University of Kent | School of Engineering

RECONSTRUCTION OF BURNER FLAMES THROUGH DEEP LEARNING

A Thesis submitted to the University of Kent
for the Degree of Master of Science by Research
in Electronic Engineering

Word count: 24,000

By

BAMIDELE OGUNJUMELO BEng

September 2022

Abstract

This MSc thesis reports the design, implementation, and experimental evaluation of a deep learning-based system for the three-dimensional (3-D) reconstruction and visualisation of fossil-fired burner flames. A literature review is given to examine all existing techniques for 3-D visualisation and characterisation of flames. Methodologies and techniques for the 3-D reconstruction of burner flames using optical tomographic and deep learning (DL) techniques are presented, together with a discussion of their advantages and limitations in their applications. Technical requirements and existing problems of the reviewed techniques are discussed.

A technical strategy, incorporating numerical simulations, DL, digital image processing and optical tomographic techniques is proposed for the reconstruction and visualisation of a flame. Based on this strategy, a 3-D flame reconstruction and visualisation system based on DL is developed. The system consists of a trained convolutional neural network (CNN) based network model and the use of a third-party software tool for visualisation. The system can use flame images acquired concurrently from eight different directions of a burner and perform a 3-D reconstruction of the flame. A numerical simulation is performed initially to examine the suitability of the DL algorithm proposed, ground truth data are generated using a mathematical model designed to mimic a flame structure and 2-D projection data are generated from each ground truth. A modified CNN model with a 1-D output dense layer is established and trained for the reconstruction of the 3-D Gaussian distribution. To determine the optimal network model architecture for this solution, various experiments were conducted using different network model parameters. A detailed description of a CNN-based network implemented for the numerical solutions is presented.

A series of experiments was conducted using flame data obtained from a laboratory-scale combustion test rig to evaluate the performance of the established CNN model. These included implementing code to perform image processing routines to prepare the dataset collected from the laboratory-scale combustion test rig. Additional datasets were also generated using OpenCV morphological transformation operations to augment the original dataset. The obtained results have proven that the implemented and trained CNN network model can reconstruct the cross-sectional slices of a burner flame based on the images obtained under various combustion conditions. It was also possible to obtain a 3-D flame structure from the reconstructed cross-sectional flame data using a 3-D visualisation tool. Results from the experiments and the performance of the implemented 3-D flame reconstruction and visualisation system based on DL are presented and discussed.

Acknowledgements

The author would specifically like to express thanks for the following:

University of Kent

Dr Md Moinul Hossain	My first supervisor, for his invaluable advice, encouragement and support made it possible for me to complete this project.
Dr Gang Lu	My second supervisor, for his support, advice and encouragement which enable me to continually make satisfactory progress.

My family members in particular my father and mother for all the support put into my education.

This thesis is dedicated to my wife Damilola Ogunjumelo and my sons Oladele and Olatoni Ogunjumelo, for their encouragement, help, support, and patience throughout the project.

Content

List of Table

List of Figures

Nomenclature

Symbol	Meaning
A	Dense layer output
B	Bias
C_1, C_2	In the SSIM function, C1 and C2 are constants to ensure stability when the denominator becomes zero.
f(x ,y)	2-D cross-sectional attenuation distribution function
F (u, v)	2-D Fourier transform
F	This coefficient represents the contribution of pixel j to the projection bin i
MAX_I	Maximum possible pixel value in the evaluated image
N	Total number of data points
P (r, θ)	One-dimensional (1-D) X-ray projections
P	Projection measurement
$\mathbf{p}(\mathbf{x}; \mu, \sum)$	Multivariate Gaussian distribution
S	Input matrix
$\mathbf{SSIM}(\mathbf{x}, \mathbf{y})$	Structural similarity function for image x and y
μ	n-dimensional mean vector
$\boldsymbol{\mu_x}$	Average pixel value of image x
$\boldsymbol{\mu_y}$	Average pixel value of image y
X	Input image pixel values
$\boldsymbol{x}$	x is an n-dimension vector
y_i	Actual value
$\hat{y}$	Predicted value
$\varnothing$	Activation function
$\boldsymbol{\sigma_x}$	Variance of image x
$\boldsymbol{\sigma_y}$	Variance of image y
$\boldsymbol{\sigma_{xy}}$	Covariance of image x and image y

List of Abbreviations

Abbreviation	Meaning
1-D	One-Dimensional
2-D	Two-Dimensional
3-D	Three-Dimensional
AI	Artificial intelligence
API	Application Programming Interface
ART	Algebraic Reconstruction Technique
CNN	Convolutional Neural network
CT	Computed Tomography
CCD	Charge-coupled Device
CFD	Computational Fluid Dynamics
CTC	Computed Tomography of Chemiluminescence
CPU	Central Processing Unit
DBN	Deep Belief Network
DL	Deep Learning
FBP	Filtered Back-Projection
FC	Fully Connected
GPU	Graphics Processing Unit
LFBP	Logical Filtered Back-Projection
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MART	Multiplicative Algebraic Reconstruction Technique
MLEM	Maximum Likelihood-Expectation Maximisation
MCP	McCulloch and Pitts
ML	Machine Learning
MSE	Mean Squared Error

MRI	Magnetic Resonance Imaging
NPU	Neural Processing Unit
OST	Optical Sectioning Tomography
OpenCV	Open-Source Computer Vision Library
PIV	Particle Image Velocimetry
PLIF	Planar Laser Induced Fluorescence
POD	Proper Orthogonal Decomposition
PSNR	Peak Signal-To-Noise Ratio
RAS	Right, Anterior, Superior
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RGB	Red Green Blue
RMSE	Root Mean Squared Error
SART	Simultaneous algebraic reconstruction technique
SIRT	Simultaneous iterative reconstruction technique
SSIM	Structural similarity index Metric
TL	Transfer Learning
TPU	Tensor Processing Unit
VT	Volumetric Tomography
VT-Net	Volumetric Tomography Network Model

1

The Importance of Three-Dimensional Reconstruction of Burner Flames

1.1 Introduction

Combustion systems are widely used globally in many industries to generate electricity and thermal energy by burning fossil fuels. For these systems to operate safely, optimal operating conditions need to be met. Furthermore, industries are now being required to reduce pollutant emissions and maximize combustion efficiency due to increasingly tighter government regulations. Combustion monitoring and diagnosis now play a vital role in the control and optimisation of combustion processes in industrial combustion systems such as boilers and gas turbines.

In a combustion process, a flame is the central reaction zone. Flame characteristics can be represented by several physical parameters including temperature, oscillation frequency, size, shape, brightness, and uniformity. By analysing these parameters, it is possible to assess the quality and efficiency of the combustion process. Therefore, monitoring, visualizing, and characterization of combustion flames has become increasingly important for a deeper understanding of combustion conditions.

With the advances in digital imaging and computing technology, digital imaging-based volumetric tomography (VT) has attracted great attention in combustion research. VT is a powerful technique for combustion diagnostics due to its capacity to visualize flame structures in three-dimension (3-D). The unique features of VT include non-intrusiveness and easy implementation, which has made such an optical imaging technique suitable for the spatial and temporal monitoring and diagnostics of combustion systems.

Traditional iterative methods such as the algebraic reconstruction technique (ART) and simultaneous iterative reconstruction technique (SIRT) have been used successfully in the 3-D reconstruction of burner flames. However, the use of these methods has some limitations such as high computational cost and restricting 3-D reconstruction to be conducted offline.

In recent years, 3-D reconstruction of burner flames through deep learning (DL) has attracted increasing interest and demonstrated an impressive performance in terms of reconstruction accuracy and computational efficiency in comparison with traditional methods.

This thesis investigates how DL techniques based on a CNN model can be utilized in reconstructing volumetric slices of a burner flame. A convolutional neural network (CNN) is used to perform 3-D volumetric reconstruction after training a CNN model. Reconstruction is accomplished using ground truth images obtained from both a numerical simulation and real flame images as training data.

1.2 Research Questions

The thesis seeks to answer the following research questions:

- How can the 3-D reconstruction speed and performance of a CNN network model-based DL 3-D rapid flame monitoring solution be improved?

- Can training data with abundant features of a flame structure be prepared to help improve the reconstruction accuracy of a flame?

- How does the performance of DL-reconstructed flames compare with those reconstructed using traditional iterative solutions?

- How does the proposed DL-based 3-D flame reconstruction system perform under different combustion conditions?

- How can the data from the output of the DL network model be used in visualising the reconstructed 3D flame structure in real-time?

1.3 Technical Challenges in 3-D Reconstruction of Burner Flames Using Deep Learning

The development of a CNN-based network model to perform 3-D volumetric reconstruction of a burner flame faces several challenges. The main technical challenges that have been identified are as follows:

- The system should be capable of performing online 3-D reconstruction using DL methods. The common approach is to perform reconstruction offline, the capability of the CNN model to provide reconstructed images in real-time with good accuracy is one of the crucial challenges.
- The generation of an adequate number of 2-D flame projection data under various combustion conditions. For the numerical simulation, some algorithms are required to generate the ground truth data. The development of suitable algorithms could be challenging for generating data for various flame conditions.
- The implementation of a framework or tools capable of prepossessing a vast amount of image data. For the training and validation of CNN models, large datasets of image data must be processed.
- The development of algorithms to extract and analyse performance metrics obtained from the CNN model during the training and validation phase. This also includes the performance evaluation of the reconstructed images.
- The training of a CNN model using a vast amount of data is very time-consuming. The use of a high-performance computer is required to reduce the training time.

1.4 Aims and Objectives

The research programme aims to develop and train a DL model for the 3-D volumetric reconstruction of burner flames. The DL model implemented will be able to reconstruct the cross-sectional slices of a burner flame based on the images obtained under various combustion conditions.

The objectives of the research programme were defined as follows:

- To carry out a comprehensive literature review related to this field.
- To establish a CNN DL model for the 3-D volumetric reconstruction of a burner flame.
- To test and validate the model through numerical simulations.
- To implement framework and workflow for creating experimental flame datasets for training DL network models.
- To carry out experiments under different combustion operation conditions.
- To reconstruct flame cross-sections and longitudinal sections using real flame image data.
- To perform 3-D volumetric reconstruction using real flame image data.
- To implement a solution for visualising the 3-D flame structure using data from the output of the established and trained DL network model.

1.5 Summary of Contributions

- 3-D flame reconstruction using DL is introduced to the computer vision community with a review of related previous work in combustion, DL, and computer vision.
- A system has been developed to generate experimental datasets of flame images based on numerical simulations.

- A CNN network model has been implemented to improve the speed and performance of DL-based 3D rapid flame monitoring.

- An evaluation of the effectiveness of DL in reconstructing 3-D flames has been conducted using flame data obtained from a laboratory-scale combustion test rig.

- A machine learning (ML) hardware accelerator has been used to obtain performance results.

1.6 Thesis Structure

The structure of this thesis is organised and presented in a logical way that contributes to the set of objectives. The major contributions of this thesis include the development of a DL model for the 3-D volumetric reconstruction of a burner flame, the evaluation and validation of the system under various operation conditions and the evaluation of the model using a ML hardware accelerator. All other relevant contents that are related to this research programme, including literature review, conclusions and suggestions for future work are also addressed.

The thesis is organised into six chapters and a summary of each chapter is depicted as follows:

- **Chapter 1** includes the importance of 3-D flame reconstructions of burner flames. It also covers the aim, objectives, and technical challenges of this research programme.

- **Chapter 2** presents a comprehensive literature survey on the techniques available for 3-D reconstruction using DL and traditional methods. Techniques that have previously been proposed and developed for this application are reviewed.

- **Chapter 3** provides background information for understanding the topic. To begin with, DL terms will be introduced, followed by concepts and definitions related to CNN. The proposed CNN model is then discussed in detail, including its design and implementation. Next, the process used for training the CNN model is described.

Following that is a description of the ML framework and performance metrics that were used in this study.

- **Chapter 4** presents a DL 3-D reconstruction experimental setup based on using data generated from a numerical simulation. Then follows the presentation of the results and the evaluation of the accuracy of the implemented network model.
- **Chapter 5** describes the experimental setup for the 3-D reconstruction of burner flames using a DL network model. Then follows the presentation of the results and the evaluation of the accuracy of the implemented network model.
- **Chapter 6** Summarises the findings and conclusions that have been drawn from this research. This also includes recommendations for future work.

2
Literature review

2.1 Introduction

A comprehensive literature survey has been conducted to review previously published research work associated with the subject of 3-D flame visualisation using traditional methods and employing deep learning techniques. This is to ascertain the extent of previous work in the field. The state-of-the-art in the subject area has been defined, following the analysis, and understanding of all relevant reference materials obtained. The review also ensures the originality of the proposed research programme, in addition to acquiring any useful background knowledge that may contribute to the work.

Recently, there has been an increasing need for reducing pollutants emanating from a low-efficient combustion process. Combustion monitoring and diagnosis techniques play a very key role in the management and control of combustion systems in industrial environments [1-3]. These non-intrusive optical techniques have contributed to the implementation of combustion diagnostic solutions for managing air pollution and the combustion efficiency of combustion systems [4].

Also, over the past couple of years, numerous studies have been conducted to investigate imaging techniques for 3-D flame monitoring and visualisation. These have included the development of various 2-D instrumentation systems for monitoring and characterisation of flames. However, these systems have some limitations. They can only visualize a flame from one direction and would not be suitable for providing enough information to properly characterise a flame for measurement purposes. Due to this limitation, 2-D imaging techniques are not included in this review.

Furthermore, another key area of flame studies is mathematical modelling which is also known as CFD (Computational Fluid Dynamics). To understand the fundamentals of flames and combustion processes, it is particularly useful to study the research in these areas.

However, they are beyond the scope of the research programme and therefore excluded from this review.

A comprehensive review of existing techniques, including deflection photography, light scattering, particle image velocimetry (PIV) and direct photography, etc. for the 3-D visualisation and characterisation of flames, was previously conducted in [5-7]. This chapter only covers available tomographic techniques for the 3-D reconstruction and characterisation of flames. This is where a flame tomographic process involves the acquisition of the 2-D image projection and the reconstruction of flame cross-sections.

Also, only a couple of passive optical tomography techniques are discussed in this review as the proposed DL-based technique in this study is based on using 2-D images acquired from multiple cameras. Laser-based tomography, electrical tomography and ultrasonic tomography are therefore excluded from this review. Other visualisation techniques that have been combined with tomographic approaches are also included due to their applicability in 3-D flame visualisation and measurement. The benefits and limitations of each technique are described in each case and their real practicability is also addressed.

Planar imaging and VT are two categories of these non-intrusive optical techniques. However, planar imaging techniques have some shortcomings such as optical access restriction and not being effective in the measurement of 3-D information in nature [8]. VT which is the other technique can overcome these shortcomings and can also provide better 3-D information for combustion diagnostics [9-10].

Computed tomography of chemiluminescence (CTC) which is one kind of VT has shown satisfactory results in reconstructing 3-D flame structures [11-13]. However, using this solution in practice comes with some limitations. The most crucial one is the high computational cost, and this is a problem as it is not possible to use the solution for real-time measurements or online monitoring.

The application of DL techniques for CTC has the potential to reconstruct a flame 3-D structure with good accuracy in real time. This helps in resolving the limitations highlighted above. Also, the high cost and complex setup of some passive optical techniques make DL approach favourable.

In this chapter, important background concepts for the understanding of this thesis will be explained. This includes tomographic and DL techniques for 3-D flame reconstruction. The section presents related tomographic and DL techniques such as:

- Passive optical techniques.
- Analytic reconstruction.
- Iterative reconstruction.
- CNN-based methods.
- Residual Neural Network (RNN) based methods.
- Transfer Learning methods.
- Hybrid DL method.

2.2 Tomographic Techniques for 3-D Flame Visualisation

2.2.1 Passive Optical Tomographic Techniques

In this section, a couple of passive optical-based tomographic techniques are discussed. This technique can be divided into two groups based on the camera setup and some other system configurations.

2.2.1.1 Single Camera Systems

A single-camera tomographic system was developed by Brisley et al. [14]. The system can perform the 3-D measurement of a burner flame using a two-colour pyrometric technique and filtered back-projection (FBP) algorithm for tomographic reconstruction of the grey-scale sections of the flames obtained from the camera. The schematic of the system is shown in Figure 2.1.

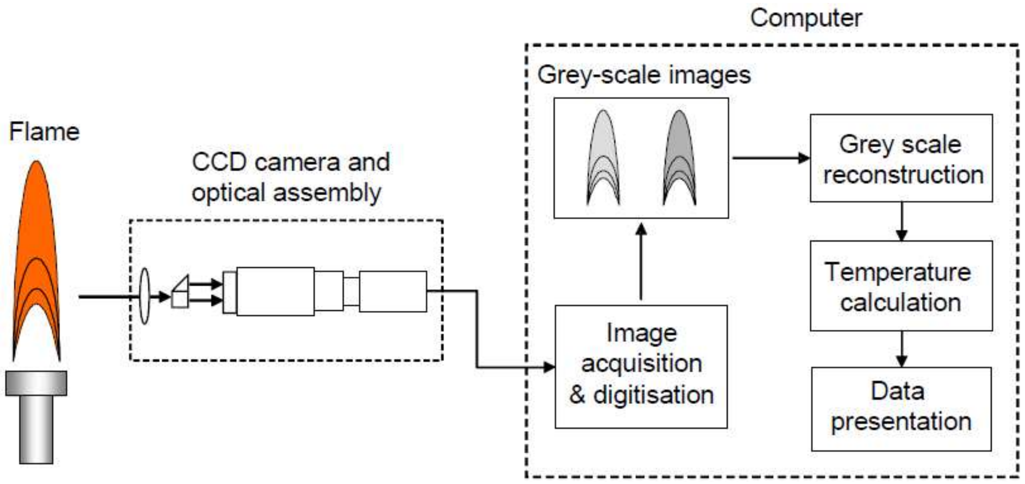

Computer
Grey-scale images
Flame
CCD camera and optical assembly
Grey scale reconstruction
Temperature calculation
Image acquisition & digitisation
Data presentation

Figure 2.1 Schematic diagram of a single camera system for 3-D flame imaging (Brisley et al. [14])

A similar approach was also followed by Yan et al. [15] for the 3-D reconstruction of flame temperature by applying FBP algorithms and two-colour pyrometry. A single-camera-based system was also used for the tomographic reconstruction of a burner flame by Veríssimo et al [16]. An image of the system is shown in Figure 2.2(a).

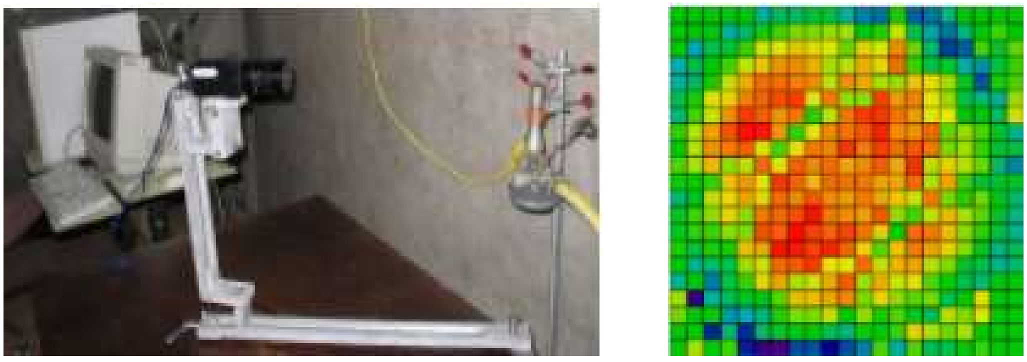

(a) (b)

Figure 2.2 Experimental setup and example results, (a) system setup and (b) reconstructed soot radical distribution (Veríssimo et al. [16])

For this approach, a single camera rotating around the flame at different angles was used in capturing the flame. Following this, the back-projection (BP) algorithm in addition to ART was then used to reconstruct the C_2 (at 515.14 *nm*) and soot (at 801 *nm*) radical distributions of the flame. This can be seen in Figure 2.2 (b).

A single camera system for the 3-D reconstruction of steady candle flames based on the maximisation of entropy technique (MENT) was implemented by Goyal et al [17]. The MENT is suitable for a situation when a very sparse set of image projections is available for the reconstruction process.

A stereoscopic tomographic technique using a single CCD camera was developed by Huang et al [18] to reconstruct the soot temperature and concentration distributions for an asymmetric diffusive flame. This technique is shown in Figure 2.3, where a CCD camera is coupled with a stereoscopic adapter used to capture flame images. Using this setup and an algorithm based on the matrix decomposition-based least squares method, it was possible to reconstruct the emission intensity distributions in the flame and obtain local soot temperature and concentration.

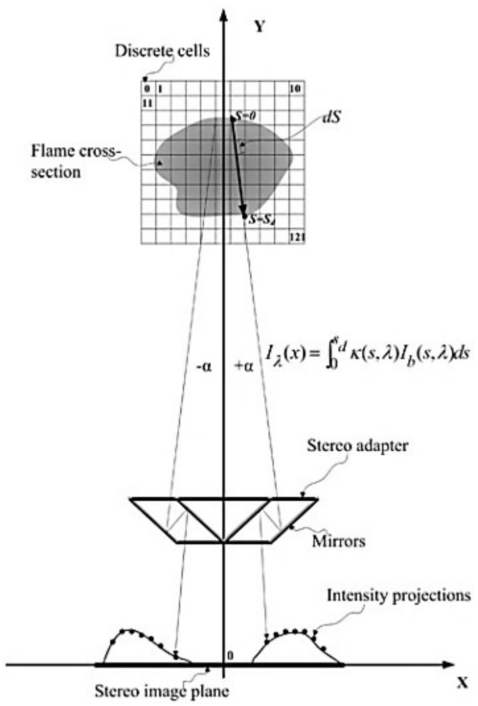
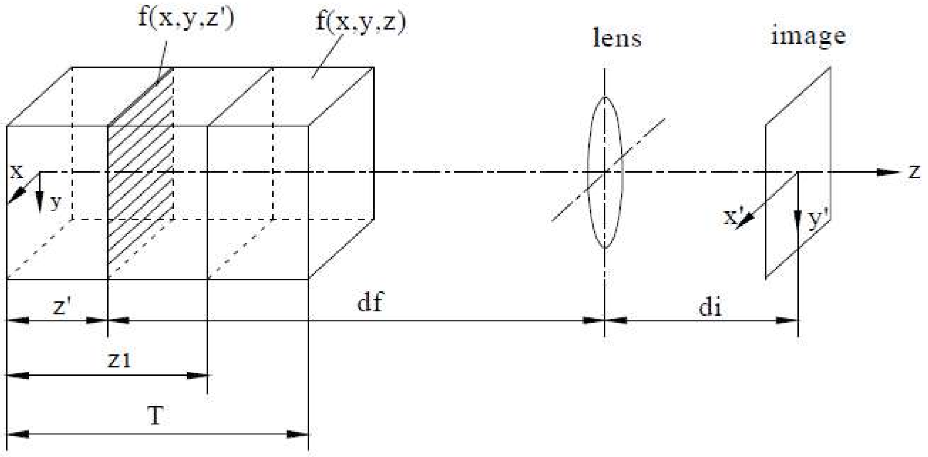
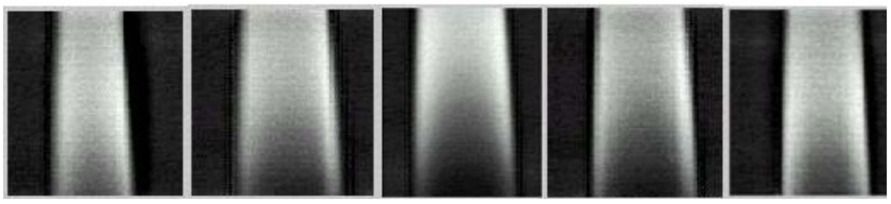


Figure 2.3 Stereoscopic image system (Huang et al. [18])

An optical sectioning tomography (OST) system for the 3-D monitoring of the flame temperature distribution of a steady-state candle flame was developed by Zhou et al. [19]. This solution is based on treating the flames as a combination of various numbers of 2-D sections. The principle of the OST is shown in Figure 2.4(a) where the original 2-D luminosity distribution of each 2-D flame section is retrieved through the supposition of section images. This is used to reconstruct the 3-D luminosity distribution of the flame as shown in Figure 2.4(b). Following this, using the relationship between the grey-level images and the temperature of the flame the 3-D temperature distributions were then reconstructed.



(a) Principle of the OST



(b) Reconstructed grey-level sections of the flame

Figure 2.4 2-D images, and reconstruction of a candle flame (Zhou et al [19].)

Gong et al. [20] also used a single CCD camera incorporating the OST and two-colour techniques to reconstruct the 3-D temperature distribution of impinging flames in an opposed multi-burner gasifier. A single camera was also used in the implementation of tomographic chemiluminescence (TC) [21, 22] and Schlieren tomography (ST) [23] techniques for 3-D flame visualisation.

The single-camera systems discussed in this section are practically easy to set up and install. They also can reconstruct a 3-D flame section. However, the single-camera system does come with some limitations. The techniques used in these systems can only be used under certain conditions which makes them not suitable for the 3D visualisation and motoring of turbulent flames [24].

2.2.1.2 Multi-Camera Systems

The use of various multi-camera-based systems has also been applied to the development of solutions for the 3-D visualisation and characterisation of flames. An image-based tomographic system using three RGB CCD cameras to capture concurrently six equiangular 2-D images of a flame was developed by Gilabert et al [25]. The schematic diagram of the system is shown in Figure 2.5.

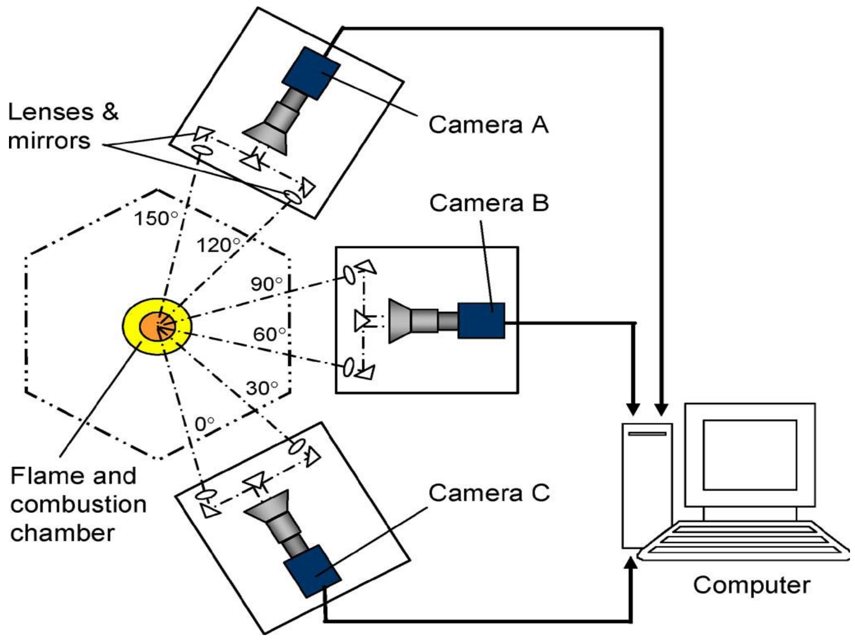
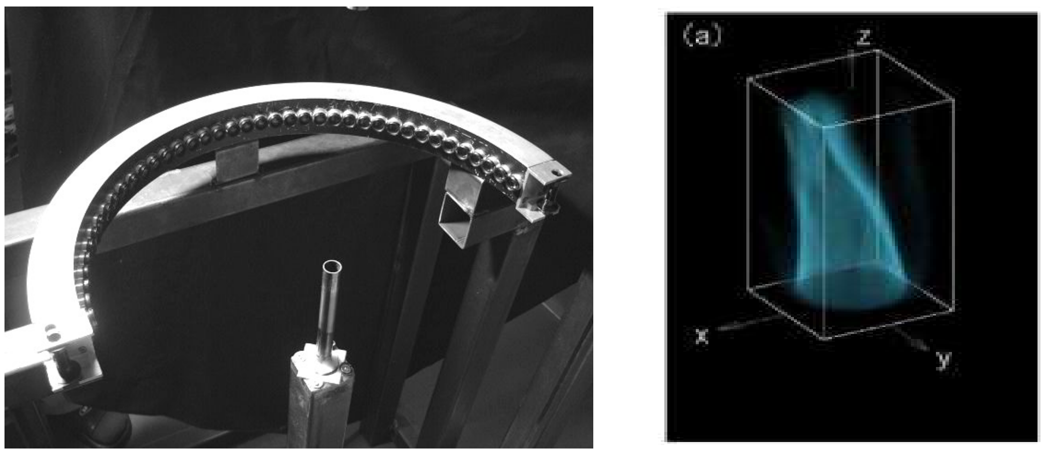


Figure 2.5 Schematic diagram of a three-camera-based tomographic system (Gilabert et al. [25])

A tomographic algorithm which combined the LFBP and ART was proposed for the 3-D reconstruction of the luminosity distribution of flames from the six image projections. Although a significant improvement was made in the flame reconstruction, the number of flame projections (only six) is still too few for a high spatial resolution in the reconstruction. In addition, the complexity of the system makes it unsuitable for installation on a large-scale furnace.

