Distortion Parameter Estimation for Fisheye Image Rectification Using Deep Neural Networks

Karit Sookpreedee

Supervised by

Dr Alexander Krull

MSc Artificial Intelligence and Machine Learning

School of Computer Science

College of Engineering and Physical Sciences

University of Birmingham

2023-24

Abstract

The widespread use of fisheye cameras in areas like autonomous driving and surveillance has highlighted the need for effective image rectification methods to address the significant distortions caused by these lenses. While deep learning has shown promise in improving rectification accuracy, many existing studies lack clarity, failing to provide model weights, codes, and detailed rectification processes, which complicates replication. This project uses a deep neural network to predict distortion parameters and the rectification program to rectify both real and synthesised fisheye images. The framework consists of three key components: dataset synthesis, model training and validation, and image rectification and evaluation. Two deep learning architectures (AlexNet and Vision Transformer) and three training schemes were explored. AlexNet delivered the best performance, particularly under Classification and Classification + Regression schemes. Moreover, the rectification program effectively corrected real fisheye images, surpassing existing methods in both image quality and quantitative metrics. This work addresses research gaps by providing comprehensive model and rectification program details and a replicable framework, laying the groundwork for future advancements in fisheye image rectification.

Keywords: Fisheye image rectification, Distortion parameter prediction, Division model, Deep neural network

Acknowledgement

During my master's degree program, I had the pleasure of meeting various individuals, including friends, professors, and faculty members. As the program drew close, I realised I had undergone significant personal growth since my initial arrival in the UK. My time spent living and studying in the UK has left me with countless invaluable and unforgettable memories. The knowledge I acquired throughout the program represents the most profound educational experience I have ever encountered. I have had the opportunity to apply and demonstrate the advanced mathematical skills I have cultivated over the years in every module of this course, particularly in my dissertation. Being a master's student in the Artificial Intelligence and Machine Learning program at the School of Computer Science of the University of Birmingham is an honour and a defining milestone in my life.

I would like to express my gratitude to Dr Alexander Krull, my supervisor, for his visionary and imaginative guidance throughout my project. His wealth of experience and expertise in computer vision provided invaluable feedback and innovative ideas, enabling me to refine the project and overcome any challenges that arose.

I would also like to convey my thanks to Dr Miqing Li for the direction and assistance given during the project presentation. His approachable demeanour and constructive feedback greatly enhanced the quality and delivery of the presentation. His valuable input and encouragement significantly contributed to improving the final report.

To all my friends and colleagues, whether in Thailand or the UK, you have been my guiding stars during my darkest moments and brought colour to my life when everything seemed blue. Your laughter, support, and advice have left an indelible mark on me, and I will always cherish these memories. I am also immensely grateful to my family in Thailand, who made it possible for me to pursue a new life in the UK and study at this university. Their unwavering belief in my potential has been the bedrock of my journey from its beginning to the present moment.

To every fragment, every experience, and every soul that has woven itself into the fabric of who I am today, I extend my deepest and most profound gratitude. You are the essence of my being, and for each of you, I am eternally thankful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer vision and deep learning advancements have greatly improved image processing techniques, particularly in rectifying distorted images from fisheye lenses. The use of fisheye cameras in applications such as autonomous driving and surveillance has made it increasingly necessary to address the substantial distortions introduced by these lenses. Despite their wide field of view, fisheye lenses can enormously impact the quality and accuracy of image-based systems. [1], [2]

The traditional methods for correcting fisheye distortions often rely on geometry-based approaches that use specific points, lines, or shapes within the image to estimate and rectify distortions. However, these methods may be limited by the complexity of the scenes, noise, and the need for manual intervention. In contrast, deep learning methods improve the rectification process by leveraging the capability of neural networks to learn from large volumes of data. Two main approaches to addressing distortion correction are direct rectified image generation and distortion parameter prediction. The former is an end-to-end network that can directly produce the rectified images. Meanwhile, the latter predicts the amount of distortion in the input image based on a distortion model and then uses a function to create the rectified image.

Despite the limitations of both deep learning approaches, the decision was made to follow the prediction of distortion parameters due to time and resource constraints associated with the direct generation of rectified images, which are unavoidable. A common issue observed in many papers that predict distortion parameters is the lack of published model weights or codes, making replication difficult. Additionally, these papers also lack details about their image rectification program. Addressing these research gaps will be the primary focus of this project.

1.1 Project objectives

This project aims to enhance the methodology for utilising deep neural networks to predict distortion parameters and to develop a rectification program for real and synthesised fisheye

images. The project will provide all the best model weights and detailed mathematical explanations for the rectification program, along with the program itself, to enable replication. The project framework consists of three main parts: 1) Dataset synthesis, 2) Model training and validation, and 3) Image rectification and evaluation, involving the image distortion and rectification program, two deep learning architectures, and three model training schemes. The main contributions of this project can be outlined as follows:

1. Developing a program to generate distorted and rectified images using the division model and its inverse, accompanied by detailed mathematical explanations.
2. Conducting four experiments comprising three tasks (Regression, Classification, Classification + Regression) and two deep learning architectures (AlexNet, Vision Transformer), concluding that both AlexNet (Classification) and AlexNet (Classification + Regression) demonstrate similarly high performance.
3. Testing the most effective models on an unseen-content testing set to validate that performance remains strong.
4. Evaluating the outcomes using Average Relative Error, Average SSIM, and Average PSNR.
5. Inferring the best model to rectify real fisheye images.
6. Comparing the obtained results with those presented in the paper [3].

1.2 Thesis structure

The report comprises six chapters. The first chapter offers an overview of the problem, existing solutions, and their limitations, followed by the project objectives and deliverables. Chapter 2 will provide the necessary background information on the project, including the mathematical model for image distortion, deep learning concepts and architectures, dataset, related works, and their research gaps. Chapter 3 will detail all components of the project framework, including the mathematical explanation for the image distortion and rectification program. Chapter 4 will present the main results and provide analysis, while Chapter 5 will compare the results with other papers and discuss the future directions and challenges encountered during the project. The report will finally conclude the project in Chapter 6.

Chapter 2

Background

2.1 Technical background

2.1.1 Mathematical models for image distortion

In Figure 2.1, the effect of distortion (black grid) on the corrected image (grey grid) is shown. Suppose (x_c, y_c) represents the coordinate on the corrected image, and (x_d, y_d) is the corresponding coordinate on the distorted image. Their relationship can be expressed using Equation 2.1, known as the Division model [4]. The distortion parameter k quantifies the degree of image distortion, and (x_m, y_m) denotes the centre of the radial distortion.

$$\begin{bmatrix} x_c - x_m \\ y_c - y_m \end{bmatrix} = \frac{1}{1 + k\left[(x_d - x_m)^2 + (y_d - y_m)^2\right]} \begin{bmatrix} x_d - x_m \\ y_d - y_m \end{bmatrix} \tag{2.1}$$

2.1.2 Deep learning and Neural network

Deep learning (DL) is a fundamental concept in the field of artificial intelligence and machine learning. It is distinguished by its capacity to learn from extensive data through layered architectures. Neural networks are the fundamental architecture for DL, serving as computational models inspired by the human brain, comprised of interconnected nodes or neurons (Figure 2.2). Each neuron receives input, processes it using an activation function, and transmits the output to subsequent layers. A neuron can be written in a mathematical equation as

$$y = f(z) = f\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{2.2}$$

where x_i are the input values, w_i are the corresponding weights, and b is the bias. The activation function f introduces a non-linear relationship between the input and output, allowing the

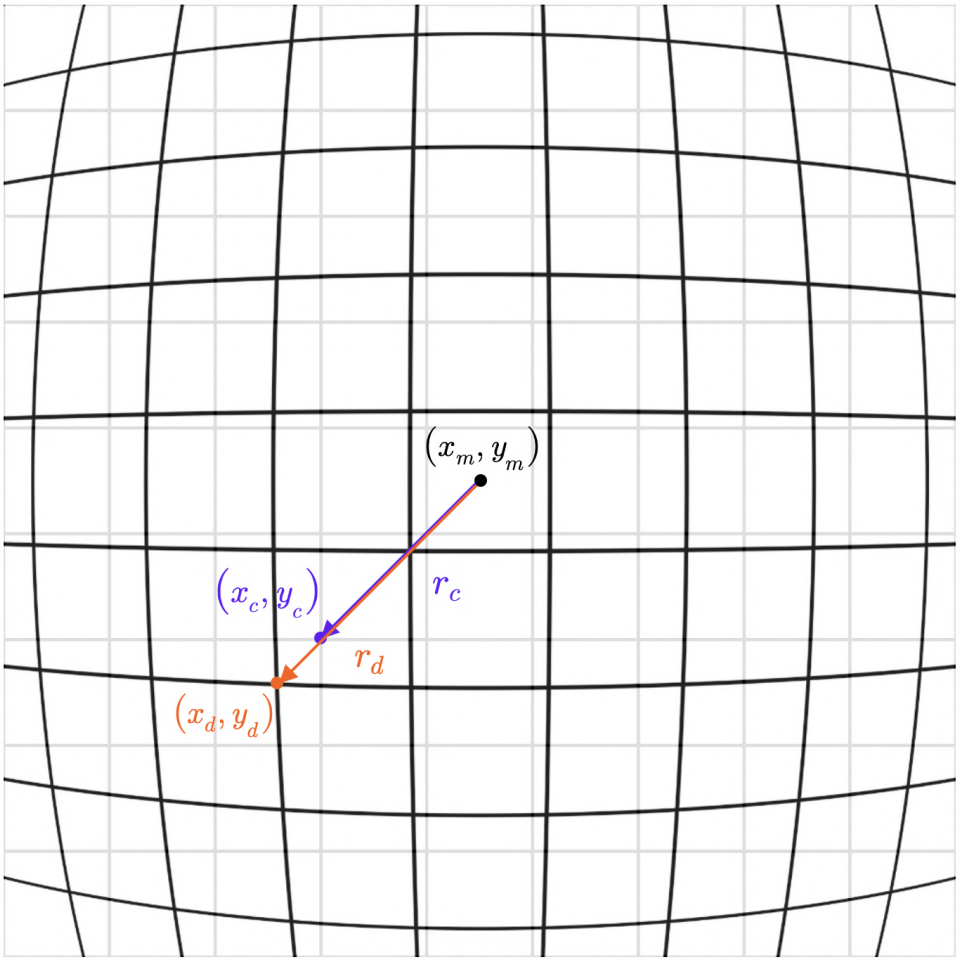

Figure 2.1: The effect of distortion (black grid) on the corrected image (grey grid);
(x_m, y_m) = The centre coordinate of the distortion,
(x_c, y_c) = A coordinate in the corrected image plane,
(x_d, y_d) = The corresponding coordinate in the distorted image plane,
r_c = The L2 distance between (x_c, y_c) and (x_m, y_m),
and r_d = The L2 distance between (x_d, y_d) and (x_m, y_m)

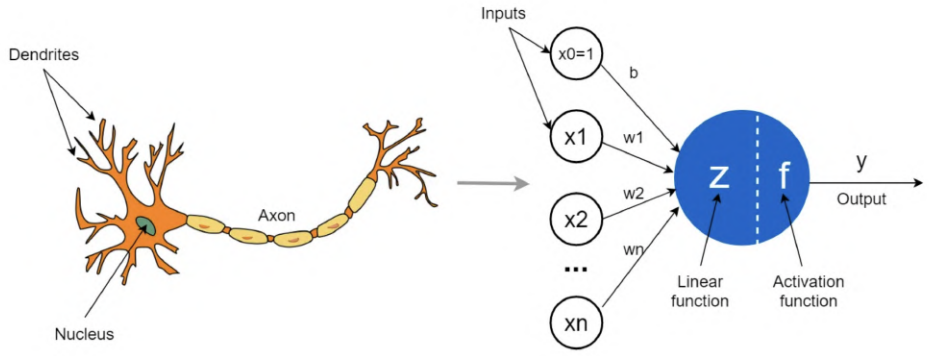

Figure 2.2: An infographic comparing the human brain's neurons and the mathematical model of neurons in DL, where Dendrites receive an input signal x_i, Nucleus are similar to $f(z)$, and the output signal y is transmitted through the Axon. [5]

network to learn and represent complex patterns. Without non-linearity, a neural network would behave like a linear regression model. Common activation functions include ReLU (Rectified Linear Unit), Sigmoid function, Softmax function, and others. Additionally, these functions trim the output range; for example, the sigmoid function compresses the output z from the range $(-\infty, \infty)$ to the range $(0, 1)$, making it useful for classification problems where the output is interpreted as a probability. Lastly, the activation function is differentiable, which is crucial for the backpropagation algorithm during the training phase.

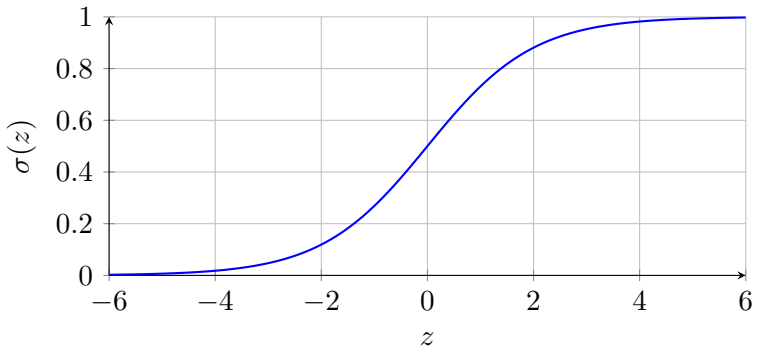

Figure 2.3: The sigmoid function: $\sigma(z) = 1/\left(1 + e^{-z}\right)$

The architectural structure typically consists of an input layer, one or more hidden layers, and an output layer. This configuration is called a fully connected layer, also known as a Multi-layer perceptron (Figure 2.4), and it is designed to create a more complex model that can better

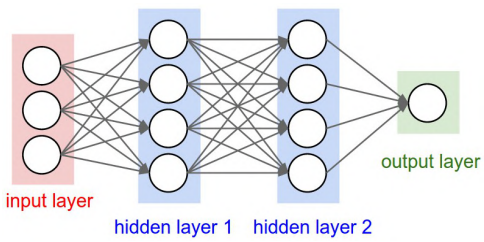

Figure 2.4: A fully connected layer or Multi-layer perceptron; neurons are combined and form a hidden layer. (A circle in a hidden layer is a single neuron.)[6]

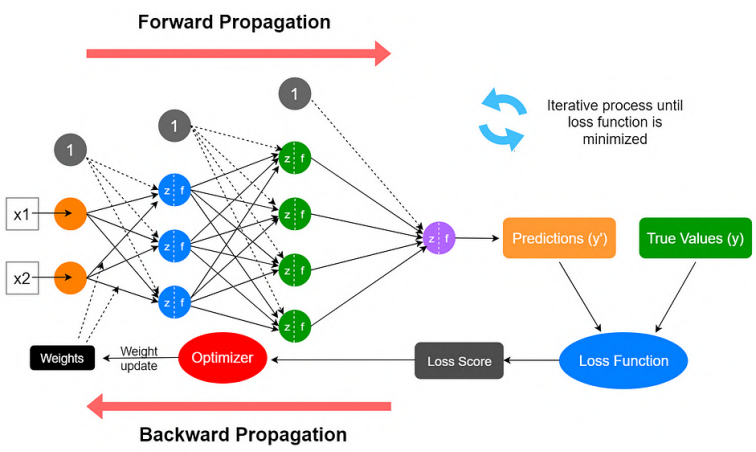

Figure 2.5: A training process for a neural network [7]

approximate complicated functions and identify patterns within datasets.

To train the network, we first must prepare a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$ containing n input values $\mathbf{x}_i$ where $\mathbf{x}_i$ is a 2-dimensional vector (as an example for Figure 2.5), and their corresponding true values y_i. During the training phase, each iteration involves two key steps. The first step is Forward propagation, where the inputs are passed through each hidden layer until they reach the final neuron to produce the final outputs (predictions) y_i'. Subsequently, the loss function computes the error between the predictions and true values. The second step involves Backward propagation, also known as backpropagation, which is an efficient algorithm used to update the network weights and biases in order to minimise the error in the loss function. Backpropagation is usually implemented in conjunction with an optimisation algorithm such as Gradient Descent. The training process continues iteratively until the loss function is minimised.

2.1.3 Convolutional Neural Network

Convolutional Neural Networks (CNNs) are a specialised class of deep learning models primarily designed for processing structured grid data, such as images. They have gained prominence due to their effectiveness in various computer vision tasks, including image classification, object detection, and more. The basic structure of a CNN can be observed in Figure 2.6, which comprises two main parts: Feature extraction and classification (assuming an image classification task). The feature extraction part is where the convolution and pooling operation are applied to enable the network to automatically learn the features and reduce the dimensionality of the input image before passing the results to the fully connected layer for classification.

