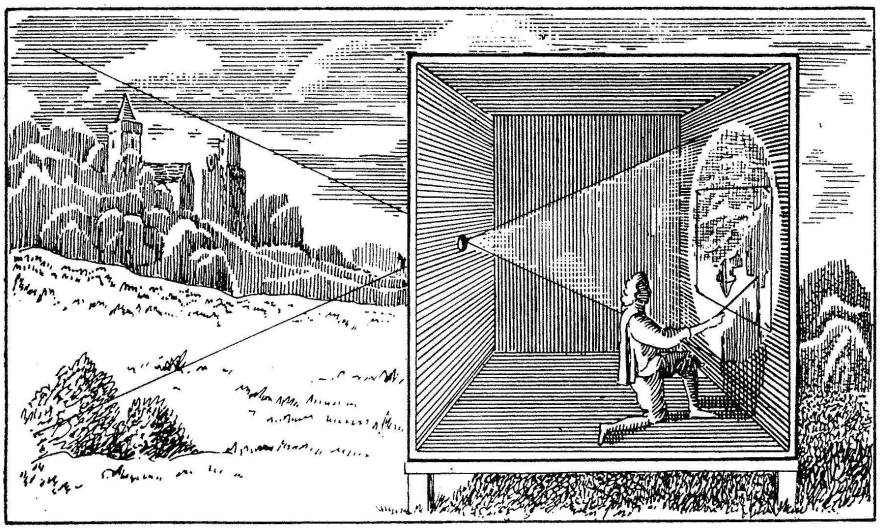

Figure 2.1: *Schematisation of a camera obscura. Source at this LINK.*

This model, also known as the *camera obscura*, is the simplest but at the same time the most accurate, allowing objects at any distance to be in focus. The idea behind the realisation of the model is very simple: place a barrier with a small aperture between the real object and a photosensitive surface, which can be a photographic film or a sensor; each point of the 3D object reflects numerous rays of light, the purpose of the infinite aperture is to select only one (or a few) of them so that a one-to-one mapping can be established between the points of the 3D object and the surface. The formal notation of the model is:

- the surface is called the image or retinal plane;
- the aperture is referred to as pinhole or camera center (O);
- the distance between image and pinhole is the focal distance (f);

- virtual image or virtual retinal plane is when the surface on which the image is generated is between the real object and the pinhole; The image and the virtual image of the object are to scale.

It can be seen that the projected space on the retinal plane:

- neither angles nor lengths are preserved;
- points that belong to lines in the real world are projected onto lines in the image (*collinearity invariance*);
- parallel lines in reality are projected as lines meeting at the horizon (mathematically they will never meet; they are said to meet at a point at infinity).

Euclidean geometry cannot handle points and lines at infinity, so projective geometry is used, which treats them as all other points and lines in space.

2.1.2 Projective geometry

The task of projective geometry is to preserve the collinearity of the points: given three points belonging to a line, although the line is changed, the transformed points still know that they are collinear. This operation is not linear so in output we will not have n-dimensional output, but rather $n+1$. The idea behind the transformation is to obtain an $n+1$-dimensional vector from one with n, where the last value of the vector contains the factor that multiplies all the other n coordinates. For example, take a 2D point $x = [x_1, x_2]$, its corresponding vectors in homogeneous coordinates will be $\tilde{x} = [sx_1, sx_2, s], \forall s \neq 0$. Given a vector $\tilde{x}$ in homogeneous coordinates, it is immediate to derive the corresponding 2D vector x:

$$\tilde{x} = [sx_1, sx_2, s] \Rightarrow x = \left[\frac{sx_1}{s}, \frac{sx_2}{s}\right] \tag{2.1}$$

s is called the scale factor in the case where the two homogeneous vectors represent the same 2D vector only at different scales, in notation:

$$\tilde{x} \sim \tilde{y} \iff \exists s \neq 0 : s\tilde{x} = \tilde{y}$$

the more $s \to 0$, the more we have to deal with a point that has a distance closer and closer to infinity.

2.1.3 Pinhole camera matrix

At this point, *projective geometry* allows us to construct the matrix underlying the pinhole model. Consider the camera obscura model, defined earlier, and assume that the retinal plane is to the left of the *aperture* (O) at a distance f (*focal distance*). The center of the reference system of the model will be O. The Z-axis of the reference system will be perpendicular to the wall with the aperture, while the X and Y axes will locate the plane of the wall.

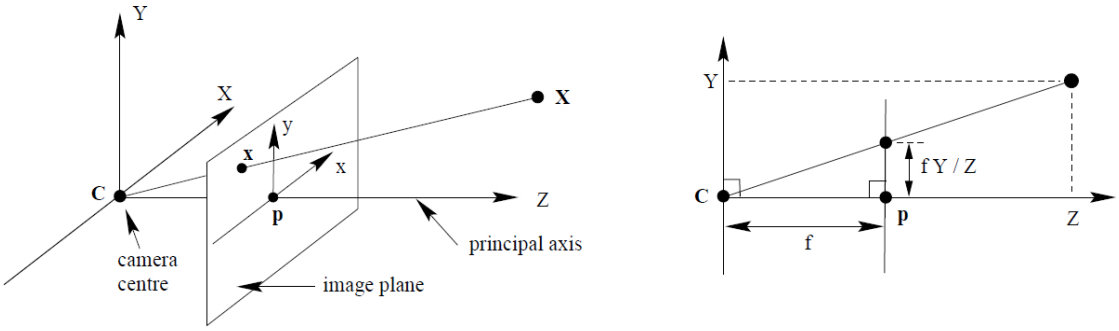

Figure 2.2: *Illustration of the pinhole camera model and its nomenclature. Source paper [5].*

Consequently, the center of the generated image will be at distance $-f$ along the Z-axis. If the real coordinates of a point $P = [X, Y, Z]$ are known, the coordinates of the point P' projected onto the plane of the image I can be calculated by means of the rule of similar triangles:

$$x = -f\left(\frac{X}{Z}\right), y = -f\left(\frac{Y}{Z}\right) \tag{2.2}$$

the minus sign in front of the coordinates indicates that the image generated by the camera obscura is *upside down*. In order to simplify the model, we use a virtual pinhole camera that generates a virtual plane placed on the Z-axis at distance f from the aperture. As a result, the coordinates will always be positive and free of the *reflection* effect:

$$x = f\left(\frac{X}{Z}\right), y = f\left(\frac{Y}{Z}\right) \tag{2.3}$$

Integrating the notions of geometry seen above, we can define the ideal

pinhole camera model:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.4}$$

The model just constructed is a geometric model: all coordinates and focal distance are calculated in meters. An additional transformation must be applied in order to transform the coordinates of the image plane from continuous to discrete. In fact, modern cameras use discrete sensors in order to capture images, and these images are measured in pixels. The mapping from continuous to discrete coordinates is as follows:

$$u = u_c + \frac{x}{w_p}, v = v_c + \frac{y}{h_p} \tag{2.5}$$

where:

- u_c and v_c are the parameters that determine the difference between the coordinates of the center of the pinhole plane and those at the upper left corner of the image, which, by convention, is the center of the system in pixel coordinates;
- h_p and w_p are the height and width of the discrete sensor cells, respectively;
- x and y are the previously defined coordinates.

The parameters $\alpha_u = \frac{f}{w_p}$ and $\alpha_v = \frac{f}{h_p}$ allow direct conversion with a single matrix multiplication from 3D scene point coordinates to pixel coordinates. Plugging the above into the ideal pinhole camera model, we obtain:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.6}$$

This gives the internal matrix of the camera.

2.1.4 Pose

In the context of Computer Vision, it is of fundamental importance to be able to identify the pose, that is, the translation and rotation of a rigid object with

respect to another. In order to be able to explain the concept more precisely, suppose we want to describe the pose of a rigid body with respect to a fixed world surrounding it: we must then define an object reference system (O) and one referring to the fixed world (W). The *translation component* (t) is a translation vector of dimension 3x1 that represents the translation along the 3 axes of one reference system with respect to the other.

$$t = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

Instead, depending on the domains in which it can be used, there are different representations for the *rotation component*:

- **Rotation matrix**: this is a matrix of size 3x3 that expresses the orientation of one frame of an object with respect to another;

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

- **Roll-Pitch-Yaw**: since a rigid body has at most 3 degrees of freedom, it is possible to describe the arbitrary orientation of an object by three independent quantities: *roll* (ϕ), *pitch* (θ) and *yaw* (ψ).

$$R\left(\phi, \theta, \psi\right)$$

  This type of notation is widely used in robotics because it is geometrically very easy to understand and only three values are needed; unfortunately, this simplicity prevents it from representing continuous values, and the final orientation depends on both the order in which the rotations are applied and whether they are applied on moving axes (*intrinsic rotations*) or fixed axes (*extrinsic rotations*);

- **Axis-Angle or Rotation Vector**: this type of representation uses a *unit vector* (u) that indicates the direction of the axis of rotation and an *angle* (θ) that reports how much the object is rotated with respect to its axis. If the vector and angle are multiplied with each other, the *rotation vector* (r) is obtained.

$$r = \left[r_x, r_y, r_z\right] = \left[u_x\theta, u_y\theta, u_z\theta\right]$$

  The disadvantages of the previous representation are overcome and transformed into strengths, however, it is difficult to match the physical orientation with the numerical values of the rotation vector. In addition, one must convert to another representation whenever one wants to apply a rotation directly to a 3D point;

- **Unit Quaternion**: this representation takes advantage of Axis-Angle encoding with 4 parameters; it is a very simple, stable and robust method; therefore, it is considered the best method to represent the orientation of

a rigid body.

$$r = \left[q_w, q_x, q_y, q_z\right]$$

Unit Quaternion does not require conversions in order to apply rotations directly to 3D points, they are more efficient than rotation matrices and do not suffer from the inability to uniquely define orientation as in Roll-Pitch-Yaw.

2.2 Deep learning background

Deep Learning (DL) is one of the most advanced branches of Machine Learning. It consists of a set of techniques based on Artificial Neural Network organised in several layers: each layer calculates values for the next one, in order to process the information more and more completely. With a sufficient amount of data, the system is able to learn the correct representation and solve machine learning problems without the need for data pre-processing, as is the case with traditional Machine Learning techniques. Although "technologically new", Deep Learning techniques have their roots in the past, it is only in the last ten years, thanks to their computational capacity and the use of large amounts of data, that their usefulness has been demonstrated in a wide range of applications, for example:

- image classification;
- language recognition and processing;
- autonomous driving;
- media and entertainment;
- security;
- medical diagnosis.

This section will explain the architectures and operations of the most commonly used neural networks.

2.2.1 Artificial neural network

An *Artificial Neural Network (ANN)* [6] is a Machine Learning algorithm that allows a computer to imitate the functioning of the human brain by learning

from experience and being able to predict an outcome from certain initial situations. This allows a machine to perform, in complete autonomy, tasks that were previously intended only for humans, such as driving a vehicle, answering a telephone or diagnosing a disease. The structure of an ANN can be represented by means of a directed graph, where the nodes are called *neurons*, while the arcs are the connections that link the neurons together and divide them into layers. The basic architecture of an Artificial Neural Network is called a *feed-forward network* and consists of three main layers:

1. **input level**: receives unprocessed input data, which can be any form of structured data such as images, text or numerical values;
2. **hidden levels**: are located between the input and output levels and are responsible for processing input data; to transform "stimuli" into a reaction, each neuron applies a function, called an *activation function*, to the weighted sum of input values in the neuron and its product becomes input to the following. The connection weights play a crucial role: they tell the neuron which signals are relevant and which are not. To function properly, none of the input signals must overwhelm the others: therefore normalising or standardising the dataset is used.
3. **output level**: produces the final results of the Neural Network's processing, such as classifications or numerical predictions.

Artificial Neural Network are a popular choice for various Machine Learning tasks because they offer numerous advantages, including:

- **versatility** in application, e.g., image and speech recognition, natural language processing, games, recommender systems, and many others;
- **adaptability** and learning to new information and/or changing environments;
- **non-linearity** in computation between input and output that allows ANNs to handle tasks impossible for linear models;
- **generalisation** of the data they have been trained on, which allows them to make predictions on cases never seen before, which is crucial for real-world applications;
- **transfer of learning**, i.e. reducing the amount of data required and calculations required for training by reusing pre-trained networks as a starting point for new tasks.

Despite having many advantages, researchers and engineers are constantly working to overcome certain limitations of this model such as requiring a large amount of data for training, computational complexity and *overfitting* (inability to generalise if the network is exposed to new data).

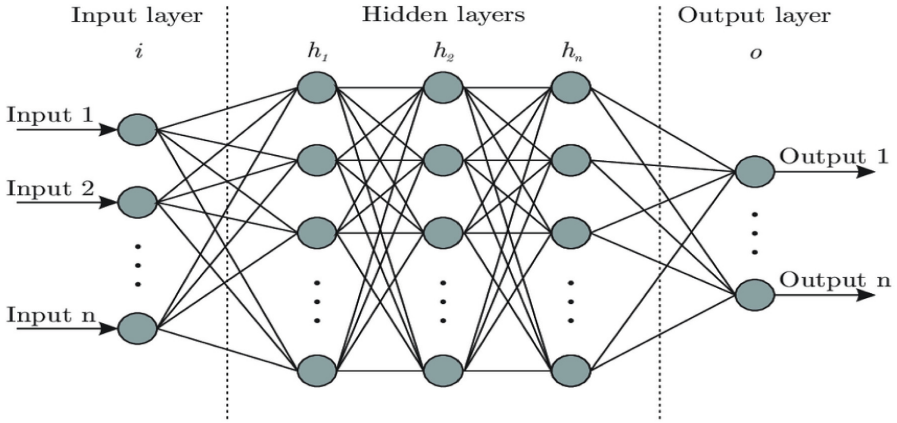

Figure 2.3: *Basic architecture of an ANN with n hidden layers. Source paper [7].*

2.2.2 Convolutional neural network

A *Convolutional Neural Network (CNN)* [8] is a particular feed-forward NN that is inspired by the organisation of the visual cortex: it is a network consisting of several stages specialised in different tasks, such as the extraction of shapes or other image features. In general, the functioning of a Convolutional Neural Network (CNN) is no different from any other feed-forward network, but the difference between the two types of network lies in the fact that within the CNNs are the *convolutional layers*. The latter extract features or characteristics of the images whose content is to be analysed through the use of filters: depending on the type used and the number, it is possible to identify features ranging from the simplest (contours, lines, colours) to entire objects. Therefore, a CNN works and classifies the image based on intrinsic elements of the image, not stopping at "the general information of the image" as is the case in classic feed-forward. The basic structure of a CNN is very simple: after an initial input layer, there is a sequence of convolutional layer, Rectified Linear Unit (ReLu) and pooling layer, up to the fully connected layer. The main blocks of this network will be analysed below:

- the **input layer** consists of a sequence of neurons capable of receiving the image information. At this level, the matrix of pixels representing the image to be processed will be passed. This means that the input will have three dimensions: a height, a width and a depth, which correspond to the three colours of the image in RGB format.

- the **convolutional layer** aims to detect features depicted in an image with high accuracy. It requires data collected from the input layer, a filter (or mask) and a feature map. During the convolution operation, the filter

(two-dimensional array of weights that determines the receptive field) is applied to an area of the image and a scalar product is calculated between the input pixels and the mask. This dot product is then inserted into an output matrix. Next, the filter moves one step, repeating the process until it has traversed the entire image. The final output of the operation is a smaller matrix of values representing the "featured image" known as a feature map or convoluted function. At the end of each convolution operation, a ReLu transformation is applied to the feature map to undo unhelpful values obtained in previous layers, introducing non-linearity into the model;

- the **pooling layer**, or also referred to as the sub-sampling layer, makes it possible to identify whether the feature being studied is present in the previous layer, and makes the image coarser, retaining the feature used by the convolutional layer. In other words, acting similarly to the previous layer, the pooling layer performs a reduction in the dimensionality of the input parameters;

- the **Fully Connected Layer (FC)** is the layer that actually performs the image classification. The pixel values of the input image are not directly connected to the output layer in partially connected layers. However, in the fully connected layer, each node in the output layer connects directly to a node in the previous layer. This layer performs the classification task based on the features extracted via the previous layers and their various filters. While convolutional and pooling layers tend to use ReLu functions, fully connected layers usually leverage a softmax activation function to classify inputs appropriately, producing a probability of 0 to 1.