A multi-lens camera-based tomography system for reconstructing the 3-D grey-scale distributions of a turbulent flame was developed by Ishino et al. [26]. This system used propane fuel and was equipped with 40 special camera lenses arranged around one side of the burner. The overview of the system can be seen in Figure 2.6(a) and an example of 3-D reconstruction results is shown in Figure 2.6(b).



(a) (b)

Figure 2.6 Multi-lens-camera tomography system (a) and (b) reconstructed 3-D image (Ishino et al. [26])

The solution utilized a black-and-white panchromatic negative film which was loaded along the circumference behind the lenses. This film was developed after flame images were taken and then digitized. However, due to the imaging setup, this solution is not practical in an industrial setting. Additionally, the processing of the film used to capture the flame images could be time-consuming, making the system not suitable for real-time flame monitoring [24].

Zhou et al [27] in their pioneer work which made it possible for the temperature inside the furnace to be viewed three-dimensionally, carried out numerical investigations into the 3-D visualisation and temperature distribution in large-scale boiler furnaces. This was done using eight CCD cameras installed around the boiler furnace to capture radiative energy images inside the furnace. The temperature distribution of the furnace sections was then derived using the images obtained. However, the solution suffered from some technical difficulties which limited its applications. These difficulties include the complexity of the installation of the system and synchronization of the eight cameras.

A multi-projection-based imaging system for the tomographic reconstruction of the 3-D temperature distributions of axisymmetric or non-axisymmetric propane flames was developed by Correia et al [28]. The system shown in Figure 2.7 can produce eight image projections using a technique where four out of the eight images are from four cameras and the other four are created virtually.

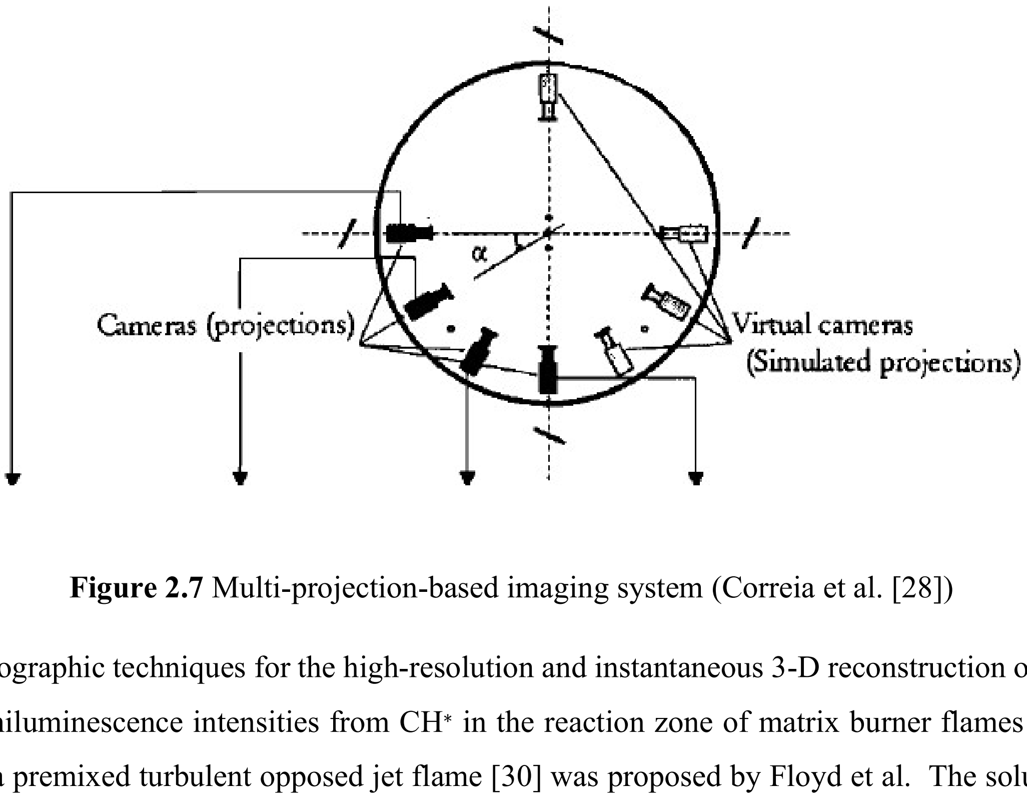


Figure 2.7 Multi-projection-based imaging system (Correia et al. [28])

Tomographic techniques for the high-resolution and instantaneous 3-D reconstruction of the chemiluminescence intensities from CH* in the reaction zone of matrix burner flames [29] and a premixed turbulent opposed jet flame [30] was proposed by Floyd et al. The solution was accomplished using ten instantaneous images from five cameras with a set of mirrors.

The experimental setup of the CTC techniques and the cross-section reconstruction of the flames is shown in Figure 2.8. However, due to the complexity involved with setting up the mirrors and cameras, the system is not suitable for practical application in the industry.

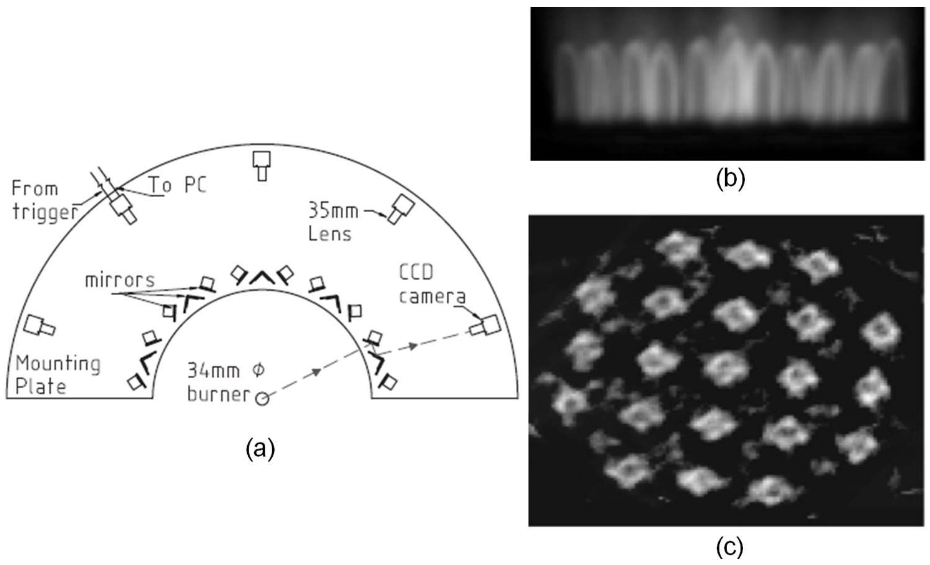


Figure 2.8 Schematic diagram of the instantaneous CTC system (a) experimental setup, (b) grey-level image, (c) cross-section reconstruction (Floyd et al.[29])

Following a discussion of some multi-camera-based systems developed for the 3-D visualisation and characterisation of flames, it is evident that these setups have been able to acquire multiple image projections and perform the 3-D reconstruction of the flame using various reconstruction algorithms.

These systems are capable of reporting more accurate reconstructions and are well-suited for unsteady and asymmetric flames. However, these systems suffer from a high cost and complex setup due to the increased number of cameras, lenses and mirrors required. Therefore, there is a need for a low-cost solution which could be easily installed in practical furnaces for the 3-D visualisation and quantitative characterisation of flames [24].

2.3 Flame Reconstruction with Computed Tomography

In this section, a brief discussion of Computed tomography (CT) techniques is discussed. This technique can be divided into two groups mainly analytical reconstruction and iterative reconstruction methods.

2.3.1 Computed Tomography

CT is a highly effective tool capable of examining the external and internal structures of many industrial applications as well as providing accurate geometrical information with remarkably high accuracy [31].

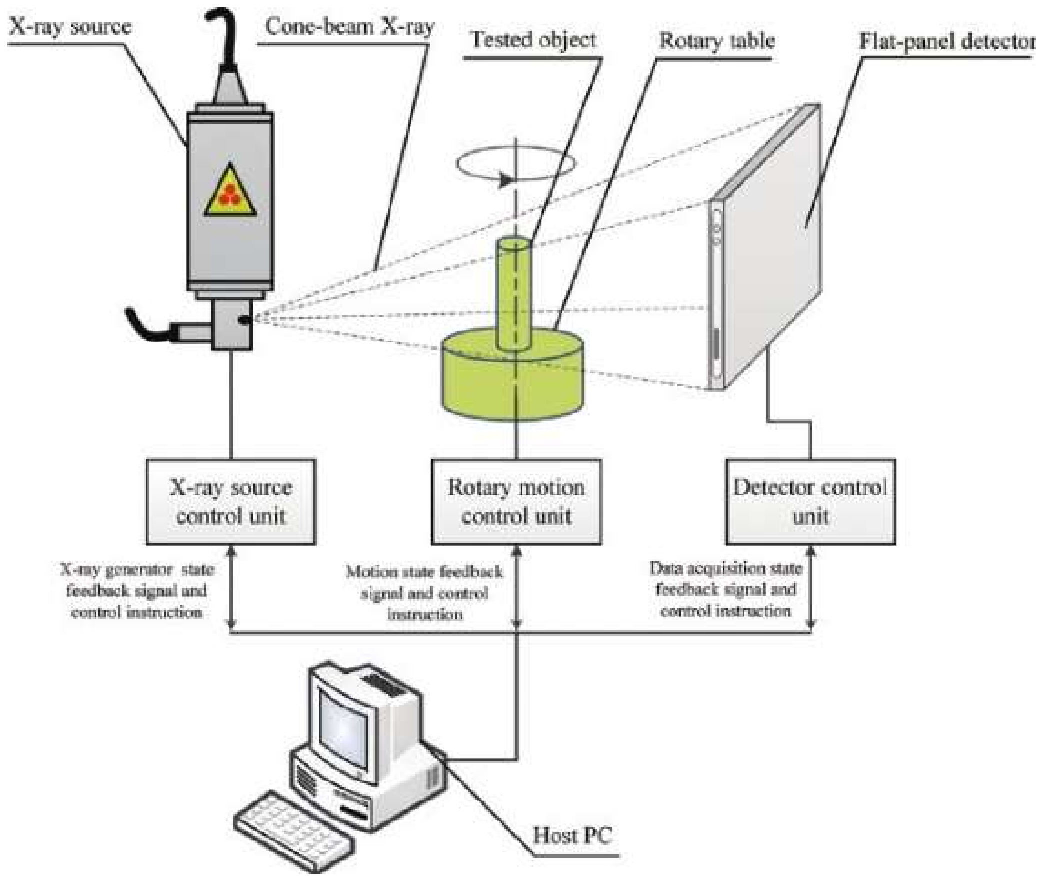

X-ray source
Cone-beam X-ray
Tested object
Rotary table
Flat-panel detector
X-ray source control unit
Rotary motion control unit
Detector control unit
X-ray generator state feedback signal and control instruction
Motion state feedback signal and control instruction
Data acquisition state feedback signal and control instruction
Host PC

Figure 2.9 The principal diagram of a cone-beam CT scanning system (Yang et al. [34])

It is a tomographic imaging tool which uses X-ray transmission through an object for providing a non-invasive technique that produces reliable information via a set of measurements (projections). This information is then used to reconstruct a cross-sectional (2-D) or a volumetric (3-D) image to study the interior of a domain [32]. The algorithms used for CT image reconstruction are based on the mathematical foundations of the Radon theorem. This was published in 1917 by an Austrian mathematician known as Johann Radon [33].

A typical CT system consists of an X-ray source, a detector, a rotary table, and a processing unit. (See Figure 2.9). This enables the system to capture the projection data of the object under examination from any angle. This data is then processed and used for visualisation and data analysis. A volumetric representation of an object can be obtained by acquiring a contiguous set of CT slices [35].

The tomographic reconstruction methods used in the CT can be grouped into two groups:

- Analytical reconstruction methods.
- Iterative reconstruction methods.

These methods would be discussed in sections 2.3.2 and 2.3.3.

2.3.2 Analytical Reconstruction

Analytical reconstruction methods treat both images and projections as continuous functions. The process of image acquisition using analytical reconstruction methods is represented as a linear and continuous operator acting on the space of object functions representing the object under examination. The acquisition process gives rise to an integral transform of the unknown function describing the object under examination. Image reconstruction is then accomplished by inverting the integral transform [36].

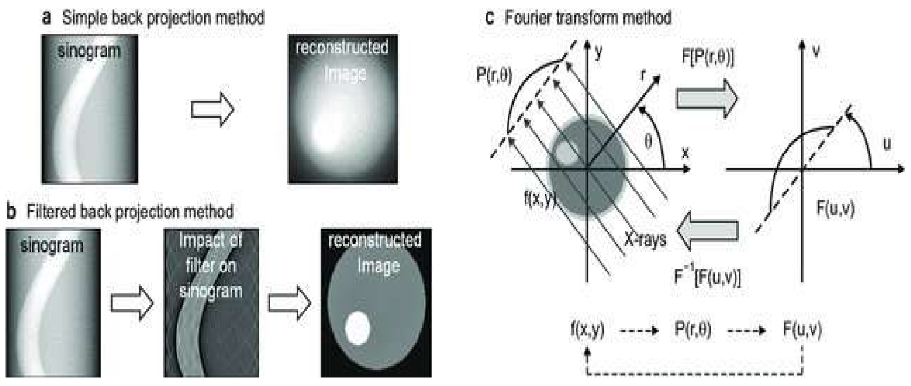

a Simple back projection method
sinogram
reconstructed image
b Filtered back projection method
sinogram
Impact of filter on sinogram
reconstructed image
c Fourier transform method
P(r,θ)
y
r
F[P(r,θ)]
v
θ
x
u
f(x,y)
X-rays
F(u,v)
F⁻¹[F(u,v)]
f(x,y) ----> P(r,θ) ----> F(u,v)

Figure 2.10 Computed tomography image reconstruction methods (Jung [37])

The diagram in Figure 2.10 shows some examples of CT image reconstruction methods: (a) Simple back projection algorithm method, (b) filtered back projection algorithm method, and (c) Fourier transform algorithm method. The process of image reconstruction in CT is a mathematical one that requires the calculation of the 2-D cross-sectional attenuation distribution function f (x, y) from a series of one-dimensional (1-D) X-ray projections P (r, θ) s as shown in Figure 2.10.

A set of parallel X-rays pass through the 2-D object of interest, these attenuated X-rays form a projection, P (r, θ). A collection of these projections at different angles during a single rotation of the X-ray-based CT system is called a sinogram. The sinogram is a linear transform of the cross-sectional image of the object and it displays all the different projections for any slices stacked together [37].

There are many types of analytical reconstruction methods. The most used analytical reconstruction methods are all in the form of filtered back projection (FBP), which uses a 1-D filter on the projection data before back projecting (2-D or 3-D) the data onto the image space.

In this study, FBP is used for the experimental work on numerical simulation. The following are the main steps involved in a filtered back projection image acquisition process [38]:

1. Data is acquired and forward projected into the sonogram space.
2. Data is filtered.
3. Filtered sinograms are back-projected into image space.

2.3.3 Iterative Reconstruction

Iterative reconstruction methods are based on discretizing the input image into pixels and a system of equations is set up to describe the imaging geometry and physics. The values of these pixels are then represented as unknown variables in a set of linear equations which are then solved using iterative algorithms [39].

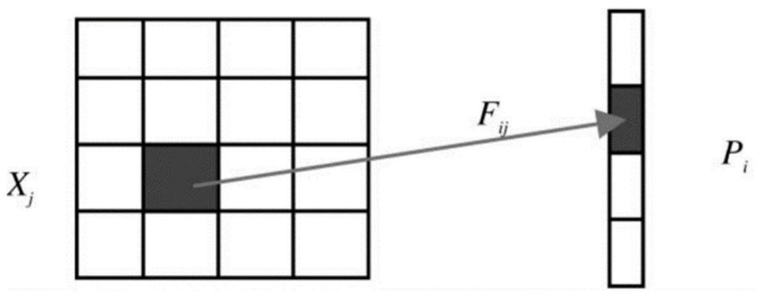


Figure 2.11 Image discretized into pixels (Zeng [39])

The setup of equations can be seen in Figure 2.11. The system of linear equations formed can be represented in the matrix form:

$$FX = P \tag{2.1}$$

where each element (Xj) in X is a pixel value, each element (Pi) in P is a projection measurement, and Fij in F is a coefficient that is the contribution from pixel j to the projection bin i. [39].

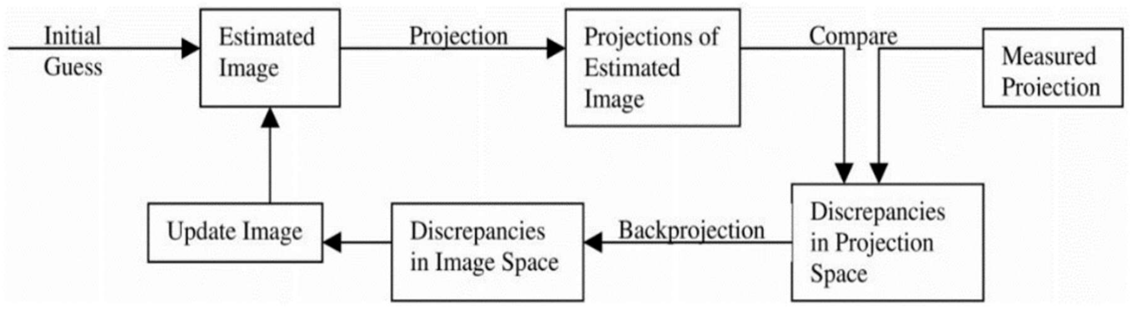

Initial Guess
Estimated Image
Projection
Projections of Estimated Image
Compare
Measured Projection
Discrepancies in Projection Space
Backprojection
Discrepancies in Image Space
Update Image

Figure 2.12 Flow chart of iterative image reconstruction process (Zeng [39])

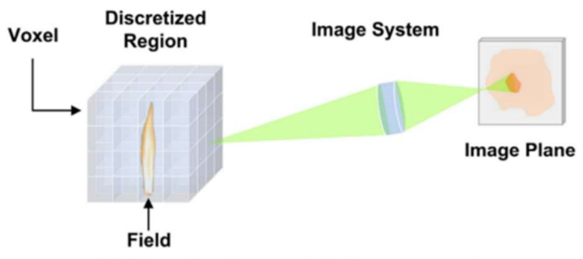
The diagram shown in Figure 2.12 describes the procedure for using an iterative algorithm. In this diagram, each loop represents one iteration. There are several types of iterative algorithms such as ART [40], SIRT [41], MART [42] and SART [43]. However, in this study, SART is used for generating ground truth data from projection samples.

2.4 Flame Reconstruction with Neural Networks and Deep Learning

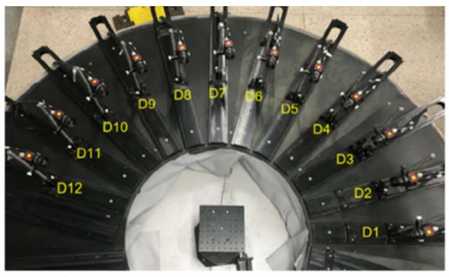
DL has recently been gaining a lot of interest due to its impressive performance in extracting features and modelling [44]. One of the most widely used DL techniques is CNN which has been applied in tomography solutions such as ultra-sonic tomography [45], magnetic resonance imaging (MRI) [46] and X-ray CT [47]. The performance of CNN in these applications has shown satisfactory results in terms of rapid data processing and image quality improvement. In the following sections, a review of some flame reconstruction solutions using DL techniques is presented.

2.4.1 Convolutional Neural Network-based Methods

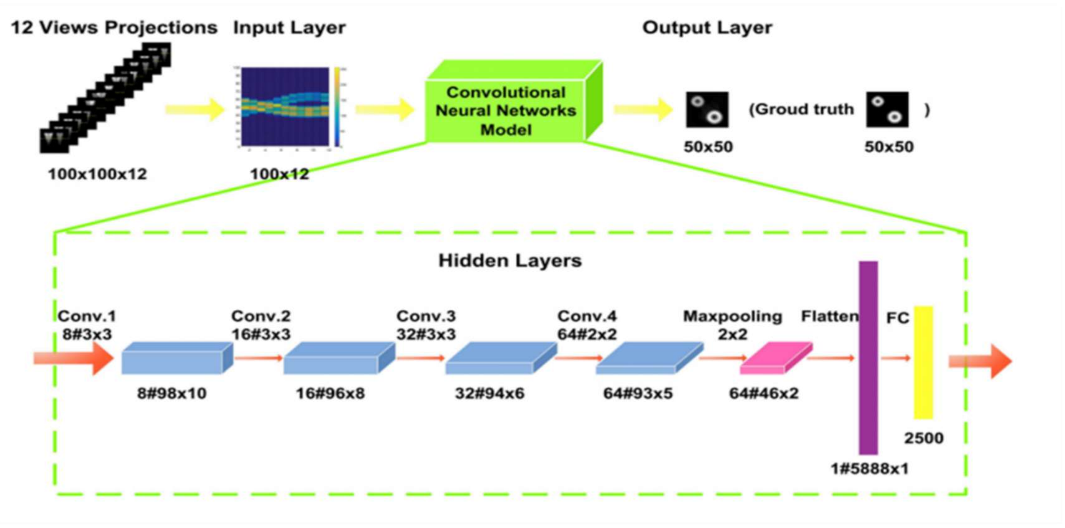
Jin et al. [48] implemented a 3-D rapid flame chemiluminescence tomography (FCT) reconstruction system based on CNN as shown in Figure 2.13. The CNN network model was established after initial work using numerical simulation where phantoms were used to mimic real flames. Following this, twelve colour CCD cameras were used to capture projection data of a real flame and this data was used to train the established CNN model.



(a) 3D projection model of reconstruction



(b) FCT system with 12 CCD cameras



(c) Architecture of CNN model for 3D FCT

Figure 2.13 3-D FCT reconstruction system using CNN (Jin et al.[48])

The accuracy of the reconstructed 3-D flame structure obtained from this system was credible. Additionally, when compared with traditional iterative techniques, this solution demonstrated a better performance whilst also accomplishing reconstruction faster.

To solve the inversion problem in the CTC system, Huang et al. [49] proposed using CNN and proper orthogonal decomposition (POD). In this study, a variety of experiments were conducted, confirming that this method was capable of reconstructing chemiluminescence distributions in three dimensions. Figure 2.14 illustrates CNN with POD. (a): training. (b): dimensionality reduction.

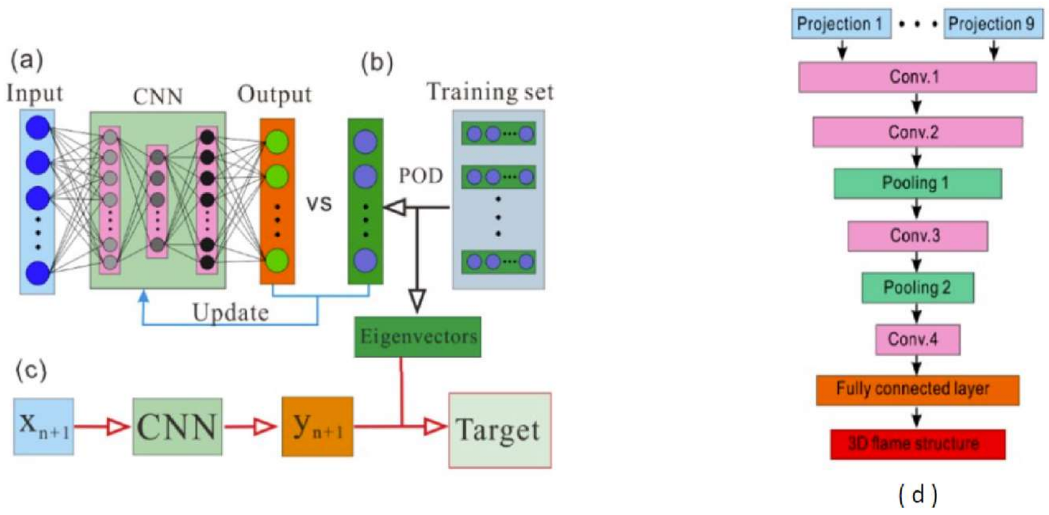
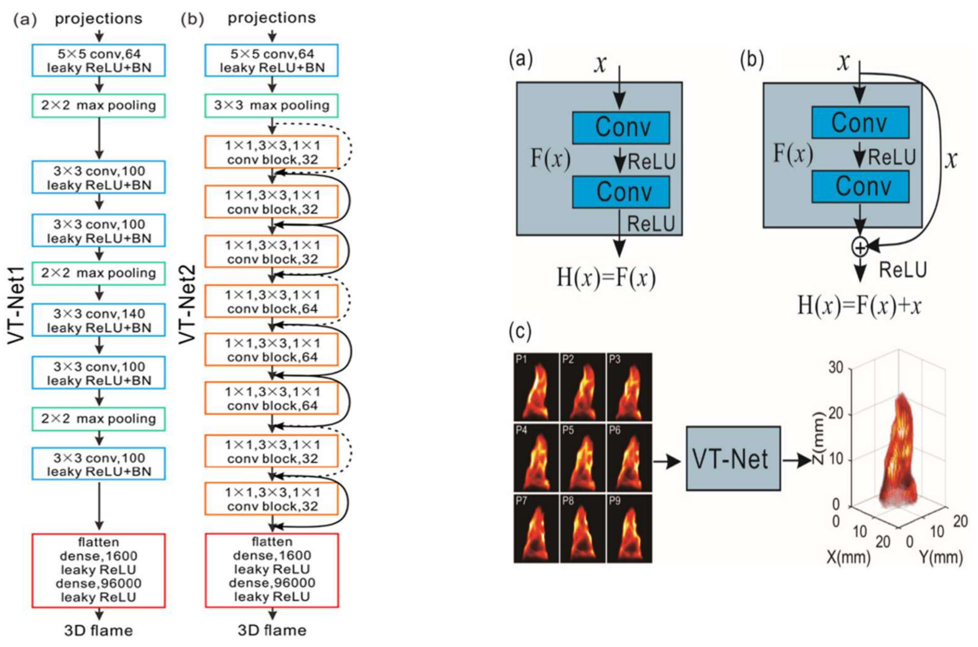


Figure 2.14 CNN using proper orthogonal decomposition (Huang et al. [49])

Figure 2-14 shows the training (a), dimensionality reduction (b), and testing processes of convolutional neural networks with a POD (c). In addition to the reconstruction accuracy, the proposed CNN model had desirable advantages in terms of computational efficiency and noise immunity. The reconstruction accuracy obtained was comparable to that of traditional inversion methods like the ART algorithm. Therefore, it is suitable for practical applications.

2.4.2 Residual Neural Network-based Methods

Huang et al. [50] implemented a VT reconstruction solution based on RNNs to reconstruct turbulent flame. For this work, two different network architectures were implemented. One was based on a standard CNN model using sequential connections (VT-Net1), and the other on RNNs using skipped connections.