The first component in the feature extraction stage is the Convolutional layer. Convolution ($*$) involves a weighted average operation on an image, taking an image matrix I and a kernel matrix K of size $(2N_1+1)\times(2N_2+1)$ as inputs and producing the feature map matrix as output

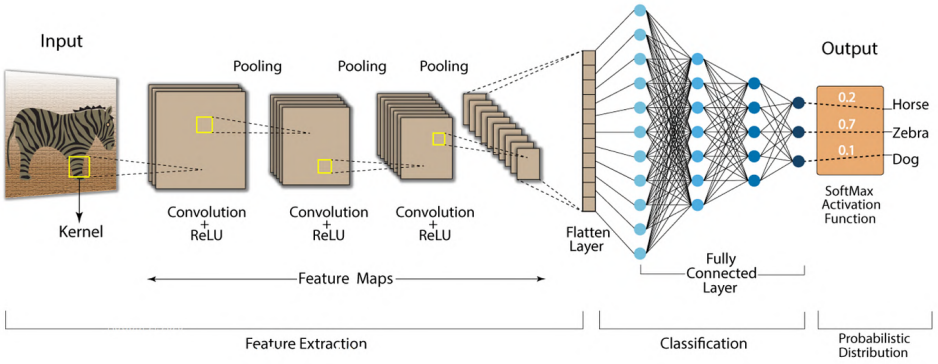

Figure 2.6: An architecture of CNN for an image classification task [8]

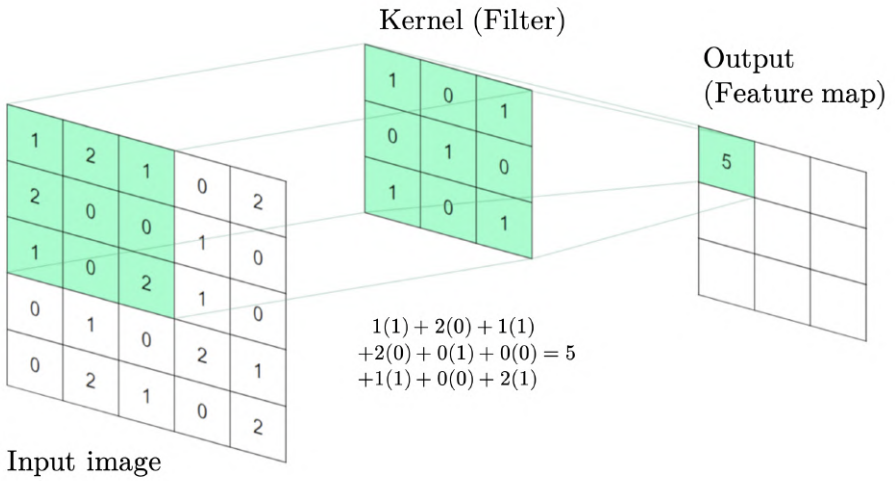

Figure 2.7: An example for computing the first element of the feature map matrix [9]

using Equation 2.3. The kernels in this network are trainable weights that move spatially across the image, identifying local patterns such as edges, textures, and more complex features in the deeper layers of the network. These kernels will be continuously adjusted during the training process in order to extract appropriate features that minimise the loss function. It is important to note that the activation function must be applied to each feature map in the convolution layer. For example, in Figure 2.6, the activation function ReLU is applied to the feature map.

$$(I * K)(x, y) = \sum_{i=-N_1}^{N_1} \sum_{j=-N_2}^{N_2} I(x - i, y - j) \cdot K(i, j) \tag{2.3}$$

Suppose an image with a volume of size $W_1 \times H_1 \times D_1$ is convoluted with K kernels, size F, stride S, and P zero-padding. The output volume size after a convolution layer $W_2 \times H_2 \times D_2$ can be calculated using Equation 2.4 and the number of trainable parameters per layer (including biases) is $K(F^2 D_1 + 1)$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1, \quad H_2 = \frac{H_1 - F + 2P}{S} + 1, \quad D_2 = K \tag{2.4}$$

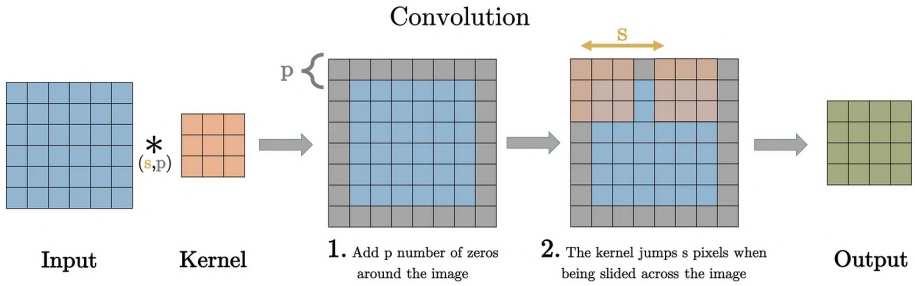

Figure 2.8: A graphic displaying the process of adding p zero-padding and moving by s strides [10]

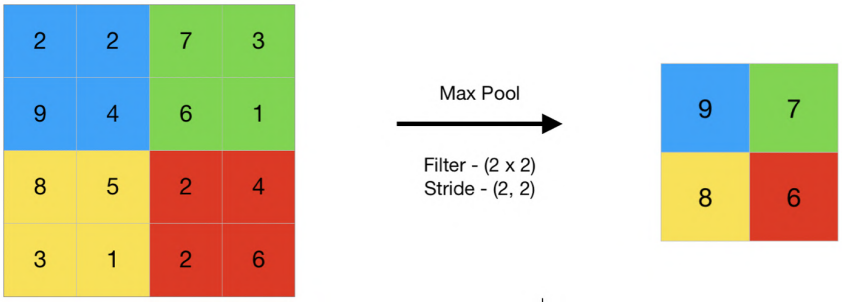

Figure 2.9: A graphic displaying the process of applying 2×2 max pooling operation with a stride of 2 to a 4×4 feature map [11]

The second component in the feature extraction stage is the Pooling layer. This layer reduces the spatial dimensions (width and height) of the input feature map, which increases the network's computational efficiency and reduces the risk of overfitting. Pooling layers achieve downsampling by summarising the features within a specific region, enabling the network to preserve crucial information while discarding less relevant details. Various methods can be used to summarise the features, such as Max pooling, which selects the maximum value within a specific window or region of the feature map. Suppose an $F \times F$ max pooling operation with a stride of S is applied to a feature map from a convolutional layer with a volume of size $W_2 \times H_2 \times D_2$. The output volume size after a convolution layer $W_3 \times H_3 \times D_3$ can be calculated using Equation 2.5. It is important to note that this layer doesn't have trainable parameters since it simply performs a downsampling method.

$$W_3 = \frac{W_2 - F}{S} + 1, \quad H_3 = \frac{H_2 - F}{S} + 1, \quad D_3 = D_2 \tag{2.5}$$

2.1.4 Vision Transformer

Vision Transformer (ViT) is a framework for tasks involving the recognition of images. It utilises the concepts of Transformer models, which were initially created for natural language processing (NLP) for tasks related to computer vision. In 2020, researchers at Google introduced the ViT,

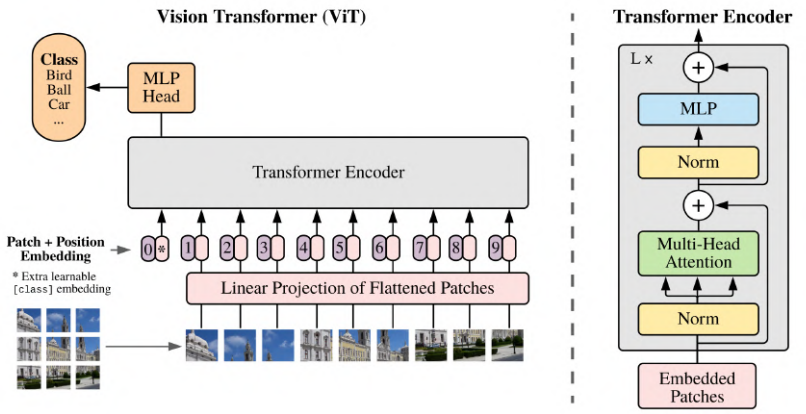

Figure 2.10: An overview architecture of ViT [12]

representing a significant change in the design of deep learning models for image analysis. This posed a challenge to the prevailing use of CNN in this field. In contrast to employing sliding kernels for feature extraction from an input image in CNN, ViT uses self-attention mechanisms to understand the relationship between patches of the complete image; this allows it to capture global dependencies. ViT only uses the encoder part of the original Transformer architecture from paper [12].

To gain insight into the workings of ViT, it is essential to begin by partitioning a complete image represented as $\mathbf{x} \in \mathbb{R}^{H \times W \times C}$. Here, H, W, and C correspond to the height, width, and channels of the image, respectively. The image $\mathbf{x}$ is split into N patches, each denoted as $\mathbf{x}_p^i \in \mathbb{R}^{1 \times (P^2 C)}; i = 1, 2, \ldots, N$ and having a resolution of P^2, where $N = HW/P^2$. Then, all patches are flattened and mapped to dimension D by a trainable linear projection $\mathbf{E} \in \mathbb{R}^{(P^2 C) \times D}$. An extra learnable class embedding $\mathbf{x}_{\text{class}} \in \mathbb{R}^{1 \times D}$ is added before the patches. After completing all necessary processes, each component is concatenated. Subsequently, the position embedded $\mathbf{E}_{\text{pos}} \in \mathbb{R}^{(N+1) \times D}$ is added to enable the network to comprehend the sequence of patches. All processes can be represented as a mathematical expression in Equation 2.6.

$$\mathbf{Z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \cdots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{\text{pos}} \tag{2.6}$$

After preparing $\mathbf{Z}_0$, it will be passed into the Transformer encoder L times. The Transformer encoder includes Layer Normalisation (LN), Multiheaded Self-Attention (MSA), and Multi-Layer Perceptron (MLP) blocks. The processes involved in the Transformer encoder can be represented by mathematical expressions in Equation 2.7 and Equation 2.8, where $l = 1, 2, \ldots, L$.

$$\mathbf{Z}'_l = MSA(LN(\mathbf{Z}_{l-1})) + \mathbf{Z}_{l-1} \tag{2.7}$$

$$\mathbf{Z}_l = MSA(LN(\mathbf{Z}'_l)) + \mathbf{Z}'_l \tag{2.8}$$

The Transformer encoder produces $\mathbf{Z}_L$. However only the first component $\mathbf{Z}_L^0$ is selected. Another Layer Normalisation is then applied to this component to obtain $\mathbf{y} = LN(\mathbf{Z}_L^0)$, which is the representation of the image. Finally, $\mathbf{y}$ is fed into the final Multi-Layer Perceptron (classification head) to make predictions for the image's class.

Let's look further into the concept of the self-attention mechanism. Suppose the input sequence $\mathbf{Z} \in \mathbb{R}^{N \times D}$. is fed to a self-attention block (SA); the initial step involves computing the Query ($\mathbf{Q}$), Key ($\mathbf{K}$), and Value ($\mathbf{V}$) matrices using a weight matrix $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{D \times D_h}$, as per Equation 2.9. Subsequently, the attention weights $\mathbf{A} \in \mathbb{R}^{N \times N}$, representing the similarity between $\mathbf{Q}$ and $\mathbf{K}$, are determined using Equation 2.10. The final output of this single self-attention operation is denoted as $SA(\mathbf{Z})$ and can be computed using Equation 2.11. To extend this idea, envision employing multiple "heads" to learn different features. If we have k heads, we would need to execute k self-attention operations concurrently and then concatenate the results using Equation 2.12, given a weight matrix $\mathbf{W}_{MSA} \in \mathbb{R}^{kD_h \times D}$. It is worth noting that the output of the Multihead self-attention $MSA(\mathbf{Z}) \in \mathbb{R}^{N \times D}$ has the same size as the initial input $\mathbf{Z}$.

$$\mathbf{Q} = \mathbf{Z}\mathbf{W}_Q, \qquad \mathbf{K} = \mathbf{Z}\mathbf{W}_K, \qquad \mathbf{V} = \mathbf{Z}\mathbf{W}_V \tag{2.9}$$

$$\mathbf{A} = \text{softmax}\left(\mathbf{Q}\mathbf{K}^\top / \sqrt{D_h}\right) \tag{2.10}$$

$$SA(\mathbf{Z}) = \mathbf{A}\mathbf{V} \tag{2.11}$$

$$MSA(\mathbf{Z}) = [SA_1(\mathbf{Z}); SA_2(\mathbf{Z}); \cdots; SA_k(\mathbf{Z})]\mathbf{W}_{MSA} \tag{2.12}$$

2.1.5 Transfer learning

Consider a scenario where a model is initially trained on a vast dataset such as ImageNet, which comprises around 14 million images across over 2000 classes [13]. This initial training process is known as pre-training. Subsequently, when given a smaller dataset of particular animal images, we aim to refine this pre-trained model to excel at classifying different animal species. This refinement process is referred to as fine-tuning. The concept of fine-tuning a pre-trained model is known as Transfer learning (illustrated in Figure 2.11), where a model developed for one task is repurposed as the basis for a model targeting a different task. Many pre-trained models are accessible for various fundamental tasks that require substantial time and resources. Leveraging these pre-trained models for a specific task with limited data, time, and resources can significantly enhance the model's performance within a short timeframe.

2.1.6 Underfitting and Overfitting

Suppose our dataset is divided into a training set and a validation set; the loss functions computed on each set are depicted in Figure 2.12. On the horizontal axis, we have the number of

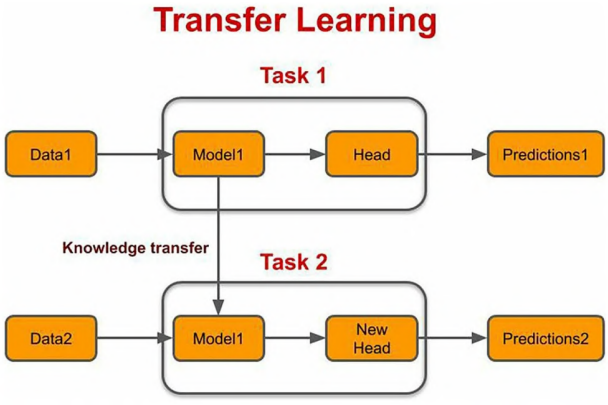

Figure 2.11: A diagram of the Transfer Learning concept [13]

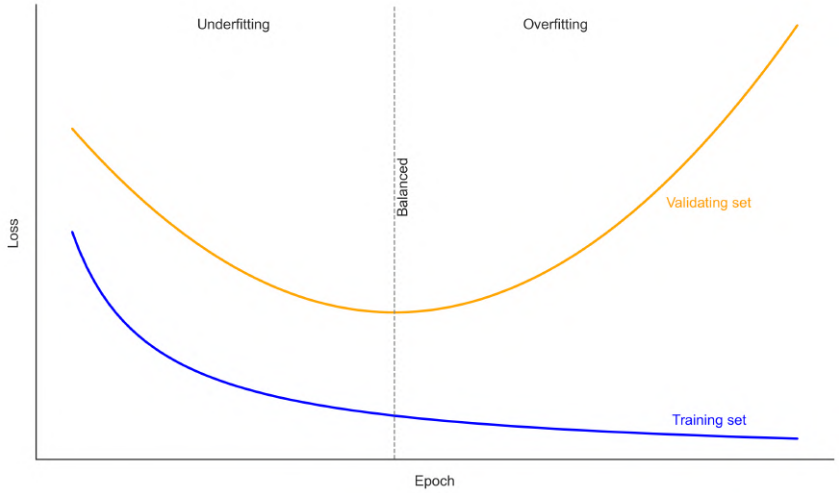

Figure 2.12: An example showing the concept of Underfitting and Overfitting

training epochs, while the vertical axis represents the loss value. During each epoch, the model aims to minimise the loss function. Thus, we expect the training loss to decrease with each epoch. The validation set is equivalent to an unseen exam for the model. A well-performing model would show low validation loss in this scenario. However, in Figure 2.12, the validation loss increases towards the end while the training loss decreases. This behaviour is typically associated with Overfitting, where the model memorises the tasks rather than truly understanding them before the exam. The optimal point, denoted by the balanced line in Figure 2.12, signifies the model's best performance before its performance begins to decline. To leverage this model effectively, we should select the model trained at this optimal epoch. Before reaching this optimal point, the model is still in the learning phase and not yet ready. This stage is referred to as Underfitting. Figure 2.13 illustrates the model's behaviour across different stages in classification and regression tasks.

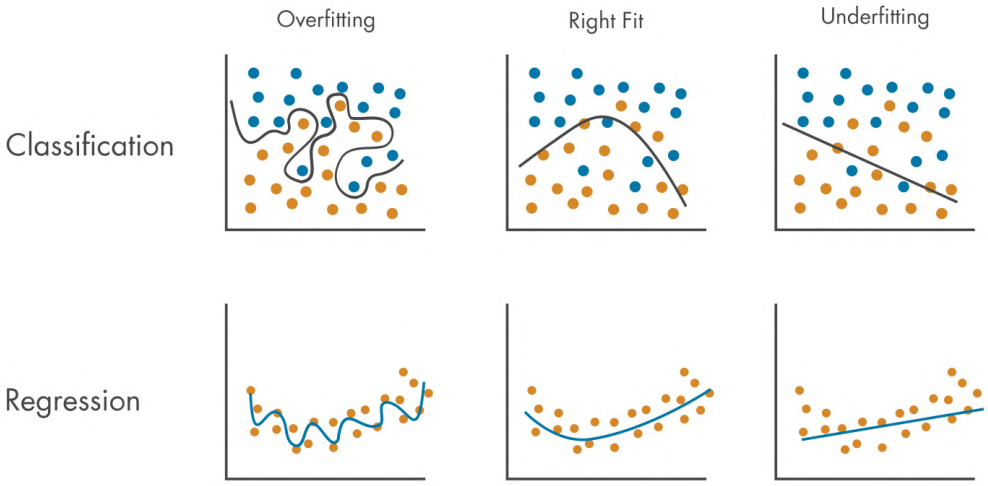

Figure 2.13: An example showing of how the model performs in classification and regression tasks at the stages of Overfitting, Right Fit (Balanced), and Underfitting. Each dot represents a sample or data point. In the classification task, there are two classes of samples (blue and orange). The model's aim is to produce the optimal boundary line to separate these two classes of samples. In the regression task, the goal of the model is to generate the best-fitting line to enclose all the samples. [14]

2.2 Dataset

The models in this project will be trained using the Places365-Standard dataset [15], which is a comprehensive collection of scenes used for recognition. This dataset includes 1,803,460 training images categorised into 365 different scene categories. The main training samples for this project consist of 1,000 crosswalk images, as they feature cars, roads, crosswalks, intersections, and more. These crosswalk images were chosen because the goal is to correct some real fisheye images from the FishEye8K dataset, which shares content similar to the selected dataset. Additionally, 100 images of train station platforms were selected for certain experiments. Further sections will provide more details about both selected datasets. The sample images will be shown in Figure 2.14.

Let's discuss the FishEye8K dataset [16] in greater detail. This dataset is specifically designed for road object detection tasks and comprises 8,000 images captured in 22 videos, featuring a total of 157,000 bounding boxes across five classes: Pedestrian, Bike, Car, Bus, and Truck. The images were captured using 18 fisheye cameras for traffic monitoring. While this dataset won't be utilised for training in this project, selected images from certain cameras, shown in Figure 2.15, will be used to evaluate the performance of the model predictions and the quality of the rectification program.

Figure 2.14: Some Crosswalk images (left) and Train station platform images (right) from the Place365 dataset [15]

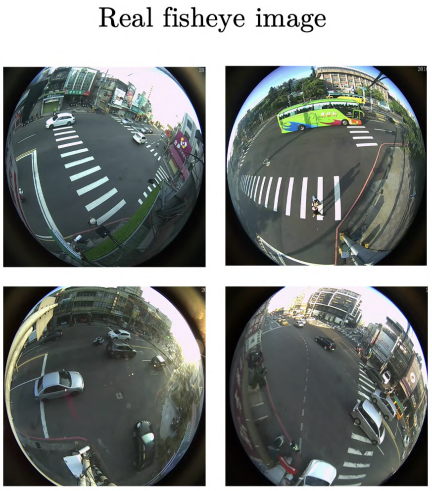

Figure 2.15: Some images from different cameras from the FishEye8K dataset [16]

Stage	Description
1.	Edge detection using Canny method. in: Input image out: Edge point coordinates and orientation.
2.	Initial estimation of the lens distortion parameter using improved Hough transform in: Edge point coordinates and orientation out: An estimation of the distortion parameter and the most voted lines
3.	Distortion parameter optimization in: An estimation of the distortion parameter and the most voted lines out: An optimized lens distortion parameter.
4.	Image distortion correction in: The input image and the optimized lens distortion model out: A distortion-free output image obtained using the estimated model.

Figure 2.16: The procedure summary for [17]'s algorithm

2.3 Related works

The correction of fisheye images can be achieved through two main methods: traditional geometry-based and deep learning-based methods. Geometry-based methods use geometric features such as specific points (e.g., vanishing points), lines, and shapes to rectify the distortion. In contrast, deep learning-based methods involve learning the distortion parameters from the distorted images to transform them back to their original undistorted state. Additionally, some models can produce undistorted images directly without learning distortion parameters.

2.3.1 Traditional geometry-based method

One simple method in traditional geometry-based methods involves correcting distortions using line features. This typically entails detecting lines in the distorted image and subsequently employing an optimisation process to rectify the distortion, restoring the appearance of these lines to a straight orientation. However, the effectiveness of line-based methods can be interfered with by noise, line detection errors, and the complexity of fitting parameters in highly distorted images.

Aleman-Flores et al. [17] developed a method to automatically correct radial distortion by utilising the improved Hough transform. They extended the traditional Hough transform, which is used to detect straight lines in images, by adding a third dimension to account for the distortion parameter. This modification enables the algorithm to detect lines and estimate the distortion simultaneously within one framework. By integrating the distortion parameter into the Hough space, the method accurately identifies distorted lines and uses them to evaluate lens distortion without requiring manual input or predefined calibration patterns. The complete algorithm comprises four steps: 1) Identifying edges in the image using the Canny edge detector. 2) Estimating the lens distortion parameter using the improved Hough transform. 3) Refining the initial estimate through optimisation techniques to minimise the error in line straightness. 4) Correcting the image using the optimised distortion parameter. The summation of the procedure can be seen in Figure 2.16.

