Deep learning and camera calibration

An investigative study

YBJH8[1]

MSc Computational Statistics and Machine Learning

Dr Eddie Edwards

Submission date: September 2023

[1]**Disclaimer:** This report is submitted as part requirement for the MSc Computational Statistics and Machine Learning at UCL. It is substantially the result of my own work except where explicitly indicated in the text. *Either:* The report may be freely copied and distributed provided the source is explicitly acknowledged
Or:
The report will be distributed to the internal and external examiners, but thereafter may not be copied or distrbuted except with permission from the author.

Abstract

The process of camera calibration is a fundamental aspect of many computer vision tasks. However, the traditional methods of camera calibration are labour intensive and require a significant amount of human supervision. The aim of this project is to investigate whether deep learning based approaches can be applied in the context of camera calibration to streamline the process and simplify the time consuming procedure. The ultimate goal of such a project is to develop a fully deep learning based camera calibration process. Traditional methods and their development were researched religiously in the literature review. Throughout the project, many ideas from the deep learning field were explored along with trying to relate the task at hand to other tasks in deep learning(such as segmentation and regression tasks) along with specialist architectures (such as UNet and ResNet). Furthermore, the accuracy and robustness of traditional camera calibration methods were thoroughly studied, making use of a user generated synthetic dataset. The main contributions of the project are a 3D chessboard simulation system that allows to generate virtually unlimited amounts of labelled data for various tasks at hand and a hybrid camera calibration system between deep learning and traditional methods, making use of deep learning solutions to improve the robustness of the traditional method.

Contents

Chapter 1

Introduction

1.1 Computer Vision

According to Professor Li Fei Fei, a Chief AI Scientist, at Google [8] computer vision is considered the "killer app" of AI. This field encompasses areas such as image understanding, robot vision, machine vision, computational vision, image analysis and scene analysis. The goal is to mimic vision and develop computer systems that can truly "see" by focusing on the abstraction of computations than just numerical calculations [0].

Creating a seeing machine" has been a challenging task that required the combined efforts of numerous brilliant individuals over the years [0]. Despite its complexity in integrating data computer vision has led to the development of various applications. These include document processing, remote sensing, radiology, microscopy, industrial inspection and robot guidance [0]. The field of computer vision plays a role in analyzing images to provide descriptions of real world objects. It finds applications in robotics, industrial automation, document processing, biometrics, automotive industry remote sensing technology navigation systems microscopy medical imaging augmented reality. Even creating aids, for visually impaired individuals [0] [0]. Computer vision, which is a subfield of intelligence involves the processing of images and patterns through techniques such, as image processing and pattern recognition. Before the cognitive aspect comes into play there is a need for preprocessing [**?**] Computer vision encompasses areas like intelligence, computer graphics and image processing along with pattern recognition [0] [0]. The strategies employed to address challenges in computer vision depend on the application domain and the type of data being analyzed [0]. While computer vision has seen growth recently and is used for tasks like analyzing content such as QR codes it still faces challenges related to

computational costs and processing speed when it comes to embedded systems or mobile devices [0]. Although computer vision shares similarities with vision in terms of perceiving data there are limitations that affect its performance including sensitivity to parameters, algorithm capabilities and precision [0]

In terms of data types used in computer vision systems various array types are employed including symbolic arrays, arrays feature maps and numerical arrays to represent input images. Additionally complex methods such as coordinates run length codes boundary codes, quadtrees or medial axis transformations can be utilized. Data structures like surface models and solid models help express components, within an image well as their attributes and relationships.

Labelled graph architectures offer a way to represent information as mentioned by Rosenfeld in 1988.

1.2 Camera Model

1.2.1 The Projective or Pinhole Camera:

Simply put a pinhole camera is a camera that doesn't have a lens and only has an opening. When light passes through this opening it creates inverted images, on the side of the camera [0]. The pinhole calibration algorithm is based on the Jean Yves Bouguet model [10] which's purely geometric and explains how points in the world are projected onto an image. Imagine a virtual image plane positioned in front of the camera that contains a representation of the scene (Figure 1.1) [0].

The pinhole camera model effectively establishes a connection between positions in an image and corresponding locations in the world. The aim of this model was to address the issue of sparse stereo reconstruction (Figure 1.2). Two images of an object were taken from angles and the goal was to identify 2D features that corresponded to points representing identical 3D positions, in both photos. These observed 2D feature points were then used to reconstruct their 3D locations.

Robots could potentially navigate through a given environment effortlessly. Identify objects, with the assistance of the generated three dimensional data [0].

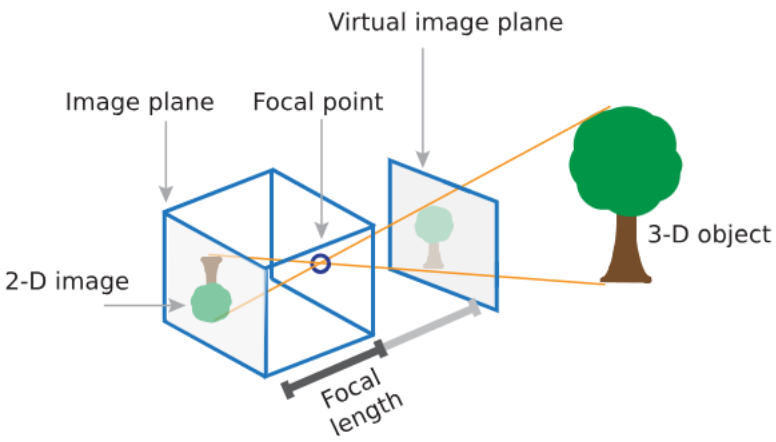


Figure 1.1: Pinhole Camera model [0]

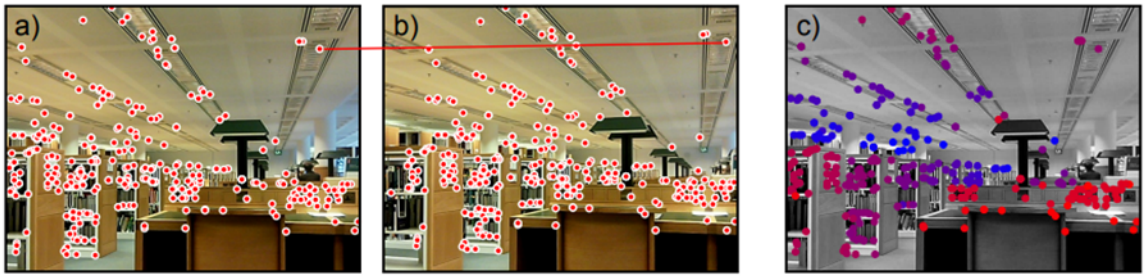


Figure 1.2: The stereo reconstruction in the photographs b) A pair of I pairs of points that correlate to the same points in the real world. c)Here, closer points are red and further points are blue using colour to represent depth. [0]

1.2.2 Alternative Camera Models

The concept of an Orthographic Model involves projecting points onto a plane, which can be adjusted through rotation scaling and translation. This model is particularly useful, in fields as it offers measurements of 3D objects without any perspective distortion (Julia et al., 2019).

On the hand the fisheye camera Model combines a camera with a fisheye lens. One advantage is that fisheye cameras don't have areas like cameras do. Additionally the use of a fisheye lens doesn't significantly increase the size or vulnerability of the imaging system [15]. Fisheye lenses have lengths and wide fields of vision enabling them to capture expansive surroundings in a single photograph. For applications that require photographs with a 180 degree field of view specific geometric models are necessary [0].

Lastly an omnidirectional camera utilizes a CCD camera to capture a field of view that includes the entire floor from its position. However direct usage of the resulting image can be challenging due, to distortion.

The front end layer processes down images, from cameras to create a unified view of the surroundings. This involves transforming image pixels from coordinates to a coordinate system [0].

One type of range imaging sensor called Time of Flight (ToF) camera captures images of the environment using infrared (NIR) light and a CMOS image sensor in real time. By measuring the time it takes for light to travel the depth of each pixel can be calculated. ToF cameras have applications, including localization, positioning, robotics, human machine interface and scene modeling. They are known for their power consumption, affordability, compact size and ability to capture images at a video frame rate of approximately thirty frames, per second [0].

1.3 Camera calibration

Camera calibration is a crucial step in computer vision activities like industrial photogrammetry, 3D reconstruction, and image-based rendering. It involves mapping the relationship between the 3D environment and 2D pictures using a camera model. Traditional approaches require multiple pictures due to their reliance on image relationships [12]. The most well-known of them is Zhang's plane camera calibration method based on the template [0]. Camera calibration is crucial in 3D reconstruction, as traditional methods cannot handle unreliable images from sources like the internet or newspapers. The calibration model, which includes camera and projector models, describes the perspective projection from the 3D environment to the 2D image planes. The accuracy of this model's depiction of projection processes significantly impacts calibration accuracy. Camera research has advanced modeling and calibration methods over the years [12].

Furthermore, a camera calibration technique that uses a straightforward reference item is good in actual scenarios because it can swiftly react to different camera combinations both inside and outside. Therefore, examining camera calibration with just one image is crucial for computer vision [0]. Camera calibration research began in photogrammetry, but subsequent studies in computer vision have extensively explored this topic [0].

3D computer vision relies on camera calibration for metric information extraction, which can be categorised into self-calibration and photogrammetric calibration [0].

Three-dimensional reference object-based calibration: The camera calibration is done by observing a calibration object whose geometry is known very precisely in 3D space. It is highly efficient to calibrate. Two or three orthogonal planes typically make up the calibration object. It is also possible to employ a plane that is being translated exactly. These techniques call for a pricey calibration tool and a complex setup [0].

Self-calibration: The methods do not require a calibration item and can recover both internal and exterior parameters from a single camera displacement in a static environment. This allows for reconstruction of 3D structure up to similarity when images are captured by the same camera with fixed internal parameters. However, this method is not fully developed and may not always yield solid results due to numerous parameters to estimate [0].

Automated zoom lenses for machine vision require reliable camera models across various lens settings. Most camera calibration methods are for fixed-parameter lenses, where the imaging process is static. Automated zoom lenses generate pictures dynamically based on lens control settings. Calibration and modeling of these lenses are more challenging than fixed-parameter lenses due to the complexity of control parameter relationships and the need to calibrate them over a range of lens settings [0].

1.3.1 Pinhole Camera at Infinity

The pinhole camera is the illustration that is most frequently used to describe the geometry of a traditional camera. The camera lens is absent from this model, and it is assumed that all of the rays pass via the camera centre. This is unmistakably a perspective projection of the points in space to the sensor plane, with the camera's centre acting as the projection's centre. With a linear function determined by the projection matrix P = K[Rt], this projection is often expressed in homogeneous coordinates for convenience [0].

The camera matrix, a 3-by-4 matrix that represents the pinhole camera specifications. The 3D world scene is mapped onto the picture plane via this matrix. The extrinsic and intrinsic characteristics are used in the calibration procedure to compute the camera matrix. The extrinsic parameters describe where the camera is in the three-dimensional scene. The optical centre and focal length of the camera are represented by the intrinsic parameters [0].

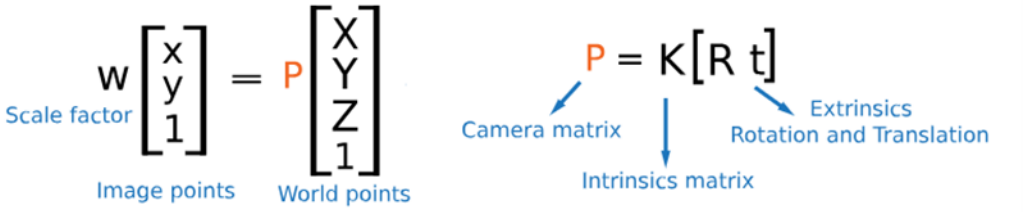


Figure 1.3: The world points are converted to camera coordinates using the intrinsic parameters. The intrinsic parameters are used to map camera coordinates into the image plane [0]

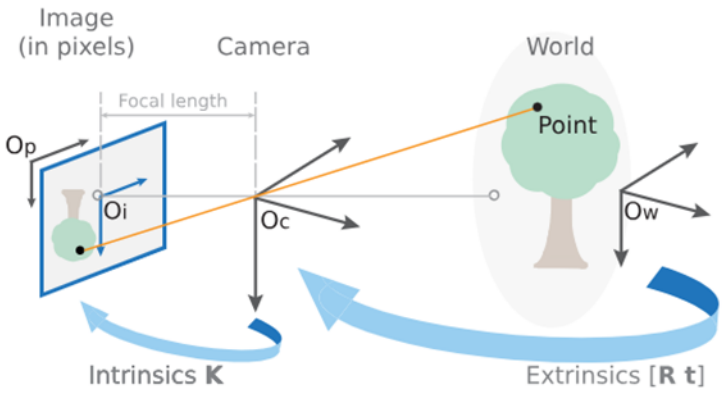


Figure 1.4: Pinhole Camera Projection [0]

1.3.2 Geometric Camera Calibration

The camera may be positioned anywhere in the world and can be pointed in any direction. The task is to ascertain how the points from the world space are projected onto the picture plane. To put it another way, the connection between worldly points and the equivalent pixels on the camera's picture must be determined [0]. To accurately convert 3D environments into digital photos, it's crucial to understand the camera's inherent properties. Users may not have access to these settings if given a random camera but can infer them from images. Calibration involves calculating intrinsic and extrinsic camera characteristics [0].

Camera calibration involves calculating the intrinsic matrix (K) and posture (R, t), with intrinsic calibration estimating K and extrinsic calibration estimating R and t. Additional factors may be used to model radial or other distortions. There are two basic methods depending on data type [0].

- The switch from the camera coordinate system to the world coordinate system as the basis for transformation. In essence, this enables us to perceive the world through the camera's vantage point. The term "camera pose" or "camera extrinsics" refers to this [0]. It enables the mapping of camera coordinates in the picture frame to pixel coordinates. Examples include the lens' optical centre, focal length, and radial distortion coefficients [6].

- The transformation that projects the points in the surrounding environment that are visible to the camera onto the picture plane of the camera. The term for this is camera intrinsics [0]. It defines the placement and orientation of the camera. With respect to a global coordinate system, this describes the rotation and translation of the camera [6].

Already stated earlier, that the camera model has two distinct types of parameters: exterior parameters, which link camera orientation to the object coordinate frame, and internal parameters, which control how the camera coordinate frame is projected onto picture coordinates. The calibration procedure, often involving non-linear optimization, estimates both parameters, using the sum of squared distances between measured and modelled control point projections as the cost function [[0].

This might be used to depict a standard pinhole camera a 3×4 homogeneous matrix where P = K[Rt]. It is referred to as the camera projection matrix. Camera P is referred to as a finite projective camera if the left-hand submatrix KR is non-singular, as it is for perspective cameras. General projective cameras are defined as cameras represented by arbitrary homogeneous 3 x 4 matrices of rank 3 [0].

Pinhole camera calibration can be easy or challenging, depending on the circumstances and accuracy required. Accuracy and managing complex distortions can be more challenging [0].

Pinhole Camera Calibration (easy): A pinhole camera may be easily calibrated for elementary tasks like comprehending fundamental perspective projection. This often entails figuring out the camera's inherent properties, such as focal length and primary point, using a few well-known calibration procedures [0]. 2. Multiple Camera Systems (Challenging): Multi-camera systems, whether stereo or multicamera, introduce additional complexity, such as the need to determine the extrinsic parameters (relative pose) [0].

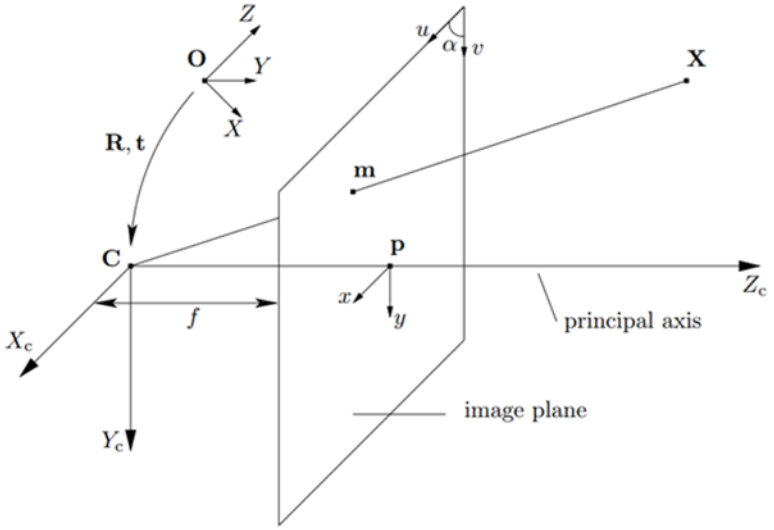


Figure 1.5: Pinhole camera model. In this instance, C serves as both the camera's centre and the coordinate frame's starting point. The normalised picture coordinate system (x, y) originates at the primary point p. The coordinate system for pixels is (u, v) [0]

Distortion and More Complex Scenarios (Challenging): There are numerous types of lens distortion that can occur in real-world scenarios including radial and tangential distortions. It can be challenging and laborious to calibrate for non-idealities and handle distortions [0].

Simultaneous localization and mapping (SLAM)(Challenging): Simultaneous localization and mapping (SLAM) is a challenging problem in robotics and computer vision. It encompasses a robot or piece of equipment traversing uncharted territories while also mapping that territory and determining its location on that map. SLAM is regarded as difficult due to the complexity of integrating sensor data, the requirement for real-time processing, and the possibility of accumulated inaccuracies [0].

3D Reconstruction (Challenging): Achieving great accuracy and managing complicated situations while employing pinhole cameras for 3D reconstruction projects can be difficult. For a precise 3D reconstruction, calibration is a key step [0].

Chapter 2

Literature Review

2.1 Camera Calibration

Camera calibration, in machine vision plays a role in converting the three world into a two dimensional image. Despite its complexity this process may not always completely eliminate errors. Therefore to ensure accurate measurements of locations and dimensions within the object space based on the resulting images it becomes essential to calibrate the camera system.

To grasp how camera calibration functions it's important to understand the components of a camera. A camera consists primarily of an image plane and a lens. The lens acts as a link between the entities in the three object space and their representation in the two dimensional image captured by the camera.

In essence camera calibration involves determining values for both extrinsic parameters of the camera system. This minimizes. Provides measurements for computer vision tasks. The intrinsic parameters deal with aspects of the camera such as length, principal point and lens distortion coefficients. On the hand extrinsic parameters describe how the camera is positioned and oriented relative, to a coordinate frame. Determining these parameters accurately allows the camera to create a map matching perspective projection points, in the object space with points on its image plane [0]. Camera calibration has applications in computer vision and computer graphics. It serves functions such as correcting lens distortions measuring real world object sizes, from their image representations and precisely locating the camera within a scene.

This literature review aims to delve into aspects of camera calibration and its development. It will explore pioneering work in the field, different camera models and techniques

challenges faced during calibration and recent advancements. The goal is to gain an understanding of this field to apply it effectively to our project.

2.1.1 Historical Background

Exploring the evolution of camera calibration provides an important insight into the field of machine vision, different approaches taken by pioneers in the field can help determine its development trajectory.

The Genesis of Camera Calibration : Camera calibration was initially created to ensure accuracy in measurements derived from photographs. The initial approach to this problem was through bundle adjustment, where a system of simultaneous equations is formulated to calculate parameters that define the geometric relation between the camera and the object space [0]. However, In 1965, Brown presented some pioneering work with bundle adjustments (which he later published in 1966 'Decentering Distortion of Lenses') [11] which catapulted the widespread application of bundle adjustment in this field, especially concerning terrestrial photographs.

Self-Calibration: By 1972, a wave of research papers began highlighting the potential of the self-calibration method for aerial photographs. This technique was able to use all image observations from different camera perspectives to determine unknown lens parameters. This prompted the International Society for Photogrammetry and Remote Sensing (ISPRS) to establish a Working Group in 1976 to explore this innovative technique further [11].

Additional Parameters: Brown's work during the 1960s and 1970s laid the foundation for "additional parameters", which improved the mathematical model for calibration [0]. Many of these parameters were not observable physical phenomena. They were integrated into the model mainly because they significantly minimized errors on the photographic plates [0]. However, the excessive use of these additional parameters led to a situation of 'overparameterisation' which - as Fraser pointed out in 1982 - could weaken the solution for target point coordinates [21].

On-the-Job Calibration: Emerging from the developments of the bundle adjustment was the "on-the-job" calibration. This method became widely accepted for close-range

camera calibration, where additional parameters are factored into account for lens distortion, focal length, and other potential unknowns [11]. The principle of this calibration lies in its application, where control points are placed strategically around the object of interest.

Setting the Global Standard: As camera calibration techniques started to frequently evolve and have significant impacts in the field, international bodies agreed to set standardized benchmarks. By the 1980s, the International Organisation for Standardisation consolidated its committees to cater specifically to the calibration needs of photographic cameras [0]. Simultaneously, organizations (like ISPRS) played a crucial role in the recommendations and procedures for calibration.

Modern Methods and Innovations: In the later part of the 20th century the development of new algorithms and methodologies started to emerge. In 1986, Ziemann proposed a detailed algorithm for camera calibration, emphasizing a systematic approach to obtaining comparable results across calibration authorities. [0]

To conclude, the journey of camera calibration has been marked by consistent advancements, with each era contributing significantly to refining methodologies and enhancing accuracy. These historical insights allow us not only to understand its developmental trajectory, but also to build upon it.

2.1.2 Camera Models

Pinhole Camera Model

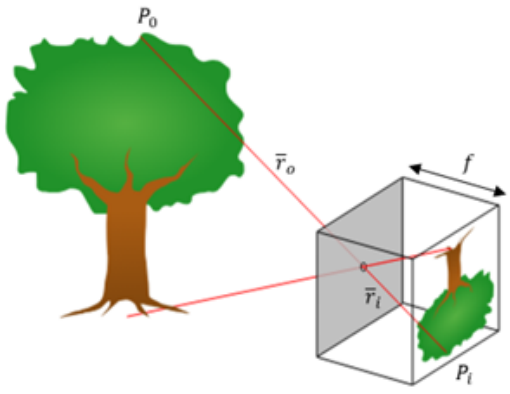


Figure 2.1: Pinhole Camera model [3]

One of the fundamental camera models is the Pinhole Camera model. The name is derived from the architecture of this model: where an opaque barrier between the image plane and 3D-scene is perforated (the size of a pinhole) to allow light to pass through between the 2 mediums. A particular point in the scene, denoted as P_0, casts a single light ray through the pinhole, projecting onto the image plane at a point P_i. This makes the geometry constrained: each point in the scene maps to a unique point on the image. [0] Consider a 3-dimensional coordinate frame (XYZ) with the origin at the pinhole. The z-axis lies on the optical axis and is oriented perpendicular to the image plane. Where the distance between the pinhole and the image plane is known as the effective focal length f. The point P_0 in the scene - as shown in the Figure 2.1 - can be represented as $\bar{r}_o = (x_o, y_o, z_o)$. Its image P_i on the image plane is denoted as $\overline{r_i} = (x_i, y_i, f)$.Notably, the z-coordinate for all points on the image plane will invariably be f. Due to the geometric similarity between certain triangles formed by the pinhole, the scene, and the image plane, a crucial relationship emerges: $\frac{\bar{r}_i}{f} = \frac{\bar{r}_o}{z_o}$ Parsing this vectorial equation we get the individual component relationships, which are known as the equations of perspective projection:

$$\frac{x_i}{f} = \frac{x_o}{z_o}$$
$$\frac{y_i}{f} = \frac{y_o}{z_o}$$

Which illustrates how a 3D point is mapped onto a 2D plane.[0]

Camera's Intrinsic Parameters: The intrinsic parameters of a camera are important in deciding how 3D points are projected onto the 2D image plane. Key among these parameters is:

- Focal lengths ($\boldsymbol{f_x}$ and $\boldsymbol{f_y}$) : These values can vary in the x and y directions due to the camera's anisotropic pixel size or the lens's non-uniform magnification properties.
- Principal point ($\boldsymbol{c_x}, \boldsymbol{c_y}$) : This is the point on the image plane where the optical axis intercepts. It's commonly assumed to be the image centre but may change due to manufacturing inconsistencies or intentional design.
- Skew coefficient: In most real-world cameras the pixel grid is rectilinear. Still there can be a slight non-perpendicularity between the two axes, represented by the skew coefficient. However, for most practical applications this is considered negligible and is set to zero. [0]

Camera's Extrinsic parameters: These parameters describe the pose of the camera in the world coordinates, encapsulated by the rotation matrix and the translation vector. The rotation matrix's degrees of freedom shows the possible orientations a camera can adopt. Typically, a 3×3 rotation matrix has 9 components but only 3 degrees of freedom due to the orthogonality and unit determinant constraints.[0]

Homogeneous Coordinates: In projective geometry, homogeneous coordinates offer a way to represent points at infinity and simplify mathematical expressions. A 2D point (x, y) is represented as $(x, y, 1)$, while a 3D point (X, Y, Z) is depicted as $(X, Y, Z, 1)$. These augmented coordinates facilitate matrix multiplication for transformations and projections.[20]

In the context of my project, to streamline the mathematical formulation and its subsequent computations, I assumed where the distortion is considered null. This approximation removes the complications induced by radial and tangential distortions, simplifying the pinhole camera model.

Lens Distortion Models

While the pinhole camera model explains the idealistic perspective projection, it does not consider the imperfections introduced by real-world lenses. These imperfections often appear as distortions, mainly as radial and tangential distortions.

Radial Distortion creates an alteration in the image magnification: leading to either barrel distortion - where the image appears to be "bulging" outwards - or pincushion distortion with the image appearing "pinched" inwards [0]. The distorted coordinates ($x_{\text{distorted}}$, $y_{\text{distorted}}$) can be expressed in terms of the undistorted coordinates (x, y) as:

$$\begin{aligned} x_{\text{distorted}} &= x\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \\ y_{\text{distorted}} &= y\left(1 + k_1 r^2 + k_2 r^4 + k_3 r^6\right) \end{aligned}$$

Where:

$r^2 = x^2 + y^2$ represents the squared distance from the origin in the image plane, and k_1, k_2, and k_3 are the radial distortion coefficients.[0]

Tangential Distortion occurs due to the misalignment of the lens elements or the image sensor, causing the image to appear skewed. These attributes can be represented through

the following equation which encapsulates both radial and tangential distortions:

$$x_{\text{corrected}} = x_{\text{distorted}} + \left[2p_1xy + p_2\left(r^2 + 2x^2\right)\right]$$

$$y_{\text{corrected}} = y_{\text{distorted}} + \left[p_1\left(r^2 + 2y^2\right) + 2p_2xy\right]$$

Where p_1 and p_2 are the tangential distortion coefficients which provide the degree of skewness [0].

