Deep Learning and Computer Vision for Text Detection and Recognition

BY

SACHITH M. GUNAWARDANE

2024

ABSTRACT

This research addresses a critical gap by exploring the integration of advanced deep learning and computer vision for text detection and recognition in images, including various structures like tables, lists, paragraphs, and headings. To quantitatively assess recognition accuracy, the study adopts a specific data template derived from the Whitireia/WelTec School of IT Programme Handbook's Course Descriptor (CD). Following the principles of the design science research methodology, four primary artefacts were developed, encompassing a data model, a tool for generating Ground Truth (GT) from PDF CD documents, an input dataset for the optical word recognition pipeline, and the optical word detection and recognition pipeline itself. The data modelling component utilises an Entity-Relationship (ER) model stored CD information. Data is managed in a MySQL database, and PDF information is processed using NLTK and PyPDF libraries. The text detection and recognition model's input is generated using the Fitz library, introducing random skewness and tilt during PDF-to-image conversion.

The outcome is a comprehensive text detection and recognition pipeline covering preprocessing, segmentation, word recognition, and postprocessing. Preprocessing corrects skewness and tilt through three approaches: Computer Vision (CV) with Hough Transformation, Multilayer Perceptron (MLP) with Fast Fourier Transformation (FFT), and Convolutional Neural Network (CNN) with FFT. The segmented image is processed to detect table and non-table structures, and text is parsed into lines and words. A Convolutional Recurrent Neural Network (CRNN) achieves high accuracy in word recognition. Postprocessing extracts information using regular expressions. Evaluation metrics include Mean Squared Error (MSE), confusion matrix for preprocessing, letter accuracy, and Connectionist Temporal Classification (CTC) loss for CRNN models. Sentence matching via ‘difflib.SequenceMatcher’ evaluates each preprocessing technique against CD information areas.

Despite limitations like CRNN's challenge with numerical (number-based) and single-digit words, the study highlights innovative text detection and recognition pipeline approaches. Further improvements can be introduced by expanding training datasets and integrating Optical Character Recognition (OCR) models for non-dictionary words to overcome the above-stated

limitations. The CRNN model demonstrates notable accuracy in predicting words, showcasing potential for innovative refinement in text detection and recognition pipelines.

The developed pipeline achieved a sentence-matching ratio of 0.79 when images were devoid of distortions like skew and tilt. However, when dealing with distorted images, the pipeline registered a lower sentence matching ratio of 0.48, underscoring the adverse impact of skewness and tilt on text detection and recognition algorithms. Skew and tilt correction models, specifically OpenCV with Hough transformation, outperformed CNN with FFT for the CD template, achieving a sentence-matching ratio of 0.62 compared to 0.55. These findings underscore the impressive capabilities of the introduced Optical Word Recognition (OWR) pipeline in effectively extracting text from complex documents and successfully navigating challenges posed by skewness and tilt distortions.

In addition to its primary contribution of enhancing the applicability of computer vision and deep learning-based optical word recognition, particularly for processing complex documents featuring structures like tables and lists, the artefact produced in this study addresses real-world challenges encountered by the School of Innovation Design and Technology and similar institutions in managing Course Descriptor (CD) documents. Currently, the CD is managed as a digital document within the institute, and it is regularly referenced for tasks such as creating course outlines, moderation forms, result summaries, and other recurring activities throughout the course delivery cycle. However, manually inputting this CD information into each of these areas is susceptible to human errors and adds a significant administrative burden. Further, the CD template used in this research is utilized across various programs at the Wellington Institute of Technology (WelTec) and Whitireia Polytechnic. These institutions can universally implement the normalized ER schema, GT generation tool, and OWR pipeline developed here to digitally manage course descriptors. This application serves as an efficient automation solution for educators and administrative personnel, simplifying CD management and facilitating cross-referencing for assignments and other documentation tasks.

Keywords: Deep Learning, Computer Vision, Text Detection, Text Recognition, Image Processing, Table Detection, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Design Science Research Methodology, Entity-Relationship (ER) Model, Natural Language Toolkit (NLTK), Skewness and Tilt Correction, Text Segmentation, Convolutional Recurrent Neural network (CRNN)

ACKNOWLEDGEMENTS

This work was completed as part of the Master of Information Technology 2024 program at the Whitireia & WelTec Education Institute, Wellington, New Zealand. I am profoundly grateful for the opportunity to undertake this significant academic endeavor in such a supportive and enriching environment.

I would like to express my deepest appreciation to my primary supervisor, Dr. Waqar Khan, as well as Tony Assadi and Dr. Marta Vos, for their steadfast support and expert guidance. Their involvement in every phase of my research was invaluable, providing me with the necessary insights and constructive feedback to navigate this complex study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
CD	Course Descriptor
CNN	Convolutional Neural Networks
CRF	Conditional Random Fields
CRNN	Convolutional Recurrent Neural Network
CTC	Connectionist Temporal Classification
CV	Computer Vision
DFT	Discrete Fourier Transform
DSR	Design Science Research
EDA	Exploratory Data Analysis
ER	Entity Relationship
FFT	Fast Fourier Transform
GT	Ground Truth
IE	Information Extraction
IS	Information Systems
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
MSE	Mean Squared Error
MSER	Maximally Stable Extremal Regions
NMS	Non-Maximal Suppression
OCR	Optical Character Recognition
OWR	Optical Word Recognition
PDF	Portable Document Format
RDBMS	Relational Database Management System
RE	Regular Expresion
RNN	Recurrent Neural Networks
WelTec	Wellington Institute of Technology

CHAPTER 1

INTRODUCTION

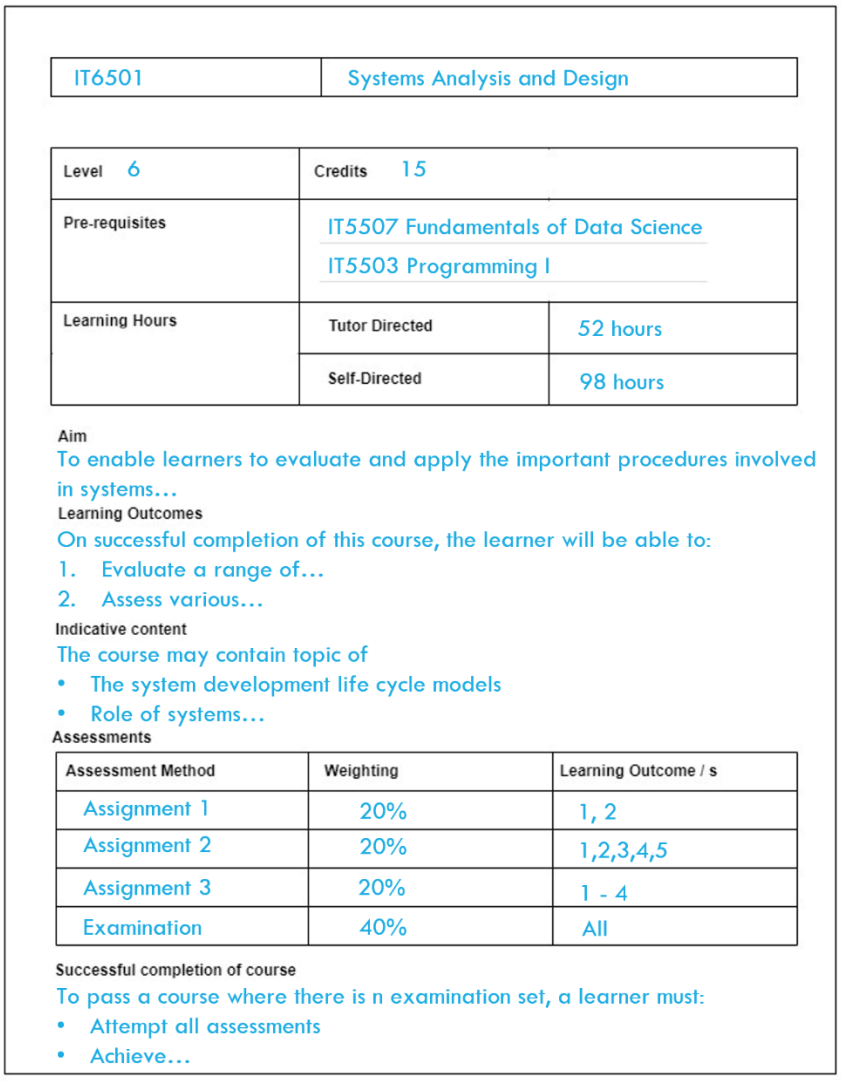
In recent years, research in the field of text detection and recognition has primarily been focused on financial document processing [1], [2], [3], which is an essential operation in many businesses to process financial statements and billing invoices. However, documents containing figures, lists, and tables have been overlooked due to their diverse and complex content. The complexity of content is driven by its type. For example, the content could be in the form of figures, lists, paragraphs, or headings. Each such type would define a dimension of complexity. This research aims to design, build, and evaluate a functional artefact for identifying and categorising textual attributes within complex documents containing lists and tabular data and provide search capability on *Information Extracted* (IE).

In the context of machine learning, data is the most crucial resource. This research aims to adopt machine learning-based approaches like *Optical Character Recognition* (OCR), eventually leading to *Optical Word Recognition* (OWR). The objective is to develop models that recognise words in a specific section of a larger document. To achieve this, the recognition system uses images of each page of a *Portable Document Format* (PDF) or a Word document as input data, and its output determines the content in each image.

The recognition can be complex due to the original document's complexity. The data in the given document needs to follow a template to perform quantitative analysis on prediction accuracy. The advantage of using template-based data is that OWR model accuracy can be analysed at a finer level. For example, conclusions could be drawn about the performance of models on tabular content independently. Similarly, independent and collective analysis can be achieved for each dimension, such as list, tabular, and text content with numbers and other complex data.

For this study, the *Course Descriptor* (CD) template for achievement-based programmes of the School of Innovation Design and Technology [4], [5] is used. However, various programmes at Wellington Institute of Technology (WelTec) and Whitireia Polytechnic use the same CD template. This study only focused on CDs for the Bachelor of Information Technology degree programme.

Figure **1** illustrates the course descriptor template. The blue text represents the information extracted by the designed model, while the black text and tables depict the document's structural elements. The CD starts with an alphanumeric course code (e.g., IT6501), where the first two characters represent the study area, and the next four digits uniquely identify the course. The course title follows the course code. The course level, credit, prerequisites, and learning hours are arranged in a tabular format where there are no actual lines in the document, but the placement of information conveys the same. The credit and level hold numeric values, while the prerequisites refer to other courses that need completion before taking this course, with multi-values comprising the course code and title.



IT6501	Systems Analysis and Design

Level 6	Credits 15	
Pre-requisites	IT5507 Fundamentals of Data Science IT5503 Programming I	
Learning Hours	Tutor Directed	52 hours
	Self-Directed	98 hours

Aim

To enable learners to evaluate and apply the important procedures involved in systems…

Learning Outcomes

On successful completion of this course, the learner will be able to:

1. Evaluate a range of…
2. Assess various…

Indicative content

The course may contain topic of

- The system development life cycle models
- Role of systems…

Assessments

Assessment Method	Weighting	Learning Outcome / s
Assignment 1	20%	1, 2
Assignment 2	20%	1,2,3,4,5
Assignment 3	20%	1 - 4
Examination	40%	All

Successful completion of course

To pass a course where there is n examination set, a learner must:

- Attempt all assessments
- Achieve…

Figure 1: Template for Course Descriptor.

In Figure **1**, the learning hours hold two numeric values: tutor directed and self-directed hours. The aim follows the template, consisting of textual content. The learning outcomes, indicative

content, and successful completion of course contents are bulleted lists. Lastly, the assessment tabular content is one of the most challenging sections, with text and numerical content having multiple representations of the same interpretation.

Research efforts have not solely concentrated on financial documents; they have been primarily aimed at resolving issues pertaining to text detection and recognition in isolation. For instance, endeavours have been made to implement deep-learning models for word identification after detection [6], [7], [8], [9], [10], [11], [12], [13], [14], [15], [16], [17]. Hence, it is apparent that a research gap exists in the domain of text detection and recognition within intricate documents such as course descriptors, which frequently encompass tables and lists comprehensively as a pipeline. In light of this, the following research questions have been formulated to bridge this gap:

> **RQ1:** How does distortion, such as skew and tilt, present in scanned images impact the text detection and recognition process built on computer vision and deep learning algorithms?
>
> **RQ2:** Which machine learning pipeline, constructed with principles from computer vision and deep learning, demonstrates better accuracy in detecting and recognizing text?

These research questions serve as guiding principles for the study. To address these research questions, the research aims to design and construct a suite of artefacts to facilitate data collection, evaluation, and text detection and recognition. The research objectives are intricately linked to the phases of the research and the development of artefacts that aid in resolving the overarching problem. Thus, the objectives of this research directly align with the research questions as follows:

> **RO1:** Develop a normalised ER model (also known as a repository) to store and retrieve section-wise CD content efficiently.
>
> **RO2:** Generate a *Ground Truth* (GT) dataset by extracting content from PDF or MS Word programme documents using Python libraries which extract text such as Tika [18], PyPDF2 [19] and docx2python [20].

RO3: Develop an OWR model using *Artificial Intelligence* (AI) and *Computer Vision* (CV) techniques to perform IE on image-based CD documents.

RO4: Evaluate the performance of the OWR model by comparing the accuracy with GT and accessing its performance when scanned images are tilted and skewed.

To evaluate the accuracy of the extracted content compared to the actual content, I created an *entity-relationship* (ER) model as part of RO1. This ER model was derived from the fields outlined in the course descriptor, ultimately establishing a comprehensive database of courses. While the primary focus of my research has been to evaluate the accuracy of the text detection and recognition pipeline, it has also paved the way for the automated management of course descriptors and the quality assurance process within the School of IT and other academic institutions. Currently, course descriptor content is solely stored in PDF format, lacking a structured database. With the implementation of this database, coupled with the artefact designed to extract ground truth data in RO2, course descriptor information can now be efficiently managed electronically. Additionally, the stored information can be utilized to generate course outlines, moderation forms, result summaries, and tasks that are recurring throughout the course delivery cycle.

RO3 entails designing and developing an optical word recognition pipeline using computer vision and deep learning techniques, including *Convolutional Neural Network* (CNN) and *Recurrent Neural Network* (RNN) models. This artefact is designed to address skewness and tilt distortions inherent in scanned images containing complex content, allowing for accurate information extraction. RO4 focuses on developing an artefact to calculate the OWR pipeline's accuracy by comparing its output against the ground truth established in RO2.

1.1 THESIS ORGANISATION

This thesis is structured into five core chapters, each contributing significantly to the overall research process and discussion:

- Chapter 1: Introduction
- Chapter 2: Literature Review
- Chapter 3: Methodology
- Chapter 4: Results and Discussion

- Chapter 5: Conclusion

The initial chapter serves as an introduction, providing the reader with a comprehensive overview of the research. It delves into the historical context, elucidates the backdrop of contemporary developments, and underscores the need for further exploration, specifically in the realm of documents such as course descriptors featuring lists and tabular data for text detection and recognition. Chapter two, the literature review, explores existing literature within the research's context. It systematically synthesises a wide array of literature, encompassing text detection and recognition techniques. This chapter culminates by addressing the challenges faced by contemporary models when dealing with text detection in documents of the course descriptor genre. The third chapter, Methodology, outlines the application of the *Design Science Research* (DSR) paradigm, specifically within the research context. It delineates the procedural sequence and design choices made at each research stage, encompassing development, evaluation, and enhancement phases. Chapter four amalgamates the research findings and in-depth discussions. It presents the research outcomes and engages in detailed discussions, aligning the findings with the core research questions. The final chapter, Conclusion, encapsulates the key findings, highlighting their broader implications, and extends recommendations for potential future work in the field.