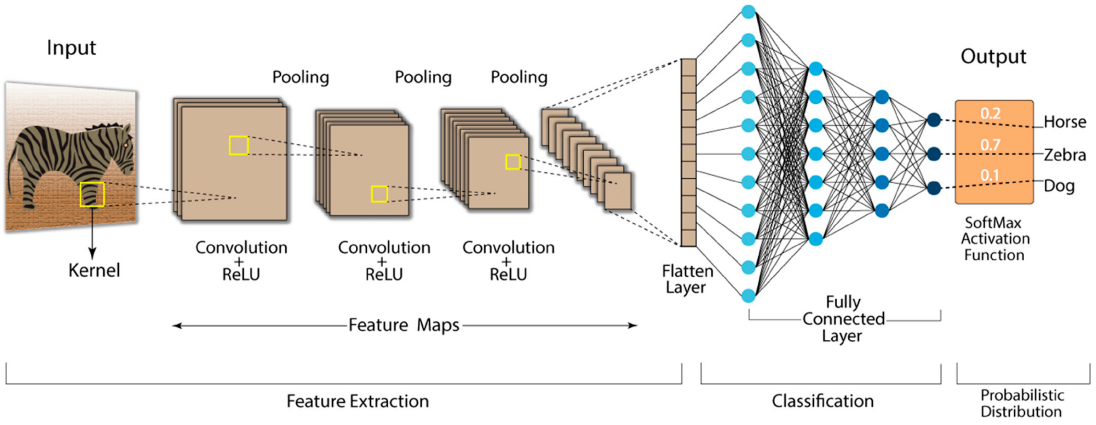

Figure 2.4: *Basic architecture of an CNN. Source paper [9].*

2.2.3 Residual network

Residual Network [10] are a special type of deep Neural Network mainly used in image processing. They stem from the intuition that if a network has

more layers, then it will progressively learn more and more complex features. [11] Therefore, more and more convolutional blocks began to be stacked, in the hope of getting better and better results; but, as the study by He et al. [10] showed, empirically there is a maximum threshold for depth with the traditional CNN: it has been shown that the best networks using convolutional layers and fully connected layers typically contain between 16 and 30 layers. in the same paper the authors conducted another experiment which compared the values of training and testing errors between a CNN with 20 layers and one with 56 are compared and it emerges that it is not overfitting, but rather the vanishing gradient problem. During training by backpropagation, the gradient assumes a value of 0 or is asymptotic to 0, thus preventing the network from updating: the weights of the layers near the input remain constant or update very slowly in contrast to the layers near the output. The ReLu activation module partly solved this problem, but the major breakthrough came with the introduction of *skip connections*. They make it possible to realise networks with multiple layers that perform at least as well as a shallow model [12]. Their structure is shown in the following figure:

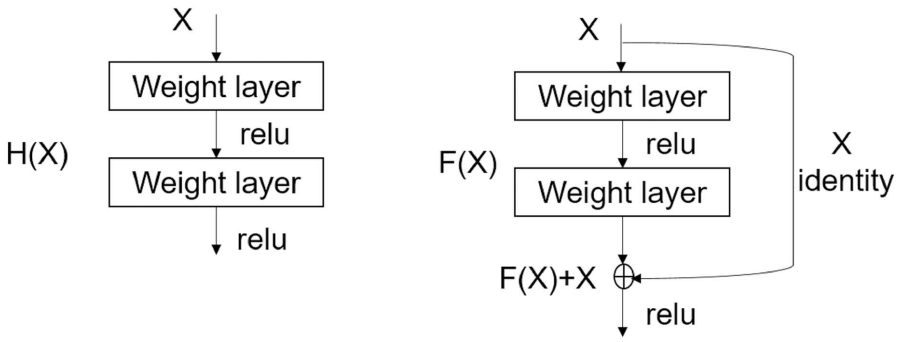

Figure 2.5: *The left shows the structure of a normal CNN, while the right shows that of ResNet with skip connections. Source paper [10].*

From what can be seen, the aim of the skip connection is to create an alternative path for the gradient to pass through during the back-propagation phase of the error: the *input* (X) is added to the result of the *non-linear transformations* (F) performed on X by the intermediate layers, producing the output:

$$H(X) = F(X, W) + X \tag{2.7}$$

Then a ReLu activation function will be applied to $H(X)$. The skip connection, also known as the *residual block*, is the fundamental element of the architecture

of residual networks. The name residual is due to the single block learning technique:

$$F(X, W) = H(X) - X \tag{2.8}$$

In general, it can be said that the residual block makes two considerations:

- $\mathbf{W = 0}$ *"If I don't learn anything new, at least I remember the past"*, i.e. if the kernels of the block levels do not highlight any features on the input X and produce feature maps with pixels of value 0, the residual block allows the output not to be cancelled and to move forward. In this case, the residual block tries to map its input (the identity) so as not to lose information, or rather, learns the residual:
$$F(X, W) = H(X) - X = X - X = 0 \tag{2.9}$$
- $\mathbf{W \neq 0}$ *"If I learn something, the past helps me anyway"*, i.e. when the layers manage to extract non-empty feature maps, the input is "perturbed" with what is extracted from the layers of the block, so as to increase its accuracy; in this case the residual block learns to map the perturbation, or rather, its residual.

2.2.4 Transformer

Transformer neural networks [13] are one of the most promising and innovative technologies in the field of Machine Learning (ML). Initially developed to improve *machine translation*, in recent years they have found applications in many other fields, including realistic image generation [14], natural language understanding [15] and medical diagnosis. The ability to handle large amounts of data and to learn autonomously have determined their success. The operation of this particular Neural Network is based on a building block called a *transformer*, which enables it to process and understand data more efficiently than traditional methods. It consists of two main components:

- the **attention module** allows the NN to "focus" only on the data needed to solve a given problem, ignoring superfluous data. This makes it possible to handle large amounts of data more efficiently and to obtain more precise results;
- the **encoding module** is responsible for transforming the data into a format that the Neural Network can understand and use to make decisions. For example, in the case of machine translation, the encoding module transforms the words of one language into a format that the NN can use to generate the translation into the other language.

Transformers have many advantages over other neural networks, including:

- the efficiency of transformers in processing input in parallel, which makes them much more efficient than sequential processing;
- the processing of sequences of arbitrary length without the need for truncation techniques;
- the capture of long-term context information even without the use of sequential processing;
- the superior performance compared to previous architectures in many areas of natural language processing.

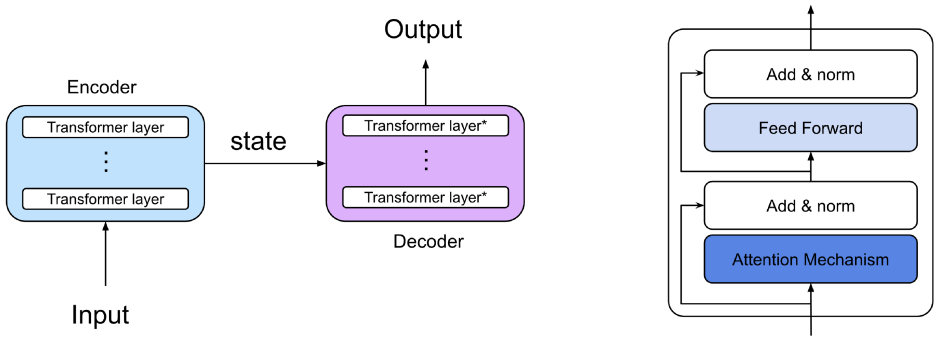

Figure 2.6: *On the left there is the Encoder-Decoder architecture, in which the input sequence is first encoded into a state vector, which is then used to decode the output sequence. On the right, there is shown the transformer layer, the encoding and decoding modules were constructed using stacks of transformer layers. Source at this LINK.*

2.3 Estimation background

Given the availability of noisy data, most computer vision problems are related to the estimation of models. Therefore, in order to better understand the work carried out in this thesis, it is necessary to expose the functioning of the main parametric estimation algorithms widely used in this field: *PnP* and *Random Sample Consensus (RANSAC)*.

2.3.1 Perspective-n-Point problem

In the early 1980s, the scholar Fischler proposed the *Perspective-n-Point problem (PnP)* [3], which is used to estimate the position and orientation of a camera from a set of 3D points and the corresponding 2D projections belonging to an

image taken by the camera itself. PnP is a fundamental problem not only for computer vision but also in robotics: in fact, it is fundamental for *visual odometry*, i.e. the process of determining the position and orientation of a robot by analysing the images from its associated camera. As seen in the section 2.1.4, the camera pose has Six Degree of freedom (6DoF), given by the 3D rotation and 3D translation of the camera with respect to the world reference system; therefore, in order to solve a PnP problem, information from at least 3 pairs of corresponding points is required. There are many algorithms that solve this type of problem and can be divided into two categories based on how many point pairs are needed as input:

- **Perspective-3-Point problem (P3P)** [16] is the algorithm for solving the PnP problem with only 3 point correspondences and returns up to four geometrically possible solutions. Its first formulation of the algorithm dates back to 1841, to date the most recent variant by Gao et al. can be found within *OpenCV* in the `calib3d` module under the function `SolvePnP`. Let be:
  - P the projection center of the camera;
  - A, B and C the selected points of the 3D world;
  - u, v and w the points in the image corresponding to the real
  - let also be $X = |PA|$, $Y = |PB|$, $Z = |PC|$, $\alpha = \angle BPC$, $\beta = \angle APC$, $\gamma = \angle APB$, $p = 2\cos\alpha$, $q = 2\cos\beta$, $r = 2\cos\gamma$, $a' = |AB|$, $b' = |BC|$, $c' = |AC|$

  According to the definitions just given, three triangles PBC, PAC and PAB are generated from which we obtain the system of equations sufficient for P3P to work out the solutions:

$$\begin{cases} Y^2 + Z^2 - YZp - b'^2 &= 0 \\ Z^2 + X^2 - XZq - c'^2 &= 0 \\ X^2 + Y^2 - XYr - a'^2 &= 0 \end{cases} \tag{2.10}$$

  Solving the system yields four possible solutions to determine the rotation and translation of the camera.

- **Efficient Perspective-n-Point problem (EPnP)** [17] is the algorithm that solves the $n > 3$ case of the PnP problem. It was developed in 2008 by Lepetit et al. and its code is available in the camera calibration and 3D reconstruction module at the `SolvePnP` function of *OpenCV*. The method is based on the idea that each of the n points can be expressed as a weighted average of 4 virtual control points: the coordinates of these 4 points will be the unknowns of the problem and that will determine the camera pose. Let us proceed with the formal definition of the method: first, the n points referring to the 3D world p_i^w and those corresponding to the image p_i^c must be defined in homogeneous form as the weighted average of the 4 control

points, respectively c_j^w and c_j^c:

$$p_i^w = \sum_{j=1}^{4} \alpha_{ij} c_j^w, p_i^c = \sum_{j=1}^{4} \alpha_{ij} c_c^w \tag{2.11}$$

$$\sum_{j=1}^{4} \alpha_{ij} = 1$$

From the above considerations, we can write the derivation of the image reference points as:

$$s_i p_i^{img} = K \sum_{j=1}^{4} \alpha_{ij} c_j^c \tag{2.12}$$

$p_i^{img} = \left[u^i, v^i, 1\right]$ are the vectors that represent the image reference points in pixel coordinates; while $c_j^c = \left[x_j^c, y_j^c, z_j^c\right]$ are the vectors that homogeneously represent the image control points. For each reference point we will then have these two equations obtained from the previous one:

$$\sum_{j=1}^{4} \alpha_{ij} f_x x_j^c + \alpha_{ij}(u_0 - u_i) z_j^c = 0 \tag{2.13}$$

$$\sum_{j=1}^{4} \alpha_{ij} f_y y_j^c + \alpha_{ij}(v_0 - v_i) z_j^c = 0 \tag{2.14}$$

By setting all the equations obtained in an M-matrix we obtain the system $Mx = 0$, to find the solution vector x containing the four control points we simply calculate the kernel of the M-matrix like this:

$$x = \sum_{i=1}^{N} \beta_i v_i \tag{2.15}$$

N is the number of null singular values in M and has a range from 0 to 4, vi corresponds to the right singular vector of M. bi are the calculated coefficients that are then refined with the Gauss-Newton algorithm. Finally, the rotation and translation that minimises the projection error of the real points p_i^w in the corresponding image points p_i^c are calculated. The solution has a computational complexity of $O(n)$.

2.3.2 RANSAC

Random Sample Consensus (RANSAC) [3] is a non-deterministic algorithm developed by Fischler and Bolles and published in SRI International in 1981.

It is based on the random selection of the generating elements of the iterative model for estimating the parameters of a model where the data set is strongly biased by the presence of many outliers, i.e. values that are distant from other available observations. Iteratively, the RANSAC algorithm alternates between an initial *hypothesis-generating phase* and a *hypothesis-estimating phase*; in order to better understand what has just been said, the pseudocode of the algorithm is given below:

1. it randomly selects a subset of points (*hypothetical inliers*) that can determine the parameters of the model;
2. the model parameters are calculated to fit the hypothetical inliers found previously;
3. all other data are tested on the calculated model, the points that fit within a tolerance defined by the loss function are included in the consensus set;
4. if the ratio of the number of points in the newly found set to the total points in the dataset is greater than the threshold to be reached, the model is found to be good and the parameters are recalculated using the entire consensus set and the algorithm terminates;
5. otherwise, the previous four points are repeated a maximum of N times.

For the probability p that at least one of the hypothetical inliers does not include an outlier to be high, the number of iterations N must be high. The relationship between these two values is as follows:

$$1 - p = (1 - u^m)^N \tag{2.16}$$

Where:

- u represents the probability that the selected point is an inlier;
- $v = 1 - u$ represents the probability of selecting an outlier;
- m is the minimum number of points required to solve the problem.

The formula determining the relationship between N and p will then be:

$$N = \frac{log(1 - p)}{log(1 - (1 - v)^m)} \tag{2.17}$$

3

Survey on image-based 6DoF pose estimation

In this chapter, the problem of calculating the pose of an object with 6DoF will be analysed. Next, the most recent and important papers concerning this topic will be reported, which have been chosen with the following criteria:

- publication date after 2016;
- number of citations greater than 5;
- modernity of the technologies used;
- input with only a single RGB image and possibly a 3D model of the object to be identified, as methods requiring multiple input images, Red, Green and Blue with Depth (RGB-D) images or requiring data from sensors (e.g. Laser Imaging Detection and Ranging (LIDAR)) are more complex to train and calibrate as well as being space-intensive models that are difficult to use;
- results obtained.

During the overview, the methods will be classified and analysed, highlighting their strengths and disadvantages, and an attempt will be made to answer questions such as: "*Does the method need a pose refinement phase?*", "*How large does the dataset need to be for it to produce interesting results?*", "*Is synthetically generated data needed for training?*", "*Can it work in real-time?*" and "*Is the accuracy appropriate for its specific application domain?*"

3.1 Problem definition and methods classification

The *6DoF pose estimation* can be considered as one of the most important tasks in many computer vision applications, such as robotics, autonomous driving, and virtual/augmented reality applications. It consists in determining the 3D rotation and translation of an object, whose shape is known a priori, with respect to the camera, through the details observable from the input image. The pose identified in the process must be expressed in one of the notations described in section 2.1.4. The problem of pose estimation is not complex to explain, but it is complicated to obtain satisfactory solutions because one can find self-occlusions or symmetries of the object that do not allow it to be unambiguously identified, unfavourable image illumination conditions and occlusions with other objects in the scene. Introducing a phase in which the portion of the image containing the object is recognised particularly helps in the estimation of its position. Early methods of pose estimation are based on geometric approaches that utilise manually annotated local features to calculate correspondences between 3D models and objects in the 2D image. *Geometric methods* are more time-consuming, with inaccurate results and may even fail when objects are geometrically complex or lack texture because local features are not easily identified. As an alternative to these, *template-based methods* exploit a dataset with 2D representations of the object from different viewpoints to search for that single representation that matches the object in the input image. The latter methods are very susceptible to occlusions and variations in illumination and, in addition, the recognition time for objects without textures increases considerably due to the large number of comparisons. With the renaissance of *deep learning*, brought about by the increase in computing power and the availability of handling large amounts of data, traditional methods have been revamped and made more efficient and performant: the idea behind this "renewal" is to employ convolutional neural networks for learning a mapping function between the 3D and 2D coordinates of the position of the object to be detected. This category of methods can be divided into:

- **Bounding box prediction and PnP algorithm-based models**, which uses a CNN that has the task of estimating the 2D projections of the corners of the 3D bounding box and then the PnP has the task of determining the position from the correspondences of the 2D features of the image and the 3D points of the CAD model;
- **Classification-based models**, consisting of a CNN that performs the clas-

sification in a single step;

- **Regression-based models**, consisting of a CNN that performs regression in a single step.

The last two categories mentioned can achieve high levels of accuracy in real cases, following training with large amounts of data. There is no single methodology to solve the problem of 6DoF pose estimation and the approach to be used varies greatly depending on the field in which one is working. Even now, many researchers apply themselves in order to find more accurate and faster solutions.

3.2 Features-based methods

Feature-based method is based on the extraction of local features from Region of Interest (RoI) or from all image pixels. The most commonly used features are, for example, keypoints, contours, line intersections or grey values. By subsequently comparing the features found in the image with those obtained from the 3D model of the object, 2D-3D correspondences are obtained. The entire process can be divided into *two steps*: the first one extracts the local features from the image and compares them with the keypoints extracted from the 3D model, then the second step calculates the 6D position by geometrically solving the correspondence problem between the 2D and 3D values obtained from the previous step. These techniques mix traditional computer vision strategies with the latest neural networks: CNNs are employed at different points in the pipeline in order to improve the performance and accuracy of the system. The main advantage of these techniques is speed and robustness to occlusions between objects and cluttered scenes, while the disadvantages are:

- symmetrical objects or lacking well-defined textures for feature calculation;
- the accuracy of the calculated position depends on the quality of the extracted keypoints;
- the 6D position identified by this type of technique requires an additional operation to be refined to obtain the final position.