2.3.2 Deep learning-based method

In more recent times, deep learning has become increasingly important. There are two approaches to addressing this issue within deep learning. The first approach involves predicting distortion parameters using neural networks on synthesised (distorted) images created by applying known distortions to clean images. The distorted images are then corrected using specific functions. The second approach, on the other hand, focuses on generating rectified images directly without the need to estimate specific distortion parameters, disregarding the use of distortion models. However, learning-based methods face a major obstacle in requiring substantial training data. Obtaining real-world paired datasets of distorted and rectified images is difficult, which is why synthetic data is frequently used. Nevertheless, models trained on synthetic data may have limited generalisability if the dataset does not contain diverse distortions.

Rong et al. [18] employed the distortion parameter prediction approach in their research. They presented their overall process in Figure 2.17. Initially, they chose images from the ImageNet dataset, considering the number of line segments in the images. In real-world scenes, straight lines appear curved in the image plane when distorted. Therefore, the presence of line segments assists CNN in learning to identify and rectify the distortion. Consequently, the chosen images should contain a substantial number of line segments. After that, they generated distorted versions of their selected images using Equation 2.1, resulting in around 55,000 training images, 6,000 validation images, and 6,000 testing images. They employed LeNet and AlexNet (a CNN architecture) to predict the distortion parameter, formulating the problem as a 401-class classification problem with the softmax loss function. The models' output corresponds to a predicted class. However, they converted the discrete predicted classes into continuous predicted distortion parameters by implementing straightforward weighted averages between the output probability vectors and all 401 distortion parameters. Finally, they assessed their outcomes using their own two metrics. The first one, Mean Length Increase, focuses on how much the correction process enhances the average length of straight lines, where higher scores indicate better correction. The second metric evaluates the standard deviation of lengths, which assesses the variability in the lengths of corrected lines, where higher values mean a more realistic and practical distortion correction. They also compared their CNN-based approach with traditional methods, including the One-Parameter Division Model method proposed by Aleman-Flores et al. The CNN-based method outperformed the traditional methods significantly, especially in scenarios where the images contained fewer line segments. Nonetheless, in some instances of highly severe distortion, there are limitations where their method also struggled to perform. This is because the distortion parameter might exceed the range their synthetic training data covers, resulting in less accurate corrections.

Li et al. [3] framework overview can be seen in Figure 2.18. They represented the geometric distortion in an image as a displacement field, also called a flow. This flow indicates how each pixel in a distorted image should shift to align with a corrected, undistorted image. They utilised CNN to predict this flow. These networks were trained on an vast synthetic dataset of distorted images, with each image paired with its corresponding undistorted version. The

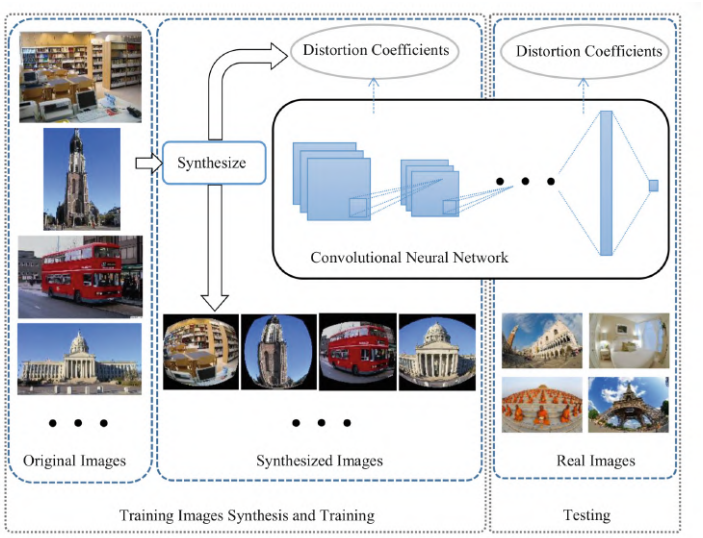

Figure 2.17: An overview of the complete framwork of [18]

network learned to map the distorted image to its displacement field throughout the training. They initially defined six distinct types of distortion models: barrel distortion, pincushion, rotation, shear, perspective, and wave distortion. Subsequently, two networks are introduced in this article. The first one is a Single-Model Network (GeoNetS). This network is designed to deal with a specific type of distortion. It predicts the distortion parameters and then generates the displacement field based on them. The other one is a Multi-Model Network (GeoNetM). This network is more adaptable and can concurrently handle various types of distortions. It involves a classification branch that first identifies the type of distortion in the image and then predicts the displacement field accordingly. This network does not depend on predefined distortion models, making it more universal. Once the flow is predicted, the model fitting stage enhances the prediction by estimating the actual distortion parameters. This incorporates methods such as the Hough Transform, which helps precisely fit the flow to the distortion model, improving the accuracy of the correction. The final step is to apply the predicted flow to the distorted image, resampling it to produce the corrected, undistorted image. They proposed an effective, iterative resampling approach that ensures high-quality results with quicker convergence. In summary, their method can predict the displacement field/flow and distortion parameters (if needed) and directly generate the corrected image. Finally, the evaluation metric utilised in this study is the endpoint error (EPE). It measures the variance between the predicted displacement (flow) and the ground truth displacement at each pixel in an image.

2.4 Research gap

The direct image generation method necessitated a much longer timeframe and a substantial amount of data to train the model. This approach is unsuitable for a 14-week master-level project with limited GPU hours obtained from the school. Therefore, this project will focus on the distortion parameter prediction approach. The main limitations of most papers using this

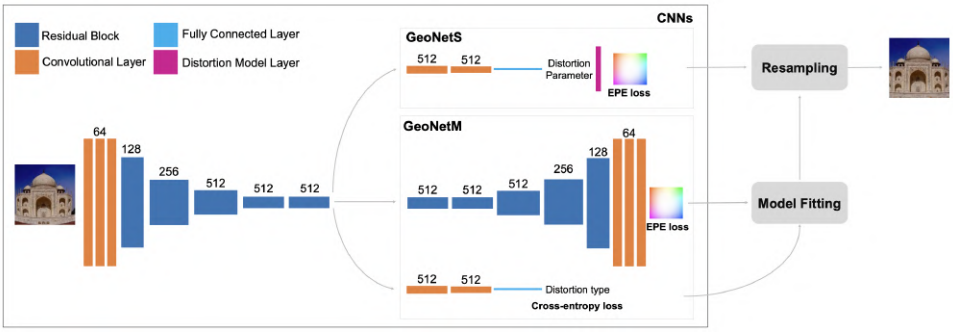

Figure 2.18: An overview of the complete framwork of [3]

approach are: 1) They have not released the codes and model weights used for predicting the distortion parameter. 2) They have not provided details or published codes about the image rectification method. This results in difficulty in replicating, improving, and explaining the task. As a result, this project draws inspiration from [18], making adjustments in settings and approaches and conducting experiments with different model architectures and training schemes. Additionally, this project aims to create an explainable image distortion and rectification program inspired by the program developed by [19], which lacks detailed information. Fortunately, [3] has made the code and model weights publicly available. Consequently, the rectified image results of this project can be compared with those from other papers despite using a different approach. When evaluating the rectified image results, the metrics used by [18] and [3] are overly complicated to implement. Therefore, this project will mainly use SSIM and PSNR as image evaluation metrics, which are widely used in other papers in this field [20], [21]. Lastly, this project seeks to rectify real fisheye images from the FishEye8K dataset to implement this project to real-world data.

Chapter 3

Methodology

The overview framework for this project can be seen in Figure 3.1. The project is divided into three main parts: 1) Dataset synthesis, 2) Model training and validation, and 3) Rectifying and image evaluation. In the first part, clean images are distorted using ground truth distortion parameters denoted as k_{gt} to create a dataset. The distorted images are then used to train models to predict k_{pred}. The difference between k_{gt} and k_{pred} is calculated using a loss function. The aim is to identify the best model through a series of experiments. Once the best model is selected, it will be employed in the testing images to obtain k_{pred}. These distorted images and k_{pred} will be used to refine the output images. Finally, the similarity and quality of the output image compared to the clean image will be measured using SSIM and PSNR as evaluation metrics.

3.1 Image distortion and rectification program

The program's overview of distorting and rectifying images is presented in Algorithm 1, with additional details explained further in this section. This program requires a clean image of size $s \times s$ and the distortion parameter k as inputs. First, it calculates the centre coordinate of the distortion (DC), followed by estimating the Resolution Scaling Factor (RSF). The RSF is then used to generate a set of base coordinates to determine distorted and rectified coordinates. Finally, interpolation is performed to create the final distorted image by sampling the original image (determining the pixel value that should be assigned to each pixel in the distorted image by looking at the original image.) The rectification process is carried out similarly. It's essential to note that the program can take the clean image as input and perform distortion and rectification in a single procedure, or it can perform each task separately.

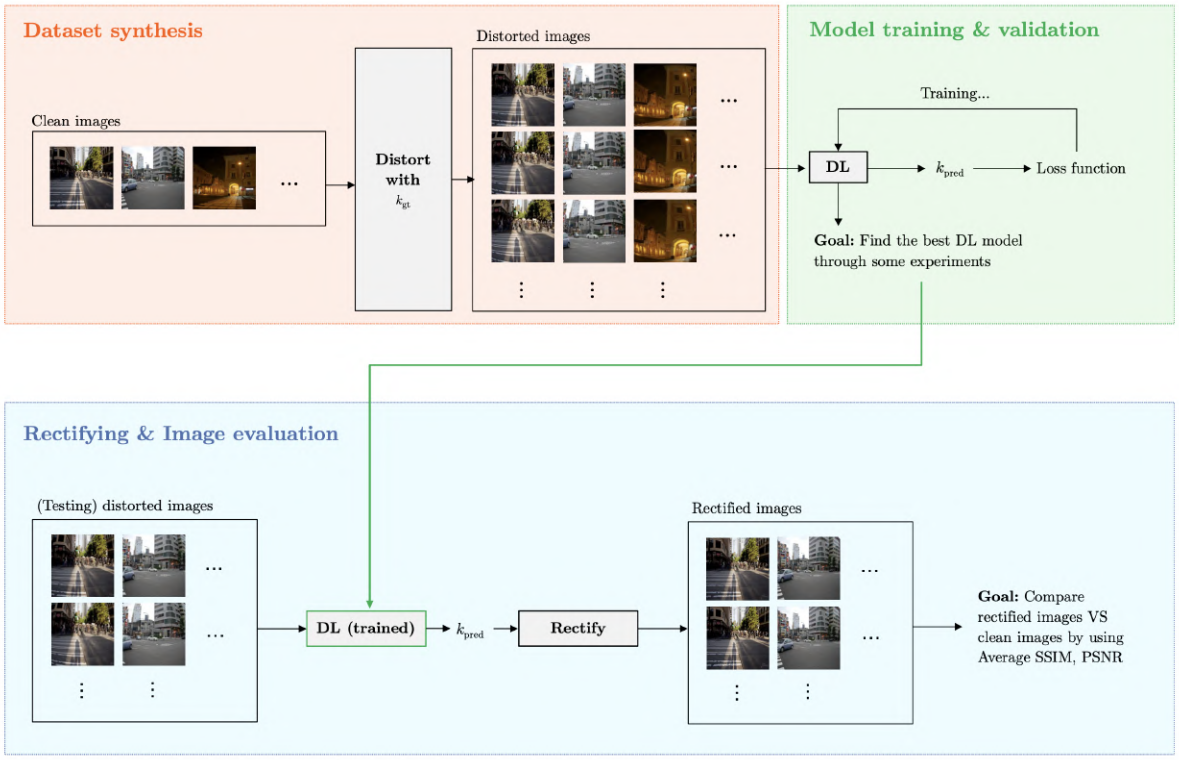

Figure 3.1: Overview framework for this project

Algorithm 1 Image distortion and rectification

Input: Clean image of size $s \times s$, distortion parameter $k \in [10^{-6}, 10^{-4}]$
Output: Distorted image of size $s' \times s'$, Rectified image of size $s' \times s'$
$DC \leftarrow \frac{s}{2} + 0.1$ ▷ Centre coordinate of the distortion
$RSF \leftarrow$ `GetRSF`(s, k, DC) ▷ Resolution Scaling Factor
Base coordinates $\leftarrow$ `GetBaseCoords`(s, k, DC, RSF)
Distorted coordinates $\leftarrow$ `GetDistortedCoords`$(k, DC,$ Base coordinates$)$
Rectified coordinates $\leftarrow$ `GetRectifiedCoords`$(k, DC,$ Base coordinates$)$
Clean image coordinates $\leftarrow \{(i, j) \in \mathcal{Z} \times \mathcal{Z} \mid i, j \in [0, s]\}$
Distorted image $\leftarrow$ `Distort`(Clean image coordinates, Distorted coordinates, Clean image)
Rectified image $\leftarrow$ `Rectify`(Base coordinates, Rectified coordinates, Distorted image)

3.1.1 The inverse of the Division model

As per Equation 2.1, calculating the coordinates for the corrected image based on the distorted coordinate, the centre coordinates, and the distortion parameter is a straightforward process. However, the synthesis of a distorted image from a clean image needs a reverse mathematical equation to transform clean image coordinates into distorted ones. The subsequent results detailing the derivation of such an equation are presented below.

Suppose we need to calculate the distorted coordinate (x_d, y_d) from the corrected coordinate (x_c, y_c), the centre coordinates (x_m, y_m), and the distortion parameter k. From Equation 2.1, we can separately write the relationship between (x_c, y_c) and (x_d, y_d) as

$$x_c - x_m = \frac{x_d - x_m}{1 + k\left[(x_d - x_m)^2 + (y_d - y_m)^2\right]} \tag{3.1}$$

$$y_c - y_m = \frac{y_d - y_m}{1 + k\left[(x_d - x_m)^2 + (y_d - y_m)^2\right]} \tag{3.2}$$

Define $r_d^2 = (x_d - x_m)^2 + (y_d - y_m)^2$. Using Equation 3.1 and Equation 3.2, we can express the following:

$$x_c - x_m = \frac{x_d - x_m}{1 + kr_d^2} \quad \longrightarrow \quad x_d = x_m + (x_c - x_m)(1 + kr_d^2) \tag{3.3}$$

$$y_c - y_m = \frac{y_d - y_m}{1 + kr_d^2} \quad \longrightarrow \quad y_d = y_m + (y_c - y_m)(1 + kr_d^2) \tag{3.4}$$

Substituting Equation 3.3 and Equation 3.4 into the definition of r_d^2 , we derive:

$$\begin{aligned} r_d^2 &= \left[(x_c - x_m)(1 + kr_d^2)\right]^2 + \left[(y_c - y_m)(1 + kr_d^2)\right]^2 \\ &= (1 + kr_d^2)^2\left[(x_c - x_m)^2 + (y_c - y_m)^2\right] \end{aligned} \tag{3.5}$$

Let $r_c^2 = (x_c - x_m)^2 + (y_c - y_m)^2$. From Equation 3.5, we can rewrite:

$$\begin{gathered} r_d^2 = (1 + kr_d^2)^2 r_c^2 \\ [k^2 r_c^2](r_d^2)^2 + [2kr_c^2 - 1](r_d^2) + r_c^2 = 0 \\ r_d^2 = \frac{(1 - 2kr_c^2) \pm \sqrt{1 - 4kr_c^2}}{2k^2 r_c^2} \end{gathered} \tag{3.6}$$

Substituting Equation 3.6 into Equation 3.3 and Equation 3.4, we obtain:

$$x_d = x_m + \frac{(x_c - x_m)\left[1 \pm \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} \tag{3.7}$$

$$y_d = y_m + \frac{(y_c - y_m)\left[1 \pm \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} \tag{3.8}$$

As $(x_c, y_c) \to (x_m, y_m)$ or $r_c^2 \to 0$, we should have $r_d^2 \to 0$ or $(x_d, y_d) \to (x_m, y_m)$. Therefore, Equation 3.7 and Equation 3.8 can be combined and simplified into Equation 3.9 (See proof: Section B.1), which will be defined as the inverse of Equation 2.1. Additionally, Equation 2.1 will be re-expressed as Equation 3.10 for comparison with Equation 3.9.

$$\begin{bmatrix} x_d - x_m \\ y_d - y_m \end{bmatrix} = \frac{1 - \sqrt{1 - 4kr_c^2}}{2kr_c^2} \begin{bmatrix} x_c - x_m \\ y_c - y_m \end{bmatrix} \tag{3.9}$$

$$\begin{bmatrix} x_c - x_m \\ y_c - y_m \end{bmatrix} = \frac{1}{1 + kr_d^2} \begin{bmatrix} x_d - x_m \\ y_d - y_m \end{bmatrix} \tag{3.10}$$

Section B.2 and Section B.3 will also provide the derivation of Equation 3.9 by using different approaches.

3.1.2 Step 1: Determine the Resolution Scaling Factor

When an image is distorted, certain areas may become stretched or compressed, leading to a reduction in the image resolution. To address this, we need to calculate the Resolution Scaling Factor (RSF). We will begin by selecting the coordinates in the middle column of the original image and then applying the distortion using Equation 3.9, given the distortion parameter k. The distorted coordinates can be observed in Figure 3.2. As we are focusing on the middle column, it will experience the highest level of distortion compared to other columns. Let's denote $y_d^{\min}$ as the lowest y-coordinate and $y_d^{\max}$ as the highest y-coordinate. RSF can be computed using Equation 3.11, where s represents the size of the original image.

$$RSF = \frac{s}{\text{Range of distorted coordinates}} = \frac{s}{y_d^{\max} - y_d^{\min}} \tag{3.11}$$

3.1.3 Step 2: Create base coordinates

The base coordinates serve as a higher-resolution version of the clean image coordinates and will be used to calculate the distorted and rectified coordinates later. To obtain the base coordinates,

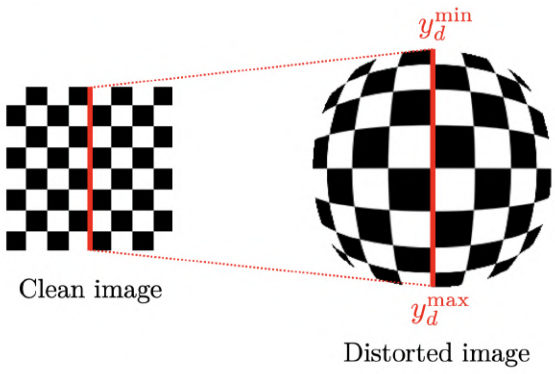

Figure 3.2: Distortion of the middle column of the clean image; In Python, image coordinates range from top $(0, 0)$ to the bottom, resulting in higher coordinates below.

we first assemble a set comprising all clean image coordinates: $\{(i, j) \in \mathcal{Z} \times \mathcal{Z} \mid i, j \in [0, s]\}$; $\mathcal{Z}$ is a set of integers, as depicted in Table 3.1. Each coordinate is represented as a positive integer following the Python format. Subsequently, based on the calculated RSF, we derive a set of base coordinates: $\{(i, j) \in \mathcal{R} \times \mathcal{R} \mid i, j \in [0, s]\}$; $\mathcal{R}$ is a set of real numbers, shown in Table 3.2. Notably, the resolution of the base coordinates is greater than that of the clean image due to the RSF. Strong distortions lead to an expanded range of distorted coordinates $(y_d^{\max} - y_d^{\min})$ and a reduced RSF, resulting in the higher resolution of the base coordinates. This higher resolution will significantly enhance precision in the subsequent interpolation step. An illustration of the base coordinates when $RSF = 0.5$ can be found in Table 3.3.

3.1.4 Step 3: Calculate the distorted and rectified coordinates

With the base coordinates in hand, Equation 3.9 and Equation 3.10 can be used to calculate the distorted and rectified coordinates, respectively. These coordinates will play a crucial role in generating the final output images in the subsequent stage. In Figure 3.3, the sizes of all coordinate sets are shown, along with the diagram for calculating the distorted and rectified coordinates. This is demonstrated in the case where the clean image coordinate size is 10×10 and an RSF value of 0.5.