Fisheye Camera model

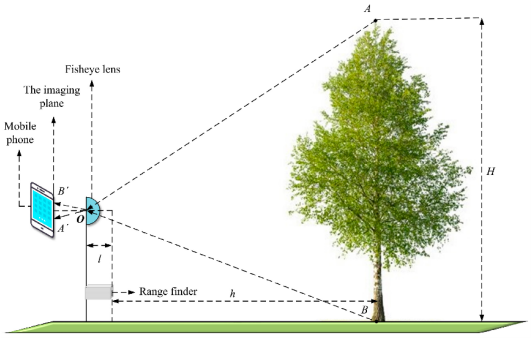


Figure 2.2: Fisheye Camera model [0]

The fisheye model further expands on the elementary pinhole model. Central to its understanding are the frames $\mathcal{F}_c$ (camera-centric) and $\mathcal{F}_i$ (image-centric). These frames interpret the relationship of a point $\mathcal{M}$ with coordinates $\mathrm{M} = (XYZ)^T$. When this point is projected, it has perspective coordinates, $\mathrm{m_p}$, and its fisheye representation, $\mathrm{m_f}$.[15]

While The pinhole (perspective) projection, is described by:

$$r_p(\theta) = f \tan\theta$$

where f is the focal length. Fisheye cameras inherently produce radial distortions. These are summarized by two fixed constraints:

- A ray coming directly along the principal axis remains undeformed, so $r_f(0) = 0$.
- As the angle k off the principal axis increases, the radial distance $r_f(k)$ from the principal point must increase monotonically.[15]

From these foundational principles, two primary models are derived:

Pinhole-derived Models, which translate the conventional perspective projection equations to accommodate the fisheye's unique radial distortions. Here is a brief example [15]:

$$r_f^1\left(r_p\right) = r_p L\left(r_p, n\right)$$

$$r_f^5\left(r_p\right) = s\log\left(1 + \lambda r_p\right)$$

Ray Capture Models, due to the fisheye's ability to capture rays at different angles, further models can express the projected radial distance in relation to the incoming angle θ. Key equations involve [15]:

$$\begin{aligned} r_f^1(\theta) &= f \\ &\vdots \\ r_f^5(\theta) &= f\left(k_1\theta + \cdots + k_n\theta^{2(n-1)+1}\right) \end{aligned}$$

In summary, the fisheye camera model presents an intricate relationship between the pinhole projection and its nuanced, angle-sensitive counterparts.

2.1.3 Camera Calibration Techniques

2D Calibration

Planar Patterns (Efficacy and Methods): Planar patterns use the geometry of flat structures. Using the camera's intrinsic and extrinsic parameters known 3D points can be related to 2D points. One of the most prominent methods using this technique is the Zhang's method [0], where multiple images of a planar pattern, such as a checkerboard help determine the camera parameters. The relationship between 3D world coordinates (X, Y, Z) and 2D image coordinates (x, y) can be expressed as:

$$s\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = P\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

Where: P is the projection matrix and s is an arbitrary scale factor. Zhang's approach determines P by capturing images of the planar pattern from various orientations.[0]

Vanishing Points (Extracting Camera Parameters): Vanishing points give insight into the direction of the world's coordinate axes. This is a more implicit method where parallel lines seem to meet in an image. Mathematically, the vanishing point v for a set of parallel lines (defined by direction d) can be established as:

$$v = KRd$$

Where:

- K is the camera's intrinsic matrix (contains parameters like focal length and principal point).
- R is the rotation matrix that defines the orientation of the world coordinates with respect to the camera coordinates. [0]

3D-2D Correspondence Calibration: Connects between known 3D world coordinates and 2D image coordinates. The DLT (Direct Linear Transformation) algorithm computes the projection matrix P from a set of known 3D points and their corresponding 2D projections. Using the equation same equation mentioned in Planar Patterns (Efficacy and Methods). However - in contrast with the Zhang method - DLT finds P by solving this equation for multiple correspondences and minimizing the reprojection error.

3D Calibration

Utilising 3D Calibration Objects: such as cubes or specially designed calibration rigs, have well-defined 3D structures that allow for accurate depth determination. The transformation between the world and camera coordinates can be determined by correlating the 3D points on the calibration object to their 2D projections on the camera's image.[20] Considering a 3D point P_w in the world and its projection p on the camera the following can be formulated:

$$p = K[R \mid t]P_w$$

Where:

- p is the homogeneous coordinate of the 2D point in the image plane.
- P_w is the homogeneous coordinate of the 3D point in the world.

- K is the intrinsic matrix (parameters like focal length, aspect ratio, and principal point coordinates).
- R is the rotation matrix.
- t is the translation vector, together describing the pose of the camera in world coordinates.[20]

One classic approach involves using a calibration object with easily identifiable feature points, such as a cube with marked corners. By rotating and translating the object in front of the camera and capturing images at each orientation, we obtain a series of 3D-to-2D correspondences that can be used to solve the above equation.[20]

Free-Form Calibration forgoes traditional structured patterns and leverages unstructured data from varied scenes. By capturing a diverse array of features, this method facilitates camera calibration. However, the technique's accuracy is contingent on the richness of observed features. The primary focus is determining 3D-2D correspondences across random scenes to deduce camera parameters.[20]

Self-Calibration

Calibration without explicit knowledge of calibration pattern: Self-calibration techniques unlock the potential to determine intrinsic and extrinsic camera parameters without the reliance on traditional aids (such as chessboards). This method holds merit particularly in scenarios, where introducing or managing a calibration pattern is impractical or disruptive. Therefore, allowing Images captured from various positions and angles to be processed to extract consistent features.[0]

Using scene features, motion, or other cues: self-calibration becomes even more compelling when one considers its ability to exploit naturally occurring features in a scene (landmarks or distinct scene elements). These static features offer a foundation for calibration. Furthermore, motion presents another avenue. If the camera's motion - even if not strictly uniform - can be deduced or captured, it offers another dimension of data for calibration. This methodology builds upon the principle that while the environment's structure remains invariant, the camera's perspective undergoes shifts, and these shifts can be algorithmically processed to derive essential calibration parameters.[0]

Camera Array and Stereo Calibration

Multi-camera calibration is primarily concerned with determining the extrinsic parameters which describe the spatial relationships between cameras, and the intrinsic parameters which relate to individual camera properties.

Stereo Calibration: Stereo systems, comprising two cameras, are the simplest form of multicamera systems. For accurate depth estimation, it is critical to determine the relative pose and orientation between the two cameras. The fundamental matrix F and the essential matrix E play pivotal roles. Given two corresponding points x_1 and x_2 in two images, their relationship can be defined as: $x_2^T F x_1 = 0$ The essential matrix (E) relates the corresponding points but is defined in terms of cameracentric coordinates. It is given by: $E = K'^T F K$ Where K and K' are the intrinsic matrices of the two cameras.[0]

Camera Arrays: As we progress from stereo to multiple cameras, the calibration process seeks to find both intrinsic parameters for each camera and the relative extrinsic parameters between cameras. Calibration techniques usually involve capturing images of a known calibration pattern from all cameras simultaneously and determining overlapping fields of view [0]. Tools like Voodoo Camera Tracker or Bundler have been developed to assist in these calibrations, utilizing feature matching across multiple views to refine camera parameters.[0]

2.1.4 OpenCV's Calibration Module

OpenCV is an open-source computer vision library which offers a comprehensive module (*calib3d*) for camera calibration and 3D reconstruction.[0] This literature review aims to highlight some of the most notable functionalities and their use (for a deeper dive visit the official OpenCV documentation)

As mentioned previously, camera calibration is the process of estimating the intrinsic, extrinsic, and lens-distortion parameters of a camera. The OpenCV calibration module provides the following to achieve this:

- Chessboard Corner Detection: The function *findChessboardCorners* assumes a vital role here. Chessboards with consistent and distinct geometric design are preferred during the calibration process. This function adeptly identifies the corners of these patterns, paving the way for further calibration steps.

- Parameter Estimation: Upon establishing point correspondences, *calibrateCamera* estimates the camera parameters, both intrinsic and extrinsic.
- Stereo Camera Orientation: *stereoCalibrate* comes into play for those delving into stereo camera setups. It sheds light on the relative orientation and position of the two cameras.[0]

It also important to mention the *solvePnP* function. Beyond traditional calibration, this function holds the capability to estimate the pose of a recognized 3D object bridging 3D-2D point correlations.[0]

In the realm of projection, the module does not falter:

- *projectPoints* emerges as a tool post-calibration, mapping 3D points onto the image plane, revealing their projected locations.
- For the 2D counterparts, *perspectiveTransform* is employed to leverage a transformation matrix designed for this very purpose.

With stereo vision making strides in modern applications, image rectification often becomes paramount after calibration. *stereoRectify* focuses on rectifying images derived from stereo pairs. The intertwined functions, in *itUndistortRectifyMap* and *remap*, cater to deriving rectified images from those that were previously distorted.

Additionally, the *calib3d* module is very versatile. It can grapple with various distortion models, including radial and tangential. The undistort function epitomizes this capability, offering images that resonate closely with real-world visuals.[0]

OpenCV's calibration process is largely contingent on the extraction of chessboard corners. While algorithms, such as the PnP hold significance - especially seen in the *solvePnP* function - the crux is anchored in the chessboard images and their translation to quantifiable camera parameters.

Areas of Caution with OpenCV: Although this module facilitates various applications in the realm of camera calibrations, it is imperative to be aware of the shortcomings to ensure an optimal outcome. The angle at which the chessboard is presented effects the calibration accuracy, where extreme angles could potentially lead to skewed results. The library's performance wavers when processing low quality images, especially those that are grainy. Additionally, accurate calibration can only be achieved through a complete and

unobstructed view of the chessboard. It is also worth noting, OpenCV performs better with specific chessboard colours which are encased in a white frame. Finally, despite the advanced capabilities of the technology, the human element retains its importance in the meticulous task of image validation.

Within the camera calibration sphere, OpenCV's *calib3d* module stands out, owing to its multifaceted features. Backed by an open-source foundation and a robust community, the module is poised for continuous advancement. It boasts compatibility with a range of camera models and calibration patterns, illustrating its versatility. Additionally, its seamless assimilation with the overarching OpenCV library facilitates a smooth transition from calibration tasks to broader application implementations, underscoring the pivotal role of OpenCV in the domain.

2.1.5 Challenges in Camera Calibration

Handling lens distortions: Lens distortions can considerably hamper the accuracy of vision-based algorithms. Contrary to the idealized models, real-world lenses inherently bend and distort incoming light. This causes the projected image to deviate from the expected representation. Radial distortion, for instance, either "pushes" pixels outward or "pulls" them inward [0]. Therefore, the challenge lies in accurately determining the radial distortion coefficients. For the mathematical representation visit *Lens distortion models*.

Calibration for wide-angle and fisheye lenses: This is particularly daunting due to their expansive field of view. Traditional camera models which are designed for regular lenses, fail to encapsulate the optical characteristics of these lenses [0]. This means entirely different mathematical models or even intricate higher-order polynomials might be required for an accurate representation. The challenge compounds when considering the entire field of view of such lenses, where distortions can be extreme.

Noise and inaccuracies: The precision required in camera calibration is often hindered by sensor noise, manufacturing tolerances in lenses, and minute misalignments [0]. Even in controlled environments determining the exact 3D-2D correspondences can be susceptible to these minor errors. The Noise is typically modelled as Gaussian $\epsilon \sim N(0, \sigma^2)$. Ensuring that this noise doesn't significantly sway the calibration parameters becomes pivotal.

Real-time calibration challenges: In cases where instantaneous feedback is required - such as in augmented reality or drone navigation - the luxury of extensive computational time isn't available. Traditional calibration methods, designed for accuracy, might not fit the bill for real-time requirements [0]. Thus, there's an intricate balance that must be struck between speed and accuracy, which becomes even more challenging with the complexities introduced by the previously mentioned challenges.

2.2 Neural Networks

In the academic domain of camera calibration, advancements in computational techniques have persistently refined our ability to achieve enhanced precision, operational efficiency, and adaptability. With the integration of deep learning, the role of neural networks has come to the forefront.

The neural network design was established aimed at pattern recognition. As the name suggests, it draws inspiration from the structural and functional attributes of the human cerebral system. At its core, it consists of a series of interconnected units, often referred to as 'neurons' or 'nodes'. These are delegated with the task of processing information. They accomplish this by assigning specific weights to their respective inputs, channelling these through a distinct activation function, and subsequently generating an output. It is this nuanced processing of inputs that enables the neural network to recognize and learn from data patterns.[0]

Delving deeper into the architecture of neural networks, one must appreciate the significance of 'layers'. A conventional neural network is demarcated into three distinct sections: an input layer, multiple intermediary or hidden layers, and a concluding output layer. Each of these layers is populated with a specific number of nodes [0]. The process begins with data being fed into the input layer, subsequently traversing through the hidden layers, and culminating at the output layer, thus completing the information flow. Mathematical representation of a single neuron's output, given input vector x and weight vector w, can be described by:

$$y = f(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$

Where: f is the activation function and b is the bias. [0]

Neural networks have seen diverse applications. In the context of camera calibration, they're primarily utilized for image recognition. These are some of the fields that employ

neural networks: camera calibration, natural language processing, medical diagnostics, and financial forecasting.[0]

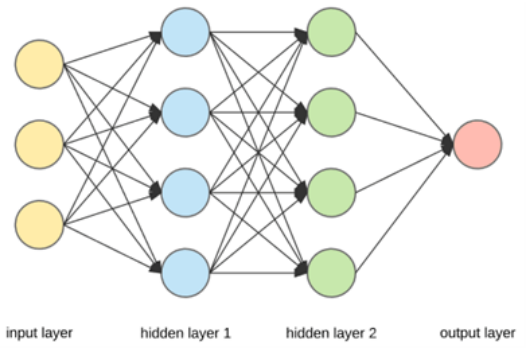


Figure 2.3: Neural Network architecture [5]

2.2.1 Historical Background

Neural networks are an influential pillar in the domain of artificial intelligence, which underwent considerable development since their conception. The journey of neural networks encompasses innovations, regressions, and a modern-day renaissance.

Rosenblatt's Perceptron: The initial conception of neural networks was proposed by Frank Rosenblatt in 1957, in the paper called "Perceptron" [0]. The perceptron was proposed as a mathematical model mimicking the neuron's basic functions. Mathematically, it is represented as: $y = f(w \cdot x + b)$

Where: w is the weight vector, x is the input vector, b is a bias, and f is an activation function, commonly the step function during the initial perceptron design.[0]

Rosenblatt's perceptron was designed for binary classification and was advertised as a promising beginning for machine learning.

Al Winter: However, in 1969, Minsky and Papert published a book highlighting the model's limitations. Particularly its inability to solve non-linear problems like the XOR problem [0] therefore, casting doubt in the model proposed by Rosenblatt. After noticing this flaw along with other factors, a catalytic effect ensued that led to reduced funding and interest in neural network research. Hence, marking the start of the "Al Winter" period.

GPUs and Big Data: The 1980s saw a revival of neural networks, primarily attributed to the backpropagation algorithm, enabling the training of multi-layer perceptron [0]. The algorithm minimized the difference between the predicted and actual output by adjusting the weights: $\Delta w_{ij} = -\alpha \frac{\partial E}{\partial w_{ij}}$

Where:

- Δw_{ij} is the weight update
- α is the learning rate,
- E is the error of the network (often a mean squared error)
- $\frac{\partial E}{\partial w_{ij}}$ is the gradient term which is computed using the chain rule. This gradient term involves propagating the error backward through the network to find out how much each neuron contributed to the error, and then adjust the weights accordingly.[7]

The renaissance of neural networks was the 2010s, where deep learning models saw a rise due to the confluence of massive datasets and the computational power of Graphics Processing Units (GPUs). Additionally, Yann LeCun, Geoffrey Hinton, and Yoshua Bengio's pioneering work on deep learning architectures and their applications in various fields accelerated the growth of neural networks.[0]

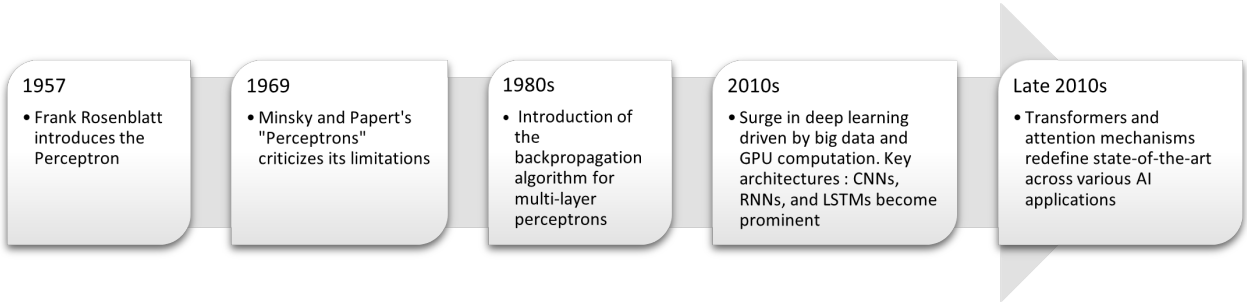


Figure 2.4: Evolution of Neural Networks

2.2.2 Mathematical aspect of Neural Networks

The operations conducted within these networks are fundamentally based on vectors, matrices, and higher-dimensional tensors, ensuring efficient processing of vast amounts of data.

Neural networks, in essence manipulate input data which are usually represented as vectors, by cascading them through hidden layers to generate an output. This is achieved using weight matrices, bias vectors, and activation functions. In a given network with L layers, the input vector x undergoes transformation at each layer l as: $a^{(l)} = f\left(W^{(l)a^{(l-1)}} + b^{(l)}\right)$

Where: $W^{(l)}$ represents the weight matrix and $b^{(l)}$ the bias vector for layer l, with f denoting an activation function.[0]

Activation Functions:

at the core of neural networks is the activation functions. These allow, an otherwise simple linear transformations network, to learn and approximate functions. The introduction of non-linearity removes the limitations of a purely linear model by allowing the network to capture intricate patterns and relationships in the data. Some of the prevalent activation functions include:

Sigmoid: Compressing values between 0 and 1: $\sigma(z) = \frac{1}{1+e^{-z}}$ [0]

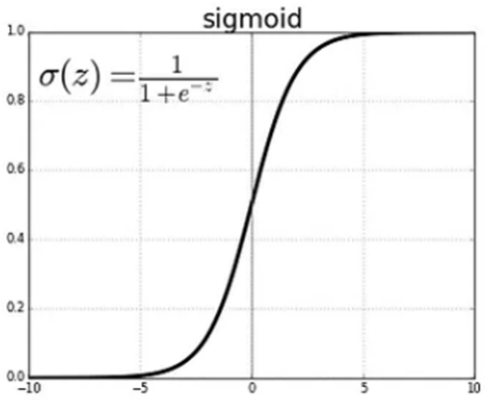


Figure 2.5: Graph illustrating the Sigmoid Activation Function [0]

ReLU (Rectified Linear Unit): Known for its computational simplicity: $\mathrm{Re}\,LU(z) = \max(0, z)$ [0]

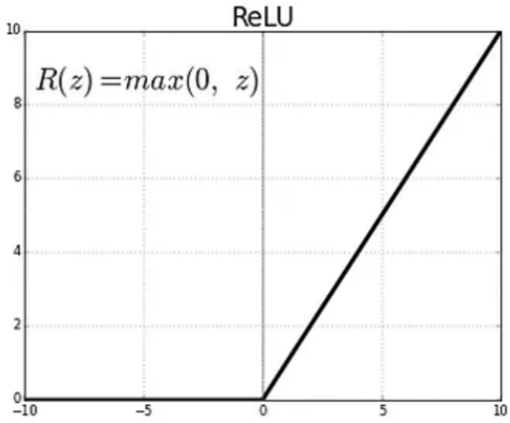


Figure 2.6: Graph illustrating the ReLU Activation Function [0]

Tanh (Hyperbolic Tangent): Ranging between -1 and 1, it offers zero-centeredness, a property that is absent in sigmoid: $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ [0]

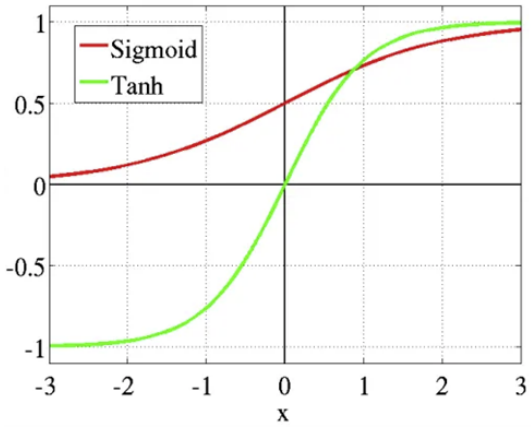


Figure 2.7: Graph illustrating the Tanh Activation Function [0]

Leaky ReLU: An enhanced version of ReLU that allows small negative values for z < 0 : Leaky ReLU$(z) = \max(\alpha z, z)$. Where α is a small positive coefficient. [0]

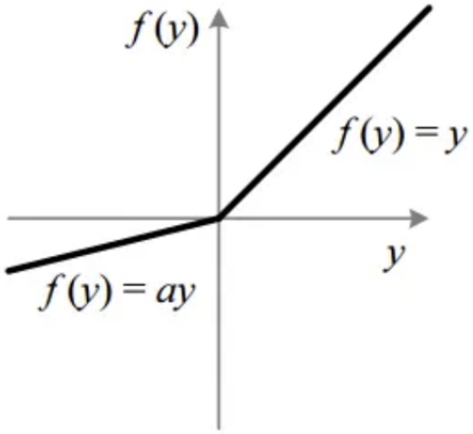


Figure 2.8: Graph illustrating the Leaky ReLU Activation Function [0]

PyTorch's Computational Perspective: Tensors & Backpropagation:

PyTorch fundamentally relies on tensors. These multi-dimensional arrays - matrices and vectors - facilitate the computations in neural networks. The beauty of PyTorch lies in its dynamic computation graph, allowing on-the-fly modifications which is immensely beneficial during model development and debugging.

In the context of training, the library employs a "define-by-run" approach, where the graph is constructed on-the-go during the forward pass. Once the forward computation is done, PyTorch can automatically compute gradients using the backward pass - due to its autograd feature. This facilitates backpropagation, which allows the network learn by adjusting its weights and biases in the direction that minimizes the error. [0]

Backpropagation: is the neural network's learning mechanism. It employs the chain rule to calculate gradients of the loss function with respect to network parameters, thereby guiding weight and bias adjustments. Given a loss L, the gradient with respect to the weight matrix of layer l is:

$$\frac{\partial L}{\partial W^{(l)}} = \frac{\partial L}{\partial a^{(l)}} \cdot \frac{\partial a^{(l)}}{\partial W^{(l)}}$$

Using the gradients, weights are updated by gradient descent: $W^{(l)} = W^{(l)} - \alpha \frac{\partial L}{\partial W^{(l)}}$

Where α symbolizes the learning rate, a hyperparameter determining the optimization's step size. It's crucial to highlight that during this iterative refinement, the network aims

to minimize the discrepancy between its predictions and the actual data, inching closer to the desired model.[0]

The essence of backpropagation is represented through the error term, $\delta^{(l)}$, for each neuron.

The output layer: $\delta^{(L)} = \nabla_a L \odot f'\left(z^{(L)}\right)$

The hidden layers: delta$^{(l)} = \left(W^{(l+1)}\right)^T \delta^{(l+1)} \odot f'\left(z^{(l)}\right)$

These allow the weight and bias adjustments to be derived from these errors:

$$\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} a^{(l-1)T}$$

$$\frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}$$

The above mathematical representations are in essence the code learning mechanism of the networks, where the process of adaptively refining these parameters ensures optimal performance. [0]

2.2.3 Optimizers

Optimizers analyse the discrepancies between the predicted data and the actual data, then determine the manner in which a model adjusts its parameters. This adjustment and the chosen optimization method, directly influences the performance, efficiency, and reliability of the neural network. Hence, a careful selection and application of these techniques is paramount to achieving optimal network performance.

Stochastic Gradient Descent (SGD):

iteratively updates the model's weights by considering a subset (or a single instance) of the dataset rather than the complete set, which is the practice in classical gradient descent. The weight update formula for SGD is: $w_{t+1} = w_t - \alpha \nabla J\left(w_t; x_t, y_t\right)$

Where:

w_{t+1} is the weight at the next iteration, w_t is the current weight, α is the learning rate, $\nabla J\left(w_t; x_t, y_t\right)$ is the gradient of the loss function J with respect to the weights for the given data point (x_t, y_t).

Stochasticity is both a blessing and a curse. By using a random subset of data, SGD can escape shallow local minima and is computationally more efficient. However, the erratic nature of its updates can also mean slower convergence and occasional divergence, particularly with higher learning rates.[9]

ADAM Optimizer (Adaptive Moment Estimation):

elegantly amalgamates the benefits of AdaGrad and RMSprop optimization algorithms. It employs both first and second moment estimations, and its updates are inherently adaptive in nature.

Given the first moment (mean) estimation m_t and the second raw moment (uncentered variance) estimation v_t the update rule for Adam is as follows:

Update rule for first and second moment estimations:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2$$

Bias-corrected moment estimations:

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}$$
$$\widehat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Parameter update rule:

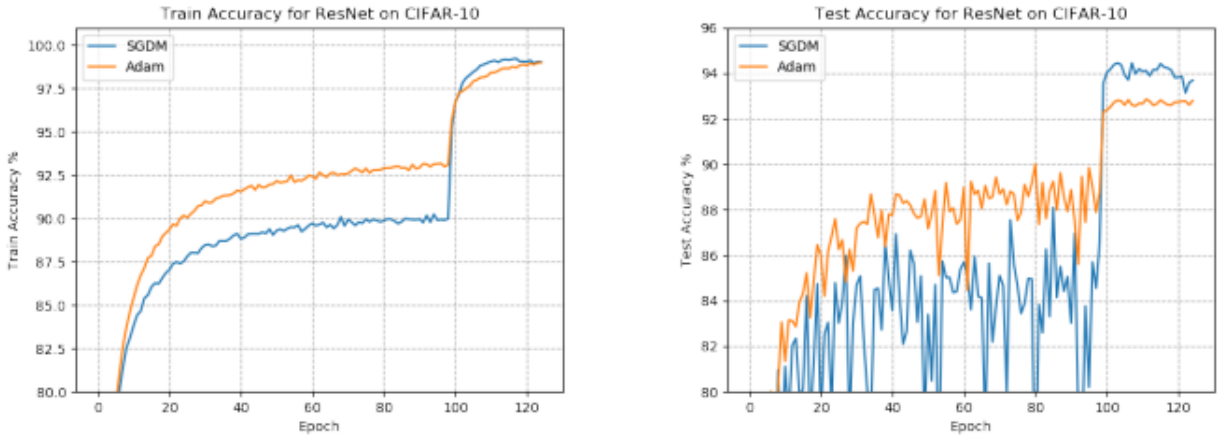
$$w_{t+1} = w_t - \alpha \frac{\widehat{m_t}}{\sqrt{\widehat{v_t}} + \epsilon}$$

Where:

- g_t is the gradient at time t,
- β_1 and β_2 are the exponential decay rates for moment estimations, typically set to 0.9 and 0.999 respectively,
- α is the step size (learning rate),typically set to 0.001
- ϵ is a small constant added to avoid division by zero, typically set to 10^{-8} [0]

Within optimization techniques, ADAM has emerged as a preferred choice due to its advanced efficiency in optimizing stochastic approximations. Specifically, its capability to offer an unbiased estimator for true gradients lends itself to expedited convergence, especially in more complex realms such as the deep learning frameworks used for camera calibration tasks. Within the context of image processing, ADAM tends to outshine SGD, predominantly during the nascent stages of model training. This heightened efficiency is largely attributed to ADAM's embedded adaptive learning rate mechanism. However, the

discourse on optimization methods would be incomplete without highlighting the merits of SGD. It evident that in certain occasions, as the learning rate of SGD is carefully tuned and calibrated, it can lead to SGD outperforming ADAM. Although ADAM dynamically adjusts its learning rates, the consistent learning rate of SGD can lead to more stable convergence in the long run. Furthermore, ADAM's adaptive mechanism might sometimes obscure certain intricacies of the data landscape which SGD, with its fixed learning rate, might capture. This facet has led many to advocate for SGD's prowess in achieving generalized and robust models, especially when deployed in scenarios that demand precision and stability over rapid convergence.[0]



(a) train **(b)** test

Figure 2.9: Comparison between SGD and ADAM opimization [0]

A Glimpse into Other Optimizers

- **RMSprop:** This optimizer modifies SGD by using a moving average of the squared gradient to normalize the gradient itself.
- **Adagrad:** scales the learning rate inversely proportional to the square root of the sum of all previous squared values of the gradient.
- **Nadam:** this is a blend of ADAM and NAG (Nesterov accelerated gradient). It combines the adaptive gradient with the Nesterov method of anticipating the next gradient.[0]

2.2.4 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) stand out as a subfield, tailored specifically for analyzing visual imagery. These networks are recognised for their proficiency in extracting spatial hierarchies of features.