CHAPTER 2

LITERATURE REVIEW

CHAPTER 3

METHODOLOGY

Design Science is a dynamic discipline dedicated to developing innovative solutions, improving existing systems, and creating artefacts that enhance human efficiency, particularly in addressing real-world challenges within societal and organisational contexts. Unlike seeking optimal outcomes, design science focuses on finding satisfactory solutions tailored to specific contexts [32]. This research embarks on the design science methodology to address a real-world problem by crafting a tool/artefact for detecting and recognising text in complex image-based documents. The subsequent sections delve into the practices and processes of the DSR methodology, exploring its application in the study context.

This chapter is organised into the following subsections.

3.1 Introduction to DSR

3.2 Problem Identification

3.3 Suggestions

3.4 Development

3.4 Evaluation

3.1 DESIGN SCIENCE RESEARCH METHODOLOGY

DSR is a rigorous process for designing artefacts to solve problems, evaluate designs, and effectively communicate results [33]. Its application can potentially bridge the gap between theory and practice [33]. DSR distinguishes itself from other methodologies like case study and action research, aligning with design science concepts. Action research and case studies are more closely linked to natural and social science. The distinction among these three types primarily lies in their research methods and objectives; action research aims to resolve or elucidate issues within a specific system by generating both practical and theoretical knowledge, case study aids in comprehending social phenomena, and DSR focuses on creating artefacts that facilitate viable solutions to real-world problems. This distinction underscores the appropriateness of DSR as the preferred research method when the study's primary goal is to design and develop a tool or artefact for text detection and recognition.

Although there are variations in DSR methodology among different contributors, the fundamental phases and elements generally remain consistent. In the context of this research, the selected variation of DSR revolves around a five-step process proposed by Vaishnavi and Kuechler [34]. Further, on top of the five steps, Manson [35] elaborates on the potential outcomes that can arise at each step of the DSR process. Figure 2 visually delineates the five distinct stages of this methodological process.

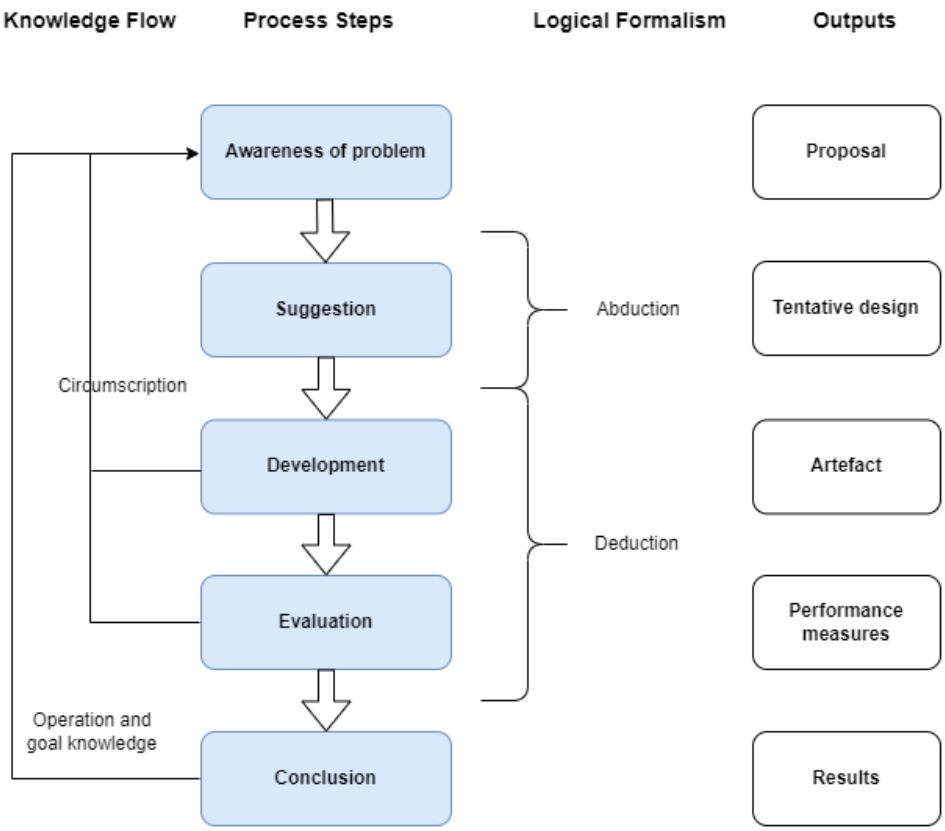


Figure 2: Design Science Research - Process Steps, as illustrated in [35].

3.1.1 Overview of the first stage: Awareness of the problem

During the initial phase of problem awareness, researchers identify and comprehend the problem, including defining the required performance for the considered system [34]. As outlined by Manson [35], upon the completion of the problem awareness stage, the researcher gains the ability to present a proposal, marking the initiation of subsequent research endeavours. This proposal should encompass substantiating evidence of the identified problem, an in-depth characterisation of the external environment, and their interactions with the envisioned artefact. It should also involve the establishment of metrics and criteria governing

the acceptance of the artefact, along with clarifying the involved stakeholders and the categories of problems to which the artefact may be linked.

3.1.2 Overview of the second stage: Suggestions

In the suggestion stage, researchers propose solutions using the abductive scientific method, employing creativity and prior knowledge. The output is one or more tentative designs to solve the defined problem [34]. Assumptions guiding artefact construction are clarified, ensuring a robust foundation.

3.1.3 Overview of the third stage: Development

The development stage involves the construction of the chosen artefact, with researchers justifying tool choices, explaining causal relationships, and validating the artefact. Circumscription, representing interactions between steps, becomes crucial for a nuanced understanding and insightful learning from the construction process [34].

3.1.4 Overview of the fourth stage: Evaluation

In the evaluation stage, artefacts developed in the previous stage undergo thorough assessment against research requirements. If necessary, researchers can return to the awareness step to refine their understanding of the problem. Performance measures are thoughtfully developed, and the evaluation mechanisms and results are meticulously detailed [35].

3.1.5 Overview of the fifth stage: Conclusion

The conclusion stage becomes the platform for presenting research findings. If the awareness of the problem is insufficient, the design cycle may restart, offering valuable insights and addressing gaps in existing theories. Here, the researcher analyses, consolidates, and records results, fostering knowledge dissemination for researchers and a wider audience accessing the research. This iterative process contributes significantly to the refinement of knowledge and the creation of impactful artefacts.

3.2 PROBLEM IDENTIFICATION

The current research addresses a critical issue within the information (text) extraction domain, particularly in the context of complex documents featuring figures, lists, and tables. The

challenge stems from the dearth of literature guiding the implementation of computer vision and deep learning techniques in handling such diverse and intricate content. As highlighted in the literature review, previous research primarily concentrates on text detection and recognition in financial documents like invoices and order forms [1], [2], [3]. While crucial for business operations, this focus neglects the broader scope of applying machine learning models to complex documents. Consequently, a noticeable research gap exists in establishing text detection and recognition (IE) pipelines tailored for complex documents.

The significance of data cannot be overstated within the realm of machine learning and deep learning. Ensuring accuracy during both the training and testing phases of models is paramount. Machine learning models often excel with training datasets but falter when confronted with unseen data. Considering this, the research methodology employs publicly available datasets for model building and training, with the creation of a novel dataset for rigorous testing. Adopting a template with a complex structure to create test data, such as a CD from the Bachelor of Information Technology degree program, allows research to gain a quality dataset for testing. This template adherence facilitates quantitative analysis of prediction accuracy, allowing for a detailed examination of accuracy levels in tabular content and lists.

Given the real-world scenario where image-based documents are predominantly processed using scanners or cameras in handheld devices, the issue of skewness or tilt in the captured images becomes inevitable. Consequently, the research intends to validate the developed text detection and recognition pipeline's performance on documents with inherent skewness or tilt, assessing its prediction accuracy under these conditions. The evaluation of the pipeline's accuracy unfolds at multiple levels. For specific tasks, model accuracy is assessed on training and validation datasets using *Mean Squared Error* (MSE) and confusion matrix, extracting metrics such as accuracy, precision, recall, and F1 score (Refer to section 3.5.2 for definitions and equations). A model surpassing 80% accuracy on validation data qualifies as a candidate for inclusion in the text detection and recognition pipeline. The pipeline accuracy is further measured by measuring sentence similarity, employing a ratio in '**SequenceMatcher**' class, which comes with the '**difflib**' library. The ratio function compares two sentences and yields a similarity score ranging from 0 to 1, considering number of correct words in the sentence and their order. A sentence matching ratio exceeding 0.60 has been set as the target level OWR pipeline to be acceptable for the intricacies of complex documents. This research was approved

by the Whitireia School of IT Board of Studies, and the approval letter is in **Error! Reference s ource not found.** Appendix A.

3.3 SUGGESTIONS

3.3.1 Literature Search

The systematic approach employed for data collection involved the use of key search terms such as “character recognition”, “computer vision”, “convolutional neural network”, “deep learning”, “neural network”, “document images”, “document passing”, “entity extraction”, “heterogeneous data”, “image processing”, “information extraction”, “invoice processing”, “NLP”, “natural language processing”, “object detection”, “OCR”, “optical character recognition”, “segmentation”, “pattern recognition”, “table detection”, “text detection”, “text localisation”, “text recognition”, and “unstructured data”. Employing Boolean arguments “AND” and “OR” enhanced search efficiency. The literature search was refined to English-language articles published between 2015 and 2023, focusing on those released predominantly after 2020 to ensure recent research.

In establishing criteria for inclusion and exclusion, a deliberate effort was made to select literature directly relevant to image text processing, character recognition, word recognition, image-based document reading, computer vision, and deep learning. Exclusion criteria were defined to eliminate studies solely focused on image processing without text processing, such as deep convolutional neural networks for computer-aided detection [36]. Additionally, literature published in languages other than English was excluded. To maintain the relevance of the literature evaluation, emphasis was placed on recent research, with the majority of collected articles from the last five years and older articles being consulted solely for historical context extraction.

A total of 47 items meeting the inclusion criteria were identified through this comprehensive search. Although the research methodologies, datasets, and statistical techniques varied across these investigations, a predominant methodology involved design science or quantitative empirical analysis for experiment comparison. Knowledge gleaned from these past research papers played a crucial role in shaping the text detection and recognition pipeline. For instance, the CRNN model discussed in the paper “An End-to-End Trainable Neural Network for Image-

based Sequence Recognition and Its Application to Scene Text Recognition" [14] was adapted and deployed as the OWR model in the proposed pipeline.

3.3.2 Solution Objectives

Within the suggestion stage, solutions are presented to address the research questions effectively. This DSR-specific phase seamlessly aligns with the formulation of solutions for the identified research objectives elucidated in the introduction section. The implementation of the research necessitates the construction of multiple artefacts to realise the final objective. The research objectives are meticulously formulated to encompass all the components involved.

First and foremost, developing a normalised ER model-based data store is undertaken to store and retrieve section-wise CD content efficiently. This foundational step ensures an adequate infrastructure for handling the diverse content within the CD.

The second objective involves the generation of a GT dataset through the extraction of content from PDF files. This process contributes to creating a comprehensive dataset that serves as a benchmark for testing the text detection and recognition pipeline.

The third objective focuses on developing the pipeline, leveraging CV and AI to facilitate text detection and recognition on image-based CD documents. This involves the integration of advanced technologies to ensure the pipeline's efficacy in handling the complexities inherent in the CD content.

The final objective encompasses evaluating the performance of the text detection and recognition pipeline. This evaluation is conducted by comparing the accuracy of sentence recognition against the GT, considering variations introduced by skewed and tilted documentation. Including such variations reflects the real-world challenges posed by document capture through scanning or imaging devices.

Figure 3 illustrates a high-level activity diagram to provide a comprehensive overview and visualisation of the solution for all the research objectives. This diagram visualises the interconnected processes and activities involved in achieving the outlined research objectives, emphasising the holistic approach undertaken in this research endeavour.

Activities falling under the database model are devised to address the first research objective. A crucial outcome of this research is the management of information extracted from course descriptors. Constructing a schema based on a normalised ER model is necessary to achieve this. This schema maintains integrity among various entities within CD documentation and ensures a swift retrieval process. Significantly, it facilitates the storage of multiple versions of the same document, extracted through distinct models designed for the text detection and recognition pipeline. Additionally, the schema allows for attribute-level accuracy verification, providing valuable insights into the performance of the tools employed. The MySQL open-source relational database is chosen as the hosting platform for the CD schema.

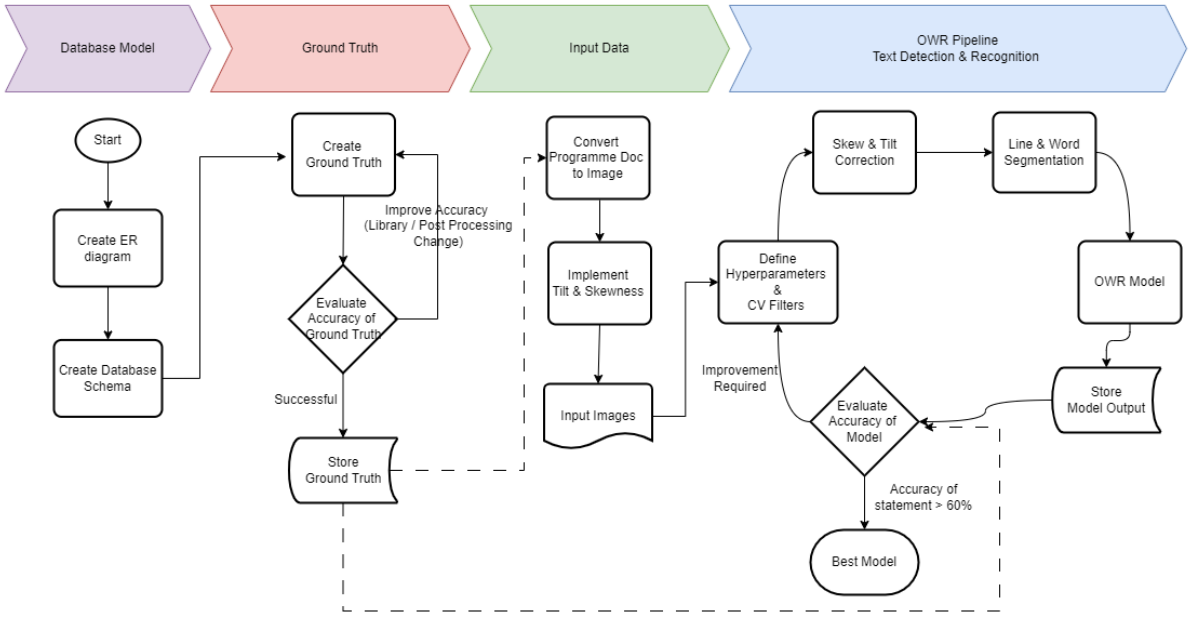


Figure 3: Activity Flow Diagram.