3.1.5 Step 4: Calculate the distorted and rectified coordinates

To create the distorted image from the clean image, we will look through all the distorted coordinates that we just calculated. We will assign the pixel values to these distorted coordinates by taking the pixel values from the clean image at the exact coordinates. In other words, we're finding the corresponding pixel value from the original clean image based on where the distorted coordinates tell us that pixel should come from. This process is known as sampling the original image at the distorted positions and can be observed on the left side of Figure 3.4.

The original image coordinates are represented as positive integers, while the distorted coordi-

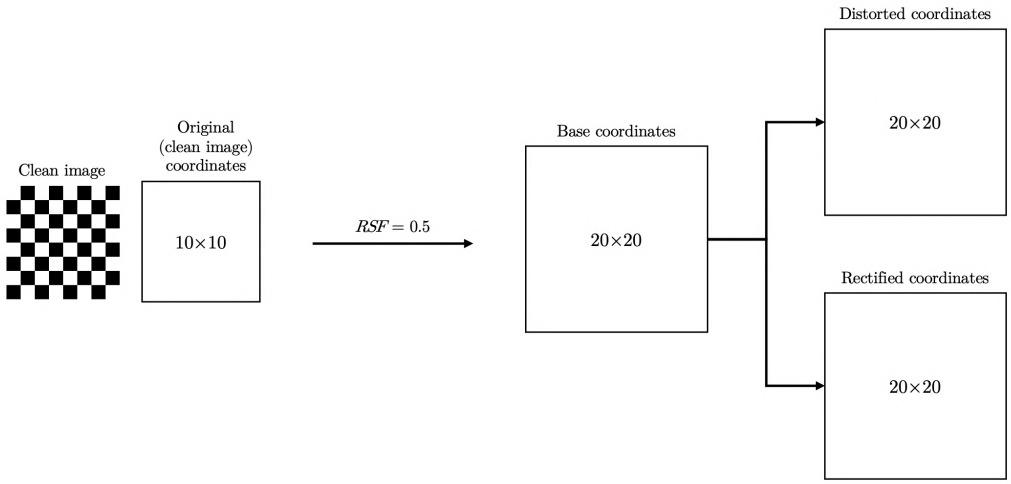

Figure 3.3: The diagram shows how the distorted and rectified coordinates are computed, with the given dimensions of an example 10×10 clean image and $RSF = 0.5$. Although the clean image already contains the coordinates within it, for better visualisation, we will display the clean image and its coordinates side by side.

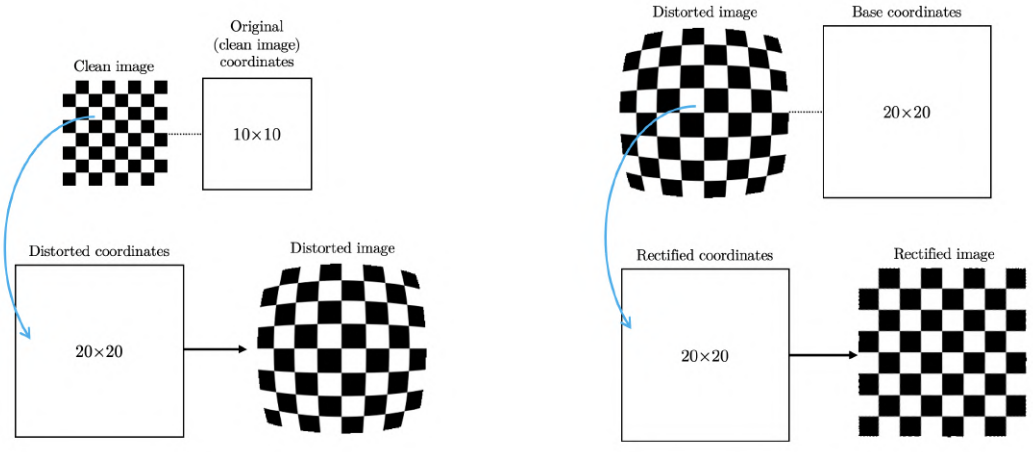

Figure 3.4: The diagram of generating the distorted image (on the left side) and rectified image (on the right side). The blue arrow represents the process of sampling the pixel values. We assume the size of the clean image is 10×10, and $RSF = 0.5$.

$(0,0)$	$(1,0)$	$(2,0)$	$(3,0)$	$\cdots$	$(s,0)$
$(0,1)$	$(1,1)$	$(2,1)$	$(3,1)$	$\cdots$	$(s,1)$
$(0,2)$	$(1,2)$	$(2,2)$	$(3,2)$	$\cdots$	$(s,2)$
$(0,3)$	$(1,3)$	$(2,3)$	$(3,3)$	$\cdots$	$(s,3)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0,s)$	$(1,s)$	$(2,s)$	$(3,s)$	$\cdots$	(s,s)

Table 3.1: Original (clean image) coordinates: Size $= s^2$

$(0,0)$	$(RSF,0)$	$(2RSF,0)$	$(3RSF,0)$	$\cdots$	$(s,0)$
$(0,RSF)$	(RSF,RSF)	$(2RSF,RSF)$	$(3RSF,RSF)$	$\cdots$	(s,RSF)
$(0,2RSF)$	$(RSF,2RSF)$	$(2RSF,2RSF)$	$(3RSF,2RSF)$	$\cdots$	$(s,2RSF)$
$(0,3RSF)$	$(RSF,3RSF)$	$(2RSF,3RSF)$	$(3RSF,3RSF)$	$\cdots$	$(s,3RSF)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0,s)$	(RSF,s)	$(2RSF,s)$	$(3RSF,s)$	$\cdots$	(s,s)

Table 3.2: Base coordinates: Size $= \lceil s/RSF \rceil^2$

$(0,0)$	$(0.5,0)$	$(1,0)$	$(1.5,0)$	$\cdots$	$(s,0)$
$(0,0.5)$	$(0.5,1)$	$(1,1)$	$(1.5,1)$	$\cdots$	$(s,1)$
$(0,1)$	$(0.5,1)$	$(1,1)$	$(1.5,1)$	$\cdots$	$(s,1)$
$(0,1.5)$	$(0.5,1.5)$	$(1,1.5)$	$(1.5,1.5)$	$\cdots$	$(s,1.5)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
$(0,s)$	$(0.5,s)$	$(1,s)$	$(1.5,s)$	$\cdots$	(s,s)

Table 3.3: Base coordinates; $RSF = 0.5$: Size $= 4s^2$

nates are real numbers. This presents a problem when attempting to retrieve the pixel value from the original image at non-integer coordinates, which do not correspond to actual pixels. Interpolation resolves this issue by enabling the estimation of pixel values at non-integer or "in-between" coordinates. As a result, interpolation is essential in creating a smooth final image, ensuring a visually cohesive outcome. The concept of interpolation is illustrated in Figure 3.5, demonstrating its functionality on an image.

Likewise, when creating the rectified image from the distorted image (the right side of Figure 3.4), we iterate through all the base coordinates. At each base coordinate, we assign the pixel values from the distorted image to the rectified coordinates. It's crucial to note that the base coordinates correspond to the coordinates in the distorted image.

3.2 Dataset synthesis

Deep learning models aim to predict the distortion parameters k based on the distorted images. Therefore, the dataset will consist of distorted images and their corresponding distortion parameters k. To begin, we will randomly generate k by defining 99 distortion levels (ranging from 0 to 99), with each level having an interval size of 10^{-6}, starting from 10^{-6} to 10^{-4}. Each clean

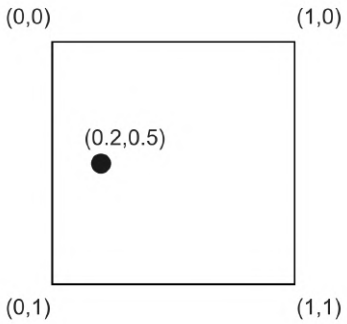

Figure 3.5: An illustration of interpolation: When given a clean image, the goal is to calculate the pixel value at coordinate $(0.2, 0.5)$, which is not on the grid of the clean image. This involves identifying the nearest four coordinates on the clean image's grid— $(0, 0)$, $(1, 0)$, $(0, 1)$, and $(1, 1)$. Subsequently, based on the pixel values of these coordinates, the pixel value at coordinate $(0.2, 0.5)$ can be calculated through interpolation.

image will be distorted across 99 levels (resulting in 99 images) with different randomly chosen k_{gt} values from each interval for each level (as shown in Table 3.4). The distorted images will serve as input features, the distortion level will be considered as a class for a classification task, and the distortion parameters k will act as target values for a regression task. The complete dataset will comprise a total of 99,000 samples (1,000 images $\times$ 99 levels), which will then be partitioned and shuffled into training, validation, and testing sets in an 80:10:10 ratio. The schematic of the synthesis process is illustrated in Figure 3.6. Prior to training, it is crucial to crop all distorted images (Figure 3.7), as failure to do so may result in the model gaining an unfair advantage by looking at the edge of the image content.

Distortion levels (99 classes)	**Interval of distortion parameter** (for each level) (Interval size = 10^{-6})	k_{gt} (randomly chosen from each interval)
0	1×10^{-6} to 2×10^{-6}	1.87×10^{-6}
1	2×10^{-6} to 3×10^{-6}	2.42×10^{-6}
2	3×10^{-6} to 4×10^{-6}	3.01×10^{-6}
⋮	⋮	⋮
96	9.7×10^{-5} to 9.8×10^{-5}	9.73×10^{-5}
97	9.8×10^{-5} to 9.9×10^{-5}	9.89×10^{-5}
98	9.9×10^{-5} to 1×10^{-4}	9.96×10^{-5}

Table 3.4: A table showing distortion levels, corresponding intervals, and an example of how k_{gt} is randomly chosen for each level.

3.3 Model training schemes

In this project, we explored three different strategies for training the models, all aimed at achieving the common objective of predicting distortion parameters k.

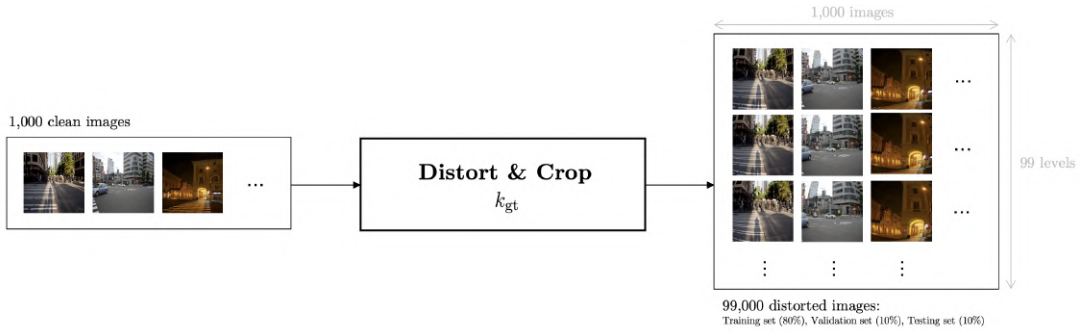

Figure 3.6: A diagram of the dataset synthesis process; The training set contains 79,200 samples, the validation set contains 9,900 samples, and the testing set contains 9,900 samples.

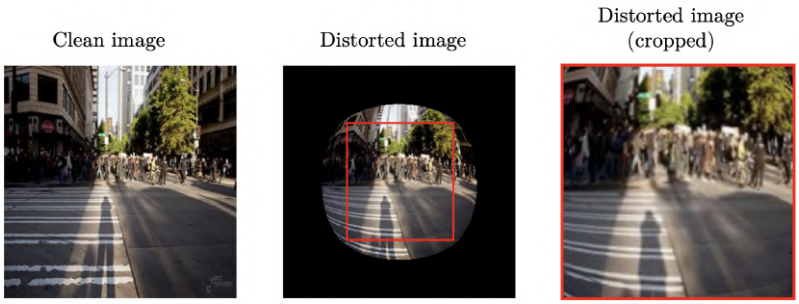

Figure 3.7: Cropping process after distortion; the process aims to create the largest square crop that fits within the round-shaped meaningful image content.

The first scheme simply treats this prediction problem as a regression task. The model aims to predict the distortion parameters k directly and utilises the Mean Squared Error (MSE) loss as a loss function for updating the weights. The training process for this regression approach is illustrated in Figure 3.8 (Top).

In the second scheme, the issue is framed as a classification problem. Instead of predicting the distortion parameters k, the model now seeks to predict the distortion level (class) and utilises the Cross-Entropy loss as the loss function to adjust the weights. The training process for this classification method is illustrated in Figure 3.8 (Middle). Here, $\mathbf{p}_{\text{gt}}$ denotes the ground truth probability vector corresponding to k_{gt}, which is a one-hot encoded vector where the ground truth class index is assigned a value of 1, and all other positions are set to 0. On the other hand, $\mathbf{p}_{\text{pred}}$ represents the predicted probability vector or softmax output vector, representing the model's confidence that the input belongs to that specific class. A depiction of $\mathbf{p}_{\text{gt}}$ and $\mathbf{p}_{\text{pred}}$ can be found in Table 3.5.

In the final scheme, referred to as Classification + Regression, the approach basically treats the problem as a classification task, resulting in the output $\mathbf{p}_{\text{pred}}$. Additionally, it computes the k_{pred} by using Equation 3.12, which basically involves performing a weighted average between $\mathbf{p}_{\text{pred}}$ and $\mathbf{k}_{\text{mid}}$. Each element of $\mathbf{k}_{\text{mid}}$ represents the midpoint for each interval of each level, serving as a single numerical representation for each level. An example of $\mathbf{k}_{\text{mid}}$ and $\mathbf{p}_{\text{pred}}$ can be observed in Table 3.5. Following this, $\mathbf{p}_{\text{pred}}$ is evaluated against $\mathbf{p}_{\text{gt}}$ using the Cross-Entropy

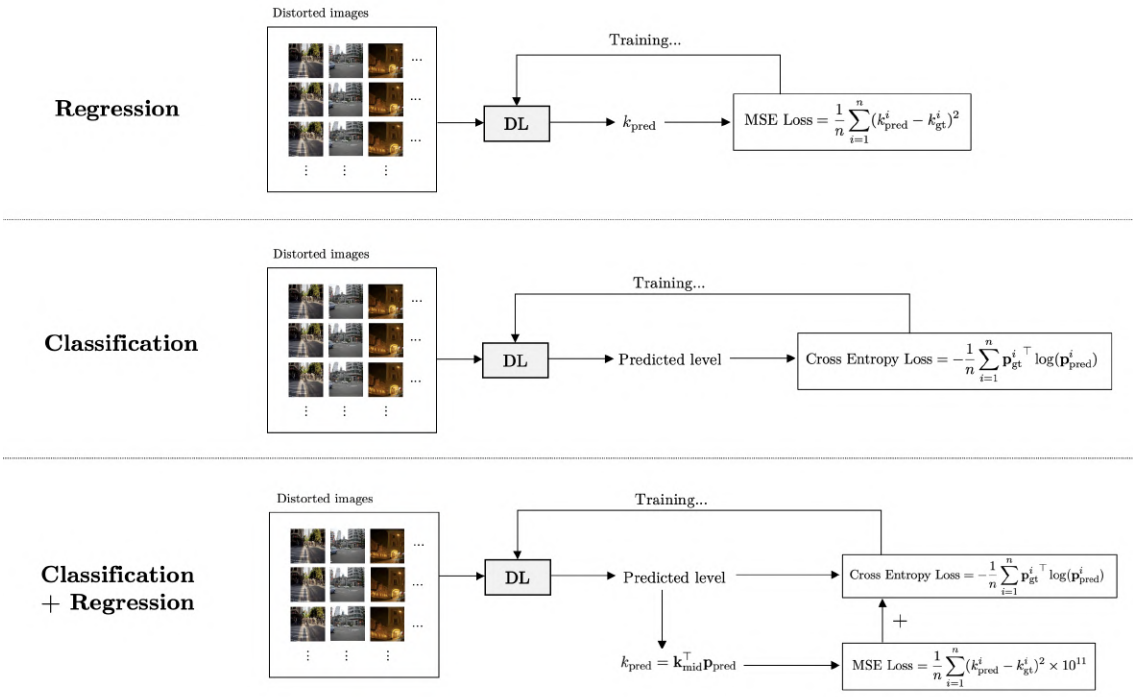

Figure 3.8: The training process for a regression, classification, and classification + regression scheme; n represents the number of training samples.

loss, while k_{pred} is compared with k_{gt} using the (Rescaled) MSE loss. The MSE loss used in this approach needs to be rescaled due to the substantially low value of the distortion parameter. This causes the MSE between the k_{gt} and k_{pred} values to be even lower compared to the Cross-entropy loss. Therefore, it is necessary to significantly rescale the MSE loss to enable the models to learn effectively alongside the Cross-entropy loss. Both loss functions are ultimately combined before updating the weights. The training process for this classification + regression scheme is depicted in Figure 3.8 (Bottom).

$$k_{\text{pred}} = \mathbf{k}^{\top}_{\text{mid}}\mathbf{p}_{\text{pred}} \tag{3.12}$$

3.4 Model architectures

In this project, we will experiment with two model architectures: AlexNet and Vision Transformer. AlexNet, a convolutional neural network (depicted in Figure 3.9), was introduced by Alex Krizhevsky and colleagues in 2012 [22]. It achieved remarkable performance by winning the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [23] and outperforming existing machine learning and computer vision methods. AlexNet's architecture includes five convolutional layers and two fully connected layers, incorporating Local Response Normalisation (LRN) and dropout as new concepts. The initial convolutional layer utilises 96 11×11 filters and is followed by max pooling with 33 filters. Subsequent convolutional layers use 5×5 and 3×3

Distortion levels (99 classes)	**Interval of distortion parameter**	$\mathbf{k}_{\text{mid}}$ (a representation for each level)	$\mathbf{p}_{\text{gt}}$	$\mathbf{p}_{\text{pred}}$
0	1×10^{-6} to 2×10^{-6}	1.5×10^{-6}	0	0.1
1	2×10^{-6} to 3×10^{-6}	2.5×10^{-6}	1	0.8
2	3×10^{-6} to 4×10^{-6}	3.5×10^{-6}	0	0.1
⋮	⋮	⋮	0	0
96	9.7×10^{-5} to 9.8×10^{-5}	9.75×10^{-5}	0	0
97	9.8×10^{-5} to 9.9×10^{-5}	9.85×10^{-5}	0	0
98	9.9×10^{-5} to 1×10^{-4}	9.95×10^{-5}	0	0

Table 3.5: A table showing $\mathbf{k}_{\text{mid}}$ corresponding to the distortion levels and intervals and an example of $\mathbf{p}_{\text{gt}}$ and $\mathbf{p}_{\text{pred}}$

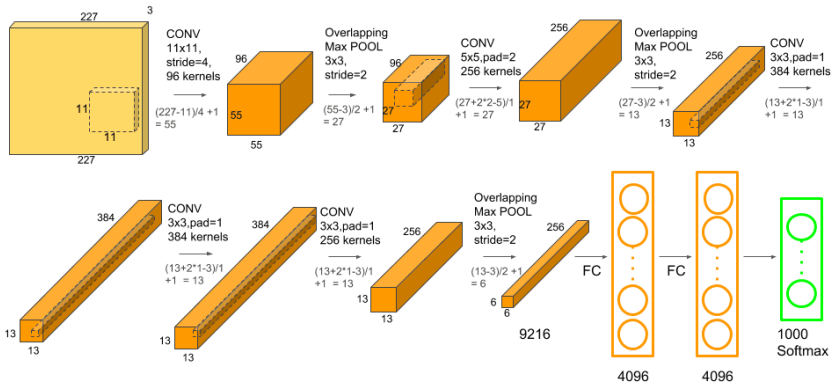

Figure 3.9: The architecture of AlexNet: The size of the final output is not fixed at 1000, as depicted in the figure. Instead, the output size varies and depends on the specific application it is being used for. [25]

filters. The total number of trainable parameters (weights) approximates 61 million [24]. Additionally, the Vision Transformer architecture, previously discussed in Subsection 2.1.4, will also be examined in this project.