(a) projections
5×5 conv,64 leaky ReLU+BN
2×2 max pooling
3×3 conv,100 leaky ReLU+BN
3×3 conv,100 leaky ReLU+BN
2×2 max pooling
3×3 conv,140 leaky ReLU+BN
3×3 conv,100 leaky ReLU+BN
2×2 max pooling
3×3 conv,100 leaky ReLU+BN
flatten dense,1600 leaky ReLU dense,96000 leaky ReLU
3D flame
VT-Net1
(b) projections
5×5 conv,64 leaky ReLU+BN
3×3 max pooling
1×1,3×3,1×1 conv block,32
1×1,3×3,1×1 conv block,64
flatten dense,1600 leaky ReLU dense,96000 leaky ReLU
3D flame
VT-Net2
(a) x Conv ReLU Conv ReLU F(x) H(x)=F(x)
(b) x Conv ReLU Conv F(x) x ReLU H(x)=F(x)+x
(c) P1 P2 P3 P4 P5 P6 P7 P8 P9
VT-Net
Z(mm) X(mm) Y(mm)

Figure 2.15 VT-Nets based 3-D flame reconstruction solution (Huang et al. [50])

The diagram in Figure 2.15 shows the architecture of the two kinds of VT-Nets implemented: (a) VT-Net1 with sequential connections; (b) VT-Net2 with skip connections; (c) The flow chart of the reconstruction process. Using turbulent flame, nine 2-D projections of the flame were captured via a customized fibre bundle and a high-speed camera. This data was then used to train both VT-Net models. The trained VT-Net models were able to rapidly reconstruct the 3-D flame structure with a high reconstruction accuracy even with a limited number of projections.

2.4.3 Transfer Learning Methods

Cai et al. [51] investigated the feasibility of using TL for a VT flame reconstruction solution (Figure 2.16). In this study, CNN network models were implemented using both transfer learning and semi-supervised learning techniques.

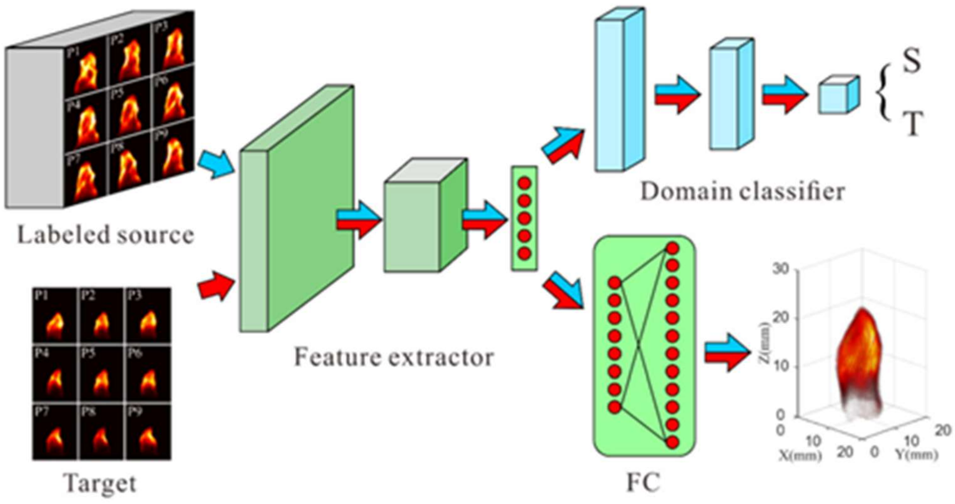


Figure 2.16 Transfer learning framework for 3-D flame reconstruction (Cai et al. [51])

This work focused on how to improve the performance of a 3-D flame reconstruction when there were limited labels and a small dataset available using TL techniques. Results from this work show that it is possible to significantly improve the reconstruction accuracy of a 3-D flame reconstruction solution even with the use of limited labels.

2.4.4 Hybrid DL Methods

The authors of Huang et al. [52] utilized a time-resolved VT technique in conjunction with DL algorithms to predict a 3-D flame evolution rapidly. An algorithm using a CNN and a long short-term memory network (LSTM) hybrid model was used in this study to rapidly reconstruct a 3-D flame structure based only on a history of 2-D projection data. Figure 2.17

shows the layout of the experimental setup. In this study, nine flame projection images were collected from a variety of different angles at various moments in time.

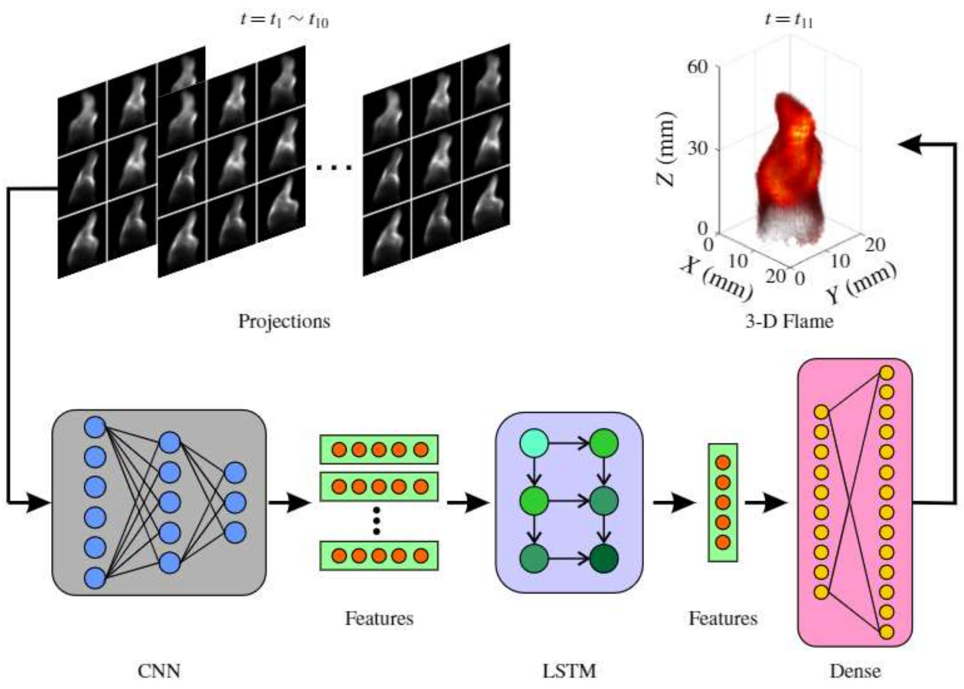


Figure 2.17 Schematic of the proposed CNN–LSTM model (Huang et al. [52])

It has been shown that the evolution of a flame in 3-D (for example, t = t11) can be predicted by using this model, which comprises a CNN, a LSTM, and a dense layer, based on its history of 2-D projections. Training for CNNs consisted of two steps. Initially, the CNN model is trained to extract features from projections, and then the LSTMs are trained to model the temporal sequence of these features.

2.5 Overview of Deep Learning Flame Reconstruction Methods

In this section, the four DL flame reconstructed methods discussed above are compared. The advantages and disadvantages of each method are presented. Following a review of the literature referenced in section 2.3, the table below (Table 2.1), is a summary of the findings.

Table 2.1 Advantages and disadvantages of DL flame reconstruction methods

Method	**Advantages**	**Disadvantages**
Convolutional Neural Network [49,48]	High computational efficiency. Fast reconstruction time. Jin et al. [48] reported a reconstruction time of 1.26 seconds on a CPU-based platform. CNN model can retrieve the 3-D distribution with credible accuracy and structural similarity. Jin et al. [48] reported a SSIM value of 0.82 for reconstructed flames.	A large dataset of approximately 10000 to 15000 is required for training purposes [48]. Long training time. Jin et al. [48] reported a training time of 1.3 hrs. Some small-scale features such as small wrinkles in the turbulent flame surface could be difficult to reconstruct. Collecting a vast amount of training data for CNN-based methods could be time-consuming.
Residual Neural Network [50]	RNN-based methods outperform the ART algorithm in reconstructing the chemiluminescence intensity among the whole flame surface. Fast reconstruction time. Huang et al. [50] reported a reconstruction time of 4 milliseconds on a CPU and GPU-based platform.	Large datasets are required for training the model. Huang et al. [50] used a training set of 24,000 samples. Long training time. Huang et al. [50] reported a training time of 35 mins. Collecting a vast amount of training data for RNN-based methods could be time-consuming.

	RNN-based methods have a better performance when compared with CNN methods in the case of limited 2-D projection data. Superb reconstruction capability e.g. Some small-scale features such as small wrinkles in the turbulent flame surface could be reconstructed easily.	
Transfer Learning [51]	Significant improvement in the reconstruction accuracy of CNN can be achieved by using transfer learning. According to Cai et al. [51], the correlation coefficient between the reconstructed flame and ground truth was larger than 0.98 for three commonly encountered application scenarios. For practical applications, transfer learning and semi-supervised learning offer promising approaches for enhancing the generalization performance of CNNs for inversion problems in VT.	The dataset available for learning is smaller. Cai et al. [51] used a dataset of 1000 samples for training. A network model previously trained for a similar problem is required.

	The training of a network model from scratch is not required and the training time is reduced significantly.	
Hybrid [52]	CNN-LSTM models can determine critical parameters of a flame structure, such as the flame surface density, wrinkle factor, flame normal direction, and flame curvature.	Compared to other training methods other than transfer learning, smaller datasets could be used. Huang et al. [52] used a dataset of 8000 samples for training.
	Faster reconstruction time when compared with other methods. Huang et al. [52] reported a reconstruction time of approximately 2 milliseconds on a CPU and GPU-based platform.	Knowledge of at least two network architectures is required.
	Improved reconstruction accuracy. According to Huang et al. [52], their 3D flame reconstruction is highly similar to the corresponding ground truth (Correlation coefficient = 0.981).	The combination of some network architectures might not be suitable for certain applications.
	The training time is shorter than that of a CNN-based model. Huang et al. [52] reported a training time of 30 mins.	

In the table above (Table 2.1), it is important to note that there are differences between the various DL flame reconstruction methods discussed in terms of the flame structure, dataset, hardware platform, code optimization and architecture of the DL model. As a result, it may be difficult to establish a direct correlation.

2.6 Summary

A review of current strategies for 3-D visualisation and quantitative characterisation of flames has been conducted. It has been shown that 3-D flame visualisation and optical tomographic techniques can be applied in both industrial and laboratory settings. Combustion processes have been improved using those systems. However, it has been shown that none of the existing techniques can reconstruct a complete 3-D structure of a flame.

It is not currently possible to perform an accurate 3-D reconstruction using a simple and low-cost system setup because no efficient algorithm has been developed for this purpose. Several techniques that have been reported are preliminary or are only suitable for use in the laboratory. Most of them suffer from problems such as low resolution, complex setup, and high costs. Due to their fundamental limitations in sensing principles, some are unsuitable for turbulent flames.

Among the various methodologies that can be applied to the reconstruction of the 3-D flame, passive optical tomography holds many obvious advantages. The main advantages are the easy setup of the 3-D flame reconstruction system, the high spatial resolution, and the low cost of the system, making it a suitable choice for 3-D flame reconstruction and characterisation of practical furnaces. Due to its non-invasive nature, passive optical tomography does not alter the natural state of the flame and can be performed either with a single camera or with a multi-camera setup.

Passive optical tomography can also be performed using either a single or a multi-camera setup. Although the single camera system is very straightforward, it is only suitable for steady flames and axisymmetric flames and is therefore unsuitable for practical furnace applications. It is possible to produce multiple image projections with multi-camera systems, and the flame reconstruction is more detailed. Those systems that rotate or scan can produce many projections and are therefore better at reconstructing images. It does, however, have the same limitation as a single camera system, which is that the flame must be stable while being viewed. Additionally, no proven tomography method has been developed that meets the requirements of real-time flame monitoring.

A review of the current state-of-the-art in this field indicates that although progress has been made in the field of visualisation and characterisation of flames over the past few years, there are still technical challenges which remain to be overcome in the field of flames. Specifically, it is essential to develop a practical hardware platform that can be used for generating enough image projections that will enable better spatial resolution and system reliability.

To enhance the performance of the system in reconstructed and characterised 3-D flames, it is also important to use efficient algorithms. Thus, a new technical strategy must be implemented in order to help in achieving a higher level of reliability and accuracy with 3-D reconstruction and quantitative flame characterisation. In the following chapters, a detailed description and discussion of the new technical strategy will be presented.

In addition to this review, a discussion of CT-based tomographic and DL methods and their application to the reconstruction of 3-D flames relevant to this study has been provided. These methods have been demonstrated to be successful in solving the problem of 3-D flame reconstruction from 2-D projection images.

For the CNN-based methods, high computational efficiency and fast reconstruction time were achieved. CNN models could also reconstruct a 3-D flame with credible accuracy and structural similarity. RNN-based methods outperform the ART algorithm in reconstructing the chemiluminescence intensity among the whole flame surface. They also reported a better performance when compared with CNN methods for the case of limited 2-D projection data. In addition to this, small-scale features such as small wrinkles in the turbulent flame surface could be reconstructed easily using RNN DL methods. The DL methods were also found to report satisfactory results. They showed a significant improvement in the reconstruction accuracy of CNN and had the advantage of not having to train a network model from scratch. As for Hybrid DL methods, the use of a CNN-LSTM-based solution enabled certain key parameters of a flame structure - such as flame surface density, wrinkle factor, flame normal direction, and flame curvature to be determined.

In summary, DL-based methods can be used to reconstruct 3-D flame structures with better performance than traditional iterative methods, and it has the potential to be applied for online 3-D flame reconstruction, meaning that no offline reconstruction of flames would be necessary. Although several DL methods have been discussed, CNN was used in this study because of its performance in the reconstruction of 3-D flame structures and due to its simplicity of implementing and training a CNN network model. It should be noted that the RNN-based methods discussed in this chapter reported fast 3-D flame reconstruction times. The use of RNNs is a better choice when processing sequential data, such as speech or natural language. CNNs, on the other hand, have shown impressive results on a wide range of computer vision problems and are better for image processing and various computer vision tasks. Since CNN has shown impressive performance in computer vision tasks such as image classification, object detection, segmentation, etc., it has been used for the study.

3
Methodology

3.1 Introduction

In this chapter, important background concepts for the understanding of this thesis will be explained. Initially, DL terms will be introduced, followed by concepts and definitions specifically for the subset of ML challenges faced during this study, namely deep convolutional network models. This is then followed by a description of the ML framework and performance metrics used for this study.

3.1.1 Introduction to Deep Neural Networks

DL is a subset of ML (Figure 3.1), which is essentially a neural network with three or more layers. The implementation of a neural network in this way with layers attempts to mimic the operations of the human brain. One of the key benefits of DL is the ability to learn enormous amounts of data.

A neural network with a single layer can make approximate predictions, but additional hidden layers in the network can help to optimize and improve the accuracy of these predictions. DL is at the leading edge of much artificial intelligence (AI) applications and services that improve automation, performing analytical and physical tasks without human intervention.

DL technology currently lies behind everyday products and services such as digital assistants, medical image analysis, face detection and recognition, credit card fraud detection and self-driving cars [54]. This is because they have been shown to perform better than established techniques in various tasks e.g., audio and speech processing, natural language processing, computer vision and medical diagnosis [55], [56].

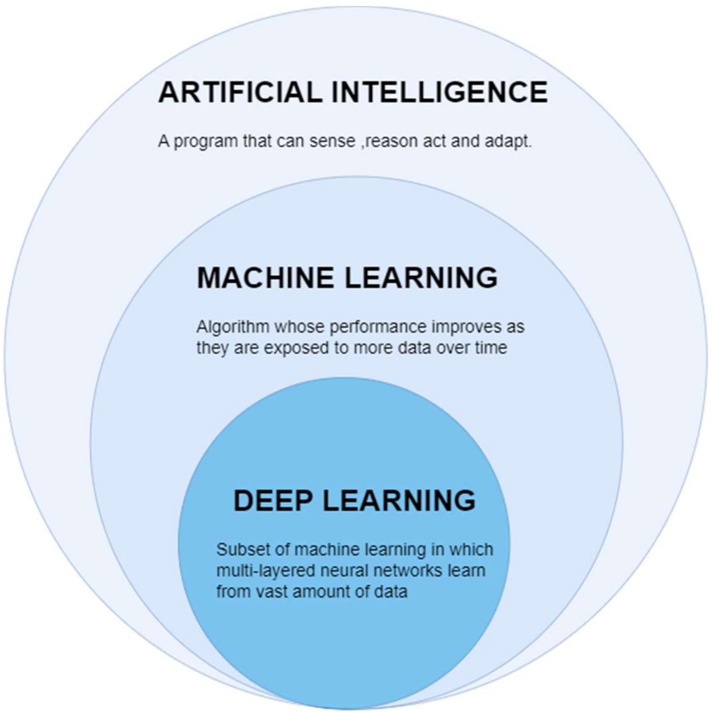

ARTIFICIAL INTELLIGENCE
A program that can sense ,reason act and adapt.
MACHINE LEARNING
Algorithm whose performance improves as they are exposed to more data over time
DEEP LEARNING
Subset of machine learning in which multi-layered neural networks learn from vast amount of data

Figure 3.1 Deep learning in the context of machine learning and artificial intelligence (Alzubaidi et al. [53])

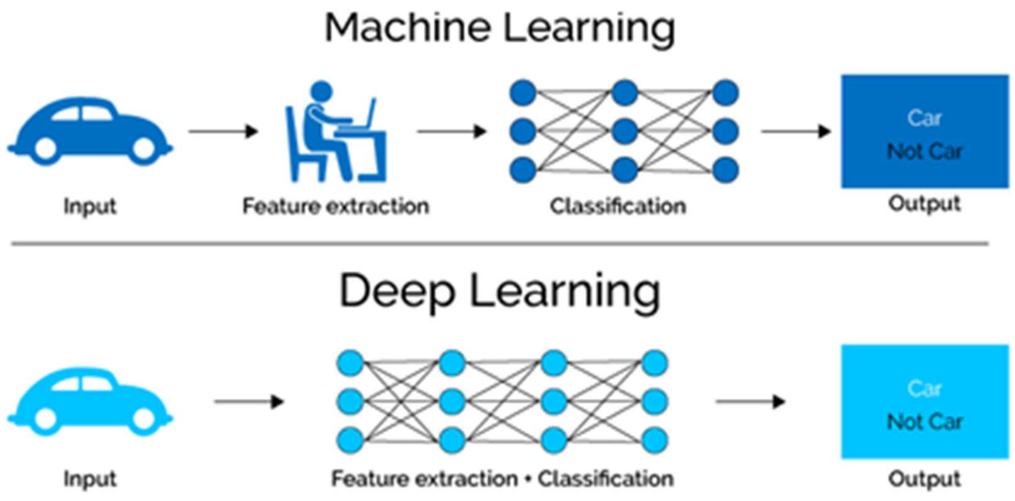
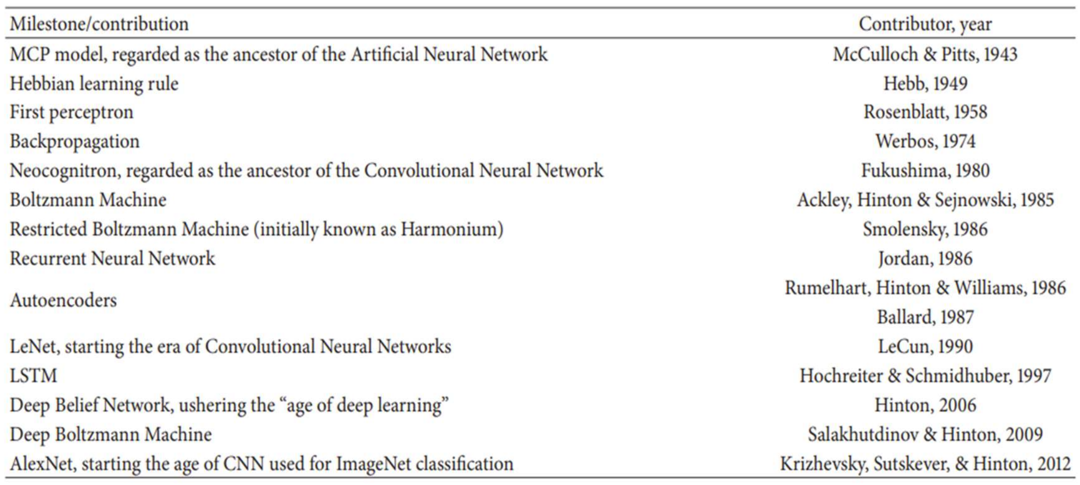


Figure 3.2 Deep learning vs Machine learning (Odi et al. [57])

The main principle of DL is that certain rules are learned from data examples, and this knowledge can then be used to make accurate predictions on new, previously unseen data. As shown in Figure 3.2, DL algorithms do not need a human present to identify and extract features from data. According to Goodfellow et al, DL enables a computer system to learn from experience and understand the world in terms of a hierarchy of concepts, and this enables the computer to learn more complicated concepts by building them up out of simpler concepts [58].

The desire to create a system that mimics the human brain drove the initial development of deep neural networks. McCulloch and Pitts worked on how the brain used interconnected cells known as neurons to produce complex in 1943. This model of a neuron by McCulloch and Pitts called a MCP model, contributed significantly to the development of deep neural networks [59].

Table 3.1 Important milestones in the history of neural networks and machine learning, leading up to the era of deep learning (Voulodimos et al. [60])



Milestone/contribution	Contributor, year
MCP model, regarded as the ancestor of the Artificial Neural Network	McCulloch & Pitts, 1943
Hebbian learning rule	Hebb, 1949
First perceptron	Rosenblatt, 1958
Backpropagation	Werbos, 1974
Neocognitron, regarded as the ancestor of the Convolutional Neural Network	Fukushima, 1980
Boltzmann Machine	Ackley, Hinton & Sejnowski, 1985
Restricted Boltzmann Machine (initially known as Harmonium)	Smolensky, 1986
Recurrent Neural Network	Jordan, 1986
Autoencoders	Rumelhart, Hinton & Williams, 1986 Ballard, 1987
LeNet, starting the era of Convolutional Neural Networks	LeCun, 1990
LSTM	Hochreiter & Schmidhuber, 1997
Deep Belief Network, ushering the "age of deep learning"	Hinton, 2006
Deep Boltzmann Machine	Salakhutdinov & Hinton, 2009
AlexNet, starting the age of CNN used for ImageNet classification	Krizhevsky, Sutskever, & Hinton, 2012

Table 3.1 above shows some major contributions to the field of deep learning. This includes Recurrent Neural Network [61], LeNet [62], LSTM [63] and one of the most significant advances came in 2006 when Hinton et al [64] introduced the Deep Belief Network.

However, some other factors have also contributed to the rise of deep learning. They include the presence of large, high-quality, and publicly available labelled datasets and advances in computer hardware architecture. For instance, innovation in computer architecture design has enabled the use of advanced computing architecture based on parallel processing units for training DL models, this has made it possible to move from traditional CPU-based architecture to other hardware accelerators units such as GPUs, TPUs and NPUs [65]. This has helped in significantly improving acceleration in DL models' training.

Furthermore, the emergence of powerful ML frameworks like TensorFlow [66], Theano [67], and PyTorch [68], allows for faster prototyping and implementation of DL solutions.

DL models can be used for a variety of complex tasks:

- Deep Belief Network (DBN) for the general classification.
- RNN for sequence learning and time series.
- CNN for classification in images, sound, and text.
- Restricted Boltzmann Machine (RBM) for feature extraction.

As this study is based on the use of CNNs, the next section would briefly describe CNN.

3.1.2 Convolutional Neural Network

CNN are a specific category of neural networks which consist of convolutional layers, pooling layers, activation layers and fully connected layers. They are used to extract features from an image or a video input. They are used for various computer vision applications like image classification, object detection, semantic segmentation, face recognition etc.

The architecture of a CNN is analogous to that of the connectivity pattern of neurons in the human brain and was inspired by the organization of the Visual Cortex. The work of D. H. Hubel and T. N. Wiesel on the functional architecture in the cat's visual cortex [69] inspired work on CNNs. Figure 3.3 shows an example of a CNN used in an image classification solution. The CNN model takes in an input image and extracts feature from it using various layers. Finally, it can predict the class of the object in the given image.

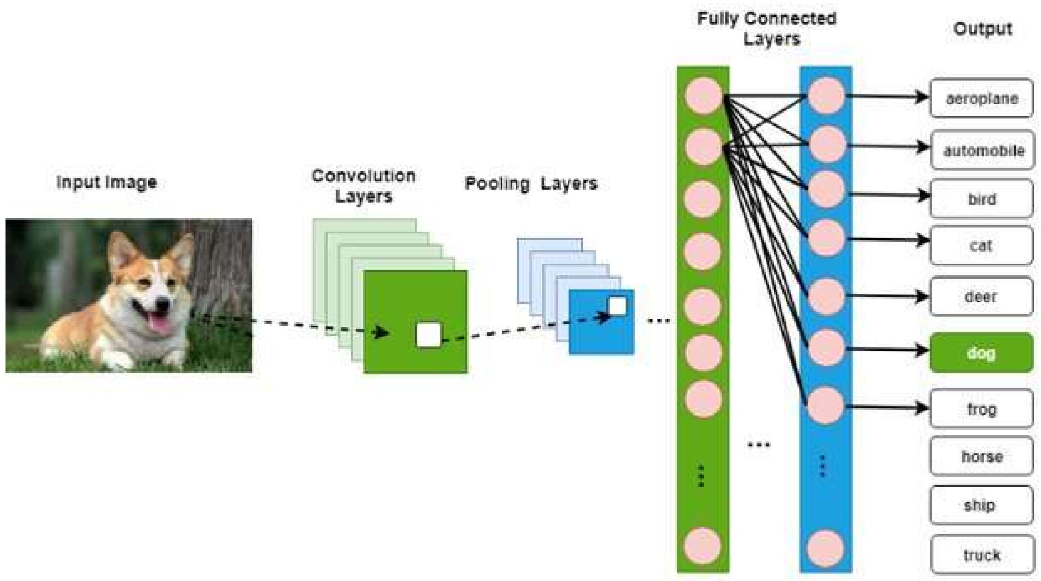


Figure 3.3 CNN-based image classification (Aamir et al. [70])

In the following sections, the various layers and connections of a convolutional network would be discussed.

3.1.2.1 Input Layer

This is the input to the CNN network. It is a video stream or an image which is made up of a collection of pixels. An example of this is the input dog image in Figure 3.3. It is usually either greyscale or RGB. The pixels which make up this input layer have a numeric value between 0 to 255, and these values represent the colour value of the pixels in the input image or video. The example in Figure 3.4 below is that of an RGB input image with 3 channels and size 4*4.

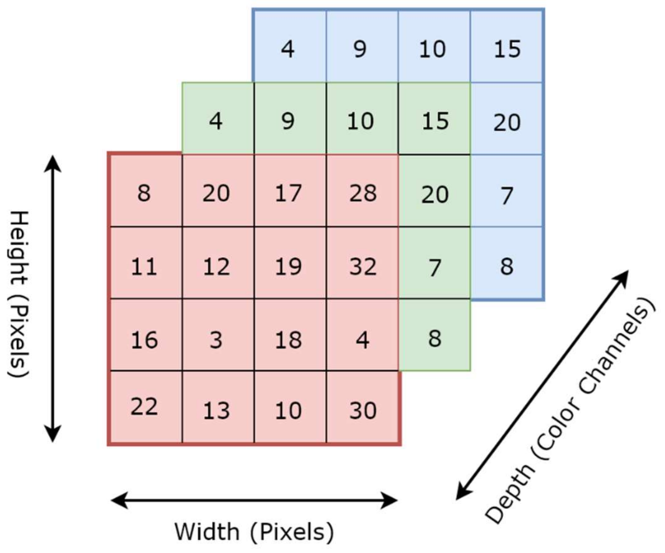


Figure 3.4 RGB-based Input layer image (Quddus [71])

3.1.2.2 Dense Layer

The dense layer is one of the most used layers in a CNN. It is deeply connected with its preceding layer in such a way that every output from the preceding layer is input and the outputs from the dense layer are passed out as inputs into the next layer. In Figure 3.5 below, the hidden layer is a dense layer.