At the core, a CNN is a deep, feed-forward artificial neural network. This architecture can be envisioned as being constructed from a multitude of layers where each layer transforms its input data to incrementally more abstract representations [0]. Unlike densely connected networks, CNNs adopt local connectivity patterns, with neurons in one layer connecting only to a small, local region of neurons in the preceding layer. This design imitates the receptive fields of the human visual system, which focuses on local spatial coherence.[0]

CNNs meticulously extract spatial hierarchies. In the early stages the elementary patterns, such as edges and corners, are detected. As the network progresses through subsequent layers, the complexity of these features evolves. From foundational elements like simple lines, the network advances to recognize shapes, which further enables the network to detect more intricate structures and objects. This hierarchical processing approach ensures that advanced features are inherently constructed from the basic, foundational elements.

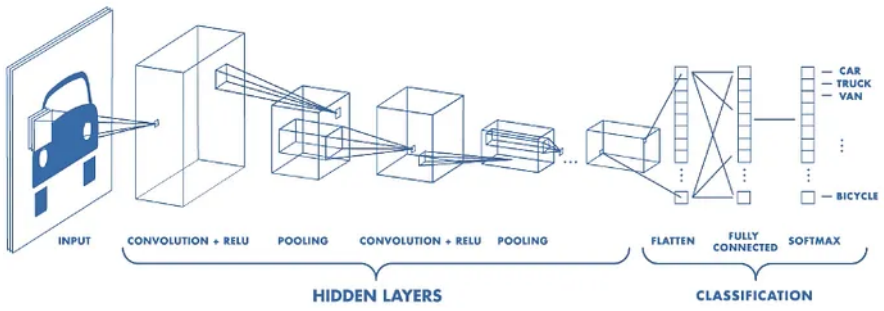


Figure 2.10: CNN Architecture [0]

Convolution Operation and Kernels:

Central to the CNN's operation is the convolution process. This operation starts with the kernel (or filter). This small matrix parses through the input image and conducts a dot product operation between itself and the portion of the input image. This results in a single pixel in the output feature map. As this process continues throughout the input

data, it generates a full feature map that emphasizes specific features in the image based on the filter values. The exact movement of this kernel is directed by two parameters called stride and padding.

Mathematically, the convolution operation between an input matrix I and a kernel K is given by the convolution operation in continuous form is defined as:

$$(I * K)(x, y) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} K(i, j) I(x - i, y - j)$$

In the context of convolutional neural networks, the sum is typically over a finite range determined by the size of the kernel (e.g., 3×3 or 5×5), rather than from $-\infty$ to ∞.[0]

So in the discrete 2D form with a finite-sized kernel, the equation becomes:

$$(I * K)\left(x_n, y_m\right) = \sum_{i=1}^{k} \sum_{j=1}^{k} K\left(x_i, y_j\right) I\left(x_{n-i}, y_{m-j}\right)$$

Where:

- I is the input image or feature map.
- K is the kernel or filter.
- x_n and y_m are the specific locations in the output feature map.
- The kernel has dimensions $k \times k$.

Padding: is the process of adding layers of pixels (often 0s) around the border of the input image to retain the input capacity's size after the convolution operation. Consequently, it helps to preserve spatial dimensions of the input.[17]

Stride: this is the parameter that dictates by how many pixels the kernel should step through the input image during the convolution process. A larger stride results in smaller spatial dimensions for the output volume, whereas a smaller stride retains more of the input volume's spatial dimensions but it more computationally expensive.[17]

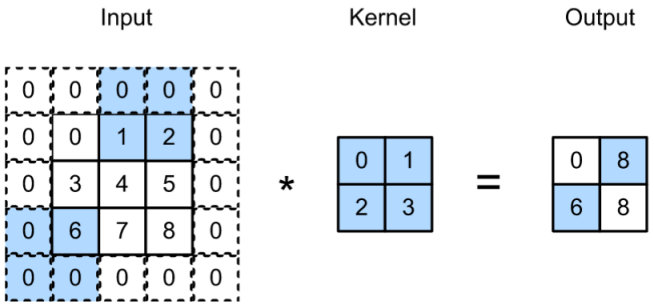


Figure 2.11: Illustration of Convolutional Operations: including padding, stride and kernels [?]

Pooling Layers: Pooling layers follow the convolutional layers to simplify the extracted features. These, reduce the spatial dimensions (width, height) of the input volume for each depth slice. Therefore, leading to fewer computational requirements and reduced chances of overfitting. The most common pooling operation is max pooling, which studies a spatial neighbourhood (like a 2×2 window) and selects the maximum value from that region.

Another variant is average pooling, which computes the average value for each patch on the input feature map.[17]

To illustrate, in the case of a 2×2 max pooling operation with a stride of 2:

$$\text{Pooled}(i, j) = \max_{0 \leq m,n \leq 1} I(2i + m, 2j + n)$$

Where:

- Pooled (i, j) is the value of the pixel in the pooled output at location i, j.
- $I(2i + m, 2j + n)$ represents the pixel value in the input image or feature map at location $2i + m, 2j + n$.

In essence, convolution operations aided by kernels, padding, and strides extract features, while pooling layers emphasize dominant ones, reducing dimensionality and creating a hierarchy of features. Together, they power the success of CNNs in various image processing tasks.

2.2.5 UNet Architecture

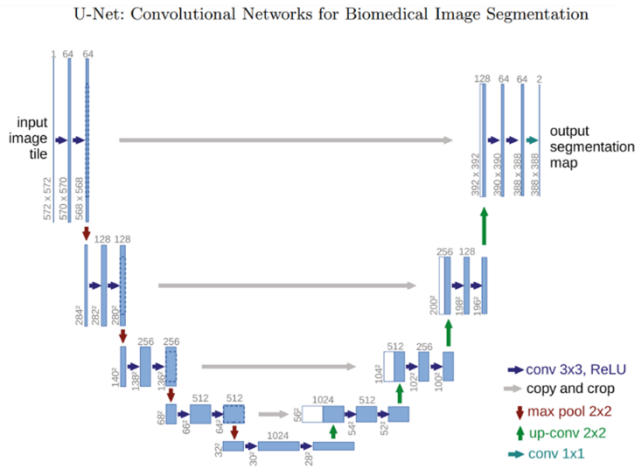


Figure 2.12: UNet Architecture [0]

UNet originates from the biomedical image domain, it was introduced by Ronneberger [36]. This architecture is tailored for semantic segmentation tasks where precise localization is essential. It merges the merits of both deep and shallow networks. The U-shaped design consists of a contracting path to capture context and an expansive path that ensures precise localization. The idea of symmetric expansion complements the contraction, which gives UNet its unique segmentation capabilities, especially in scenarios with fewer labelled data.[0]

Downsampling, Upsampling, and Skip Connections

Downsampling: The contracting path follows the typical architecture of a convolutional network. It consists of repeated application of two 3×3 convolutions, each followed by a rectified linear unit (ReLU) and a 2×2 max pooling operation with stride 2 for downsampling.

Upsampling: On the expansive path, every step consists of an up-convolution (or deconvolution) that halves the number of feature channels and doubles the spatial dimension. This is followed by a concatenation with the corresponding feature map from the contracting path which is cropped, and two 3×3 convolutions - each followed by a ReLU.

Skip Connections: These connect the contracting path to the expansive path. They provide a mechanism to use high-resolution features from the contracting path during the upsampling process, ensuring finer localization.

Activation Functions: The sigmoid activation function is often used in the final layer where binary segmentation is required. This ensures the output values range between 0 and 1 , rendering them as probability scores suitable for binary classification tasks. [0]

2.2.6 ResNet Architecture

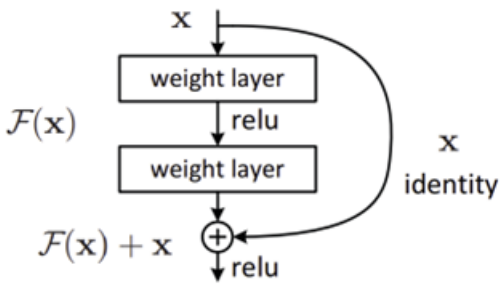


Figure 2.13: ResNet Architecture [0]

Deep neural networks with their ability in capturing intricate patterns in data, have become a staple in modern machine learning. Yet, as networks grow deeper, challenges arise that threaten their efficiency.[0]

The architecture's depth introduces problems of vanishing and exploding gradients. As gradients are propagated backward through the layers, they can shrink (vanish) or grow (explode), which adversely affects learning. Mathematically, given a deep feed-forward network with L layers, activations $a^{[l]}$ and weights $W^{[l]}$, the gradient descent update rule is [0]:

$$w_k \to w_k' = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \to b_l' = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

$$\Delta W^{[l]} = -\eta \frac{\partial \mathcal{L}}{\partial W^{[l]}}$$

If $\frac{\partial \mathcal{L}}{\partial W^{[l]}}$ is very small, the weights scarcely update, leading to vanishing gradients. However, if it's too large it leads to the weights updating drastically, therefore, causing the exploding gradient problem.

Residual Blocks - The Solution to Deep Network Training: ResNet combats these issues by introducing residual blocks. These allow the gradient to be directly back-propagated to earlier layers, which ensures smoother learning. A residual block can be represented as:

$$y = F\left(x, \{W_i\}\right) + x$$

Where $F\left(x, \{W_i\}\right)$ is the residual mapping to be learned. By reformulating the layers as learning residuals instead of direct mappings, ResNet alleviates the vanishing gradient issue.[0]

Identity Shortcut Connections (Bridging the Gap): These connections in ResNet allow outputs from previous layers to bypass one or more layers and directly add to the deeper layers' output. Therefore, creating a is a direct path for the gradient during back-propagation ("shortcut"). These connections ensure that the gradient remains substantial throughout the backpropagation process in conjunction with ReLU activation functions. Yet, it's worth noting that the popular ReLU can sometimes lead to the "dying ReLU" problem, where neurons output a constant zero and are incapable of updating, mitigated to an extent by ResNet's skip connections. For instance, in a sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$ the gradient is extremely small for large negative or positive values of z, leading to vanishing gradients. But with skip connections, ResNet ensures the gradient flow remains substantial.[0]

2.2.7 Enhancing Generalization: Dropout Layers

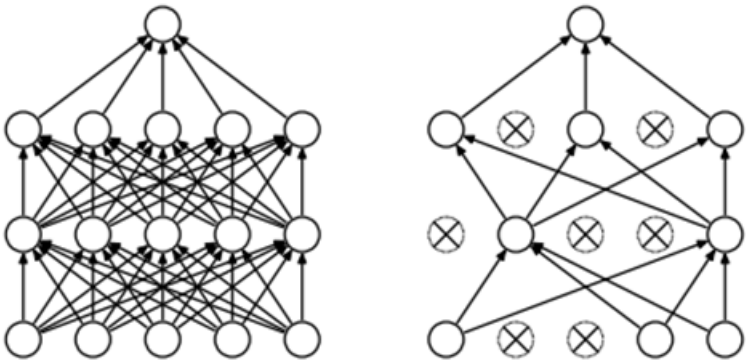


Figure 2.14: Comparison of a Neural Network witout and with Droput Layers [0]

One of the key points of training robust neural networks lies in their ability to generalize effectively to unseen data. Dropout layers emerged as a powerful regularization technique, that address this by introducing an element of randomness into the training process. By turning off, or "dropping", neurons with a defined probability, p, dropout ensures that the network remains resilient against the menace of overfitting. The rationale behind dropout is intriguing: the random omissions make it untenable for any two subsequent neurons to become exclusively dependent on each other [0]. This strategic neuron deactivation creates an environment where the network, during the testing phase, behaves akin to the amalgamation of several neural networks that share parameters.[0]

How Dropout Affects Forward and Backward Pass: Consider $h(x)$ a linear transformation of an input x with dimensionality d_i into a d_h-dimensional space. This transformation can be captured as:

$$h(x) = xW + b$$

Delving deeper, for an activation function $a(h)$ and its dropout-induced counterpart $a(h)$, where $M \sim B\left(p_h\right)$ and rect(h) is a rectifier, the following relations hold:

$$a(h) = \text{rect}(h)$$

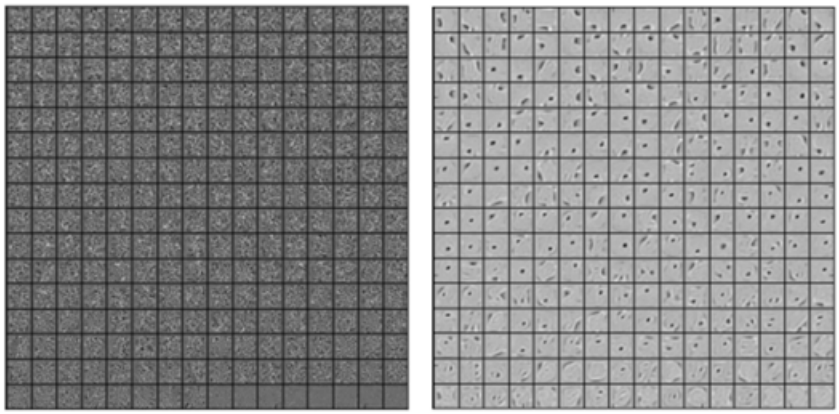
Which delineates the pure, undisturbed activation function during the testing phase.

$$\bar{a}(h) = M \odot \text{rect}(h)$$

Which encapsulates the activation with dropout during the training regimen.

To approximate a more balanced unit activation, Srivastava [0] advocate for the scaling of activations $a(h)$ by p during the testing phase.[0]

Comparison of Networks with and without Dropout: Dropout's impact is most pronounced when we juxtapose neural networks with and without its influence. Networks incorporating dropout exhibit a remarkable resistance to overfitting, a notorious issue where a model becomes overly tailored to the training data and consequently stumbles when presented with new, unseen data. Furthermore, by using dropout, we're not merely training a singular model. Instead, given the stochastic deactivation of neurons, we're orchestrating the training of a plethora of models simultaneously, albeit in a shared-parameter space.[0]



(a) Without dropout (b) Dropout with $p = 0.5$.

Features learned on MNIST with one hidden layer autoencoders having 256 rectified linear units.

Figure 2.15: The effects of the Dropout Layer on Feature Extraction[0]

2.2.8 Recurrent Neural Networks (RNNs) and LSTM

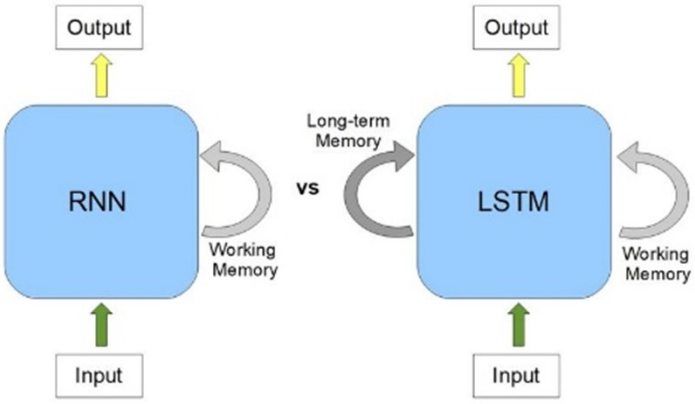


Figure 2.16: RNN and LSTM Architecture [0]

RNNs have notably garnered attention due to their unique capability to handle sequences, a feature paramount in tasks like natural language processing and time series analysis. Introduced as early as the 1980s, their design captures temporal dynamics, making them apt for sequential data. [18]

Structure, Function, and Challenges: The foundation of an RNN is its recurrent nature. Unlike feed-forward neural networks, RNNs maintain a loop within the neuron, allowing information persistence. Mathematically, the hidden state h_t at time t is calculated as:

$$h_t = \sigma\left(W_x x_t + W_h h_{t-1} + b\right)$$

(where σ is the activation function, W_x and W_h are weight matrices, x_t is the input at time t, and b is the bias vector.)

This equation, while capturing the essence of an RNN, also exposes its vulnerability: the vanishing and exploding gradient problem. As sequences grow, the gradients, during backpropagation, tend to diminish to zero or escalate uncontrollably.[14]

Recurrent Neural Networks operate on sequences, carrying forward information through time. A major challenge for vanilla RNNs is their struggle to preserve long-term dependencies. Recognizing the importance of certain features from early stages of input sequences and keeping them throughout the network's operation is a formidable task for these networks.

LSTMs: The Improved RNN for Longer Sequences

The solution to this problem was the development of the Long Short-Term Memory (LSTM) unit. Hochreiter and Schmidhuber were the first to propose this in 1997, with various modifications introduced thereafter. This research follows Graves for the LSTM's implementation.

At its core, the LSTM unit differs from a traditional RNN. Instead of just computing a weighted sum of input signals and applying a nonlinear function, each LSTM unit preserves a memory, represented as $\left(c_t^j\right)$, at time t. The activation or output of the LSTM unit is governed by: $h_t^j = o_t^j \tanh\left(c_t^j\right)$

with o_t^j functioning as an output gate determining memory exposure.

Through introduction of the various gates (input, output, forget), LSTMs have the ability to choose what to remember and what to discard. Furthermore, LSTM units are purposefully designed with the ability to judge whether to retain existing memory using these gates. If an LSTM unit identifies a crucial feature early in an input sequence, it can easily maintain this information over extended periods, thereby capturing potential long-range dependencies.[13]

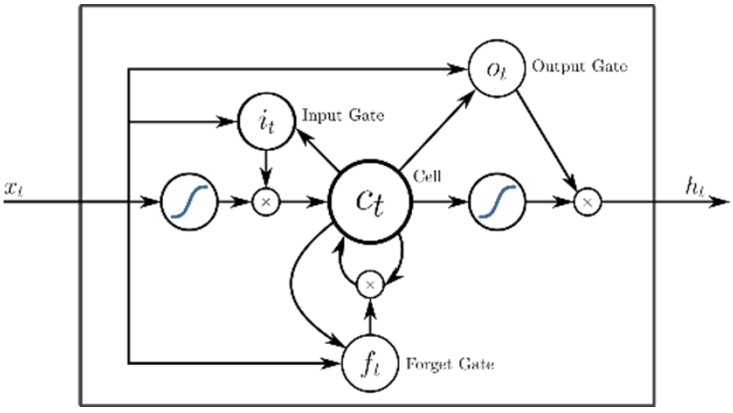


Figure 2.17: LSTM's internal make-up [0]

Gate Mechanics in LSTMs: Forget, Input, and Output Gates Explained These gates in the LSTM play pivotal roles [13]:

- **(Output Gate o_t^j :** Controls how much memory gets revealed and is given by:

$$o_t^j = \sigma\left(W_o \boldsymbol{x_t} + U_o \boldsymbol{h}_{t-1} + V_o \boldsymbol{c}_t\right)^j$$

where σ is the logistic sigmoid function.

- **(Memory Cell Update:** The memory cell c_t^j is updated using the formula: $c_t^j = f_t^j c_{t-1}^j + i_t^j \widetilde{c_t^j}$

  This equation represents the blending of past memory with new content.

- **Forget Gate $\left(\boldsymbol{f_t^j}\right)$ and Input Gate $\left(\boldsymbol{i_t^j}\right)$:** Determine the degree of retention of past memory and the addition of new content, respectively. These are represented as:

$$f_t^j = \sigma\left(W_f \boldsymbol{x_t} + U_f \boldsymbol{h_{t-1}} + V_f \boldsymbol{c_{t-1}}\right)^j$$
$$i_t^j = \sigma\left(W_i x_t + U_i h_{t-1} + V_i c_{t-1}\right)^j$$

2.2.9 Self-Attention Mechanism and Transformers

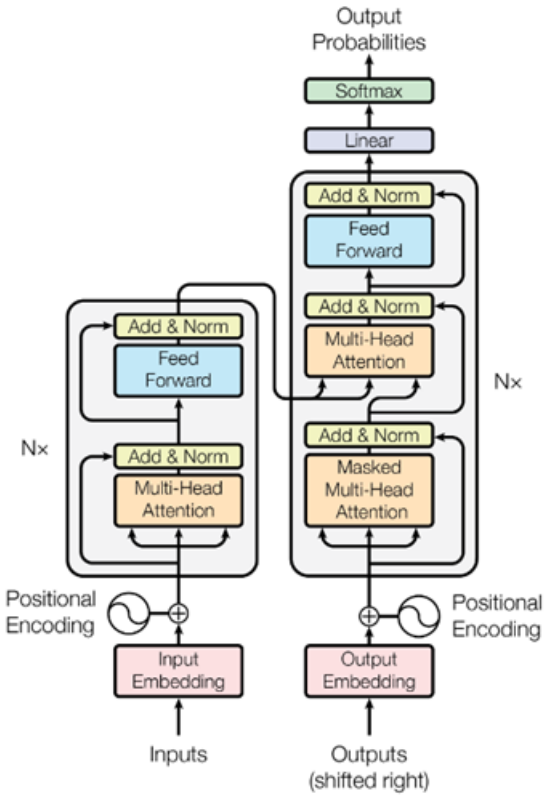


Figure 2.18: Transformer Architecture [0]

Sequence processing has conventionally been the forte of recurrent architectures like RNNs and LSTMs. However, as we delve into intricate tasks like language translation and sentence embedding, the limitations of these architectures surface in the form of scalability and long range dependency challenges. It was imperative for a novel approach to sequence understanding to evolve. Enter the world of Self-Attention Mechanisms and Transformers.

Why Attention? The Limitations of Traditional Sequence Processing The sequential nature of data, such as text, often necessitates architectures that can consider long-range dependencies. Traditional methods (such as RNNs) face challenges when processing long sequences due to their inherent sequential processing; because it limits parallelism and poses difficulties in capturing long-range dependencies. However, this is addressed by weighing different parts of the input sequence distinctly; which allows the model to "attend" to specific parts of the input when producing an output.

The Transformer Architecture: Multi-Head Self-Attention and Positional Encoding Introduced by Vaswani [0] in the landmark paper "Attention is All You Need", the Transformer architecture revolutionized sequence processing. At the heart of the Transformer is the self-attention mechanism, which allows it to weigh input tokens based on their relevance to a given query. This is mathematically captured by the scaled dot-product attention:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

The architecture goes further by employing multiple attention heads known as multi-head attention. Each head captures different types of relationships in the data. To aggregate the diverse information from these heads, the outputs (often termed as "heads") are concatenated and linearly transformed:

$$\text{MultiHead}(Q, K, V) = \text{ Concat } (\text{head}_1, \ldots, \text{head}_h)\, W^O$$

Where: $\text{head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$

The Transformer also introduced positional encoding, ensuring the model could infer the position of tokens in a sequence, as the self-attention mechanism is inherently permutation invariant. The sinusoidal form of positional encoding was chosen due to its ability to allow the model to extrapolate to sequence lengths beyond those seen in training:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{\text{model}}}}}\right)$$

[0]

Applications: BERT, GPT, and Modern NLP Breakthroughs The versatility and efficiency of the Transformer architecture have led to its adoption in various state-of-the-art models in Natural Language Processing (NLP). The most prominent examples at the moment are BERT (Bidirectional Encoder Representations from Transformers) developed by Google AI [16] - which focuses on bidirectional training by predicting masked portions of a sentence - and GPT (Generative Pre-trained Transformer) developed by OpenAl - which is able to generate contextually relevant text.

2.3 Related Work

A literature review is crucial for camera calibration techniques because it establishes the groundwork for future research by providing insights into current knowledge, spurring creativity, directing experimentation, and assuring the thorough assessment of novel approaches. It is an essential element in the research process that enables scholars to make significant contributions to the area.

2.3.1 The work of: Workman et al.

Workman et al. [0] discussed on their paper that It was not feasible to calibrate "in the wild" images, such as those gathered from photo-sharing websites and publicly accessible outside webcams, using such conventional techniques. New issues in camera calibration are motivated by this difficulty, and new approaches are necessary. In this research, they explore the use of a deep convolutional neural network (CNN) to determine a camera's focal length from a single picture. Their method was incredibly quick and acts directly on the raw picture densities with no per-image parameter setup.

It is difficult to extract geometrical specifics of the scene resulting from a single shot. Numerous techniques have been developed for metrology, 3D layout analysis, and camera

calibration. Frequently, these techniques make use of reference objects such as coplanar or concentric circles or planar calibrations. Vanishing points are used in other methods to explain the geometric scene structure. The development of natural signals like solar refractive phenomena and sun position has also been used to calibrate images captured "in the wild." To calculate camera focal length from raw pixel intensities, a deep convolutional neural network has been trained on millions of real-world images. This is in accordance with a developing field of research that investigates deep learning methods to estimate geometric image attributes, which include generating a metric depth map from a single image [0].

Existing datasets have either been restricted to a single camera or need for laborious post-hoc calibration, which is frequently impractical. To get around this, a dataset was created by fusing camera models calculated using 1D structure from motion (1DSfM) with pictures. AlexNet, one of the most popular CNN architectures, has been employed. This architecture, which was first created for multi-class object classification, has been successfully modified for applications including scene characterization and image style identification. A streamlined form of a pinhole camera. An image location, p = [u, v,]T, was projected from a world point, P = [X, Y, Z]T, given the camera intrinsics, K, extrinsic rotation, R, and translation, t. This might be simplified to a pinhole camera model using the formula for the focal length, zero skew, the principal point in the centre of the picture, square pixels, and an aligned camera frame to the world frame. The objective was to calculate the horizontal field of view, which has a one-to-one mapping with focal length for a certain picture width [0]. High-level vision techniques require focus length for proper functioning. Single-view calibration techniques are ineffective without precise geometric calibration objects. A deep convolutional neural network has developed a quick approach to determine focal length from raw pixels, outperforming benchmarks like an automated method based on orthogonal vanishing points [0].

Findings: In order to determine focal length of a camera in an individual image, a pivotal problem in the discipline as and forensic analysis, AlexNet, a deep convolutional neural network, (CNN) has been employed. The technique was rapid and forthright that acts on raw image densities excluding per image parameter setting. AlexNet, a popular CNN architecture, has been trained on millions of real-world photos to estimate camera focal length using raw pixel intensities. This approach aligns with an emerging area of study exploring deep learning approaches to estimate geometric picture attributes, in-

cluding calculating a metric depth map from a single image. AlexNet, a popular CNN architecture, has been modified for scene characterization and image style identification. Existing datasets are often limited to single cameras or require post-hoc calibration, making it impractical. A dataset was created by combining 1D structure from motion models with pictures. A pinhole camera model was used to calculate the horizontal field of view, which was determined by the focal length, zero skew, principal point in the center, square pixels, and an aligned camera frame to the world frame. A deep convolutional neural network has developed a fast method for determining focal length (f) from raw pixels, surpassing automated methods like orthogonal vanishing points [0].

2.3.2 The work of: Hold-Geoffroy et al.

The authors, Hold-Geoffroy et al. [0] suggested a deep convolutional neural network that outperforms existing approaches in terms of standard L2 error when inferring camera calibration parameters from a single picture. To learn how people react to camera estimating mistakes, they carry out a sizable human perception research. They create a novel perceptual calibration measure and show its efficacy. The network was trained using autonomously produced samples from a sizable panoramic dataset and was utilised for a variety of tasks, including virtual object insertion, picture retrieval, and compositing. Moreover, proposed to develop a deep neural network to train on a single picture to directly predict camera parameters, especially the focal length, pitch, and roll. Recent research on CNN-based focal length and horizon estimates was comparable to this. By jointly estimating all the parameters and training on automatically generated sample images from a sizable panoramic dataset, researchers greatly improved their findings.