Under the ground truth initiative, activities are designed to create an artefact capable of extracting text from PDF documents and processing/storing it in the schema designed for CDs. This module serves a dual purpose, enabling the processing of similar PDF documents adhering to the CD template with modifications in the postprocessing module. The postprocessing module is a shared component built to process CD specification attributes systematically.

After the ground truth creation, the next step is input data, which involves generating input data for the testing process. This is achieved through another artefact that reads PDF file pages sequentially, converts the read pages into images, and introduces random skewness and tilt.

This step simulates real-world scenarios where image-based documents are captured with variations.

The final stage in this sequence is the OWR pipeline, designed to detect and recognise text in image-based CD documents. The processed information is then integrated into the schema created during the database model stage. The OWR pipeline is segmented into filters, hyperparameter tuning, a module for handling/correcting skewness and tilt, tabular area detection, and segmentation into lines and words. The concluding module is dedicated to recognising words. Each of these sections will be expounded upon in greater detail in the subsequent section, development.

3.4 DEVELOPMENT

This subsection elucidates the methodologies and procedures implemented during the developmental stage. The section encompasses schema design, the persistence layer for extracted information, the design of artefacts to extract GT, the design of artefacts for creating a novel dataset for testing using CDs, the pipeline for text detection and recognition, which internally embeds multiple models for skewness and tilt correction, a model/logic for the identification of tabular and non-tabular information, segmentation models for lines and words, the OWR model, the postprocessing model, and lastly, an evaluation model to assess the accuracy of sentences identified by the pipeline.

3.4.1 Data Model

Employing a normalised ER schema coupled with a relational database is a cornerstone of database management systems, particularly in the context of information extraction from complex documents. Adopting a normalised ER schema enhances data integrity and contributes to efficient storage, streamlined query performance, and a more adaptable change management process.

The normalised ER schema is pivotal in upholding data integrity by systematically mitigating redundancy and dependency issues. This is achieved by organising data into logical, correlated tables, thereby avoiding duplicate information. The normalisation process further addresses dependency anomalies, including insertion, update, and deletion anomalies, ensuring that alterations to the database do not give rise to inconsistencies or errors [37], [38].

The efficiency gains from normalisation are particularly vital when handling substantial datasets, as it optimises storage space by systematically eliminating redundant data. This streamlined structure resulting from normalisation facilitates effective storage utilisation and augments the overall system performance, a critical aspect in tasks such as information extraction where swift and precise data retrieval is paramount.

Moreover, a normalised schema significantly amplifies adaptability to changes in the data structure. This adaptability allows modifications to the data model without causing widespread disruptions, a crucial flexibility when refining the information extraction pipeline [37], [38], [39].

For hosting the normalised schema, MySQL, an open-source *relational database management system* (RDBMS), emerges as a sensible and cost-effective choice. Aligned with academic and research environments due to its open-source nature, MySQL obviates licensing costs and enhances accessibility. The robustness inherent in MySQL ensures the stable and reliable hosting of databases, a critical facet for the seamless implementation of an information extraction system where data integrity and availability are paramount [37], [38].

Considering the abovementioned factors, a normalised ER schema has been deliberately chosen for storing and retrieving data from CDs in this research. This approach facilitates the verification process at both entity and attribute levels and enables a nuanced examination, such as determining the accuracy of extracting specific elements like course titles or learning outcomes. The db4free [40] MySQL platform has been selected to host the RDBMS. This platform, offered by db4free.net, provides a cost-free testing and development server for the latest version of MySQL, making it an ideal choice for educational purposes. Additionally, all artefacts for this research are developed using Python in the Google Colab [41] development environment. Utilising an online database server with these tools streamlines the interfacing process, further enhancing the research's practicality and accessibility.

Figure 4 encapsulates an ER diagram [42] delineating the entities and their interconnections within the CD domain. This ER diagram encompasses six principal entities: “Programme”, “Course”, “Course Pre-requisites”, “Learning Outcome”, “Assessment”, and “Completion”. Each entity is equipped with specific attributes; for instance, the “Completion” entity includes attributes such as “Completion ID” and “Criteria”, with the underlined attribute signifying its unique/primary key – in this instance, the “Completion ID”. Additionally, each entity

establishes relationships with other entities, exemplified by the “Course” and “Completion” entities demonstrating a one-to-many relationship between them.

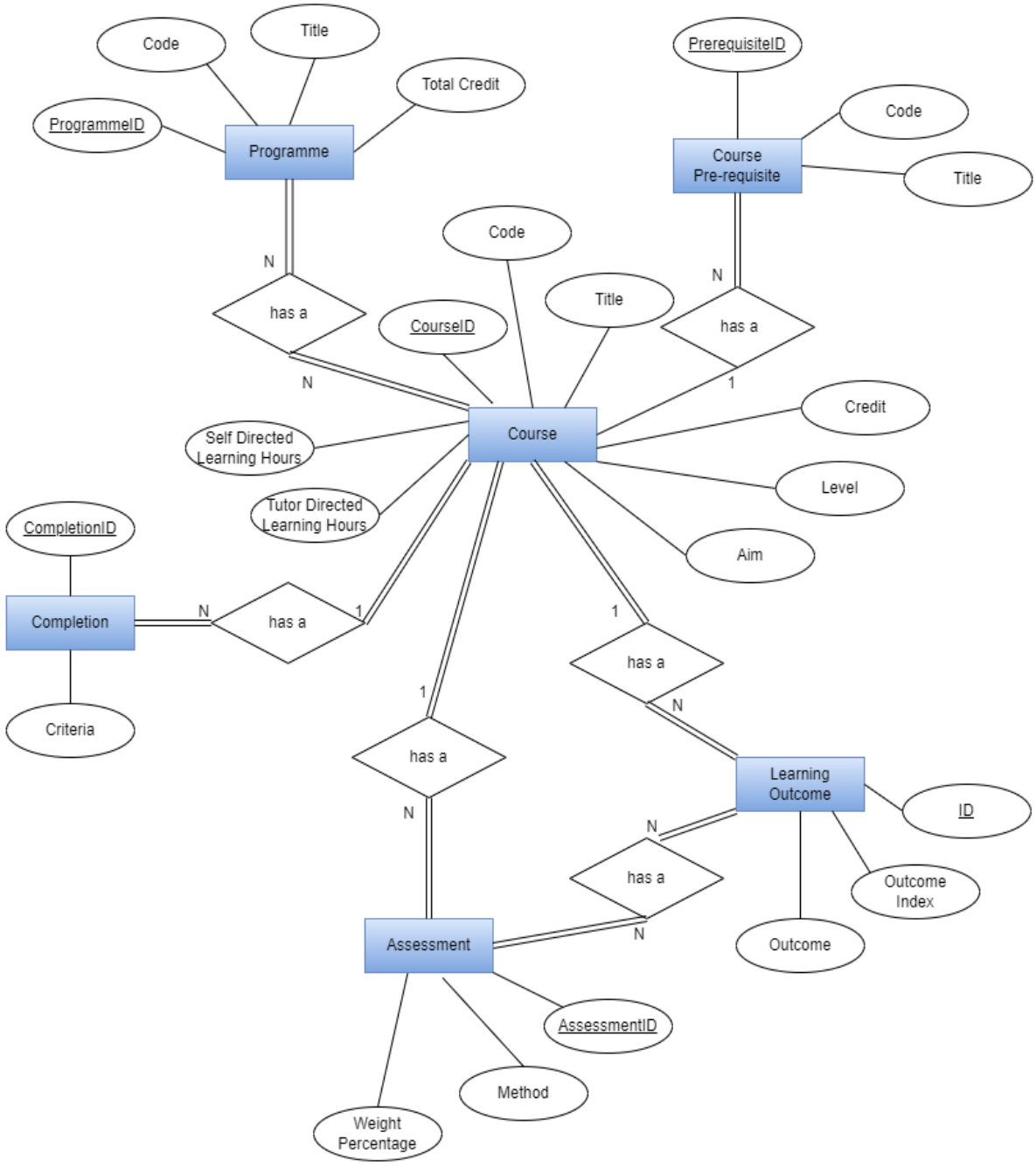


Figure 4: Entity-Relationship Diagram.

This diagram not only provides a comprehensive overview of the data structure for CD but also serves as a foundational resource for the development of a class diagram, database objects in the form of SQL scripts, and interfacing modules. Figure 5 elucidates the SQL script detailing

the implementation of the course and completion entities, thereby contributing to the practical manifestation of the conceptual data model.

```
3
4  CREATE TABLE course (
5     courseID INTEGER(11) NOT NULL AUTO_INCREMENT,
6     code VARCHAR(10) NOT NULL,
7     title VARCHAR(100) NOT NULL,
8     credit INTEGER(11) DEFAULT 0,
9     level INTEGER(11) DEFAULT 0,
10    self_directed_learning_hrs INTEGER(11) DEFAULT 0,
11    tutor_directed_learning_hrs INTEGER(11) DEFAULT 0,
12    aim VARCHAR(4000) DEFAULT NULL,
13    versionID INTEGER(11) NOT NULL,
14    pageNo INTEGER(11) NOT NULL,
15    PRIMARY KEY (courseID),
16    KEY FK_course_version_id (versionID),
17    CONSTRAINT FK_course_version_id FOREIGN KEY (versionID) REFERENCES version (versionID) ON DELETE CASCADE ON UPDATE CASCADE
18 ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
19
20
21 CREATE TABLE completion (
22    completionID INTEGER(11) NOT NULL AUTO_INCREMENT,
23    criteria VARCHAR(2000) NOT NULL,
24    courseID INTEGER(11) NOT NULL,
25    PRIMARY KEY (completionID),
26    KEY FK_completion_course_id (courseID),
27    CONSTRAINT FK_completion_course_id FOREIGN KEY (courseID) REFERENCES course (courseID) ON DELETE CASCADE ON UPDATE CASCADE
28 ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
29
```

Figure 5: SQL Scripts.

In the SQL scripts, a distinctive identifier for each table is established using AUTO_INCREMENT, a feature managed by the MySQL RDBMS. This mechanism automatically increments the reference, such as “CourseID”, for the Course table with each insertion into the table. Mandatory attributes are rigorously constrained with NOT NULL, ensuring that these fields consistently contain values. The PRIMARY KEY designation defines the unique key for the table, while FOREIGN KEY establishes relationships with referencing tables. The close correlation between the ER diagram and the SQL scripts is evident, emphasising that an accurately defined ER model greatly facilitates the creation of tables.

Figure 6 provides a visual representation of the implementation of the persistence layer (**owr_data_model_v1.ipynb**) for the Course. The figure illustrates the functionality of the ‘**inser_course**’ program, which takes all attributes of a course as input parameters. Upon capturing and inserting the data into the table, the program returns the unique Course ID. This Course ID is essential for other programs, enabling the establishment of relationships among various entities and tables.

```python
def insert_course(conn, code, title, credit, level, self_directed_learning_hrs,
                  tutor_directed_learning_hrs, aim, versionID, pageNo):

  # Placeholder for the returned courseID
  course_id = None

  try:

      if conn.is_connected():
          cursor = conn.cursor()

          # SQL query to insert data into the 'course' table
          insert_query = """
              INSERT INTO course (
                  code, title, credit, level, self_directed_learning_hrs,
                  tutor_directed_learning_hrs, aim, versionID, pageNo
              ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
          """

          # Execute the query with the provided data
          cursor.execute(insert_query, (
              code, title, credit, level, self_directed_learning_hrs,
              tutor_directed_learning_hrs, aim, versionID, pageNo
          ))

          # Commit the changes
          conn.commit()

          # Retrieve the last inserted ID
          course_id = cursor.lastrowid

  except Error as e:
      title = "Error while inserting to course table"
      log_error(conn, title, str(e))

  finally:
      # Close the cursor and connection
      if conn.is_connected():
        cursor.close()

  return course_id
```

Figure 6: Persistence layer - Insert Course.

The '**owr_data_model_v1**' package consolidates all operations into MySQL tables designed to host CD information. Consequently, this package becomes integral to the OWR pipeline for persisting the extracted information. These artefacts collectively conclude the Database Model implementation phase in the research, encompassing the storage of CD information based on entities and attributes for GT and the OWR pipeline. Ultimately, these components will be instrumental in verifying the accuracy of the newly developed models.

3.4.2 Ground Truth

Ground truth creation is a pivotal stage in the process of generating test data. Given the decision to utilise CDs as templates for test data, extracting CD information from PDF-format programme documents becomes a critical task. The precise extraction of this information holds paramount significance for the success of the research, as the evaluation of OWR models and pipelines centres on the accuracy of the GT creation process. Thus, during this phase, the primary objective is to develop an artefact capable of extracting CD information from PDF programme documents and creating GT.

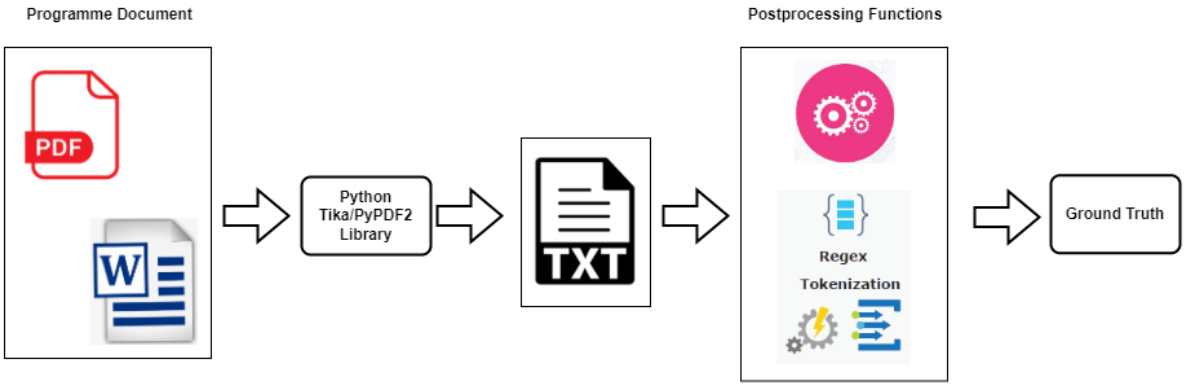
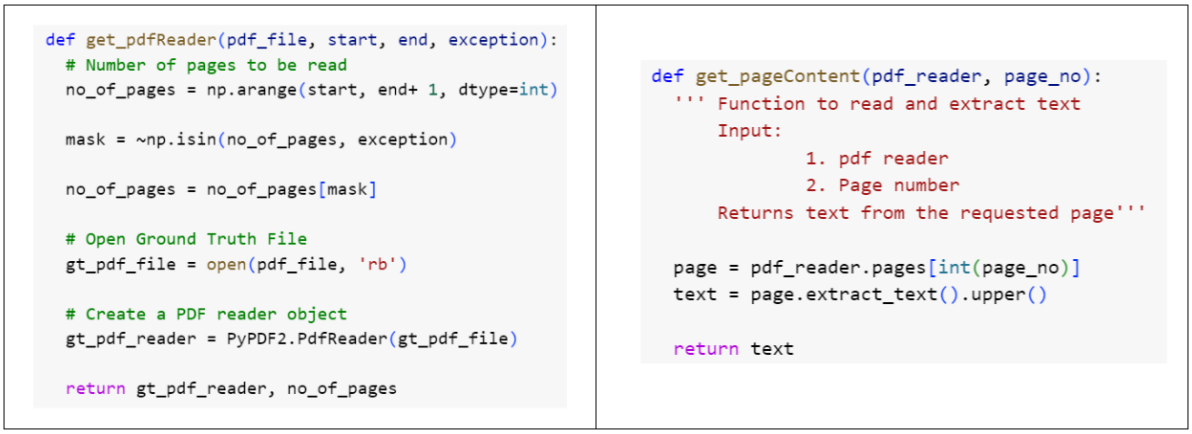


Figure 7: Information Extraction Process for Ground Truth.