In order to improve performance and reduce training time, we will be utilising pre-trained AlexNet and Vision Transformer models for fine-tuning. The pre-trained weights from the `torchvision` [26] package will be employed for this purpose. This package is an integral part of the PyTorch ecosystem and is specifically tailored to support computer vision tasks. It offers a variety of tools and functionalities that streamline the process of working with image data, particularly in the realm of deep learning. Additionally, `torchvision.models` provide pre-trained models for popular neural network architectures such as ResNet, VGG, AlexNet, and MobileNet, which can be directly used for inference or fine-tuned on a custom dataset.

3.5 Evaluation metrics

We will conduct the evaluation in two parts. The first part will assess the performance of the models, while the second part will focus on evaluating the similarity and quality between the original clean image and the rectified image for the rectification program.

In the training schemes for this project, various loss functions are used. These functions are not directly comparable because each scheme aims to learn different aspects. Nevertheless, they all ultimately converge towards the common objective, which is k_{pred}. It's important to highlight that despite the model predicting the distortion level in the classification task (Figure 3.8 (Middle)), we can still calculate k_{pred} using Equation 3.12, as exemplified in Table 3.5. Hence, the metric that can be used to compare models from different schemes is the Relative Error [27], as defined in Equation 3.13. This metric measures the difference between k_{pred} and k_{gt} as a proportion of k_{gt}. The division by k_{gt} normalises the error and allows for contextualising the error relative to the magnitude of k_{gt}.

$$\text{Average Relative Error} = \frac{1}{n'} \sum_{i=1}^{n'} \left(100\% \times \frac{\left| k^i_{\text{pred}} - k^i_{\text{gt}} \right|}{k^i_{\text{gt}}} \right) \tag{3.13}$$

where n' denotes the number of test samples in the test set.

In order to assess the output image from the rectification program, we will utilise SSIM (Structural Similarity Index) and PSNR (Peak Signal-to-Noise Ratio) as done in numerous studies within the field [20], [21]. SSIM, developed by Wang et al. [28], is a metric typically used to measure the similarity between two images. In this case, it is used to quantify the accuracy of the rectified image compared to the original clean image. It evaluates image quality by examining luminance (average intensity) (≡ pixel mean), contrast (≡ variance), and structural similarity (≡ covariance). SSIM values range from 0 to 1, with a value of 1 indicating perfect similarity [29]. SSIM can be expressed as

$$\text{SSIM}(I_r, I_o) = \frac{(2\mu_r\mu_o + c_1)(2\sigma_{ro} + c_2)}{(\mu_r^2 + \mu_o^2 + c_1)(\sigma_r^2 + \sigma_o^2 + c_2)} \tag{3.14}$$

where,

- I_r = the rectified image.
- I_o = the clean image.
- μ_r = the pixel mean of the rectified image.
- μ_o = the pixel mean of the clean image.
- σ_r^2 = the variance of the rectified image.
- σ_o^2 = the variance of the clean image.
- σ_{ro} = the covariance between the rectified and clean image.
- c_1, c_2 = positive constants for preventing having a zero denominator.

PSNR is a widely used metric for assessing the quality difference between original and processed images. In this case, it is used for distorted images and rectified images. PSNR value approaches infinity; therefore, a higher value signifies that the rectified image contains higher-quality content [29]. Typically, a PSNR value of 23 to 26 dB is considered good for the fisheye image rectification [30], [21]. PSNR can be expressed as

$$\text{PSNR} = 10 \cdot \log_{10} \left(\frac{\text{MAX}_{I_o}^2}{\text{MSE}} \right) \quad ; \quad \text{MSE} = \frac{1}{HW} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} [I_o(i,j) - I_r(i,j)]^2 \tag{3.15}$$

where,

- MAX_{I_o} is the maximum possible pixel value of the clean image. For an 8-bit image, MAX_{I_o} is typically 255.
- MSE is the Mean Squared Error calculated as described on the right side of Equation 3.15.
- $I_o(i,j)$ is the pixel value at position (i,j) in the clean image.
- $I_r(i,j)$ is the pixel value at position (i,j) in the rectified image.
- H and W are the dimensions (height and width) of the images.

Chapter 4

Results

In this chapter, the main outcomes of the project are presented. The first section demonstrates the impact of distorted images by incorporating all distortion parameters within a specific range. Subsequently, four experiments were conducted involving three training schemes and two deep learning architectures, with the Average Relative Error serving as the model evaluation metric for comparison and model selection. Additionally, a new unseen-content testing set was utilised to assess the model and rectify distorted images based on the predicted distortion parameters. These rectified images were then evaluated alongside their corresponding clean images using SSIM and PSNR. Lastly, real fisheye images were used to evaluate the project's overall performance.

4.1 The distortion effects for different distortion parameters

In this section, the distortion program will be utilised to distort the clean image (Figure 4.1). Figure 4.2 displays the distorted images for various $k \in [10^{-6}, 10^{-4}]$. It is evident that, within this range, the distortion varies widely from low to high. Therefore, in this project, the range of values for k has been selected accordingly. Additionally, it is worth noting that nearby k values on the same row show similar distorted effects that are difficult to distinguish visually.

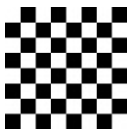

Figure 4.1: A checkered flag image: a good example to demonstrate distortion effects

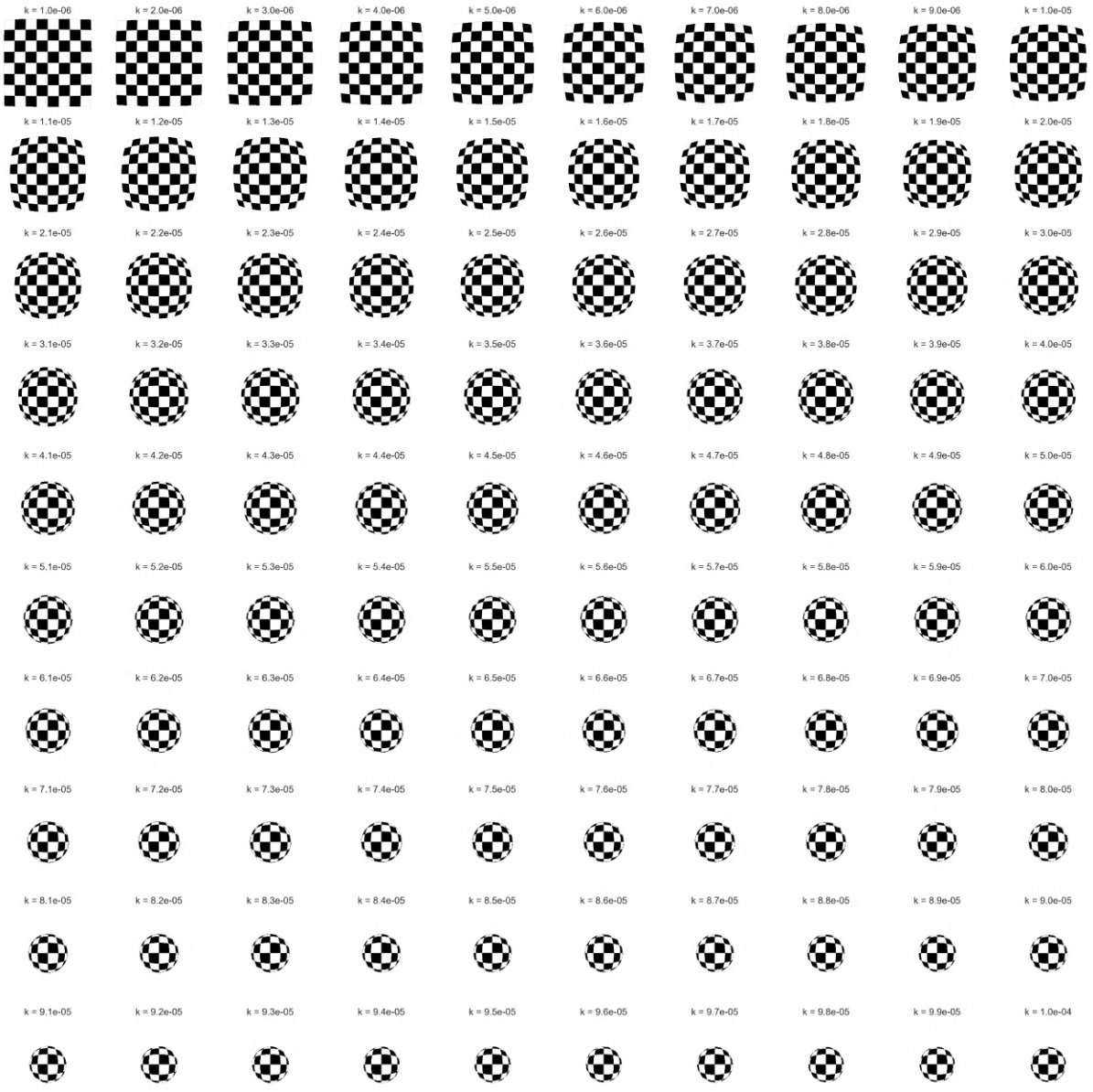

Figure 4.2: The distortion of the checkered flag image when using $k = 10^{-6}$ to 10^{-4}.

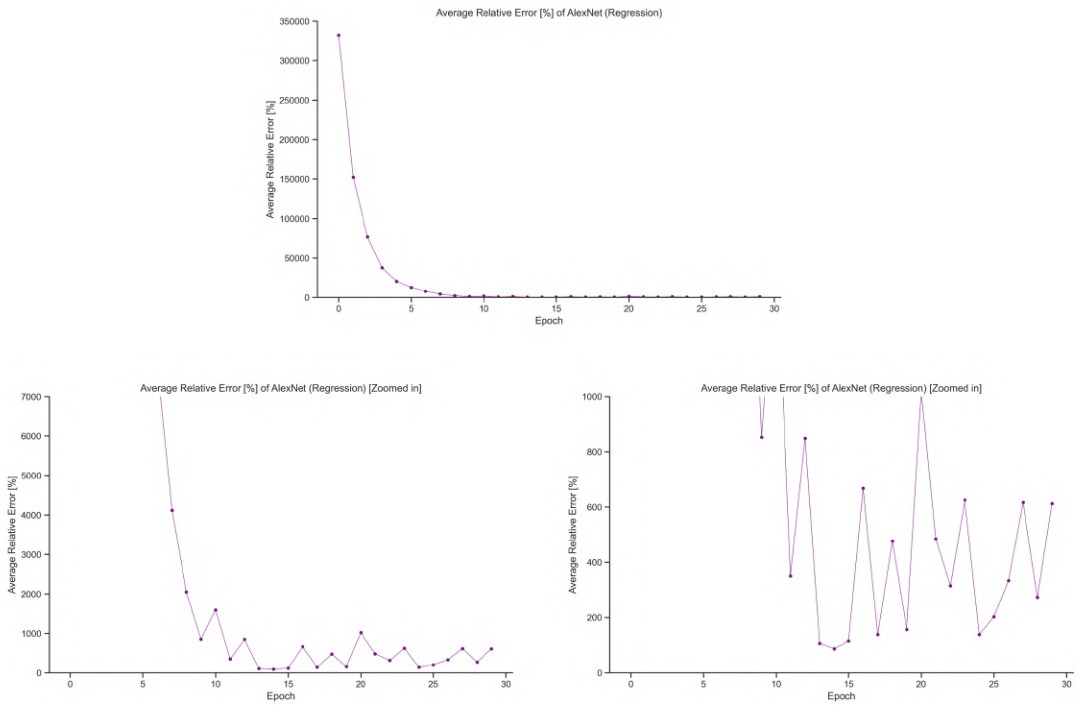

Figure 4.3: Average Relative Error of AlexNet (Regression) for 30 epochs; The two images below are just the zoomed version of the top image.

4.2 The comparison of Average Relative Error of all experiments

In this project, four experiments were conducted involving three different schemes (Regression, Classification, Classification + Regression) and two deep learning architectures (AlexNet, Vision Transformer). The model evaluation metric that is used here is the Average Relative Error, which is computed using the k_{gt} and k_{pred} values obtained from running the model on all testing samples in the **testing set**. For clarification, the complete dataset is split into a training set, a validation set, and a testing set. The training and validation sets are used for training and validating models, and the plot showing the relationship between training loss and validation loss will be examined later on. The testing set is reserved for evaluating (inferring) the models using the best weights obtained from the experiments.

The first experiment focused on training AlexNet using a regression scheme, which is a direct approach for predicting a single real-valued number, k_{pred}, from a distorted image. The loss function for this scheme is a Mean Squared Error (MSE) loss, which calculates the difference between k_{pred} and k_{gt}. However, the Average Relative Error across all 30 training epochs displayed a surprisingly poor result in Figure 4.3. This outcome may be attributed to the small distortion parameter values, which required rescaling and normalisation before training, or using the L1 loss function since the squaring effect of the MSE loss on the distortion parameter leads to significantly smaller values.

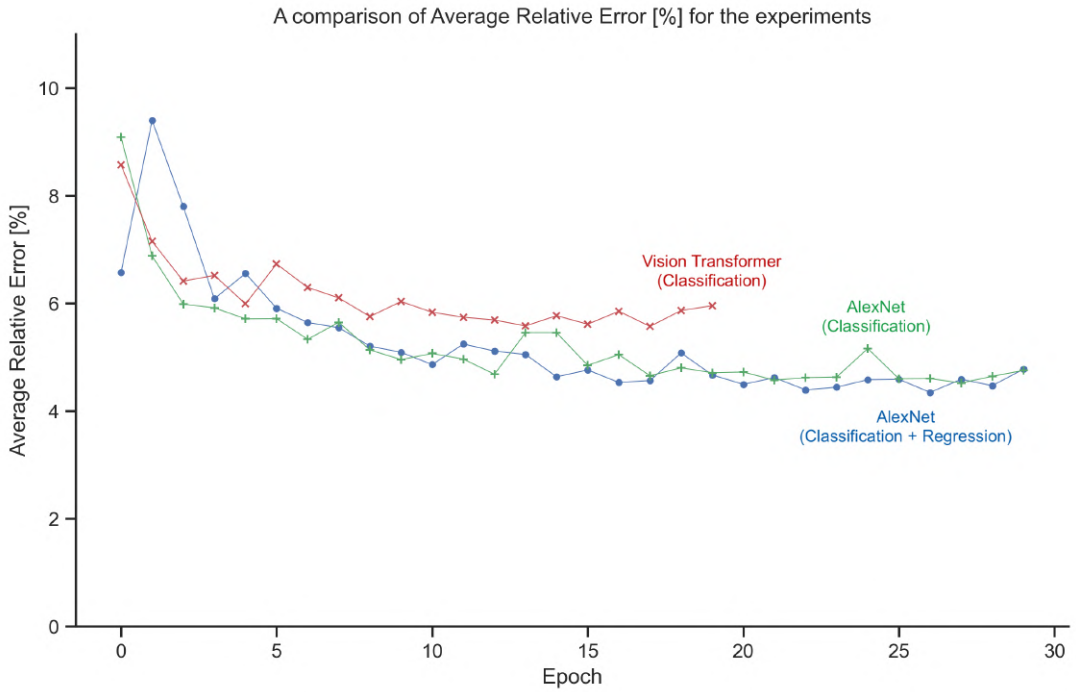

Figure 4.4: Average Relative Error of AlexNet (Classification), AlexNet (Classification + Regression) for 30 epochs and Vision Transformer (Classification) for 20 epochs

Next, we approached the issue as a classification problem, with the model tasked with predicting the distortion level, basically treating it as a class number. This led us to define the distortion levels as outlined in Table 3.4. Although we utilise cross-entropy as the loss function, we can still measure model performance by employing the Average Relative Error as an evaluation metric. This is done by converting the predicted distortion level into k_{pred}, as detailed in Equation 3.12. The results demonstrated a significant improvement compared to the AlexNet (Regression), as depicted in Figure 4.4. Consequently, we also trained the cutting-edge Vision Transformer as a classification scheme with a limited duration of 20 epochs due to time and resource constraints. Figure 4.4 shows that AlexNet (Classification) slightly outperforms Vision Transformer (Classification). This observation coincides with the understanding that Vision Transformer, a data-intensive deep learning architecture, demands substantial training time and data to showcase its optimal performance.

The additional Classification + Regression scheme was also tested, specifically on the AlexNet model. In this scheme, the predicted distortion level is converted into k_{pred}. The idea of this scheme is to combine the essence of the cross-entropy loss (classifying the distortion level) with the rescaled MSE loss (regressing the k_{pred}). It was observed that most MSE loss values from AlexNet were in the order of magnitude of 10^{-11}, while the cross-entropy loss values ranged from 10^{0} to 10^{1}. Because the MSE loss was substantially lower than the cross-entropy loss, it needed to be rescaled to make its impact more meaningful when combined with the cross-entropy loss for effective model training. However, the Average Relative Error for AlexNet (Classification + Regression) was found to be similar to that of AlexNet (Classification), as shown in Figure 4.4.

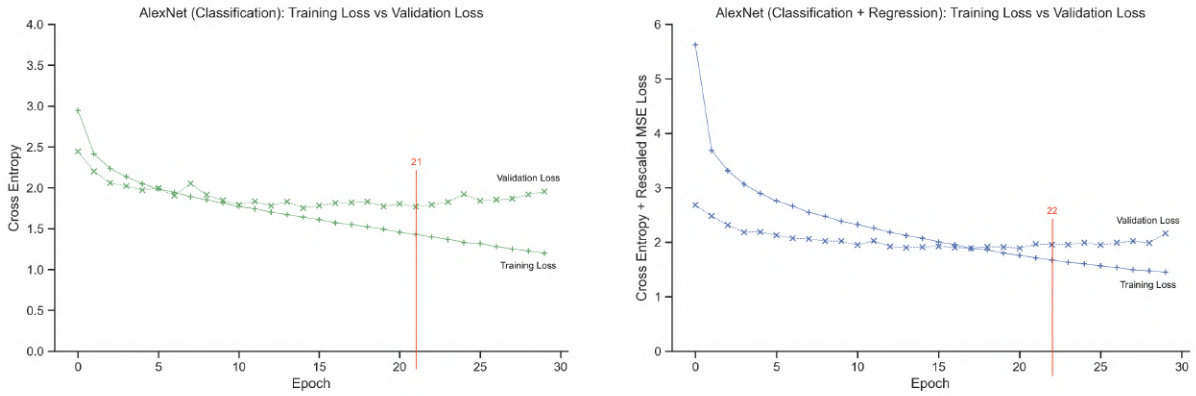

Figure 4.5: [Left]: Training Loss VS Validation Loss of AlexNet (Classification); The best epoch is 21. [Right]: Training Loss VS Validation Loss of AlexNet (Classification + Regression); The best epoch is 22.

Out of all the experiments conducted, both the AlexNet (Classification) and AlexNet (Classification + Regression) models performed quite well and produced very similar results. To make the best use of the weights for future steps, we need to select the optimal epoch for both models. This involves identifying the epoch with the lowest Average Relative Error, ensuring that it is not underfitted or overfitted. After careful consideration of the Average Relative Error depicted in Figure 4.4 and the training versus validation loss shown in Figure 4.5, we have chosen to utilise the weights from epoch 21 for AlexNet (Classification) and epoch 22 for AlexNet (Classification + Regression). The corresponding best Average Relative Errors are documented in Table 4.1.