This type of methods can be a good solution when the object to be identified has an easily recognisable silhouette and accurate keypoints. Although the two-stage procedure with position refinement is very time-consuming, some

methods can be performed in real-time or near real-time. In order to train and test the performance of strategies of this type, datasets provided by the literature (LINEMOD, PASCAL 3D+, KITTI) or created ad hoc are used.

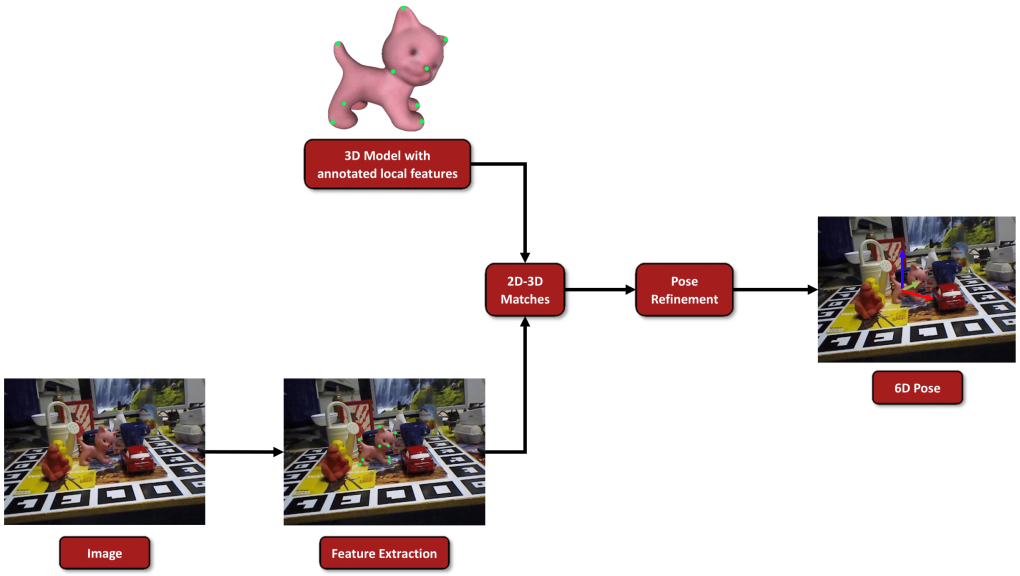

Figure 3.1: *Outline on the general functioning of the Feature-based methods.*

In the literature we can find several solutions belonging to this category, the most important of which include:

- **those that implement CNNs to extract features and a shape fitting algorithm to calculate the final position.** In [18] the authors construct a network with a pipeline consisting of object detection, keypoint location and finally pose refinement. Peng et al. [1] construct a particular CNN, called PVNet, which was very successful. Pixel-wise Voting Network succeeds in calculating 2D-3D correspondences with excellent results by regressing pixel-wise vectors to keypoints. For each keypoint, a spatial distribution is thus obtained, which will be the input for the PnP; at the end of the process, we will have the final position identified. Its performance is very good and robust even in the event of occlusions, and the network can also operate in real-time. Several studies have focused on improving this network: Zhu et al. [19] added Atrous Spatial Pyramid Pooling and Distance-Filtered PVNet to improve occlusion cases, while You et al. [20] improved performance by including a system on PVNet of projection loss and a discriminative refinement network;
- **those based on multi-stage pipelines.** Zaoh et al.'s network [21] exploits YOLOv3 to identify the object; after selecting keypoints in the object, a ResNet101-based Keypoint Detector (KPD) is trained. Finally, PnP finds the position of the object thanks to the matches of the previously found points;

- **those that maintain a pipeline similar to the previous cases exploit a CNN for both object detection and key point estimation and subsequently use geometric algorithms to refine the final result.** Zhao et al. [22] implemented an end-to-end ResNet-based network trained to recognise viewpoints consistent with the geometry and semantics of the image. Also by the same authors, Object Keypoint-based Pose Estimation (OK-POSE) is introduced in [23], which recognises 3D keypoints due to relative transformations between image pairs compared to classical labelling and CAD models;
- **those that exploit synthetic data for training.** A two-stage pipeline consisting of 2 CNNs was proposed by Nath Kundu et al. [24]. After learning how to extract the invariant position of local descriptors from the image, the first CNN takes care of extracting the keypoints; the second provides the final position, after reworking the information from various correspondence maps;
- **those dealing with complex objects.** Chen et al. [25] propose a method consisting of three steps: object detection, feature detection and pose estimation. This method is effective with metallic, shiny and untextured objects.

3.3 Template-based methods

This type of method is organised in such a way as to have an *offline stage* and an *online stage*: in the former, the aim is to construct a dataset consisting of various templates obtained from the 3D model of the object to be identified, while in the latter, the actual procedure for establishing the 6D position takes place. The construction of the dataset is fundamental in order to obtain good results; it is composed of a series of synthetic renderings, obtained by changing position and orientation, so as to have different points of view distributed on an imaginary sphere having the 3D object model as its centre. In the position estimation phase, the input image is compared with each synthetic image contained in the dataset thanks to a sliding window algorithm. A similarity value will determine the best match, which will be returned as the output of the process. The strengths of this category are:

- very good handling in the case of objects without textures;
- very good accuracy in the case of an exhaustively generated dataset.

On the other hand, its weaknesses are:

- the high sensitivity to variations in brightness and occlusions between objects;
- the need to strike a balance between execution speed and accuracy, both of which depend on the number of templates in the dataset; speed is inversely proportional to the number of elements in the dataset, while accuracy is directly proportional. Many CNN-based approaches modify the cost functions in order to solve this problem efficiently.

As mentioned earlier, this category of methods can achieve high accuracy if it has a *large dataset* at the expense of execution time, which is why it is rare for these methods to work in *real-time*. Due to the use of customised datasets, pose refinement is not necessary, however, the methodology employed may sometimes require it. In some cases, CNNs have been used to solve the problem.

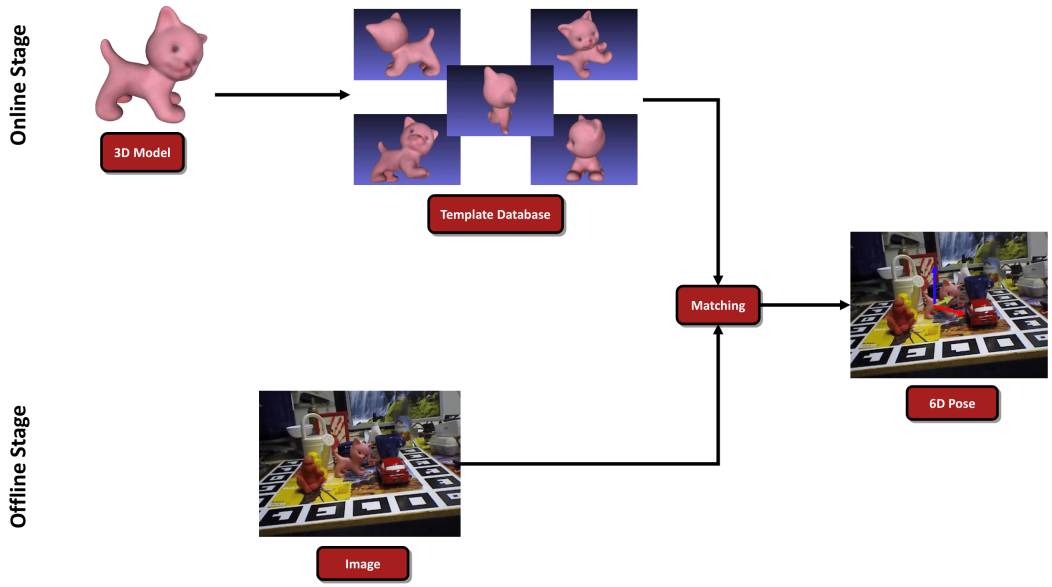

Figure 3.2: *Outline on the general functioning of the Template-based methods.*

In the literature we can find several solutions belonging to this category, the most important of which include:

- **older ones that implement geometric approaches.** The method presented in paper [26] exploits colour transformation and vectorisation to achieve more compact representations and a better calculation of correspondence. Instead of proposing a method based on the computation of poses given known instances, the authors of [27] introduce Bags of Boundaries (BoB), which only allows the identification of matches on a summary of edges and can therefore estimate the poses of unknown instances. Exploiting edges, Ulrich et al. [28] also estimate a discrete pose that is then refined by a 2D match with edge features; instead, the Levenberg-Marquardt Algorithm (LMA) [29] allows the calculation of 3D camera matches;

- **those that focus on reducing execution time.** In the paper [30] by Konishi et al., the Perspectively Cumulated Orientation Feature (PCOF) is proposed to handle a specific range of object poses; while Hierarchical Pose Trees (HTP) are based on grouping the poses of 3D objects and reducing the resolution of the models;
- **those that handle cases where objects are textureless.** Munoz et al. [31] exploit the coarse position information coming from a detector and make matches with the object edges. The same authors in [32] use the Cascade Forest Template (CFT): it allows the mismatch between the initial layout and the current layout to be learnt through the use of regression forests for each template;
- **those used in tracking**: These strategies are used for tracking initialisation and pose restoration following occlusions or exit from the camera viewpoint. The paper [33] presents a new segmentation strategy based on a local colour histogram;
- **those that focus on providing the exact position of symmetrical objects**: Corona et al. [34] realise a special loss function to handle this particular case;
- **those that implement neural networks to improve performance**: the network (usually a CNN) is given as input an RGB image and a depth map for each viewpoint and outputs the 6D position. In the paper [35], the auto-encoder denoising, a special type of NN that learns the representations of rendered 3D objects, is implemented. A cross-domain adaptation approach, using CaffeNet for both the off-line and on-line stages, was used by the authors of the paper [36].

3.4 Direct prediction or Learning-based methods

DL-based method is based on convolutional neural networks. Although it requires a large amount of labelled data in the training phase, it achieves very good results for estimating the 3D position and rotation of the target object. This type of methods can be divided into *one-stage* or *two-stage* depending on whether or not they require a "refinement" phase of the pose parameters using the PnP algorithm. It is precisely because of this last stage that two-stage CNNs perform better than one-stage CNNs in the case of small objects and multiple objects. However, PnP suffers from cases where point correspondences are affected by occlusions.

We do not always have enough real data to complete the network training phase, so we often resort to the *Domain Randomization technique*: this strategy consists of printing different positions of the 3D object model on a real background

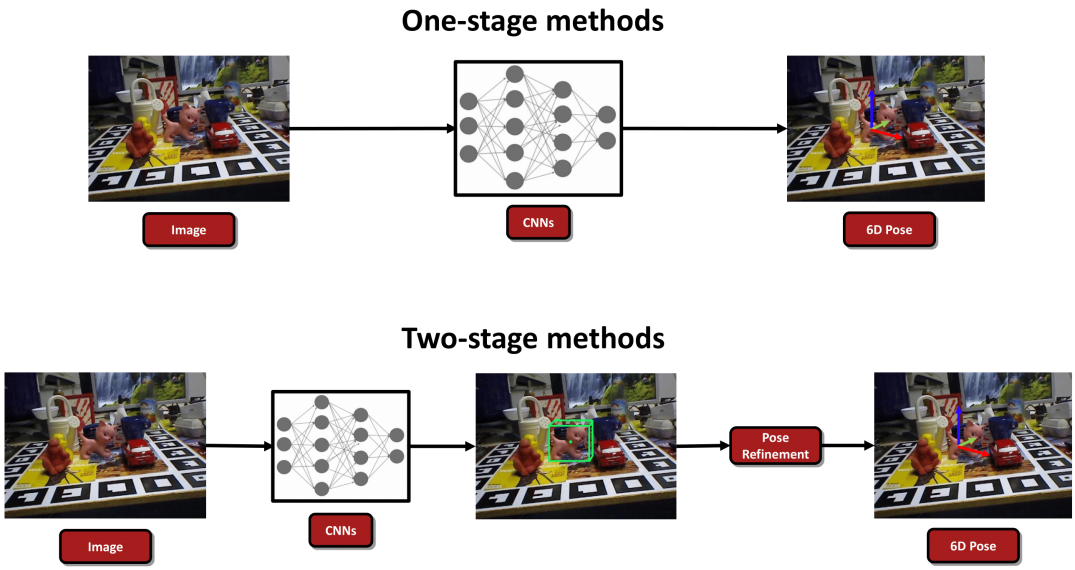

Figure 3.3: *Difference in operation between one-stage and two-stage methods.*

and then applying different augmentation techniques, e.g. variable illumination, contrast, blurring and occlusion using small monochrome patches. This technique is very useful for increasing accuracy, however the synthetic generation of occlusions does not help the system much in this main challenge of pose calculation. The advantages of these strategies are:

- powerful and excellent results;
- excellent performance even with rich backgrounds and partially occluded objects.

On the other hand, the problems that plague these methodologies are:

- the training time of the method is long;
- in the case of many large occlusions in the input images, they are not very robust;
- in some situations they are plagued by *overfitting*, i.e. the inability to generalise the problem when faced with new input data.

In recent years, this category has become very popular due to the increase in computing power and the possibility of processing and handling large amounts of data for system training. It can be divided into three main categories of methods:

- **Bounding box prediction and PnP algorithm-based methods** (*see section 3.4.1*);

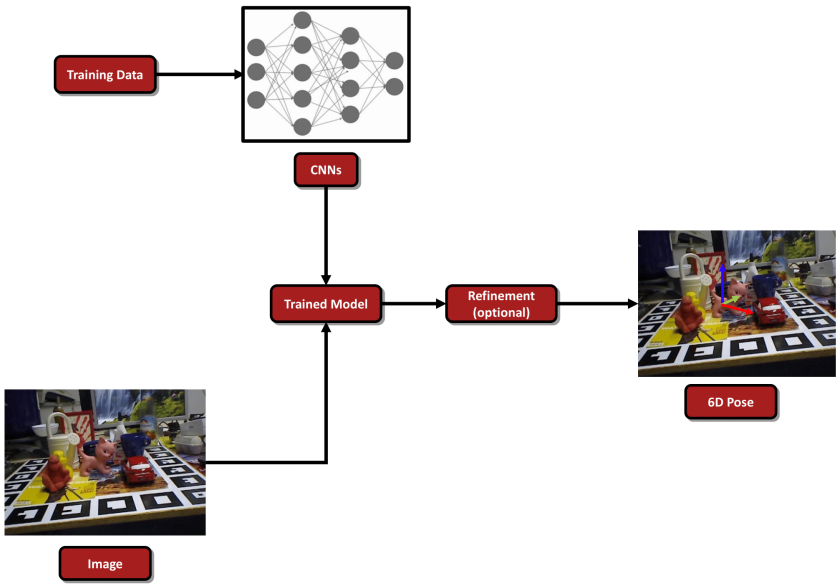

Figure 3.4: *Outline on the general functioning of the Learning-based methods.*

- **Classification-based methods** (*see section 3.4.2*);
- **Regression-based methods** (*see section 3.4.3*).

In the sections indicated above, their operation and the main recent examples that can be found in the literature will be explained.