Figure **7** offers an overview of the GT creation process, which is destined to become another artefact arising from this research. The initial steps of the GT creation process involve extracting information and text from PDF files. To achieve this, I explored Python libraries such as Tika [43], PyPDF2 [44], and docx2python [45]. Each of these libraries caters to specific document processing needs. Apache Tika distinguishes itself with its versatility, supporting a wide array of file formats and providing a unified interface for content extraction.

On the other hand, PyPDF2 is a specialised Python library designed for efficient PDF manipulation, offering features like merging and splitting PDFs. Its ability to extract pages page by page provides precise control over PDF content, a crucial feature for dealing with extensive documents. In the domain of Microsoft Word documents, docx2python streamlines the extraction of text and metadata from DOCX files.



```python
def get_pdfReader(pdf_file, start, end, exception):
  # Number of pages to be read
  no_of_pages = np.arange(start, end+ 1, dtype=int)

  mask = ~np.isin(no_of_pages, exception)

  no_of_pages = no_of_pages[mask]

  # Open Ground Truth File
  gt_pdf_file = open(pdf_file, 'rb')

  # Create a PDF reader object
  gt_pdf_reader = PyPDF2.PdfReader(gt_pdf_file)

  return gt_pdf_reader, no_of_pages
```

```python
def get_pageContent(pdf_reader, page_no):
  ''' Function to read and extract text
      Input:
              1. pdf reader
              2. Page number
      Returns text from the requested page'''

  page = pdf_reader.pages[int(page_no)]
  text = page.extract_text().upper()

  return text
```

Figure 8: PyPDF2 Implementation.

While both Tika and PyPDF2 are suitable for extracting CD content from PDF programme documents, it is essential to note that program documents encompass more than just CDs. Therefore, prioritising control over selecting pages for text extraction becomes paramount when choosing a library. In consideration of this, the PyPDF2 library was adopted for text extraction. Figure **8** visually represents the functions developed for extracting text from PDF documents. The '**get_pdfReader**' function takes the path to the programme file and the range of pages to be extracted as input, returning a '**pdfReader**' object and page numbers. The output from '**get_pdfReader**' serves as input to the '**get_pageContent**' function, providing the text extraction of the CD page. Figure **9** illustrates the output from the '**get_pageContent**' function, emphasising that a single CD in the program document is confined to one page [44].

While these libraries excel at extracting text from documents, they may lack the context and structure essential for making sense of the extracted text, as evident in the PyPDF2 output visualised in Figure **9**. Therefore, postprocessing after text extraction becomes paramount to gaining meaningful insights. Essential techniques for this purpose include regular expressions, tokenisation, and pattern recognition. Regular expressions, characterised by powerful sequences of characters, facilitate the identification and extraction of specific patterns within text. Tokenisation involves breaking down text into smaller units, such as words or phrases, creating a structured representation for more straightforward analysis [46]. Pattern recognition techniques, ranging from simple keyword matching to advanced machine learning algorithms, enable the identification of complex structures or themes within the text. These methods collectively contribute to extracting relevant information supporting tasks like named entity recognition, sentiment analysis, and information retrieval. The synergy of regular expressions,

tokenisation, and pattern recognition empowers researchers and developers to unlock insights from diverse textual sources, enhancing the capabilities of natural language processing and information extraction systems [47].

```
NEW ZEALAND CERTIFICATE IN INFORMATION TECHNOLOGY ESSENTIALS (LEVEL 4)  – PROGRAMME DOCUMENT  95 SD7501  WEB APPLICATION DEVELOPMENT

LEVEL  7 CREDITS  15
PRE-REQUISITES  IT5507 FUNDAMENTALS OF DATA SCIENCE
 SD6502 PROGRAMMING II
LEARNING HOURS  TUTOR DIRECTED    52 HOURS
 SELF-DIRECTED    98 HOURS
AIMS
TO ENABLE THE LEARNER TO:
• EVALUATE AND APPLY THE USE OF APPROPRIATE PLATFORM AND ARCHITECTURE, FOR THE DEVELOPMENT OF WEB
APPLICATIONS.
• INTEGRATE APPLICATIONS WITH A DATABASE AND LEARN HOW TO ACCESS WEB DATA USING MANAGED DATA PROVIDERS
AND OBJECTS.
• INVESTIGATE THE SECURITY CHALLE NGES AND SECURITY MODELS FOR WEB APPLICATIONS.
LEARNING OUTCOMES
ON SUCCESSFUL COMPLETION OF THIS COURSE, THE LEARNER WILL BE ABLE TO:
1. EVALUATE  THE BUSINESS, TECHNICAL AND SOCIAL IMPLICATIONS OF WEB APPLICATION DEVELOPMENT.
2. ANALYSE  THE BACKGROUND AND UNDERLYING PRINCIPLES OF WEB APPLICATION DEVELOPMENT IN THE SELECTED
FRAMEWORK.
3. DESIGN AND IMPLEMENT AN APPROPRIATE SECURE INTERNET APPLICATION SOLU TION TO AN UNSTRUCTURED PROBLEM.
4. RESEARCH AND CRITICALLY EVALUATE NEW TOOLS AND TECHNOLOGIES IN RELATION TO INTERNET APPLICATION
DEVELOPMENT.
INDICATIVE CONTENT
THE COURSE WILL CONTAIN THE FOLLOWING TOPICS:
ASP.NET FRAMEWORK, DEVELOPMENT TOOLS AND ENVIRONM ENT , MVC FRAMEWORK , MVC ROUTING, TRACING & DEBUGGING , AJAX
& JQUERY , DATA ACCESS AND DATA BINDING , ASP.NET CORE WITH ENTITY FRAMEWORK  , ASP.NET CORE IDENTITY , RESTFUL WEB
SERVICES AND WEB API ’S, WEB APPLICATION SECURITY , WINDOWS PRESENTATION FOUNDATION (WPF) , FRONTEND DEVELOPMENT
PLATFORMS FOR WEB APPLICATION , WEB APPLICATION DEPLOYMENT
ASSESS MENT
ASSESSMENT METHOD  WEIGHT %  LEARNING OUTCOMES
RESEARCH REPORT AND PRESENTATION  25%  1, 2, 4
PROGRAMMING PROJECT  35%  3, 4
FINAL EXAM  40%  1, 2, 3, 4

SUCCESSFUL COMPLETION OF COURSE
TO PASS A COURSE WHERE THERE IS AN EXAMINATION SET, A LEARNER MUST:
• ATTEMPT ALL ASSESSMENTS
• ACHIEVE AN AVERAGE MARK OF 50%  OR ABOVE OVER ALL ASSESSMENTS, INCLUDING THE EXAMINATION
• ACHIEVE A MARK OF 40%  OR ABOVE IN THEIR FINAL EXAMINATION
```

Figure 9: PyPDF2 Output.

All these techniques were deployed to extract GT from the PyPDF2 output. Table 1 provides a high-level overview of the keywords, techniques, and logic employed to extract information for each entity and its attributes. It is noteworthy that, aside from the database schema, the postprocessing package (‘**owr_postprocess_module**’) is the only artefact closely tied to CD elements. Generalising this solution to other similar templates requires modification of CD-specific content with appropriate keywords.

Entity	Attribute	Technique	Start Words	Stop Words	Logic
Course	Course Code	RE / Pattern / Token		LEVEL, CREDITS	Text with 6 characters long, start with 2 letters, end with 2 numeric and occurring before any stop words.
Course	Course Title	Pattern / Token	Value of course code	LEVEL	Text between specified start and stop words.
Course	Level	RE / Pattern / Token	LEVEL	AIM	First numeric value occurs between the start word and the stop word.
Course	Credit	RE / Pattern / Token	CREDITS	AIM	First numeric value occurs between the start word and the stop word.
Course	Tutor Hours	RE / Pattern / Token	LEARNING, TUTOR, TUTOR-DIRECTED, TUTORDIRECTED	AIM, AIMS	First numeric value occurs between the start word and the stop word.
Course	Self Directed Hours	RE / Pattern / Token	LEARNING, SELF, SELF-DIRECTED, SELFDIRECTED	AIM, AIMS	First numeric value occurs between the start word and the stop word.
Course	Aim	Pattern / Token	AIM	LEARNING OUTCOMES	Text occurs between the start word and the stop word.
Course Prerequisite	Code, Title	RE / Pattern / Token	LEARNING, HOURS, TUTOR, TUTOR-DIRECTED, TUTORDIRECTED, DIRECTED, -DIRECTED, PRE-REQUISITES, -REQUISITES, REQUISITES, PREREQUISITE', NONE, Value for Tutor Hours	PRE-REQUISITES, -REQUISITES, REQUISITES, PREREQUISITES, Valuer for Self Hours, CREDITS	Text with 6 characters long starts with 2 letters, ends with 2 numeric and occurs between any start word and any stop words as code. Excluding code, the remaining text from the above condition has been extracted as a title.

Learning Outcome	Index, Outcome	RE / Pattern / Token	LEARNING, OUTCOMES, OUTCO, MES, LEAR, NING	INDICATIVE, CONTENT, CONTE, NT, •, ASSESSMENTS, METHOD, WEIGHTING, WEIGHT, OUTCOME/S, OUTCOMES	Numeric value occurring between the start word and the stop word as index. Excluding index remaining Text from above condition has been extracted as an outcome.
Assessment	Weight	RE / Pattern / Token	ASSESSMENTS , METHOD, WEIGHTING, WEIGHT, LEARNING, OUTCOME/S, OUTCOMES	SUCCESSFUL, COMPLETION, OF, COURSE	First numeric values occur before '%' between the start words and stop words.
Assessment	Method	RE / Pattern / Token	ASSESSMENTS , METHOD, WEIGHTING, WEIGHT, LEARNING, OUTCOME/S, OUTCOMES	SUCCESSFUL, COMPLETION, OF, COURSE	Text before Weight values from the above logic have been extracted as a method.
Assessment	Outcomes	RE / Pattern / Token	ASSESSMENTS , METHOD, WEIGHTING, WEIGHT, LEARNING, OUTCOME/S, OUTCOMES	SUCCESSFUL, COMPLETION, OF, COURSE	Numeric values following weight values are treated as indices mapping to learning outcomes. When encountering a '-', any numbers generated between the numeric values present on either side of the '-' are also considered, for instance, in the case of '1-5'.
Completion	Criteria	Pattern / Token	ASSESSMENTS , METHOD, WEIGHTING, WEIGHT, LEARNING, OUTCOME/S, OUTCOMES, SUCCESSFUL, COMPLETION, OF, COURSE	RESOURCES	Text occurs between the start word and the stop word.

Table 1: Postprocessing Logic.

Figure **10** presents a code snippet illustrating the course code extraction function, implementing the logic described in Table 1. This function operates systematically by loading all words from the entire document into a vector, searching from top to bottom until it encounters the first instances of the words "Level" and "Credits." Subsequently, starting from the positions of "Level" and "Credits," the function performs a recursive backward search toward the first element of the vector. It looks for words with six characters in length, beginning with two alphabetical characters and ending with numeric characters, allowing for the identification of the course code.

```
def get_courseCode(text):
  '''
  This function is designed to extract Course Code
  Input: Page text
  Output: Course Code, Position for Level and Credit
  Logic:  1. Track Level and Credit positions in List
          2. Reverse search for
                a. Text with length 6
                b. Start with 2 characters
                c. End with 2 numeric
  '''

  # extract words
  words = word_tokenize(text)

  # search string
  search_str = ['Level','Credits']
  word_gap = 3

  # find the index of the search string
  search_pos = []
  for i, item in enumerate(words):
    if item.lower() == search_str[0].lower():
    '''
    ...
    '''

  # regular expression patterns for code
  pattern1 = r"\b\w{6}\b" # word with 6 positions
  pattern2 = r"\b^[a-zA-Z]{2}\w+" # start with 2 characters
  pattern3 = r"\w+\d{2}$" # end with 2 numbers

  code = None
  if len(search_pos) > 0:
    for i in range(search_pos[0], -1, -1):
      matches1 = re.findall(pattern1, words[i])
      matches2 = re.findall(pattern2, words[i])
      matches3 = re.findall(pattern3, words[i])

      if len(matches1)> 0 and len(matches2)> 0 and len(matches3)> 0:
        code = words[i]
        break

  return (code, search_pos)
```

Figure 10: Preprocessing Logic for Course Code.

However, complexities may arise due to word distortion (e.g., introducing spaces within words), necessitating further postprocessing specific to the processed document. Consequently, such exceptional cases have been parameterised in the GT model for easy maintainability. Several examples of these exceptional cases are demonstrated in Figure **11**.

```
def get_endPoint4Exceptions(sentence):
  end_text_exception = ['learning outco mes','lear ning outcomes',
                        'identify and explain contemporary','critically analyse ethical issues'
                       ]
```

Figure 11: Exceptions.