Model	**Best epoch**	**Average Relative Error** (of the model at that epoch)
AlexNet (Classification)	21	4.57%
AlexNet (Classification + Regression)	22	4.39%

Table 4.1: A table showing Average Relative Error of the model each at the best epoch

4.3 The unseen-content testing set

We divided the entire dataset into training, validation, and testing sets. All previous experiments up to this point were conducted using this original testing set (crosswalk images). As each experiment aimed to predict k_{pred}, each k_{gt} for every sample was randomly selected, meaning that the models had yet to see this k_{gt} before (which is a general fact for deep learning training). However, due to the synthesis (distortion) of each image at 99 levels, the model may have been exposed to the content of some images, even during the testing phase. Table 4.2 displays the results of testing (evaluating/inferring) the models on the new testing set, which was synthesised from the unseen-content images (train station platform images). When assessing

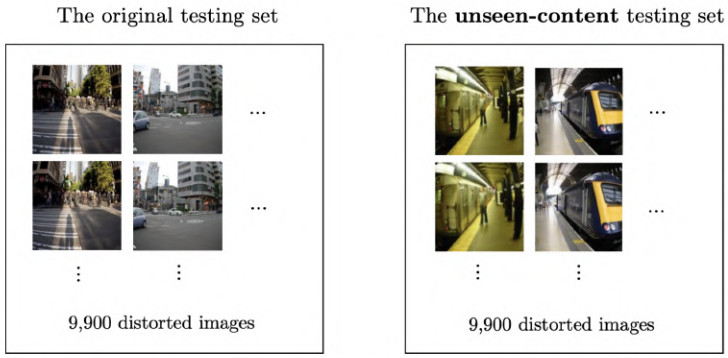

Figure 4.6: An example of distorted images in the original testing set (left) and the new test set containing distorted images from unseen-content images (right).

the models on the unseen-content testing set, it was observed that the Average Relative Error from this table is approximately 13%, which may initially seem high. However, it is important to note that a 13% error is still considered acceptable in this context. For instance, in the event that the model has predicted $k_{\text{pred}} = 1 \times 10^{-5}$, with a Relative Error of 13%, the possible values for k_{gt} can be calculated as

$$\begin{aligned} \frac{|k_{\text{pred}} - k_{\text{gt}}|}{k_{\text{gt}}} &= 0.13 \\ k_{\text{gt}} &= k_{\text{pred}}/(1 \pm 0.13) \\ k_{\text{gt}} &= 0.88 \times 10^{-5}, \quad 1.15 \times 10^{-5} \end{aligned} \tag{4.1}$$

As illustrated in Figure 4.7, it is evident that for $k = 1 \times 10^{-5}$, 0.88×10^{-5}, or 1.15×10^{-5}, the distorted images appear similar and are difficult for the eyes to distinguish. Thus, despite an Average Relative Error of approximately 13%, we can still regard this as a precise prediction.

Model	**Average Relative Error** (testing on the original testing set)	**Average Relative Error** (testing on the unseen-content testing set)
AlexNet (Classification)	4.57%	13.17%
AlexNet (Classification + Regression)	4.39%	13.85%

Table 4.2: A table showing the Average Relative Error for AlexNet (Classification) and AlexNet (Classification + Regression) when testing on the original testing set and the new unseen-content testing set.

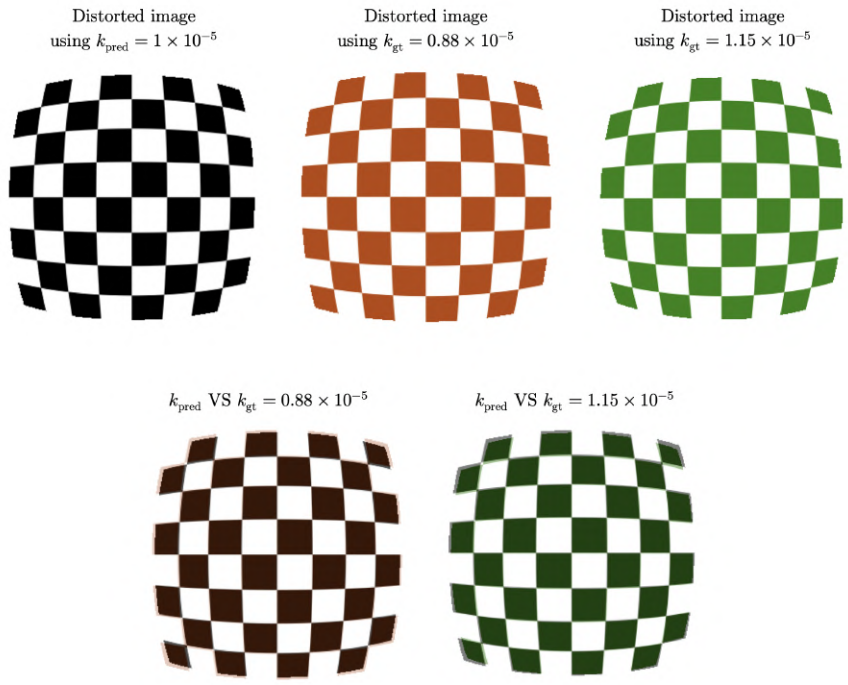

Figure 4.7: The top row displays the distorted images using different k. The bottom row illustrates plotting each rectified image (using k_{gt}) overlay on the rectified image (using k_{pred}).

4.4 Rectifying and Image evaluation

We have obtained the weights for the best models: AlexNet (Classification) and AlexNet (Classification + Regression), which are prepared to analyse distorted images and predict k_{pred}. In this section, we will present the outcomes of using k_{pred} to create a rectified image from the corresponding distorted image and evaluate the similarity and quality of the rectified and clean images. Additionally, we will also assess both the rectified and clean images using k_{gt} for rectification instead of k_{pred}. This entire process can be visualised in Figure 4.8. Furthermore, the results are summarised in Table 4.3, demonstrating that our models have done an outstanding job in predicting an accurate k_{pred} for rectification, as the SSIM and PSNR values computed on the clean image and the rectified image generated by k_{pred} are closely aligned with those generated by k_{gt}. Additionally, we have included some good cases in Figure 4.9, along with their corresponding quantitative results in Table 4.4.

4.5 Rectifying the real fisheye image

Figure 4.10 displays some real fisheye images selected from [16] alongside the corresponding rectified images and the k_{pred}. The previous experimentation involved using samples in a specific format during the dataset synthesis step using a distortion and rectification program. Consequently, for non-synthesised, real fisheye images, the results may not be accurate and cannot be compared quantitatively. In essence, real fisheye images were not synthesised with any specific

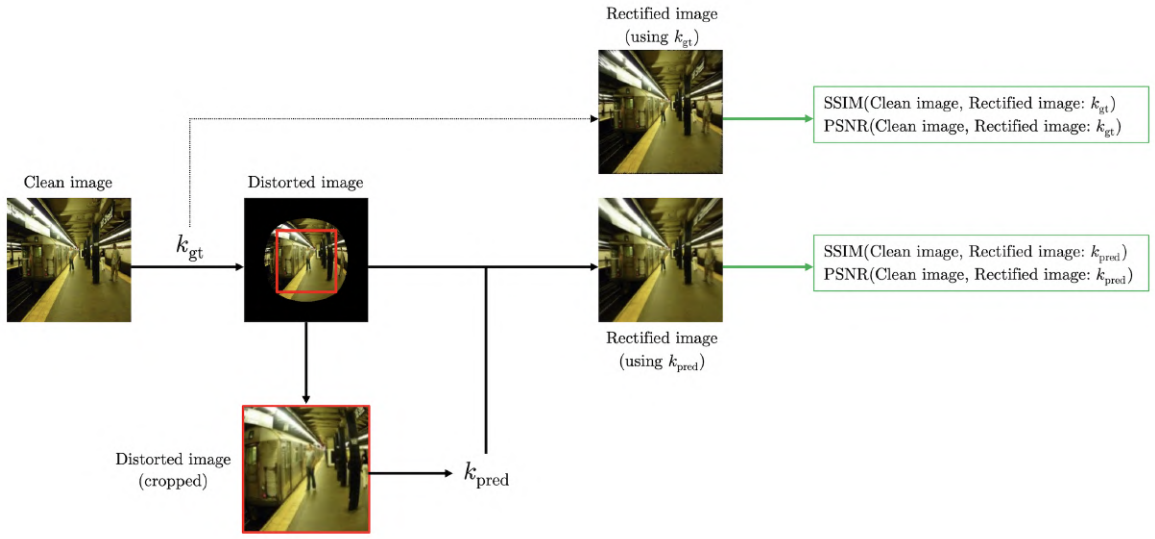

Figure 4.8: A rectifying and image evaluating diagram: First, the clean image is distorted by k_{gt}, which is then cropped. Subsequently, a (trained) model predicts k_{pred}. The rectified image is generated using the distorted image and k_{pred}. Furthermore, k_{gt} is also used to create the rectified image for comparison. Finally, the quality of the output images is evaluated using SSIM and PSNR. It's important to note that this diagram illustrates the process for a single sample. To assess all samples in the testing set, we will calculate the Average SSIM and Average PSNR.

k value and did not have original clean images. As a result, Relative Error, SSIM, or PSNR cannot be utilised for comparison. Thus, we attempted to rectify each real fisheye image using various k values, as depicted in Figure 4.11 to Figure 4.13. Considering Figure 4.11 to Figure 4.13, all the k that is used to create the best-rectified images for the real fisheye images 1, 2, and 3 should be within $[1.1 \times 10^{-5}, 2.1 \times 10^{-5}]$, which is approximate to each k_{pred} that was predicted from the model.

Typically, when we distort the image with a certain k_{gt} value, black pixels surround the distorted image, as shown in the distorted image column in Figure 4.9, allowing the rectified image's pixels to remain within the boundary during rectification. However, as real fisheye images were not synthesised in this manner, the rectified images are cropped. In Algorithm 1, we need base coordinates to rectify the image and the size of the clean image to create base coordinates. Here, the size of the clean image is assumed to be the same as those in the dataset, i.e., 256×256, regarding that a clean image does not exist for a real fisheye image.

Figure 4.9: Five clean images, chosen from the unseen-content testing set, were utilised to perform distortion, prediction, and rectification processes sequentially, yielding outstanding outcomes.

	Real fisheye image	Rectified image	k_{pred}
Real fisheye image 1			1.57×10^{-5}
Real fisheye image 2			1.26×10^{-5}
Real fisheye image 3			1.54×10^{-5}

Figure 4.10: Each real fisheye image and the corresponding rectified image using each k_{pred}

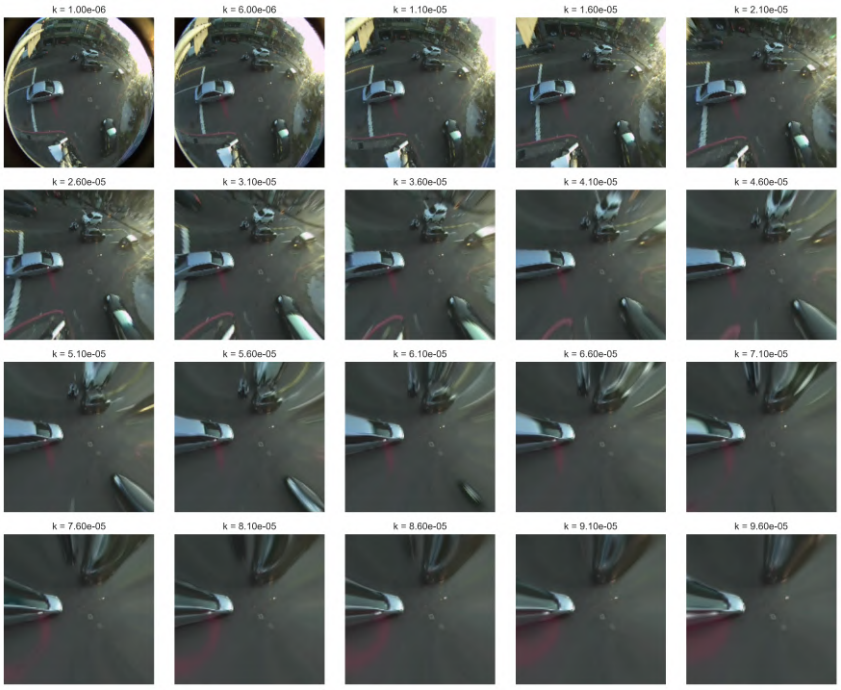

Figure 4.11: The rectified images from the real fisheye image 1 using various k; the best-rectified image is the one that used the $k \in [1.1 \times 10^{-5}, 2.1 \times 10^{-5}]$.

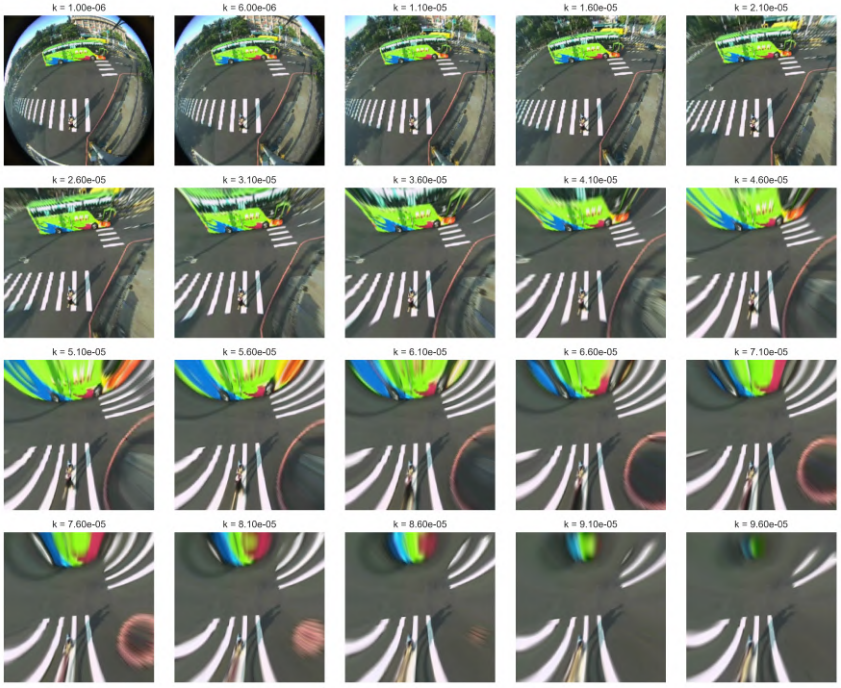

Figure 4.12: The rectified images from the real fisheye image 2 using various k; the best-rectified image is the one that used the $k \in [1.1 \times 10^{-5}, 2.1 \times 10^{-5}]$.

Rectify using...	**Model**	**Max SSIM**	**Min SSIM**	**Average SSIM**	**Max PSNR** [dB]	**Min PSNR** [dB]	**Average PSNR** [dB]
k_{pred}	AlexNet (Class.)	0.93	0.10	0.46	35.44	27.96	29.41
	AlexNet (Class.+Reg.)	0.92	0.08	0.45	35.47	28.00	29.39
k_{gt}	-	0.93	0.22	0.60	36.21	28.00	29.94

Table 4.3: The maximum, minimum, and average SSIM and PSNR computed on the clean and rectified images using k_{pred} and k_{gt}. These results were obtained using the unseen-content testing set.

Image	k_{gt}	k_{pred}	**Relative Error**	**SSIM** (k_{gt})	**SSIM** (k_{pred})	**PSNR** [dB] (k_{gt})	**PSNR** [dB] (k_{pred})
Image 1	7.50×10^{-6}	7.48×10^{-6}	0.20%	0.93	0.93	33.35	33.36
Image 2	1.70×10^{-5}	1.76×10^{-5}	3.55%	0.95	0.90	34.98	33.74
Image 3	3.87×10^{-6}	3.75×10^{-6}	3.07%	0.94	0.92	33.65	33.07
Image 4	1.08×10^{-5}	1.09×10^{-5}	0.88%	0.95	0.93	35.87	35.04
Image 5	4.64×10^{-6}	5.23×10^{-6}	12.70%	0.94	0.90	33.97	33.11

Table 4.4: The quantitative results for five images in Figure 4.9. These results include k_{gt}, k_{pred}, the Relative Error between k_{gt} and k_{pred}, SSIM, and PSNR values computed on both the clean and rectified images using k_{pred} and k_{gt}.

Figure 4.13: The rectified images from the real fisheye image 3 using various k; the best-rectified image is the one that used the $k \in [1.1 \times 10^{-5}, 2.1 \times 10^{-5}]$.

Chapter 5

Discussion

This section will cover some interesting results, highlight areas that need improvement, and outline project limitations and future improvement guidelines.

5.1 Comparing with the other paper

In the Research gap subsection, it was noted that comparing the results with most papers following a similar approach is challenging due to the lack of detailed information for replication. However, [3] has made their model weights and code available for replication (despite using a different approach). A comparison of the rectified image results (from the unseen-content testing set) from my project and theirs is presented in Figure 5.1. Although they utilised evaluation metrics different from the ones used in this project, we still compared the images using SSIM and PSNR for consistency, as depicted in Table 5.1. The results of my work outperform those of [3] for the five sample images in terms of quality and quantity. This issue arises because a piece of their program remains unreleased, as indicated in [31]. Therefore, we are unable to achieve optimal results. A comparison of the rectified image results (from the real fisheye images) from my project and theirs is also presented in Figure 5.2.

Image	**SSIM** (my works)	**SSIM** ([3])	**PSNR** [dB] (my works)	**PSNR** [dB] ([3])
Image 1	0.93	0.80	33.36	32.11
Image 2	0.90	0.49	33.74	30.32
Image 3	0.92	0.70	33.07	30.70
Image 4	0.93	0.79	35.04	34.17
Image 5	0.90	0.70	33.11	31.47

Table 5.1: The quantitative results for five images in Figure 4.9. These results include SSIM, and PSNR values computed on both the clean and rectified images using my works and [3].