The geometric camera model that was employed in this study was presented. According to the pinhole camera concept, $\mathbf{P}_{\text{im}} = [\lambda u \lambda v]^T$ represents the pixel coordinates of a 3D point $\mathbf{P}_{\text{w}}$. $\text{T} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] \, [\mathbf{P}_{\text{w}} \mid 1]^T$

In this study, the terms "camera calibration" relate to this streamlined geometric camera model's vertical field of view, pitch, and roll.

$$\mathbf{p}_{\text{im}} = [\lambda u \lambda v \lambda]^T = \mathbf{K}[\mathbf{R} \mid \mathbf{t}] \, [\mathbf{p}_{\text{w}} \mid 1]^T$$

K is the camera projection matrix (camera intrinsics), and R and t are the camera rotation and translation in the global reference frame (camera extrinsics) in homogeneous coordinates. as further condensing the model to square pixels, no skew, and the picture

centre at the main point, the projection matrix K is represented as K[2].

The objective was to train a deep network to infer from a single image the camera roll, pitch, and field of view. The SUN360 database, which has a wide collection of 360-degree panoramas, served as inspiration for the project and as a resource for obtaining photographs and ground-truth camera specifications. Each was subjected to the extraction of seven rectified pictures using a standard pinhole camera model with random settings. A DenseNet model was adopted and pre-trained on ImageNet, replacing the final layer with three distinct heads: one for estimating the horizon angle, the second one for evaluating the horizon's distance from the centre of the image, and a third one to determine the vertical field of the image [0].

Findings: The Deep Convolutional Neural Network (DCNN) outperforms conventional methods for extracting camera calibration parameters from an individual image. The DCNN was trained on samples from a large-scale panoramic dataset, and was used for image insertion, image retrieval, and image compositing. The network was trained on a geometric camera model, which includes the vertical field of view, pitch, and roll. The SUN360 database served as inspiration for the project, and a DenseNet model was pre-trained on ImageNet. The network was used to predict camera parameters, including roll, pitch, and field of view, and was used for various tasks [0].

2.3.3 The work of: Wakai et al.

In this paper, Wakai et al. [0], discussed about the intrinsic camera model is a crucial tool in predicting extrinsic and intrinsic camera parameters from a single image. However, these methods can degrade in fisheye images due to mismatches between actual and expected projections. A generic camera model is proposed to address these distortions and improve learning-based calibration methods. In this study, the suggested camera model, which is based on a closed-form solution for several fisheye cameras, is first discussed. The learningbased calibration approach for recovering rotation and fisheye distortion was then disclosed. A novel loss function's nomenclature and mechanism are finally described. Two categories of calibration techniques exist: learning-based and geometry-based approaches. Geometric-based techniques are highly accurate but require a calibration object, while learning-based techniques can calibrate cameras without a calibration item, known as deep single-image camera calibration. Extrinsic (rotation and translation) and intrinsic (image sensor and distortion parameters) characteristics are estimated through camera calibra-

tion. Strong constraints based on the calibration object or line detection have been used to construct geometric-based calibration algorithms. For the steady optimisation of calibration, this constraint unambiguously depicts the relationship between world coordinates and picture coordinates. It has been proposed to use generic camera models to calibrate various fisheye cameras. To extract the features and details of images, DenseNet-161 was trained on ImageNet [0].

Findings: Camera calibration techniques are crucial for predicting extrinsic and intrinsic camera parameters from a single image. However, these methods can degrade in fisheye images due to mismatches between actual and expected projections. A generic camera model is proposed to address these distortions and improve learning-based calibration methods. Two categories of calibration techniques exist: learning-based and geometry-based approaches. Geometric-based techniques are highly accurate but require a calibration object, while learning-based techniques can calibrate cameras without a calibration item. Geometric-based calibration algorithms use strong constraints to optimize calibration. The use of generic camera models and DenseNet-161 on ImageNet has been proposed for calibrating fisheye cameras [0].

2.3.4 The work of: Zhang et al.

Zhang et al. [0], the researchers present in their paper a deep learning-based technique for automatically estimating focal length, distortion characteristics, and rotation angles between images from PTZ cameras, thereby reducing the complexity of self-calibration, a process that involves online computation for camera calibration. This research offered a deep learningbased technique to automatically estimate the focal length and distortion characteristics of both pictures as well as the rotation angles between them given an image pair acquired by a PTZ camera. There is no publicly accessible large-scale dataset for calibrating PTZ cameras for training and testing DNNs. In order to get around this problem, panoramas have been used to generate massive datasets that may be used to train and test the algorithms that have been suggested. The dataset for this study was created using the SUN360 database, which is freely accessible [0].

Findings: Researchers have developed a deep learning-based technique for automatically estimating focal length, distortion characteristics, and rotation angles between images from PTZ cameras. This technique reduces the complexity of self-calibration, a process

that involves online computation for camera calibration. The researchers used panoramas to generate massive datasets for training and testing DNNs, using the SUN360 database, which is freely accessible. This method can be used to estimate the rotation angles between images and improve PTZ camera calibration [0].

Chapter 3

Method

3.1 Data generation

The challenge of obtaining authentic chessboard images equipped with their intrinsic and extrinsic parameters posed a significant constraint. To address this, a strategic approach was adopted: the generation of synthetic images of chessboards. Utilising 3D rendering techniques, models of chessboards were meticulously designed. These models were then subjected to simulations, factoring in given intrinsic parameters that define the internal properties of the simulated camera. Furthermore, to ensure realism and applicability, the simulations were conducted with the models viewed through specified extrinsic parameters, representing the position and orientation of the camera in relation to the object. This method not only circumvented the limitations of unavailable real-world images but also provided a controlled environment to understand the effects and implications of varying these parameters.

3.1.1 Intrinsic choice

The quality of the synthetic chessboard images is closely tied to the choice of intrinsic parameters. To achieve this, a systematic approach was employed.

Initially, a total of 2000 combinations of the intrinsic parameters were formed. These parameters included: fx, fy, cx, and cy. From personal calibrations with my device, it was found that the focal lengths fx and fy usually lie between 500 and 800 . To provide a broader scope, I also considered values up to 1600 , drawing inspiration from devices such as GoPros. The fx values were chosen from a range of $[200, 1600]$. Following that,

fy values were selected from the range $[fx- \ 300, fx + 300]$, making sure they stayed within the defined boundaries. It's worth noting that, while fx and fy were not strictly identical, they were not chosen as completely independent values, reflecting the behavior of real-world cameras.

For the images, which have a resolution of 320 by 320 in RGB format, the optical center (represented by cx and cy) is typically around the middle of the image. To capture this, the values for cx and cy were chosen from:

$$cx \sim U\left(\frac{320}{2} - 0.15 \times \frac{320}{2}, \frac{320}{2} + 0.15 \times \frac{320}{2}\right)$$
$$cy \sim U\left(\frac{320}{2} - 0.15 \times \frac{320}{2}, \frac{320}{2} + 0.15 \times \frac{320}{2}\right)$$

This approach allowed the optical centre to shift by up to 15% in any direction. In the end, three distinct sets of the 2000 combinations were created, with each set earmarked for training, validation, or testing.

3.1.2 Pytorch3D

PyTorch3D (library), is a reusable component package for Deep Learning using 3D data. PyTorch3D's distinguishing features:

- It creates a data structure that may be used to manipulate and store 3D objects.
- It includes efficient operations on triangular meshes (a type of polygon mesh commonly used in computer graphics) such as graph convolution, project transformations, and so on with various mesh renderers.

The following list of industrial deep learning applications that benefit from PyTorch3D includes:

- It offers a collection of loss functions and 3D operators that are quick and differentiable, allowing researchers to incorporate them into current state-of-the-art deep learning systems straight away.
- By providing engineers and researchers with a toolkit and enabling the effective implementation of research with complicated 3D inputs, it may be utilised to advance the intersection of 3D and deep learning. This is accomplished through the use of

heterogeneous batching capabilities, efficient and optimised operators, and a modular rendering API [0].

3.1.3 Chessboard design

The chessboard was designed with accuracy and practicality in mind. Therefore, the following parameters were carefully chosen:

- The number of rows and columns - that form the chessboard- were predetermined to always fall between 5 and 10 . Specifically the columns were consistently programmed to be the longer side of the board.
- To ensure flexibility and varied chessboard designs, the individual squares that form the chessboard had to be constrained too. Within the 3D rendering space the square sizes were allowed to vary only between 1 to 5 units.

Alternating pure black and white colours were used for the squares to ensure the recognisable pattern of a traditional chessboard was maintained. The placement of the chessboard mesh was strategically chosen to be on the XY plane, lying flat, ensuring a depth (in the z-direction) of 0 . Recognising the potential need for corner masks in future analysis, a proactive approach was taken. The coordinates of the inlying corners of the chessboard were saved. This decision will facilitate an easier and more efficient method of generating masks for the chessboard corners when required.

3.1.4 Rendering shaders

A pivotal aspect of the data generation process involved the exact rendering of the chessboard images. To accomplish this, specific functionalities within the pytorch3D framework were employed. Most importantly, these 2 functions: *HardPhongShader* and *MeshRasterizer*. The *HardPhongShader* function was utilised for its precise ability to simulate the interaction of the light with different surfaces. Similarly, the *MeshRasterizer* function was used for its built-in feature of the framework. This function facilitated the transformation of the 3D chessboard models into 2D images.

Through the combined application of these functions, the rendering process was optimized to produce high-quality synthetic images of the chessboard.

Phong Shading

Phong shading is a more precise interpolation-based method for rendering polygons. It was originated by Phong Bui Tuong. It optimises the Gouraud shade by providing a more accurate representation of the shade of any smooth surface. Instead of the intensity values, it interpolates the normal vector [24]. Gouraud shading is a computer graphics technique that simulates the effects of light and color on an object's surface using intensity-interpolation. It computes the intensity of each pixel, linearly interpolating intensity values over the surface to shade a polygon [23].

Phong shading interpolates vertex normals, followed by per-pixel lighting based on the resultant piecewise linear normal field, rather than computing lighting for each vertex and linearly interpolating the resulting colour values inside triangles. As a result, per-pixel Phong shading for surface splats is the primary approach for high-quality point-based rendering [0].

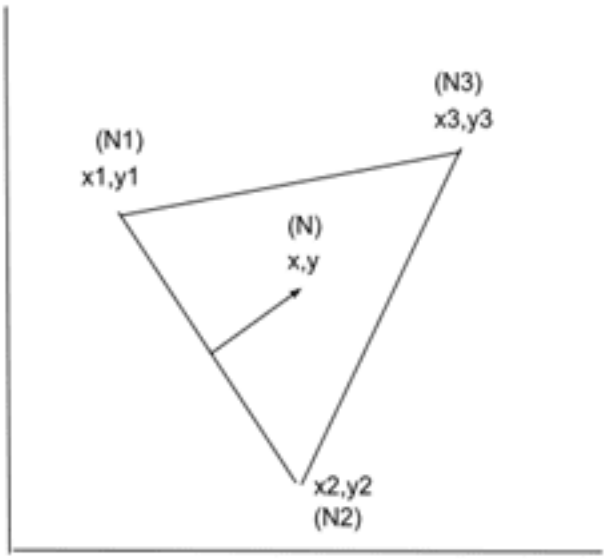


Figure 3.1: Surface Norm Interpolation [24]

The projected pixel intensities of the surface points must be determined by applying the lighting model throughout each scan [24].

Advantages: Due to the linear intensity interpolation, some dark and brilliant intensity streaks known as mach bands occur on the surface; this approach decreases the mach band

effects and shows more realistic highlights. It is more accurate than Gourand Shading.

Disadvantages: It is slower than Gourand Shading. It necessitates additional calculations; therefore, the cost of shading rises with each succeeding step [24].

Rasterizer

Rasterize, often known as rasterise, is the process of turning a vector image (a mathematically defined representation of points and curves) to a raster image (a picture made up of discrete pixels). In simple terms, scene geometry is converted to pixels for display on a screen or printing. Rasterization is a technique often used in real-time 3D graphics processing to swiftly transform pictures for display on a computer monitor [1].

PyTorch 3D Mesh Rasterizer has been accustomed for this project. The form of the intended output picture is used to rasterize a group of meshes.

Each mesh is rasterized onto a distinct image of shape $(image_{size}, image_{size})$ where $image_{size}$ is an integer or a tuple. There are two aspect ratios to be careful of:

When an image's desired size is non-square (i.e., a tuple of (H, W) where $\mathrm{H} ! = \mathrm{W}$), the aspect ratio must be carefully taken into account [0].

Rasterization is the conversion of a vector-based picture or object into a bitmap representation, which is a mathematical representation of forms and lines into a pixel grid. This process converts the vector graphics format into a collection of pixels for display on a screen, activating display pixels with specific colours to create the desired image.[2]

3.1.5 World axis

A notable discrepancy was observed between the default world coordinate frames of PyTorch3D and OpenGL. Specifically, in PyTorch3D, the $+Z$ direction is oriented into the screen, while in OpenGL, it points outwards. However, it is crucial to note that both coordinate frames maintain a right-handed orientation. Furthermore, the Normalized Device Coordinate (NDC) system in PyTorch3D adheres to a right-handed convention, contrasting with the left-handed NDC system in OpenGL. The projection matrix in OpenGL is responsible for this handedness switch.

Within the realm of 3D data analysis, there are four essential coordinate systems that users should acquaint themselves with:

- **World Coordinate System:** This particular system serves as the foundation for the space in which the 3D object or scene resides, often simply termed as the 'world'.

[0]

- **Camera View Coordinate System:** This system originates from the image plane with its Z-axis oriented perpendicularly to it. Within PyTorch3D's framework, it's established that +X directs towards the left, +Y rises upwards, and +Z extends away from the image plane. The transformation from the world system to the view system involves applying a rotation R and subsequently a translation T.[0]

- **NDC Coordinate System:** Sometimes described as the view volume, this system is in a normalized format and defines the part of the object or scene that is rendered. Under the PyTorch3D framework for square images, the coordinates $(+1, +1$,znear) designate the top left near corner, whereas $(-1, -1,$ zfar) mark the bottom right distant corner of the volume. For non-square images the shorter side of the section in the plane fluctuates between $[-1, 1]$, while the extended side varies from $[-s, s]$, where s is the aspect ratio ensuring $s > 1$. This ratio is derived by diving the longer side by the shorter side. The transformation from the view system to NDC incorporates the camera projection matrix P.[0]

- **Screen Coordinate System:** This system offers another perspective of the view volume, with its XY coordinates expressed in pixel dimensions rather than a normalized format. Here, the coordinate $(0, 0)$ is attributed to the top left extremity of the first pixel, while (W, H) corresponds to the lower right extremity of the last pixel. [0]

3.1.6 Look at transform

For the generation of extrinsic parameters corresponding to the views of the chessboard, the 'look at view transform' in PyTorch3D served as the primary tool. The decision to constrict both the elevation and azimuth between -55 and 55 degrees emerged from the need to offer a variety of viewing angles, while avoiding angles that might be perceived as too radical.

Addressing the question of distance is pivotal. Given that a chessboard observed from a consistent distance dd, but with parameters fx, fy and $2fx, 2fy$ appears twice as large, the distance parameter demands careful consideration. The goal was to prevent the chessboard from either dominating the view or appearing diminutive. As a consequence, the distance was chosen to be proportional not only to the chessboard length (determined by the longest side in rendering space using 3D units) but also in relation to the fx, fy parameters. The

formula for this is as follows:

$$\mathrm{d} \sim U\left(1.2 \times l + \frac{l_c \times (b_{scaling} - 1)}{1.5}, 2.3 \times l_c + 1.5 \times l_c \times (b_{scaling} - 1)\right)$$

Where:

- d is the distance
- l_c is the chessboard length
- $b_{scaling}$ is the move back scaling, this is given by fx divided by 200 , this is the minimum allowable fx value.

The coefficients, shown in the equation were derived through rigorous testing, the experiment went through multiple round of rendering and adjustments to ensure optimal visibility.

Considering the 'look at' aspect of the transform, it requires a specific 3D coordinate to which the camera directs its attention. Drawing inspiration from typical human behaviour during photography, the center of the chessboard was deemed as the focal point. Consequently, a 2D Gaussian distribution was employed with its mean centered at the X, Y coordinate of the chessboard's center. Given that the chessboard resides on a flat plane, the z coordinate becomes redundant.

The covariance matrix was then defined as:

$$\begin{bmatrix} 0.25 \times \text{row\_length}/2 & 0 \\ 0 & 0.25 \times \text{column\_length}/2 \end{bmatrix}$$

This matrix showcases 2 independent Gaussian components. With a standard deviation of $0.25x$ (distance between centre and outer edge). To introduce further variability, a roll angle for the camera was integrated. This angle represents the camera's rotation in its place around the normal vector connecting the pinhole to the object it observes, was randomly sampled between -20 and 20 degrees.

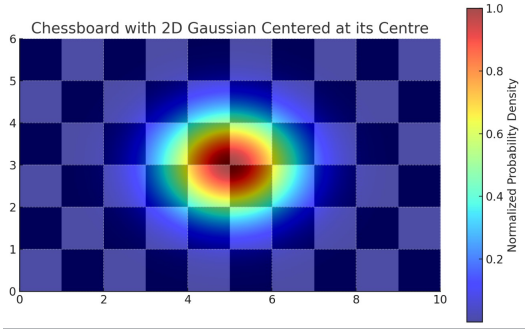


Figure 3.2: visualisation for camera direction of gaze on chessboard

Azimuth, Distance and Elevation

In the azimuth elevation coordinate system, azimuth and elevation are measured in degrees, between the r axis (which is the z axis). Their projections are on the xz and yz planes. This coordinate system is similar to coordinates but has some differences in its definition. The range or separation from the origin is represented by r [19].

Here are the formulas used to convert azimuth elevation coordinates into coordinates:

$$z = \sqrt{\frac{r^2 \cos^2(\text{azimuth})}{1 + \cos^2(\text{azimuth}) \tan^2(\text{elevation})}}$$
$$x = z \tan(\text{azimuth})$$
$$y = z \tan(\text{elevation})$$

The reverse transformations are:

$$\text{azimuth} = \arctan\left(\frac{x}{y}\right)$$
$$\text{elevation} = \arctan\left(\frac{y}{z}\right)$$
$$r = \sqrt{x^2 + y^2 + z^2}$$

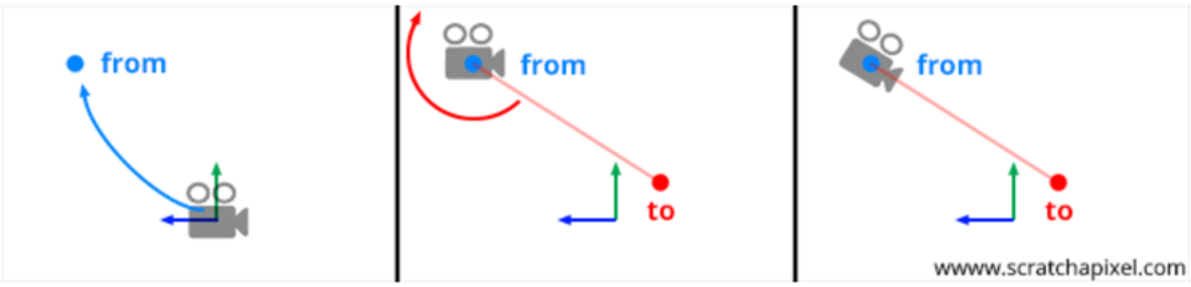
[?].

To maintain consistency when describing positions, it's important to adhere to these rules in both a coordinate system with X, Y, and Z axes and a polar coordinate system with azimuth, elevation, and distance. While some systems may not have a spot, assuming the middle of space as the origin would be a reasonable choice.

Azimuth refers to the angle that, when viewed from above, shows angles to the left (or counterclockwise) while 0 degrees indicates straight ahead. Elevation, on the other hand, represents an angle with upward positive angles and a horizontal angle of 0 degrees in front of it. Distance is measured as a normalized distance with 1.0 representing the radius of a sphere [4, **?**].

Look_at_Function

Positioning the camera in a 3D environment is crucial. However, in Scratchapixel tutorials, experts often establish the camera's position and rotation using a 4×4 matrix known as the camera-to-world matrix (remember that scaling a camera doesn't make sense). Nevertheless, manually creating this 4×4 matrix can be challenging. Fortunately, there is a technique called the look-at approach. The idea behind it is simple; you only need two points in space to configure the camera's orientation and position—one point representing where the camera is located and another point indicating its target or aim. Let's refer to these points as "from" and "to" respectively [0] `www.scratchapixel.com`.



The look-at approach is generally straightforward and effective; however, it does have one drawback. The axis that moves forward comes close to the axis we use to calculate the axis when the camera is vertical and facing directly up or down. When these two axes are parallel, such as when the forward vector's either $(0, 1, 0)$ or $(0, -1, 0)$, it represents a situation. However, in this case, the cross product cannot provide a vector. Unfortunately, this problem cannot be fully resolved.

To locate any point on the sphere, elevation (elev) and azimuth (azim) are sufficient. However, they do not allow for all viewing angles centered on an object. By introducing a roll parameter that rotates the camera around its viewing axis [25], the look_at_view_transform function can cover all perspectives centered on an object.

3.1.7 Antialiasing

Antialiasing is a computer graphics method that removes aliasing effect. The appearance of jagged edges, or "jaggies," in a rasterized picture (an image generated using pixels) is the aliasing effect. The problem of jagged edges is caused by picture distortion when scan conversion is performed with low-frequency sampling, commonly known as Undersampling. When real-world objects with smooth, continuous curves are rasterized with pixels, aliasing occurs.[22]

There was prominent obstacle during the data generation process. This was the presence of jagged lines in the rendered images. These distorted edges are often caused as a byproduct of the rasterization process and can severely impact the quality and the accuracy of the subsequent analyses. For this project, the issue was counteracted by using the downsampling approach. Initially, the images were rendered at a significantly higher resolution of 960×960 pixels. Subsequently, the images were downsampled to the desired resolution of 320×320 pixels. The rationale behind this approach is that higher resolution images are more detailed - due to their pixel count - therefore minimising the effects of jaggies lines in the final product. The function used to carry out this process is called *pxtorch.nn.Functional.interpolate*, where the $mode = area$ setting was applied.

This ensured the most optimal outcome, where the resulting downsampled images retained maximum detail while effectively minimizing the jaggedness.

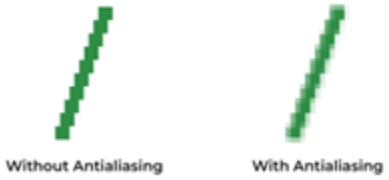


Figure 3.3: Anti-Aliasing [22]

3.1.8 Masks generation

The chessboard images were rendered using a pure green background (similar to a green screen). Therefore, for each other the rendered iterations of the chessboard it was easy to extract binary masks of where the chessboard is located in the image by filtering, using the green channel of the RGB images.

Using the 3D world coordinates of the chessboard corners, for each camera (intrinsics)

and view (extrinsics) I projected the corner points onto 2D image space and generated a grayscale image. This mask for the corners was created by placing circles of radius 3 pixels at the corner positions, and choosing the intensity of the grayscale value depending on how far the pixel was from centre pixel (as long as it was within a radius of 3). Thus, for each rendered image - along with it - were generated two binary masks. One locating the chessboard in the image and the other one locating the inlier corners (crosses of the chessboards) in the image.

3.1.9 Data split

For the 2000 cameras in the training set, I generated 45 images for each camera. For the 2000 cameras in the validation set, I generated 15 images each, and similarly, for the 2000 cameras in the testing set, I generated 15 images each. Thus, the distribution was in a ratio of 45 : 15 : 15 or a percentage equivalent of 60% : 20% : 20% for train, validation, and test splits, respectively. Along with the rendered images for each set, there is also a CSV file. Each row in this file corresponds to an image and consists of the following attributes/columns: *image_name*, fx, fy, cx, cy, *rodrigues_0*, *rodrigues_1*, *rodrigues_2*, tx, ty, tz, *corner1x*, *corner1y*, *corner2x*, *corner2y*, *corner3x*, *corner3y*, *corner4x*, *corner4y*, *rows*, *columns*, and *square_size*. These represent the name of each rendered image file, its intrinsic parameters, and its extrinsic parameters. Which are the name of each rendered image file, its intrinsic parameters, its extrinsic parameters. Where rotation is stored in axis angle form, the normalized axis $[rodrigues0, rodrigues1, rodrigues2]$ gives the axis of rotation, and the magnitude of this vector gives us the angle of rotation. The corner coordinates are respectively of the x and y coordinates in image space of the 4 main corners of the chessboard. Starting from the top right corner, then bottom right corner, then top left corner and lastly the bottom left corner. The rows, columns and square size give us information about the chessboard object viewed for a given image.

3.2 Architecture and learning choices

3.2.1 Data normalisation

When loading the CSV files with the information about each image, as described in the previous section, I decided to normalize the fx, fy, cx, cy, *rodrigues0*, *rodrigues1*, *rodrigues2*, tx, ty, and tz. This normalization was achieved by computing the mean and

standard deviation of each attribute using the training set. The normalization formula used was given by:

$$\text{normalized} = \frac{\text{unnormalized} - \text{mean}}{\text{std}}$$

This essentially zero-centres the outputs. This decision stemmed from the observed disparity in scale among the parameters, which can be detrimental during the neural network training phase. The act of normalization is a widely accepted preprocessing measure in the domain of neural networks, primarily due to its efficacy in expediting training and ensuring convergence. The merits of normalization, encompassing facets such as swift convergence and training solution stability, stand as foundational knowledge in the realm of neural network training. A plethora of normalization techniques have been proposed, a comprehensive overview of which can be found in *this research paper*.

Pytorch: The training process was facilitated using the PyTorch library. By leveraging its CUDA support, the training was considerably accelerated on a 12GB RTX-3060 GPU.

3.2.2 Augmentations

Augmentations help avoid the network overfitting the seen training data, while allowing the network to generalize better. For the chessboard images the alternating square colours were rendered in pure black and white, which might not be the case with real world chessboards. Thus I applied colour augmentation to each chessboard image by randomly replacing the pure white RGB values $[255, 255, 255]$, with a randomly sampled "whiteish" RGB $[w, w, w]$ where w was sampled uniformly from the range $[200, 255]$. Similarly, the black squares were colour augmented by blackish colours by replacing the pure black RGB[0,0,0], by RGB[b,b,b] where b was uniformly sampled from the range $[0, 70]$. This created augmented chessboard colours. At this point I had images of colour augmented chessboards, however with a fixed solid fill background. So, next I applied some background augmentations. Using 29 possible background images, I used the chessboard masks generated along with the rendered images. This was done in an effort to extract the portion of the image containing the chessboard and superpose on the background image. However before doing so, since I only chose 29 possible background images, I decided to use these 29 images to generate a bigger range of possible backgrounds.

Thus, I augmented the background images as well: this is done in order to make the model more robust by providing different training information. Some of the examples

are: variation in the orientation by flipping the background horizontally or vertically. Tilt simulation by rotating the background by up to 20 degrees. Resizing to emulate different viewpoints (can be scaled between 30% and 100% of its original size, and the aspect ratio of this crop can vary between 0.75 and 1.33). Lastly, lighting conditions were emulated by changing attributes like brightness, contrast randomly between 0 and 0.5. Additionally changes in saturation and hue parameters (between 0 and 0.3), changes in colour palette helped create different environment tones and camera colour calibrations.

This gave rise to colour augmented images of chessboards with varying backgrounds. The last augmentation I applied was to simulate blur and colour jitter. This is possible through 2 techniques: blurring and colour adjustment. This resulted in the creation of images, with backgrounds and enhanced colours. The final step I took was to add a blur and adjust the colours. During this process two main modifications are made to the image; blurring and colour adjustment. To achieve blurring a Gaussian Blur transformation is applied using a kernel size and sigma value. The kernel size was randomly chosen from numbers between 3 and 7 to determine the size of the window used for blurring- where larger values create blurring effects. Similarly, Sigma values within a fixed range of 0.01 to 4 were used to determine the deviation of the filter. A higher value spreads out the blur widely while a lower sigma results in a subtler blur effect. After applying the blur I made colour adjustments to the image by altering brightness, contrast, saturation and hue levels by amounts. Brightness is adjusted within a change of 30% affecting how dim or bright the image appears. Contrast is also limited within a range to control variations between light areas in the image. Saturation influences how vibrant or muted colours appear in the image while hue allows for shifting, up to 15% to alter its colour theme. As a result of these carefully selected parameters to implement the modification - such as limiting the kernel size and the range of colours – I was able to introduce controlled randomness. The purpose of this uncertainty is to generate variations while still maintaining the characteristics of the original image. These transformations imitate real life differences found in images, due to factors like lighting conditions, camera configurations or subtle variations, in scene colours.