3.4.3 Input Data

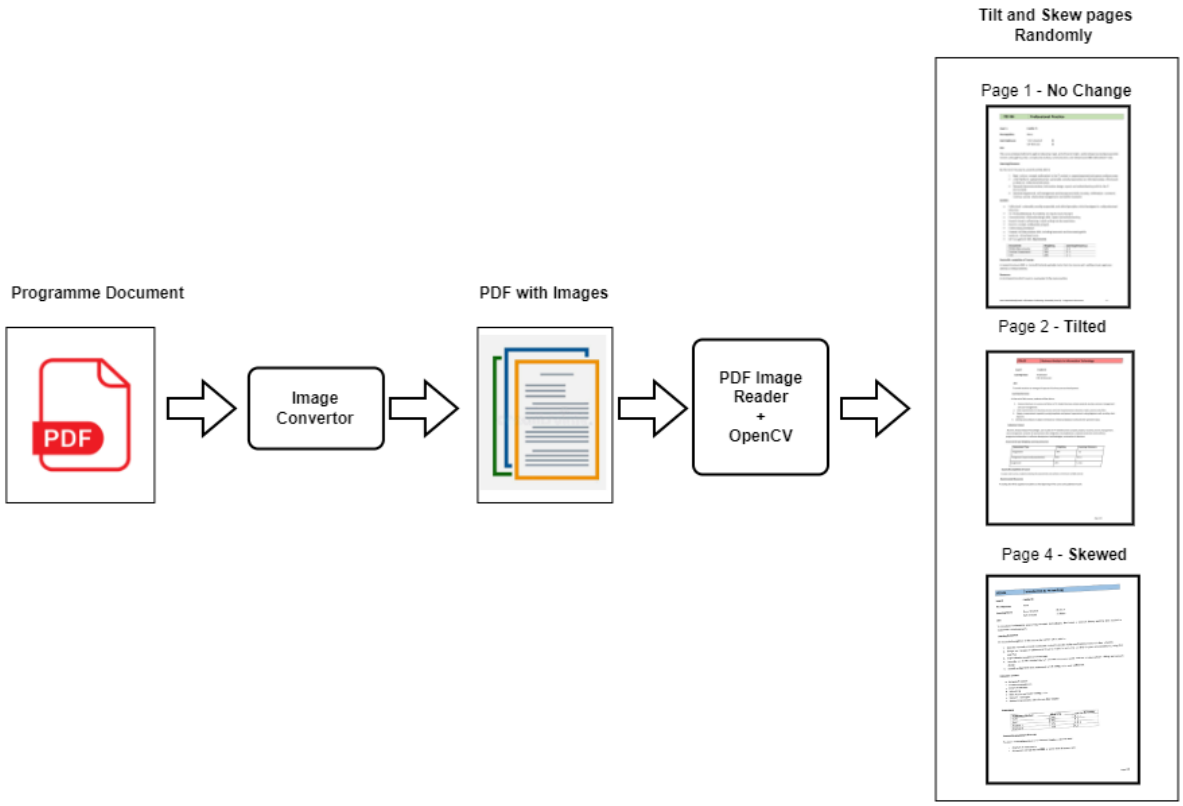


Figure 12: Input Data.

As part of the research to develop an artefact extracting textual information from image-based complex documents, the input data task involves creating test images from CDs sourced from PDF programme documents. A two-step process is implemented to achieve this, as illustrated

in Figure **12**. The PDF file is read as the first step, and its pages are extracted as images. The Python '**Fitz**' library [48], an integral component of the PyMuPDF (MuPDF) package, is a robust tool for PDF manipulation, offering versatile functionalities. Notably, the '**get_pixmap**' function within Python '**Fitz**' facilitates the extraction of pages from PDF files as images. Leveraging MuPDF's lightweight and high-performance PDF rendering engine, Python '**Fitz**' ensures the efficient processing of even large PDF documents. Figure **13** depicts the program's implementation, taking PDF file path, page range for conversion, and image directory path as inputs to convert PDF pages to PNG-formatted image files.

```
def convert_pdf_to_images(pdf_path, image_path, start_page=0, end_page=0 ):
  # Open the PDF file
  pdf_doc = fitz.open(pdf_path)

  # Ensure the start page is valid
  if start_page < 0:
    start_page = 0
  if end_page < 0:
    end_page = len(pdf_doc)

  # Iterate through the pages
  for page_number in range(start_page, end_page):
    # Skip 1 page which is not a CD
    if page_number + 1  in exception_pages:
      continue

    # Get the page
    page = pdf_doc[page_number]

    # Convert the page to an image
    pix = page.get_pixmap(matrix=fitz.Matrix(2.0, 2.0))

    # Create a Pillow image from the PyMuPDF pixmap
    img = Image.frombytes("RGB", [pix.width, pix.height], pix.samples)

    # Save the image to a file
    img.save(f'{image_path}/page_{page_number + 1}.png', 'PNG')
```

Figure 13: PDF to Image.

In the real world, image-based text files often exhibit skewness or tilt introduced during the file creation, such as scanning or capturing with mobile phones. Random skewness and tilt are introduced into image files to simulate such distorted images. For this purpose, the '**cv2.getRotationMatrix2D**' and '**cv2.warpAffine**' functions in the OpenCV library [49] are used to introduce skewness into image files through rotation operations. The '**cv2.getRotationMatrix2D**' function calculates the 2D affine transformation matrix, defining a rotation around a specified centre point and an angle of rotation. Key parameters include the rotation centre, rotation angle, and scaling factor. The obtained transformation matrix is then applied to the image using the '**cv2.warpAffine**' function, effectively introducing rotation, scaling, and translation. Adjusting the rotation angle in '**getRotationMatrix2D**' enables the

controlled introduction of skewness to the image. Figure **14** illustrates the function built to introduce skewness, with a similar function deployed to introduce tilt based on the tilt angle [50], [51].

```python
def rotate_images_with_skewness(img, angle):

    # Perform the rotation
  rows, cols, _ = img.shape
  #print('rows: ',rows, 'cols: ',cols)

  new_width = int(abs(cols * math.cos(math.radians(angle))) + abs(rows * math.sin(math.radians(angle))))
  new_height = int(abs(cols * math.sin(math.radians(angle))) + abs(rows * math.cos(math.radians(angle))))
  #print('new width:',new_width,'new height:',new_height)

  rotation_matrix = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)
  #print('rotation matrix: ',rotation_matrix)


  # Adjust the translation part of the matrix to ensure the image fits within the new dimensions
  rotation_matrix[0, 2] += (new_width - cols) / 2
  rotation_matrix[1, 2] += (new_height - rows) / 2
  #print('new rotation matrix: ',rotation_matrix)

  rotated_image = cv2.warpAffine(img, rotation_matrix, (new_width, new_height), borderMode=cv2.BORDER_CONSTANT, borderValue=(255, 255, 255))

  return rotated_image
```

Figure 14: Skew Function.

Two main sets of image libraries are generated as part of the input data creation. The first set includes all pages from the PDF file without skewing or tilting. The second set involves randomly skewing and tilting some of the images based on a percentage specified by the user. Figure **15** showcases the user interface for the input creation, introducing 45% skewness, 45% tilt, and 10% without distortions. Furthermore, the tool outputs a CSV file into the same image directory, containing image names, indicators of skewness or tilt, and the respective angles. This information serves as labels (ground truth) for this specific dataset.

```
Enter percentage for Normal Images: 10
Enter percentage for Skewed Images: 45
Enter 1 for 'input' folder, 2 for 'tmp_skewed' folder or 3 for 'tmp_tilt' folder: 1

 Percentages selected are Normal: 10.0, Skewed: 45.0 and Tilted: 45.0
Location of images: /content/gdrive/MyDrive/OWR/data/input
Image /content/gdrive/MyDrive/OWR/data/tmp_input/page_120.png not found. Skipping.
Input Data Created
```

Figure 15: User Interface - Input Data.

3.4.4 OWR Pipeline for Text Detection and Recognition

The OWR pipeline incorporates multiple processors dedicated to detecting and recognising text, as Figure **16** illustrates the activity flow. The initial step involves rectifying skewness and tilt in the text image, called the preprocessing stage, followed by a segmentation process. This

segmentation is multi-faceted, encompassing various levels. The first level of segmentation distinguishes between tabular and non-tabular information, while the second level identifies lines containing text. The final level of segmentation focuses on identifying individual words within those lines. Subsequently, the recognised words are forwarded to the OWR model for recognition. The recognised words undergo postprocessing using a package developed in the GT module. This postprocessing step serves to discern information pertaining to entities and attributes. Ultimately, these identified values are persistently stored in the database schema. The upcoming sections will delve into more granular details concerning each of these critical areas.

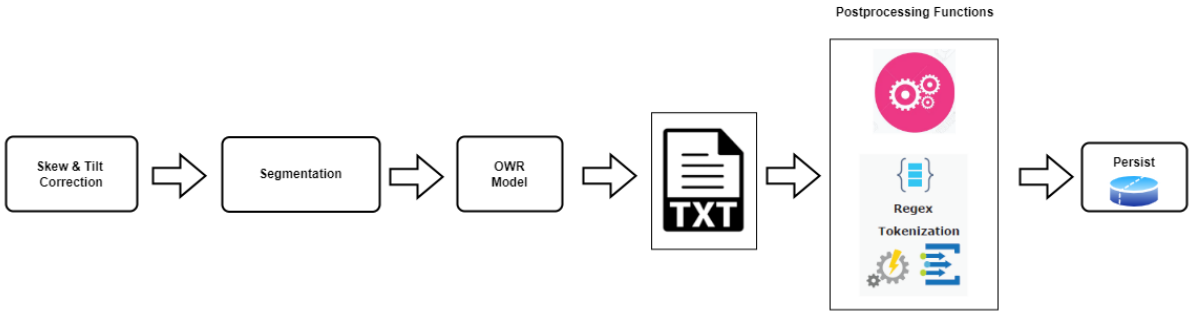


Figure 16: OWR Pipeline.

3.4.5 Preprocessing

The preprocessing and segmentation phases are paramount in the context of OWR and OCR systems. The accuracy of the OWR or OCR system hinges on the execution of these two phases. Preprocessing, in particular, plays a pivotal role in facilitating the distinction between characters/words and their background. Several crucial preprocessing techniques are employed to achieve this, including Binarization, Noise Removal, Thinning (Skeletonization), and Skew or Tilt Correction.

Binarisation, a fundamental preprocessing technique, involves the transformation of a colour image into an image composed solely of black and white pixels (where Black pixel value = 0 and White pixel value = 255). Traditionally, this is accomplished by setting a threshold (typically at 127, as it is the midpoint of the pixel range 0-255). If the pixel value exceeds the threshold, it is deemed a white pixel; otherwise, it is considered a black pixel. However, this approach might not consistently yield the desired results due to variations in lighting conditions. As such, the critical challenge in binarisation lies in determining an appropriate

threshold that adapts to the prevailing lighting conditions. *Otsu's Binarization* [52] is a technique designed to address this challenge. It calculates a threshold for the entire image, taking into account various image characteristics, such as lighting conditions, contrast, and sharpness. This calculated threshold is then employed to binarise the image. Figure 17 illustrates the implementation of Otsu's Binarization using OpenCV libraries.

```
# Binarization with Otsu's
def set_Otsu_Binarization(img):
  # Apply Otsu's Binarization
  _, otsu_binary = cv2.threshold(img, 0, 255,
                                 cv2.THRESH_BINARY + cv2.THRESH_OTSU)

  # Invert the binary image
  img = 255 - otsu_binary

  return img
```

Figure 17: Otsu's Binarisation OpenCV Implementation.

Another effective solution for coping with dynamic lighting conditions is *Adaptive Thresholding* [53], [54]. Unlike global thresholding techniques, which assign a single threshold for the entire image, adaptive thresholding assigns specific thresholds for localised regions based on the characteristics of their neighbouring pixels. This approach results in a smoother transition and greater adaptability. The implementation of adaptive thresholding is illustrated in Figure 18. Adaptive thresholding has been deployed in this research due to its enhanced flexibility and superior results. It considers the locality of image regions for precise binarisation, which is a valuable advantage.

It's important to note that many of the documents referred to in the research were created initially with black text on a white background. However, in the electronic realm, black is often represented as 0, while white is represented as 255. Given the interest in text rather than the background and the fact that many algorithms are designed to work with numerical values, the documents colours were inverted. This inversion renders the text in white and the background in black, enabling algorithms to process the text effectively. The visualisation of the adaptive threshold application of the CD document is shown in Figure 19.

```
# Binarization with Adaptive
def set_Adaptive_Binarization(img):
  # Apply Adaptive Thresholding
  adaptive_binary = cv2.adaptiveThreshold(img, 255,
                                          cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                          cv2.THRESH_BINARY, 11, 2)

  # Invert the binary image
  img = 255 - adaptive_binary

  return img
```

Figure 18: Adaptive Threshold for Binarisation.

Noise removal emerges as a vital stage to enhance image quality by eliminating small, high-intensity dots or patches that may disrupt the overall visual clarity. Noise removal is a versatile process applicable to both coloured and binary images. To achieve this, a range of techniques can be employed, including functions such as '**fastNLMeansDenoisingColored**' and the utilisation of **Gaussian kernels**. For instance, the '**fastNLMeansdenoisingcolored**' function effectively reduces noise in coloured images, enhancing their overall visual fidelity. Moreover, the Gaussian kernel, a well-established tool, plays a significant role in noise reduction, contributing to the refinement of image quality by smoothing out unwanted irregularities [55].

In the context of handwritten text recognition, another crucial technique is ***skeletonisation***. Recognising that various writers possess unique writing styles, including different stroke widths, achieving uniformity in stroke width is essential for accurate recognition. Thinning and skeletonisation are integral steps in this process. They work to standardise the width of strokes, making handwritten text recognition more consistent and precise. However, it is worth noting that noise or handwritten text is notably absent in the selected dataset created from the Bachelor of Information Technology degree program. Therefore, the deployment of these specific preprocessing techniques is unnecessary in the case of these CDs, which predominantly contain typewritten text [56].

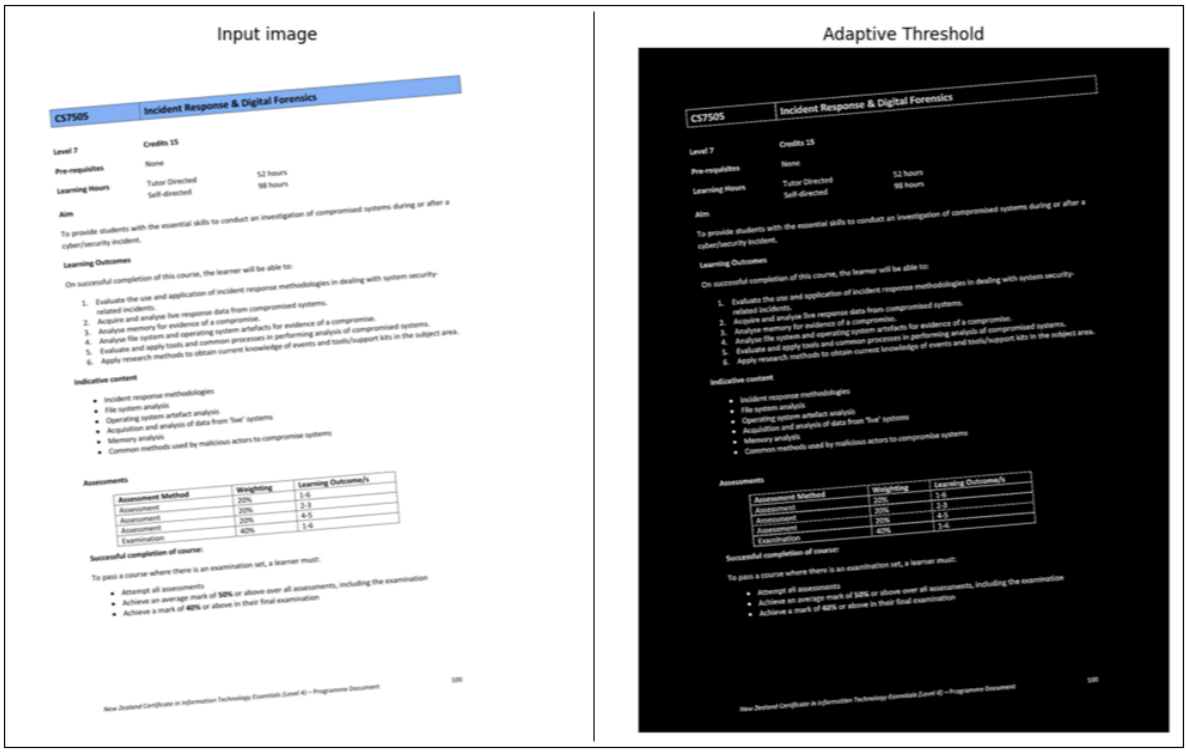


Figure 19: Adaptive Threshold on CD.