In feedforward mode operation, computation in the dense layer is performed as shown in (3.1):

$$A = Ø\,(W^{T}S + B) \tag{3.1}$$

where A is the output, Ø refers to the activation function, W is the weights, B is the bias and S is the input matrix. The dense layer performs a matrix multiplication between the matrix of weights and the input matrix. This is then added to a bias vector before an activation function operates on the results

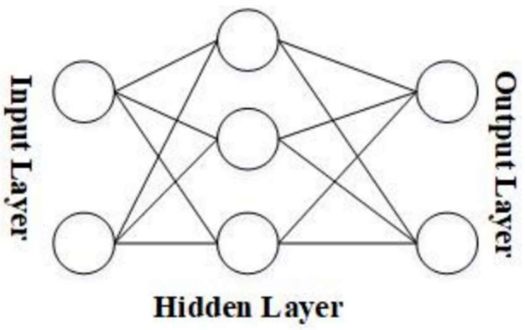


Figure 3.5 Schematic description of a dense layer (Kong et al [72])

3.1.2.3 Convolutional Layer

The convolutional layer is particularly important in a CNN. The bulk of the computational operations occurs in the convolutional layer [73]. Dot product operations are performed by the convolutional layer between two matrices which are:

1. The set of learnable parameters is known as a kernel or filter.
2. The restricted portion of the receptive field of the image [74].

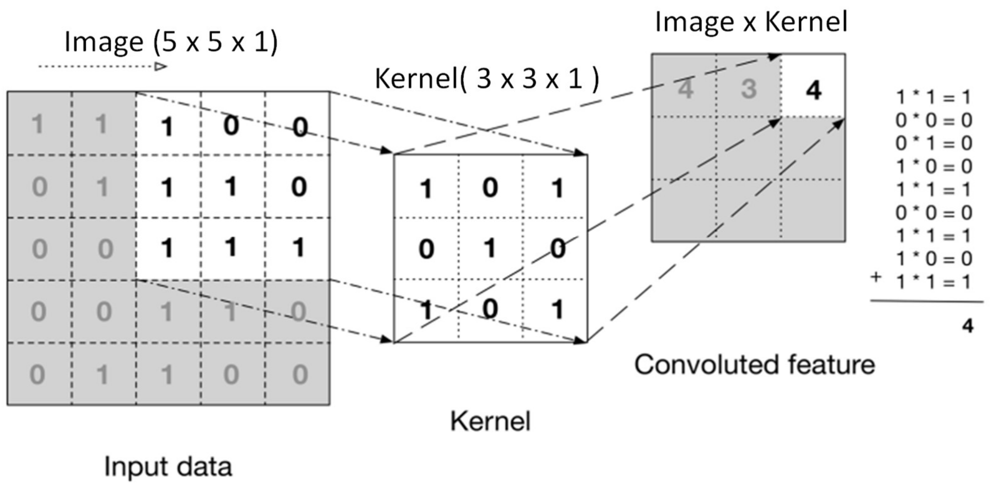


Figure 3.6 Convoluting operation (Nuzzo [75])

From Figure 3.6, the dot product operation between the kernel matrix and the image produces a convoluted feature which could be referred to as a feature map. In a forward pass operation, the kernel traverses from the left to the right of the image with a certain sliding size called stride. This continues until it traverses the whole width of the image after which it moved down and starts the process again. This continues until the whole image is traversed. The diagram in Figure 3.7 shows the movement of the kernel matrix across the image. The entire process described above is called convolution.

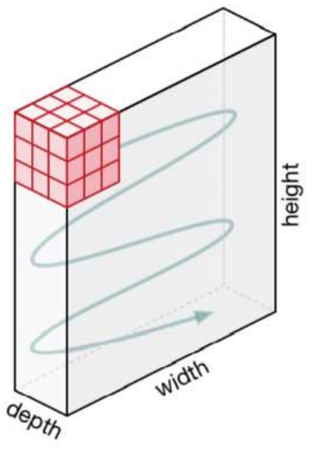


Figure 3.7 Movement of the kernel matrix across an image (Zahirovic [74])

3.1.2.4 Pooling Layer

The pooling layer is used to reduce the spatial size of the feature map obtained from a convolution operation. This helps in speeding up the computation time and ensuring that features of the input image are not lost.

Two types of pooling layers are available in a CNN, they are:

1. Max Pooling.
2. Average Pooling.

Max pooling is the most used pooling layer in CNNs. The operation of max pooling involves selecting the maximum value from a portion of the image covered by the kernel. However, average pooling selects the average values from the bit of the image covered by the kernel. The diagram in Figure 3.8 shows the operations of both pooling layers. In this study, Max pooling has been used due to its high accuracy in image classification tasks [76].

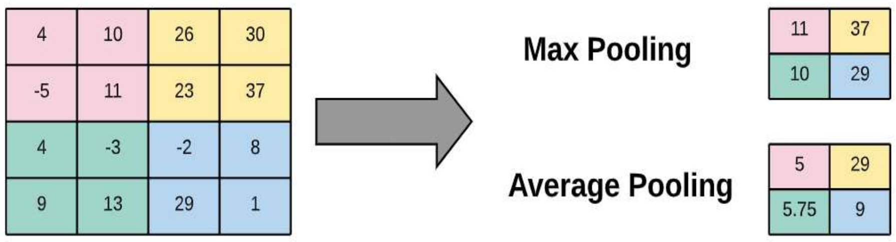


Figure 3.8 Max and Average pooling (Martinez-Soltero et al. [77])

3.1.2.5 Fully Connected Layers

The fully connected (FC) layer is normally located at the end of the CNN after feature extraction has been performed using convolutional and pooling layers. The neurons in this layer are usually fully connected to the neurons in the preceding layer. They are used for making predictions by the network and collecting the final nonlinear combinations of features.

3.1.3 Activation Functions

Activation functions take an input which is usually a single number and perform mathematical operations on this input. They help in deciding if a neuron will fire or not and by doing this, they influence the output of layers in a CNN.

Table 3.2 Activation functions for CNN (Verdhan [78])

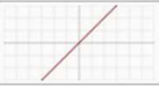
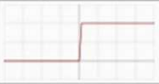
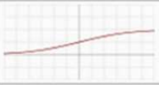
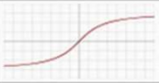
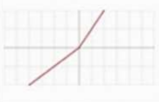
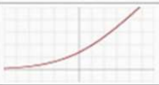
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x \neq 0 \\ ? & \text{for} \quad x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Table 3.2 contains details of commonly used activation functions such as ReLU, Tanh and Linear. In this study, the linear activation function which is also known as the identity function was used in the CNN architecture implemented. It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input.

3.2 Deep Learning Approach

The approach used in this study is to establish a CNN model, train the model and then use the model to reconstruct cross-sectional slices of a burner flame by taking 2-D projection data as input into the model. This is a similar approach to that discussed in section 2.3. To establish an optimal solution for the proposed DL-based 3-D flame reconstruction system, several experiments were conducted to determine the required CNN hyperparameters.

In the following section, the CNN architecture used in this study is presented and the procedures used in implementing, training, and measuring the performance of the CNN model are discussed.

3.2.1 Design and Implementation CNN Network Model

The proposed CNN model architecture is based on the use of seven convolutional layers in the hidden layers. The convolutional layers are used in extracting features from the input projection data. The choice of seven convolutional layers was determined by experimenting with the different number of layers.

For the proposed solution, the use of seven convolutional layers provided the best results in terms of reconstruction accuracy. The diagram in Figure 3.9 shows the architecture of the proposed CNN model.

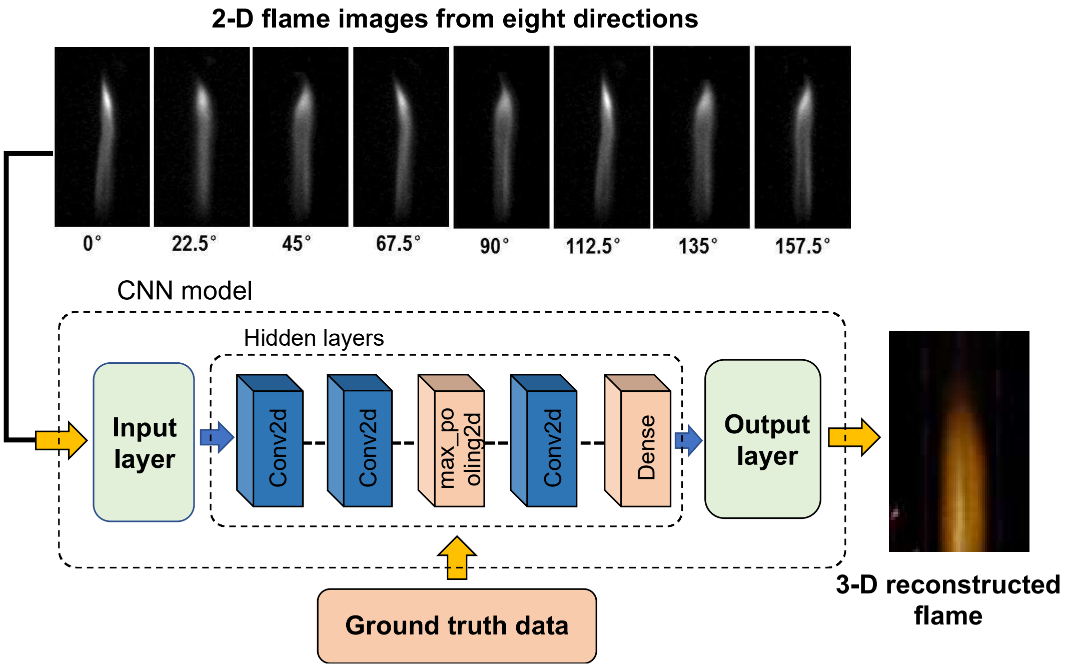


Figure 3.9 Proposed CNN model

To implement the CNN model, the Keras Conv2D class [79] was used. This class included some important parameters that could be tuned during the training process. The purpose of this layer is to create a convolution kernel that is convolved with the input of the layer to produce a tensor of outputs [80]. This section now describes the configuration used in the implementation of the Conv2D layers and some other CNN model parameters. An example of the Conv2D layer code implementation using Keras is shown in Appendix 7.

The first parameter required by Conv2D is the number of filters to be learned by the convolutional layer. Early in the network architecture (i.e., closer to the actual input image) convolutional filters are learned less frequently, while later layers (i.e., closer to the output predictions) learn more frequently [81]. For this study, following the experimentation of various filter values, the approach used was to start with a small filter size of 8 in the first Conv2D layer and increase the values using powers of 2. The range of the filters used was [8, 16, 32, 64] with the final layer having a filter size of 64. However, for the CNN model used in the numerical simulation, the filter size was fixed at 32 for all layers. The reconstruction of a 3-D bivariate Gaussian distribution plot using this approach produced satisfactory results during the reconstruction process.

The next Conv2D parameter is the filter_size parameter. It determines the kernel dimensions in Keras. Common kernel dimensions are 1x1, 3x3, 5x5, and 7x7 and this can be passed as (1, 1), (3, 3), (5, 5) or (7, 7) combinations of tuples in Keras. A 3x3 kernel size is used in this study as it is the most popular choice, and it is being used by every DL practitioner out there [82]. The results obtained using the kernel size were satisfactory. Also, larger kernel sizes could increase the complexity of the network model and the network model training time.
The strides parameter is another Conv2D parameter which specifies the "step" the convolution takes along the x and y axes of the input volume, it is a two-tuple of integers. The default value of (1,1) was used for the value of stride for the implementation of the CNN model. The use of this default value implied the following:

- A convolutional filter is applied to the current location of the input volume.
- As the filter moves one pixel to the right, the input volume is once again filtered.
- The process is repeated until the far-right border of the volume is reached, at which point the filter is shifted one pixel down and restarted from the far-left border.

Keras Conv2D supports two padding types: valid and same. The valid parameter prevents zero padding of the input volume. If a value is set to "same", zeros are evenly distributed across both sides of the input, as well as up and down both sides of the input. For most of the layers in this CNN model, this value was set to “same,”. Three max pooling layers were employed in between the convolutional layers to reduce the spatial size of the feature map that was obtained from a convolutional layer.

According to section 3.1.3 of this design, the linear activation function is used in each Conv2D layer. This activation function was chosen because the 3-D flame reconstruction method based on DL was a regression problem. Additionally, the experimentation of several other activation functions confirmed that the linear activation function was well suited to this problem.

For the implementation of the CNN model used in the numerical solution, leaky Relu activations layers were used in between the Conv2D to ensure fast convergence and avoid gradient problems. However, these layers were later removed as they did not seem to affect the performance of the CNN model.

A flattening layer is used at the output of the final max pooling layer to convert the output data from max pooling into a 1-D array for input into the final layer which is a single dense layer. The dense layer in this model is the output of this CNN. Data from this layer is then fed into a 3-D visualisation tool to view the cross-sectional slices and render a 3-D image of the flame.

As a last step, a learning rate and optimization algorithm had to be chosen. In contrast to the classical stochastic gradient descent algorithm that is most commonly used, Adam is a simple but powerful optimization algorithm that can be used to update network weights iteratively based on training data. Due to the recommendation that the Adam optimizer is the best overall choice to use for DL applications [83], this optimizer was used in this design.

For this work, a learning rate of 0.00008 was used for the numerical simulation and 0.001 for the experiment based on real flame data. In addition to using information derived from performance studies in [84], several learning rate values were also experimented with to determine an optimal learning rate value.

3.2.2 Training

The training of a CNN model is based on an iterative process of updating and adjusting the values of the weights of the CNN neurons. The process starts by initializing all weights to small random values and feeding the network with the training dataset of 2-D projection data and its corresponding ground truth data.

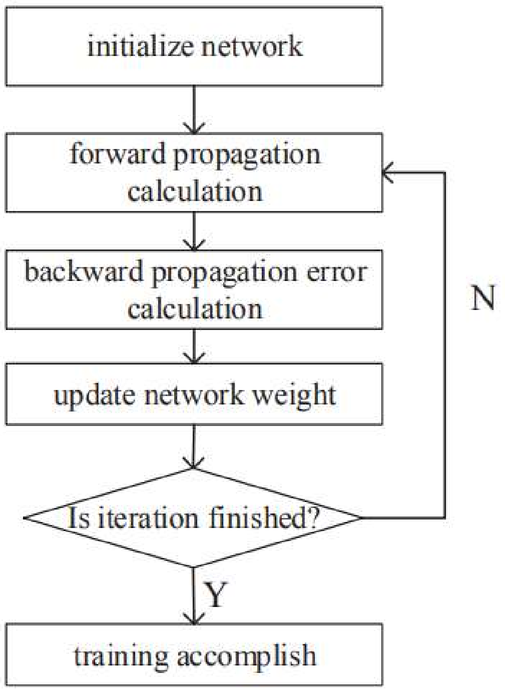


Figure 3.10 CNN training process (Wang [85])

The training then involves a series of forward and backward propagation phases until learning stops. An overview of the training process followed in this study is shown in Figure 3.10.

3.3 Performance Metrics

In this section, the training metrics used during the training and testing of the established CNN model are presented. These metrics help to check how well the CNN model has learned.

Performance metrics have been used to evaluate the quality of the reconstructed image obtained from rendering the data obtained from the CNN model. These performance metrics are discussed in this section.

In this study, the Mean Absolute Error, Mean Square Error, and Root Mean Square Error were used to evaluate the performance of the CNN model. Also, two of the most common and widely used standard metrics in the computer vision and image processing domain are used for comparing the quality of the reconstructed images. These metrics are the structural similarity index Metric (SSIM) and peak signal-to-noise ratio (PSNR).

3.3.1 Mean Absolute Error

The mean absolute error (MAE) is the average of the absolute difference between the predicted and actual values in the dataset.

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}| \tag{3.2}$$

In (3.2), the equation for the MAE is shown, where $\hat{y}$ and y_i is the predicted value and actual value, respectively. N is the total number of data points.

3.3.2 Mean Squared Error (Loss)

The mean squared error (MSE) shown in (3.3) represents the average of the squared difference between the actual and predicted values in the dataset. This is also referred to as loss or MSE loss. The smaller the value, the better as this shows that the CNN model has improved learning during training.

$$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2 \tag{3.3}$$

3.3.3 Root Mean Square Error

The root mean squared error (RMSE) is the square root of the mean squared error. It measures the standard deviation of residuals.

$$RMSE = \sqrt{MSE} \ = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2} \tag{3.4}$$

3.3.4 Structural Similarity Index

The structural similarity (SSIM) index is a common image quality metric used in the image processing community. It evaluates the three visual properties of an image: luminance, contrast, and structure.
The structural similarity can be defined as:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{3.5}$$

Where μ_x is the average pixel value of image x, μ_y is the average pixel value of image y , σ_x is the variance of image x , σ_y is the variance of image y, σ_{xy} is the covariance of image x and image y , c_1 and c_2 are variables to stabilize the division with weak denominator [86]. SSIM index usually has a value that ranges from 0 to 1. A value of 1 implies that the image of the reconstructed image perfectly matches the original one.

3.3.5 Peak Signal-to-Noise Ratio

Peak Signal-to-Noise Ratio refers to the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation [87].

PSNR can be defined as:

$$PSNR = 10 * log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{3.6}$$

where MSE is the mean squared error, and MAX_I is the maximum possible pixel value in the evaluated image. As the MSE value approached zero, the PSNR value approaches infinity; this shows that a higher image quality is obtained when a higher PSNR value is reported [88]

3.4 Machine Learning Toolchain

In this section, a brief description of the ML tools, development environment and framework used in this study will be presented.

3.4.1 Python

Python is an open-source high-level general-purpose programming language which was developed in 1989 by Guido van Rossum [89]. It is a powerful programming language and is one of the most common programming languages used in ML applications as it provides various powerful libraries useful for DL. The version of Python used in this master's thesis is Python 3.7.0.

3.4.2 TensorFlow

TensorFlow is an open-source platform for ML. It has a collection of tools, libraries, and resources in place to speed up the development of ML applications. The APIs provided by TensorFlow are based on both the C++ and Python programming languages [66]. The version of TensorFlow used in this master's thesis is 2.8.0.

3.4.3 Keras

Keras is a high-level framework written in Python commonly used for implementing ML solutions. The framework is easy to use and learn. It provides an interface to TensorFlow and is Python-friendly. Keras was developed to enable faster prototyping of ML solutions [90]. The version of Keras used in this master's thesis is 2.9.

3.4.4 Google Colaboratory

Google Colaboratory, or "Colab" for short, is a cloud computing product from Google Research. Colab enables developers to write and execute Python code through a web browser. It achieves this using a hosted Jupyter notebook service. Colab enables developers to write and execute Python code through a web browser. It achieves this using a hosted Jupyter notebook service. The product also provides access to computing resources including CPUs, GPUs and TPUs [91].

The table (Table 3.3) shows the basic Colab CPU hardware specification which was used during the generation of additional datasets. However, for training the CNN model implemented in this study, a hardware configuration based on the use of on Tensor Processing Unit (TPU) was used. TPUs are hardware acceleration units designed by Google to accelerate ML applications programmed with TensorFlow [92].

Table 3.3 Google Colaboratory hardware specifications (Kegenbekov et al [93])

Cloud Computer Hardware Specifications	
CPU Model Name	Intel(R) Xeon(R)
CPU Frequency	2.30 GHz
GPU Model Name	Nvidia K80
GPU Performance	4.1 TFLOPS
GPU Memory Clock	0.82 GHz
No. CPU Cores	2
CPU Family	Haswell
Available RAM	12GB
Disk Space	25GB

The TPU board has 64 GB high bandwidth memory and 180 teraflops of floating-point performance, Figure 3.11 shows an image of the TPU used in the Colab setup. The use of the TPU helped in reducing the time taken to train the implemented CNN network model.

Figure 3.11 Goggle TPU with four cores [94]

3.5 Image Processing and 3D Visualisation Tools

In this section, the image processing tools used to generate and prepare experimental datasets, as well as for the visualisation of the 3-D structure of the reconstructed flame are presented.

3.5.1 ImageMagick

ImageMagick is free and open-source software provided in either binary or source code format. It is used for displaying, creating, converting, editing, and modifying raster images. The program was invented in 1987 by John Cristy. It can read and write more than two hundred image file formats. It consists of image manipulation utilities that can be accessed via a command-line interface. While ImageMagick lacks the robust graphical user interfaces of Adobe Photoshop and GIMP to edit images, it does come with a basic native X Window GUI (called IMDisplay) for rendering and manipulating images on Unix-like operating systems, as well as API libraries for a range of programming languages. Image file formats are identified by magic numbers in the program [95]. The version of ImageMagick used in this master's thesis is 7.0.11-9.

3.5.2 Slicer 3D

Slicer 3D is an image-computing software application that allows for the visualisation and analysis of medical image-computing datasets in 2-D, 3-D, and 4-D. This software application supports all commonly used datasets, including images, segmentations, surfaces, annotations, transformations, etc. Visualisation is available on a desktop and in virtual reality. The software also has an integrated Python console as well as the ability to function as a Jupyter notebook kernel with remote 3-D rendering features [96]. The version of Slicer 3D used in this master's thesis is 4.11.20210226 r29738 / 7a593c8.

3.6 Summary

In this chapter, the principles of DL and CNN have been described and discussed. Although various DL models were described, the CNN model has been used in this study due to its performance in computer vision and image processing applications. A prototype 3D flame reconstruction system based on DL has been designed and implemented using a CNN model

The various parts of the CNN-based network model have been described in detail explaining its characteristics and functionality. The implementation of the CNN model was based on the use of 2-D convolutional layers. The reasons for choosing the configuration and parameters used in the implementation of the CNN model have also been addressed. This was accomplished by training and evaluating the implemented network model using several parameter settings and configurations. Additionally, optimal values were determined based on recommendations for using these parameters in DL applications

A description of the training process for the CNN model and the performance metrics required for analysing and evaluating the accuracy of the established CNN model has also been presented. For this study, the CNN model was evaluated using MAE, MSE, and RMSE. For evaluating the reconstructed images, two of the most widely used metrics in computer vision and image processing are used. This was the SSIM and PSNR.

In addition, a description of the software development environment and the ML framework utilized in this study was provided This also includes details of the image processing tools used to create and prepare experimental datasets as well as to visualize a 3-D reconstructed image.

4

Numerical Simulation

4.1 Introduction

A series of experiments was conducted using numerical simulations. These simulations were used to create two cases of phantoms which were designed to mimic a 3-D conical single flame. The images obtained were used in training a DL network model. The objectives of the experiments were:

- to create experimental image datasets from numerical simulations.

- to establish a DL model for the 3-D volumetric reconstruction of a 3-D image which was like a 3-D conical single flame.

- To evaluate and validate the established DL model.

This chapter gives a detailed description of the numerical simulation, dataset collection and preparation, and reconstruction results obtained using the established network model described in Chapter 3.

4.2 Numerical Simulation

To investigate the performance of the established CNN-based network model for the 3-D volumetric reconstruction of flames, simulative studies have been conducted in this section. A multivariate Gaussian distribution 3-D plot using Python and Matplotlib was used to create ground truth images which had a similar shape to a flame.

The multivariate Gaussian distribution is defined as:

$$p(x;\mu,\Sigma) = \frac{1}{\sqrt{(2\pi)^n|\Sigma|}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right) \tag{4.1}$$

Where x is an n-dimension vector, μ is the n-dimensional mean vector and Σ the n x n covariance matrix [97]. For this study, the bivariate Gaussian distribution was used, and this was the case when $n = 2$.

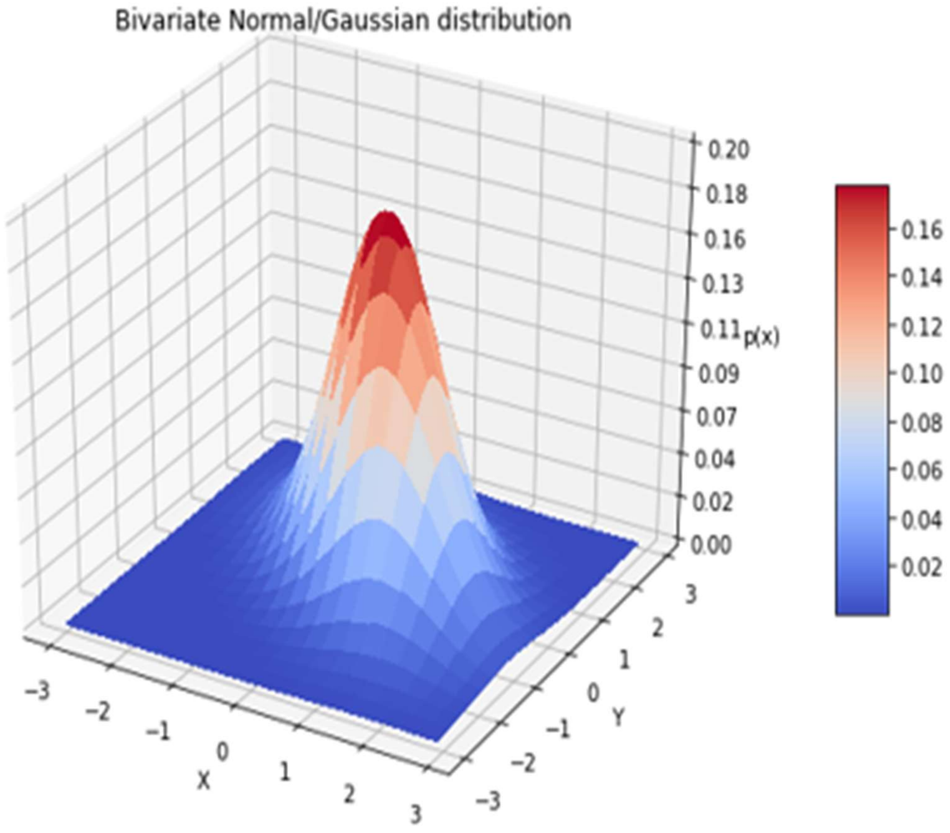


Figure 4.1 3-D surface plot using bivariate Gaussian distribution

The plot in Figure 4.1 is an example of a bivariate Gaussian distribution plot. The ranges (-3 to 3) for the X and Y axis of the plot are constructed using the meshgrid function from NumPy.

For the numerical simulation, a uniformly distributed set of random numbers in the range from 0.1 to 0.2 were generated for the mean vector (μ). This was done to slightly alter the samples generated. The data for the Z domain was then acquired using a bivariate Gaussian distribution density function. This was used to produce the Gaussian range over the Z-axis. The figure below (Figure 4.2) is an example of a grayscale ground truth data image used as a dataset for training the CNN-based network model implemented in this study.

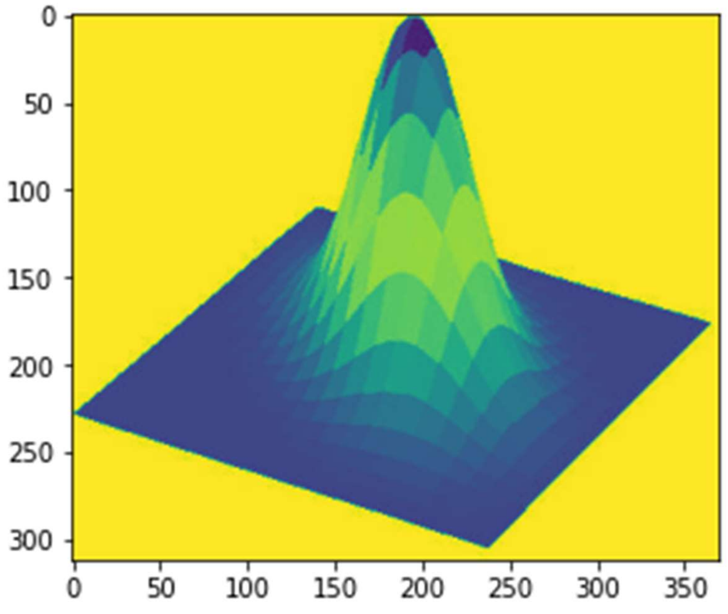


Figure 4.2 Greyscale ground truth data image

The diagram in Figure 4.3 below shows the process used in this numerical simulation experiment for generating datasets, training, testing, and evaluating the CNN-based network model established for the 3-D volumetric reconstruction of a 3-D image which was like a 3-D conical single flame.