3.2.3 Dice loss

In the field of computer vision, the Dice coefficient is an used statistic to determine the similarity between two images. It was later transformed into a loss function called Dice Loss in 2016 [0].

The Dice coefficient calculates how much two sets, A and B resemble each other. A

represents the predicted segments while B represents the ground truth segments. The coefficient ranges from 0 to 1 where 0 indicates no overlap between the sets and 1 signifies equivalence.

The formula for calculating the Dice coefficient (DSC) is as follows:

$$DSC = \frac{2 \times |A \cap B|}{|A| + |B|.\varepsilon}$$

In this project, ε is set to be equal to 10^6. Here, $|A|$ represents the number of items in set A, $|B|$ denotes the number of elements in set B, and $|A \cap B|$ indicates how many items are common to both sets [0].

Based on this formula, we can determine the Dice Loss:

$$\text{Dice Loss} = 1 - DSC$$

To ensure that extreme cases do not lead to results, ε is added to the denominator. Thus,

$$\text{Dice Loss (DL)} = 1 - \frac{2 \times |A \cap B|}{|A| + |B| + \varepsilon}$$

When a dice loss of 0 is obtained, it means that there is an overlap between the segmentations of predicted and ground truth images. There is no indication of overlap based on a Dice loss value of 1.

3.2.4 MAPE loss

The *Mean Absolute Percentage Error* (MAPE) is an used measure, for evaluating the accuracy of forecasting models. It calculates the percentage difference between the predicted values and the actual values. In terms it measures how much the predictions deviate from the observed values in percentage terms. To calculate MAPE we find the difference between each observed value and its corresponding predicted value then express it as a percentage of the value. By averaging these percentage errors across all observations we obtain the MAPE score. Lower MAPE values indicate a fit between predictions and actual values with an ideal MAPE of 0% indicating alignment, between predictions and observations.

3.2.5 Projection loss 2D

The reprojection error is a tool, for assessing the accuracy of parameter estimation. It measures how close the transformed object point aligns with the corresponding image point taking into account distortion, rotation and translation matrices. By comparing the results of the corner finding procedure with this transformation we can calculate the norm to determine the level of precision. To obtain an error value we compute the mean of these errors across all calibration photos [0].

Reprojection errors highlight any disparities between a 3D position and its indicated position in an image [0].

In camera calibration and computer vision understanding reprojection error is crucial for tasks like 3D reconstruction and camera pose estimation. It quantifies the difference between marked points and their reprojected counterparts, on an image.

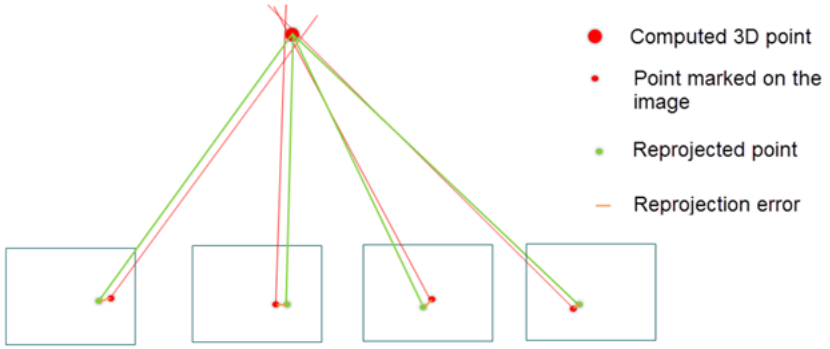


Figure 3.4: assesses the difference, between the points captured or detected in an image and the corresponding points that are projected from world coordinates onto the image using the estimated camera parameters. [0]

By utilizing the camera calibration settings we can project the resulting point P onto the cameras again which yields new points $\hat{p}$ that are close to the initial ones p. Subsequently we calculate the distance, between the "reprojected" point and its original counterpart.

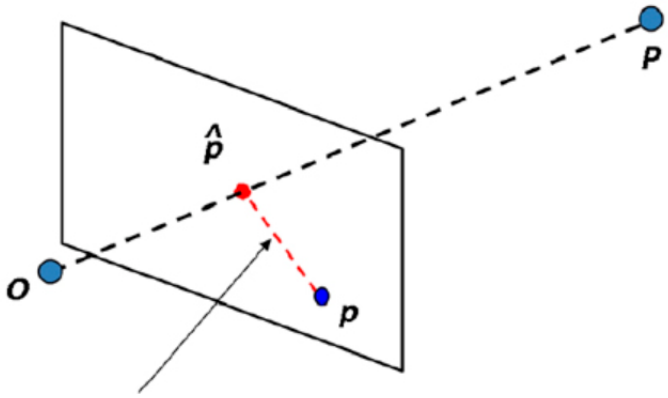


Figure 3.5: showcases an example of a reprojected point. [0]

3.2.6 Projection loss 3D

The 3D reprojection error, in camera calibration, is a metric used to measure how accurately a 3D point is projected onto its 2D position in an image. To calculate this error we begin by selecting a known 3D point in the world that corresponds to a 2D point in the image. Using the extrinsic parameters of the camera, we then project this 2D point back into the space. The discrepancy between this projected 3D point and its true location represents the 3D reprojection error. In essence, this error helps us understand any deviations between where we expect a point to be (based on camera calibration) and its actual position, thus highlighting any potential inaccuracies in the calibration process.

Chapter 4

Experiments and Results

4.1 Segmentation trained with vs without noise/blurring

Here I experimented to see how the colour jitter and Gaussian blurring augmentations mentioned in the previous chapter affect generalisation and performance. I trained a UNet inspired model (see **Unet architecture** for more details) twice; one time with on the training images without the colour jitter and Gaussian blurring augmentations and the second time with the augmentations applied. The UNet architecture has an encoder and decoder that respectfully downsample and and upsample the signal passing through the network. Specifically, the designed architecture consists of 4 double convolutional blocks (dconv_down1 to dconv_down4) with max pooling(kernel size 2by2) inbetween them to downsample the spatial information while gradually doubling the channel size from the initial 64 to 512. In the decoder section, I placed 3 double convolutional blocks (dconv_up3 to dconv_up1 in the architecture diagram) that gradually halve the channels from 256 to 64 while doubling the spatial dimensions using bilinear upsampling. As in the standard UNet we have skip connections from the encoder outputs concatenated to the input of each of the decoder components that allow the model to combine high and low level features. Dropout layers were also used with dropout probability of 10%. Lastly, the architecture makes use of 2 separate heads each consisting of a single 1by1 convolution responsible for sliding over the feature map(which after the downsampling and upsampling has the same spatial dimension as the input image) with 64 channels and aggregating the information from these 64 channels into a single channel producing a mask prediction consisting of

probabilities at each pixel describing whether the network believes the pixel belongs to the class of interest. To ensure the 2 mask outputs of the last layer at each pixel are between 0 and 1, I used a sigmoid function to remap the outputs to this range. The model was trained on the chessboard images to output two separate masks, one locating the chessboard mask and the other locating the inlier points of the chessboard.

4.1.1 Training setup

For both training sessions the batch size was fixed at 12 images using the SGD optimiser with learning rate 1e-1 and momentum 0.9. For each image in the batch, the dice loss function was used to compare each of the 2 predicted masks with the 2 ground truth masks and the sum of the values for the dice loss (for the chessboard mask and chessboard corners) was used as the overall training loss. The models were both trained for 7 epochs. The idea behind having two heads connected to one network is that of tackle this multi-task learning problem while exploiting what the model learns for each task with each other. Intuitively, learning to predict the chessboard position is a useful skill in trying to predict the chessboard corners and vice versa (as the features the network will learn should have a significant overlap).

4.1.2 Performance

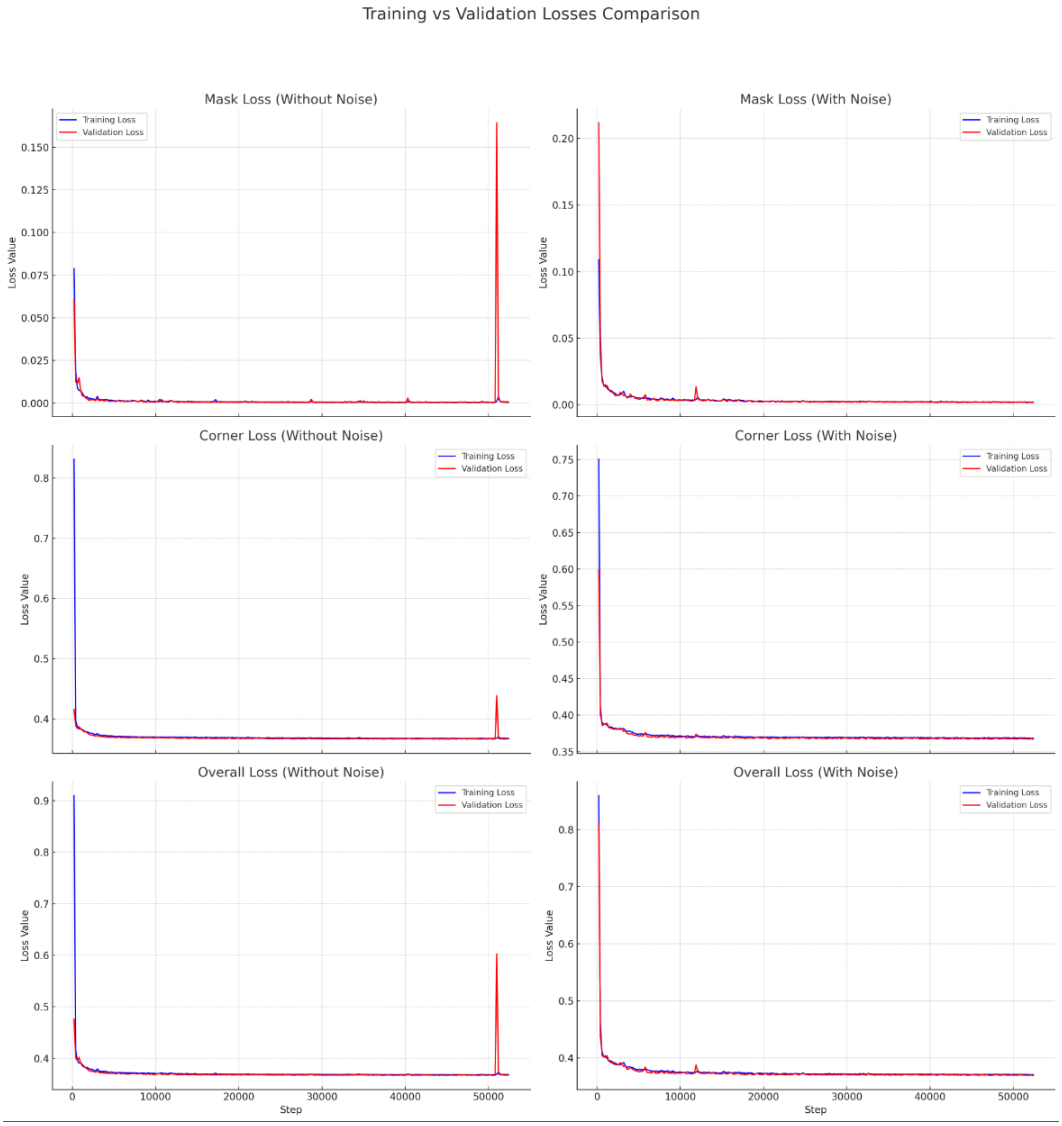


Figure 4.1: Training and validation losses for the two models

	Mask Loss	Corner Loss	Overall Loss
Without Noise	0.000405336	0.366651	0.367056
With Noise	0.00175417	0.367043	0.368798

Table 4.1: Final validation Dice loss values for models trained with and without noise. Lower values indicate better performance with a perfect overlap having a loss of 0.

Looking at the loss values the mask losses are very close to 0 which indicates the model is very good at detecting the chessboard and gets an almost perfect overlap. For the corner losses, both models end with values close to 0.367 which while being good it is not as robust as the chessboard losses. However, this is to be expected as segmenting the corners of varying sized chessboard from different views and intrinsics is a much harder task than trying to segmenting the chessboard itself. The overall, losses remain close for both the noise model and the without noise model however the without noise model has a slightly lower value which is intuitive as the model is learning in a less varied and less difficult data space and predicting for no noise/blur images should be easier.
The without noise model seems to have an unexpected spike at the end thus I will cut the training before that and use weights from well before that point in the next section.

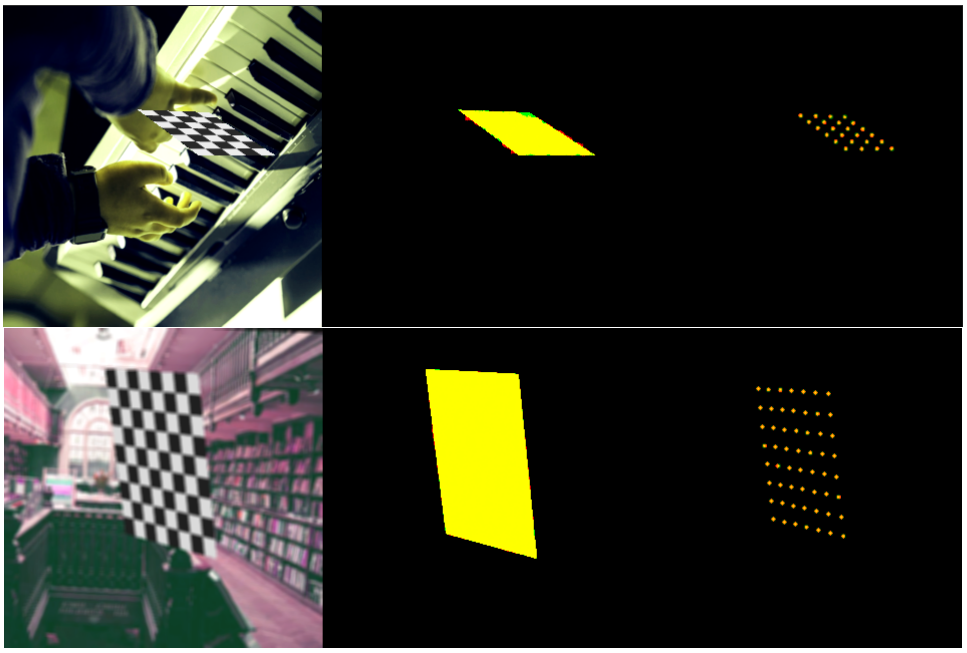


Figure 4.2: Images to showcase the training chessboard images with background. The top row is from the early training stages of the no noise model while the second one is from the later training stages of the with noise model. Note the blur in the second chessboard image. For the masks, the true binary 1 regions are coloured in green while the true binary 0 regions are coloured in black. Similarly the predicted binary 1 regions are coloured in red thus where the true 1s(green) and predicted 1s (red) overlap the colours add up to yellow. Thus visually the green regions represent the regions missed by the model and the red spots represent where the model predicted 1 wrongly. The first row from earlier training stages is returned just to demonstrate such.

4.1.3 Evaluation on rendered test set with varying noise/blurring

From now on I will refer to the 3 losses as loss 1,2 and 3 for ease.

Type	Level	Loss 1 Mean	Loss 1 Std	Loss 2 Mean	Loss 2 Std	Loss 3 Mean	Loss 3 Std
Overall	-	0.75034	0.432492	0.841923	0.273841	1.59226	0.706333
Blur	0.0	0.00135826	0.00507923	0.367693	0.00371708	0.369051	0.00851357
Blur	1.0	1	2.90581e-08	1	2.12e-08	2	4.68969e-08
Blur	2.0	1	6.84032e-09	1	0	2	0
Blur	3.0	1	0	1	0	2	0
Jitter	0.0	0.750103	0.433075	0.841735	0.274275	1.59184	0.70735
Jitter	1.0	0.750126	0.433035	0.841765	0.274225	1.59189	0.707259
Jitter	2.0	0.750174	0.432953	0.841814	0.274139	1.59199	0.707091
Jitter	3.0	0.750956	0.431625	0.842378	0.273182	1.59333	0.704804

Table 4.2: Statistics for the "No Noise" model's batch losses on the test set.

Type	Level	Loss 1 Mean	Loss 1 Std	Loss 2 Mean	Loss 2 Std	Loss 3 Mean	Loss 3 Std
Overall	-	0.266468	0.42764	0.537433	0.26902	0.803902	0.696583
Blur	0.0	0.00115444	0.00077878	0.367506	0.00116619	0.36866	0.00149632
Blur	1.0	0.00418295	0.0115144	0.368214	0.00670342	0.372397	0.0180263
Blur	2.0	0.0666203	0.149596	0.419162	0.09192	0.485782	0.241113
Blur	3.0	0.993916	0.0225432	0.994852	0.0137592	1.98877	0.0351416
Jitter	0.0	0.256392	0.428447	0.531478	0.269325	0.78787	0.69772
Jitter	1.0	0.2568	0.427959	0.531592	0.269017	0.788391	0.696916
Jitter	2.0	0.264887	0.425391	0.536341	0.267444	0.801228	0.692758
Jitter	3.0	0.287796	0.428703	0.550322	0.270293	0.838118	0.698881

Table 4.3: Statistics for the model with noise's batch losses on the test set.

As expected the model with noise generalises better and the model without noise has average losses higher than its counterpart as it is less robust to noise due to not having encountered during training. For both the models, increeasing blur has a significant effect and increase on the loss values. The model with noise showed the biggest negative effect at the highest level of blur which is to be expected as during training it only encountered levels of jitter and blur up to 2. The effect of jitter is nowhere near as strong as the effect of blur. The standard deviations remain relatively low across the different levels indicating a somewhat consistent performance.

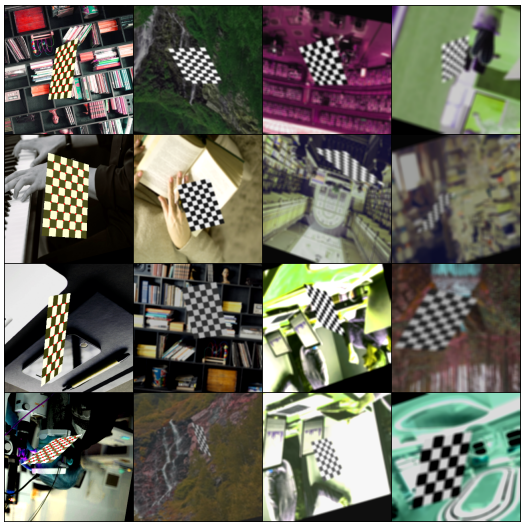


Figure 4.3: No noise model on rendered test set. Increasing blur from left to right, increasing jitter from top to bottom. Yellow superposition of where the model predicts the chessboard mask to be and in red where the model predicts the corners to be.

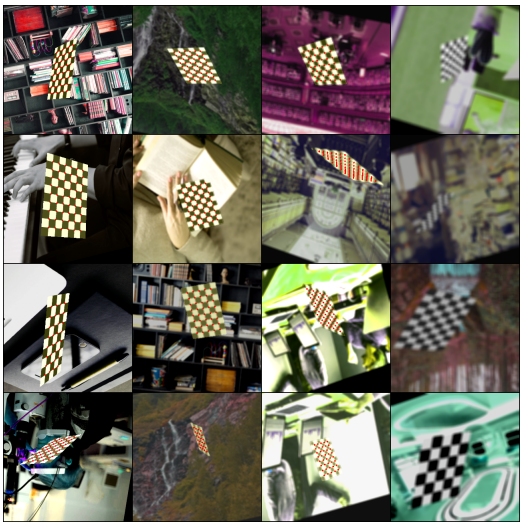


Figure 4.4: With noise model on rendered test set. Increasing blur from left to right, increasing jitter from top to bottom. Yellow superposition of where the model predicts the chessboard mask to be and in red where the model predicts the corners to be.

4.1.4 Evaluation on real test set with varying noise/blurring

For the real images test we don't have ground truths thus we can only get qualitative results.

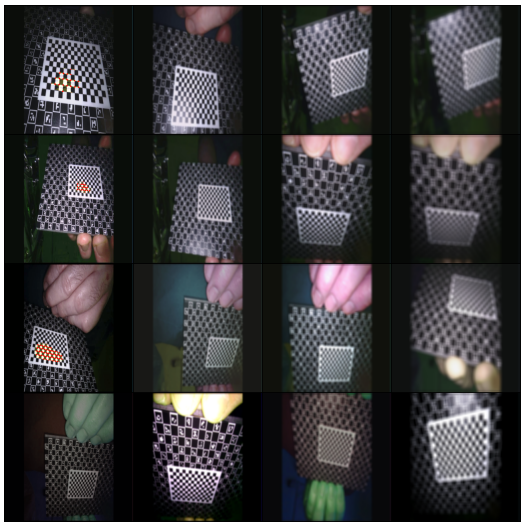


Figure 4.5: No noise model on real test set . Increasing blur from left to right, increasing jitter from top to bottom. Yellow superposition of where the model predicts the chessboard mask to be and in red where the model predicts the corners to be.

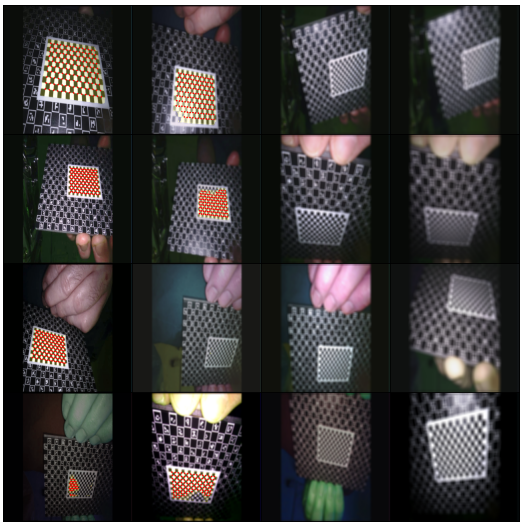


Figure 4.6: With noise model on real test set. Increasing blur from left to right, increasing jitter from top to bottom. Yellow superposition of where the model predicts the chessboard mask to be and in red where the model predicts the corners to be.

4.1.5 Discussion

As expected the model trained with noise generalises much better. The qualitative results on the real dataset seem to indicate that the performance of both models deteriorates from the rendered case to the real case which can be attributed to distribution shift (for example the rendered images were not subjected to light reflection off the chessboard which we can see in some of the real images).

4.2 ParamNet loss

In this section we try to predict the intrinsic and extrinsic ground truth values of our rendered chessboard images using a ResNet based model (more specifically see **ResNet50**

based for more details). Here I decided to utilise the predefined Resnet50 class from the PyTorch library and switched the final few layers. The final layer is originally a fully connected classification layer that I removed and took the output of the ResNet layer previous to this and flattened it. Thus the ResNet became a feature extraction backbone. I took the feature vector from this backbone and concatenated it with a single scalar value (i.e. the square size of the chessboard viewed in the input image) and passed it through a series of fully connected layers(roughly halving the feature vector length) alternating with ReLu activation functions. Lastly, the network split into two fully connected layer heads, the first one regressing the 4 intrinsic parameters and the second one regressing the 6 extrinsic parameters. We investigate the choice of loss function from 3 very intuitive options.

4.2.1 MAPE

The first loss function simply computes the percentage error between our predicted values and the true values. Here I set the training loop to run for 4 epochs with batch size 30 and learning rate 1e-4 with SGD optimiser and momentum 0.9. However, no matter the training configuration changes (SGD VS Adam or different learning rates/momentum) the network did not show hints of learning behaviour as visible in **the training and validation losses in the early epochs** below.

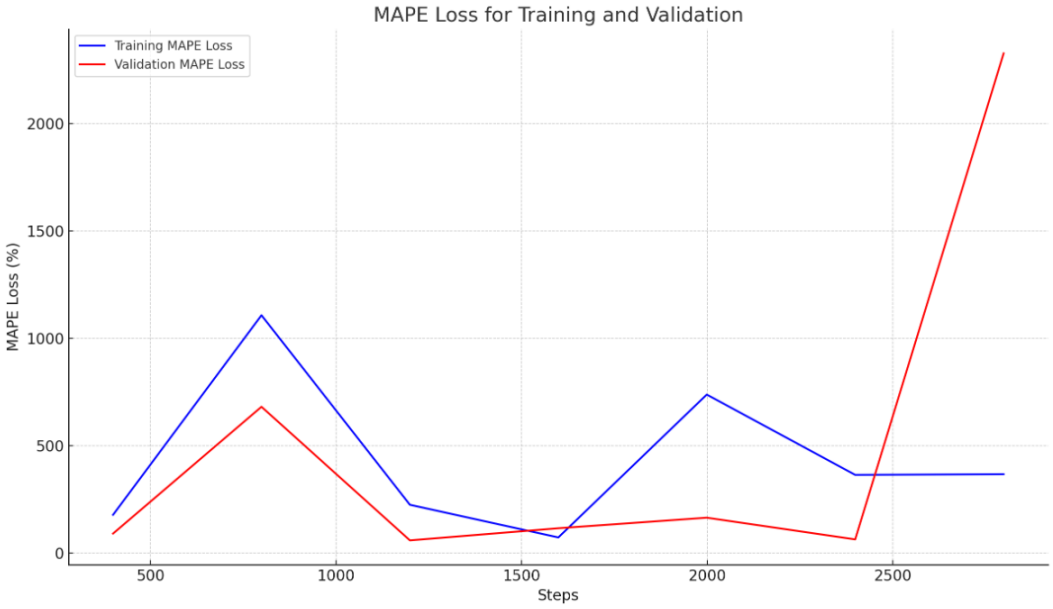


Figure 4.7: Training and validation MAPE losses

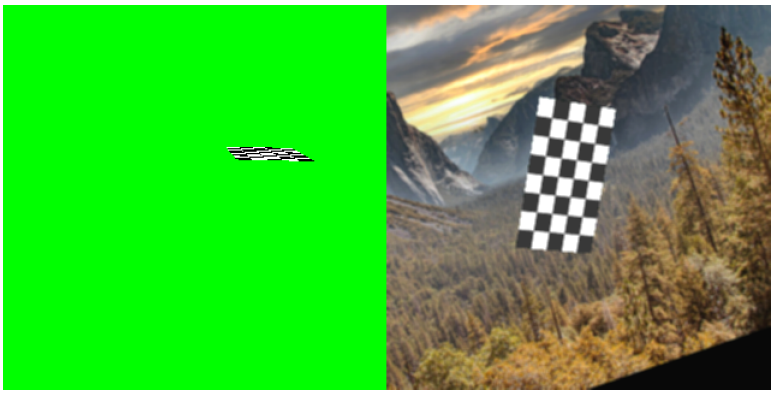


Figure 4.8: Projection of chessboard into 2D space using the parameters found by training on MAPE.

4.2.2 Projection loss in 2D

Next I decided to investigate the Projection loss in 2D, inspired by traditional camera calibration as this is often used as a metric for judging the success of a calibration. The training setup remains the same here. This choice of loss function showed much better learning behaviour.