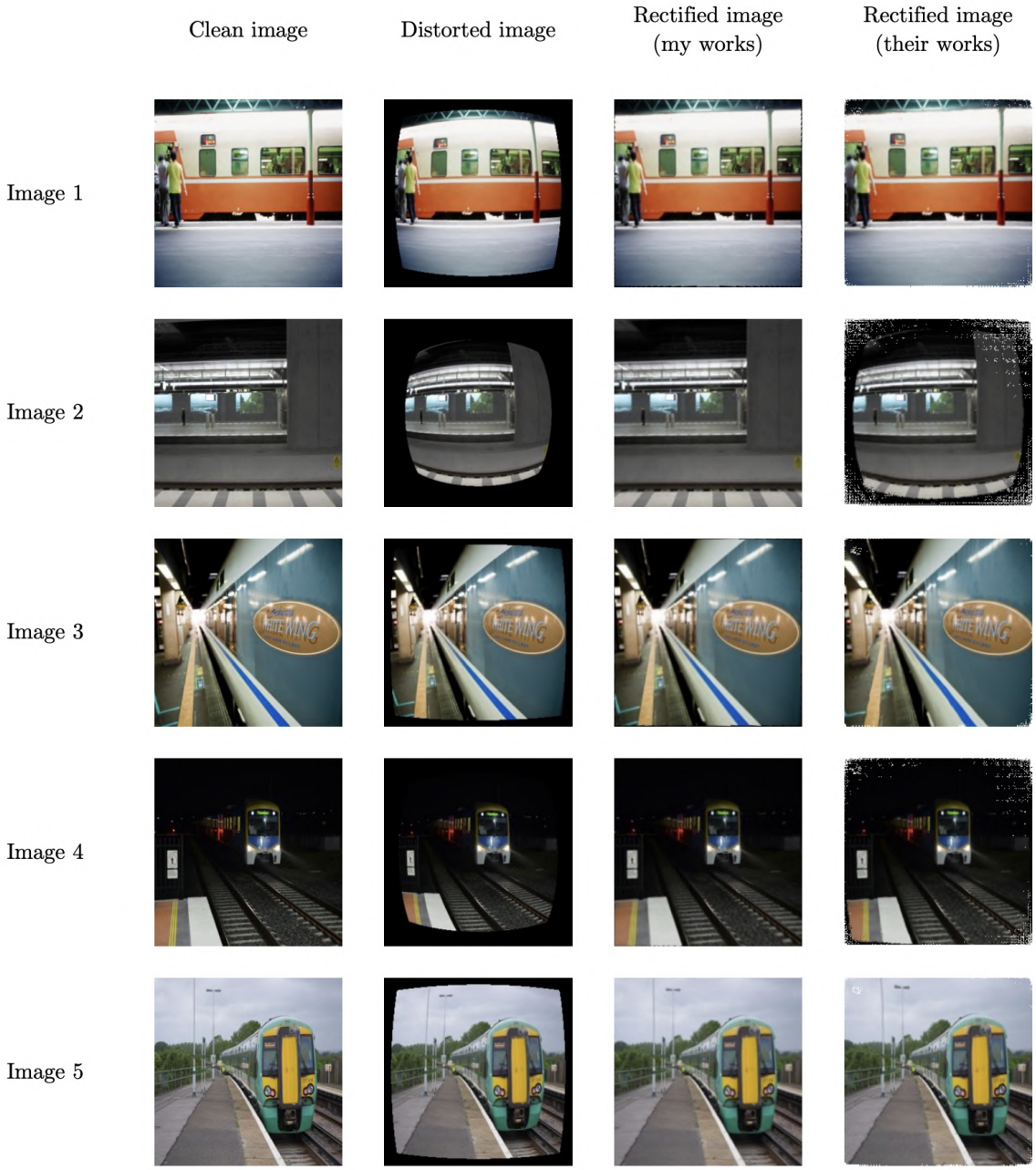

Figure 5.1: A comparison of the rectified images (from the unseen-content testing set) of my works and [3]

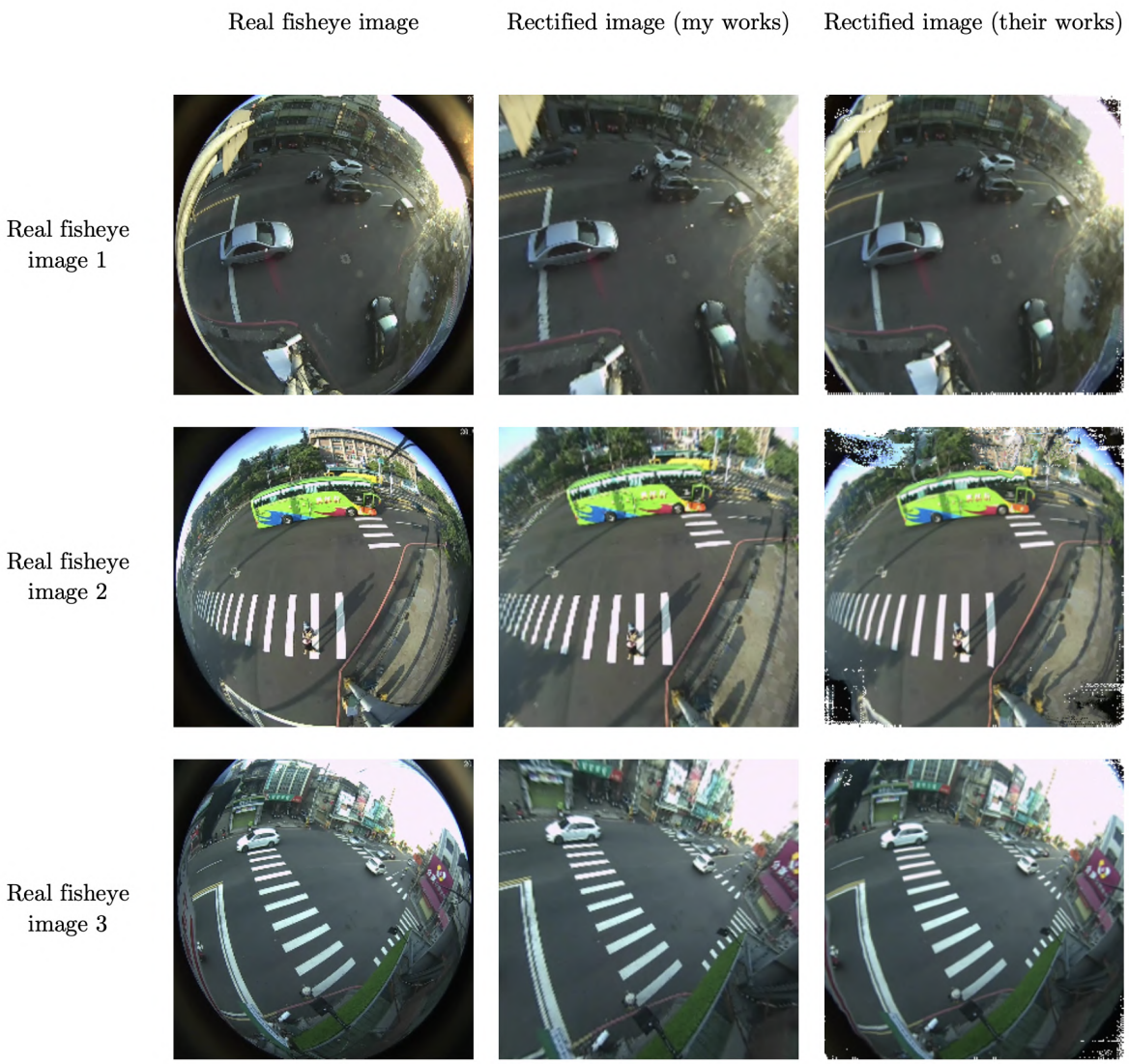

Figure 5.2: A comparison of the rectified images (from the real fisheye images) of my works and [3]

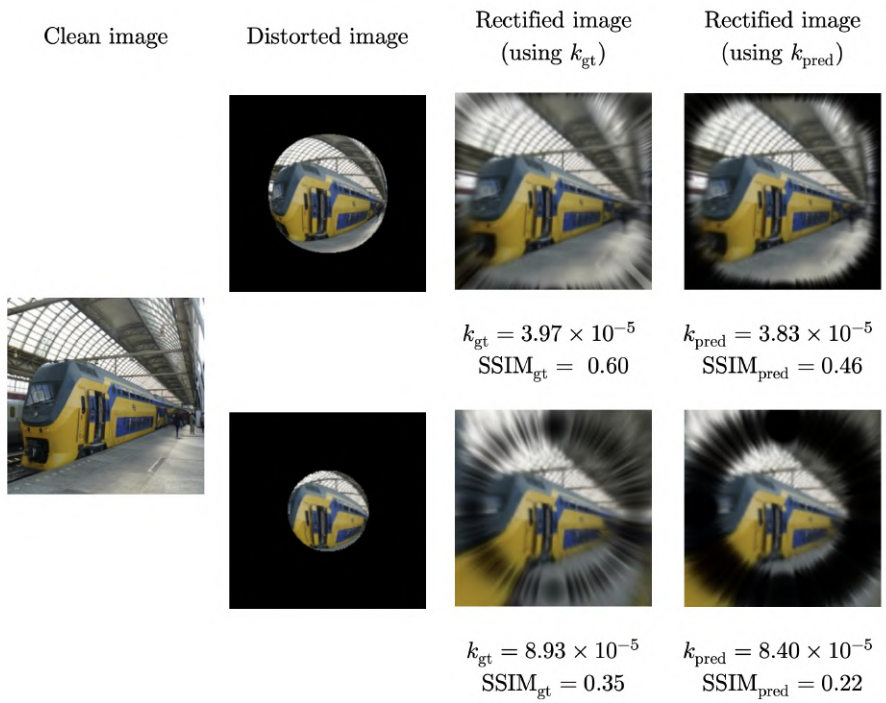

Figure 5.3: Distorted images using $k = 3.97 \times 10^{-5}$ (Top row) and $k = 8.93 \times 10^{-5}$ (Bottom row) and the rectified images using k_{gt} and k_{pred}; $SSIM_{gt}$ and $SSIM_{pred}$ are also provided for all rectified images.

5.2 Future works

5.2.1 Selecting a better range of k

In Table 4.3, the average SSIM calculated on the clean and rectified images using k_{gt}, denoted as $SSIM_{gt}$, is 0.60. It is noticeable that $SSIM_{gt}$ is less than 0.60 when k_{gt} is greater than or equal to 3.8×10^{-5}. Upon examining the distorted images when k_{gt} is greater than or equal to 3.8×10^{-5} in Figure 4.2, it becomes evident that they share a very similar appearance and suffer from reduced quality caused by higher distortion. Figure 5.3 depicts the impact of high distortion on a clean image, visually showing a significant loss of information, particularly in the corner. Therefore, this leads to decreased quality of the rectified image even when rectified using k_{gt}. Thus, we can claim that $k \in [1 \times 10^{-6}, 3.8 \times 10^{-5}]$ is sufficient to produce well-rectified results that address all the distortions in fisheye images.

5.2.2 Performing image selection before synthesising a dataset

In [18], they chose specific images for their dataset based on the number of line segments in the images. Line segments are essential for helping models recognise and correct distortion, so the selected images should have many line segments. This approach could also benefit my project in creating a more robust dataset. They utilised the Hough transform to detect shapes,

particularly lines, in the images. They introduced a selection factor, denoted as α, to determine which images to incorporate into the dataset. This factor is defined as the ratio of the average length of the longest line segments: l_{mean} to the average image size: $(H + W)/2$, expressed as Equation 5.1. By adjusting the value of α, they could control the strictness of their selection criteria—greater α values resulted in the selection of images with more line segments.

$$l_{\text{mean}} \geqslant \alpha \cdot \frac{(H + W)}{2} \tag{5.1}$$

5.2.3 Using other distortion models

From the first row of Equation 3.6, we can write the division model as: (the proof is provided in Equation B.9)

$$r_c = \frac{r_d}{1 + kr_d^2} \tag{5.2}$$

To make the effect of fisheye images more realistic, we can add more distortion parameters to the mathematical model like this: [32]

$$r_c = \frac{r_d}{1 + \sum_{i=1}^{n} k_i r_d^{2i-1}}; \quad n = 1, 2, 3, \ldots \tag{5.3}$$

There are also additional distortion models to explore, such as the polynomial model, expressed as: [32]

$$r_c = \sum_{i=1}^{n} k_i r_d^{2i-1}; \quad n = 1, 2, 3, \ldots \tag{5.4}$$

The multiple distortion parameters present a challenge, requiring the development of new methods for synthesising and predicting them. Additionally, the distortion and rectification program cannot currently be used, as its inverse form involves solving quadratic equations (limited to the second-order degree polynomial) (refer to Equation 3.9). Meanwhile, Equation 5.3 and Equation 5.4 contain higher-order polynomial terms for r_d, making it more difficult to determine their inverses.

5.2.4 Redesigning the image rectification program

In the rectification program results for rectifying real fisheye images, it is indicated that we require base coordinates to rectify the image and the size of the clean image to generate base coordinates. The assumed size of the clean image is 256×256 before utilising the program to rectify the real fisheye images. This assumption is based on the program's specific design

for distorting and rectifying the dataset in a particular format for training purposes. It is important to note that the rectification program for real fisheye images should not rely on the size of the clean images, as they do not exist for real fisheye images. Consequently, the rectification program will need to be revised and edited in the future.

5.2.5 Mean Absolute Error (MAE) loss function

A viable adjustment is switching from the MSE loss function for training the regression task to using the MAE loss function, as outlined in the left side of Equation 5.5. The MSE loss function (the right side of Equation 5.5), being quadratic, can substantially decrease when applied to a minimal error value, potentially causing issues during model training and leading to unsatisfactory results, as depicted in Figure 4.3. On the contrary, the MAE loss function, also referred to as L1 loss, is linear, and using it may enhance the model's performance.

$$\text{MAE Loss} = \frac{1}{n}\sum_{i=1}^{n}\left|k^i_{\text{pred}} - k^i_{\text{gt}}\right| \quad ; \quad \text{MSE Loss} = \frac{1}{n}\sum_{i=1}^{n}(k^i_{\text{pred}} - k^i_{\text{gt}})^2 \tag{5.5}$$

5.2.6 More computational resources

For optimal results, it is crucial to adjust the training process by increasing the volume of training data and the number of epochs when working with the Vision Transformer. This model requires substantial data and consumes significant computational resources and time to produce satisfactory results. Therefore, it is suggested that the model be trained using a large amount of data and provided with more training time.

5.3 Project difficulties

1. Replicating existing solutions has proven to be challenging due to the lack of released weights or code. However, this project effectively addressed this issue and achieved most of its goals.

2. There are multiple experiments to be conducted, all of which involve training deep learning models. Given the limited GPU resources provided by the school, it has been difficult to conduct all the necessary experiments. To mitigate this challenge, the school's GPUs are utilised for model training while using my laptop to develop the distortion and rectification code. This strategy significantly saved time.

3. The paper [21] discusses the direct rectified image generation approach. In the initial stages of the project, substantial time was spent attempting to train this model. However, the results were unsatisfactory, and certain parts of the model could not be executed. To

address this challenge, the decision was made to abandon this approach, shifting the main focus to the distortion parameter prediction approach instead.

4. The distortion and rectification program, inspired by [19], did not initially align with the project framework and remained unexplained. After debugging and testing the code, it became necessary to provide an explanation for the mathematical principles behind this code, as no such explanation had been published. Proving the mathematics and adjusting the code to align with the proof took some time, but it was done thoroughly.

Chapter 6

Conclusions

In summary, this project seeks to correct real fisheye images by training deep learning models to predict the distortion parameters of the fisheye model before using the rectification program to produce the corrected output images. Initially, an exploration was made into understanding how fisheye distortion is represented in computer vision and existing rectification solutions, including traditional geometry-based and deep learning-based methods. This project focuses on the latter, which can be divided into direct rectified image generation and distortion parameters prediction approaches. Despite both deep learning approaches having their drawbacks, the distortion parameters prediction approach was opted for due to time and resource constraints inherent in the direct rectified image generation approach.

The framework of this project was organised into three main parts, with inspiration drawn from [18]. This included dataset synthesis, model training and validation, and rectifying and image evaluation. The image distortion and rectification code was edited based on [19], with detailed explanations of the mathematical operations involved. Subsequently, four experiments were carried out, encompassing three training schemes and two deep-learning architectures.

The experiments involved testing three different schemes and two deep-learning architectures, and it was found that the AlexNet model performed the best when using the Classification and Classification + Regression schemes, achieving the lowest Average Relative Error of 4.57% and 4.39%, respectively. When the models were tested on a new dataset with unseen content, the Average Relative Error increased to around 13%, which is acceptable because similar k will generate similar distorted images. The error of 13% can be neglected and considered as precise in this context. Additionally, the models were employed for the real fisheye images and achieved satisfying results. Finally, the project's results were also compared with those of [3], and it was found that our results outperformed those of [3] regarding both image quality and quantitative metrics.

Bibliography

[1] V. H. Duong, D. Q. Nguyen, T. Van Luong, H. Vu, and T. C. Nguyen, Robust data augmentation and ensemble method for object detection in fisheye camera images, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jun. 2024, pp. 7017–7026.

[2] Y. Qian, M. Yang, X. Zhao, C. Wang, and B. Wang, Oriented Spatial Transformer Network for Pedestrian Detection Using Fish-Eye Camera, *IEEE Transactions on Multimedia*, vol. 22, no. 2, pp. 421–431, 2020. DOI: 10.1109/TMM.2019.2929949.

[3] X. Li, B. Zhang, P. V. Sander, and J. Liao, Blind geometric distortion correction on images through deep learning, in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 4850–4859. DOI: 10.1109/CVPR.2019.00499.

[4] A. Fitzgibbon, Simultaneous linear estimation of multiple view geometry and lens distortion, in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990465.

[5] R. Pramoditha, The Concept of Artificial Neurons (Perceptrons) in Neural Networks, https://towardsdatascience.com/the-concept-of-artificial-neurons-perceptrons-in-neural-networks-fab22249cbfc, Accessed: 2024-08-11, 2021.

[6] F.-F. Li and E. Adeli, CS231n Convolutional Neural Networks for Visual Recognition, https://cs231n.github.io/convolutional-networks/, Accessed: 2024-08-11, 2024.

[7] R. Pramoditha, Overview of a Neural Network's Learning Process, https://medium.com/data-science-365/overview-of-a-neural-networks-learning-process-61690a502fa, Accessed: 2024-08-11, 2022.

[8] N. Shahriar, What is Convolutional Neural Network — CNN (Deep Learning), https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5, Accessed: 2024-08-11, 2023.

[9] M. Liani, DNND 2: Tensors and Convolution, https://cs231n.github.io/convolutional-networks/, Accessed: 2024-08-11, 2023.

[10] A. Anwar, What is Transposed Convolutional Layer? https://cs231n.github.io/convolutional-networks/, Accessed: 2024-08-11, 2020.

[11] S. Khosla, CNN — Introduction to Pooling Layer, https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/, Accessed: 2024-08-11, 2023.

[12] A. Dosovitskiy *et al.*, An image is worth 16x16 words: Transformers for image recognition at scale, 2021. arXiv: 2010.11929 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2010.11929.

[13] T. Moustakas and K. Kolomvatsos, Homogeneous transfer learning for supporting pervasive edge applications, *Evolving Systems*, vol. 15, pp. 1–17, Nov. 2023. DOI: 10.1007/s12530-023-09548-3.

[14] MathWorks, What Is Overfitting?, https://www.mathworks.com/discovery/overfitting.html, Accessed: 2024-08-11.

[15] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, Places: A 10 Million Image Database for Scene Recognition, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, no. 6, pp. 1452–1464, 2018. DOI: 10.1109/TPAMI.2017.2723009.

[16] M. Gochoo *et al.*, FishEye8K: A Benchmark and Dataset for Fisheye Camera Object Detection, in *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2023, pp. 5305–5313. DOI: 10.1109/CVPRW59228.2023.00559.

[17] M. Alemán-Flores, L. Alvarez, L. Deniz, and D. Santana-Cedrés, Automatic lens distortion correction using one-parameter division models, *Image Processing On Line*, vol. 4, Nov. 2014. DOI: 10.5201/ipol.2014.106.

[18] J. Rong, S. Huang, Z. Shang, and X. Ying, Radial lens distortion correction using convolutional neural networks trained with synthesized images, in *Computer Vision – ACCV 2016*, S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato, Eds., Cham: Springer International Publishing, 2017, pp. 35–49, ISBN: 978-3-319-54187-7.

[19] Reinderien, Fast Implementation of the 1 Parameter Division Distortion Model, https://stackoverflow.com/questions/77889635/fast-implementation-of-the-1-parameter-division-distortion-model, Accessed: 2024-07-28, 2024.

[20] J. Fan, J. Zhang, S. J. Maybank, and D. Tao, Wide-angle image rectification: A survey, 2021. arXiv: 2011.12108 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2011.12108.

[21] Z. Liao, W. Zhou, and H. Li, DaFIR: Distortion-Aware Representation Learning for Fisheye Image Rectification, *IEEE Transactions on Circuits and Systems for Video Technology*, vol. PP, pp. 1–1, Jan. 2023. DOI: 10.1109/TCSVT.2023.3315967.