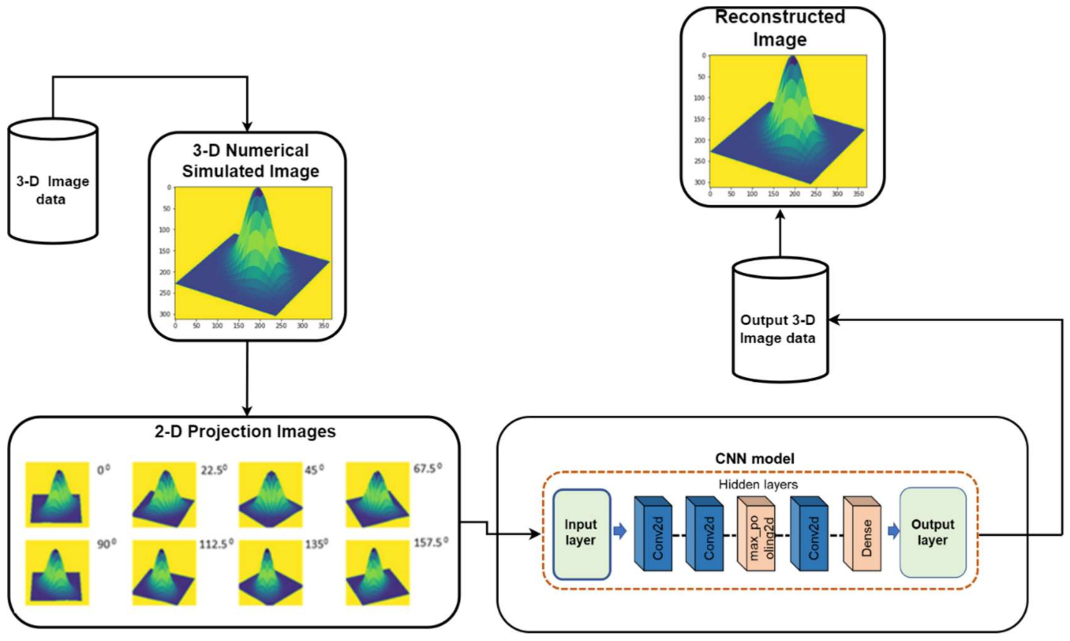


Figure 4.2 Numerical simulation process

In the following sections, the steps involved in the above process such as dataset collection and preparation, network model training and evaluation are discussed.

4.3 Data Collection and Preparation

2-D images-based projection datasets were generated from the numerical simulated phantom and this dataset was used in training the process. In this section, the collection and preparation of the datasets used in training the proposed CNN-based network model are discussed.

4.3.1 Ground Truth Dataset

The ground truth data used in training the network model is a collection of images obtained from the numerical simulation process using bivariate Gaussian distribution. Ground truth data were generated under two sets of conditions. This meant two sets of ground truth samples were generated. In each of these conditions, the parameters of the bivariate Gaussian distribution were altered using constrained random values during the generation of the ground truth samples. This helped in ensuring that the samples for the ground truth dataset did not all have a similar SSIM value.

Table 4.1 Generated ground truth dataset

Ground Truth Sample	**Number of Samples**	**SSIM (Mean)**	**SSIM (Standard deviation)**	**MSE (Mean)**	**MSE (Standard deviation)**	**Total Number of Samples**
A	1000	0.9	0.13	1088	1853	
B	1000	0.8	0.16	3031	3073	
						2000

The average SSIM value for Sample A was 0.9 and 0.8 for Sample B. Images in Sample A and B were all slightly different from each other. Table 4.1 above shows the ground truth dataset prepared for this experiment. The total number of ground truth samples used in this experiment was 2000.

The original dimension of the ground truth image obtained from the numerical simulation was (180,60). However, to match the specification of the final dense layer in the output of the established CNN model, this ground truth data was converted into a 1-D vector with a dimension of (10800). The data shape of the output ground truth data is shown in Table 4.2

4.3.2 Projection Dataset

The projection dataset used in this experiment was generated using ground truth data obtained from the numerical simulation. Using ImageMagick, a software tool for displaying, creating, converting, modifying, and editing raster images [95], code was written to capture 2-D projection images from the numerically simulated image.

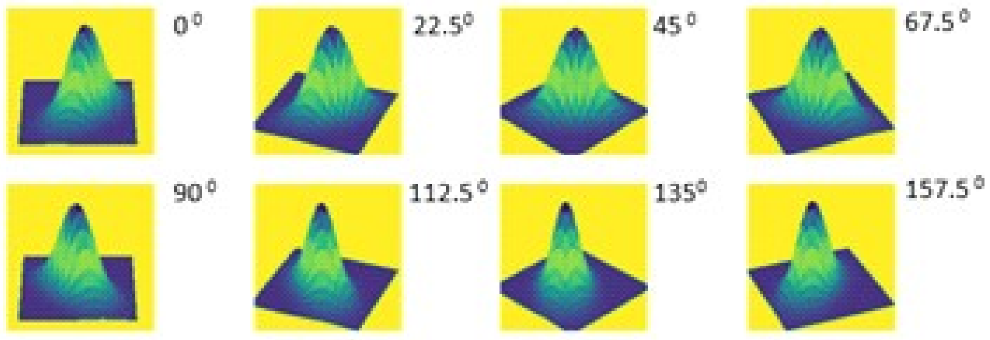


Figure 4.3 Sample of 2-D projection data images

2-D images of the ground truth data were taken from 8 angles between 0^0 and 180^0 (Figure 4.3). Using this approach, a total of 1000 projection dataset samples were generated for each set of ground truth samples. This data was then used in training, testing, and validating the established network model.

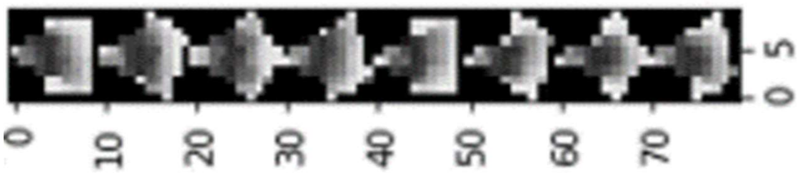


Figure 4.4 Input image data based on eight 2-D projections.

To resolve out-of-memory issues in the development environment, the dimensions of the projection data image were reduced from (216, 288) to (10,10). This produced input image data with a dimension of (8,10,10). Where eight was the number of projections and (10,10), the size of each projection data. This input image shown in Figure 4.4 was used as the input data for the CNN model.

Table 4.2 Numerical simulation dataset composition

Data Spilt	**Usage**	**Output Data Shape**	**Input Data Shape**
60%	Training	(600,10800)	(600, 80,10)
20%	Validation	(200,10800)	(200, 80,10)
20%	Testing	(200,10800)	(200, 80,10)

The dataset was also split into three parts: training, validation, and testing. This was done to ensure that there was data available for carrying out the performance evaluation of the network during training. 20% was set aside for validation during training, 60% was used to train the network model and 20% was for testing after training the model. Table 4.2 shows the composition of the dataset used for training the established network model.

4.4 Model Establishment

A series of tests were performed using several convolutional layers and different hyperparameter values to determine the best architecture for the solution. The CNN model was then implemented using seven 2-D convolutional layers with each having a linear activation function. The output of the CNN model consists of a flattened layer and a dense layer. The network model is described in detail in section 3.2.1.

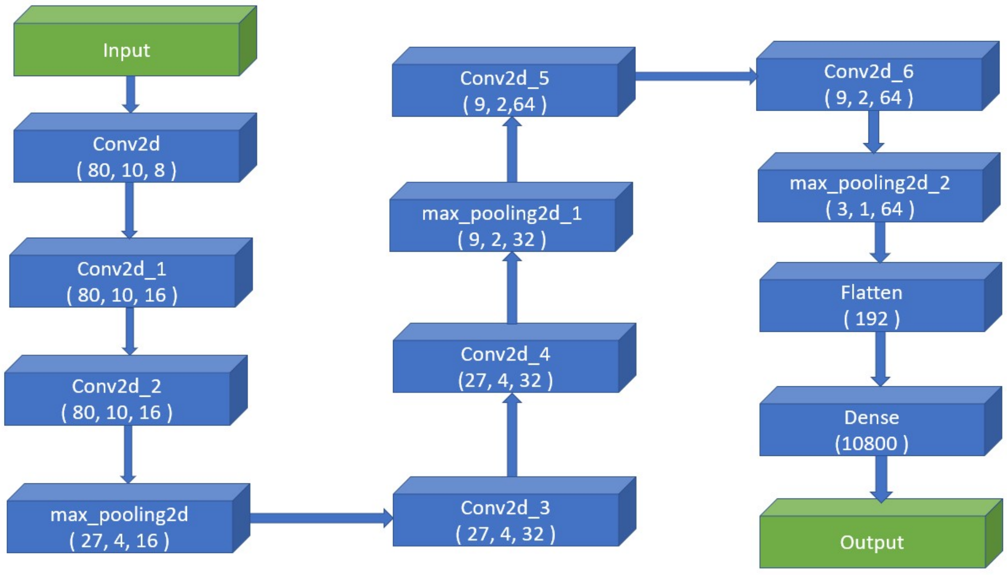


Figure 4.5 Overview of numerical simulation CNN model architecture

An overview of the actual network architecture used for this experiment is shown in Figure 4.6. The input dimension shown in the figure for the first Conv2D layer of the network was (80, 10) which was derived from the 2-D projection dataset dimension of (8, 10, 10). The dense layer output dimension of (10800) was derived from the ground-truth dataset dimension of (180,60). The code snippet for implementing this CNN model is shown in Appendix 4.

Following the implementation of the CNN model, the next phase involved was training the network model. For training the network model, the training dataset was shuffled to mix up the samples, and the network was trained with a variety of hyperparameters and iterations to determine which one provided the best performance.

For this work, a learning rate of 0.008 and a batch size of 64 were used during the training of the network. The network was trained for 100 epochs. As both training and validation losses did not seem to reduce further at this point, it implied that the model has reached a steady state and is no longer learning. Therefore, 100 was selected as this epoch value for training the network model.

The CNN model was trained using the Google Colab high-performance computing cloud-based hardware. The hardware accelerator used during the training was the TPU. The hardware specifications of the Google Colab are described in section 3.4.4.

The plots in Figure 4.7 (a) show the training and validation loss across the training time for all three sample datasets used in this study. The horizontal axis on the plot represents time in epoch units. A complete iteration through the training dataset is referred to as one epoch. The vertical axis represents the output of the loss function. The loss function is the MSE and the ADAM optimizer [83] is used. This is described in more detail in Section 3.

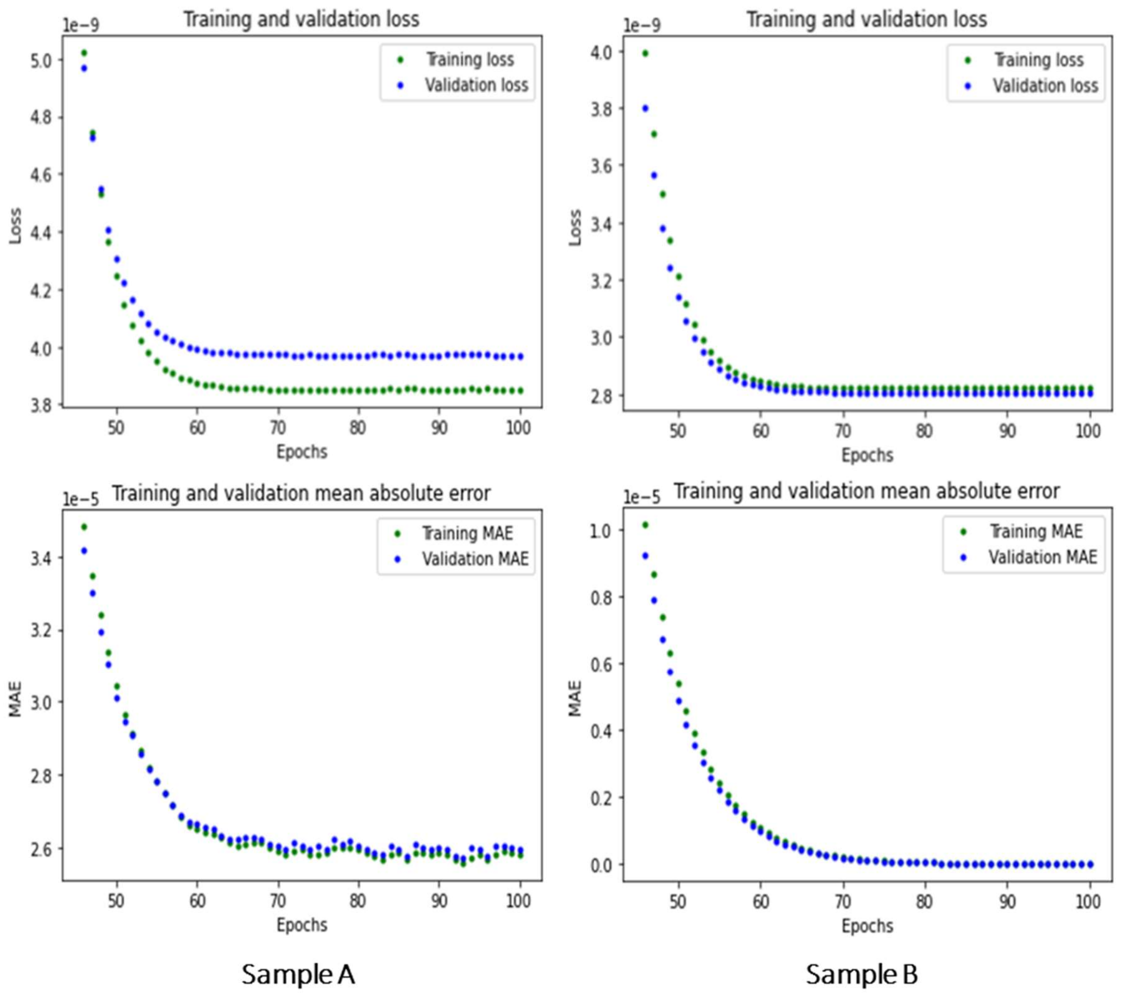


Figure 4.7 CNN Network model training metrics

From the plot, the training and validation loss decreases rapidly over the first 60 epochs, before flattening out. Also, comparing the value of the first and last epoch, the network has improved during training [98].

For Sample A, the validation loss is slightly higher than the training loss. This could indicate that the model is overfitting the training data. Generally, overfitting occurs when a model matches the training data too closely, to the point where it memorizes noise or other irrelevant patterns in the training data which are difficult to generalize to unknown, new data. The model may perform well on the training data, but poorly on the validation or test data since it has essentially memorized the training data instead of learning generalizable patterns. It is also possible that the model is still generalizing well to new data, but that the difference may be due to random fluctuations or noise in the training data

Also, the validation and training MAE drops to an exceedingly small value of ~ $2.6e^{-5}$ after about 70 epochs for Sample A, whilst that of Sample B drops to zero after about 70 epochs. In general, this is a good sign, as it indicates that the model is becoming more accurate as it is trained. As MAE decreases, the model is better at predicting the target variable, which can be used to measure performance.

4.5 Model Evaluation

4.5.1 Numerical Simulation Reconstruction Results

The SSIM metric was used to compare the similarity of the reconstructed image with that of the ground truth image. The images in Figure 4.8 show the reconstructed and the original ground truth images. From Table 4.3, the image reconstructed with Sample A trained CNN model reported the highest SSIM value.

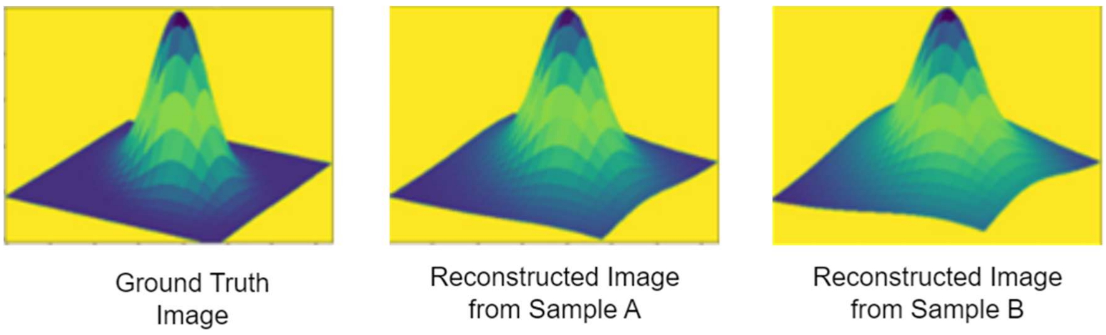

Ground Truth Image
Reconstructed Image from Sample A
Reconstructed Image from Sample B

Figure 4.8 3-D reconstruction accuracy

Table 4.3 Performance metrics

Ground Truth Sample	**Test MAE**	**Test Loss**	**SSIM**	**MSE**
A	2.70E-05	4.25e-09	0.76	1172.09
B	2.78E-05	2.83e-09	0.71	2729.56

According to the performance metrics reported in Table 4.3 for Samples A and B, both samples have low mean absolute errors (MAEs) and test losses, which indicates that the model is performing well. MAE measures the average absolute difference between predicted and actual values. The smaller the MAE, the more accurate the predictions. The MAE for Sample A is slightly lower than the MAE for Sample B in this case, which indicates that Sample A's predictions are slightly more accurate than Sample B's predictions.

Test loss is a measure of the difference between predicted and actual values for the test dataset. Having smaller test losses indicates that the model performs better on the test dataset. Sample B has a slightly lower test loss than Sample A, which suggests that the model performs slightly better on Sample B's test dataset than Sample A's.

In summary, the results show that Sample A and Sample B perform well when it comes to these specific samples, but further analysis and testing are needed to evaluate the model's generalizability and consistency. However, these results demonstrate that the established CNN network model could be used to reconstruct 3-D flames.

The time taken to reconstruct the 3-D bivariate Gaussian distribution using the implemented CNN was 0.08 seconds. This was achieved by running the test on the TPU hardware within Google Colab. Compared to iterative methods such as ART, and SART, this reconstruction time was faster. As an example, Jin et al. [48] reported a reconstruction time of 170.25 seconds using ART.

However, as discussed in section 2.5, other flame reconstruction methods differ in flame structure, dataset, hardware platform, code optimization, and DL model architecture. As a result, direct correlation may be difficult.

4.6 Summary

Various experiments have been conducted using a numerical simulation and CNN-based network model to evaluate the effectiveness of using DL in reconstructing numerically simulated images that had a similar shape to a 3-D flame structure. A detailed description has been provided of the design, implementation, and training of the CNN-based network model in detail.

In addition, detailed descriptions of the generation and preparation of the experimental datasets have been provided. Experimental ground truth datasets which had a similar shape to a 3-D flame structure were generated using a mathematical model in the form of a 3-D bivariate Gaussian distribution plot. This was achieved using Python programming language and third-party libraries which included Matplotlib.

In the experiments conducted, the ground truth datasets were generated under two different conditions and, for each of these conditions, the numerical simulation variables were manipulated in such a way that all the ground truth datasets in each sample had a similar SSIM value. Using this approach, 2 samples of ground truth datasets were made available for the experiment. They were samples A and B. The average SSIM value for Sample A was 0.9 for Sample B and 0.8.

The experiment consisted of 2000 ground truth datasets with 1000 datasets for each sample. To generate the projection dataset, code was implemented using ImageMagick APIs to create eight 2-D projections from the 3-D bivariate Gaussian distribution-based ground truth image. These projections were obtained from 8 angles between 0^0 and 180^0. For each set of ground truth samples, 1000 projection datasets were generated.

A CNN model based on seven 2-D convolutional layers with a linear activation function was established following a series of tests using different convolutional layers and hyperparameter values. Following the training of the established network model, the 3-D bivariate Gaussian distribution was successfully reconstructed using eight 2-D projection data. The network model performance was verified under two sets of ground truth samples. The obtained results have proven that the implemented and trained network model can reconstruct the 3-D bivariate Gaussian distribution with good accuracy and structural similarity.

5
3-D Reconstruction of Flames

5.1 Introduction

A series of experiments was conducted to generate experimental flame images. The flame images were then used in training a DL network model. The objectives of the experiments were:

- to generate experimental flame image datasets.
- to investigate the 3-D flame reconstruction accuracy of the developed DL network model (described in Chapter 3).
- to investigate the performance of the developed DL network model under a range of combustion conditions.

The data used to generate experimental flame images was obtained from the laboratory-scale combustion test rig (Figure 5.1) in the Instrumentation Research Laboratory, at the University of Kent. The data was originally collected for a research study on 3-D reconstruction and characterisation of fossil-fuel-fired burner flames [24].

A description of the laboratory-scale combustion test rig is presented. Subsequently, the dataset collection and preparation are described. This is then followed by the network model training procedure. Finally, the reconstruction results are discussed.

5.2 Laboratory-scale Combustion Test Rig

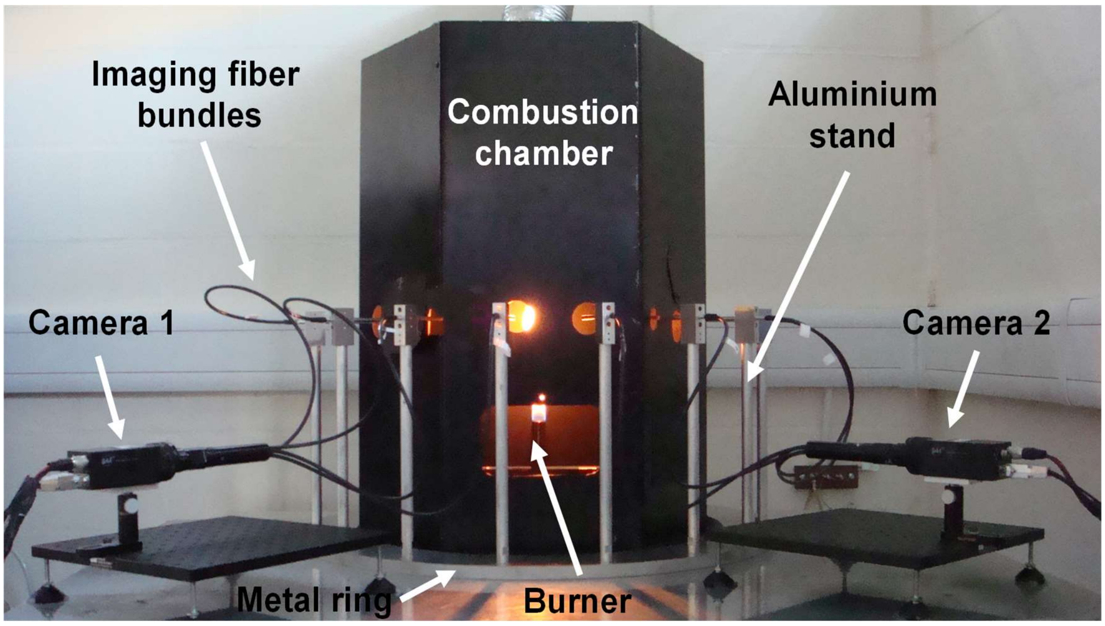


Figure 5.1 Laboratory-scale combustion test rig

The combustion rig shown in Figure 5.1 was used in this study. The rig was modified to allow the imaging system to access flame from eight different directions, it also had a new burner designed for the generation of a stable and luminous flame.

The rig consists of a hexagonal chamber, and a metal circular ring with eight aluminium holding stands placed around the combustion chamber. A single burner is positioned vertically at the centre of the chamber and eight adjustable mechanisms were designed and fixed on the top of the stands for holding the fibre bundles. A detailed description of the rig can be found elsewhere in [24]

5.3 Data Collection and Preparation

Flame datasets generated under six different combustion conditions were used in training the network model. This was to establish a robust network model. The flame images are shown in Figure 5.2 under six different combustion conditions.

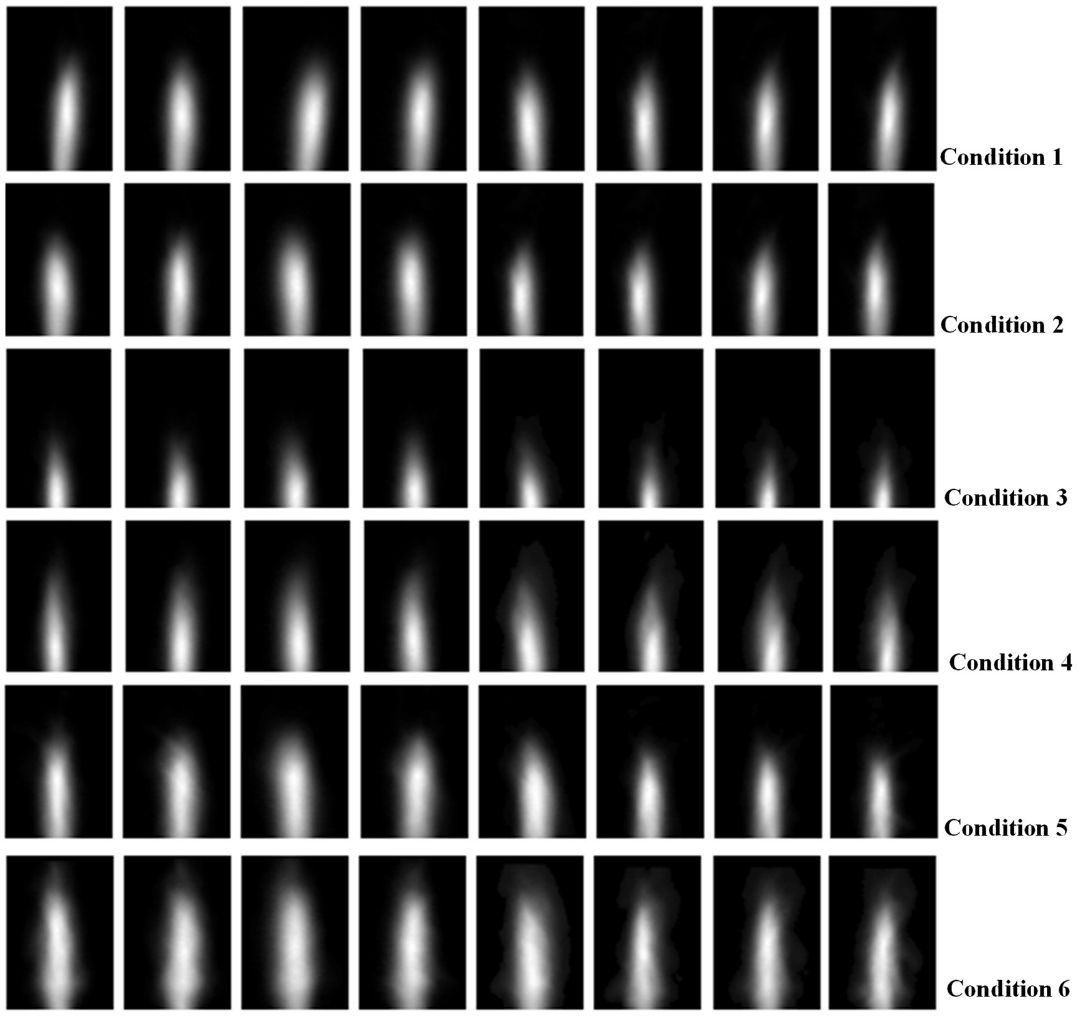


Figure 5.2 Flame images generated under six different combustion conditions

To train the network model, a dataset based on only six flame conditions is not sufficient. DL network models learn to model patterns in underlying data [98]. To help the network learn more effectively, additional datasets were generated using the initial set of flame conditions collected from the test rig. In this section, the process used in generating and preparing the projection dataset for use in training the network model is presented.