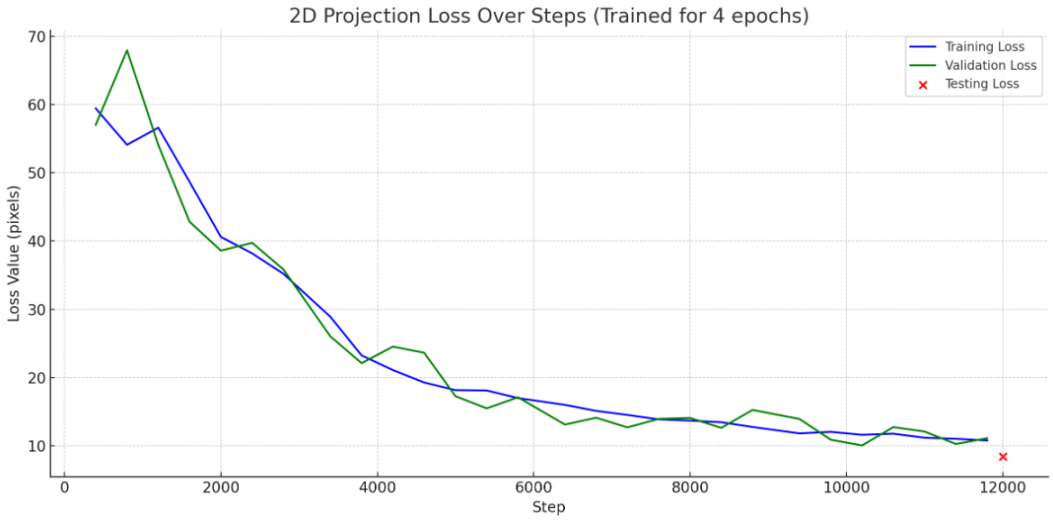


Figure 4.9: Training, validation and test 2D projection losses

Figure 4.10: Projection of chessboard into 2D space using the parameters found by training on 2D projection error.

4.2.3 Projection loss in 3D

Lastly, I decided to investigate using the 3D projection loss (equivalent of the previous loss but with the euclidean distance considered in 3D space rather than 2D) for training.

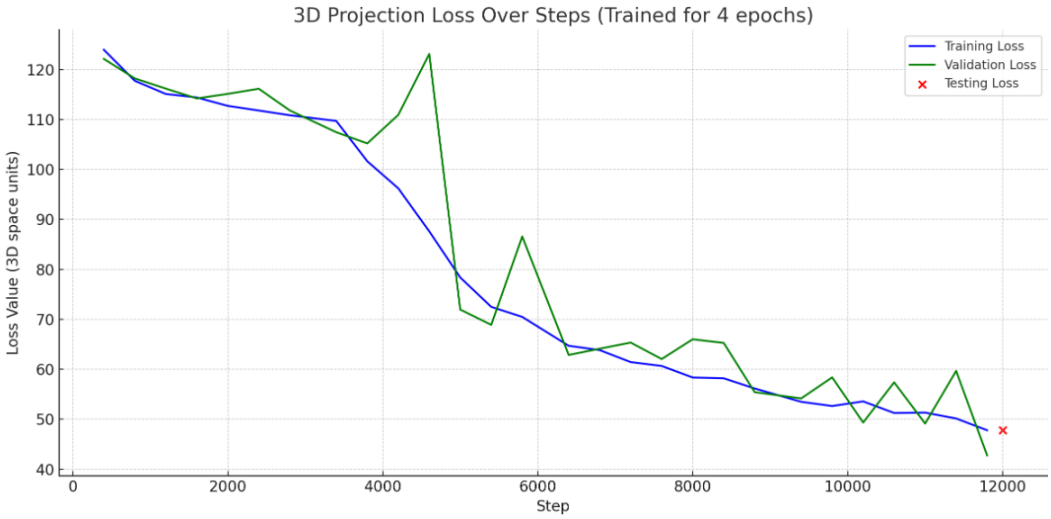


Figure 4.11: Training, validation and test 3D projection losses

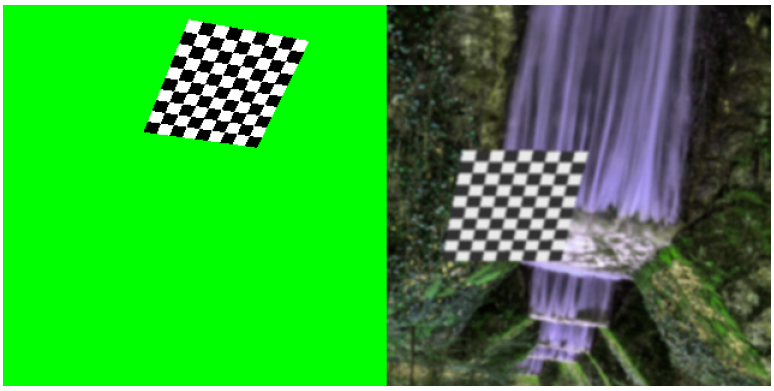


Figure 4.12: Projection of chessboard into 2D space using the parameters found by training on 3D projection error.

4.2.4 Discussion

The 2D projection loss showed the best learning behaviour out of the three. Inspecting the 3D training and validation losses, we see the network learns to decrease the loss however the process shows instability. Minimising a 3D based distance metric from looking at a 2D image might be too challenging for the network. Looking at the projections of the chessboard into 2D image space using the found paramaters next to the input images the network predicts from, again the 2D projection loss based scenario seems like the best option of the three.

4.3 ParamNet with masks vs without

When doing a classic camera calibration, the first step taken by OpenCV is to detect the chessboard inlier corner points so that it can set up correspondences with the real world 3D object. Inspired by this process, I investigated whether providing the masks locating the chessboard corners and chessboard helps the network to learn better or quicker. Intuitively, the chessboard mask should tell the network what part of the image to pay attention to and the corners' masks should tell the chessboard what parts of the chessboard to pay attention to and extract features from. To do this, I decided to combine the segmentation UNet model with the ResNet regression model (**see here**). To do this I took the UNet and ResNet based architectures discussed previously, passed the input image through the UNet portion which gave two 1 channel binary masks. Then I concatenated these two masks to the input image along the RGB channel which changed its dimension from 3 to 5 (and modified the ResNet model to handle this change)before passing it to the ResNet part just as before.

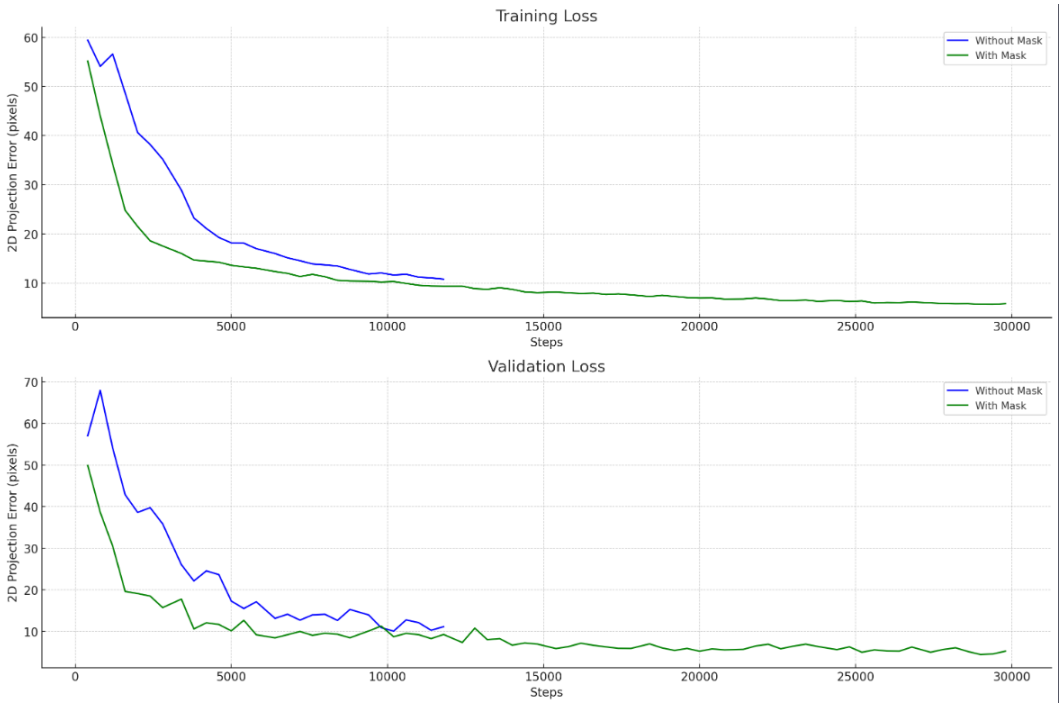


Figure 4.13: 2D projection error for UNet+ResNet vs just ResNet

4.3.1 Discussion

It is clear that the model benefits from explicitly extracting the corner and chessboard location masks through the UNet first. The addition of the masks to the image input result in a faster and more stable convergence.

4.4 ParamNet depth

Lastly, I just investigate whether there is any benefit in deepening the ResNet model from ResNet50 to ResNet152.

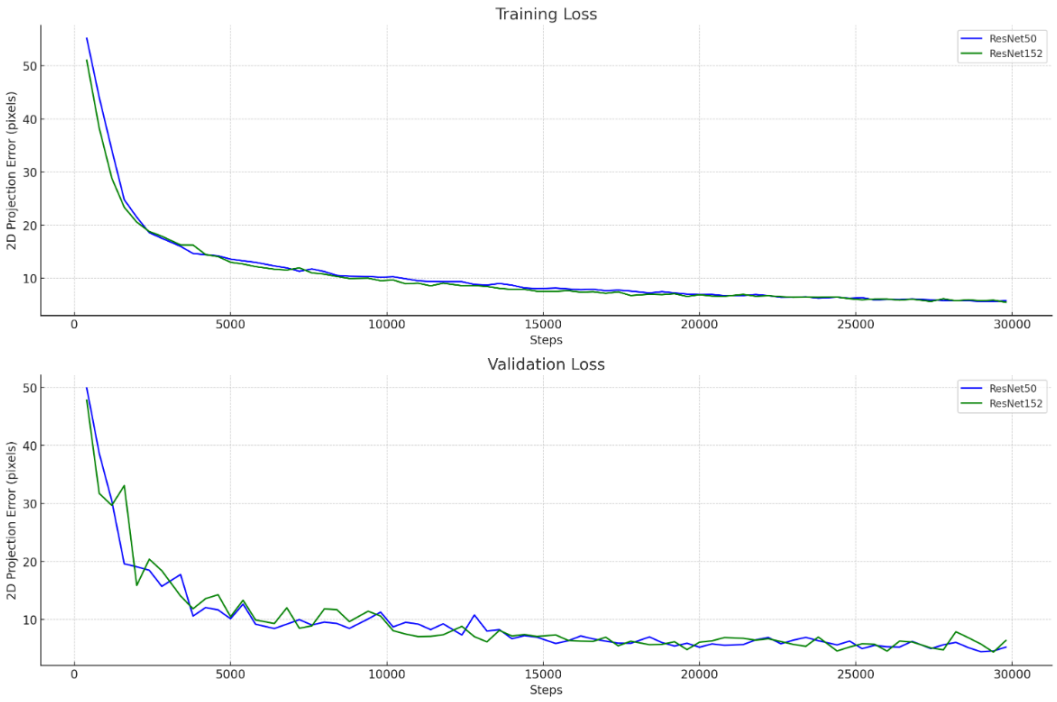


Figure 4.14: 2D projection error for ResNet50 vs ResNet152

4.4.1 Discussion

There is no real added benefit from deepening the network. Though the ResNet152 initially learns quicker both models converge to the same loss and looking at the validation losses the shallower model seems to be more stable.

4.5 OpenCV

In this part I investigated the accuracy and robustness of the openCV calibration. OpenCV camera calibration provides a set of intrinsics and extrinsics for each image along with the reprojection error as a metric of calibration quality. However, this is just a measure of how well the 3D object points (inlier corners of chessboard) project back near the correspondence points found by the corner finding algorithm. Thus this is highly dependent on the quality of the correspondence points detected and as we will soon see, there are possible complications. One very important thing to note is that the way we're testing the calibration process in this section is very unconventional in the sense that in the real world a human input is present at each step of the calibration process(for example to discard poor images or poor correspondence points) however it is necessary to take this approach as it would impossible to test all the cameras in the test set by hand.

4.5.1 Number of images

We first investigate the effect of the number of images provided for a openCV camera calibration using our test set. Our test contains 2000 simulated cameras, each with 15 images of chessboard from a range of views. For each camera, we respectively pass 1,5,10 or 15 images(i.e requested images) and return the results below.

Metric	Mean	Standard Deviation
OpenCV Predicted Error (pixels)	0.304846	0.295331
2D Projection Error (pixels)	45.661351	49.352465
3D Reprojection Error (3D spcae units)	111.603902	163.141410
MAPE (%)	409.520709	3759.569964

Looking at the table it is obvious how poorly openCV performs without human assistance and how it can easily be unaware of its faults. The true mean 2D projection error is much higher than what openCV think it is and looking at the other two metrics we can see that openCV performs quite poorly on the test set. Looking at the figures below, we can see that the calibration process totally fails often especially when given low number of images to work with (which is to be expected). Looking at the the frequencies of number of images used during calibration, we can see that it mostly uses low number of images even though it is often being passed 10-15 images, the calibration for the vast majority of the time utilises less than 3 images. This is indicating that the calibration process (or rather the corner finding algorithm) struggles in finding the chessboard corners for the test

set.

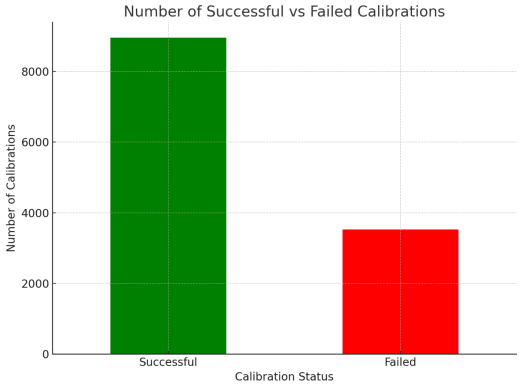


Figure 4.15: Succesful and failed calibrations.

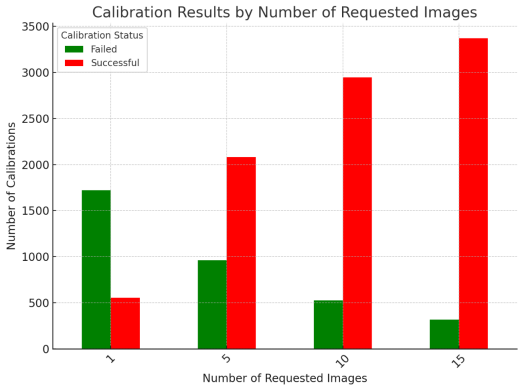


Figure 4.16: Succesful and failed calibrations by requested images.

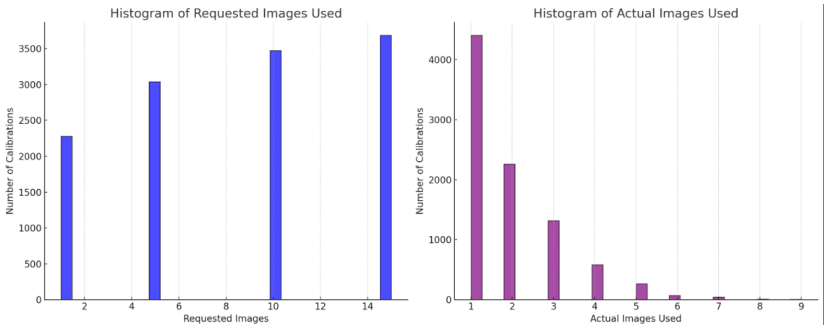


Figure 4.17: Requested and actual images used for calibrations.

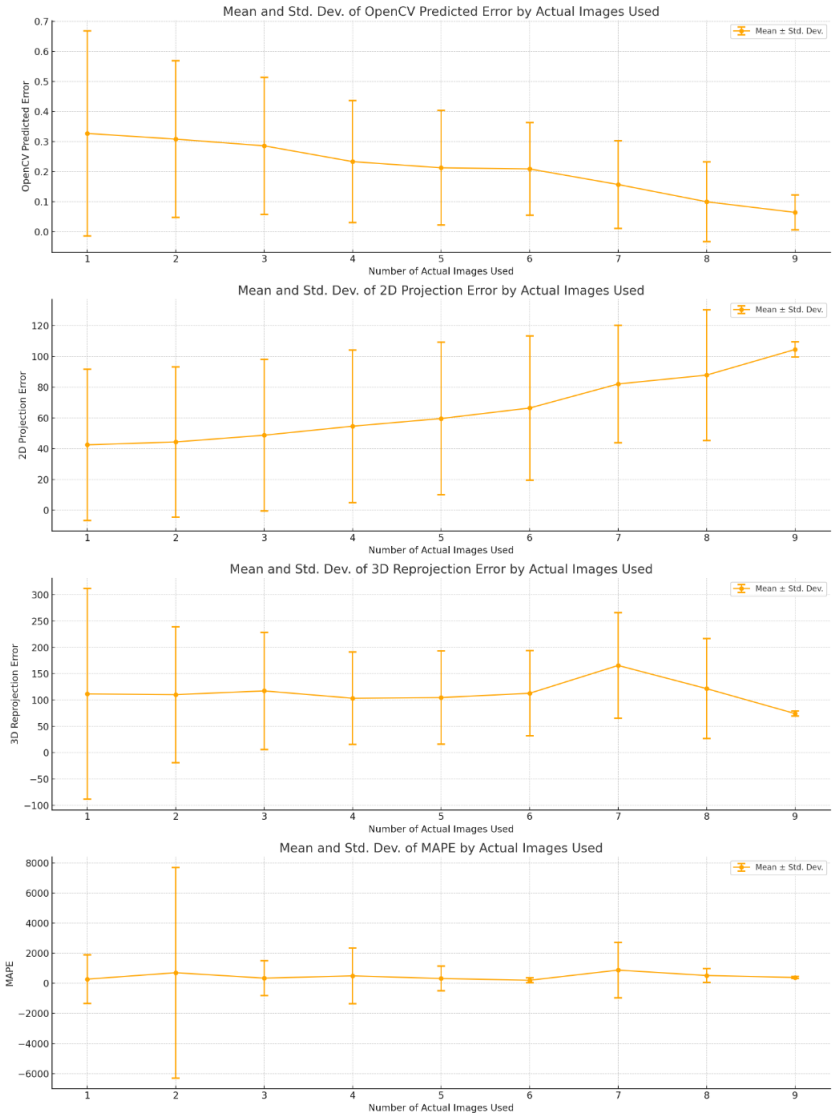


Figure 4.18: Mean and standard deviation of errors.

As expected, the errors tend to be more consistent (smaller standard deviation) the more images used during calibration. What's unexpected is that even though the openCV predicted error shows a decreasing behaviour with number of images used, the true 2D error actually increases. This is counterintuitive however I suspect there might be some images/views in the dataset that are particularly challenging for openCV to detect the corners of and as the number of images used in the calibration increases, the system just a higher chance of encountering such images and deviate the calibration.

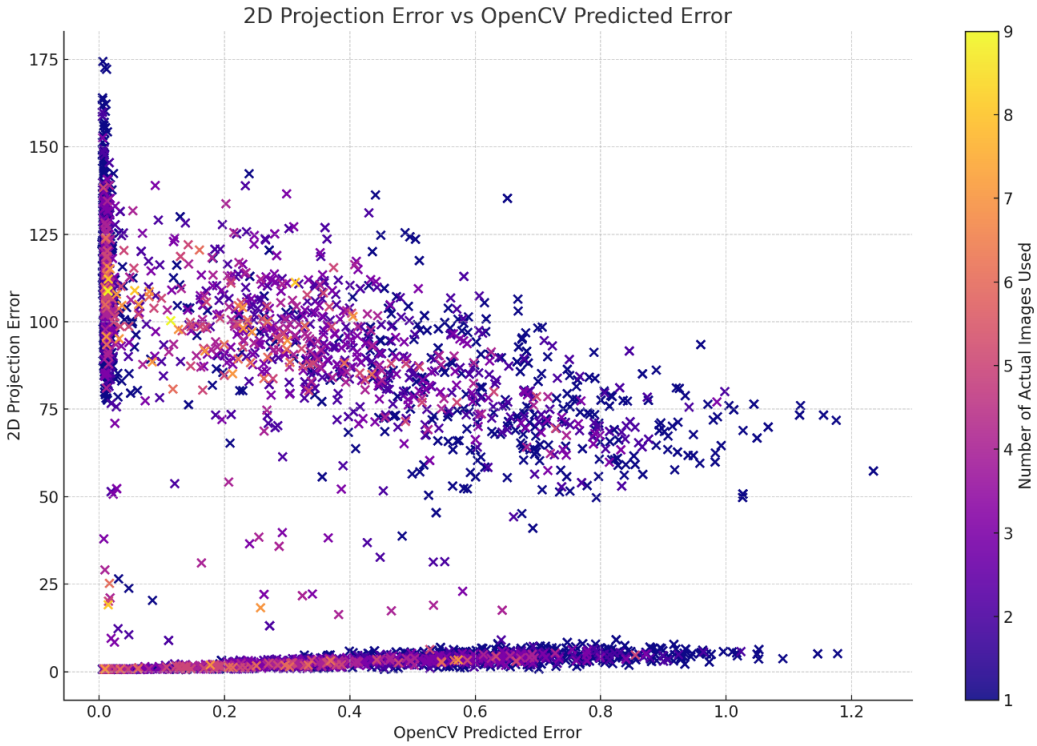


Figure 4.19: OpenCV predicted error vs true 2D projection error.

Looking at the above figure, we notice 3 main clusters of scattered points. One strip at the bottom that seems to correspond to cases where openCV seems to predict the error close to the true 2D error. One vertical strip where the errors don't match at all and where openCV believes to have a very small error but in reality the error is very large (100s of pixels instead of subpixel error). And the last band is the least concentrated.

Looking at some images with detected points from the dataset, it becomes obvious how easily the corner detection can misjudge corners or sometimes even reverse the order of detection of the corners completely which causes pixel errors near 100.

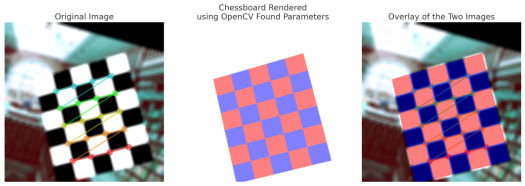


Figure 4.20: Here the white and blues squares correpond to each otehr while the black and red squares correpond to each other. The corner detection starts at the bottom right (marked by the red portion of the chain connecting the corners) instead of starting at the top left corner of the chessboard and that reverses the chessboard completely as indicated by the white squares not being aligned with the blue squares.

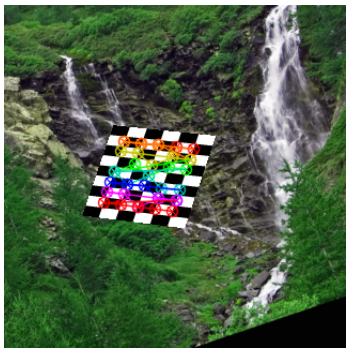


Figure 4.21: Example of how the corner detector can fail.

4.5.2 Evaluation on rendered test set with varying noise/blurring

Here I reutilise the blur and jitter transformations defined a few sections ago. As we can see, blurring has a much more detrimental effect than jitter. We also notice by comparing the distributions of opencv predicted errors and the true 2D projection errors, that again that opencv vastly underestimates the errors and it's unaware of its shortcomings.

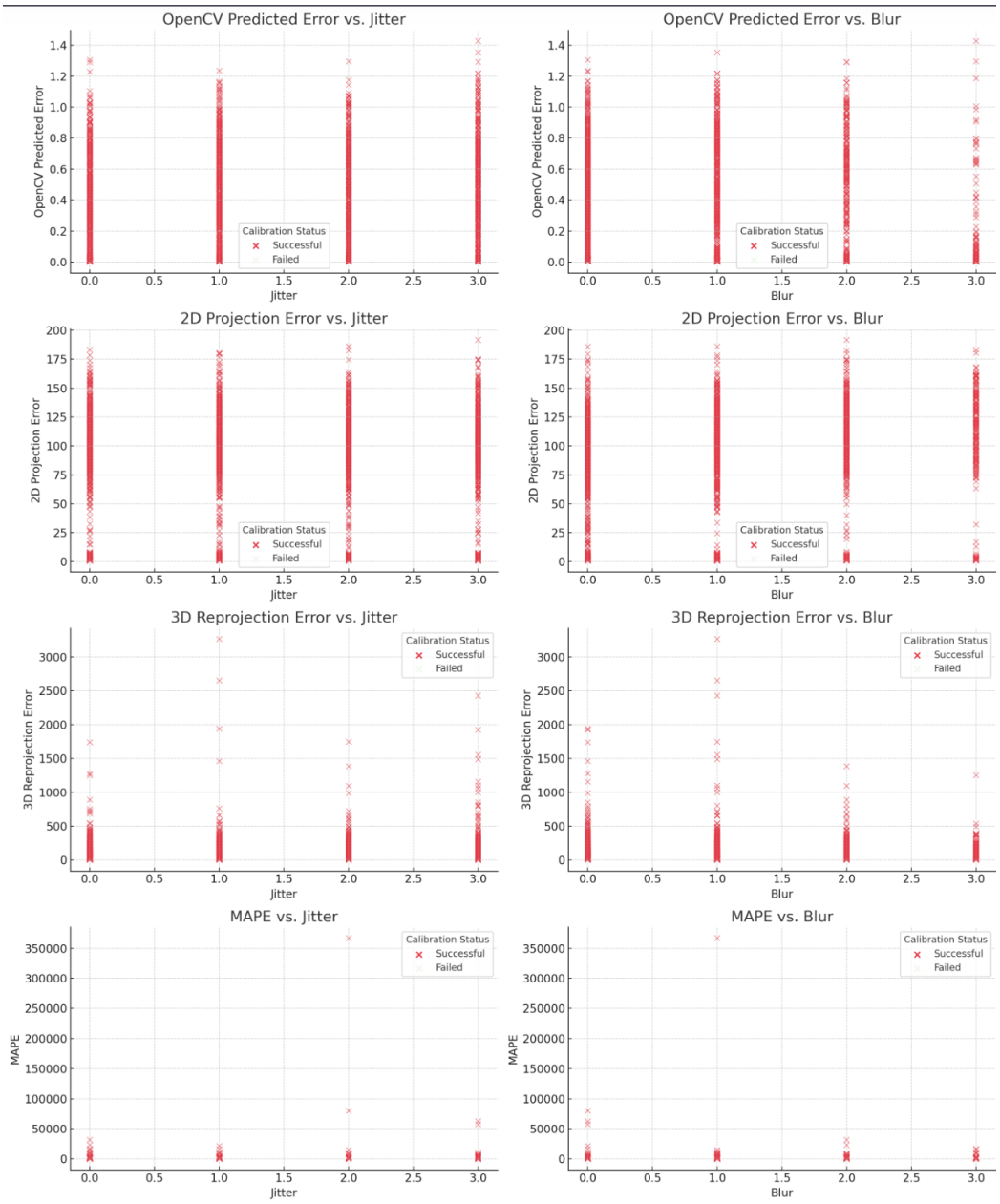


Figure 4.22: Effect of jitter and blurring on errors.

4.5.3 Evaluation on real test set with artificial varying noise/blurring

As we can see below, increasing levels of blur significantly reduce the amount of images usable for the calibration process. Furthermore, we see that theopencv predicted reprojection error goes down as blur increases but this is probably due to the decreasing number of calibration images giving less constraints and making it easier to reduce the error.