3.4.1 Bounding box prediction and PnP algorithm-based methods

The general structure of the methods belonging to this subcategory has two-stages: in the first phase we have the use of a CNN to recognise the object and the regression of the projections of the identified 3D keypoints on the image takes place; in the second phase, thanks to the PnP algorithm, the actual pose of the object is calculated. In order to achieve good accuracy in the coordinates of the bounding boxes, additional pre-processing with long manual annotations is required. Although the pipeline is multi-stage and long, some strategies can be executed in real-time. The most recent strategies found in the literature differ only in step one, i.e. how the correspondences between 3D and 2D points are calculated to be "fed" to the PnP algorithm. Some of the most important methods include:

- the method proposed by Rad and Lepetit in [37] is called **BB8**. It consists of a series of CNNs: the first deals with semantic segmentation, the second estimates the eight vertices of the bounding box projections and finally the third, following the PnP algorithm, performs a refinement on the pose of the identified object. This method has been so successful that it has been the subject of further studies to make it more robust in cases where the occlusions are very opaque [38] and to improve its performance by replacing the use of the PnP with the Collinear Equation Layer [39] or the Bounding Box Equation [40];

- **those based on the ideas of the You Only Look Once (YOLO) [41] and BB8 networks.** YOLO6D is a fully convolutional network developed by Tekin et al. [42]: it performs a regression on the bounding box projections in the image, like BB8; it also provides precise and accurate results without refinement and without suffering from pose discretization like the SSD-6D method [43];

- **those that, in order to provide more robust results in the case of occlusions, do not consider the object as a global entity and do not calculate a unique pose.** An innovative 6D pose estimation framework was introduced by Hu et al. [44]: it is based on the segmentation of the visible parts of the object and these influence the local pose predictor in the 2D keypoint estimation process. In the paper by Li et al. [45], it is explained how CDPN can provide robust and accurate results of occluded objects or objects without textures, by calculating translation and rotation separately: the former is estimated by means of Scale-Invariant Translation Estimation (SITE), the latter by means of object-level coordinate estimation and a Mask Coordinate-Confidence Loss (MCC Loss). Pix2Pose, proposed by Park et al. [46], is an auto-encoder network that predicts the 3D coordinates of the object for each of its pixels and its error; the values obtained were used at several stages of the process to estimate 2D-3D correspondences and the final pose. To handle cases of occlusion and object symmetries, the method uses a new loss function called transformer loss. Dense Pose Object Detector (DPOD) is an encoder-decoder network, devised by Zakharov et al. [47], which performs a regression of the mask and 2D-3D correspondences. The results of the network are refined through the use of a CNN;

- **those that address both occlusion cases and the lack of real labelled images.** In their paper, by exploiting the Domain Randomization (DR) strategy, Li et al. [48] implement a robust 6DoF pose estimation approach: a first network identifies the pixels belonging to the target object and, subsequently, a Self-supervised Siamese Pose Network returns the coordinates and segmentation information;

- **those that can be performed in real-time.** In the paper [49], an end-to-end method is proposed, which is optimal in cases of occlusion between objects and where the object is devoid of textures, and which exploits a CNN for the estimation of 2D-3D correspondences. The authors of the paper [50] focused on robotics and implemented a client-server architecture for

keypoint recognition and pose estimation, basing it on YOLOv3. TQ-NET, developed by Liu et al. [51], is a network that receives the object location and its bounding box, previously identified by an algorithm, and predicts a translation vector T and quaternion Q, which will be converted into a rotation matrix R. TQ-NET may be a very simple network, but its strengths lie in the fact that it provides accurate, real-time results. Finally, DSC-PoseNet by Yang et al. [52] is the most recent method that exploits 2D bounding boxes to estimate pose: it works well with both real and synthetic data and, through differential rendering, calculates the pose of objects.

Studies are evaluated on custom datasets generated specifically for the case or datasets that are made available in the literature can be used to measure the accuracy of the estimation process, the most common of which include: LINEMOD, Occluded LINEMOD, T-LESS, YCB-Video and ACCV.

3.4.2 Classification-based methods

This sub-category of methods treats the 6D position estimation problem as a *single-stage*, *discrete-pose classification problem*: a CNN calculates the probability distribution in the pose space and subsequently, analysing it with the information from the 3D model, derives the position and 3D rotation of the target object. Despite being single-stage and despite the fact that they rarely require pre-processing and/or pose refinement steps, it is difficult for these strategies to work in real-time. The most important methods include:

- **SSD-6D** is a method in which the authors of the paper [43] modified the SSD detection framework [53] so that it can recognise detections and rotations in three dimensions. After a NN has detected the target object in an RGB photo and returned its 2D bounding box, each instance of the output is associated with a set of the most probable 6D poses; the 3D rotation is decomposed into discrete views and plane rotations and treated as a classical classification problem by the network. Two datasets were used to train the network: a real one for bounding box detection and a synthetic one for rotations;

- **those that are only trained on synthetic datasets.** The NN, designed by Su et al. [54], was trained with synthetic 3D renderings of objects superimposed on real images and is also able to work in real situations with real objects. The method proposed by the paper [55] has a modular architecture consisting of a detector and a viewpoint estimator. It is robust because it combines a CNN with a 3D pose estimator based on high-resolution instances. The output provided is not a 6DoF pose, it needs PnP and refinement steps;

- **those that are only trained on real datasets.** In the paper [56] GS3D demonstrates that the pose estimation problem can also be solved using only real data: it consists of a Faster R-CNN detector which, starting from the input image and the 2D bounding box parameters and classifies the rotation; a basic cuboid, called guidance, is created and then projected onto the image plane thanks to the newly obtained rotation combined with the 2D bounding box and other knowledge of the scenario. 3D Subnet exploits the idea of guidance by refining it. 6D-VNet by Zou et al. [57] is a network that is used in the context of autonomous driving;

- **those that do not separate the object detection and pose estimation phases on two different networks.** Poirson et al. [58] devised a very fast single shot detector method that does not use image resampling and uses convolutions to identify the object and estimate its position. In the paper by Mousavian et al. [59], a method consisting of an extended detector that, by training a CNN, performs a regression of the object's size and orientation is presented. This data is then combined with geometric constraints in order to estimate the translation and 3D bounding box. The network proposed by Xu et al. [60] consists of two parts: the first, thanks to the Region Proposal Network (RPN), allows the 2D region proposal to be generated, while the second deals with the simultaneous prediction of the position, orientation and size of the object and finally returns the 3D pose.

Normally, studies of this type are mainly evaluated on these datasets: PASCAL 3D+ and KITTI.

3.4.3 Regression-based methods

In this subcategory of methods, 6D pose estimation is considered as a *regression problem*: they use a CNN to determine a position estimate and then regress the object's 6D pose parameters directly from the input image. These strategies produce the output in a single-stage, but often require the introduction of a preliminary object detection stage to facilitate and speed up the process. They are among the best-known and best-performing systems, requiring neither preprocessing nor post-processing of data because they derive the 6D pose through a single-stage pipeline. They are proposed as end-to-end networks that can be trained and executed in real-time to overcome the problem that they are time and computationally intensive methods. Some of the most important methods include:

- **PoseCNN**, designed by Xiang et al. [61], appears to be among the best performing methods for solving the pose estimation problem using only RGB images as input. It consists of a CNN that performs object segmentation, rotation and camera distance estimation in two steps. At this point, feature

maps of different resolutions are obtained from the input image, which the network reprocesses in 3D translation and 3D rotation. The limitations of this method are the need for a refinement step and the impossibility of analysing images in which the target object is present several times. In [62], a method that focuses on the rotation between the objects and the camera is presented, exploiting a modified version of VGG-M Network that consists of a feature network and a pose network;

- **Deep-6DPose**, presented in paper [63] by Do et al., is an end-to-end network consisting of RPNs (to identify regions of interest) and Mask-RCNs;

- **those being developed in the context of autonomous driving.** Hara et al. [64] propose three variants of approaches that consider the rotation of objects viewed from the side to the centre of the input image. The first is based on a discretized process, while the other two are similar and differ only in the loss function. In contrast, MonoPSR is a method, implemented by Ku et al. [65], that estimates the 6D position of the object using hints and shape reconstruction. Rambach et al. [66] modified the architecture of PoseNet [67] and added a new loss function to facilitate training. This new network is able to derive the 6D pose of an object by means of a synthetic image with enhanced object edges from its 3D rendering;

- **those that are first trained on synthetic datasets and then validated on real data.** In the paper [68], the authors implemented a network consisting of two cascading components: a Dilated Residual Network (DRN) that generates the segmentation mask and a pose interpreter network that takes the results of the first component as input and analyses them together with the image;

- **those that turn out to be the most recent and still little known.** Assuming that the objects were rigid and that their 3D model was available, Hu et al. [69] constructed a network that, starting from the 2D-3D correspondences of each keypoint, performs a direct position regression. It consists of a local feature extractor, a feature aggregation module and a global inference module. The system described by the paper [70] consists of two CNN networks: one for segmentation and one for pose estimation; it is designed to calculate both mask and pose even when training data is scarce. The paper by Liu et al. [71] focuses on the use of triplets: after generating them from binary images in the training phase, they are given to a triplet network that identifies features, instead, the poses are reference information. Finally, a regression network estimates the final position. Modifying the bease structure of PoseCNN [61], Capellen et al. [72] implemented ConvPoseCNN; this network starts with feature extraction by means of a convolutional backbone VGG16, then two fully convolutional branches calculate an initial pixel-wise semantic segmentation and then the central direction and depth. The results are fed into a third branch that estimates the quaternions for each pixel. Geometry Guided Direct Regression Network (GDR-Net), set out in the paper by Wang et al. [73], is based on both direct and indirect geometric methods: after identifying the objects of interest, it selects the RoI for each detection. The RoIs are

given as input to a network that will compute the intermediate geometric feature maps; finally, in order to obtain the 6D position of the object, the Dense Correspondences and Surface Region Attention must be provided to a Patch-PnP algorithm. Trabelsi et al. [74] constructed an end-to-end network for estimating the 6D pose of an object consisting of two modules: the first, the pose proposal module, is responsible for classifying the object and then giving an initial pose estimate; while the second, called the pose refinement module, combines a differentiable render with an iterative refiner (MARN). The objective of the paper by Hu et al. [75] is to perform regressions of 6D pose estimates at multiple scales of spatial objects and to achieve this they used a Feature Pyramid Network. Su et al. [76] implemented SynPo-Net, a CNN whose pooling layers were replaced by convolutional ones to improve the accuracy of the output.

4

Methods and Implementations

This chapter will detail the architecture of PVNet and how it works, the module that has been added to the network (DSAC) and some modifications made to the code in order to be able to carry out the experiments.

4.1 PVNet Architecture

The network used in this work is *Pixel-wise Voting Network*, also known as *PVNet* [1]. As already mentioned in the section 3.2, PVNet is a recent *two-steps pose estimation method* based on the prediction of the position of the 2D keypoints of the object in the input image and their correlation with the keypoints of the 3D model using the PnP algorithm. The particularity of this method lies in the fact that the prediction of the 2D keypoints is given by each pixel of the image belonging to the object: each of them provides *unit vectors*, i.e. the directions of the lines that connect the pixel to each keypoint of the object; subsequently, for each keypoint of the object, all the unit vectors associated with it are considered and are aggregated by RANSAC to find the final position. Once all final 2D coordinates have been identified, PnP calculates the correspondences between the keypoints of the image and those of the model; to improve this last step, it was decided to provide not only the position of the keypoints but also their probability distribution in order to give PnP more freedom and stability in identifying 2D-3D point correspondences and to give more weight to those with a lower distribution.

4.1.1 Keypoints Selection

This method for calculating the 2D-3D correspondence of keypoints requires a *3D CAD model* of the object to be located and its 3D keypoints. Many of the more recent methods [77]–[79], which adopt a strategy similar to PVNet, use the eight vertices of the 3D bounding box of the object as keypoints. However, the authors of PVNet have shown that taking points belonging to the surface of the object as keypoints results in less variance in their localisation. The points on the surface were calculated using the *Farthest Point Sampling Algorithm (FPS)* [80]: this is a greedy algorithm that, starting from a single random sample of points (in this case from the object's centroid alone), at each iteration selects the point furthest from the selected set and adds it to the group; the process ends as soon as the desired number of points is reached. The authors of PVNet conducted an experiment to find out which number of keypoints taken on the surface of the object produces the best results between 4, 8 and 12 and it turned out that it is 8 that is the best value. Therefore the value of K adopted will be 9, where the first eight are *keypoints on the surface* and the ninth is the *centroid* of the object.

4.1.2 First Step: Keypoints Localization

The architecture of PVNet is based on a "pre-trained" ResNet-18 [10] to which modifications were made so that it takes as input a single RGB image and returns as output two elements: the *mask* in which the object identification is reported and the *unit vectors* of each pixel to each keypoint. Then, thanks to the *RANSAC-based voting* of all unit vectors referring to the same keypoint, confidence scores are obtained, which in turn determine the assumptions of the 2D positions of the keypoints. Based on these, the *mean* and *covariance* of the spatial probability distribution for each is estimated. This method yields better results than the direct regression of the keypoint positions in cases of cluttered scenes, in cases where a keypoint is occluded or where it is out of the image. In this phase PVNet performs semantic segmentation and vector-field prediction: each pixel p of the input image is associated with a semantic label that defines whether or not it belongs to the object and a unit vector $\mathbf{v}_k(\mathbf{p})$ that represents the direction from the pixel to the 2D keypoint $\mathbf{x}_k$ of the object. A unit vector $\mathbf{v}_k(\mathbf{p})$ is defined

as:

$$\mathbf{v}_k(\mathbf{p}) = \frac{\mathbf{x}_k - \mathbf{p}}{\|\mathbf{x}_k - \mathbf{p}\|_2} \tag{4.1}$$

First, all pixels belonging to the mask must be identified by looking at their semantic label. Then two pixels of the mask are randomly selected and the intersection of their unit vectors is entered as the possible position of the *keypoint* $\mathbf{x}_k$ in the set of *hypotheses* $\{\mathbf{h}_{k,i}|i = 1, 2, ..., N\}$ and this is repeated until N hypotheses are obtained. At the end all pixels in the mask can vote on which is the best hypothesis according to the following formula:

$$w_{k,i} = \sum_{\mathbf{p}\in\mathbf{O}} \mathbb{I}\left(\frac{(\mathbf{h}_{k,i} - \mathbf{p})^T}{\|\mathbf{h}_{k,i} - \mathbf{p}\|_2}\mathbf{v}_k(\mathbf{p}) \geq \theta\right) \tag{4.2}$$

Where:

- $w_{k,i}$ is the voting score for a hypothesis $\mathbf{h}_{k,i}$;
- $\mathbb{I}$ is the indicator of the function;
- $p \in O$, pixel that belongs to the object;
- θ is the threshold, set to 0.99.

The hypotheses selected will be those with a higher voting score because they will turn out to be the most likely keypoint positions. Finally, for keypoint $\mathbf{x}_k$ we calculate the *mean* μ_k and the *covariance* $\mathbf{\Sigma}_k$ defined as:

$$\mu_k = \frac{\sum_{i=1}^{N} w_{k,i}\mathbf{h}_{k,i}}{\sum_{i=1}^{N} w_{k,i}} \tag{4.3}$$

$$\mathbf{\Sigma}_k = \frac{\sum_{i=1}^{N} w_{k,i}\left(\mathbf{h}_{k,i} - \mu_k\right)\left(\mathbf{h}_{k,i} - \mu_k\right)^T}{\sum_{i=1}^{N} w_{k,i}} \tag{4.4}$$

For the learning of the unit vectors in the training phase, it was chosen to use the smooth l_1 *loss function*, defined as:

$$l(\mathbf{w}) = \sum_{k=1}^{K}\sum_{\mathbf{p}\in O} l_1(\mathbf{\Delta v}_k(\mathbf{p}; \mathbf{w}|_x) + l_1(\mathbf{\Delta v}_k(\mathbf{p}; \mathbf{w}|_y) \tag{4.5}$$

$$\mathbf{\Delta v}_k(\mathbf{p}; \mathbf{w}) = \tilde{\mathbf{v}}_k(\mathbf{p}; \mathbf{w}) - \mathbf{v}_k(\mathbf{p}) \tag{4.6}$$

Where:

- $\mathbf{w}$ are the network parameters:
- $\tilde{\mathbf{v}}_k$ is the predicted vector;
- $\mathbf{v}_k$ is the ground truth of the vector;
- $\Delta\mathbf{v}_k|_x$ and $\Delta\mathbf{v}_k|_y$ are two elements of $\Delta\mathbf{v}_k$.

On the other hand, a *softmax cross-entropy loss* was used for training semantic labels.

4.1.3 Second Step: Uncertainty-Driven PnP

At this stage given the *mean* μ_k and the *covariance* $\mathbf{\Sigma}_k$ of each *keypoint* K, the *pose* (R, t) of the object is calculated, defined as a Mahalanobis distance-minimisation problem:

$$\arg\min_{R,t} \sum_{k=1}^{K} \left(\tilde{\mathbf{x}_k} - \mu_k\right)^T \mathbf{\Sigma}_k^{-1} \left(\tilde{\mathbf{x}_k} - \mu_k\right) \tag{4.7}$$

$$\tilde{\mathbf{x}_k} = \pi \left(R\mathbf{X}_k + \mathbf{t}\right) \tag{4.8}$$

where:

- $\tilde{\mathbf{x}_k}$ is the 2D projection of the coordinates of keypoint k;
- $\mathbf{X}_k$ are the coordinates of keypoint k;
- π is the perspective projection function.

EPnP initialises the parameters of *translation* t and *rotation* R of the object pose based on the four keypoints that have the covariance matrices with the lowest values. Finally, taking into account the minimisation of reprojection errors, the formula 4.7 is solved using the Levenberg-Marquardt algorithm [29].