The following table (Table 5.1) outlines the flame conditions used in the study. The conditions were obtained by using different fuel flow rates on the combustion test rig

Table 5.1 Flame conditions

Flame Condition	**Fuel flow rate (L /m)**
1	0.2
2	0.3
3	0.4
4	0.5
5	0.6
6	0.7

5.3.1 Projection Dataset

The projection dataset contained six samples of 2-D laminar diffusion flame images obtained under different combustion conditions (Figure 5.2). The number of images in each sample was eight. These images were obtained from eight different angles.

To generate additional projection datasets, morphological transformations were performed on the original 2-D images. Morphology is a technique used in image processing which is based on the shape and form of objects. These methods apply a structuring element to an input image, creating an output image of the same size [99]. A structuring element is a matrix that identifies the pixel in the image being processed and defines the neighbourhood used in the processing of each pixel [100].

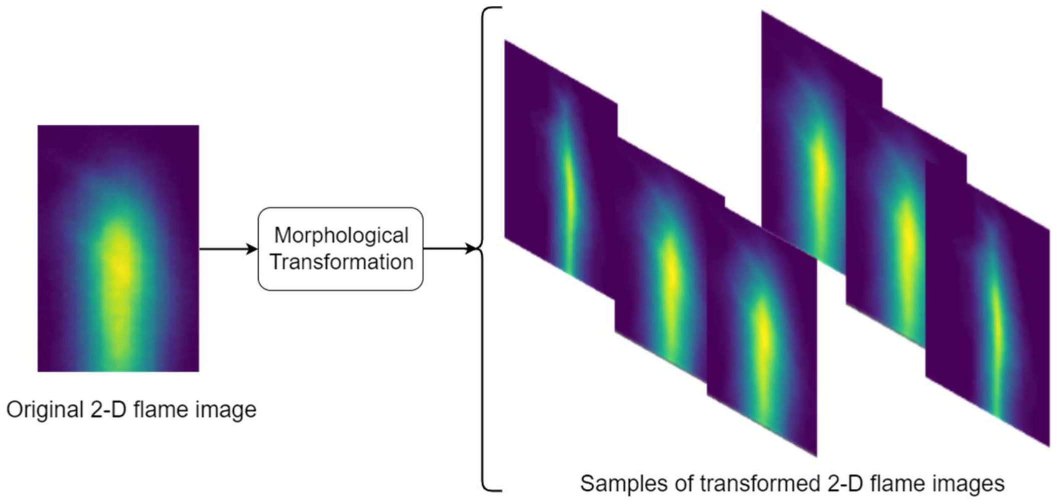

Original 2-D flame image
Morphological Transformation
Samples of transformed 2-D flame images

Figure 5.3 Generating additional datasets using OpenCV Morphological transformation operator

One of the most basic operations used for morphological transformations is erosion. Erosion removes pixels on object boundaries. The number of pixels added or removed from the objects in an image depends on the size and shape of the structuring element used to process the input image. Morphological erosion removes floating pixels and thin lines so that only substantive objects remain [101].

OpenCV morphological transformation operator Erosion was used in transforming the shape of the original projection images. Details of the OpenCV operation can be found in the OpenCV documentation on morphological transformations [102]. The output of this process was a set of projection images which were slightly like the original input image passed into the morphological transformation operator (Figure 5.3). This operation was performed on all the 2-D flame images obtained from the laboratory-scale combustion test rig. To obtain a wide range of images, random values were used in the Open CV Erosion operator for both the kernel and iteration values.

Using this approach, a total of 100 projection dataset samples were generated for each condition. This included ten samples of the original 2-D flame images obtained for each condition. These samples were added to help in ensuring that the dataset contained images with different shapes and structures. In total 600 dataset samples were made available for training, testing, and validating the established network model.

The table below (Table 5.2) shows the number of flame images used in the dataset preparation. For each projection dataset sample, there were eight projection images. The dimensions of each 2-D image in the projection dataset were (8, 145, 90). Where 8 was the number of projections and (145,90), the size of each projection data. Hence, with a dataset of 600, these provided a total of 4800 2-D images for the training and evaluation of the network model.

Table 5.2 Flame images used in dataset preparation

Combustion conditions	**Original images**	**Transformed images**	**Number of images from each combustion condition**	**Total number of images in the dataset**
1	10	90	100	
2	10	90	100	
3	10	90	100	
4	10	90	100	
5	10	90	100	
6	10	90	100	
				600

T**able 5.3** Dataset composition

Data spilt	**Usage**	**Output data shape**	**Input data shape**

60%	Training	(360,26100)	(360, 200,25)
20%	Validation	(120,26100)	(120, 200,25)
20%	Testing	(120,26100)	(120, 200,25)

Also, the dataset was split into three parts: training, validation, and testing. This was done to ensure that there was data available for carrying out the performance evaluation of the network during training. 20% was set aside for validation during training, 60% was used to train the network model and 20% was for testing after training the model. Table 5.3 shows the composition of the dataset used for training the network model.

5.3.2 Ground Truth Dataset

The ground truth data used in training the network model is the collection of cross-sectional slices of the reconstructed flame data. This data was generated using techniques described in Chapter 3. The algorithm used in generating ground truth data was the SART algorithm (Figure 5.4).

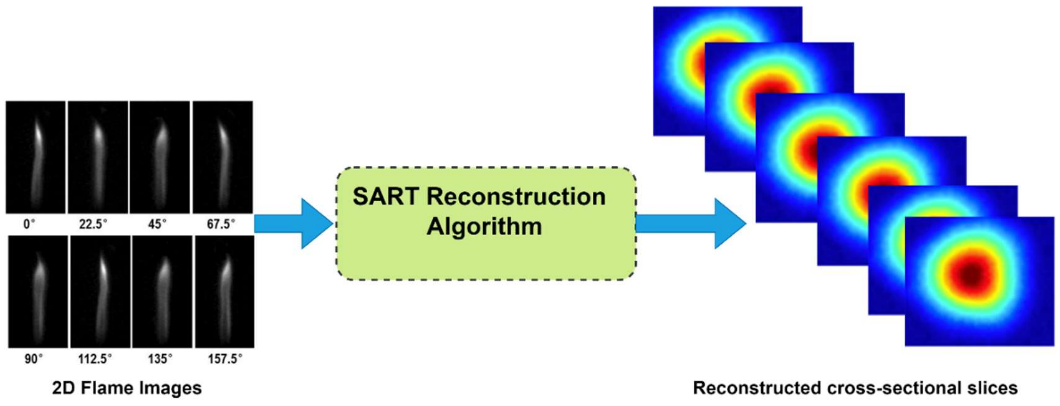

0°
22.5°
45°
67.5°
90°
112.5°
135°
157.5°
2D Flame Images
SART Reconstruction Algorithm
Reconstructed cross-sectional slices

Figure 5.4 Ground truth generation using SART

In total, 600 ground truth dataset was generated. The dimension of each ground truth dataset was (145,90, 90). For each dataset, there are 145 slices each containing a 2-D image with a dimension of (90, 90).

Prior to training the network model, the training dataset was shuffled to mix up the samples and the 2-D projection data images were reduced to a size of (8, 25, 25) and ground truth data reduce to (145, 45, 40). This was done to reduce the size of input data to the network model. To determine the hyperparameters that provided the best performance, the network was trained initially using different values and iterations. For this work, a learning rate of 0.001 and a batch size of 16 were used during the training of the network. These figures reported the best performance results following training [84].

5.4 Model Establishment

The network model implemented is composed of seven 2-D convolutional layers with each having a linear activation function. The output consisted of a flattened layer and a dense layer. The code snippet for the implementation of the CNN model is shown in Appendix 3.

The network model is described in detail in section 4.3.3, however, an overview of the actual network architecture used for this experiment is shown below in Figure 5.5. The input dimension shown in the figure for the first Conv2D layer of the network was (200, 25) which was derived from the 2-D projection dataset dimension of (8, 25, 25). The dense layer output dimension of (261,000), was derived from the ground-truth dataset dimension of (145, 45,40).

The network is implemented in Python using Keras under the TensorFlow framework. The network was trained for 1000 epochs. This value was chosen as both training and validation loss did not seem to reduce further at this stage. This implied that the model has reached a steady state. The time taken to train the network was 2 hours on the Google Colab high-performance computing cloud-based hardware. The hardware accelerator used during the training was the TPU. The hardware specifications of the Google Colab are described in section 4.4.3.

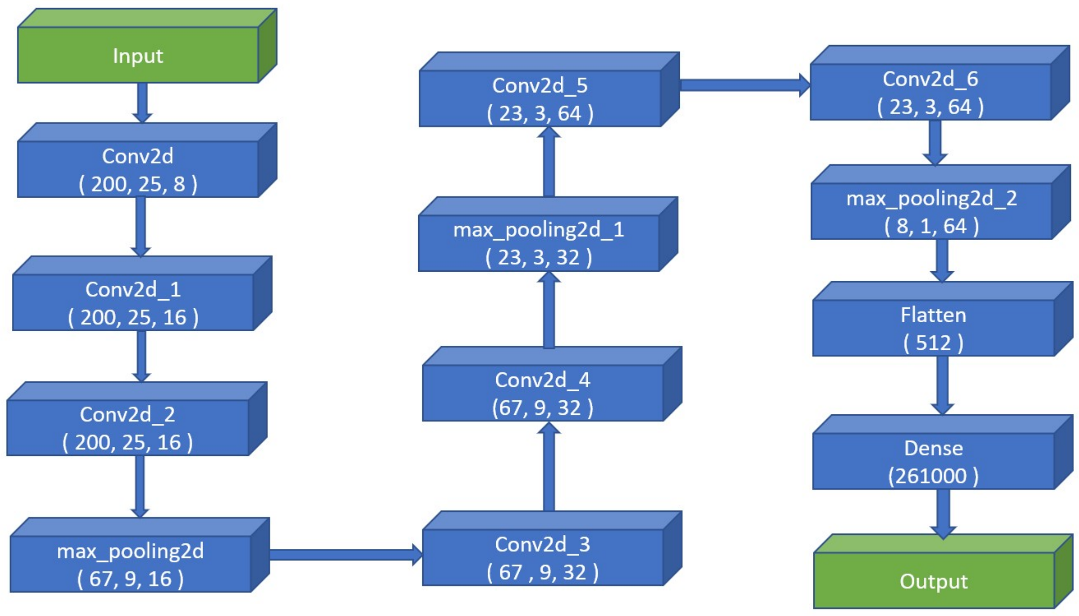


Figure 5.5 Overview of the implemented network architecture for 3D flame reconstruction

During the training, the weights and biases in the network model are adjusted in every iteration until the error is minimized, and validation is used to give an estimate of the network model performance during training, Finally, the model is evaluated with previously unseen data. This helps in detecting overfitting.

The plot in Figure 5.6 (a) shows the training and validation loss. This vertical axis represents the output of the loss function, and the horizontal axis is the time in epoch units.

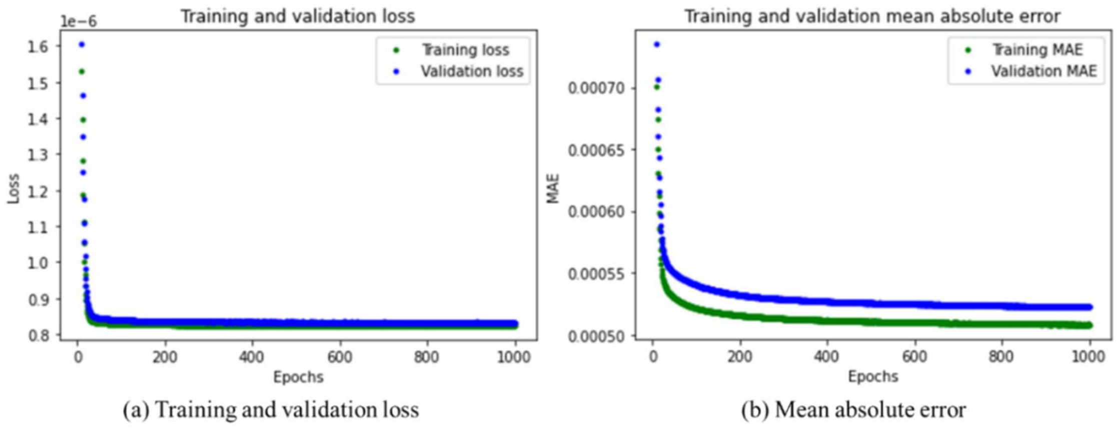


Figure 5.6 Network model training metrics

The smaller the loss value the better the network model has learnt patterns in the underlying data. Following a comparison of the first and last epoch during the training, the network has improved during training, going from a loss of ~ 1.6e^{-06} to a smaller value of 0.8e^{-06}.

The plot in Figure 5.6 (b) shows the MAE during training and validation. The MAE is described in Chapter 4. The MAE decreased over time for both the training and validation. The final training MAE is 5.0e-04 and that for validation is 5.2e-04.

5.5 Model Evaluation

This section evaluates the CNN model performance according to the performance metrics described in Section 3.3. The reconstruction of flame sections and the 3-D volumetric reconstruction of the flame are also discussed. In conclusion, the CNN model's 3-D flame reconstruction time is discussed.

In several reconstructed slices, there was insufficient data to allow the 3-D visualisation tool described in section 3.5.2 to produce a 3-D representation of the reconstructed flame. As a result, a 3-D image of the reconstructed flame could not be created using these slices. This was particularly evident at the top of the flame and in some slices just above the middle of the flame section.

As a result, only the reconstruction results of the bottom 30 slices of the flame have been considered in this analysis. In the following figure (Figure 5.7), 44 images of the reconstructed slices are displayed to illustrate certain images that could not be utilized by the tool for 3-D visualisation

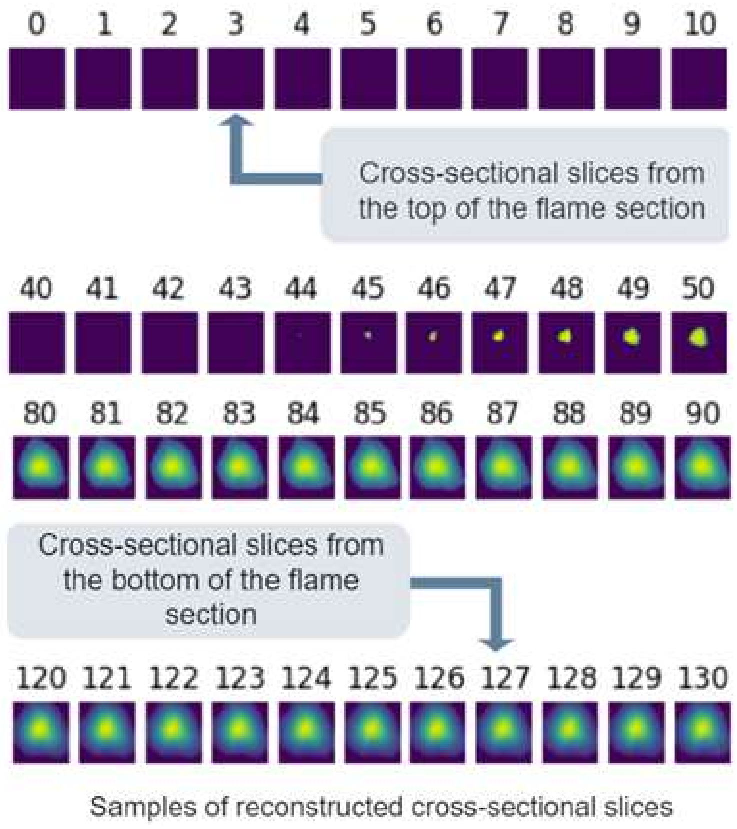

0 1 2 3 4 5 6 7 8 9 10
Cross-sectional slices from the top of the flame section
40 41 42 43 44 45 46 47 48 49 50
80 81 82 83 84 85 86 87 88 89 90
Cross-sectional slices from the bottom of the flame section
120 121 122 123 124 125 126 127 128 129 130
Samples of reconstructed cross-sectional slices

Figure 5.7 Reconstructed cross-sectional slices

The graphs in Figures 5.8, 5.9 and 5.10 show the results of the flame reconstructions using the trained network model and the data set aside for testing. The ground truth slices are compared with the reconstructed slices. The metrics SSIM, RMSE and PSNR which are discussed in section 4 were used in examining the accuracy of the reconstruction. The test data contain 120 flame samples.

In Figure 5.8 the reconstruction accuracy was evaluated by observing the mean SSIM of the reconstructed slices. The mean SSIM value was 0.77 ± 0.07. An average structural similarity index (SSIM) value of 0.77 indicates that, on average, the similarity between the ground truth image and the images from the reconstructed slices is fairly high.

It also appears that the reconstruction accuracy was not consistent across the flame samples used in the test. This can be explained by the fact the reconstructed slices contained some background noise which required filtering. The removal of this background noise before comparing the reconstructed slice with the ground truth image could help in increasing the reported SSIM value. Additional gains could also be achieved through further modification of the network model and the use of different hyperparameters during training.

The RMSE can be seen in Figure 5.9. According to the analysis, the mean RMSE value was 9.7e-04 ± 3.0e-4. This indicates that the CNN model made predictions with a relatively small error and that error variability across samples is also small. A PSNR value of 60.9 ± 2.87 was obtained during the analysis. Figure 5.10 shows the PSNR reported. This indicates that the reconstructed flame slices have high quality and relatively low variability in quality across samples. The RMSE and PSNR show a relatively high level of reconstruction accuracy, which is consistent with the results found in the SSIM.

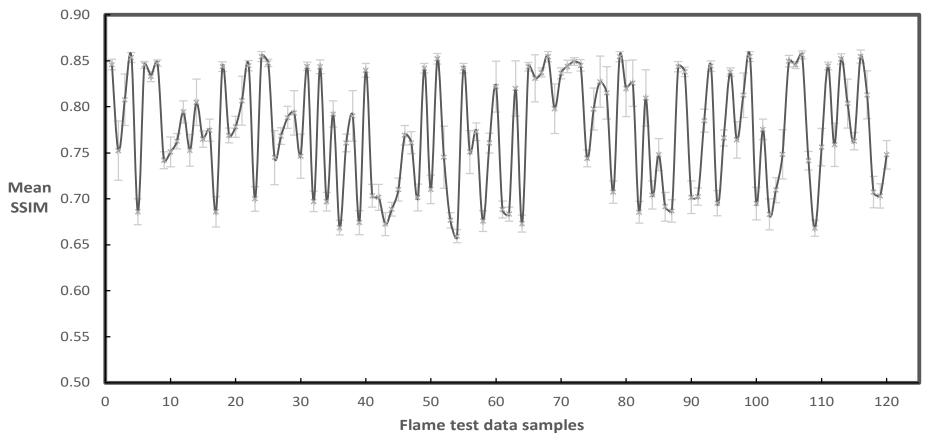


Figure 5.8 Average SSIM of reconstructed slices

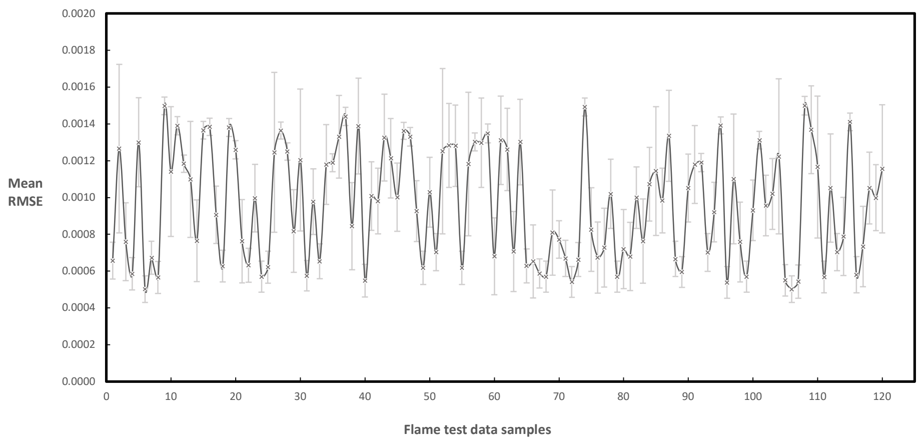


Figure 5.9 Average RMSE of reconstructed slices

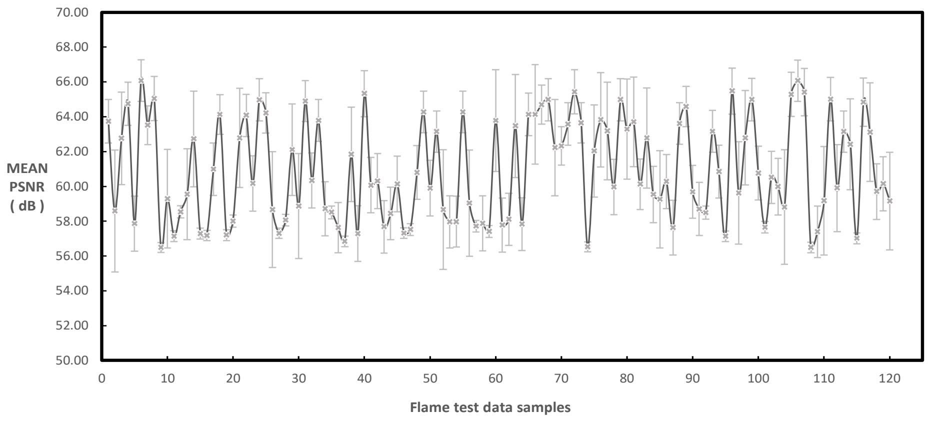


Figure 5.10 Average PSNR of reconstructed slices

5.5.1 Reconstruction of Flame Cross-sections

The cross-sections of the flame were reconstructed using the trained network model. Using a 3-D visualisation tool, data obtained from the output of the network model were used in viewing the cross-sectional slices. Figure 5.11 shows an example of the grey-level reconstructions of flame cross-sections of condition 6.

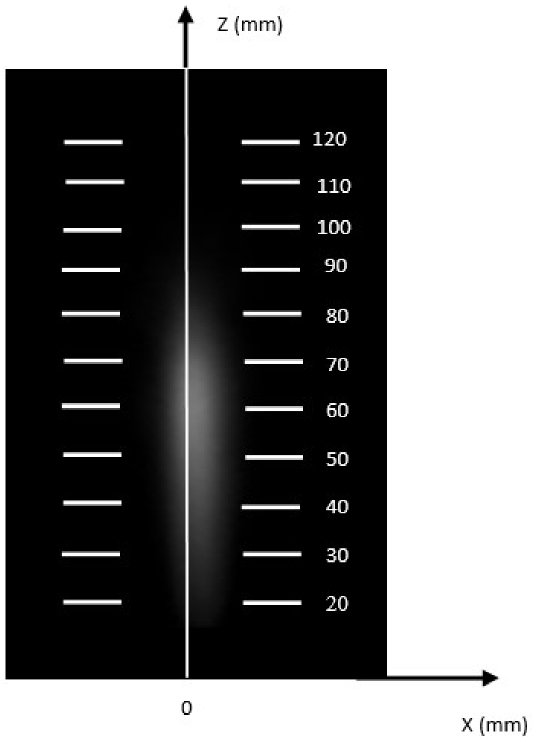


Figure 5.11 2-D Image of the flame

The reconstructions in Figure 5.12 show variations of the grey levels on the different cross-sections. Some direct observations can be made: cross-sections (a), (b) and (c) appear to be more homogenous than that in the lower cross-sections. These cross-sections are further away from the burner outlet. The circularity of the cross-section is degraded along the burner axis (Z-axis, refer to Figure 5.11) with the lower cross-sections being more circular.

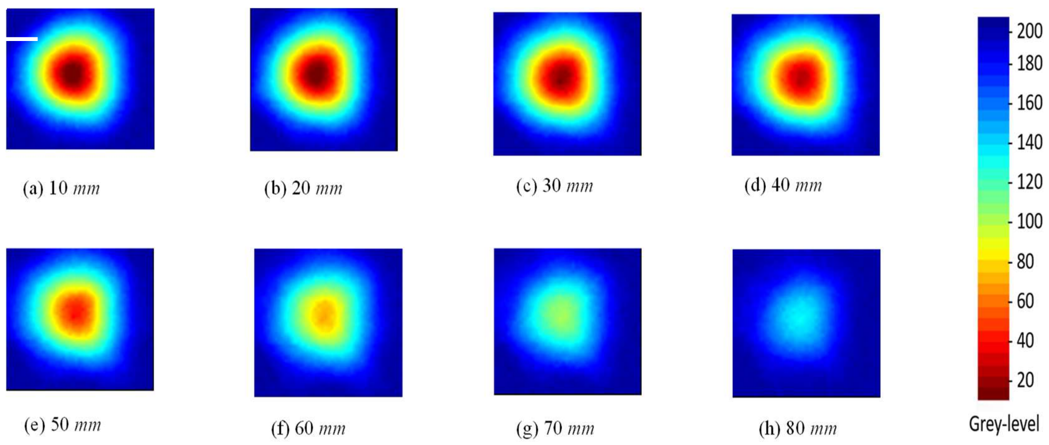


Figure 5.12 Grey-level distributions of flame cross-sections at different heights

5.5.2 Reconstruction of Flame Longitudinal Sections

The reconstruction of the flame longitudinal sections can be computed repeatedly for a chosen pixel row along Z-axis. The data obtained can also be used to reconstruct the longitudinal sections of the flame, as shown in Figure 5.13.

The results presented in Figure 5.13 refer to sections viewed from the direction of 0° (refer to Figure 5.11). In Figure 5.13, a radial distance of 0 *mm* depicts the longitudinal section along the burner axis, while radial distances -2 *mm* and -4 *mm* present two longitudinal sections back-off the burner axis, and 2 and 4 *mm* present two sections forward the burner axis. As the radial distances 0, 2 and -2 mm are close to the burner axis, similar luminous profiles have been observed and these agree well with the original 2-D image of the flame (Figure 5.2).

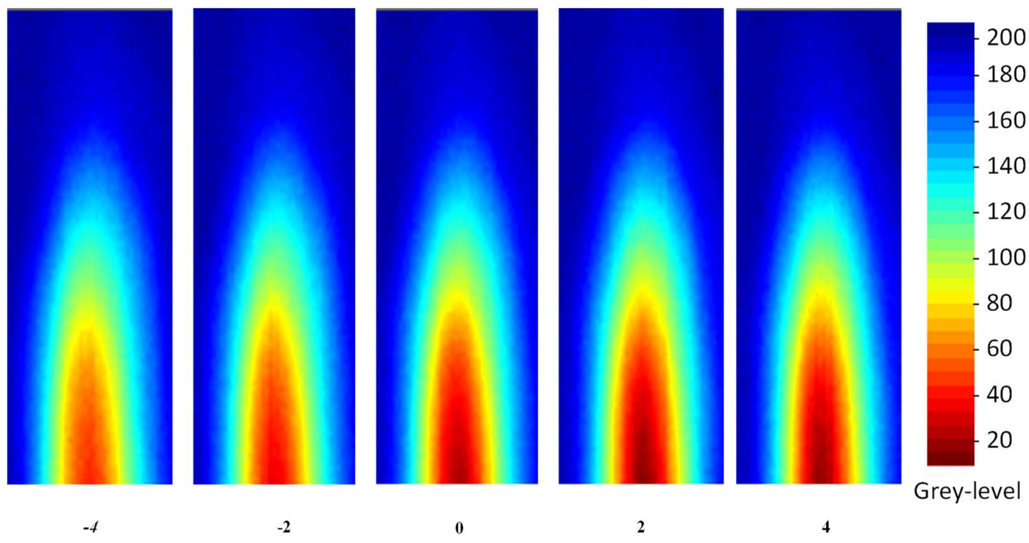


Figure 5.13 Grey-level distributions of longitudinal sections

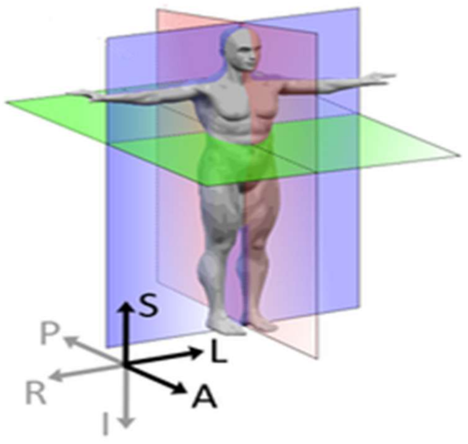
5.5.3 Flame Volumetric Reconstruction

Once the grey-level reconstructions of the flame are completed, the 3-D volumetric reconstruction was performed using a 3-D visualisation tool. The tool which is known as 3D Slicer is widely used for medical, biomedical, and related imaging research.