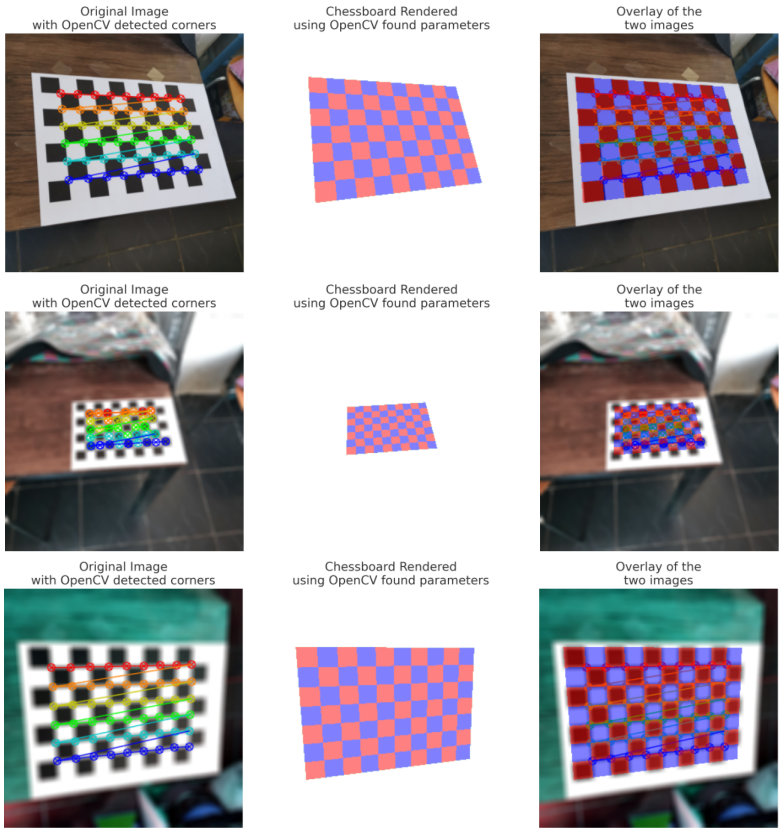


Figure 4.23: Performance of OpenCV on real images under increasing blur and jitter (top too bottom).

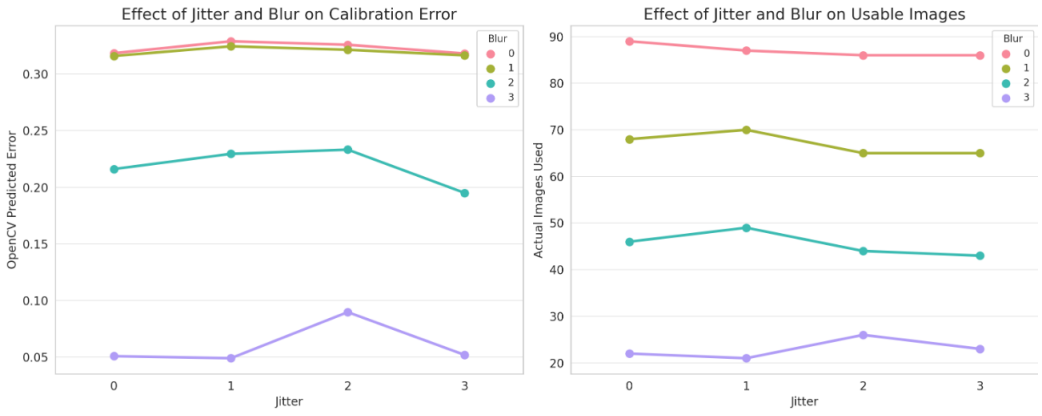


Figure 4.24: Opencv projection error and number of images whose corners get detected under different levels of blur and jitter.

4.5.4 Discussion

As seen so far, blur has a much bigger impact on the calibration process than the colour jittering. Furthermore, blur is especially dangerous because it can vastly reduce the amount of images available for calibration, which in turn reduces the opencv predicted reprojection error. We also noticed how off the opencv reprojection error can be from reality.

4.6 ParamNet

Next we evaluate the ParamNet model on the test set.

4.6.1 Evaluation on rendered test set

Metric	Mean	Standard Deviation
2D_projection_error	4.5405	0.7384
3D_projection_error	180.0848	29.6678
mape_loss	82.5590	178.1264

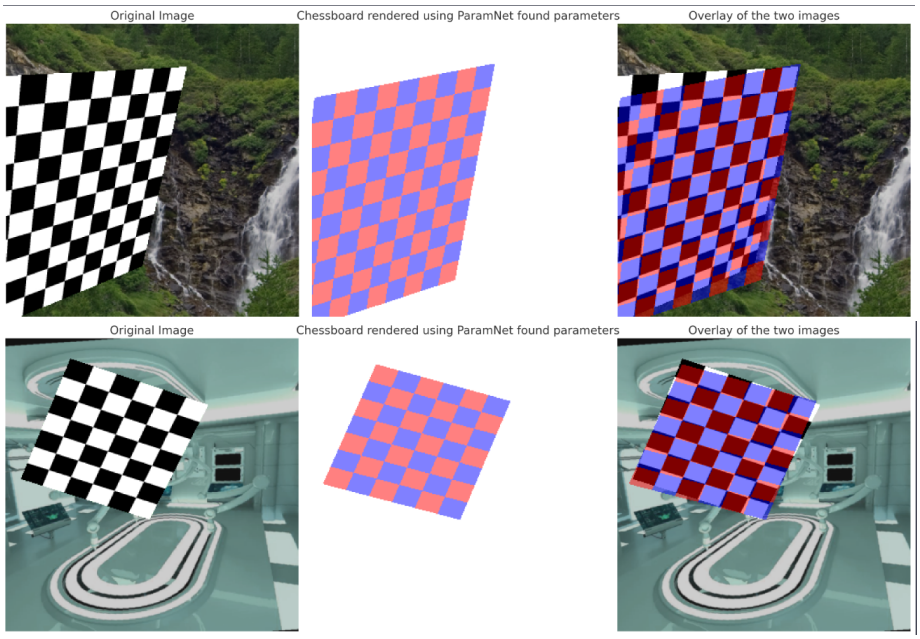


Figure 4.25: Rendering using ParamNet found parameters and comaparing to synthetic data.

4.6.2 Evaluation on real images

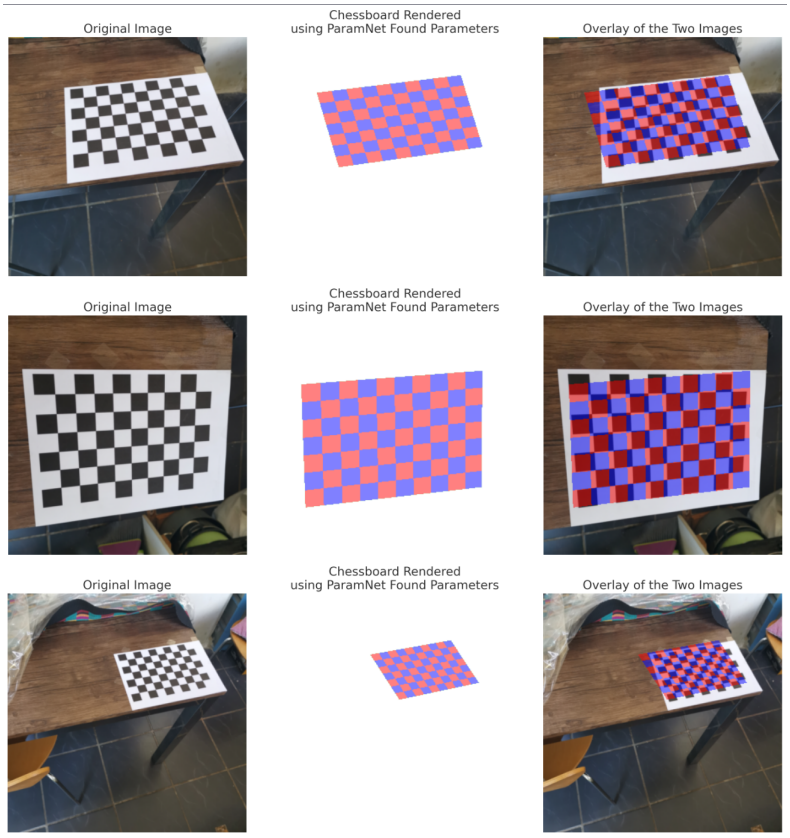


Figure 4.26: Rendering using ParamNet found parameters and comaparing to real data.

4.6.3 Discussion

As we can see the paramaters predicted by ParamNet tend to have more consistent 2D errors than openCV. However, it is important to note that openCV manages to achieve subpixel errors while the neural network consistently stays around 4.5 pixel error which is quite significant and not a very useful result. Looking at the overlayed image and predicted parameter renderings we can see that ParamNet does better on the synthetic data than the real images however the respective overlays are much better for openCV. Furthermore, there is an obvious pitfall with the ParamNet architecture that is the fact that it is trying to predict the intrinsic and extrinsic parameters from a single image view, and it it well known how camera calibration benefits from multiple images. Moving closer to the chessboard will inevitably make it look bigger however if we consider staying fixed in place but doubling the focal parameters, the resulting view might very well be the same thus when trying to

predict the intrinsic and extrinsic parameters from a single image using the 2D projection error, the network has to navigate a loss space riddled with many local minima. Thus the investigation in the ParamNet's ability to predict the parameters was purely to determine whether the system has the learning capabilities to extract features from chessboard images that help it minimise the 2D projection error. I tried training the model with a similar setup and with multiple images however the network did no show any hints of learning behaviour. I suspect that trying to predict the the intrinsic and extrinsic parameters all at once (which is the approach I took) proves to be too difficult as traditionally an iterative optimisation is taken over the two sets of parameters.

4.7 CornerNet

4.7.1 training setup: batch size, learning rates and momentum, dropout rate

Here, I trained the ResNet50 model to predict the coordinates of the 4 corners of the chessboard. Similarly to ParamNet, I used the ResnNet as a backbone to extract features and passed it through fully connected layers alternating with ReLu's and halving its feature dimension. At the end, the network split into 8 fully connected layer heads each predicting one of the x or y coordinates of the 4 corners. The model was trained for 14 epochs with batch size of 50 images, learning rate 1e-3 and SGD optimiser. The loss was a simple norm function between the 8-dimensional predicted and ground truth vectors. Thus the loss is actually the sum of the norms across the 4 corners we're regressing on.

4.7.2 Performance

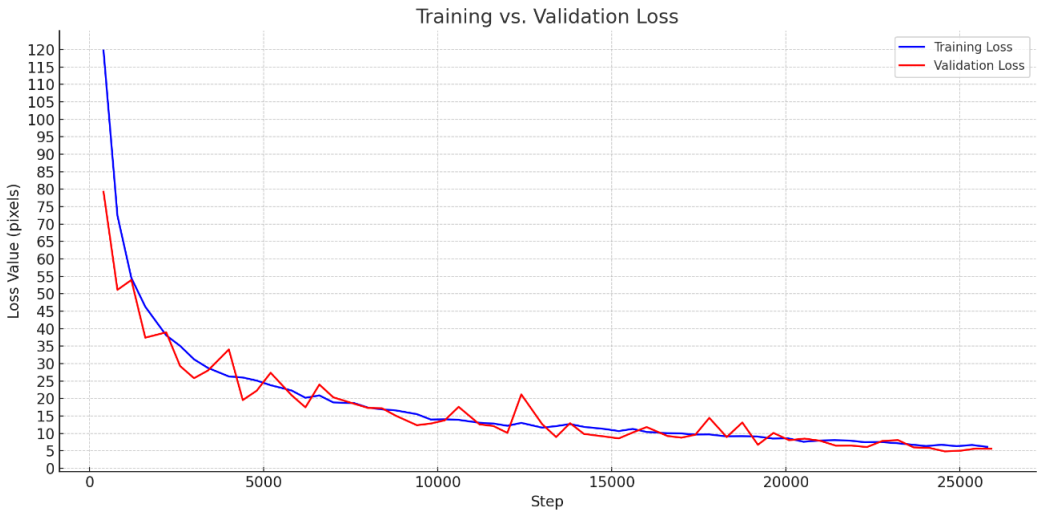


Figure 4.27: Training and validation loss for CornerNet. This is over the 4 corners thus when the loss reaches around 5 at the end of training, the per corner error is actually close to 5/4.

Looking at the loss we can see the network performs quite well in predicting the corner coordinates with average pixel error close to 1.

4.7.3 Evaluation on rendered test set with varying noise/blurring

Again here we apply the levels of blurring and jitter defined in the previous sections.

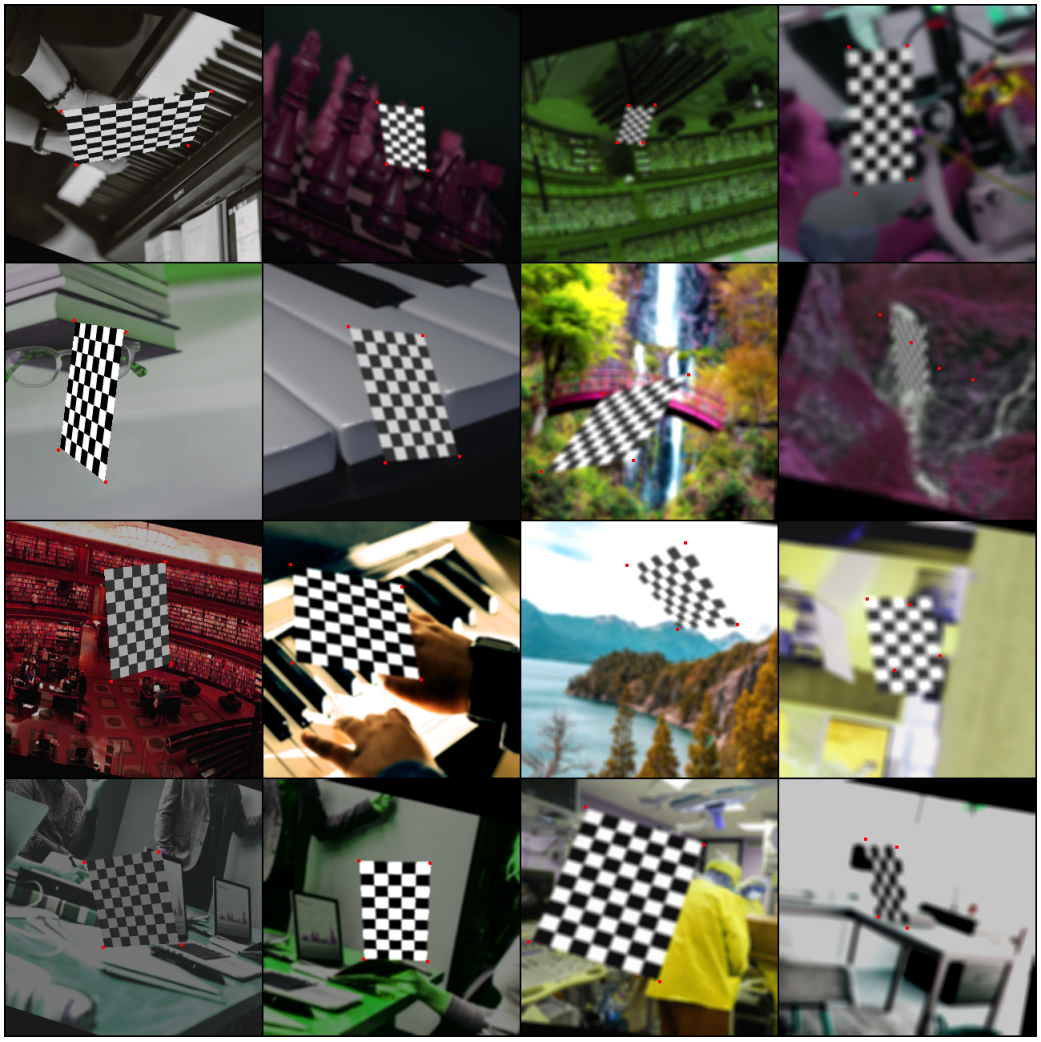


Figure 4.28: Evaluation rendered test set. Red pixels indicate where the model detects the corners. Blur increasing left to right, jitter increasing top to bottom.

4.7.4 Evaluation on real test set with varying noise/blurring

Figure 4.29: Evaluation on real images. Red pixels indicate where the model detects the corners. Blur increasing left to right, jitter increasing top to bottom.

4.7.5 Discussion

As we can see, the corner dregression network performs quite well on the rendered images and generalises to the real images as well. Blur seems to have a much higher effect on the corner detection than jitter. Very extreme views result in poor corner detections.

4.8 OpenCV + CornerNet

In this section we once again investigate the performance of OpenCV however this time using our CornerNet corner detector instead of the native one OpenCV comes with.

4.8.1 Performance

Metric	Mean	Standard Deviation
OpenCV Predicted Error (pixels)	0.672089	0.566346
2D Projection Error (pixels)	2.302642	2.370247
3D Reprojection Error (3D space units)	59.263816	91.025435
MAPE (%)	89.596683	110.011528

From the Table we can already see a massive increase in performance. All the mean errors (except for OpenCV predicted error) are much smaller and have smaller standard deviations suggesting they're more stable than the pure OpenCV calibration. Furthermore, the OpenCV predicted error is much closer to the true error suggesting that introducing our CornerNet has made the calibration more aware of the pitfalls it was falling for previously.

4.8.2 Number of images

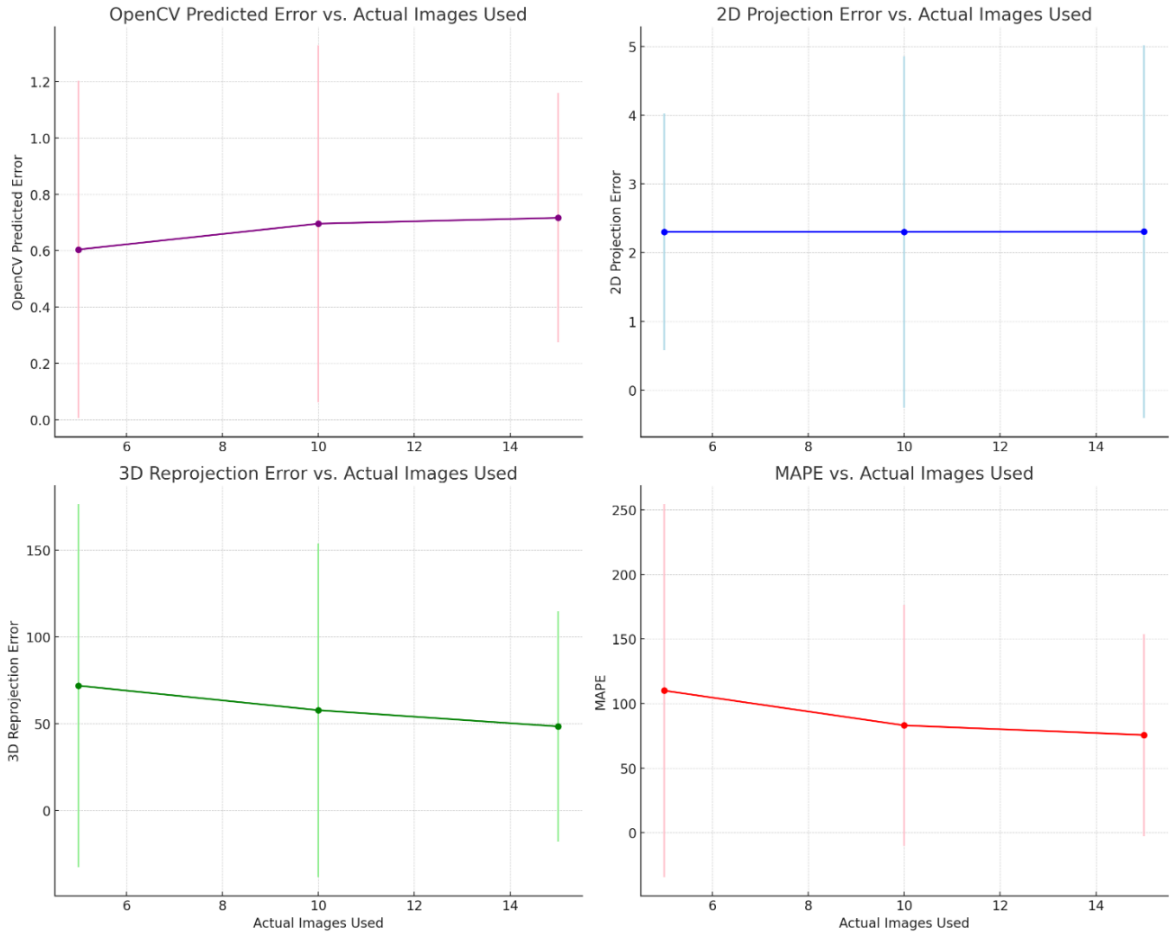


Figure 4.30: Mean and standard deviation of errors. As we can see these vary much less now showing stability from adding the deep learning corner detector. Furthermore, the mean does not change much with the number of images indicating that the corner detector does not struggle on specific images or otherwise it would have been more susceptible to noise. Lastly, no values close to 100 for the projection error as was the case with pure OpenCV calibration suggesting the corner detector doesn't find misleading point correspondences like the standard OpenCV corner detector

.

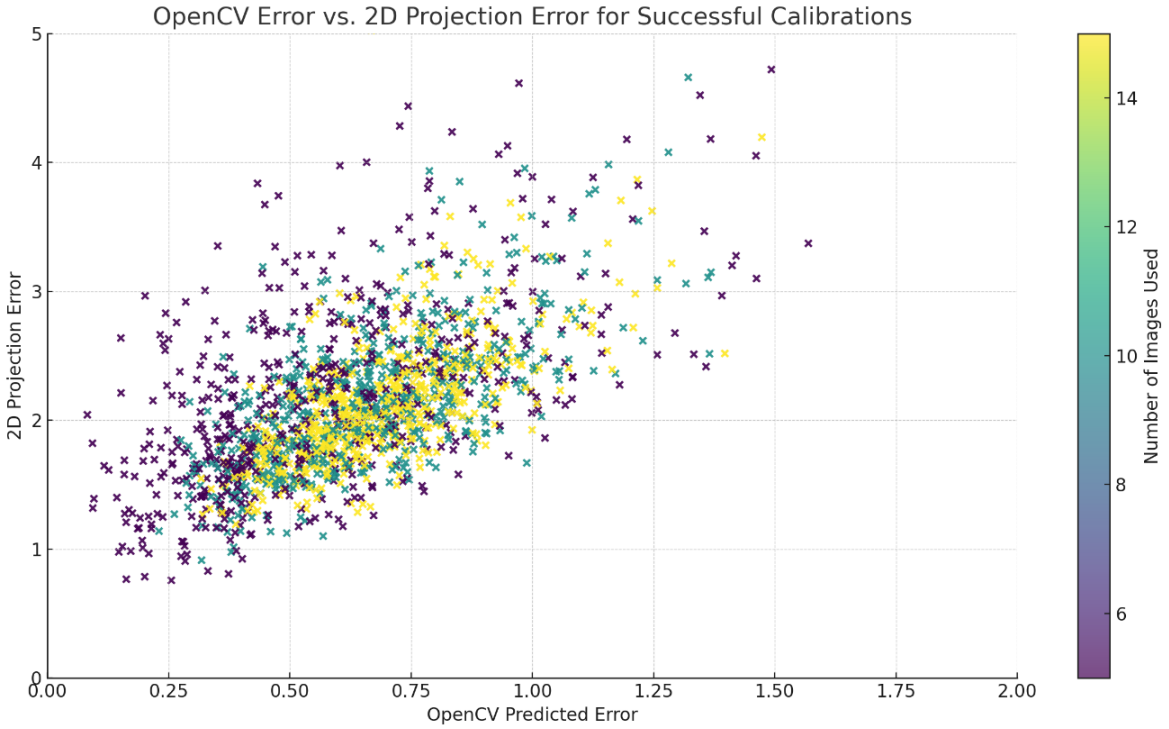


Figure 4.31: OpenCV predicted errors vs true 2D projection errors. The predicted errors are much closer to the true nesnand seem to fit a linear trend. Using lower number of images tends to give more variation in the error while using the higher amounts of images for calibration tends to give lower errors

.

4.8.3 Evaluation on rendered test set with varying noise/blurring

Metric	Mean	Standard Deviation
OpenCV Predicted Error	1.23	1.44
2D Projection Error	8.39	52.47
3D Reprojection Error	462.59	36756.60
MAPE	613.40	46882.13

4.8.4 Evaluation on real test set with artificial varying noise/blurring

Statistic	OpenCV Predicted Error
Mean	5.507280
Standard Deviation	3.605718

4.8.5 Discussion

As we can see, there is a significant discrepancy between the OpenCV predicted error and true 2D projection error on the rendered test set. The standard deviations suggest that the noising process introduces great variability in the prospects of success of calibration. Comparing just the openCV predicted errors, the performance degrades from rendered to real images.

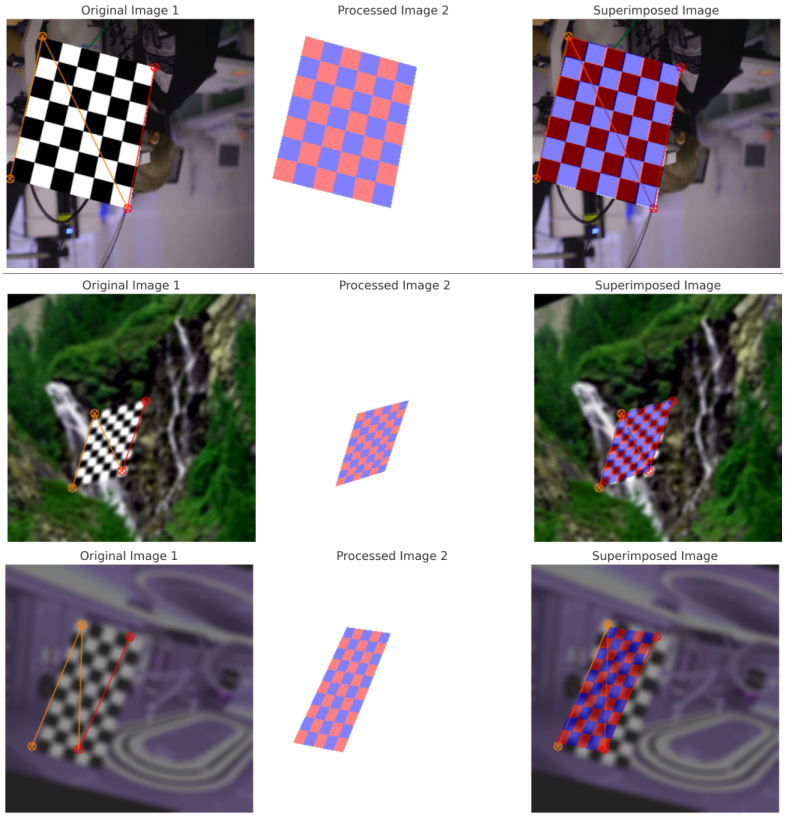


Figure 4.32: CornerNet detection on synthetic images for increasing levels of blur and jitter with superposition of rendering of predicted paramaters of OpenCV + CornerNet.

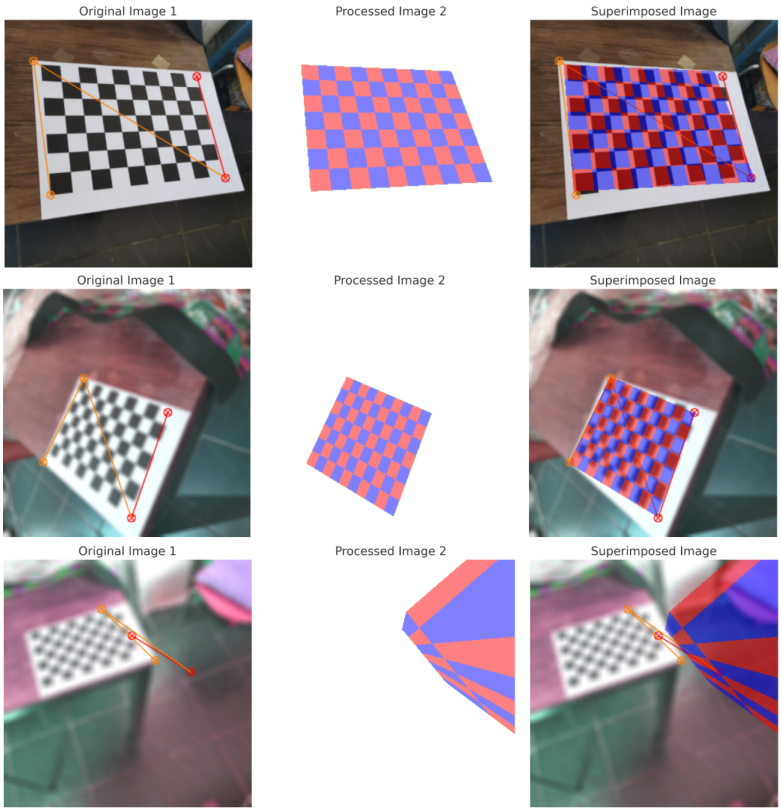


Figure 4.33: CornerNet detection on real images for increasing levels of blur and jitter with superposition of rendering of predicted paramaters of OpenCV + CornerNet.