4.2 DSAC Module

Taking the general ideas contained in the papers [81], [82], Donadi and Pretto created *Keypoints Voting Network (KVN)* [2], a "stereo" version of PVNet to which a new module was added that performs the RANSAC algorithm in a differential manner, called *DSAC*. This differential approach proved to be very effective and provided good results; therefore, in this work, it was decided to include DSAC in

a "mono" version of PVNet in order to assess any benefits to the pose estimation task. The differences between the standard PVNet approach and the proposed DSAC module will be analysed next. Following the RANSAC procedure, a set of hypotheses is realised for each keypoint by means of minimum set sampling: in this case, two *non-parallel unit vectors* (v_r, v_s) taken from distinct *pixels* (p_r, p_s) belonging to the object mask are selected, obtaining that all N_h *hypotheses* of the set ($h_{j,l}$ with $l = 0, ..., N_h - 1$) respect the following equality:

$$h_{j,l} = p_r + av_r = p_s + bv_s \tag{4.9}$$

where a and b belong to $\mathbb{R}$. Now, we must introduce the *scalar product between two vectors* ($\perp$) defined as: $u \perp w = u_x w_y - u_y w_x = |u||w| \sin(\theta)$, where θ means the angle between the vectors u and v. This operation is differential in the case that the two vectors are not parallel. Since by hypothesis the two unit vectors are not parallel, the scalar product can be used to solve formula 4.9 in the following way:

$$a = \frac{(p_s - p_r) \perp v_s}{v_r \perp v_s} \tag{4.10}$$

In the case of parallel vectors, the assumptions are considered *invalid* because the gradient cannot be calculated. Next comes the stage of calculating the inliers for the hypotheses just found: it considers a pixel to be an inlier if the predicted unit vector turns out to be very "close" to that of the ground truth. This "closeness" is calculated thanks to the *cosine similarity* and if the latter turns out to be equal to or greater than a *threshold* (t) equal to 0.99 then *pixel p* is considered an inlier (*see formula 4.2*). As this is not a differentiable step, the authors decided to replace the procedure of counting inliers with a *sum of the scores of individual pixels*, defined as:

$$S_I(h_{j,l}) = \sum_{p \in \Sigma} M(p) f(S_C(V_j(p), h_{j,l} - p)) \tag{4.11}$$

where:

- f is a differentiable or sub-differentiable function in the interval $[-1; 1]$;
- S_C is the cosine similarity operator.

Whereas the *probability distribution* for all valid hypotheses is given by the

softmax operator:

$$P(h_{j,l}) = \frac{\exp\left(S_I(h_{j,l})\right)}{\sum_{\substack{l'=0 \\ h_{j,l'}valid}}^{N_h} \exp\left(S_I(h_{j,l'})\right)} \tag{4.12}$$

While in the hypothesis calculation only some pixels of the object are interpolated, it is important to note that in this formula all pixels of the object make their contribution to the calculation of the probability distribution and this will be important for the minimisation of the loss function of the model. In the formula 4.11, the function f serves to remap the cosine similarity into an inlier score: whereas in PVNet it corresponds to a *Heaviside function* centred in t, in this case it corresponds to a *sub-differentiable leaky-ReLu* similar to a piece-wise linear function. Specifically, the new function f in the DSAC module is:

$$f(s;t,v) = \begin{cases} \frac{1-v}{1-t}(s-1)+1 & s \geq t \wedge s \leq 1 \\ \frac{v}{t+1}(s+1) & s \geq -1 \wedge s \leq t \end{cases} \tag{4.13}$$

The formula 4.13 is governed by the parameters *inlier threshold* (t) and *soft-inlier value* (v). Due to the use of equation 4.11 as a scoring function, a uniform optimisation is achieved: since each pixel will express a non-zero rating for each hypothesis, there will be no null values in the case of presumed outliers. The loss function of the model has also changed and is given by the minimisation of the following equation:

$$l_{KVN} = l_{mask} + l_{DSAC} \tag{4.14}$$

where:

- l_{mask} is the cross-entropy loss function of the mask segmentation
- l_{DSAC} is a loss function of the predicted keypoints which considers the mean square error of the keypoint on the hypothesis distribution:

$$l_{DSAC} = \sum_{j=0}^{N} \sum_{\substack{l'=0 \\ h_{j,l'}valid}}^{N_h} P(h_{j,l}) \cdot \left\| h_{i,l} - k_j^* \right\|^2 \tag{4.15}$$

  this loss function allows backward propagation of probability gradients to each individual pixel of the object. To have a controlled amplitude of the hypothesis distribution a *softmax temperature* of $\frac{1}{\alpha}$ is used, where α is trained with a *loss* l_1 on the entropy of the distribution. This strategy aims

to stabilise the training by eliminating all hypothesis distributions that would not bring any advantage to the model.

4.3 Implementations and Code Changes

The main changes made to the code to carry out the experiments are:

- **the architecture of PVNet was modified so that it could be adapted to the DSAC module.** The module was taken from the open source code made available by Donadi et al. in their paper [2]. Since the module was implemented on a "stereo" network, i.e. a network that has as input two images of the same scene from two different points of view, a clean-up of the code was necessary in order to make the code all "mono". In addition, the module parameters were entered as variables in the YAML configuration file, so that they could be easily modified;
- **the deprecated libraries of the PnP module were updated.** The original header and source files written in C++ were modified, the python file that starts the C++ source code was updated to the modified files, and finally the old method was also replaced in the evaluator;
- **the addition of a new method for reading the custom synthetic dataset** (*see section 5.1 for details*) and **other improvements** for handling datasets by object and for terminal display of the progress of the dataset preparation operation;
- **the inclusion of the 3D keypoints Root-Mean-Square Error (RMSE) metric in the list of model evaluation metrics** (*see section 5.3 for details*);
- **the addition of the `visualise_hue_votes` method** that allows the orientation of the versors to be displayed as a Hue channel in the Hue, Saturation and Lightness (HSL) representation of the selected image colours during evaluation;
- **other improvements** to the code including the elimination of hard-coded values and the insertion of editable variables from the YAML configuration file.

At the end of all these modifications, we obtain a network that is easily modified thanks to the *YAML configuration file*: it contains parameters defining the directories of the model and dataset files, network training parameters, DSAC-related parameters and finally test parameters (use of the PnP type). In this way, it is easy to carry out the experiments of interest on the four different versions of PVNet and be able to identify the effectiveness of the modules, establishing which is the best variant of the network. The variants examined are:

- PVNet with DSAC and Uncertainty PnP;
- PVNet with DSAC and EPnP;
- PVNet with Uncertainty PnP;
- PVNet with EPnP.

5

Experiments and results

In this chapter, the technical details of the various experiments conducted are described: the strategies for generating synthetic datasets in the training phase, the network parameters used, the breakdown of the dataset used during training and the evaluation of the best network model found, and the metrics implemented in the evaluation phase. Finally, an overall table with all the results obtained will be discussed.

5.1 LINEMOD Dataset and Generation of Synthetic Datasets

In the paper [4] Stefan Hinterstoisser et al. introduce a *new dataset* consisting of 15 different 3D objects without textures called LINEMOD and a new framework that utilises a Kinect for modelling, sensing (focusing on a template-based approach, also called LINEMOD) and tracking 3D objects. Due to the fact that the dataset features *cluttered scenes*, *objects without textures* and *variations in lighting* conditions, it has been very successful and has become a *standard for benchmarking* methods that aim to solve the problem of object pose estimation. Since PVNet requires a large labelled dataset in order to be able to learn how to detect the pose of an object, the authors of PVNet decided to use it. In order to prevent overfitting of the network during training, the authors of PVNet decided to combine synthetically generated images with real data (*an example can be seen in figure 5.1*) taken from the LINEMOD dataset.

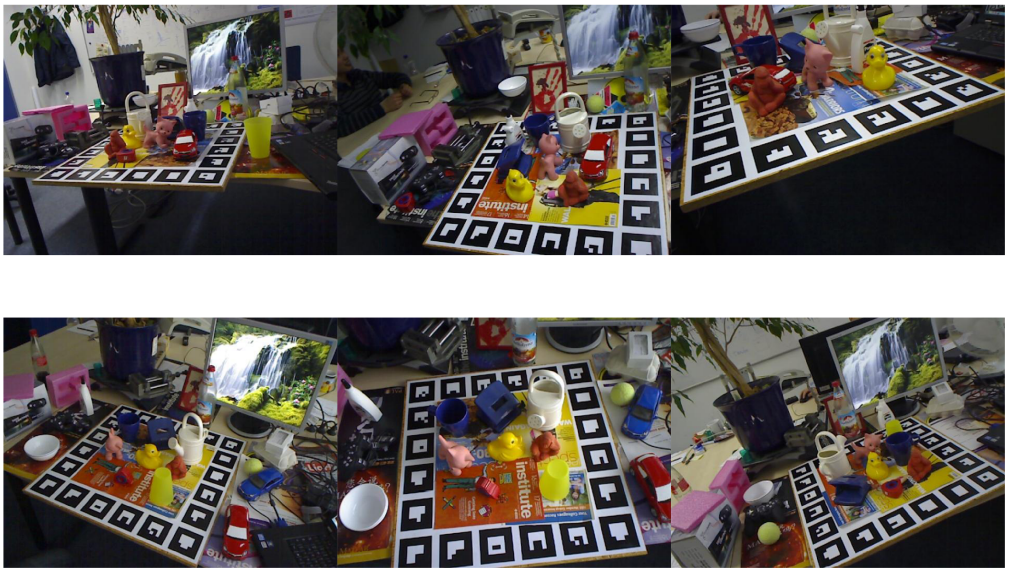

Figure 5.1: *Examples of RGB images of real dataset. The first line shows examples of RGB images for the "cat" object and the second for the "duck" object.*

A typical dataset for training PVNet contains:

- the RGB $640x480$ images, used as input, in which the pose of the object is to be identified;
- the masks of the object to be identified in black and white one for each image to be used as ground truth;
- the object positions (translation and rotation) for each image to be used as ground truth;
- the 3D CAD model of the object;
- the diameter in m of the sphere inscribing the object;
- the camera calibration K-matrix in separate space format.

The *Cut and Paste* strategy was used for the creation of a synthetic dataset of each LINEMOD object to train PVNet; this method exploits the image processing technique called *Poisson Blending* [83] and *blur*, contained in the open source library Pillow (*Python Image Library*); for further details on the procedure and the shared code see the paper [84] and the GitHub repository link: `https://github.com/debidatta/syndata-generation`. At the end of the procedure you will have:

- **10000 "render" images**, each of which consists of a randomly chosen background on which the 3D object has been superimposed; associated with each of these is the corresponding mask and pose of the object itself that can be used as ground truth. *An example can be seen in the first row of figure 5.2;*

- **10000 "fuse" images**, each of which consists of a randomly chosen background on which the 3D object has been superimposed together with the other LINEMOD objects, creating cluttered scenes and occlusions; each of these is associated with the relative mask and pose of the object itself to be used as ground truth. *An example can be seen in the second row of the figure 5.2.*

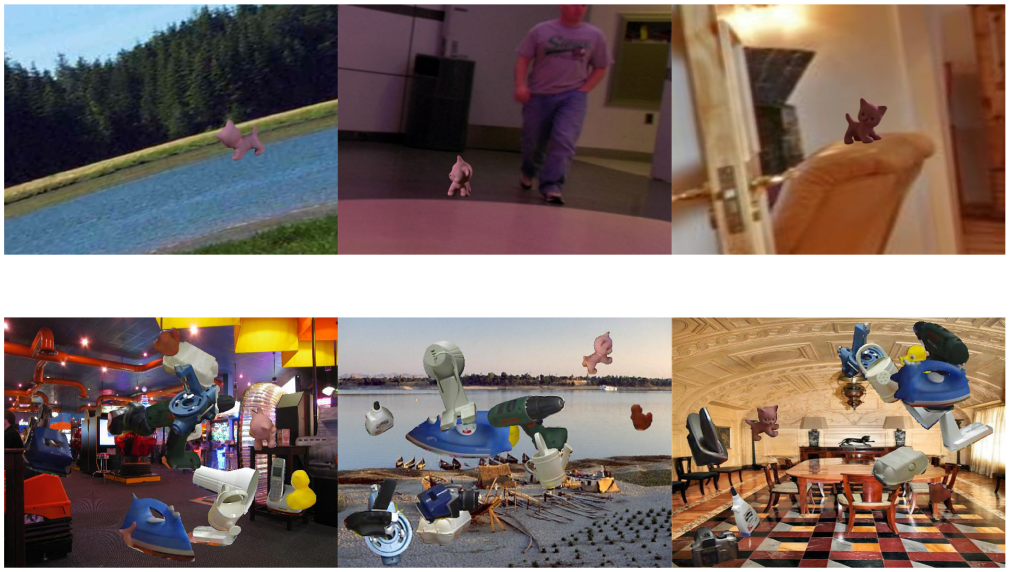

Figure 5.2: *Examples of RGB images of synthetic dataset generated by the authors of PVNet. The first line shows examples of RGB images for the render images and the second for the fuse images of "cat" object.*

To carry out the tests on the PVNet network, the authors have shared only the synthetic dataset of the "cat" object, generated with the strategy described above; in the experiments it will be called the *PVNet synthetic dataset* (specifying which of the image collections will be used). However, they have also made available an *application* called `generate_pvnet_dataset.py` that can synthetically generate datasets for network training from a 3D CAD model: it allows a collection of views of the object to be printed out, randomly choosing the surface colour, the position of the light and the background image, taken from a given folder of images. The experiments carried out were mainly based on the "cat" and "duck" object (the latter was chosen because it is the object that PVNet finds most difficult to recognise) and for both, synthetic datasets were generated with the application and the following information in the command:

- the 3D CAD model file;
- the YAML file containing all the camera parameters;

- the output folder where the dataset will be saved;
- the folder from which to take the background of the synthetic image; this was created by selecting 1000 random images chosen from the SUN397 dataset [85], the same one used by the PVNet authors;
- the unit of measurement of the CAD model used: m;
- the scale factor of the object: 1;
- the vertical axis of the model, in this case $z\text{-}axis$;
- the subdivision level of the object rotation, set to 2;
- the sampling interval of the distance in m of the object from the camera from 0.55 to 1.25 with step 0.14;
- the vertical sampling interval in m from the camera from -0.15 to 0.15 with step 0.075;
- the horizontal sampling interval from -0.2 to 0.2 with step 0.05.

The datasets thus generated consist of 19440 *synthetic images*, an example of the synthetic images generated by this application can be seen in the figure 5.3. In the experiments it will be called *custom synthetic dataset* (specifying the object of interest).

Figure 5.3: *Examples of RGB images of synthetic dataset generated with the command. The first line shows examples of RGB images for the "cat" object and the second for the "duck" object.*

5.2 Procedure for Creating the Trained PVNet Model

Next, the steps necessary to build a trained PVNet model capable of pose estimation of the object concerned are explained. The main steps are: the dataset preparation phase, the training phase and the phase in which the performance of the model is evaluated.

5.2.1 Preparation Phase and Dataset Partitioning

Launching the command `prepare_linemod_dataset.py` from the terminal followed by the path to the folder containing the LINEMOD dataset and the object on which the training is to be performed starts the procedure for *preparing the dataset for PVNet*. The images of the real dataset and the synthetic dataset are divided between those which will be useful for the training phase, the validation phase and the test phase. At the end we will obtain three JSON files useful for the next steps:

- **train.json**, containing for each individual image in the dataset useful for training the network, all the information about the image and its ground truth, all the paths useful for finding the masks and the image itself. In the case of the "cat" object there are initially 177 real images to which synthetic ones are added (custom or "fuse" and "render"), while for "duck" there are initially 189 real images to which synthetic images are added (custom only);
- **val.json**, containing for each individual image in the dataset useful for the validation phase of the model, all the information about the image and its ground truth, all the paths useful for finding the masks and the image itself. In the case of the "cat" object, there are 501 real images, while for "duck" there are 532;
- **test.json**, containing for each individual image in the dataset useful for the testing phase of the best model, all information about the image and its ground truth, all paths useful for finding the masks and the image itself. In the case of the "cat" object there are 1002 real images, while for "duck" there are 1065;

5.2.2 Training Phase and Parameters

Once the files containing the dataset annotations are ready, you can run the training command `pvnet_train.py` from the terminal, with the following information:

- the path to the folder containing the object dataset;
- the path to the folder where the trained models will be saved;
- the number of training examples in a forward/backward step; this batch parameter depends on the system specification on which the network is trained, by default 2 is chosen;
- the number of epochs in which training is to take place, set at 150;
- the number of epochs after which an evaluation is to be carried out on the validation dataset, in this case 10;
- the number of epochs after which the model is to be saved in the training folder, in this case 10;
- the configuration file, containing not only the parameters of the network during training and testing and of the added modules, but also parameters that regulate the behaviour of the network: for example, the Boolean value `resume` if set to `true` allows the network to recover the training at the point where it was left (*useful to activate in the event of a crash*).

At the end of the training procedure, in the folder dedicated to the trained model you will have all the *saved checkpoints of the network* (from which the best model will be selected) and a "record" folder in which losses and evaluation metrics are recorded so that they can be monitored in a local tensorboard session. All training was performed on the *DEI Cluster platform*, for more information on the hardware and the use of *Slurm Workload Manager* please refer to the guide at this link: `https://clusterdeiguide.readthedocs.io/en/latest/index.html`.

5.2.3 Evaluation Phase

Once the network has finished training, the best model will be saved in a "`best_model`" subfolder within the training files folder. Launching the command `pvnet_eval.py` from the terminal followed by the path to the folder containing the object dataset, the path to the `best_model` folder and the YAML configuration file (used to enable/disable the modules added to the network) performs an evaluation of the model on the test set. In addition, images of the test set can be displayed with the 3D bounding boxes identified in blue and the ground truth in green.