[22] A. Krizhevsky, I. Sutskever, and G. Hinton, Imagenet classification with deep convolutional neural networks, *Neural Information Processing Systems*, vol. 25, Jan. 2012. DOI: 10.1145/3065386.

[23] O. Russakovsky *et al.*, Imagenet large scale visual recognition challenge, 2015. arXiv: 1409.0575 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1409.0575.

[24] M. Z. Alom *et al.*, The history began from alexnet: A comprehensive survey on deep learning approaches, 2018. arXiv: 1803.01164 [cs.CV]. [Online]. Available: https://arxiv.org/abs/1803.01164.

[25] A. Pujara, Concept of AlexNet:- Convolutional Neural Network, https://medium.com/analytics-vidhya/concept-of-alexnet-convolutional-neural-network-6e73b4f9ee30, Accessed: 2024-08-11, 2020.

[26] T. maintainers and contributors, *Torchvision: Pytorch's computer vision library*, https://github.com/pytorch/vision, 2016.

[27] F. Wu, H. Wei, and X. Wang, Correction of image radial distortion based on division model, *Optical Engineering*, vol. 56, p. 013 108, Jan. 2017. DOI: 10.1117/1.OE.56.1.013108.

[28] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, Image quality assessment: From error visibility to structural similarity, *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[29] A. Horé and D. Ziou, Image Quality Metrics: PSNR vs. SSIM, in *2010 20th International Conference on Pattern Recognition*, 2010, pp. 2366–2369. DOI: 10.1109/ICPR.2010.579.

[30] Z. Liao, H. Feng, S. Liu, W. Zhou, and H. Li, Rofir: Robust fisheye image rectification framework impervious to optical center deviation, 2024. arXiv: 2406.18927 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2406.18927.

[31] X. Li, GeoProj, https://github.com/xiaoyu258/GeoProj/issues/18, Accessed: 2024-08-02, 2023.

[32] S. Yang, C. Lin, K. Liao, C. Zhang, and Y. Zhao, Progressively complementary network for fisheye image rectification using appearance flow, in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 6344–6353. DOI: 10.1109/CVPR46437.2021.00628.

Appendices

Appendix A

GitLab Repository

The implementation codes for this project have been uploaded to the University of Birmingham's GitLab. You can access them through the following URL: https://git.cs.bham.ac.uk/projects-2023-24/kxs1035.git. In the GitLab repository, there are the Inference.zip file, vision-main.zip file, and all of the .ipynb files.

The .ipynb files are the main Jupyter notebooks used to run the project's results. The numbers at the beginning of each file name correspond to the different stages of the project: 1) Dataset synthesis, 2) Model training and validation, and 3) Image rectification and evaluation. Additionally, 4) and 5) are for inference purposes only. However, you cannot execute any code without the accompanying folder named "vision-main" (also provided), which should be located in the same directory as these .ipynb files. Therefore, the complete FisheyeRectification.zip file (30GB) (Figure A.1) containing the dataset, .csv files, weights for all epochs for the four experiments, and the compared result from paper [3] (located in the GeoProj folder) will be uploaded in this OneDrive link.

However, if you only want to infer the trained model to predict the distortion parameter and rectify the sample synthesised and real fisheye images, simply extract the Inference.zip only (475MB) (Figure A.2). You can run each cell sequentially to view the results or modify the input cell to change your distorted/real fisheye image and obtain the desired results. It's important to note that all input RGB images must be sized 256×256.

Furthermore, all of the code will be revised and updated in the future on my personal GitHub repository, which is accessible here: https://github.com/TK-KXS.

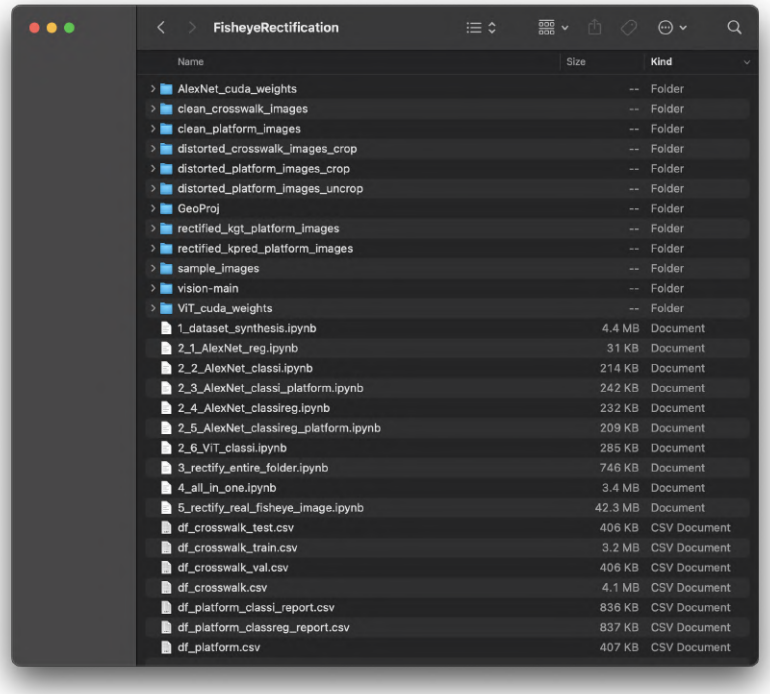

Figure A.1: The extraction of FisheyeRectification.zip containing the complete implementation of this project

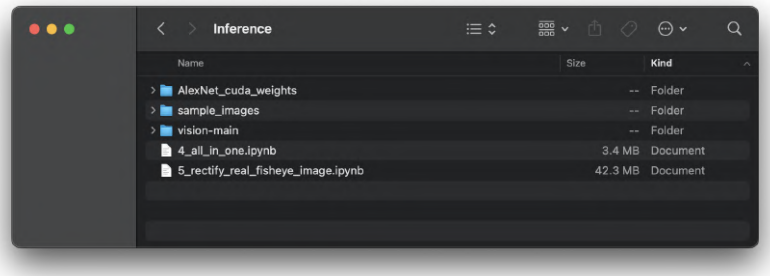

Figure A.2: The extraction of Inference.zip containing only the inference part of this project

Appendix B

Additional mathematical derivation

B.1 The proof of obtaining Equation 3.9 from Equation 3.7 and Equation 3.8

For clarity, let us consider only the x-axis as described in Equation 3.7. As shown, the value of x_d has two solutions:

$$x_d = x_m + \frac{(x_c - x_m)\left[1 + \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} \quad \text{or} \quad x_m + \frac{(x_c - x_m)\left[1 - \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2}, \tag{B.1}$$

where $r_c^2 = (x_c - x_m)^2 + (y_c - y_m)^2$.

Our objective is to ensure that $r_d^2 \to 0$ as $r_c^2 \to 0$. In other words, we require that $(x_d, y_d) \to (x_m, y_m)$ as $(x_c, y_c) \to (x_m, y_m)$. We now need to determine which of the two solutions ensures that $x_d \to x_m$.

Consider the first solution from Equation B.1:

$$\lim_{(x_c,y_c)\to(x_m,y_m)} \frac{(x_c - x_m)\left[1 + \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} = \dots$$

$$\begin{aligned}
\dots &= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{(x_c - x_m)\left[1 + \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} \cdot \frac{\left[1 - \sqrt{1 - 4kr_c^2}\right]}{\left[1 - \sqrt{1 - 4kr_c^2}\right]} \\
&= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{(x_c - x_m)\left[1 - (1 - 4kr_c^2)\right]}{2kr_c^2\left[1 - \sqrt{1 - 4kr_c^2}\right]} \\
&= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{4kr_c^2(x_c - x_m)}{2kr_c^2\left[1 - \sqrt{1 - 4kr_c^2}\right]} \\
&= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{2(x_c - x_m)}{1 - \sqrt{1 - 4kr_c^2}} \\
&:= L
\end{aligned} \tag{B.2}$$

Using the binomial approximation $(1+\lambda)^n \approx 1+n\lambda$ when $|\lambda| < 1$ and $|n\lambda| \ll 1$, and considering that $r_c^2 \to 0$, we can approximate $\sqrt{1 - 4kr_c^2} \approx 1 - 2kr_c^2$. Substituting this into the expression for L, we get:

$$\begin{aligned}
L &= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{2(x_c - x_m)}{1 - (1 - 2kr_c^2)} \\
&= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{x_c - x_m}{kr_c^2} \\
&= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{x_c - x_m}{k\left[(x_c - x_m)^2 + (y_c - y_m)^2\right]}
\end{aligned} \tag{B.3}$$

Since $k \neq 0$ and the denominator is a quadratic term while the numerator is linear, the denominator approaches zero more rapidly than the numerator. Consequently, the limit evaluates to:

$$\lim_{(x_c,y_c)\to(x_m,y_m)} \frac{x_c - x_m}{k\left[(x_c - x_m)^2 + (y_c - y_m)^2\right]} = \infty, \tag{B.4}$$

which indicates that this is not the solution we are seeking.

Now, let us examine the second solution from Equation B.1:

$$\lim_{(x_c,y_c)\to(x_m,y_m)} \frac{(x_c - x_m)\left[1 - \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} = \dots$$

$$\begin{aligned}
\dots &= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{(x_c - x_m)\left[1 - \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} \cdot \frac{\left[1 + \sqrt{1 - 4kr_c^2}\right]}{\left[1 + \sqrt{1 - 4kr_c^2}\right]} \\
&= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{(x_c - x_m)\left[1 - (1 - 4kr_c^2)\right]}{2kr_c^2\left[1 + \sqrt{1 - 4kr_c^2}\right]} \\
&= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{4kr_c^2(x_c - x_m)}{2kr_c^2\left[1 + \sqrt{1 - 4kr_c^2}\right]} \\
&= \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{2(x_c - x_m)}{2} \\
&= 0
\end{aligned} \tag{B.5}$$

Thus, the second solution of Equation B.1 is the one we must choose because:

$$\begin{aligned}
\lim_{(x_c,y_c)\to(x_m,y_m)} x_d = x_m \quad &\Longleftrightarrow \quad \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{(x_c - x_m)\left[1 - \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} = 0 \\
\lim_{(x_c,y_c)\to(x_m,y_m)} y_d = y_m \quad &\Longleftrightarrow \quad \lim_{(x_c,y_c)\to(x_m,y_m)} \frac{(y_c - y_m)\left[1 - \sqrt{1 - 4kr_c^2}\right]}{2kr_c^2} = 0
\end{aligned} \tag{B.6}$$

QED

B.2 Obtaining Equation 3.9 using a vector approach

As depicted in Figure 2.1, the problem can be formulated in a 2D vector space by translating all coordinates—referred to as vectors from now—by x_m along the x-axis and y_m along the y-axis. This translation effectively shifts the center of distortion to the origin $[0, 0]^\top$, while the corrected points (x_c, y_c) and the distorted points (x_d, y_d) are now represented as vectors $[x_c - x_m, y_c - y_m]^\top$ and $[x_d - x_m, y_d - y_m]^\top$, respectively.

To determine the distorted vector $\mathbf{d} = [x_d - x_m, y_d - y_m]^\top$ given the corrected vector $\mathbf{c} = [x_c - x_m, y_c - y_m]^\top$, we use the fact that all distorted vectors share the same direction as their corresponding corrected vectors but differ in magnitude. Our task, therefore, is to compute the direction of the unit corrected vector $\widehat{\mathbf{d}}$ and the magnitude of the distorted vector $\|\mathbf{d}\|$.

To find $\widehat{\mathbf{d}}$, we simply compute $\widehat{\mathbf{c}}$, as the distorted vector will have the same direction as the corresponding corrected vector. Thus, $\widehat{\mathbf{d}}$ can be computed as:

$$\widehat{\mathbf{d}} = \widehat{\mathbf{c}} = \frac{\mathbf{c}}{\|\mathbf{c}\|} = \frac{1}{\|\mathbf{c}\|}\begin{bmatrix} x_c - x_m \\ y_c - y_m \end{bmatrix} \tag{B.7}$$

Next, to find $\|\mathbf{d}\|$, we use Equation 3.6, where $r_d = \|\mathbf{d}\|$ and $r_c = \|\mathbf{c}\|$, here. This can be rewritten as:

$$\|\mathbf{d}\|^2 = (1 + k\|\mathbf{d}\|^2)^2\|\mathbf{c}\|^2 \tag{B.8}$$

Given that $\|\mathbf{d}\|, \|\mathbf{c}\|, (1 + k\|\mathbf{d}\|^2) \geqslant 0$, we obtain from Equation B.8:

$$\|\mathbf{d}\| = (1 + k\|\mathbf{d}\|^2)\|\mathbf{c}\| \tag{B.9}$$

Thus, $\|\mathbf{d}\|$ can be computed as:

$$\begin{aligned} \|\mathbf{d}\| &= (1 + k\|\mathbf{d}\|^2)\|\mathbf{c}\| \\ k\|\mathbf{c}\|\|\mathbf{d}\|^2 - \|\mathbf{d}\| + \|\mathbf{c}\| &= 0 \\ \|\mathbf{d}\| &= \frac{1 - \sqrt{1 - 4k\|\mathbf{c}\|^2}}{2k\|\mathbf{c}\|} \end{aligned} \tag{B.10}$$

where we use the proof in Section B.1 for selecting only one solution from Equation B.10.

Finally, $\mathbf{d}$ can be computed as:

$$\begin{aligned} \mathbf{d} &= \|\mathbf{d}\|\widehat{\mathbf{d}} \\ &= \frac{1 - \sqrt{1 - 4k\|\mathbf{c}\|^2}}{2k\|\mathbf{c}\|} \cdot \frac{1}{\|\mathbf{c}\|}\begin{bmatrix} x_c - x_m \\ y_c - y_m \end{bmatrix} \\ &= \frac{1 - \sqrt{1 - 4k\|\mathbf{c}\|^2}}{2k\|\mathbf{c}\|^2}\begin{bmatrix} x_c - x_m \\ y_c - y_m \end{bmatrix} \end{aligned} \tag{B.11}$$

Substituting back $\mathbf{d} = [x_d - x_m, y_d - y_m]^\top$ and $\|\mathbf{c}\| = r_c$, we obtain:

$$\begin{bmatrix} x_d - x_m \\ y_d - y_m \end{bmatrix} = \frac{1 - \sqrt{1 - 4kr_c^2}}{2kr_c^2}\begin{bmatrix} x_c - x_m \\ y_c - y_m \end{bmatrix} \tag{B.12}$$

which is the same result as Equation 3.9.

QED

B.3 Obtaining Equation 3.9 by solving the system of circle equations

Starting from the Equation 3.1 and Equation 3.2, we initially assume that the center of the distortion (x_m, y_m) is at the origin, i.e., $(x_m, y_m) = (0, 0)$. This assumption simplifies the equations to the following form:

$$x_c = \frac{x_d}{1 + k\left(x_d^2 + y_d^2\right)} \quad \longrightarrow \quad x_d^2 + y_d^2 - \left(\frac{1}{kx_c}\right) x_d + \frac{1}{k} = 0 \tag{B.13}$$

and

$$y_c = \frac{y_d}{1 + k\left(x_d^2 + y_d^2\right)} \quad \longrightarrow \quad x_d^2 + y_d^2 - \left(\frac{1}{ky_c}\right) y_d + \frac{1}{k} = 0 \tag{B.14}$$

Equation B.13 and Equation B.14 represent the general form of circle equations. To find the coordinates (x_d, y_d), we need to solve this system of equations.

To find the intersection points of two circles given by their general equations, we start with the following system:

$$x_d^2 + y_d^2 + D_1 x_d + E_1 y_d + F_1 = 0 \quad \text{(Circle 1)} \tag{B.15}$$
$$x_d^2 + y_d^2 + D_2 x_d + E_2 y_d + F_2 = 0 \quad \text{(Circle 2)} \tag{B.16}$$

Here, the constants (D_1, E_1, F_1) and (D_2, E_2, F_2) define the positions and sizes of the two circles.

Step 1: Subtract the Equations

Subtract Equation B.16 from Equation B.15 to eliminate the quadratic terms x_d^2 and y_d^2:

$$(D_1 - D_2)x_d + (E_1 - E_2)y_d + (F_1 - F_2) = 0 \tag{B.17}$$

This simplifies to the linear equation:

$$Ax_d + By_d + C = 0 \tag{B.18}$$

where $A = D_1 - D_2$, $B = E_1 - E_2$, and $C = F_1 - F_2$.

Step 2: Solve the Linear Equation for y_d

Solve the linear equation for y_d:

$$y_d = \frac{-Ax_d - C}{B} \quad (; B \neq 0) \tag{B.19}$$

Step 3: Substitute into One of the Circle Equations

Substitute the expression for y_d back into one of the original circle equations. For instance, substituting $y_d = \frac{-Ax_d - C}{B}$ into Equation B.15 results in:

$$\begin{gathered} x_d^2 + \left(\frac{-Ax_d - C}{B}\right)^2 + D_1 x_d + E_1 \left(\frac{-Ax_d - C}{B}\right) + F_1 = 0 \\ \left(\frac{A^2}{B^2} + 1\right) x_d^2 + \left(\frac{2AC}{B^2} - \frac{E_1 A}{B} + D_1\right) x_d + \left(\frac{C^2}{B^2} - \frac{E_1 C}{B} + F_1\right) = 0 \end{gathered} \tag{B.20}$$

This equation is quadratic in x_d and can be solved using the quadratic formula.

Substituting the known values from Equation B.13 and Equation B.14 into Equation B.20, we get:

$D_1 = -\frac{1}{kx_c}$
$E_1 = 0$
$F_1 = \frac{1}{k}$
$D_2 = 0$
$E_2 = -\frac{1}{ky_c}$
$F_2 = \frac{1}{k}$
$A = -\frac{1}{kx_c}$
$B = \frac{1}{ky_c}$
$C = 0$

Substituting these variables into Equation B.20, we obtain:

$$\begin{aligned}
\left(\frac{y^2}{x^2}+1\right)x_d^2 - \frac{1}{kx_c}x_d + \frac{1}{k} &= 0 \\
x_d &= \frac{\frac{1}{kx_c} - \sqrt{\frac{1}{k^2x_c^2} - 4\left(\frac{y^2}{x^2}+1\right)\frac{1}{k}}}{2\left(\frac{y^2}{x^2}+1\right)} \\
x_d &= \frac{\frac{1}{kx_c} - \sqrt{\frac{1-4k(x_c^2+y_c^2)}{k^2x_c^2}}}{2\left(\frac{x_c^2+y_c^2}{x_c^2}\right)} \\
x_d &= x_c \cdot \frac{1-\sqrt{1-4k(x_c^2+y_c^2)}}{2k(x_c^2+y_c^2)}
\end{aligned} \tag{B.21}$$

We select the appropriate solution for x_d using the criteria discussed in Section B.1.

To find y_d, we employ a similar approach. Substituting $x_d = \frac{-By_d - C}{A}; A \neq 0$ into Equation B.15 yields:

$$y_d = y_c \cdot \frac{1-\sqrt{1-4k(x_c^2+y_c^2)}}{2k(x_c^2+y_c^2)} \tag{B.22}$$

Finally, by substituting

x_d with $x_d - x_m$,
x_c with $x_c - x_m$,
y_d with $y_d - y_m$,
y_c with $y_c - y_m$, and
$r_c^2 = (x_c - x_m)^2 + (y_c - y_m)^2$

we obtain:

$$\begin{bmatrix} x_d - x_m \\ y_d - y_m \end{bmatrix} = \frac{1-\sqrt{1-4kr_c^2}}{2kr_c^2} \begin{bmatrix} x_c - x_m \\ y_c - y_m \end{bmatrix} \tag{B.23}$$

This result is consistent with Equation 3.9.

QED