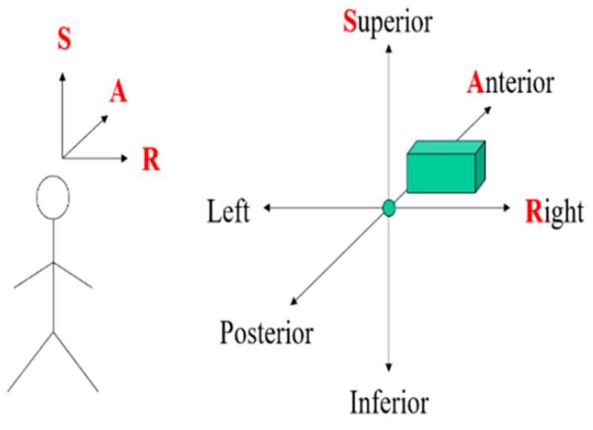
3D slicer is configured to display images using the anatomical coordinate system (Figure 5.14). This is the most important coordinate system for medical imaging techniques. The anatomical space consists of three planes to describe the anatomical position of a human [103].

The three planes are as follows:

- the axial plane is parallel to the ground and separates the head (Superior) from the feet (Inferior).
- the coronal plane is perpendicular to the ground and separates the front (Anterior) and the back (Posterior) [103].
- the *sagittal plane* separates the Left from the Right.

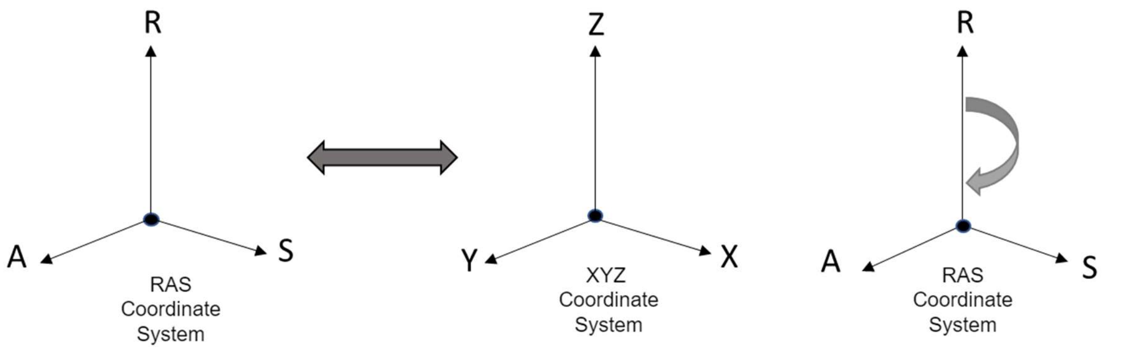


(a) Anatomical coordinate system



(b) RAS coordinate system

Figure 5.14 Anatomical and RAS coordinate system (John [104])



(a) Mapping of RAS to the XYZ Coordinate System

(b) Rotation along the R axis on a RAS Coordinate System

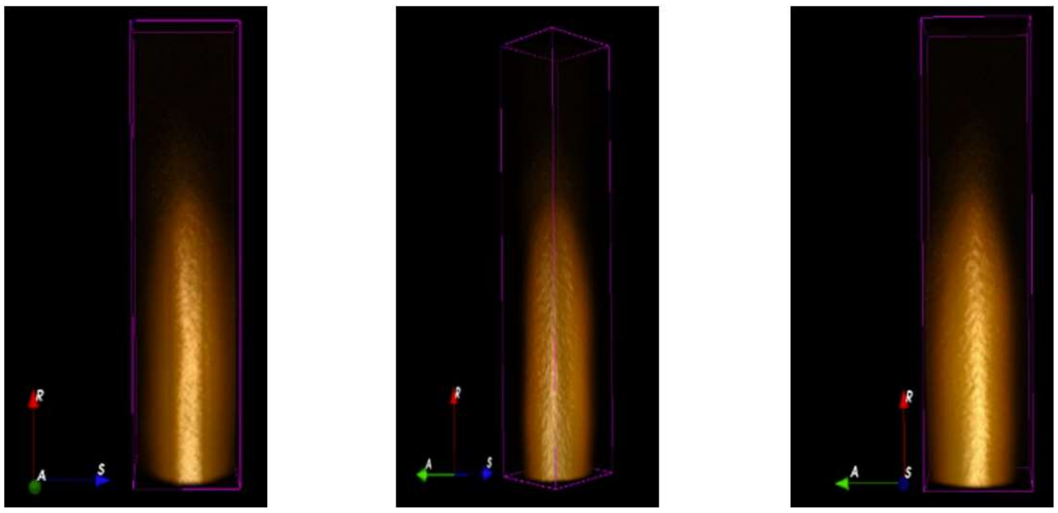
Figure 5.15 Mapping and rotation of the RAS coordinate system

However, most medical applications use different variations of this 3-D view. The actual one used in the 3D slicer is the RAS coordinate system (Figure 5.15(b)), where R stands for Right, A for anterior and S for Superior [104].

To understand the different views of the 3-D volumetric reconstructions figures shown below, the RAS coordinate system used in 3D-Slicer was mapped to the XYZ coordinate system. This mapping shown in Figure 5.15(b) is particular to the configuration used in the tool for this experiment. For instance, a rotation along the R-axis in the RAS coordinate system is equivalent to a rotation along the Z-axis in an XYZ coordinate system.

The images in Figure 5.16 illustrate examples of the 3-D reconstructed flame structure obtained from the network model. These images are based on 3 different views which were obtained by rotating the image by 45 degrees along the R-axis of the RAS coordinates (Figure 5.15 (b)). This rotation was in the direction of the Anterior axis along the SA plane.

Starting with the image in Figure 5.16 (a), the view is rotated 45 degrees along the R axis to get the image in Figure 5.16 (b) and then rotated again by 45 degrees along the R axis to get the image in Figure 5.16 (c).



(a) 3D Reconstructed flame structure before rotation

(b) 3D Reconstructed flame structure rotated along the R axis by 45 degree

(c) 3D Reconstructed flame structure rotated along the R axis by 90 degree

Figure 5.16 3-D Reconstructed flame structure

5.5.4 3-D Flame Reconstruction Time

To carry out a direct comparison with methods used in other studies, there are several differences in the flame structure, the hardware platform, the code optimization, and the deep learning model architecture which make it impossible to draw any firm conclusions. The table below shows the 3-D flame reconstruction time reported in the study. Both CPU and TPU hardware were used on the Google Colab to obtain the results.

Table 5.4 Reconstruction time

Reconstruction method	**Hardware platform**	**Reconstruction time (s)**
CNN (This study)	TPU	0.24
	Intel(R) Xeon (R) CPU, 2.30GHz	1.07
CNN (Jin et al. [48])	Intel Core i7-8750H CPU, 2.20 GHz	1.26
MART (Jin et al. [48])		143
ART (Jin et al. [48])		170.25

The TPU described in section 3.4.4 is a hardware accelerator for ML workloads, so the 3-D flame reconstruction time reported is expected to be faster than that reported from the CPU. The results from this study demonstrate this. Additionally, the CPU time reported in this study is slightly faster than that reported by Jin et al. [48]. These results are expected since Xeon CPUs are designed for high-performance and high-reliability server and workstation applications. In general, Core i7 CPUs are intended for use by consumers.

Furthermore, although Google Colab's specific Xeon CPU model is not disclosed publicly, the service provides users with access to high-performance computing resources for a variety of ML, data science, and other computationally intensive tasks. This may explain why the results obtained using Google Colab were faster.

5.6 Summary

A series of experiments have been conducted using real flame data obtained from the laboratory-scale combustion test rig to evaluate the effectiveness of using DL in reconstructing 3-D flames. A detailed description has been provided of the design, implementation, and training of the CNN-based network model in detail. Detailed descriptions of the generation and preparation of the experimental flame datasets have been provided.

In the experiments conducted, the 2-D flame projection images were obtained from eight different directions under various combustion conditions using a laboratory-scale combustion test rig. This dataset was further augmented using OpenCV morphological operators. For the ground-truth dataset, the SART algorithm was used in generating cross-sectional slices taking the 2-D projection data as inputs. Using this approach, a total of 600 datasets were made for the real flame 3-D reconstruction experiment.

The flame cross-sectional slices were reconstructed using a CNN-based network model implemented in the numerical simulation experiment, and using the ground truth data the SSIM, RMSE and PSNR of the reconstructed slices were computed. The obtained results have proven that the implemented and trained network model can reconstruct the cross-sectional slices of a burner flame based on the images obtained under various combustion conditions.

Also, using a 3-D visualisation tool, it has been possible to obtain a 3-D volumetric flame structure from the reconstructed cross-sectional flame data. This tool was also used to visualize the cross-sectional and longitudinal sections of the flame. The result of this study is encouraging and shows that DL techniques can be a useful tool for the 3-D reconstruction of burner flames.

6
Conclusion & Recommendations for Future Work

6.1 Introduction

The research work presented in this thesis is concerned with the design, implementation, and experimental evaluation of a DL-based system for the 3-D reconstruction of and visualisation of fossil-fired burner flames.

To develop an approach and methodology for this study, a review of current techniques for 3-D visualisation and characterisation of flames has been conducted, as well as an evaluation of CT-based tomographic and deep learning methods and their application to 3-D flame reconstruction. Following a review of these methods, the objectives and aims of the study were developed. Thus, an objective was set to develop a CNN-based DL network model for the reconstruction of 3-D volumetric information of a burner flame.

A prototype system based on DL and tomography reconstruction has been developed for taking data of 2-D flame projections from eight different directions around the flame and reconstructing cross-sectional slices of the flame. Initially, the CNN model was evaluated using a numerical simulation, followed by the implementation of the model using flame data. A set of six flame datasets based on six combustion conditions were used to train and evaluate the DL system. The results presented have demonstrated that the system can reconstruct a 3-D flame with good accuracy and structural similarity under a range of combustion conditions.

This chapter presents conclusions drawn from the work and outlines recommendations for future work.

6.2 Conclusions

6.2.1 Deep Learning Model

As CNN has demonstrated outstanding performance in computer vision tasks such as image classification, object recognition, segmentation, etc., it has been used in this study. The CNN model designed and implemented in this study has seven 2-D convolutional layers and a dense output layer, each with linear activation functions. Datasets of 2-D flame images were used as inputs to the model. Ground truth data was generated from 2-D flame images using the SART algorithm. However, for the numerical simulation, the CNN model was trained using numerically simulated images.

Detailed information about the CNN training process and the performance metrics needed for analysing and evaluating its accuracy has also been provided. PSNR and SSIM, two of the most widely used metrics in computer vision and image processing were used to evaluate the reconstructed images in this study. Additionally, the performance of the implemented CNN model was assessed using MAE, MSE, and RMSE metrics. Also included is a description of the software development environment and ML framework employed in this study. As well as the tools that were used to create and prepare experimental datasets, this includes the tools used to visualize the 3-D flame.

6.2.2 Numerical Simulation

To evaluate DL's effectiveness in reconstructing numerical simulation images with a similar shape to 3-D flames, several experiments were conducted using a numerical simulation and CNN-based network model. Detailed information about the CNN-based network model's design, implementation, and training is provided. Further, the experimental datasets have been described in detail. Using a mathematical model in the form of a 3-D bivariate Gaussian distribution plot, experimental ground truth datasets with the shape of a 3-D flame structure were generated. The ground truth datasets were generated under two different conditions for the experiments conducted. Two samples of ground truth datasets were provided. SSIM values for Samples A and B were 0.9 and 0.8, respectively

In the experiment, 2000 ground truth datasets were generated with 1000 datasets for each sample. Code implemented using ImageMagick APIs was used to create eight 2-D projections from the 3-D ground truth image based upon a bivariate Gaussian distribution. A total of 1000 projection datasets were generated for each set of ground truth samples. A CNN model based on seven 2-D convolutional layers with a linear activation function was developed following a series of tests with different convolutional layers and hyperparameters.

By using eight 2-D projection data sets, the established network model was able to successfully reconstruct the 3-D bivariate Gaussian distribution. It has been demonstrated through the achieved results that the implemented and trained network model has the capability of reconstructing the 3-D bivariate Gaussian distribution with good accuracy and structural similarity. The implemented CNN reconstructed the 3-D bivariate Gaussian distribution image in 0.08 seconds. Google Colab TPU hardware was used for the test. Reconstruction time was faster than iterative methods like ART and SART.

6.2.3 3-D Flame Reconstruction

The effectiveness of DL in reconstructing 3-D flames has also been evaluated through a series of experiments utilizing real flame data from a laboratory-scale combustion test rig. Detailed descriptions of the design, implementation and training of the CNN-based network model have been provided. The generation, preparation, and analysis of experimental flame datasets have also been presented. A laboratory-scale combustion test rig was utilized to obtain 2-D flame projection images under different combustion conditions during the experiments conducted. This dataset was further augmented using OpenCV morphological operators. The 2-D flame projection data obtained was used to generate cross-sectional slices for the ground-truth dataset using SART. A total of 600 dataset samples were generated for testing, training, and validation purposes.

With the aid of a CNN network model implemented in the numerical simulation experiment, flame cross-sectional slices were reconstructed, and the SSIM, RMSE and PSNR were calculated based on the ground truth data. Based on the experimental flame dataset obtained under various combustion conditions, the network model was trained, implemented, and used to reconstruct cross-sectional slices of a burner flame. From the reconstructed cross-sectional flame data, the volumetric 3-D flame structure has also been visualized using a 3-D visualisation tool. Additionally, this tool enabled the visualisation of cross-sectional and longitudinal flame sections. This study shows that the DL learning technique can be a useful tool for the 3-D reconstruction of and visualisation of burner flames.

6.2.4 Dataset

It has been discussed in section 2.5 that when working on CNN-based DL flame reconstruction methods, a large dataset of approximately 10000 to 15000 is usually required for training purposes. However, for this study, a small dataset of 360 was used for training the implemented CNN model.

In this case, the use of fewer datasets for 3-D flame reconstruction is justified due to limitations in the availability and accessibility of data when the research was undertaken. Moreover, there were resource constraints, such as limited storage space and processing power, which made data augmentation challenging.

The effect of using a smaller training dataset may lead to several limitations and drawbacks:

1. Model accuracy is reduced: Smaller training datasets may not capture the full complexity and diversity of the data, resulting in less accurate predictions. Models trained on small datasets may not generalize well to new data.

2. Increased overfitting risk: Smaller training datasets increase the likelihood of overfitting, where the model memorizes the training data instead of learning its

underlying patterns and relationships. Consequently, a model may perform well on the training dataset but poorly on the new data.

3. Data representation is limited: In real-world applications, the model may run into a variety of variations and scenarios that are not covered by a smaller training dataset. As a result, a model could be biased towards certain types of data and may not be able to handle new, unknown data effectively.

4. Inability to detect outliers: A smaller training dataset makes it more difficult to detect outliers or anomalies, which may adversely affect the model's performance.

However, as discussed in section 6.3, the collection of a larger dataset would be very beneficial for future research.

6.2.5 3-D Flame Reconstruction Time

The 3-D flame reconstruction time of the established CNN system was obtained using both CPU and TPU hardware on Google Colab. The reconstruction time for the CPU was reported as 1.07 seconds and the reconstruction time for the TPU was 0.24 seconds. In general, the reconstruction time was faster than iterative methods like ART and SART.

6.3 Recommendations for Future Work

The research work presented in this thesis has demonstrated the viability and potential of the DL-based system for the 3-D reconstruction of flames. There are, however, several areas that need further research and improvements in the near future. The key areas are identified and stated as follows:

6.3.1 Larger Dataset

In future studies, the collection of larger datasets would significantly improve the accuracy of the deep learning-based 3-D flame reconstruction solution implemented and also enable the CNN model to perform better with various flame conditions. Most of the literature reviewed in section 2.3 used datasets containing at least 10,000 samples for training and evaluation of the network model.

In this study, 360 samples were used for training and evaluating the CNN network model implemented for 3-D flame reconstruction. Hardware limitations contributed to not having the capability of collecting larger samples. The availability of a high-performance computer with a large memory specification would help in resolving this issue.

Furthermore, the availability of datasets obtained under more combustion conditions would be beneficial for future experiments. In this study, only six combustion conditions were used during the generation of datasets.

6.3.2 CNN Model Optimisation

It is recognised that some work was done to improve the network model implemented. This was limited to changing the number of convolutional layers and trying different activation functions in the CNN network model architecture. Also, some tests were conducted using different learning rate values. However, in the future, a more detailed investigation can be undertaken. These could include experimenting with proven CNN network architectures and modifying them for this solution. Additional changes could also be made by adding more layers and updating the properties of these layers in the CNN model.

6.3.3 Automation of 3-D flame Visualisation

In addition, in the present study, during the 3-D flame reconstruction, data obtained from the output of the CNN network model must be processed offline and rendered using a 3-D visualisation tool. This defeats the purpose of using DL methods as one of the benefits was a faster reconstruction time which should enable online monitoring of a combustion flame. In future, this process could be automated in such a way that the data from the CNN model output could be visualized in real-time without the need for a third-party visualisation tool.

6.3.4 Hardware Accelerators and Edge Computing

Furthermore, DL algorithms in ML applications require large data sets and are computationally intensive. To accelerate these algorithms, CPUs, GPUs, and other hardware-accelerated processing units have been deployed in edge computing devices. As a part of this study, 3-D reconstruction performance results were obtained using Google Colab's TPU hardware which is a hardware accelerator for ML workloads. However, further research is needed to investigate the use of hardware accelerators and edge computing in DL to reconstruct flames in real-time.

Also, edge computing devices tend to have limited power and memory bandwidth while requiring extremely fast responses. On edge computing, this limitation may affect the performance of the tomographic reconstruction of flames using DL. Nevertheless, future studies could focus on exploring ways to optimize the performance of tomographic reconstruction of flames through DL on edge computing devices. This will be accomplished by researching novel ways of implementing DL algorithms on edge computing devices

References

[1] M. Bozkurt, M. Fikri, and C. Schulz, "Investigation of the kinetics of OH* and CH* chemiluminescence in hydrocarbon oxidation behind reflected shock waves", *Applied Physics B: Lasers and Optics,* vol.107, pp. 515–527, 2012.

[2] V. N. Nori and J. M. Seitzman, "CH* chemiluminescence modeling for combustion diagnostics", *Proceedings of the Combustion Institute*, vol.32, pp. 895–903, 2009.

[3] J. B. Michael, P. Venkateswaran, J. D. Miller, M. N. Slipchenko, J. R. Gord, S. Roy, and T. R. Meyer, "100 kHz thousand-frame burst-mode planar imaging in turbulent flames", *Optical Letters*, vol.39, pp. 739-742, 2014.

[4] C. Ruan, F. Chen, W. Cai, Y. Qian, L. Yu, X.i Lu, "Principles of non-intrusive diagnostic techniques and their applications for fundamental studies of combustion instabilities in gas turbine combustors: A brief review", *Aerospace Science and Technology*, vol.84, pp. 585-603, 2019.

[5] M. Zhang, J. Wang, W. Jin, Z. Huang, H. Kobayashi, L. Ma, "Estimation of 3D flame surface density and global fuel consumption rate from 2D PLIF images of turbulent premixed flame", *Combustion and Flame*, vol.162, pp. 2087-2097, 2015.

[6] H. C. Bheemul, "Three-dimensional visualisation and quantitative characterisation of combustion flames," PhD Thesis, University of Greenwich, 2004.

[7] P. Brisley, "Three-dimensional temperature measurement of combustion flames using digital imaging techniques," MSc, Electronic Engineering, The University of Kent, Canterbury, 2006.

[8] M. Zhang, J. Wang, W. Jin, Z. Huang, H. Kobayashi and L. Ma, "Estimation of 3-D flame surface density and global fuel consumption rate from 2-D PLIF images of turbulent premixed flame", *Combustion and Flame*, vol.162, pp. 2087-2097, 2015.

[9] L. Hecong, S. Bin and C. Weiwei, "kHz-rate volumetric flame imaging using a single camera", *Optics Communications*, vol. 437, pp. 33-43, 2019.

[10] T. Yu, H. Liu, J. Zhang, W. Cai, and F. Qi, "Toward real-time volumetric tomography for combustion diagnostics via dimension reduction", *Optics Letters*, vol.43, pp. 1107-1110, 2018.

[11] J. Floyd, P. Geipel and A.M. Kempf, “Computed tomography of chemiluminescence (CTC): instantaneous 3-D measurements and phantom studies of a turbulent opposed jet flame”, *Combustion and Flame*, vol.158, pp. 376–391, 2011.

[12] K. Mohri, S. Görs, J. Schöler, A. Rittler, T. Dreier, C. Schulz and A. Kempf, “Instantaneous 3-D imaging of highly turbulent flames using computed tomography of chemiluminescence”, *Applied Optics*, vol.56, pp.7385–7395, 2017.

[13] D. Ebi and N.T. Clemens, “Simultaneous high-speed 3-D flame front detection and tomographic PIV”, *Measurement Science and Technology*, vol. 27, 2016.

[14] P. Brisley, G. Lu, Y. Yan, and S. Cornwell, "Three-dimensional temperature measurement of combustion flames using a single monochromatic CCD camera," *Instrumentation and Measurement, IEEE Transactions on,* vol. 54, pp. 1417-1421, 2005.

[15] Z. Yan, Q. Liang, Q. Guo, G. Yu, and Z. Yu, "Experimental investigations on temperature distributions of flame sections in a bench-scale opposed multi-burner gasifier," *Applied Energy,* vol. 86, pp. 1359-1364, 2009.

[16] A. S. Veríssimo, T. L. Pedro, and A. O. Toledo, "Bi-dimensional reconstruction of a bunsen burner flame," *8th International Congress of Mechanical Engineering*, Ouro Preto, Brazil, pp. 1-8, 2005.

[17] A. Goyal, S. Chaudhry, and P. M. V. Subbarao, "Direct three dimensional tomography of flames using maximization of entropy technique," *Combustion and Flame*, pp. 173–183, 2013.

[18] Q. Huang, F. Wang, D. Liu, Z. Ma, J. Yan, Y. Chi, and K. Cen, "Reconstruction of soot temperature and volume fraction profiles of an asymmetric flame using stereoscopic tomography," *Combustion and Flame,* vol. 156, pp. 565-573, 2009.

[19] B. Zhou, W. Shimin, X. Chuanlong, and Z. Jianyong, "3-D flame temperature reconstruction in optical sectioning tomography," *Imaging Systems and Techniques, . IST '09. IEEE International Workshop* , pp. 313-318, 2009

[20] G. Yan, G. Qinghua, L. Qinfeng, Z. Zhijie, and Y. Guangsuo, “Three-Dimensional temperature distribution of impinging flames in an opposed multiburner gasifier”, *American Chemical Society*, vol. 51, 2012.

[21] W. Cai, X. Li, F. Li, and L. Ma, "Numerical and experimental validation of a three-dimensional combustion diagnostic based on tomographic chemiluminescence," *Optics Express*, vol. 21, pp. 7050-7064, 2013.

[22] A. W. Nicholas and R. D. James, "Tomographic reconstruction of OH* chemiluminescence in two interacting turbulent flames," *Measurement Science and Technology*, vol. 24, pp. 024013-024023, 2013.

[23] A. Schwarz, "Multi-tomographic flame analysis with a schlieren apparatus," *Measurement Science and Technology*, vol. 7, pp. 406-413, 1996.

[24] M. M. Hossain, "Tomographic characterization of burner flames through digital imaging and image processing", PhD Thesis, School of Engineering and Digital Arts, University of Kent, Canterbury, 2014.

[25] G. Gilabert, G. Lu, and Y. Yan, "Three-dimensional tomographic reconstruction of the luminosity distribution of a combustion flame," *Instrumentation and Measurement, IEEE Transactions on*, vol. 56, pp. 1300-1306, 2007.

[26] Y. Ishino, T. Kei, S. Sakiko, and O. Norio, "Non-Scanning 3D-CT measurement with 40-lens tracking camera for turbulent propane/air rich-premixed flame," *6th International Energy Conversion Engineering Conference (IECEC), American Institute of Aeronautics and Astronautics*, pp. 1-6, 2008

[27] H.Zhou, S.-D. Han, F. Sheng, and C. Zheng, "Visualisation of three-dimensional temperature distributions in a large-scale furnace via regularized reconstruction from radiative energy images: numerical studies*," Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 72, pp. 361-383, 2002.

[28] D. P. Correia, P. FerrÃO, and A. Caldeira-Pires, "Advanced 3D emission tomography flame temperature sensor," *Combustion Science and Technology*, vol. 163, pp. 1-24, 2001.

[29] J. Floyd and A. M. Kempf, "Computed tomography of chemiluminescence (CTC): high resolution and instantaneous 3-D measurements of a matrix burner," *Proceedings of the Combustion Institute*, vol. 33, pp. 751-758, 2011.

[30] J. Floyd, P. Geipel, and A. M. Kempf, "Computed tomography of chemiluminescence (CTC): instantaneous 3D measurements and phantom studies of a turbulent opposed jet flame," *Combustion and Flame*, vol. 158, pp. 376-391, 2011.

[31] A. Mohsen, Industrialized Computerized Axial Tomography (CAT-CT), 2017

[32] S. Coban, “Practical approaches to reconstruction and analysis for 3-D and dynamic 3-D computed tomography”, PhD Thesis, Department of Mathematics, University of Manchester, pp. 12, 2017.

[33] O. Scherzer, R. Ramlau, “The first 100 years of the Radon transform”, Inverse Problems, 2018.

[34] M. Yang, S. Duan et al, “Extra projection data identification method for fast-continuous-rotation industrial cone-beam CT”, *Journal of X-ray Science and Technology*”, 2013.

[35] T. Buzug, “Fundamentals of X-ray physics, Computed tomography: from photon statistics to modern cone-beam CT”. Springer, Berlin, Heidelberg, 2008.

[36] D. Panetta, N.Camarlinghi, “3-D Image Reconstruction for CT and PET A Practical Guide with Python”, CRC Press, 2022.

[37] H. Jung, “Basic Physical Principles and Clinical Applications of Computed Tomography”, *Progress in Medical Physics*, pp.1-17, 2021.

[38] 3-D image reconstruction. Online. (Last access: 19/09/2022), Available: https://humanhealth.iaea.org/HHW/MedicalPhysics/NuclearMedicine/ImageAnalysis/3Dimagereconstruction/index.html

[39] G.L. Zeng, “Computerized Medical Imaging and Graphics”, pp. 97-103, 2001.

[40] R. Gordon, R. Bender, and G. T. Herman, “Algebraic reconstruction techniques (ART) for 3-dimensional electron microscopy and X-Ray photography”, *Journal of Theoretical Biology*, vol. 29, pp. 471,1970.

[41] P. Gilbert, “Iterative methods for 3-dimensional reconstruction of an object from projections”, *Journal of Theoretical Biology*, vol. 36, pp. 105, 1972.

[42] D. Verhoeven, “Multiplicative algebraic computed tomographic algorithms for the reconstruction of multidirectional interferometric data”, *Optical Engineering*, vol. 32, pp. 410-419, 1993.

[43] A. H. Andersen and A. C. Kak, “Simultaneous algebraic reconstruction technique (SART) – A superior Implementation of the ART algorithm”, *Ultrasonic Imaging*, vol. 6, pp. 81-94, 1984.

[44] J. Schmidhuber, “Deep learning in neural networks: An overview”, *Neural Networks*, vol. 61, pp. 85-117, 2015.