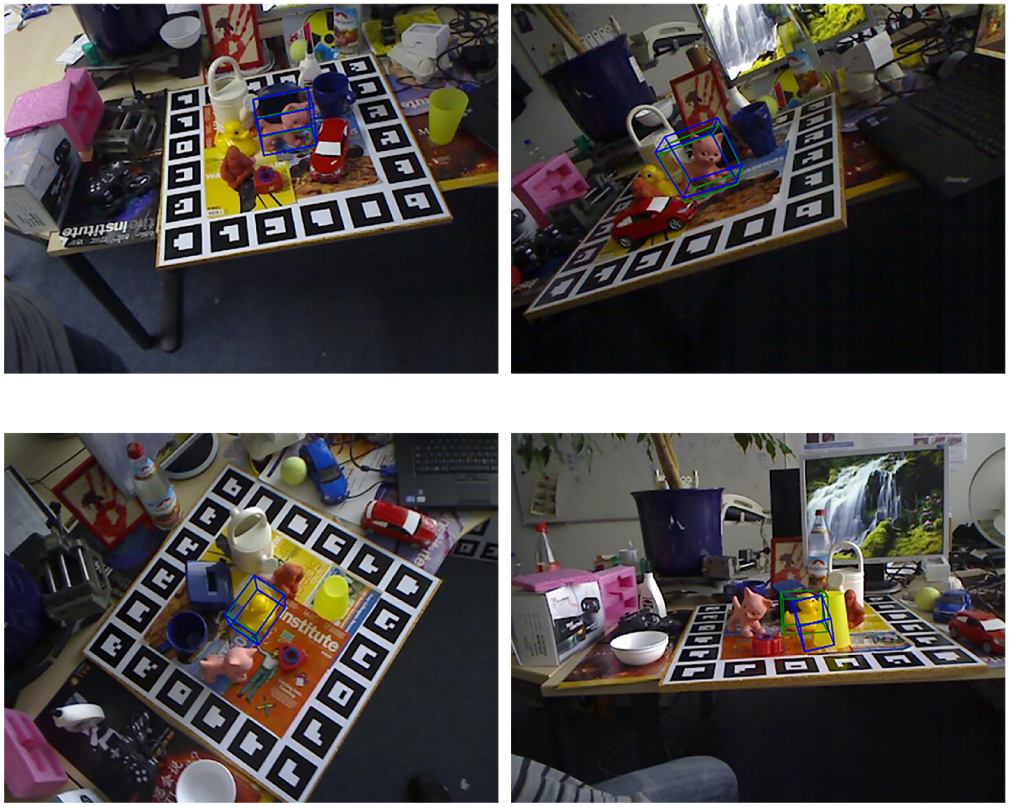

Figure 5.4: *Output of the evaluation phase: in the first line we have the object "cat" with an example of good estimate and one of bad; in the second line we have the same thing but referring to "duck."*

5.3 Description of Metrics Used

The metrics used in the evaluation of the trained models are:

- **Mask ap70 metric**: this returns the percentage of samples whose intersection over union value is equal to or greater than 0.7; intersection over union is calculated as the ratio between the area of overlap between the projected bounding box and that of the ground truth and their union;
- **2D projections metric**: this calculates the percentage of samples where the average distance between the projected 2D keypoints and the ground truth keypoints is less than a certain threshold, in this case 5 pixels;
- **ADD metric**: the Average 3D distance of the model points measures the percentage of "*accurately*" predicted poses in the test set. If the average distance of the 3D model points in the predicted coordinate system to that of the ground truth is less than one-tenth of the diameter of the object, then the predicted pose of the object can be said to be accurate;
- **5 cm 5 degrees metric**: this metric returns the percentage of samples that have the transformation matrix with at most 5 centimetres of difference in translation and 5 degrees in rotation from the ground truth matrix;

- **3D keypoints RMSE**: the root square mean errors of the 3D keypoints returns the percentage of how many samples have the square root of the mean square difference between the vectors of the predicted 3D keypoints and those of the ground truth below a certain threshold, in this case 5 pixels. The formula for calculating the RMSE is:

$$RSME = \sqrt{\frac{\sum (\mathbf{y}_i - \tilde{\mathbf{y}}_i)^2}{N}} \tag{5.1}$$

  Where:

  - $\tilde{\mathbf{y}}_i$ is the expected value for the $i\text{-}th$ observation;
  - $\mathbf{y}_i$ is the observed value for the $i\text{-}th$ observation;
  - N is the sample size.

5.4 Overall Table of Experiments

This section contains general tables of the experiments conducted according to the object considered, so that the reader can compare the performance of network variants for each experiment. The following tables show:

- the **network type** (Net. type) to indicate whether the DSAC module is present or not;
- the **scoring function** and **loss parameters** if DSAC is present. For the scoring function, both the type of function used and the two parameters `dsac_beta` and `dsac_threshold` are reported, which go to define the Sigmoid function or vary the "elbow" point of the Piece-Wise function. For the loss function, on the other hand, the weight to be given to the loss components of the mask and DSAC is indicated;
- whether **Uncertainty PnP** or **EPnP** is present;
- how the model's **training dataset** is constituted, whether real data is present and the type of synthetic data used;
- the **metrics** obtained from the configuration.

Overall Results Table for "CAT" Object

Net. Type	Parameters		Un.	Dataset				Metrics				
	Score f.	Loss f.	PnP	Real	Synthetic R	Synthetic F	Synthetic C	2D Proj.	ADD	5cm 5°	Mask ap70	3D Kps RSME
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.41	79.44	95.31	99.70	91.52
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	82.14	95.31	99.40	92.02
DSAC	Piece-Wise dsac_β = 0.5 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	77.55	93.81	99.60	90.22
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.2	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	78.74	92.62	99.30	89.82
DSAC	Sigmoid dsac_β = 1.7 dsac_t = 0.001	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	76.75	93.01	99.60	87.65
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X			99.40	73.25	92.42	99.70	88.12
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X			99.00	71.26	88.82	99.60	82.74
DSAC	Piece-Wise dsac_β = 0.5 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X			99.20	62.08	87.43	99.60	79.74
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.2	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X			99.10	64.77	84.63	99.70	80.44
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X	X	X		98.60	75.85	95.51	99.60	88.82
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X			X	97.51	55.09	72.66	94.11	71.46
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X			X	0.97305	64.77	80.74	95.21	79.04
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X			X	97.51	59.98	78.94	95.41	75.45
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X			X	98.00	65.37	81.34	98.30	79.04
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X		X	X	99.20	74.45	94.01	99.50	87.82
DSAC	Piece-Wise dsac_β = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X		X	X	99.10	72.26	90.92	99.40	85.03

(Continued on next page)

(Continued on previous page)

	dsac_$t = 0.1$											
	Piece-Wise	$1 * l_{Mask}$										
DSAC	dsac_$\beta = 0.3$	$1 * l_{DSAC}$		X		X	X	99.30	76.95	93.81	99.60	90.02
	dsac_$t = 0.1$											
	Piece-Wise	$1 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$1 * l_{DSAC}$		X		X	X	98.80	76.95	92.22	99.60	87.43
	dsac_$t = 0.1$											
	Piece-Wise	$1 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$1 * l_{DSAC}$	X		X	X		98.30	44.81	83.13	96.41	61.88
	dsac_$t = 0.1$											
	Piece-Wise	$1 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$1 * l_{DSAC}$			X	X		98.70	63.17	86.03	98.70	80.94
	dsac_$t = 0.1$											
	Piece-Wise	$1 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$1 * l_{DSAC}$	X				X	57.49	15.47	19.56	45.41	25.45
	dsac_$t = 0.1$											
	Piece-Wise	$1 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$1 * l_{DSAC}$					X	62.38	24.15	28.04	42.62	34.23
	dsac_$t = 0.1$											
	Piece-Wise	$1 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$1 * l_{DSAC}$	X		X			59.18	21.16	27.05	52.99	30.84
	dsac_$t = 0.1$											
	Piece-Wise	$1 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$1 * l_{DSAC}$			X			71.36	28.54	32.04	68.56	39.42
	dsac_$t = 0.1$											
	Piece-Wise	$1.2 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$0.8 * l_{DSAC}$	X		X	X		98.40	47.51	82.24	99.50	66.27
	dsac_$t = 0.1$											
	Piece-Wise	$1.2 * l_{Mask}$										
DSAC	dsac_$\beta = 0.1$	$0.8 * l_{DSAC}$			X	X		98.70	52.40	88.92	99.30	69.56
	dsac_$t = 0.1$											
Standard PVNet			X	X	X	X		99.60	70.06	97.61	99.90	84.13
Standard PVNet				X	X	X		99.70	75.36	97.41	99.80	88.42
Standard PVNet				X	X			99.40	67.17	92.81	99.80	82.04
Standard PVNet			X	X			X	98.60	68.36	88.72	98.60	80.35
Standard PVNet				X			X	98.80	68.46	88.12	99.00	81.84
Standard PVNet			X	X		X	X	98.80	73.65	96.01	98.20	86.83
Standard PVNet				X		X	X	99.40	71.76	96.31	99.70	86.13

(Continued on next page)

(Continued on previous page)

Standard PVNet		X		X	X		98.70	56.29	93.01	99.40	71.46
Standard PVNet				X	X		98.70	53.79	79.94	99.50	69.26
Standard PVNet		X				X	42.52	19.76	24.45	33.03	27.94
Standard PVNet						X	44.11	23.45	19.46	36.53	31.84
Standard PVNet		X		X			63.37	24.85	24.85	64.87	33.93
Standard PVNet				X			60.88	22.95	23.35	62.68	32.93

Table 5.1: *Overall table with the results of the experiments carried out on the "cat" object.*

Overall Results Table for "DUCK" Object

Net. Type	Parameters		Un. PnP	Dataset		Metrics				
	Score f.	Loss f.		Real	Synthetic Custom	2D Proj.	ADD	5cm 5°	Mask ap70	3D Kps RSME
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X	X	92.39	37.84	56.34	94.09	67.70
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X	X	93.05	33.62	47.42	93.71	62.07
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	85.73	30.80	51.55	93.33	58.87
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	89.67	34.37	49.86	93.15	64.04
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$	X		X	08.45	04.51	25.73	61.60	22.16
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$			X	13.15	08.36	26.39	54.27	34.84
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1.2 * l_{Mask}$ $0.8 * l_{DSAC}$	X		X	11.46	05.45	30.99	62.82	24.13
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1.2 * l_{Mask}$ $0.8 * l_{DSAC}$			X	06.76	04.88	26.29	55.87	24.41

(Continued on next page)

(Continued on previous page)

Standard PVNet		X	X	X	94.27	36.24	64.98	97.09	63.19
Standard PVNet			X	X	95.78	36.81	64.88	98.22	67.51
Standard PVNet		X		X	05.73	03.29	14.46	62.91	24.98
Standard PVNet				X	07.32	03.01	18.78	54.65	19.62

Table 5.2: *Overall table with the results of the experiments carried out on the "duck" object.*

5.5 Discussion of Results

The discussion of the results obtained is subdivided by type of experiment and partial tables are given showing significant examples to confirm the thinking expressed. It is recommended to also compare with the general tables 5.1 and 5.2 to get a more complete view of the results. Further possible verifications are proposed at the end of each experiment in order to analyse the case in more detail.

5.5.1 Experiment 1: DSAC or Standard PVNet?

In tables 5.3 and 5.4, one can see the difference between the values of the best DSAC model and the best standard PVNet model, for "cat" and "duck" respectively. Except in the case with real and synthetic custom datasets, DSAC proved to bring a good increase in ADD, slightly penalising the other metrics. Therefore, it can be said that the use of DSAC was very effective in the case of both the "cat" and "duck" object; in fact, it raised the percentage of accurate object pose prediction (ADD), proving to be more efficient and effective than the standard PVNet version. This is due to the fact that DSAC is a differential approach: the votes of the pixels in the mask no longer contain null values, and therefore all actively participate in both the voting phase for the best location hypothesis for the keypoint and the error backpropagation phase. For more details on the difference between the presence of DSAC in the network and standard PVNet, see section 4.2. The following images are intended to show

how close the keypoints predicted with the DSAC module are to those of ground truth and compare them with those predicted by the PVNet standard.

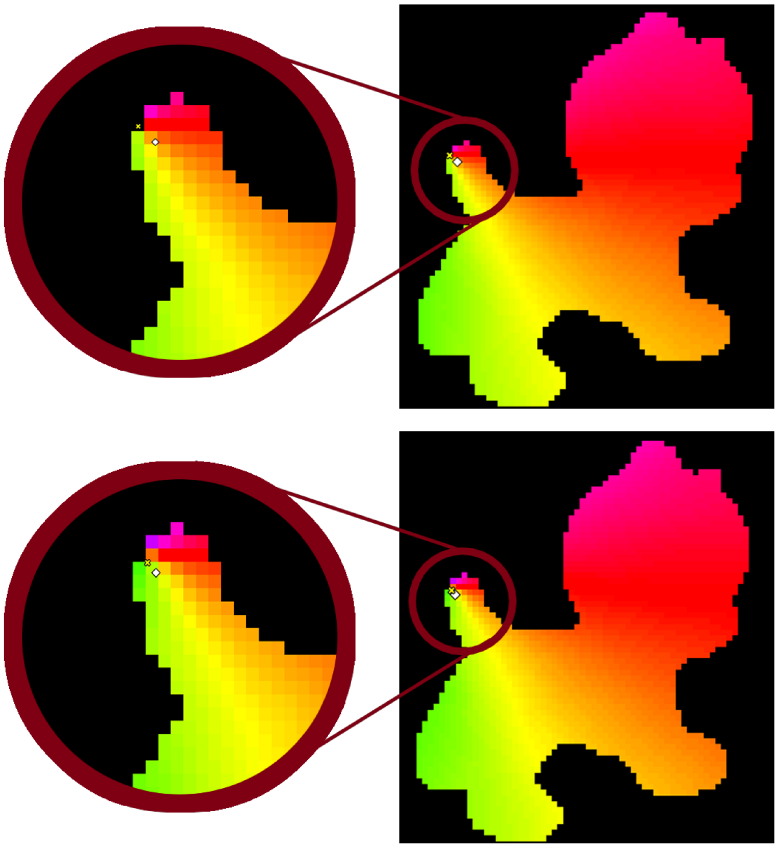

Figure 5.5: *Example of prediction of keypoint number 4 in the test set image with ID = 99: on the top we have the case of the Standard PVNet model while on the bottom we have the model with the DSAC module. The yellow "**X**" corresponds to the predicted keypoint, while the white "**Diamond**" for ground truth.*

Experiment 1 for "CAT" Object											
Net. Type	**Parameters**		**Un. PnP**	**Dataset**				**Metrics**				
	Score f.	**Loss f.**		**Real**	**Synthetic**			**2D Proj.**	**ADD**	**5cm 5°**	**Mask ap70**	**3D Kps RSME**
					R	**F**	**C**					
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	**82.14**	95.31	99.40	**92.02**
Standard PVNet				X	X	X		**99.70**	75.36	**97.41**	**99.80**	88.42

Table 5.3: *Comparison table of the best model obtained with the DSAC module and the best standard PVNet model for the "cat" object.*

Experiment 1 for "DUCK" Object										
Net. Type	**Parameters**		**Un. PnP**	**Dataset**		**Metrics**				
	Score f.	**Loss f.**		**Real**	**Synthetic Custom**	**2D Proj.**	**ADD**	**5cm 5°**	**Mask ap70**	**3D Kps RSME**
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X	X	92.39	**37.84**	56.34	94.09	**67.70**
Standard PVNet			X	X	X	**94.27**	36.24	**64.98**	**97.09**	63.19

Table 5.4: *Comparison table of the best model obtained with the DSAC module and the best standard PVNet model for the "duck" object.*

5.5.2 Experiment 2: Best Scoring Function Parameters

Once it was established that the DSAC module improved system performance, it was decided to test which combination of scoring function parameters would maximise the results on average. A first series of tests (*see table 5.5*) was performed with the entire synthetic dataset provided by the authors of PVNet with the following parameters for the scoring function:

1. Piece-Wise function type with `dsac_beta` = 0.3 and `dsac_threshold` = 0.1 which corresponds to the *blue function* in the image 5.6;

2. Piece-Wise function type with `dsac_beta` = 0.1 and `dsac_threshold` = 0.1 which corresponds to the *purple function* in the image 5.6;

3. Piece-Wise function type with `dsac_beta` = 0.5 and `dsac_threshold` = 0.1 which corresponds to the *green function* in the image 5.6;

4. Piece-Wise function type with `dsac_beta` = 0.3 and `dsac_threshold` = 0.2 which corresponds to the *red function* in the image 5.6;

5. Sigmoid function type with `dsac_beta` = 1.7 and `dsac_threshold` = 0.001 which corresponds to the *yellow function* in the image 5.6;