3.4.6 Skew and Tilt Correction

Skew and tilt correction are crucial steps in document processing, primarily when dealing with scanned documents. Scanning devices often introduce slight skewness or tilt, which are common issues encountered in scanned materials. As outlined in section 3.4.3, which discusses input data, input dataset constructed intentionally introduces random skewing and tilting into documents. This dataset aims to mimic real-world scenarios and addresses the need for detecting and correcting skewness and tilting, particularly concerning text extraction. The following solutions were deployed to address the skewness and tilt distortions:

1. ***Computer Vision with Hough Transformation:***

   This approach leverages computer vision techniques and Hough transformation to detect and correct skew and tilt in scanned documents.

2. ***Multilayer Perceptron (MLP) Model with Fast Fourier Transformation:***

   The MLP, a deep learning model, is employed as a supervised learning model. It must be trained on labelled data to make predictions based on extracted features. Fast Fourier Transformation processes the data in this case and facilitates skew and tilt correction.

3. ***CNN Model with Fast Fourier Transformation:***

   Like the MLP model, the CNN model is a deep learning approach known for its effectiveness in image-related tasks. Fast Fourier Transformation aids in extracting relevant features, enabling the correction of skew and tilt.

Both MLP and CNN are supervised learning models, meaning they require training on a dataset with labelled ground truth. In the case of skew and tilt correction, a dataset of images with known skewness and tilt angles is indispensable for training these models effectively. However, the input dataset generated from CDs may not suffice for this purpose. Firstly, the volume of such data is limited. Secondly, using the CD data for testing and training can lead to overfitting, where models learn to fit closely to the parameters of the training data but may perform poorly on unseen data.

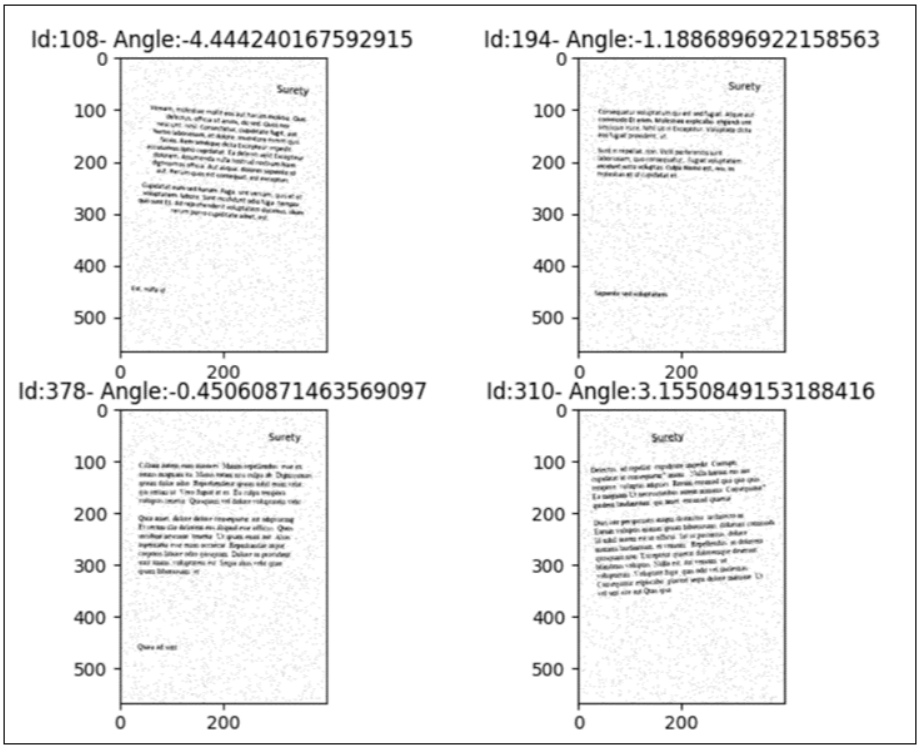


Figure 20: Noisy and Rotated Scanned Documents - Skewed.

To address this challenge, I have opted for the "Noisy and Rotated Scanned Document" dataset available on Kaggle [57] to train the models tailored for skewness correction. This dataset provides a rich diversity of document images and includes the crucial ground truth labelling necessary for robust model training. By incorporating this external dataset, intend to ensure that DL models are comprehensively prepared to tackle a broad spectrum of skew and tilt

scenarios, thereby enhancing their capacity for precise and reliable skew and tilt correction when applied to real-world tasks.

The “Noisy and Rotated Scanned Document” dataset encompasses 500 scanned images, and each is deliberately skewed at a known angle of skewness (ground truth). The range of skewed angles spans from -5° to 5° relative to the horizontal axis. A selection of sample images from this dataset is visually presented in Figure **20**, offering a glimpse into the dataset's content.

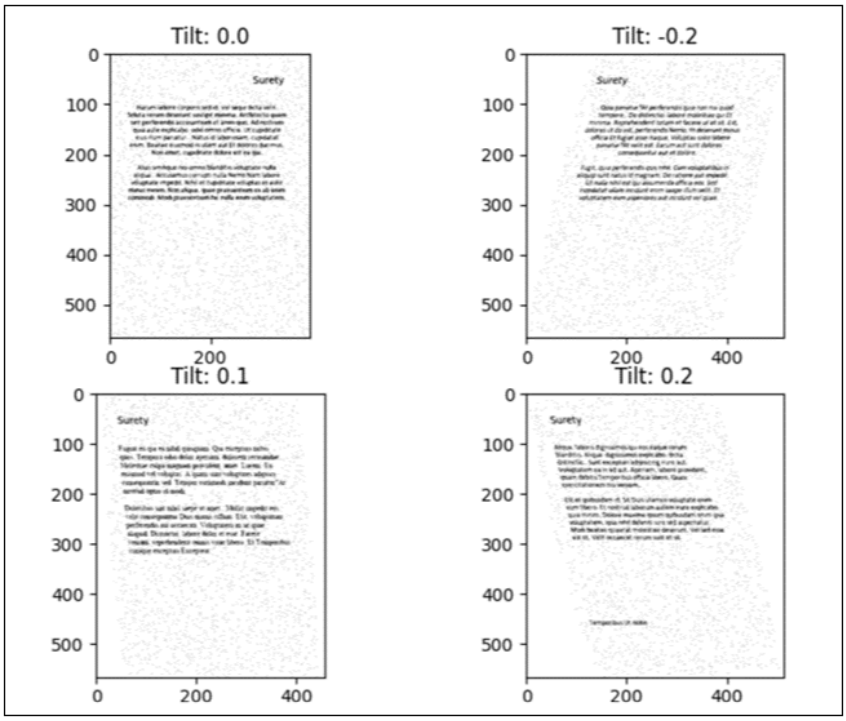


Figure 21: Noisy and Rotated Scanned Documents - Tilted.

Furthermore, an additional set of images was explicitly developed and designed for tilt correction. In this process, the ground truth information from the skewed dataset was leveraged to correct the skewness while also introducing random tilts within a range of angles (-0.2°, -0.1°, 0°, 0.1°, 0.2°). A representation of these images, reflecting the newly introduced tilts, is provided in Figure 21. These datasets played a pivotal role in training the aforementioned models, ensuring their proficiency in handling skewness and tilting challenges effectively.

3.4.7 Computer Vision with Hough Transformation

The model was exclusively designed employing computer vision (OpenCV) libraries for the purpose of angle detection in skewed and tilted documents and subsequent correction. The

overarching process adopted for both scenarios shares common high-level steps. It commences by reading the image in grayscale to simplify subsequent processing. Subsequently, a Canny edge detector is applied to identify the edges delineating text and tables within the document. Finally, the Hough transformation detects the lines connecting most of the edges. The angle associated with the most detected lines is selected as the basis for skewness correction. Figure **22** visualises the line detection process deploying Hough transformation [58], [59], [60].

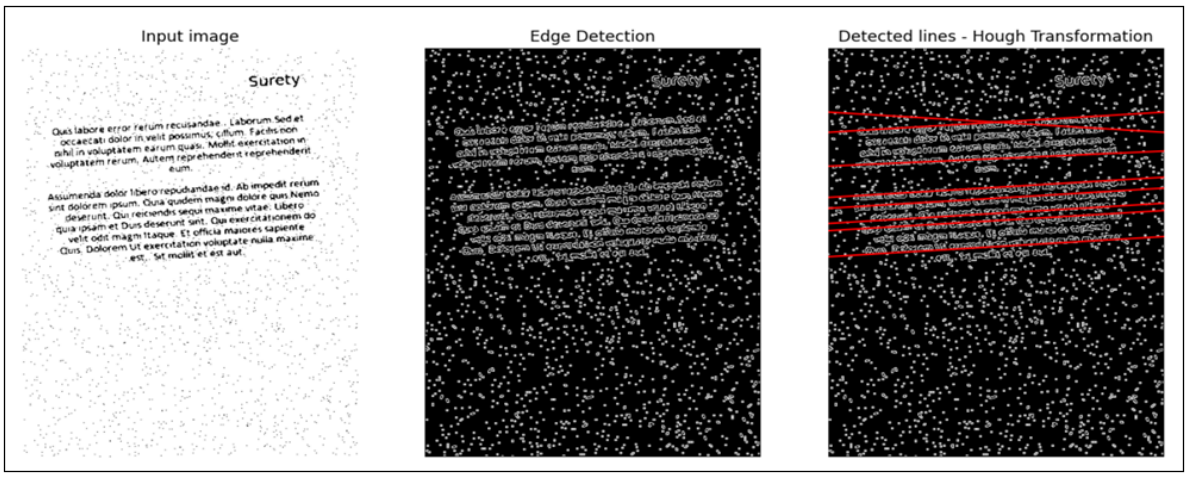


Figure 22: OpenCV - Skewness Correction.

The Canny edge is pivotal in identifying edges within images, which are essential for various computer vision tasks. The algorithm applies gradient-based edge detection techniques, emphasising regions of rapid intensity change. It operates in multiple steps: smoothing the image to reduce noise, computing gradients to determine edge strength and direction, suppressing weak edges, and finally, applying edge tracking by hysteresis to connect adjacent edge points. The Canny edge detector effectively identifies the boundaries of objects in images, making it a crucial tool for object recognition, image segmentation, and feature extraction [61].

On the other hand, the Hough transformation, specifically the Hough line transform, is employed to detect straight lines within images. This technique was initially developed for the purpose of identifying lines in images despite irregularities, noise, or gaps. The Hough line transform converts each point in the image into a parameter space, where lines are represented by their polar coordinates (angle and distance). By assessing the accumulation of intersecting points in this parameter space, the algorithm can determine the presence and characteristics of lines in the original image. OpenCV's Hough line transform function, '**HoughLines**', is a powerful tool for applications such as lane detection in autonomous vehicles, shape

recognition, and image analysis. Additionally, the '**HoughLinesP**' function extends this approach to identify line segments, making it a versatile tool for a wide range of computer vision applications that rely on line and edge detection. Figure **23** illustrates the implementation of Canny edge detection and Hough transformation in the skewness detection function [59], [60].

```
def cd_Hough_Transform(img, angle_range, offset, step):
  # Use canny edge detector
  if(len(img.shape) != 2):
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
  edges = canny(img)

  # Classic straight-line Hough transform
  tested_angles = np.deg2rad(np.arange( 90.0 + angle_range[0] - offset , 90.0 + angle_range[1] + offset, step))
  h, theta, d = hough_line(edges, theta=tested_angles)

  _, angles,_ =hough_line_peaks(h, theta, d)

  angle = np.rad2deg(mode(angles)[0])
  #Choosing angle using vote
  fixed_angle = -(90.0 - angle)

  return angle,fixed_angle
```

Figure 23: OpenCV Implementation.

Much like the approach used for the skewed dataset, the same procedural steps are applied to determine the tilt angles within the tilted dataset. Figure **24** elucidates the process and offers a visual representation of the tilt correction procedure to provide a visual depiction of these steps and the outcome. However, it has come to my attention that while the Hough transformation effectively identifies horizontal lines, it faces challenges detecting vertical lines within the provided dataset.

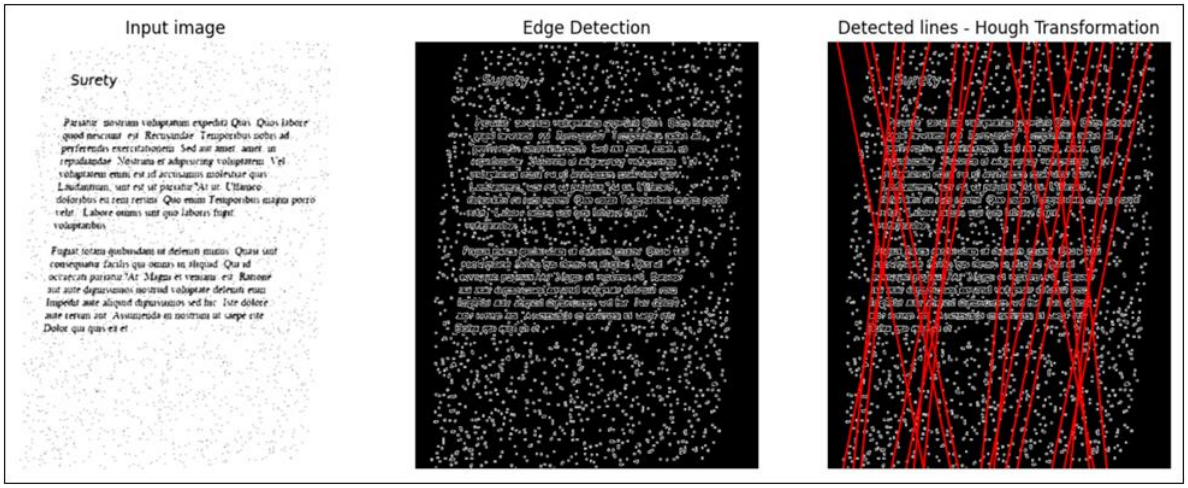


Figure 24: OpenCV - Tilt Correction - Option1.

A decision has been made to introduce a third dataset specifically tailored for tilt correction to resolve the problem of not detecting vertical lines. This new dataset systematically incorporates a straightforward table into all the documents. This approach draws inspiration from the CD document, which features tables designed for capturing assessment data. For a glimpse into this novel dataset, Figure **25** showcases a sample representation of the tilted data enriched with tables.