[45] T. Lähivaara, L. Kärkkäinen, J.M.J. Huttunen and J.S. Hesthaven, “Deep convolutional neural networks for estimating porous material parameters with ultrasound tomography, *The Journal of the Acoustical Society of America*, pp. 1148-1158, 2018.

[46] G.Katti, SA Ara, and A.Shireen, “Magnetic resonance imaging (MRI) - A review”. *International Journal of Dental Clinics*, vol. 3, 2011.

[47] H. Chen, Y. Zhang, W. Zhang, P. Liao, K. Li, J. Zhou and G. Wang, “Low-dose CT via convolutional neural network”, *Biomedical Optics Express*, vol.8, pp.679-694, 2017.

[48] Y. Jin, W. Zhang, Y. Song, X. Qu, Z. Li, Y. Ji, and A. He, "Three-dimensional rapid flame chemiluminescence tomography via deep learning," *Optics Express*, vol. 27, pp. 27308-27334, 2019.

[49] J.Huang et al, “Tomographic reconstruction for 3D flame imaging using convolutional neural networks”, *12th Asia-Pacific Conference on Combustion*, 2019

[50] J. Huang, H. Liu, Q. Wang, W. Cai, “Limited-projection volumetric tomography for time-resolved turbulent combustion diagnostics via deep learning”, *Aerospace Science and Technology*, vol.106, 2020.

[51] W.Cai, J. Huang, A.Deng and Q.Wang, “ Volumetric reconstruction for combustion diagnostics via transfer learning and semi-supervised learning with limited labels”, *Aerospace Science and Technology*, 2021

[52] J. Huang, H. Liu and W. Cai, “Online in situ prediction of 3-D flame evolution from its history 2-D projections via deep learning”, *Journal of Fluid Mechanics*, 2019

[53] L. Alzubaidi, J. Zhang, A.J. Humaidi et al., “Review of deep learning: concepts, CNN architectures, challenges, applications, future directions”, *Journal of Big Data*, vol. 8, 2021.

[54] Deep learning. Online. (Last access: 20/09/2022), Available: https://www.ibm.com/cloud/learn/deep-learning

[55] D. Yu and L. Deng, “Deep learning and its applications to signal and information processing”, *IEEE Signal Processing Magazine*, vol. 28, no. 1, pp. 145-154, 2011.

[56] O. Ronneberger, P. Fischer and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation”, 2015

[57] U.Odi and T.Nguyen, “Geological Facies Prediction Using Computed Tomography in a Machine Learning and Deep Learning Environment”, *2018 Unconventional Resources Technology Conference*, 2018

[58] I. Goodfellow, Y. Bengio and A. Courville, *Deep Learning*, MIT Press, 2016.

[59] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity”, *Bulletin of Mathematical Biology*, vol. 5, no. 4, pp. 115–133, 1943.

[60] A.Voulodimos, N .Doulamis, A. Doulamis and E. Protopapadakis, “Deep Learning for Computer Vision: A Brief Review”, *Computational Intelligence and Neuroscience*, pp. 1-13, 2018.

[61] M.I. Jordan, “Serial order: a parallel distributed processing approach”, *Advances in Psychology*, vol.121, pp. 471-495, 1997.

[62] Y. LeCun, B. Boser, J. Denker et al., “Handwritten digit recognition with a back-propagation network,” *Advances in Neural Information Processing Systems 2 (NIPS*89)*, D. Touretzky, Ed., Denver, CO, USA, 1990.

[63] S. Hochreiter and J. Schmidhuber, “Long short-term memory”, *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[64] G. E. Hinton, S. Osindero, and Y.W. Teh, “A fast learning algorithm for deep belief nets”, *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[65] Y. Chen, Y. Xie, L. Song, F. Chen and T. Tang, “A Survey of Accelerator Architectures for Deep Neural Networks, *Engineering*”, vol.6, pp. 264-274, 2020.

[66] TensorFlow. Online. (Last access:20/09/2022), Available: https://www.tensorfow.org

[67] B. Frederic, P. Lamblin, R. Pascanu et al., “Theano: new features and speed improvements,” *Deep Learning and Unsupervised Feature Learning, NIPS 2012 Workshop*, 2012

[68] Pytorch. Online. (Last access: 20/09/2022), Available: https://pytorch.org

[69] D. H. Hubel and T. N. Wiesel, “Receptive fields, binocular interaction, and functional architecture in the cat’s visual cortex”, *The Journal of Physiology*, vol. 160, pp. 106–154, 1962.

[70] M. Aamir, R. Ziaur, W. Abro et al, “An Optimized Architecture of Image Classification Using Convolutional Neural Network”, *International Journal of Image, Graphics and Signal Processing*, vol. 11, pp. 30-39, 2019.

[71] J. Quddus, “Machine Learning with Apache Spark Quick Start Guide", Packt Publishing, 2018

[72] J.Kong and C.Wang, “Resolution Enhancement for Low-resolution Text Images Using Generative Adversarial Network”, *MATEC Web of Conferences*, 2018

[73] N. Aloysius and M. Geetha. “A review on deep convolutional neural networks.” *2017 International Conference on Communication and Signal Processing (ICCSP),* pp. 0588-0592, 2017.

[74] A. Zahirovic, Using Deep Learning to Remove Computed Tomography Artifacts due to Hip Replacement, Master Thesis, Lund University, 2020

[75] F. Nuzzo, Sanity Checks for Explanations of Deep Neural Networks Predictions, Master Thesis, Polytechnic University of Turin, 2020.

[76] B. Sabri, “A Comparison between Average and Max-Pooling in Convolutional Neural Network for Scoliosis Classification”, *International Journal of Advanced Trends in Computer Science and Engineering*, pp. 694, 2020.

[77] G.Martinez-Soltero, A.Alanis, N.Arana-Daniel and C.Lopez-Franco, “Semantic Segmentation for Aerial Mapping”, *Mathematics. 8*, 2020

[78] V.Verdhan, "Computer Vision Using Deep Learning: Neural Network Architectures with Python and Keras", Apress, 2021

[79] Keras Conv2D Class. Online. (Last access: 20/09/2022), Available: https://keras.io/api/layers/convolution_layers/convolution2d

[80] Keras Convolution Layers. Online. (Last access: 20/09/2022), Available: https://keras.io/api/layers/convolution_layers/convolution

[81] Keras Conv2D & Convolution Layers. Online. (Last access: 20/09/2022), Available: https://pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers

[82] Choosing a convolution filter or kernel size. Online. (Last access: 20/09/2022), Available: https://medium.com/analytics-vidhya/how-to-choose-the-size-of-the-convolution-filter-or-kernel-size-for-cnn-86a55a1e2d15

[83] S. Ruder, "An overview of gradient descent optimization algorithms", 2016

[84] I.Kandel and M.Castelli, "How Deeply to Fine-Tune a Convolutional Neural Network: A Case Study Using a Histopathology Dataset", *Applied Sciences*, 10, 2020.

[85] E. Wang, "Deep Fusion Feature Based Object Detection Method for High Resolution Optical Remote Sensing Images", *Applied Sciences*, pp. 15, 2019

[86] Z. Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity", *IEEE Transactions on Image Processing*, vol. 13, pp. 600-612, 2004.

[87] PSNR. Online. (Last access: 20/09/2022), Available: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio

[88] A. Horé and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM", *20th International Conference on Pattern Recognition*, pp. 2366-2369, 2010.

[89] G.Rossum, "The History of Python: A Brief Timeline of Python". Online. (Last access: 20/09/2022), Available: https://github.com/eltonlaw/the-history-of-python/blob/master/markdown/a-brief-timeline-of-python.md

[90] Keras. Online. (Last access: 20/09/2022), Available: https://keras.io/about/

[91] Google Colaboratory. Online. (Last access: 20/09/2022), Available: https://research.google.com/colaboratory/faq.html#:~:text=Colaboratory%2C%20or%20%E2%80%9CColab%E2%80%9D%20for,learning%2C%20data%20analysis%20and%20education.

[92] Cloud TPU. Online. (Last access: 20/09/2022), Available: https://cloud.google.com/blog/products/gcp/cloud-tpu-machine-learning-accelerators-now-available-in-beta

[93] Z. Kegenbekov and I. Jackson, “Adaptive Supply Chain: Demand–Supply Synchronization Using Deep Reinforcement Learning”, Algorithms, 2021

[94] Using a TPU in Google Colab. Online. (Last access: 20/09/2022), Available: https://jannik-zuern.medium.com/using-a-tpu-in-google-colab-54257328d7da

[95] ImageMagick. Online. (Last access: 20/09/2022), Available: https://en.wikipedia.org/wiki/ImageMagick

[96] A. Fedorov et al, “ 3D Slicer as an Image Computing Platform for the Quantitative Imaging Network”, *Magnetic Resonance Imaging*, 2012

[97] Visualizing the bivariate Gaussian distribution. Online. (Last access: 20/09/2022), Available: https://scipython.com/blog/visualizing-the-bivariate-gaussian-distribution/

[98] P. Warden and D. Situnayake, *TinyML*, O'Reilly Media, 2019.

[99] K. Sreedhar and B. Panlal, “Enhancement of images using morphological transformations” *International Journal of Computer Science & Information Technology (IJCSIT),* Vol 4, No 1, 2012.

[100] Structuring Elements. Online. (Last access: 20/09/2022), Available:

[101] Dilation and Erosion. Online. (Last access: 20/09/2022), Available: https://uk.mathworks.com/help/images/morphological-dilation-and-erosion.html

[102] Morphological Transformations. Online. (Last access: 20/09/2022), Available: https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html

[103] Slicer Coordinate Systems. Online. (Last access: 20/09/2022), Available: https://www.slicer.org/wiki/Coordinate_systems

[104] C. John, Slicer’s Coordinate Systems. Online. (Last access: 20/09/2022), Available: https://www.na-mic.org/w/img_auth.php/3/3f/Coordinate_Systems_Demystified.ppt

Appendix 1 Program for generating numerical simulation experimental data

```
# TensorFlow is an open source machine learning library
import tensorflow as tf

import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import axes3d
import os, sys
import numpy as np

# Pandas is a data manipulation library
import pandas as pd

# Math is Python's math library
import math
import scipy.io
import scipy.stats

from PIL import Image

# Keras is TensorFlow's high-level API for deep learning
from tensorflow import keras


# Improting Image class from PIL module
from PIL import Image, ImageOps

from scipy import misc
from numpy import asarray
from numpy import array
from numpy import empty
from numpy import ones
from numpy import zeros
from numpy import save

import imageio
import io
import cv2
import IPython
import random
```

```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import array_to_img
from keras.preprocessing.image import save_img

seed = 1
np.random.seed(seed)

# Number of sample datapoints
SAMPLES = 200
NO_OF_PROJECTIONS = 8

# Derive Multivariate normal pdf
def multivariate_normal_pdf(X, mean, sigma):
  """Multivariate normal PDF over X (n_samples x n_features)"""
  P = X.shape[1]
  det = np.linalg.det(sigma)
  norm_const = 1.0 / (((2*np.pi) ** (P/2)) * np.sqrt(det))
  X_mu =  X - mu
  inv = np.linalg.inv(sigma)  * random.uniform(0.8, 2.6)
  d2 = np.sum(np.dot(X_mu, inv) * X_mu, axis=1)
  return norm_const * np.exp(-0.5 * d2)

# Generate ground truth figure
def get_mvd_image(mvd_val):
  fig = plt.figure(figsize=(10, 7))
  ax = fig.gca(projection='3d')
  ax.set_axis_off()
  ax.plot_surface(x, y, mvd_val, rstride=3,cstride=3,
                cmap=plt.cm.coolwarm,linewidth=1,
                antialiased=False)
  return ax

# Generate a uniformly distributed set of random numbers
#in the range from  0.1 to 0.6  for mean vector
u_values = np.random.uniform(low=0.0, high=0.2,
                      size=SAMPLES).astype(np.float32)
np.random.shuffle(u_values)

##### TO CREATE A SERIES OF PICTURES
def make_views(ax,angles,elevation=None, width=4, height = 3,
             prefix='tmprot_',**kwargs):
   """
```

```
    Makes jpeg pictures of the given 3d ax, with different angles.
    Args:
        ax (3D axis): te ax
        angles (list): the list of angles (in degree) under which to
                       take the picture.
        width,height (float): size, in inches, of the output images.
        prefix (str): prefix for the files created.

    Returns: the list of files created (for later removal)
    """

    files = []
    ax.figure.set_size_inches(width,height)

    for i,angle in enumerate(angles):
        ax.view_init(elev = elevation, azim=angle)
        fname = '%s%03d.jpeg'%(prefix,i)
        ax.figure.savefig(fname)
        files.append(fname)

    return files



##### TO TRANSFORM THE SERIES OF PICTURE INTO AN ANIMATION
def make_movie(files,output, fps=10,bitrate=1800,**kwargs):
    """
    Uses mencoder, produces a .mp4/.ogv/... movie from a list of
    picture files.
    """
    output_name, output_ext = os.path.splitext(output)
    command = { '.mp4' : 'C:/Users/dele.ogunjumelo/Documents/mplayer-svn-38151-
x86_64/mencoder
                     "mf://%s" -mf fps=%d -o %s.mp4 -ovc lavc\
                     -lavcopts vcodec=msmpeg4v2:vbitrate=%d'
                     %(",".join(files),fps,output_name,bitrate)}

    command['.ogv'] = command['.mp4'] + ';
                      ffmpeg -i %s.mp4 -r %d %s'%(output_name,fps,output)

    print(command[output_ext])
    output_ext = os.path.splitext(output)[1]
    os.system(command[output_ext])
```

```
def make_gif(files,output,delay=100, repeat=True,**kwargs):
    """
    Uses imageMagick to produce an animated .gif from a list of
    picture files.
    """
    loop = -1 if repeat else 0
    os.system('C:\Program Files\ImageMagick-7.0.11-Q16-HDRI\imagic_folder\magick
            convert  -delay %d -loop %d %s %s'

            %(delay,loop," ".join(files),output))


def make_strip(files,output,**kwargs):
    """
   Uses imageMagick to produce a .jpeg strip from a list of
    picture files.
    """
    os.system('C:\Program Files\ImageMagick-7.0.11-Q16-HDRI\imagic_folder\magick
            montage  -tile 1x -geometry +0+0 %s %s'%(" ".join(files),output))



##### MAIN FUNCTION
def rotanimate(ax, angles, output, **kwargs):
    """
    Produces an animation (.mp4,.ogv,.gif,.jpeg,.png) from a 3D plot on
    a 3D ax

    Args:
        ax (3D axis): the ax containing the plot of interest
        angles (list): the list of angles (in degree) under which to
                    show the plot.
        output : name of the output file. The extension determines the
                kind of animation used.
        **kwargs:
            - width : in inches
            - heigth: in inches
            - framerate : frames per second
            - delay : delay between frames in milliseconds
            - repeat : True or False (.gif only)
    """

    output_ext = os.path.splitext(output)[1]
    files = make_views(ax,angles, **kwargs)
    D = { '.mp4' : make_movie,
          '.ogv' : make_movie,
          '.gif': make_gif ,
          '.jpeg': make_strip,
          '.png':make_strip}

    D[output_ext](files,output,**kwargs)


def crop_center(pil_img, crop_width, crop_height):
    img_width, img_height = pil_img.size
    return pil_img.crop(((img_width - crop_width) // 1.7,
                         (img_height - crop_height) // 1.2,
                         (img_width + crop_width) // 2,
                         (img_height + crop_height) // 2))



##### Create projections
def CreateProjections(sample_num,prj_angle):
    # convert image to numpy  array
    img_array = []
    prj_image = []

    # create figure
    fig = plt.figure(figsize=(10, 7))

    # setting values to rows and column variables
    rows = 4
    columns = 4

    for projection_num in range(0, NO_OF_PROJECTIONS):

prj_image.append(Image.open(r"C:\\Users\\dele.ogunjumelo\\3drecon\\"+f"tmprot_00{project
ion_num}.jpeg"))

      # change to grayscale
      prj_image[projection_num] = ImageOps.grayscale(prj_image[projection_num])

      # remove white background
      prj_image[projection_num] = ImageOps.invert(prj_image[projection_num])

      # crop image
      prj_image[projection_num]= crop_center(prj_image[projection_num],130,130)
      # Adds a subplot at the first position
      fig.add_subplot(rows, columns, (projection_num+1))

      # showing image
      plt.imshow(prj_image[projection_num])
      plt.axis('off')
      plt.title(prj_angle[projection_num])

      # save the image with a new filename
      prj_filename                                                                                =
"C:\\Users\\dele.ogunjumelo\\3drecon\\ssim_data\\input\\"+f"prj_data_{sample_num}_{proje
ction_num}.jpeg"
      img_array.append(img_to_array(prj_image[projection_num]))
      print(img_array[projection_num].shape)
      save_img(prj_filename, img_array[projection_num])
      plt.close()



if __name__ == '__main__':

  pdf_x_data = []
  pdf_y_data = []
  pdf_z_data = []
  mu_values = []

  for k in range(0, SAMPLES):
     mu_values.append(np.array([u_values[k],u_values[k]]))

  # Fixed covariance matrix
  sigma = np.array([[1, -.5], [-.5, 1]])

  pdf_samples = []
  input_samples =[]

  #X, Y ranges are constructed with the “meshgrid” function from numpy.
  x, y = np.mgrid[-3:3:.1, -3:3:.1]

  X = np.stack((x.ravel(), y.ravel())).T

  for k in range(0, SAMPLES):
      mu = mu_values[k]
      #norm = multivariate_normal_pdf(X, list(mu_values[k]), sigma*mu).reshape(x.shape)
      norm = multivariate_normal_pdf(X, list(mu), sigma).reshape(x.shape)
      pdf_z_data.append(norm)
```

```
        pdf_x_data.append(x)
        pdf_y_data.append(y)
        ground_truth_3d_data = np.concatenate(( pdf_x_data[k],  pdf_y_data[k],  pdf_z_data[k]))
        gt_filename                                                                                     =
"C:\\Users\\dele.ogunjumelo\\3drecon\\ssim_data\\output\\"+f"ground_truth_{k}.jpeg"
        figit = get_mvd_image(norm)
        figit.figure.savefig(gt_filename)
        plt.close()
        gt_image = imageio.imread(gt_filename)
        gt_image = cv2.cvtColor(gt_image, cv2.COLOR_BGR2GRAY)
        gt_image = cv2.bitwise_not(gt_image)  # remove white background
        image_array = img_to_array(gt_image)   # convert image to numpy array
        save_img(gt_filename, image_array)

        # Do projections
        angles = np.linspace(0,180, 9)[:-1] # Take 8 angles between 0 and  180
        print("Projection Angles",angles)
        rotanimate(figit, angles,'movie.jpeg',delay=20)
        CreateProjections(k,angles)
```

Appendix 2 Program for preparing 3-D flame data

```
# TensorFlow is an open source machine learning library
import tensorflow as tf

import matplotlib.pyplot as plt
from matplotlib import cm
from mpl_toolkits.mplot3d import axes3d
import os, sys
import numpy as np

# Pandas is a data manipulation library
import pandas as pd

# Math is Python's math library
import math
import scipy.io

import scipy.stats

from PIL import Image

# Keras is TensorFlow's high-level API for deep learning
from tensorflow import keras


# Improting Image class from PIL module
from PIL import Image, ImageOps

from scipy import misc
from numpy import asarray
from numpy import array
from numpy import empty
from numpy import ones
from numpy import zeros
from numpy import save

import imageio
import io
import cv2
import IPython
import random
import statistics
```

```
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.preprocessing.image import array_to_img
from keras.preprocessing.image import save_img

from skimage.metrics import structural_similarity as ssim
from skimage.metrics import mean_squared_error
from skimage.transform import radon, rescale
from skimage.transform import iradon_sart

from skimage.transform import resize



y=0
x=15
h=160
w=95

box = (180, 138, 550, 450)



def crop_center(pil_img, crop_width, crop_height):
    img_width, img_height = pil_img.size
    return pil_img.crop(((img_width - crop_width) // 1.7,
                         (img_height - crop_height) // 1.2,
                         (img_width + crop_width) // 2,
                         (img_height + crop_height) // 2))





## Capture projection data for SET1
PROJECTIONS = 8
projection_data = []
projection_samples = []
projection_samples_1 = []

# setting values to rows and column variables
row_s = 4
column_s = 4
```

```
# create figure
fig1 = plt.figure(figsize=(10, 7))

for projection_num in range(0, PROJECTIONS):
   fname_in = f"/content/drive/MyDrive/3d_recon/flame_data/Set1/"+f"R{projection_num}.b
mp"
   input_image = imageio.imread(fname_in);
   input_image = input_image[5:150,20:110]    #crop image

   # convert image to numpy array
   projection_data.append(asarray(input_image))
   projection_samples.append(projection_data[projection_num])
   projection_samples_1.append(projection_data[projection_num])

   # Adds a subplot at the 1st position
   fig1.add_subplot(row_s, column_s, (projection_num+1))

   # showing image
   plt.imshow(projection_samples[projection_num])
   plt.axis('off')



projection_samples[7].shape

len(projection_samples)


###################################################################
##generate data sets using Morphological transformations
##https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html
###################################################################

###generate projection dataset using original flame data
def transform_1(reference_image):
kernel = np.ones((5,5),np.uint8)
return cv2.erode(reference_image,kernel,iterations = 1

def transform_2(reference_image):
kernel = np.ones((5,5),np.uint8)
return cv2.erode(reference_image,kernel,iterations = 2
################################################################################
##
### generate data for flame condition 1   -  170 samples
##
##  [ using projection samples: 0 - 7 ]
##
################################################################################
##
PROJECTION_DATASET_0_1 = 119  #70%
for dataset in range(0, PROJECTION_DATASET_0_1):
for projection_num in range(0, PROJECTIONS):
generated_samples.append(projection_samples[projection_num])
projection_dataset.append(generated_samples)
generated_samples = []

PROJECTION_DATASET_0_2 = 26
for dataset in range(0, PROJECTION_DATASET_0_2):
for projection_num in range(0, PROJECTIONS):
generated_samples.append(transform_1(projection_samples[projection_num]))
projection_dataset.append(generated_samples)
generated_samples = []

PROJECTION_DATASET_0_3 = 25
for dataset in range(0, PROJECTION_DATASET_0_3):
for projection_num in range(0, PROJECTIONS):
generated_samples.append(transform_2(projection_samples[projection_num]))
projection_dataset.append(generated_samples)
generated_samples = []

len(projection_dataset)

projection_samples_data = np.asarray(projection_dataset)
projection_samples_data.shape
```

Appendix 3 Program for CNN model implementation used for 3-D flame reconstruction

```
recon_model =  tf.keras.Sequential()
recon_model.add(Conv2-
D(8, kernel_size=(3, 3),activation='linear',input_shape=(200,25,1),padding='same'))

recon_model.add(Conv2-D(16, (3, 3), activation='linear',padding='same'))
recon_model.add(Conv2-D(16, (3, 3), activation='linear',padding='same'))
recon_model.add(MaxPooling2-D((3, 3),padding='same'))

recon_model.add(Conv2-D(32, (3, 3), activation='linear',padding='same'))
recon_model.add(Conv2-D(32, (3, 3), activation='linear',padding='same'))
recon_model.add(MaxPooling2-D((3, 3),padding='same'))

recon_model.add(Conv2-D(64, (3, 3), activation='linear',padding='same'))
recon_model.add(Conv2-D(64, (3, 3), activation='linear',padding='same'))
recon_model.add(MaxPooling2-D((3, 3),padding='same'))

recon_model.add(Flatten())

recon_model.add(Dense(261000, activation='linear'))

optimizer = keras.optimizers.Adam(learning_rate=0.0001)
recon_model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])

recon_model.summary()
```

Appendix 4 Program for CNN model implementation used for numerical simulation

```
recon_model =  tf.keras.Sequential()

recon_model.add(Conv2D(8, kernel_size=(3, 3),activation='linear',input_shape=(80,10,1),pad
ding='same'))

recon_model.add(Conv2D(16, (3, 3), activation='linear',padding='same'))
recon_model.add(Conv2D(16, (3, 3), activation='linear',padding='same'))
recon_model.add(MaxPooling2D((3, 3),padding='same'))

recon_model.add(Conv2D(32, (3, 3), activation='linear',padding='same'))
recon_model.add(Conv2D(32, (3, 3), activation='linear',padding='same'))
recon_model.add(MaxPooling2D((3, 3),padding='same'))

recon_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
recon_model.add(Conv2D(64, (3, 3), activation='linear',padding='same'))
recon_model.add(MaxPooling2D((3, 3),padding='same'))

recon_model.add(Flatten())

recon_model.add(Dense(10800, activation='linear'))  ## Used for generating results

optimizer = keras.optimizers.Adam(learning_rate=0.00008)  ## The learning rate. Defaults to
0.001
recon_model.compile(optimizer=optimizer, loss='mse', metrics=['mae'])  ## Used for generati
ng results

recon_model.summary()
```

Appendix 5 Program for CNN training and evaluation metrics

```
#
#history_1=recon_model.fit(x_train, y_train, epochs=100, batch_size=8, validation_data=(x_
validate, y_validate))


history_1=recon_model.fit(x_train, y_train, epochs=1000, batch_size=16, validation_data=(x
_validate, y_validate))


# Plot a graph of the loss
train_loss = history_1.history['loss']
val_loss = history_1.history['val_loss']


epochs = range(1, len(train_loss) + 1)


plt.plot(epochs, train_loss, 'g.', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()



plt.plot(epochs[SKIP:], train_loss[SKIP:], 'g.', label='Training loss')
plt.plot(epochs[SKIP:], val_loss[SKIP:], 'b.', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
plt.clf()


# Plot of a  graph of mean absolute error
train_mae = history_1.history['mae']
val_mae = history_1.history['val_mae']
```

```
plt.plot(epochs[SKIP:], train_mae[SKIP:], 'g.', label='Training MAE')
plt.plot(epochs[SKIP:], val_mae[SKIP:], 'b.', label='Validation MAE')
plt.title('Training and validation mean absolute error')
plt.xlabel('Epochs')
plt.ylabel('MAE')
plt.legend()
plt.show()
```

Appendix 6 Program for checking the accuracy of reconstructed flame

```
from skimage.metrics import structural_similarity as ssim
from skimage.metrics import mean_squared_error

box = (190, 145, 560, 400)


#check accuracy of reconstructed image
gt_orig_filename = f"/content/drive/MyDrive/3d_recon/mldata_0_100/gt_test.jpeg"
gt_orig_image.figure.savefig(gt_orig_filename)
orig_gt_image = Image.open(gt_orig_filename)
orig_gt_image = orig_gt_image.convert('L')
orig_gt_image = orig_gt_image.crop(box)
orig_gt_image = np.array(orig_gt_image)


gt_predicted_filename = f"/content/drive/MyDrive/3d_recon/mldata_0_100/gt_predicted.jpeg
"
gt_predicted_image.figure.savefig(gt_predicted_filename)
predicted_image =  Image.open(gt_predicted_filename)
predicted_image =  predicted_image.convert('L')
predicted_image =  predicted_image.crop(box)
predicted_image =  np.array(predicted_image)

mse_none = mean_squared_error(orig_gt_image, orig_gt_image)
ssim_none = ssim(orig_gt_image, orig_gt_image,
data_range=orig_gt_image.max() - orig_gt_image.min())

mse_predicted  = mean_squared_error(orig_gt_image, predicted_image)
ssim_predicted = ssim(orig_gt_image, predicted_image,
data_range=predicted_image.max() - predicted_image.min())

fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 4),
sharex=True, sharey=True)
ax = axes.ravel()
label = 'MSE: {:.2f}, SSIM: {:.2f}'

ax[0].imshow(orig_gt_image)
ax[0].set_xlabel(label.format(mse_none, ssim_none))
ax[0].set_title('Ground Truth Image')
```

```
ax[1].imshow(predicted_image)
ax[1].set_xlabel(label.format(mse_predicted, ssim_predicted))
ax[1].set_title('Predicted Image')

plt.tight_layout()
plt.show()
```