Chapter 5

Conclusion and Future Work

the objective of this project was to investigate whether deep elarning techniques can be applied to simplify the time and labour consuming camera calibration procedure. The robustness and accuracy of traditional methods such as OpenCV were studied and found to be susceptible to noise and easily perturbed into wrong solutions while being totally unaware. A 3D simulation system was developed to generate synthetic data for a variety of tasks involving chessboards (including segmentation and regression). The simulation system is a powerful tool as it is able to generate virtually unlimited amounts of labelled data for different tasks at hand (segmentation of different parts of the chessboard, regressing corner coordinates or generating specific chessboard views at given intrinsics). This makes it not only a useful tool in the computer vision community but the use of simulation models to generate synthetic data is also a formidable companion to supervised learning in general. Deep learning approaches to aide in camera calibration were explored along with different intuitive notions of loss measure that can be used for training a neural network to regress the intrinsic and extrinsic parameters of a camera. Though I was unable to find a fully deep learning based method to find the camera parameters, I investigated the robustness and accuracy of traditional methods on the synthetically generated dataset and combined traditional camera calibration with deep learning to give a more robust solution. A direct estimation of the intrinsic and extrinsic parameters proved to be a challenging task due the usually iterative optimisation approach required for the two sets of parameters. The calibration process was mainly investigated for single image inputs, however due to ambiguity of different image views corresponding to multiple potential parameters, the problem is rendered extremely challenging. A natural continuation of the project would be to try and extend to multiple inputs at once. Furthermore, such

a method could benefit from something such as self-attention layers which could provide weights to utilise more informative chessboard images better. Furthermore, if given a live video feed of a chessboard moving about, there are temporal and dependencies inbetween frames that could be captured using something like an RNN/LSTM. While, only the 4 main corners of the chessboard were regressed during the project, we could extend the regression tasks to include all the inlier corners of the chessboard which wuld then directly feed into a traditional calibration pipeline. However, an easier approach might be to utilise the UNet generated inlier corner segmentation predictions, take the centroids of each blob/dot representing an inlier corner and use the 4 regressed main corner coordinates to order them (as the blobs are unordered). Such an ordering could be achieved using something like a homography mapping the 4 regressed corner coordinates to 4 corners of a straight view / upright grid and using the same homography to map the blob centroid coordinates onto the grid structure for ordering. Lastly, while the 3D model was highly essential throughout the project and served its purpose well. However, the former could also be improved and made more realistic by modelling more complex textures and lighting effects (glossy reflective surfaces for example) to better model the real world chessboard images and decrease distribution shift. One more improvement to the synthetic data generation process would be to generate images with different aspect ratios and resolutions unlike during the project were only square aspect ratios at a resolution of 320 by 320 pixels were considered.

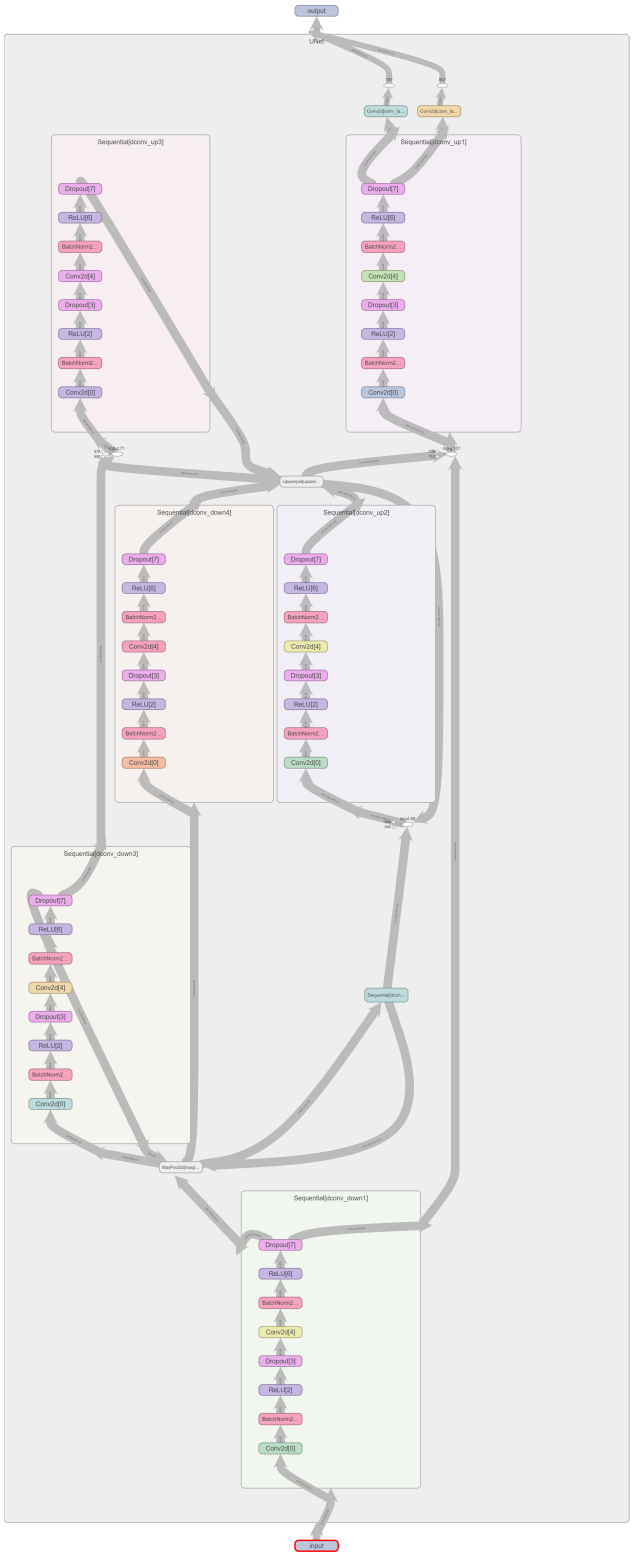


Figure 1: Unet based architecture in more detail

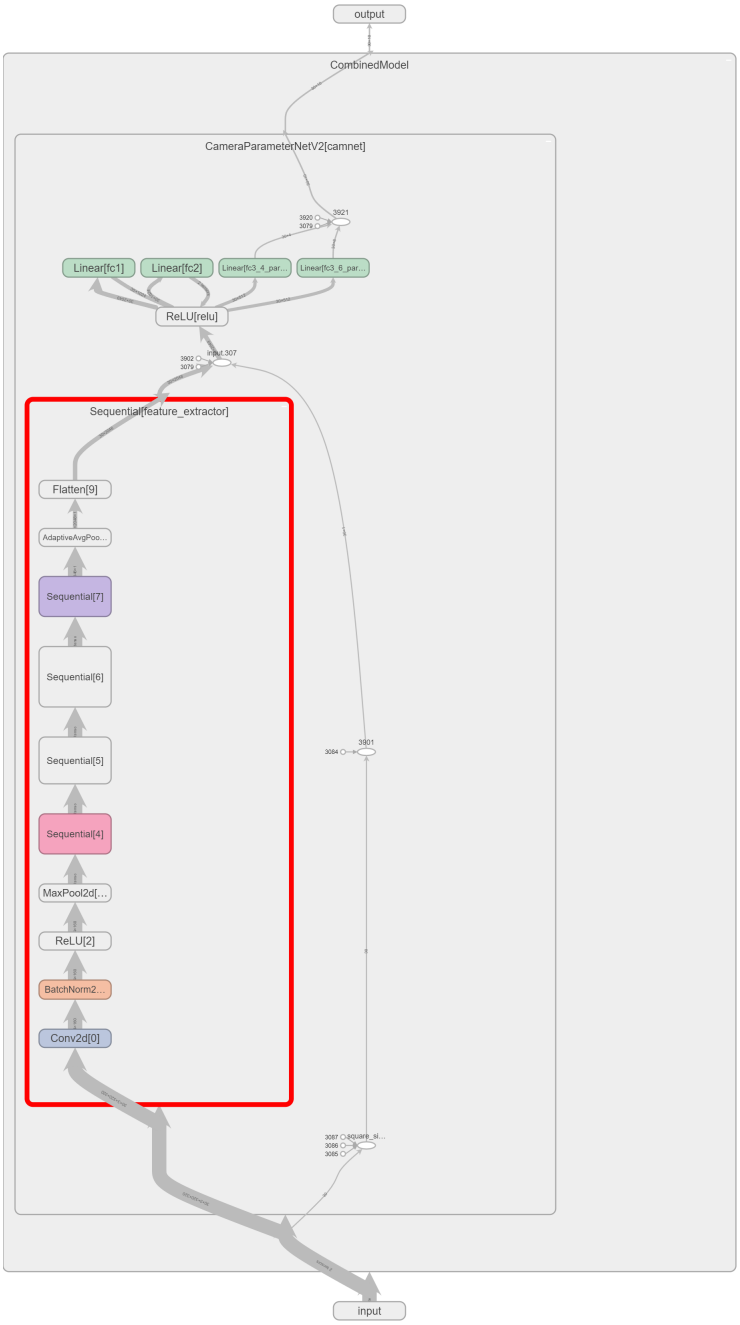


Figure 2: ResNet50 based architecture diagram.

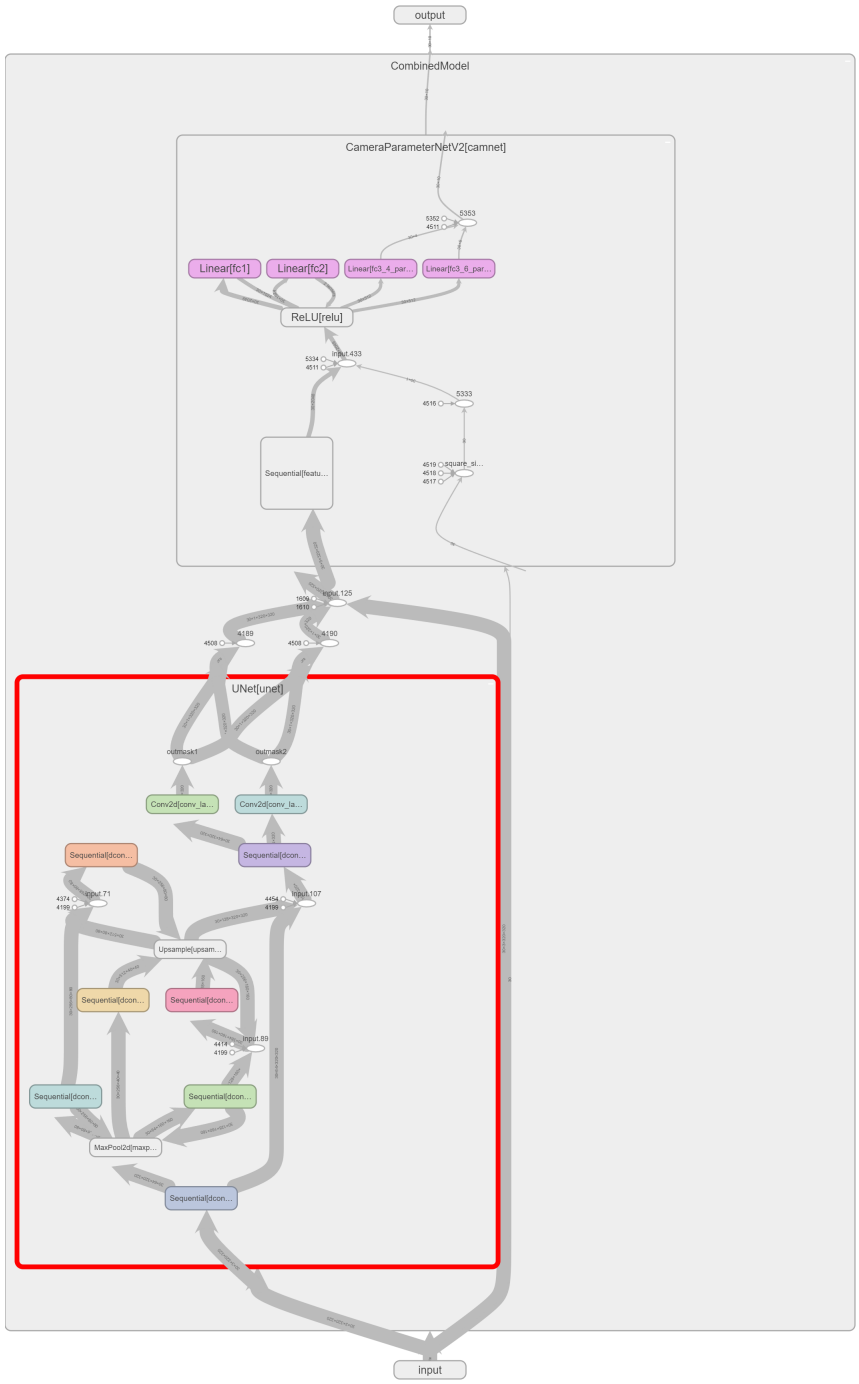


Figure 3: Combined UNet and ResNet.

Bibliography

[1] What is rasterize? Online, https://www.computerhope.com/jargon/r/rasterize.htm

[2] What is rasterize? how does rasterization impact file sizes? — lenovo us. Online, https://www.lenovo.com/us/en/glossary/rasterize/?orgRef=https%253A%252F%252Fwww.google.com%252F

[3] (Aug 2023), https://en.wikipedia.org/wiki/Pinhole_camera_model#/media/File:Pinhole-camera.svg

[4] adm.ebu.io: Coordinate system - ebu adm guidelines. Online (nd), https://adm.ebu.io/reference/excursions/coordinate_system.html, accessed 12 Sep. 2023

[5] Aqeel, S.: Deep learning for predictive analytics in healthcare (03 2019). https://doi.org/10.1007/978-3-030-14118-9

[6] Bhatt, D.: A comprehensive guide for camera calibration in computer vision. Analytics Vidhya, říjen (2021)

[7] Bishop, C.M.: Neural networks for pattern recognition. Oxford university press (1995)

[8] Blake, A.: Computer Vision. Cambridge University Press (2021), https://www.cambridge.org/core/books/vision/computer-vision/D736324647A3753650BCA7252B168344, [Accessed 13 Sep. 2023]

[9] Bottou, L.: Large-scale machine learning with stochastic gradient descent. In: Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers. pp. 177–186. Springer (2010)

[10] Bouguet, J.: Camera calibration toolbox for matlab. computational vision at the california institute of technology. Computational Vision at the California Institute of Technology (2012)

[11] Brown, D.: Decentering distortion of lenses (1966), https://api.semanticscholar.org/CorpusID:117271607

[12] Chen, X., Xi, J., Jin, Y., Sun, J.: Accurate calibration for a camera–projector measurement system based on structured light projection. Optics and Lasers in Engineering **47**(3-4), 310–319 (2009)

[13] Chung, J., Gulcehre, C., Cho, K., Bengio, Y.: Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555 (2014)

[14] Ciaburro, G., Venkateswaran, B.: Neural Networks with R: Smart models using CNN, RNN, deep learning, and artificial intelligence principles. Packt Publishing Ltd (2017)

[15] Courbon, J., Mezouar, Y., Eckt, L., Martinet, P.: A generic fisheye camera model for robotic applications. In: 2007 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1683–1688. IEEE (2007)

[16] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[17] Dumoulin, V., Visin, F.: A guide to convolution arithmetic for deep learning. arXiv preprint arXiv:1603.07285 (2016)

[18] Elman, J.L.: Finding structure in time. Cognitive science **14**(2), 179–211 (1990)

[19] examples.itk.org: Cartesian to azimuth elevation — v5.4.0. Online (nd), https://examples.itk.org/src/core/transform/cartesiantoazimuthelevation/documentation#, accessed 12 Sep. 2023

[20] Faugeras, O.: Three-dimensional computer vision: a geometric viewpoint. MIT press (1993)

[21] Fraser, C.: On the use of non-metric cameras in analytical non-metric photogrammetry. International Archives of Photogrammetry and Remote Sensing **24**(5), 156–166 (1982)

[22] GeeksforGeeks: What is anti-aliasing? Online (2019), https://www.geeksforgeeks.org/antialiasing/, accessed 16 Sep. 2023

[23] GeeksforGeeks: Gouraud shading in computer graphics. Online (2022), https://www.geeksforgeeks.org/gouraud-shading-in-computer-graphics/, accessed 12 Sep. 2023

[24] GeeksforGeeks: Phong shading computer graphics. Online (2023), https://www.geeksforgeeks.org/phong-shading-computer-graphics/, accessed 12 Sep. 2023

[25] GitHub: in-plane camera rotation parameter for look$_a t_v iew_t ransformissue927facebookresearch/pytorch3$ accessed 15 Sep. 2023

Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. MIT press (2016)

Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: 2013 IEEE international conference on acoustics, speech and signal processing. pp. 6645–6649. Ieee (2013)

H., L.: Dice coefficient! what is it? Medium (2023), https://medium.com/@lathashreeh/dice-coefficient-what-is-it-ff090ec97bda, [Online]

Hartley, R., Kang, S.B.: Parameter-free radial distortion correction with center of distortion estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence **29**(8), 1309–1321 (2007)

Hartley, R., Zisserman, A.: Multiple view geometry in computer vision. Cambridge university press (2003)

Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision. Cambridge University Press, New York (2017)

Hata, K., Savarese, S.: Cs231a course notes 1: Camera models (nd), https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf, accessed 1 May 2020

Haykin, S.: Neural networks: a comprehensive foundation. Prentice Hall PTR (1998)

He, B., Li, Y.: A novel method for camera calibration using vanishing points. In: 2007 14th International Conference on Mechatronics and Machine Vision in Practice. pp. 44–47. IEEE (2007)

He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)

Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: Neural networks for perception, pp. 65–93. Elsevier (1992)

Hold-Geoffroy, Y., Sunkavalli, K., Eisenmann, J., Fisher, M., Gambaretto, E., Hadap, S., Lalonde, J.F.: A perceptual measure for deep single image camera calibration. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2354–2363 (2018)

Islam, A., Hossain, M.A., Jang, Y.M.: Interference mitigation technique for time-of-flight (tof) camera. In: 2016 eighth international conference on ubiquitous and future networks (ICUFN). pp. 134–139. IEEE (2016)

Jadon, S.: A survey of loss functions for semantic segmentation. In: 2020 IEEE conference on computational intelligence in bioinformatics and computational biology (CIBCB). pp. 1–7. IEEE (2020)

Julia, L.F., Monasse, P., Pierrot-Deseilligny, M.: The orthographic projection model for pose calibration of long focal images. Image Processing On Line **9**, 231–250 (2019)

Kannala, J., Brandt, S.S.: A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. IEEE transactions on pattern analysis and machine intelligence **28**(8), 1335–1340 (2006)

Kannala, J., Heikkilä, J., Brandt, S.S.: Geometric camera calibration. Wiley encyclopedia of computer science and engineering **13**(6), 1–20 (2008)

Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization (2014). arXiv preprint arXiv:1412.6980 **15** (2017)

Konda, K.R., Bouthillier, X., Memisevic, R., Vincent, P.: Dropout as data augmentation. ArXiv **abs/1506.08700** (2015), `https://api.semanticscholar.org/CorpusID:6703663`

Kopparapu, S., Corke, P.: The effect of noise on camera calibration parameters. Graphical Models **63**(5), 277–303 (2001)

Krishna, N.: Image formation and pinhole model of the camera. Medium (2022), [Accessed on 9 September 2023]

Kumar, R.K., Ilie, A., Frahm, J.M., Pollefeys, M.: Simple calibration of non-overlapping cameras with a mirror. In: 2008 IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–7. IEEE (2008)

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE **86**(11), 2278–2324 (1998)

Li, B., Heng, L., Koser, K., Pollefeys, M.: A multiple-camera system calibration toolbox using a feature descriptor-based calibration pattern. In: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 1301–1307. IEEE (2013)

MathWorks: What is camera calibration? - MATLAB & Simulink - MathWorks United Kingdom (nd), https://uk.mathworks.com/help/vision/ug/camera-calibration.html, accessed 21 Feb. 2022

MATLAB: Introduction to deep learning: What are convolutional neural networks? video (Mar 2017), https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convol
html

Minsky, M., Papert, S.: An introduction to computational geometry. Cambridge tiass., HIT **479**(480), 104 (1969)

Nwankpa, C., Ijomah, W., Gachagan, A., Marshall, S.: Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378 (2018)

OpenCV: Camera calibration, https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.
html

OpenCV: Opencv: Camera calibration. Available at: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html (nd)

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems **32** (2019)

Pollefeys, M., Van Gool, L., Vergauwen, M., Verbiest, F., Cornelis, K., Tops, J., Koch, R.: Visual modeling with a hand-held camera. International Journal of Computer Vision **59**, 207–232 (2004)

Prince, S.J.D.: Computer Vision: Models, Learning, and Inference. Cambridge University Press (2012). 10.1017/CBO9780511996504

Pulli, K., Baksheev, A., Kornyakov, K., Eruhimov, V.: Real-time computer vision with opencv. Communications of the ACM **55**(6), 61–69 (2012)

pytorch3d.readthedocs.io: rasterizer — pytorch3d 0.2.0 documentation. Online (nd), https://pytorch3d.readthedocs.io/en/v0.6.0/modules/renderer/rasterizer.html#:~:text=Rasterize%20a%20batch%20of%20meshes, accessed 10 Sep. 2023

Ramalingam, S.: Generic imaging models: Calibration and 3d reconstruction algorithms. Ph.D. thesis, Institute National Polytechnique de Grenoble-INPG (2006)

Ravi, N., Reizenstein, J., Novotny, D., Gordon, T., Lo, W.Y., Johnson, J., Gkioxari, G.: Accelerating 3d deep learning with pytorch3d. arXiv:2007.08501 (2020)

Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18. pp. 234–241. Springer (2015)

Rosenblatt, F.: The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review **65**(6), 386 (1958)

Rosenfeld, A.: Computer vision: basic principles. Proceedings of the IEEE **76**(8), 863–868 (1988)

Ruder, S.: An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747 (2016)

Schneider, D., Schwalbe, E., Maas, H.G.: Validation of geometric models for fisheye lenses. ISPRS Journal of Photogrammetry and Remote Sensing **64**(3), 259–266 (2009)

ScienceDirect Topics: Phong shading - an overview — sciencedirect topics. Online (nd), `https://www.sciencedirect.com/topics/computer-science/phong-shading`, accessed 12 Sep. 2023

scratchapixel.com: Placing a camera: the lookat function. Online (nd), `https://www.scratchapixel.com/lessons/mathematics-physics-for-computer-graphics/lookat-function/framing-lookat-function.html`, accessed 17 Sep. 2023

Sharma, S., Sharma, S., Athaiya, A.: Activation functions in neural networks. Towards Data Sci **6**(12), 310–316 (2017)

sharmaniti437: Introduction to pytorch3d. Online (2021), `https://timebusinessnews.com/introduction-to-pytorch3d/`, accessed 17 Sep. 2023

Snavely, N., Seitz, S.M., Szeliski, R.: Photo tourism: exploring photo collections in 3d. In: ACM siggraph 2006 papers, pp. 835–846 (2006)

Song, J., Zhao, Y., Song, W., Zhou, H., Zhu, D., Huang, Q., Fan, Y., Lu, C.: Fisheye image detection of trees using improved yolox for tree height estimation. Sensors **22**(10), 3636 (2022)

Srivastava, N., Hinton, G.E., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. J. Mach. Learn. Res. **15**, 1929–1958 (2014), https://api.semanticscholar.org/CorpusID:6844431

Szeliski, R.: Computer Vision. Texts in Computer Science, Springer London (2011). 10.1007/978-1-84882-935-0

Szeliski, R.: Computer vision: algorithms and applications. Springer Nature (2022)

Tang, M., Huang, Z., Yuan, Y., Wang, C., Peng, Y.: A bounded scheduling method for adaptive gradient methods. Applied Sciences **9**(17), 3569 (2019)

Targ, S., Almeida, D., Lyman, K.: Resnet in resnet: Generalizing residual architectures (Mar 2016), https://doi.org/10.48550/arXiv.1603.08029

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)

Wakai, N., Sato, S., Ishii, Y., Yamashita, T.: Rethinking generic camera models for deep single image camera calibration to recover rotation and fisheye distortion. In: European Conference on Computer Vision. pp. 679–698. Springer (2022)

Walters, D.: Computer Vision, p. 431â€“435. John Wiley and Sons Ltd., GBR (2003)

Wang, J.m., Tsai, C.T., Cherng, S., Chen, S.W.: Omni-directional camera networks and data fusion for vehicle tracking in an indoor parking lot. In: 2006 IEEE International Conference on Video and Signal Based Surveillance. pp. 45–45. IEEE (2006)

Wiley, V., Lucas, T.: Computer vision and image processing: a paper review. International Journal of Artificial Intelligence Research **2**(1), 29–36 (2018)

Willson, R.G., Shafer, S.A.: Perspective projection camera model for zoom lenses. In: Optical 3D Measurement Techniques II: Applications in Inspection, Quality Control, and Robotics. vol. 2252, pp. 149–158. SPIE (1994)

Wilson, A.C., Roelofs, R., Stern, M., Srebro, N., Recht, B.: The marginal value of adaptive gradient methods in machine learning. Advances in neural information processing systems **30** (2017)

Workman, S., Greenwell, C., Zhai, M., Baltenberger, R., Jacobs, N.: Deepfocal: A method for direct focal length estimation. In: 2015 IEEE International Conference on Image Processing (ICIP). pp. 1369–1373. IEEE (2015)

Yan, K., Liu, E., Tian, H., Zhao, R., Zhang, Z.: Calibration of camera intrinsic parameters using a single image. In: Proceedings of the 3rd International Conference on Robotics and Artificial Intelligence. pp. 39–45 (2017)

Yasrab, R., Pound, M.P., French, A.P., Pridmore, T.P.: Phenomnet: Bridging phenotype-genotype gap: A cnn-lstm based automatic plant root anatomization system. bioRxiv (2020), https://api.semanticscholar.org/CorpusID:218539212

Yu, W., Xie, J., Wu, X., Liu, K.: Research of improved zhang's calibration method. In: 2017 Chinese Automation Congress (CAC). pp. 1423–1427. IEEE (2017)

Zhang, C., Rameau, F., Kim, J., Argaw, D.M., Bazin, J.C., Kweon, I.S.: Deepptz: Deep self-calibration for ptz cameras. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision. pp. 1041–1049 (2020)

Zhang, Y.J.: Camera calibration. In: 3-D Computer Vision: Principles, Algorithms and Applications, pp. 37–65. Springer (2023)

Zhang, Z.: A flexible new technique for camera calibration. IEEE Transactions on pattern analysis and machine intelligence **22**(11), 1330–1334 (2000)

Ziemann, H.: Thoughts on a standard algorithm for camera calibration. Progress in Imaging Sensors pp. 41–48 (1986)

Ziemann, H., El-Hakim, S.F.: On the definition of lens distortion reference data with odd-power polynomials. The Canadian Surveyor **37**(3), 135–143 (1983)

Appendix A

Other appendices, e.g. code listing

The code is available in the following GitHub repository link: `https://github.com/rrahmn/MSc_project_code`