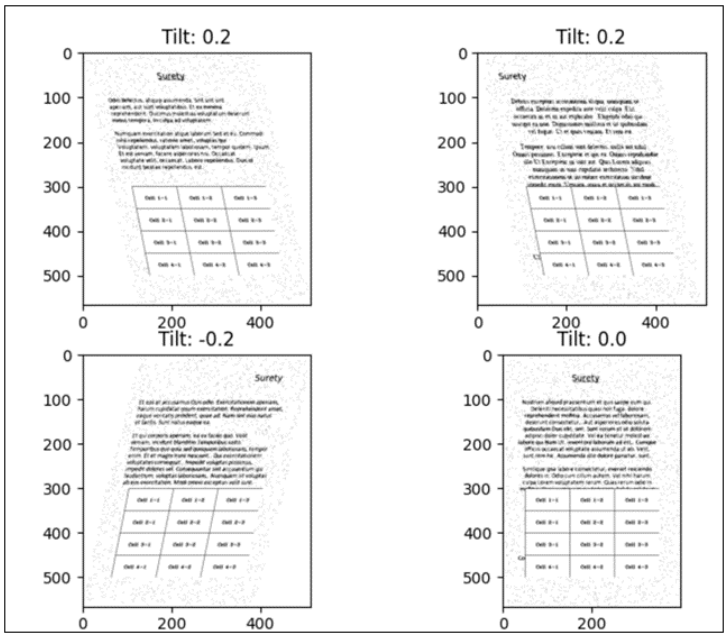


Figure 25: Tilted Documents with Table.

The output in Figure **25** was generated by establishing a table within the document at predefined coordinates. Figure **26** provides a visual representation of the code segment employed to position a table within the document.

```
# Create a blank image with a white background
table = np.ones((table_height_pixels, table_width_pixels, 3), dtype=np.uint8) * 255  # White background

# Draw lines for the table
for ii in range(1, table_height):
    y = ii * cell_height
    cv2.line(table, (0, y), (table_width_pixels, y), (0, 0, 0), border_thickness)  # Black horizontal lines

for j in range(1, table_width):
    x = j * cell_width
    cv2.line(table, (x, 0), (x, table_height_pixels), (0, 0, 0), border_thickness)  # Black vertical lines

# Draw a black rectangle around the entire table (border)
cv2.rectangle(table, (0, 0), (table_width_pixels, table_height_pixels), (0, 0, 0), border_thickness)
```

Figure 26: Draw Table.

As anticipated, the introduction of tables significantly enhances the ability to detect vertical lines using the Hough transformation, as demonstrated in Figure **27**.

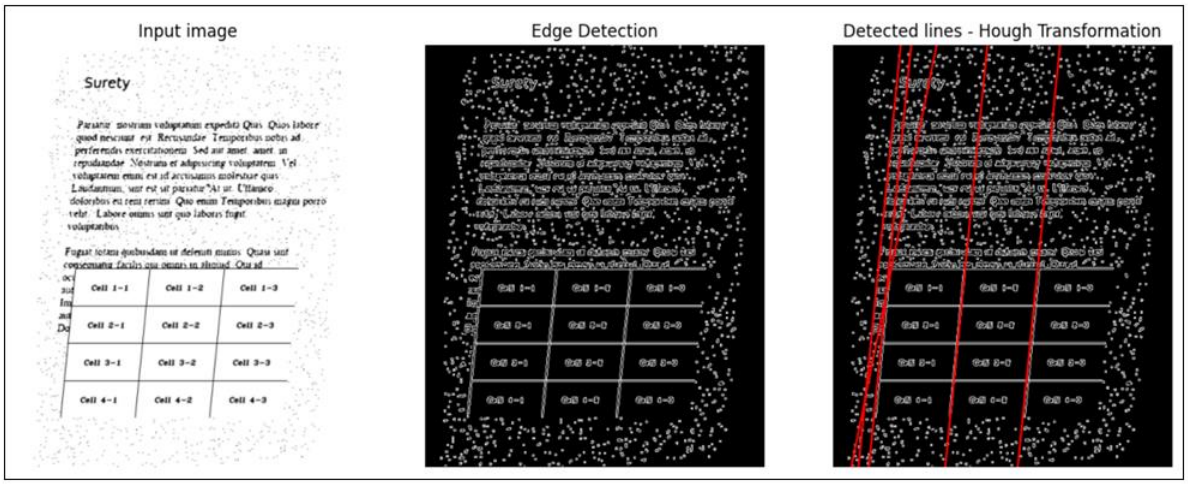


Figure 27: OpenCV - Tilt Correction - Option2.

3.4.8 Fast Fourier Transformation

MLP and CNN models rely on numerical representations, leveraging matrices to extract features and predictors, which are then used to train their neural networks for predicting desired outcomes. In this scenario, the input matrix constitutes the image; its pixel values are treated as numerical attributes. However, when dealing with text documents, each document inherently differs in content, varying the number of paragraphs and exhibiting diverse text styles. Extracting features from these inherently heterogeneous documents to map skewness and tilting is similar to comparing apples to oranges. The mere textual content and its representation do not inherently provide a substantial advantage for predictive modelling. Moreover, the exact skew or tilt angle may apply to documents of different types, and attempting to learn features from individual documents to predict a common angle is not a meaningful endeavour.

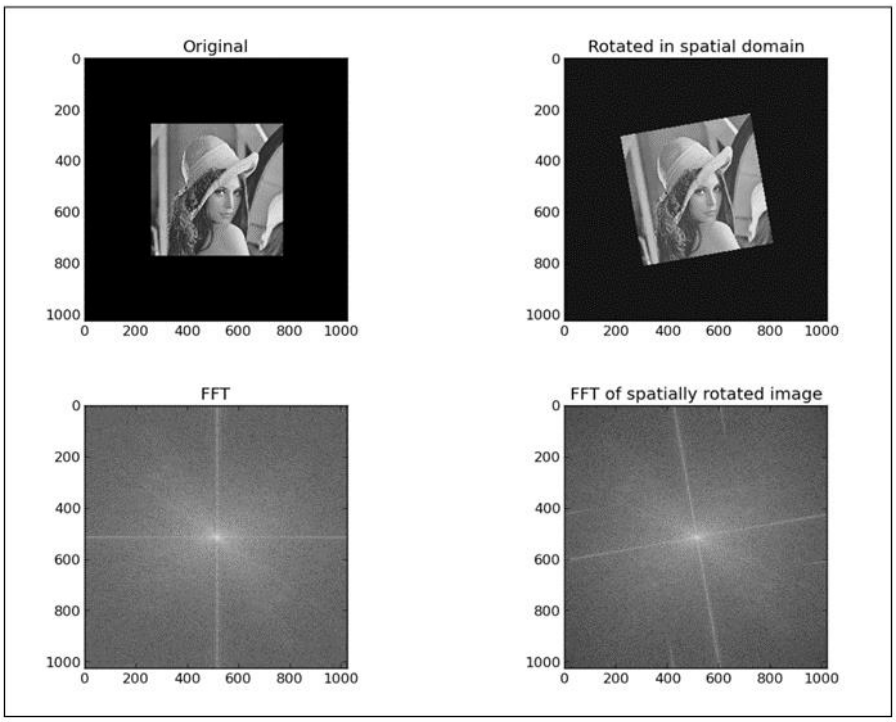


Figure 28: Fast Fourier Transforms.

To solve this intricate challenge, the images were preprocessed into a more standardised format that exclusively conveys their rotational angle. A noteworthy credit is owed to a contributor on Quora [62] who suggested an ingenious approach, which entails performing a *Fast Fourier Transform* (FFT) on the scanned images as a preliminary step. This method, initially employed for denoising images and extracting frequency or amplitude information, is adept at unveiling the angular orientation of the scanned document. This transformative approach not only simplifies the feature extraction process but also ensures that the prediction of skewness and tilting is grounded in a more uniform and reliable representation of the documents, transcending the inherent diversity in textual content and style.

The FFT is a cornerstone of image analysis, extensively utilised to unveil the underlying frequency components within images. Its primary function is to convert an image from the spatial domain into the frequency domain. In this domain, an image is expressed as a superposition of sinusoidal components with varying magnitudes and phases. FFT enables the separation of these constituent frequencies, providing valuable insights into the image's texture, patterns, and structural elements. Figure 28 visually represents the FFT applied to a rotated image. The FFT transformation effectively provides discernible features that a neural network can utilise. Significantly, this approach simplifies the intricacies associated with

varying document layouts and contents, facilitating the network's ability to process and interpret these diverse documents with a unified and robust approach [63], [64], [65].

Figure **29** shows the implementation of FFT as a crucial preprocessing step for images. This transformation is integral for gaining insights into the frequency domain characteristics of the images by employing the '**denoise_tv_chambolle**' function from the '**skimage.restoration**' module to refine this process. This function plays a pivotal role in reducing noise within the images, enhancing the overall clarity of the FFT output. By setting the weight parameter to 1.0 and configuring the multichannel to 0, the program achieved a denoised FFT representation that aids in the subsequent processing stages.

```
cd_img_path = tmp_skewed_path +'/'+cd_names[18]
cd_img = cv2.imread(cd_img_path)
cd_img = cd_img[:,:,0] #zeroth component is the red from RGB channel ordering

cd_img = denoise_tv_chambolle(cd_img, weight=1.0, multichannel=0)

f = cv2.dft(np.float32(cd_img))
fshift = np.fft.fftshift(f)
f_abs = np.abs(fshift) + 1.0
f_img = 20 * np.log(f_abs)
```

Figure 29: FFT Implementation.

Additionally, the preprocessing involves the utilization of '**cv2.dft**' and '**np.fft.fftshift**' functions. The '**cv2.dft**' function [66] is crucial for applying the *Discrete Fourier Transform* (DFT) to the images, facilitating the analysis of frequency components. Meanwhile, '**np.fft.fftshift**' helps rearrange the FFT output quadrants to ensure that the low-frequency components are positioned at the centre. Together, these functions contribute to a comprehensive preprocessing strategy, enabling a more effective and accurate image data analysis within the frequency domain.

3.4.9 MLP Model with Fast Fourier Transformation

The first DL model deployed to detect skewness and tilting is MLP. MLP model is a type of artificial neural network used for supervised learning tasks, such as classification. It comprises multiple layers of interconnected nodes or neurons, organised into an input layer, one or more hidden layers, and an output layer. The model employs a feedforward process to learn features, where it takes input data and propagates it through the network, passing through each layer and applying weights and activation functions. The model extracts and transforms the raw input

features into higher-level representations learned through iterative training using labelled data. These learned features are then used in the output layer to make classification decisions, where the model assigns class labels to the input data based on the patterns it has identified in the features, aiming to minimise the difference between its predictions and the true class labels in the training data.

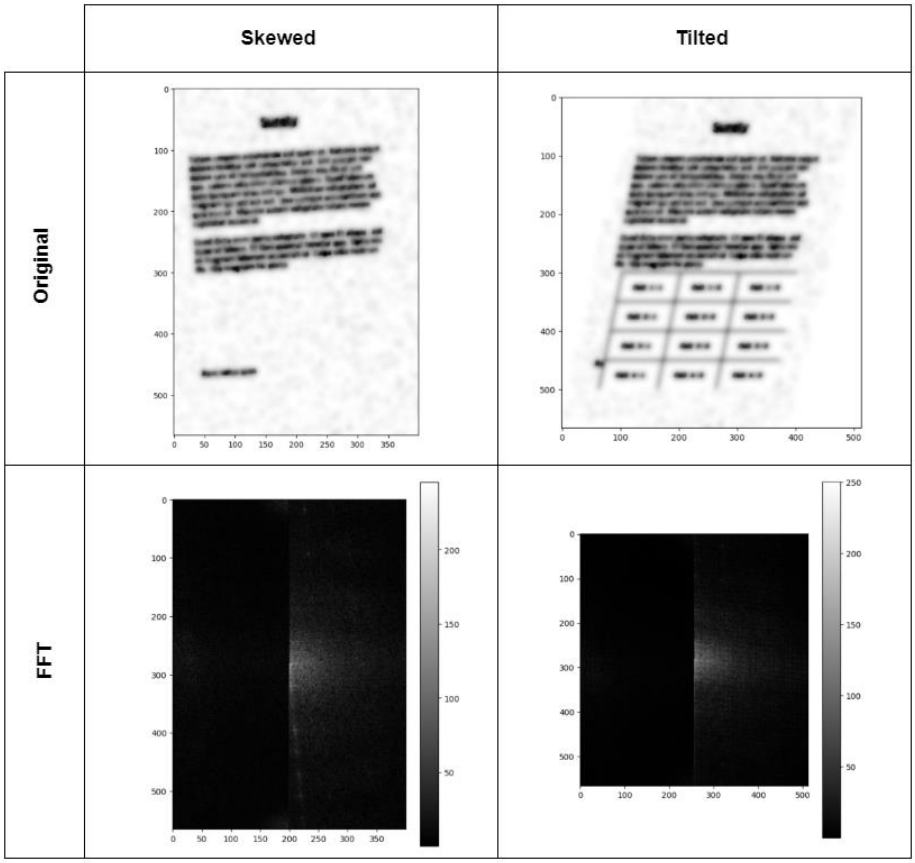


Figure 30: Fast Fourier Transforms on Training Dataset.

In the training of the MLP model, two distinct datasets were employed: the "Noisy and Rotated Scanned Document" dataset served as the foundation for the skew correction model, while a specially crafted dataset of tilted images was developed to train the tilt correction model. These datasets come with corresponding labels, providing an ideal setting for supervised learning methodologies. As elaborated in Section 3.4.8, all images underwent preprocessing through FFT. Figure **30** visually captures a comparative analysis, displaying the FFT application on both skewed and tilted images side by side. The FFT of the skewed image distinctly reveals an angle in the vertical lines aligned with the skew angle inherent in the image. Conversely, the FFT of the tilted image illustrates uniform angles across all vertical lines, reflecting the tilt of

the text. Notably, these angled vertical lines on tilted images persist regardless of the presence or absence of a table component.

```
# Create Train, Val and Test dataset with 60%, 20% and 20% of labeled data
X_train, X_val, Y_train, Y_val = train_test_split(X, Y, test_size=0.4, random_state=42)
X_val, X_test,  Y_val, Y_test = train_test_split(X_val, Y_val, test_size=0.5, random_state=42)
```

Figure 31: Train, Val and Test Split.

Each dataset comprises 500 images, and a strategic division was implemented for model training, validation, and testing. 60% of the data was allocated for training, 20% for validation, and the remaining 20% for testing, adhering to a stratified split. Figure **31** provides a visualization of the implementation of the train-validation-test split, utilizing the '**train_test_split**' function from the '**sklearn**' library [67].

```
# Model with Multilayer Perceptron (MLP)

mlp_model = Sequential()

mlp_model.add(Dense(512, activation='relu',
                    input_shape=(pixel_count,)))

mlp_model.add(Dense(512, activation='relu'))

mlp_model.add(Dense(512, activation='relu'))

mlp_model.add(Dense(512, activation='relu'))
mlp_model.add(Dropout(0.2))

mlp_model.add(Dense(512, activation='relu'))
mlp_model.add(Dropout(0.2))

mlp_model.add(Dense(n_classes, activation = 'softmax'))

mlp_model.compile(optimizer = 'adam',
                  loss = 'categorical_crossentropy',
                  metrics = ['accuracy'] )
```

Figure 32: MLP Model Implementation.