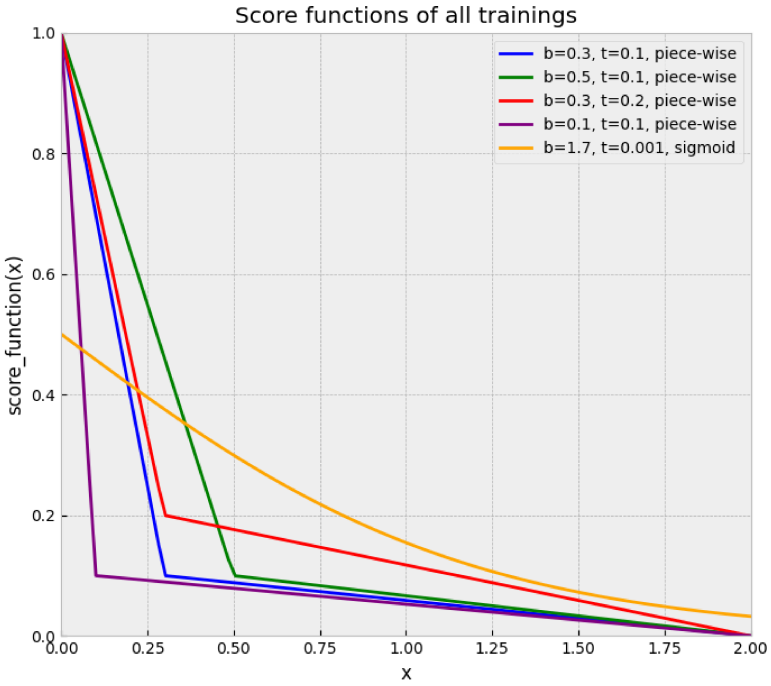

Figure 5.6: *Graph containing the scoring functions used in experiment 2.*

Experiment 2 - part 1 for "CAT" Object

Net. Type	Parameters: Score f.	Parameters: Loss f.	Un. PnP	Dataset: Real	Dataset: Synthetic R	Dataset: Synthetic F	Dataset: Synthetic C	Metrics: 2D Proj.	Metrics: ADD	Metrics: 5cm 5°	Metrics: Mask ap70	Metrics: 3D Kps RSME
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		**99.41**	79.44	**95.31**	**99.70**	91.52
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	**82.14**	**95.31**	99.40	**92.02**
DSAC	Piece-Wise dsac_$\beta = 0.5$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	77.55	93.81	99.60	90.22
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.2$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	78.74	92.62	99.30	89.82
DSAC	Sigmoid dsac_$\beta = 1.7$ dsac_$t = 0.001$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		99.10	76.75	93.01	99.60	87.65

Table 5.5: *Summary table of the first part of the search for the best parameters for the DSAC module with PVNet synthetic dataset ("fuse" and "render") and a small amount of real images.*

The *Sigmoid function* type was abandoned in subsequent tests because it was found to be less performing and more complicated to train. It was decided to retry the first four configurations using only half of the dataset (only the 10000 "render" images with the real ones) in order to be able to assess on average which combination was better, see table 5.6 for details; it turned out that combinations 1 and 2 were good variants on which to run the next experiments.

Experiment 2 - part 2 for "CAT" Object

Net. Type	Parameters		Un. PnP	Dataset				Metrics				
	Score f.	Loss f.		Real	Synthetic R	F	C	2D Proj.	ADD	5cm 5°	Mask ap70	3D Kps RSME
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X			**99.40**	**73.25**	**92.42**	**99.70**	**88.12**
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X			99.00	71.26	88.82	99.60	82.74
DSAC	Piece-Wise dsac_$\beta = 0.5$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X			99.20	62.08	87.43	99.60	79.74
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.2$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X			99.10	64.77	84.63	**99.70**	80.44

Table 5.6: *Summary table of the second part of the search for the best parameters for the DSAC module with PVNet synthetic dataset (only "render") and a small amount of real images.*

5.5.3 Experiment 3: Uncertainty PnP or EPnP?

After having identified the two best combinations for the scoring function, it was decided to carry out the subsequent experiments considering the four possible network variants presented at the end of the section 4.3. In this way, the difference in behaviour between Uncertainty PnP and EPnP could be verified in more detail. This test showed that, overall, for the "cat" object 5.7, the performance of the network drops if both the DSAC module and Uncertainty PnP are present; in the case of standard PVNet and Uncertainty PnP, on the other hand, on average, the metrics are slightly better. For the "duck" object 5.8, what was said earlier is also confirmed, only it is less evident due to the smaller number of tests performed.

Experiment 3 for "CAT" Object												
Net. Type	**Parameters**		**Un. PnP**	**Dataset**				**Metrics**				
	Score f.	**Loss f.**		**Real**	**Synthetic**			**2D Proj.**	**ADD**	**5cm 5°**	**Mask ap70**	**3D Kps RSME**
					R	**F**	**C**					
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X	X	X		98.60	75.85	**95.51**	**99.60**	88.82
Standard PVNet			X	X	X	X		99.60	70.06	**97.61**	**99.90**	84.13
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		**99.41**	**79.44**	95.31	99.70	**91.52**
Standard PVNet				X	X	X		**99.70**	**75.36**	97.41	99.80	**88.42**

Table 5.7: *Comparative table for the "cat" object showing the difference in system performance in relation to the type of PnP used.*

Experiment 3 for "DUCK" Object										
Net. Type	**Parameters**		**Un. PnP**	**Dataset**		**Metrics**				
	Score f.	**Loss f.**		**Real**	**Synthetic Custom**	**2D Proj.**	**ADD**	**5cm 5°**	**Mask ap70**	**3D Kps RSME**
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$	X	X	X	**93.05**	33.62	47.42	**93.71**	62.07
Standard PVNet			X	X	X	94.27	36.24	**64.98**	97.09	63.19
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	89.67	**34.37**	**49.86**	93.15	**64.04**
Standard PVNet				X	X	**95.78**	**36.81**	64.88	**98.22**	**67.51**

Table 5.8: *Comparative table for the "duck" object showing the difference in system performance in relation to the type of PnP used.*

5.5.4 Experiment 4: There is a Dependence of Real Data in Training?

This experiment aims to test whether or not the system has a certain dependency on the real images in the dataset being trained. From the "cat" object 5.9, a certain dependency emerges, although it is from the "duck" object 5.10 that it is most evident: all metrics are significantly lower in the case of completely synthetic images. This experiment mainly highlights two important aspects:

- **how real data are an excellent learning resource for neural networks**, but they are very expensive and time-consuming to implement;
- **The importance of having a good synthetic dataset generation strategy that produces robust, consistent and well-structured data.** A synthetic dataset must have statistically well-diversified and distributed data that maintains sufficient realism to make the model suitable for real scenarios. In addition, *customisation* and *parameterisation* must be maximised in the generation phase, so that developers can adapt the synthetic dataset as much as possible to the specific needs of the neural network. This point will be analysed specifically in the next experiment.

Experiment 4 for "CAT" Object

Net. Type	Parameters		Un. PnP	Dataset				Metrics				
	Score f.	Loss f.		Real	Synthetic R	Synthetic F	Synthetic C	2D Proj.	ADD	5cm 5°	Mask ap70	3D Kps RSME
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		**99.10**	**82.14**	**95.31**	**99.40**	**92.02**
Standard PVNet				X	X	X		**99.70**	**75.36**	**97.41**	**99.80**	**88.42**
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$			X	X		98.70	63.17	86.03	98.70	80.94
Standard PVNet					X	X		98.70	53.79	79.94	99.50	69.26

Table 5.9: *Table showing the dependence of the system on the real data in the training phase for the "cat" object.*

Experiment 4 for "DUCK" Object										
Net. Type	**Parameters**		**Un. PnP**	**Dataset**		**Metrics**				
	Score f.	**Loss f.**		**Real**	**Synthetic Custom**	**2D Proj.**	**ADD**	**5cm 5°**	**Mask ap70**	**3D Kps RSME**
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	**85.73**	**30.80**	**51.55**	**93.33**	**58.87**
Standard PVNet				X	X	**95.78**	**36.81**	**64.88**	**98.22**	**67.51**
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$			X	13.15	08.36	26.39	54.27	34.84
Standard PVNet					X	07.32	03.01	18.78	54.65	19.62

Table 5.10: *Table showing the dependence of the system on the real data in the training phase for the "duck" object.*

5.5.5 Experiment 5: More Effective Synthetic Data Generation Strategy

This experiment aims to compare different strategies for generating synthetic datasets and identify the perfect mix of data for the training dataset. The question arises: *"if real data allows the network to achieve excellent results, why such an effort to generate synthetic data?"* The main advantages of this approach are:

- **COSTS**. Real data are very expensive to collect and require a lot of time and resources to optimise. Datasets are often inaccurate, distorted and require an extremely expensive labelling process. Synthetic data, for the same quality, reduces the overall cost and thus allows developers to create more models, which benefits research and development;

- **TIMING**. Training requires huge amounts of data. Retrieving real datasets of this size is time-consuming and making these datasets usable can be an even more arduous task. Synthesised data is scalable, helps to develop robust datasets quickly and complements real data by allowing better training that guarantees more accurate predictions;

- **QUANTITY**. Real datasets have physical limits to their size. Synthetic data can be generated in large quantities without any particular difficulty. This results in more robust decision-making neural networks and thus greater reliability;

- **PRIVACY**. Collecting data has many privacy implications for the people who interact with the monitored systems. While synthetic data have no privacy regulations.

As already mentioned in the previous experiment, it is important to adopt a synthetic data generation strategy that produces data as close to the real thing as possible: Table 5.11 shows that the "`generate_pvnet_dataset`" application provided by the PVNet authors (which enabled the creation of the custom synthetic dataset) is not as efficient as the **Cut and Paste** strategy in the paper [84]. From table 5.12, it can be seen that the biggest difference between the two strategies lies in the fact that the second one has the 10000 "fuse" images: the inclusion of the other LINEMOD objects and the realisation of cluttered scenes and occlusions of the object significantly increased the learning of the model. It would be interesting to be able to conduct further tests on:

- **other LINEMOD objects** besides "cat" and "duck" in order to detect further differences;
- **other strategies for generating synthetic data** such as, for example, the one using the BLENDER program (*see link for more details:* `https://github.com/zju3dv/pvnet-rendering`).

Experiment 5 - part 1 for "CAT" Object

Net. Type	Parameters		Un. PnP	Dataset				Metrics				
	Score f.	Loss f.		Real	Synthetic R	Synthetic F	Synthetic C	2D Proj.	ADD	5cm 5°	Mask ap70	3D Kps RSME
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		**99.41**	**79.44**	**95.31**	**99.70**	**91.52**
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		**99.10**	**82.14**	**95.31**	**99.40**	**92.02**
Standard PVNet				X	X	X		**99.70**	**75.36**	**97.41**	**99.80**	**88.42**
DSAC	Piece-Wise dsac_β = 0.3 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X			X	97.51	59.98	78.94	95.41	75.45
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$		X			X	98.00	65.37	81.34	98.30	79.04
Standard PVNet				X			X	98.80	68.46	88.12	99.00	81.84

(Continued on next page)

(Continued on previous page)

Table 5.11: *Comparative table of strategies for generating synthetic datasets and showing the effectiveness of using the "Cut and Paste" strategy for generating synthetic data.*

Experiment 5 - part 2 for "CAT" Object												
Net. Type	**Parameters**		**Un. PnP**	**Dataset**				**Metrics**				
	Score f.	**Loss f.**		**Real**	**Synthetic**			**2D Proj.**	**ADD**	**5cm 5°**	**Mask ap70**	**3D Kps RSME**
					R	**F**	**C**					
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		**99.41**	**79.44**	**95.31**	**99.70**	**91.52**
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X	X	X		**99.10**	**82.14**	**95.31**	99.40	**92.02**
DSAC	Piece-Wise dsac_$\beta = 0.3$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X		X	X	99.30	76.95	93.81	99.60	90.02
DSAC	Piece-Wise dsac_$\beta = 0.1$ dsac_$t = 0.1$	$1 * l_{Mask}$ $1 * l_{DSAC}$		X		X	X	98.80	76.95	92.22	**99.60**	87.43

Table 5.12: *Table highlighting the importance of the 10000 "fused" images of the "Cut and Paste" strategy, thanks to its cluttered scenes and occlusions.*

5.5.6 Experiment 6: Variation in Loss Function Weights

During the experiments, it became apparent that the Mask ap70 metric decreased both when custom synthesised data was inserted and/or real data was removed during training and with the presence of the DSAC module. For the former case, the previous experiment showed us that the generation strategy used is not particularly effective and that we need to look for generation strategies that synthesise images that are more random and "close" to reality. In the second case, since a good object mask represents a criticality in the quantity and quality of votes, it was decided to give a different weight to the components of the loss function 4.14 if DSAC module is present in the network:

- to the l_{DSAC} component a weight of 0.8;
- to the l_{Mask} component a weight of 1.2.

Tables 5.13 and 5.14 show the results obtained divided by object, and it can be seen that on average, the values of the metrics increase slightly with the

exception of ADD, which is somewhat penalised when EPnP is present. It would be interesting to investigate the weighting of the loss function components by changing both combination and values.

Experiment 6 for "CAT" Object

Net. Type	Parameters		Un. PnP	Dataset				Metrics				
	Score f.	Loss f.		Real	Synthetic R	F	C	2D Proj.	ADD	5cm 5°	Mask ap70	3D Kps RSME
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$	X		X	X		98.30	44.81	**83.13**	96.41	61.88
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$			X	X		**98.70**	**63.17**	86.03	98.70	**80.94**
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1.2 * l_{Mask}$ $0.8 * l_{DSAC}$	X		X	X		**98.40**	**47.51**	82.24	**99.50**	**66.27**
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1.2 * l_{Mask}$ $0.8 * l_{DSAC}$			X	X		**98.70**	52.40	**88.92**	**99.30**	69.56

Table 5.13: *Comparative table of the same models trained with a different loss function in the case of the "cat" object.*

Experiment 6 for "DUCK" Object

Net. Type	Parameters		Un. PnP	Dataset		Metrics				
	Score f.	Loss f.		Real	Synthetic Custom	2D Proj.	ADD	5cm 5°	Mask ap70	3D Kps RSME
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$	X		X	08.45	04.51	25.73	61.60	22.16
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1 * l_{Mask}$ $1 * l_{DSAC}$			X	**13.15**	**08.36**	**26.39**	54.27	**34.84**
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1.2 * l_{Mask}$ $0.8 * l_{DSAC}$	X		X	**11.46**	**05.45**	**30.99**	**62.82**	**24.13**
DSAC	Piece-Wise dsac_β = 0.1 dsac_t = 0.1	$1.2 * l_{Mask}$ $0.8 * l_{DSAC}$			X	06.76	04.88	26.29	**55.87**	24.41

Table 5.14: *Comparative table of the same models trained with a different loss function in the case of the "duck" object.*

6

Conclusions

The purpose of this paper is to explain the importance of *6D pose estimation* in the field of computer vision and to highlight how researchers are working to perfect it with the help of increasingly *innovative tools* and *approaches*. The *results of experiments* carried out on one of the main pose estimation methods, called PVNet, are analysed:

- **the introduction of the DSAC module**, realised by Donadi and Pretto [2]. It proves to be very effective and brings numerous advantages over the standard version of PVNet;
- **The parameterisation of the scoring function of the DSAC module**, which on average returns better results. The best configuration identified turns out to be the Piece-Wise function with `dsac_beta` and `dsac_threshold` parameters equal to 0.1.
- **the influence of the type of PnP used.** Uncertainty PnP did not prove very useful after introducing the DSAC module, it performs better with EPnP;
- **the dependence or otherwise of the model on the real data in the training phase.** The network needs a small amount of real images in order to achieve a good level of learning;
- **the most effective synthetic data generation strategy.** The "Cut and Paste" strategy used by the PVNet authors turns out to produce data that is more varied and approximate to the real data than the "`generate_PVNet_dataset`" application that generated the custom synthetic dataset;
- **the influence of weights in the loss function.** In the case of "imprecise" synthetic data, changing the weights of the loss function components can benefit the network.

Developing an experimental thesis of this type has allowed me to put into practice the knowledge I learnt during my Computer Vision course, to go into more detail on the programming of neural networks and to understand how they actually work. I am satisfied with the results obtained and grateful for having had the opportunity to contribute, certainly in a small way, to the studies inherent to 6D pose estimation.

References

[1] S. Peng, Y. Liu, Q. Huang, X. Zhou, and H. Bao, "Pvnet: Pixel-wise voting network for 6dof pose estimation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4556–4565. DOI: 10.1109/CVPR.2019.00469.

[2] I. Donadi and A. Pretto, *Kvn: Keypoints voting network with differentiable ransac for stereo pose estimation*, 2023. arXiv: 2307.11543 [cs.CV].

[3] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692. [Online]. Available: https://doi.org/10.1145/358669.358692.

[4] 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," vol. 7724, Oct. 2012, ISBN: 978-3-642-37330-5. DOI: 10.1007/978-3-642-33885-4_60.

[5] R. Dai and I. Akyildiz, "Akyildiz, i.f.: A spatial correlation model for visual information in wireless multimedia sensor networks. ieee transactions on multimedia 11(6), 1148-1159," *Multimedia, IEEE Transactions on*, vol. 11, pp. 1148–1159, Nov. 2009. DOI: 10.1109/TMM.2009.2026100.

[6] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. DOI: 10.1017/CBO9781107298019.

[7] I. Sarker, H. Alqahtani, F. Alsolami, A. Khan, Y. Abushark, and M. K. Siddiqui, "Context pre-modeling: An empirical analysis for classification based user-centric context-aware predictive modeling," *Journal Of Big Data*, vol. 7, Jul. 2020. DOI: 10.1186/s40537-020-00328-3.

[8] Y. Lecun, "Generalization and network design strategies," English (US), in *Connectionism in perspective*, R. Pfeifer, Z. Schreter, F. Fogelman, and L. Steels, Eds. Elsevier, 1989.

[9] V. Ikawati, Y. Indrasary, and A. Prihatmanto, "Prediction of covid-19 disease using x-ray images with deep learning algorithm," *Journal of Applied Science and Advanced Engineering*, vol. 1, pp. 28–34, Jan. 2023. DOI: 10.59097/jasae.v1i1.11.

[10] "Deep residual learning for image recognition," Jun. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90.

[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[12] R. K. Srivastava, K. Greff, and J. Schmidhuber, *Training very deep networks*, 2015. arXiv: 1507.06228 [cs.LG].

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2023. arXiv: 1706.03762 [cs.CL].

[14] Y. Liu, Y. Zhang, Y. Wang, F. Hou, J. Yuan, J. Tian, Y. Zhang, Z. Shi, J. Fan, and Z. He, *A survey of visual transformers*, 2022. arXiv: 2111.06091 [cs.CV].

[15] S. Latif, A. Zaidi, H. Cuayahuitl, F. Shamshad, M. Shoukat, and J. Qadir, *Transformers in speech processing: A survey*, 2023. arXiv: 2303.11607 [cs.CL].

[16] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 930–943, 2003. DOI: 10.1109/TPAMI.2003.1217599.

[17] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o(n) solution to the pnp problem," *International Journal of Computer Vision*, vol. 81, Feb. 2009. DOI: 10.1007/s11263-008-0152-6.

[18] G. Pavlakos, X. Zhou, A. Chan, K. G. Derpanis, and K. Daniilidis, "6-dof object pose from semantic keypoints," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 2011–2018. DOI: 10.1109/ICRA.2017.7989233.

[19] Y. Zhu, L. Wan, W. Xu, and S. Wang, "Aspp-df-pvnet: Atrous spatial pyramid pooling and distance-filtered pvnet for occlusion resistant 6d object pose estimation," *Signal Processing: Image Communication*, vol. 95, p. 116 268, 2021, ISSN: 0923-5965. DOI: https://doi.org/10.1016/j.image.2021.116268. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0923596521001120.

[20] J.-K. You, C.-C. J. Hsu, W.-Y. Wang, and S.-K. Huang, "Object pose estimation incorporating projection loss and discriminative refinement," *IEEE Access*, vol. 9, pp. 18 597–18 606, 2021. DOI: 10.1109/ACCESS.2021.3054493.

[21] Z. Zhao, G. Peng, H. Wang, H.-S. Fang, C. Li, and C. Lu, *Estimating 6d pose from localizing designated surface keypoints*, 2018. arXiv: 1812.01387 [cs.CV].

[22] W. Zhao, S. Zhang, Z. Guan, H. Luo, L. Tang, J. Peng, and J. Fan, "6d object pose estimation via viewpoint relation reasoning," *Neurocomputing*, vol. 389, pp. 9–17, 2020, ISSN: 0925-2312. DOI: https://doi.org/10.1016/j.neucom.2019.12.108. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231220300333.

[23] W. Zhao, S. Zhang, Z. Guan, W. Zhao, J. Peng, and J. Fan, "Learning deep network for detecting 3d object keypoints and 6d poses," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 14 122–14 130. DOI: 10.1109/CVPR42600.2020.01414.

[24] J. N. Kundu, M. V. Rahul, A. Ganeshan, and R. V. Babu, "Object pose estimation from monocular image using multi-view keypoint correspondence," in *Computer Vision – ECCV 2018 Workshops*, L. Leal-Taixé and S. Roth, Eds., Cham: Springer International Publishing, 2019, pp. 298–313, ISBN: 978-3-030-11015-4.

[25] C. Chen, X. Jiang, W. Zhou, and Y. Liu, "Pose estimation for textureless shiny objects in a single rgb image using synthetic training data,"

ArXiv, vol. abs/1909.10270, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:202718955.

[26] Z. Cao, Y. Sheikh, and N. K. Banerjee, "Real-time scalable 6dof pose estimation for textureless objects," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2441–2448. DOI: 10.1109/ICRA.2016.7487396.

[27] N. Payet and S. Todorovic, "From contours to 3d object detection and pose estimation," in *2011 International Conference on Computer Vision*, 2011, pp. 983–990. DOI: 10.1109/ICCV.2011.6126342.

[28] M. Ulrich, C. Wiedemann, and C. Steger, "Combining scale-space and similarity-based aspect graphs for fast 3d object recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 10, pp. 1902–1914, 2012. DOI: 10.1109/TPAMI.2011.266.

[29] J. J. Moré, "The levenberg-marquardt algorithm: Implementation and theory," in *Numerical Analysis*, G. A. Watson, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 105–116, ISBN: 978-3-540-35972-2.

[30] Y. Konishi, Y. Hanzawa, M. Kawade, and M. Hashimoto, "Fast 6d pose estimation from a monocular image using hierarchical pose trees," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 398–413, ISBN: 978-3-319-46448-0.

[31] E. Muñoz, Y. Konishi, V. Murino, and A. Del Bue, "Fast 6d pose estimation for texture-less objects from a single rgb image," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 5623–5630. DOI: 10.1109/ICRA.2016.7487781.

[32] E. Muñoz, Y. Konishi, C. Beltran, V. Murino, and A. Del Bue, "Fast 6d pose from a single rgb image using cascaded forests templates," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4062–4069. DOI: 10.1109/IROS.2016.7759598.

[33] H. Tjaden, U. Schwanecke, and E. Schömer, "Real-time monocular pose estimation of 3d objects using temporally consistent local color histograms," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 124–132. DOI: 10.1109/ICCV.2017.23.

[34] E. Corona, K. Kundu, and S. Fidler, "Pose estimation for objects with rotational symmetry," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7215–7222. DOI: 10.1109/IROS.2018.8594282.

[35] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, *Implicit 3d orientation learning for 6d object detection from rgb images*, 2019. arXiv: 1902.01275 [cs.CV].

[36] F. Massa, B. C. Russell, and M. Aubry, "Deep exemplar 2d-3d detection by adapting from real to rendered views," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 6024–6033. DOI: 10.1109/CVPR.2016.648.

[37] 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," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 3848–3856. DOI: 10.1109/ICCV.2017.413.

[38] M. Oberweger, M. Rad, and V. Lepetit, "Making deep heatmaps robust to partial occlusions for 3d object pose estimation," in *Computer Vision – ECCV 2018*, V. Ferrari, M. Hebert, C. Sminchisescu, and Y. Weiss, Eds., Cham: Springer International Publishing, 2018, pp. 125–141, ISBN: 978-3-030-01267-0.

[39] J. Liu and S. He, *6d object pose estimation based on 2d bounding box*, 2019. arXiv: 1901.09366 [cs.CV].

[40] J. Liu and S.-F. He, "6d object pose estimation without pnp," *ArXiv*, vol. abs/1902.01728, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:59604476.

[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

[42] B. Tekin, S. N. Sinha, and P. Fua, "Real-time seamless single shot 6d object pose prediction," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 292–301. DOI: 10.1109/CVPR.2018.00038.

[43] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again," in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 1530–1538. DOI: 10.1109/ICCV.2017.169.

[44] Y. Hu, J. Hugonot, P. Fua, and M. Salzmann, "Segmentation-driven 6d object pose estimation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 3380–3389. DOI: 10.1109/CVPR.2019.00350.

[45] Z. Li, G. Wang, and X. Ji, "Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 7677–7686. DOI: 10.1109/ICCV.2019.00777.

[46] K. Park, T. Patten, and M. Vincze, "Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 7667–7676. DOI: 10.1109/ICCV.2019.00776.

[47] S. Zakharov, I. Shugurov, and S. Ilic, "Dpod: 6d pose object detector and refiner," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 1941–1950. DOI: 10.1109/ICCV.2019.00203.

[48] Z. Li, Y. Hu, M. Salzmann, and X. Ji, "Robust rgb-based 6-dof pose estimation without real pose annotations," *ArXiv*, vol. abs/2008.08391, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:221172799.

[49] L. Kästner, D. Dimitrov, and J. Lambrecht, "A markerless deep learning-based 6 degrees of freedom pose estimation for mobile robots using rgb data," in *2020 17th International Conference on Ubiquitous Robots (UR)*, 2020, pp. 391–396. DOI: 10.1109/UR49135.2020.9144789.

[50] X. Zhang, Z. Jiang, and H. Zhang, "Real-time 6d pose estimation from a single rgb image," *Image and Vision Computing*, vol. 89, pp. 1–11, 2019, ISSN: 0262-8856. DOI: https://doi.org/10.1016/j.imavis.2019.06.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0262885619300964.

[51] J. Liu, S. He, Y. Tao, and D. Liu, "Realtime rgb-based 3d object pose detection using convolutional neural networks," *IEEE Sensors Journal*, vol. 20, no. 20, pp. 11 812–11 819, 2020. DOI: 10.1109/JSEN.2019.2946279.

[52] Z. Yang, X. Yu, and Y. Yang, "Dsc-posenet: Learning 6dof object pose estimation via dual-scale consistency," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3906–3915, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:233181892.

[53] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 21–37, ISBN: 978-3-319-46448-0.

[54] H. Su, C. R. Qi, Y. Li, and L. J. Guibas, "Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2686–2694. DOI: 10.1109/ICCV.2015.308.

[55] J. Josifovski, M. Kerzel, C. Pregizer, L. Posniak, and S. Wermter, "Object detection and pose estimation based on convolutional neural networks trained with synthetic data," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6269–6276. DOI: 10.1109/IROS.2018.8594379.

[56] B. Li, W. Ouyang, L. Sheng, X. Zeng, and X. Wang, "Gs3d: An efficient 3d object detection framework for autonomous driving," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1019–1028. DOI: 10.1109/CVPR.2019.00111.

[57] W. Zou, D. Wu, S. Tian, C. Xiang, X. Li, and L. Zhang, "End-to-end 6dof pose estimation from monocular rgb images," *IEEE Transactions on Consumer Electronics*, vol. 67, no. 1, pp. 87–96, 2021. DOI: 10.1109/TCE.2021.3057137.

[58] P. Poirson, P. Ammirato, C.-Y. Fu, W. Liu, J. Kosecká, and A. C. Berg, "Fast single shot detection and pose estimation," in *2016 Fourth International Conference on 3D Vision (3DV)*, 2016, pp. 676–684. DOI: 10.1109/3DV.2016.78.

[59] A. Mousavian, D. Anguelov, J. Flynn, and J. Koecká, "3d bounding box estimation using deep learning and geometry," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5632–5640. DOI: 10.1109/CVPR.2017.597.

[60] B. Xu and Z. Chen, "Multi-level fusion based 3d object detection from monocular images," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2345–2353. DOI: 10.1109/CVPR.2018.00249.

[61] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, *Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes*, 2018. arXiv: 1711.00199 [cs.CV].

[62] S. Mahendran, H. Ali, and R. Vidal, "3d pose regression using convolutional neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2017, pp. 494–495. DOI: 10.1109/CVPRW.2017.73.

[63] T.-T. Do, M. Cai, T. Pham, and I. Reid, *Deep-6dpose: Recovering 6d object pose from a single rgb image*, 2018. arXiv: 1802.10367 [cs.CV].

[64] K. Hara, R. Vemulapalli, and R. Chellappa, *Designing deep convolutional neural networks for continuous object orientation estimation*, 2017. arXiv: 1702.01499 [cs.CV].

[65] J. Ku, A. D. Pon, and S. L. Waslander, "Monocular 3d object detection leveraging accurate proposals and shape reconstruction," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11 859–11 868. DOI: 10.1109/CVPR.2019.01214.

[66] J. Rambach, C. Deng, A. Pagani, and D. Stricker, "Learning 6dof object poses from synthetic single channel images," in *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, 2018, pp. 164–169. DOI: 10.1109/ISMAR-Adjunct.2018.00058.

[67] A. Kendall, M. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 2938–2946. DOI: 10.1109/ICCV.2015.336.

[68] J. Wu, B. Zhou, R. Russell, V. Kee, S. Wagner, M. Hebert, A. Torralba, and D. M. Johnson, "Real-time object pose estimation with pose interpreter networks," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6798–6805. DOI: 10.1109/IROS.2018.8593662.

[69] Y. Hu, P. Fua, W. Wang, and M. Salzmann, "Single-stage 6d object pose estimation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 2927–2936. DOI: 10.1109/CVPR42600.2020.00300.

[70] Y. Wang, S. Jin, and Y. Ou, "A multi-task learning convolutional neural network for object pose estimation," in *2019 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2019, pp. 284–289. DOI: 10.1109/ROBIO49542.2019.8961594.

[71] Y. Liu, L. Zhou, H. Zong, X. Gong, Q. Wu, Q. Liang, and J. Wang, "Regression-based three-dimensional pose estimation for texture-less objects," *IEEE Transactions on Multimedia*, vol. 21, no. 11, pp. 2776–2789, 2019. DOI: 10.1109/TMM.2019.2913321.

[72] C. Capellen., M. Schwarz., and S. Behnke., "Convposecnn: Dense convolutional 6d object pose estimation," in *Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2020) - Volume 5: VISAPP*, INSTICC, SciTePress, 2020, pp. 162–172, ISBN: 978-989-758-402-2. DOI: 10.5220/0008990901620172.

[73] G. Wang, F. Manhardt, F. Tombari, and X. Ji, *Gdr-net: Geometry-guided direct regression network for monocular 6d object pose estimation*, 2021. arXiv: 2102.12145 [cs.CV].

[74] A. Trabelsi, M. Chaabane, N. Blanchard, and R. Beveridge, "A pose proposal and refinement network for better 6d object pose estimation," in *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2021, pp. 2381–2390. DOI: 10.1109/WACV48630.2021.00243.

[75] Y. Hu, S. Speierer, W. Jakob, P. Fua, and M. Salzmann, *Wide-depth-range 6d object pose estimation in space*, 2021. arXiv: 2104.00337 [cs.CV].

[76] Y. Su, J. Rambach, A. Pagani, and D. Stricker, "Synpo-netaccurate and fast cnn-based 6dof object pose estimation using synthetic training," *Sensors*, vol. 21, no. 1, 2021, ISSN: 1424-8220. DOI: 10.3390/s21010300. [Online]. Available: https://www.mdpi.com/1424-8220/21/1/300.

[77] M. Oberweger, M. Rad, and V. Lepetit, "Making deep heatmaps robust to partial occlusions for 3d object pose estimation: 15th european conference, munich, germany, september 8-14, 2018, proceedings, part xv," in. Sep. 2018, pp. 125–141, ISBN: 978-3-030-01266-3. DOI: 10.1007/978-3-030-01267-0_8.

[78] 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," Oct. 2017, pp. 3848–3856. DOI: 10.1109/ICCV.2017.413.

[79] B. Tekin, S. Sinha, and P. Fua, "Real-time seamless single shot 6d object pose prediction," Jun. 2018, pp. 292–301. DOI: 10.1109/CVPR.2018.00038.

[80] P. Kamousi, S. Lazard, A. Maheshwari, and S. Wuhrer, "Analysis of farthest point sampling for approximating geodesics in a graph," *Computational Geometry*, vol. 57, Nov. 2013. DOI: 10.1016/j.comgeo.2016.05.005.

[81] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother, "Dsac - differentiable ransac for camera localization," Nov. 2016.

[82] E. Brachmann and C. Rother, "Learning less is more - 6d camera localization via 3d surface regression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2018.

[83] P. Pérez, M. Gangnet, and A. Blake, "Poisson image editing," *ACM Trans. Graph.*, vol. 22, pp. 313–318, Jul. 2003. DOI: 10.1145/1201775.882269.

[84] D. Dwibedi, I. Misra, and M. Hebert, "Cut, paste and learn: Surprisingly easy synthesis for instance detection," Oct. 2017, pp. 1310–1319. DOI: 10.1109/ICCV.2017.146.

[85] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba, "Sun database: Large-scale scene recognition from abbey to zoo," Jun. 2010, pp. 3485–3492. DOI: 10.1109/CVPR.2010.5539970.