Figure **32** illustrates the implementation of the MLP model. The input image, with dimensions of 566 in height and 400 in width, is effectively converted into a vector of 226,400 (566 x 400) data points and then fed into '**input_shape**' in the MLP model. The MLP model is constructed with five dense layers, each comprising 512 nodes, and ReLU serves as the activation function for each layer. For the skewed dataset, the output layer is designed with eleven classes aimed at predicting the rotated angle from the classes [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]. The activation function used in the output layer is SoftMax. For the tilted dataset, the model architecture remains the same except for the output layer, which is adapted to feature five classes. These

classes target the prediction of tilt within the range [-0.2, -0.1, 0, 0.1, 0.2]. Both models employ the "Adam" optimiser to refine parameters and "categorical_crossentropy" as the loss function and are tailored to optimise their parameters based on accuracy. Figure **33** visually represents the MLP model architecture for skewed data, highlighting the neural network's structural configuration for these specific tasks.

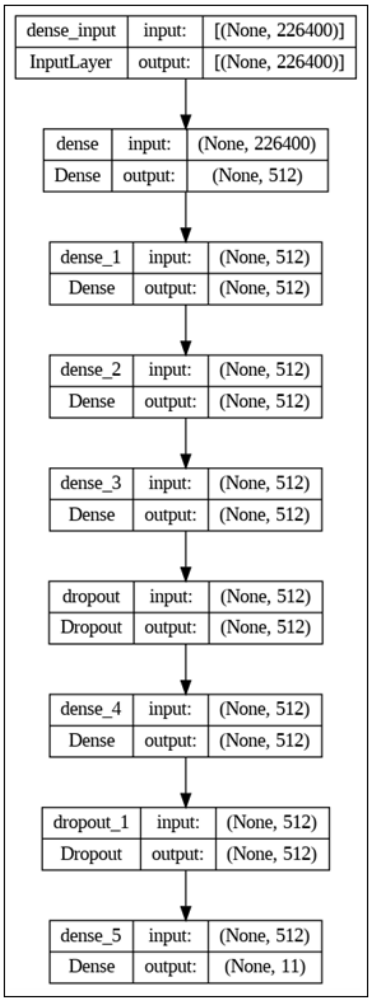


Figure 33: - MLP Architecture.

3.4.10 CNN Model with Fast Fourier Transformation

The primary distinction between MLP and CNN lies in their approach to feature extraction and their suitability for image-processing tasks. MLP models learn directly from the input features and then train their network for predictions. On the other hand, CNN takes a fundamentally different path. It applies convolution operations on the input image, extracting features through filters or kernels before initiating the training process to make predictions. This convolution

operation allows CNNs to capture hierarchical and spatial features within images, which is a critical advantage for image processing tasks. Much like the implementation of the MLP model, the preprocessing algorithm involving FFT and the division of data was also applied in the case of the CNN model [68].

```
# Model with CNN

cnn_model = Sequential()

cnn_model.add(Conv2D(filters = 32, kernel_size = (3,3), padding = 'Same', activation = 'relu', input_shape = (566,400,1)))
cnn_model.add(MaxPooling2D(pool_size=(2,2)))

cnn_model.add(Conv2D(filters = 64, kernel_size = (3,3), padding = 'Same', activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

cnn_model.add(Conv2D(filters = 96, kernel_size = (3,3), padding = 'Same', activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

cnn_model.add(Conv2D(filters = 96, kernel_size = (3,3), padding = 'Same', activation = 'relu'))
cnn_model.add(MaxPooling2D(pool_size=(2,2), strides=(2,2)))

cnn_model.add(Flatten())
cnn_model.add(Dense(512, activation='relu'))
cnn_model.add(Dense(n_classes, activation='softmax')) # 11 classes (-5 to 5)

cnn_model.compile(optimizer = 'adam', loss = 'categorical_crossentropy', metrics = ['accuracy'] )
cnn_model.summary()
```

Figure 34: CNN Model Implementation.

Figure **34** illustrates the implementation of the CNN model. The CNNs are especially well-suited for image processing due to their innate ability to understand local patterns, textures, and spatial relationships. By applying convolution and pooling layers, CNNs automatically detect edges, shapes, and more complex image structures. This feature extraction process significantly reduces the dimensionality of the data, making it computationally efficient and preventing overfitting. Moreover, CNNs inherently consider the spatial arrangement of pixels, which is crucial for recognising objects and patterns in images. These unique capabilities of CNNs make them the preferred choice for various image-related tasks, such as image classification, object detection, and segmentation, as they harness the power of feature extraction to enhance the accuracy and effectiveness of image processing models [68].

The CNN model developed, illustrated in Figure **34**, starts with an input layer with dimensions of 566 in height and 400 in width. This input is then subjected to a sequence of convolutional and pooling layers strategically structured for feature extraction. Similar to the MLP model, the output layer is configured differently for skewed and tilted data, tailored to predict their respective number of classes. An interesting point is the substantial difference in the total trainable parameters between the two models: the CNN model features 43,168,325 trainable

parameters, whereas the MLP model encompasses 116,970,501 trainable parameters. Figure **35** visualises the CNN model's architectural configurations for skewed prediction tasks.

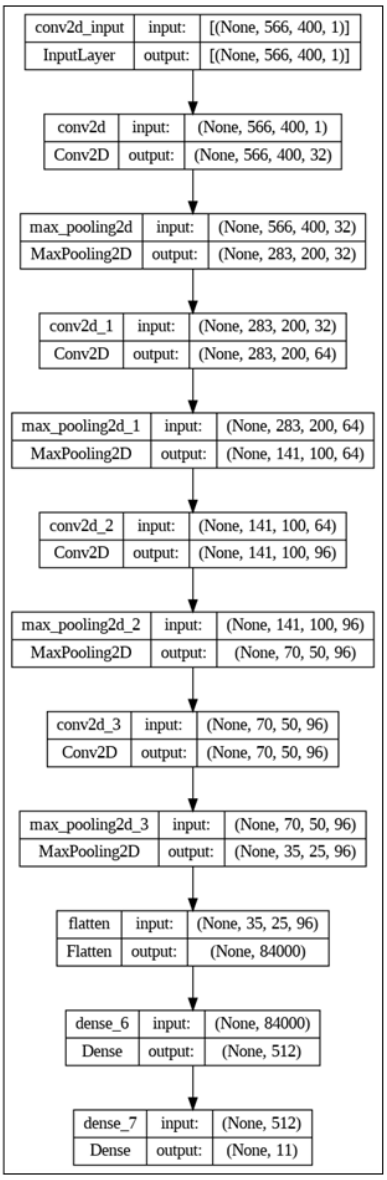


Figure 35: CNN Architecture.

3.4.11 Segmentation for Tabular Data

Segmentation for CDs occurs at multiple levels, with the initial step being the identification of tabular information within the document. This enables tabular data processing based on column and row positioning. Figure **36** depicts the implementation of Morphological operations for detecting tabular areas [69].

```
# Image width
kernel_length = np.array(img).shape[1]//80

# A verticle kernel of (1 x kernel_length)
verticle_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1,kernel_length))

# A horizontal kernel of (kernel_length x 1)
hori_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (kernel_length,1))

# A kernel of (3 x 3) ones
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))


# Morphological operation to detect vertical lines from an image
img_temp1 = cv2.erode(img, verticle_kernel, iterations=3)
vertical_lines_img = cv2.dilate(img_temp1, verticle_kernel, iterations=3)

# Morphological operation to detect horizontal lines from an image
img_temp2 = cv2.erode(img, hori_kernel, iterations=3)
horizontal_lines_img = cv2.dilate(img_temp2, hori_kernel, iterations=3)

# Weighting parameters, this will decide the quantity of an image to be added to make a new image.
alpha = 0.5
beta = 1.0 - alpha


# This function helps to add two image with specific weight parameter to get a third image as summation of two image.
img_final_bin = cv2.addWeighted(vertical_lines_img, alpha, horizontal_lines_img, beta, 0.0)
img_final_bin = cv2.erode(~img_final_bin, kernel, iterations=2)

(thresh, img_final_bin) = cv2.threshold(img_final_bin, 128,255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)

# Find contours for image, which will detect all the boxes
contours, hierarchy = cv2.findContours(img_final_bin, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Figure 36: Implementation of Morphological Operation.

The process involves the detection of contours using vertical and horizontal kernels in OpenCV. Vertical kernels are created with the '**getStructuringElement**(cv2.MORPH_RECT, (1, kernel_length))' function to emphasize vertical structural elements. This vertical kernel undergoes erosion to highlight vertical edges and is then dilated to restore emphasized vertical lines, accentuating the vertical features. Similarly, horizontal kernels are generated using '**getStructuringElement**(cv2.MORPH_RECT, (kernel_length, 1))' to highlight horizontal structural elements. The horizontal kernel undergoes erosion and dilation, emphasizing horizontal features. Combining the resulting images through a bitwise OR operation captures both vertical and horizontal features [70].

Morphological operations, such as erosion and dilation, play a crucial role in enhancing or suppressing specific features in an image. Erosion emphasizes structural elements by eroding away the boundaries of foreground objects, while dilation restores emphasized features by enlarging boundaries.

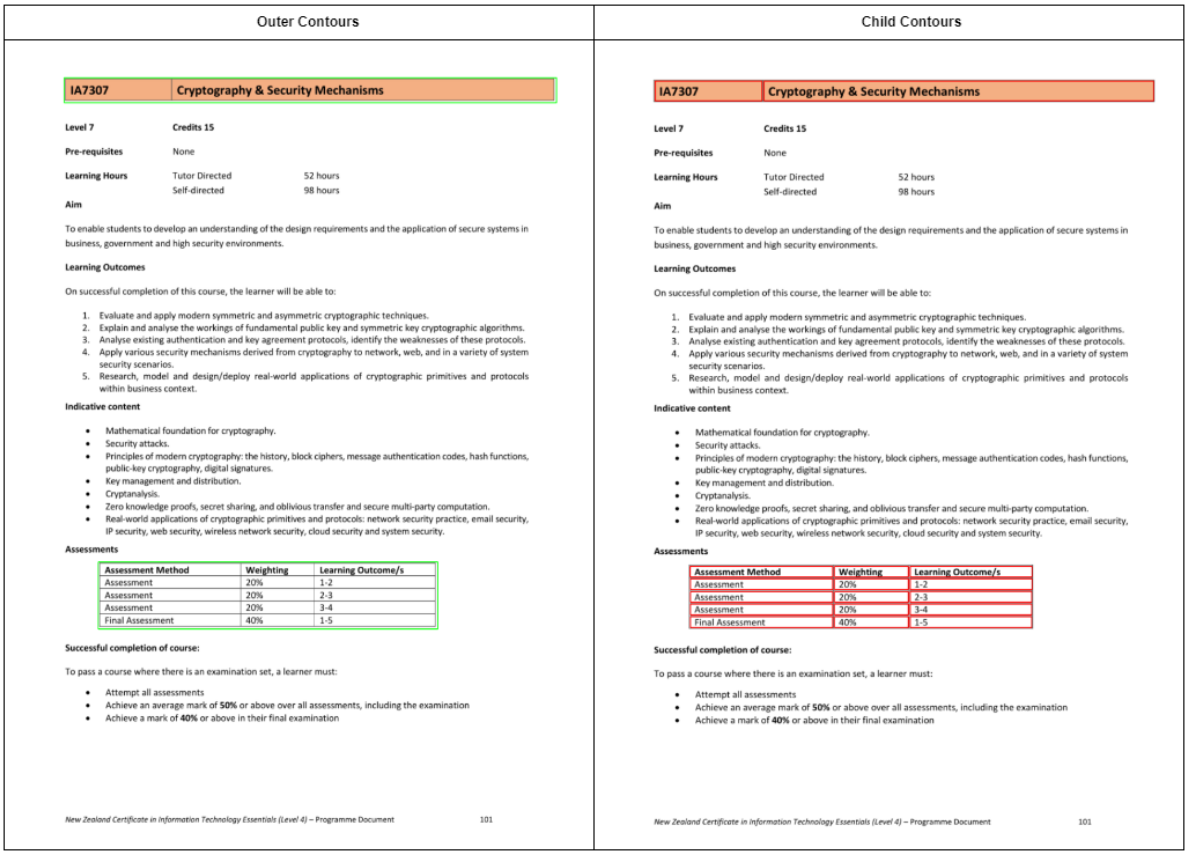


Figure 37: Contours.

After morphological operations, the '**findContours**' [71] function is then used to identify contours in the final binary image. This function returns a list of contours with hierarchical relationships. The hierarchical representation helps distinguish between parent and child contours, providing information on outer contours and their corresponding child contours. Examining this hierarchy allows for identifying outer contours and their child contours, offering a comprehensive understanding of the image structure. Figure **37** displays the outer and child contours identified for the CD document. In the CD document, two main outer contours hold the course code, title, and course assessment details. The OWR pipeline processes this information in three subunits, extracting contours for line and word segmentation based on input parameters. The operation involves extracting contours, connecting outer and child contours to extract course code and title in the first step, followed by course assessment details, and finally, the rest of the information excluding those two outer contours. Figure **38** illustrates the implementation of the function for this process.

```python
def getTableImg(orgImg, outer_rectangles, extract=-1):
  ''' Retuen Table from Image
    As per the CD template there are two tables
    1. Code and Course Title
    2. Assessments
    Input Parameter
    Extract:
      1 - Code and Course Title
      2 - Assessments
      -1 - Without Code, Course Title and Assessments
  '''
  orgImg = orgImg.copy()

  if extract == 1 and len(outer_rectangles) > 0:
    x1, y1, x2,y2 = outer_rectangles[0]
    return orgImg[y1:y2, x1:x2]
  elif extract == 2 and len(outer_rectangles) > 1:
    x1, y1, x2,y2 = outer_rectangles[1]
    return orgImg[y1:y2, x1:x2]
  else:
    for rect in outer_rectangles:
      x1, y1, x2, y2 = rect
      cv2.rectangle(orgImg, (x1, y1), (x2, y2), 0, thickness=cv2.FILLED)
    return orgImg
```

Figure 38: Function to Return Tabular Areas.

3.4.12 Line Segmentation

Line segmentation constitutes the second level of segmentation in the OWR pipeline, aiming to identify lines of words within a given image area. This identification enables the processing of each line individually for subsequent text detection and recognition. The line identification process mirrors the approach used for recognizing tabular data.

```python
def getLines(img):
  kernel = np.ones((CharHeight4Line, CharWidth4Line), np.uint8)
  try:
    dilated = cv2.dilate(img, kernel, iterations=1)

    #cv2_imshow(dilated)

    (contours, heirarchy) = cv2.findContours(dilated.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
    sorted_contours_lines = sorted(contours, key = lambda ctr : cv2.boundingRect(ctr)[1]) # (x, y, w, h)

    line_contours = []
    for line in sorted_contours_lines:
      x, y, w, h = cv2.boundingRect(line)
      line_contours.append((x, y, x + w, y + h))

  except: line_contours = []

  return line_contours
```

Figure 39: Line Segmentation - Implementation.

In this process, '**cv2.dilate**' is employed with a specific kernel to identify the lines. The kernel plays a crucial role, and in this research, the finely tuned kernel is set to (6, 185), specifying the height and width. Adjusting these parameters becomes necessary when applying the

function to a different document with similar characteristics. Figure **39** illustrates the function implementation, and Figure **40** provides a side-by-side view, depicting the dilated image with the kernel and the final result of contour detection [72].

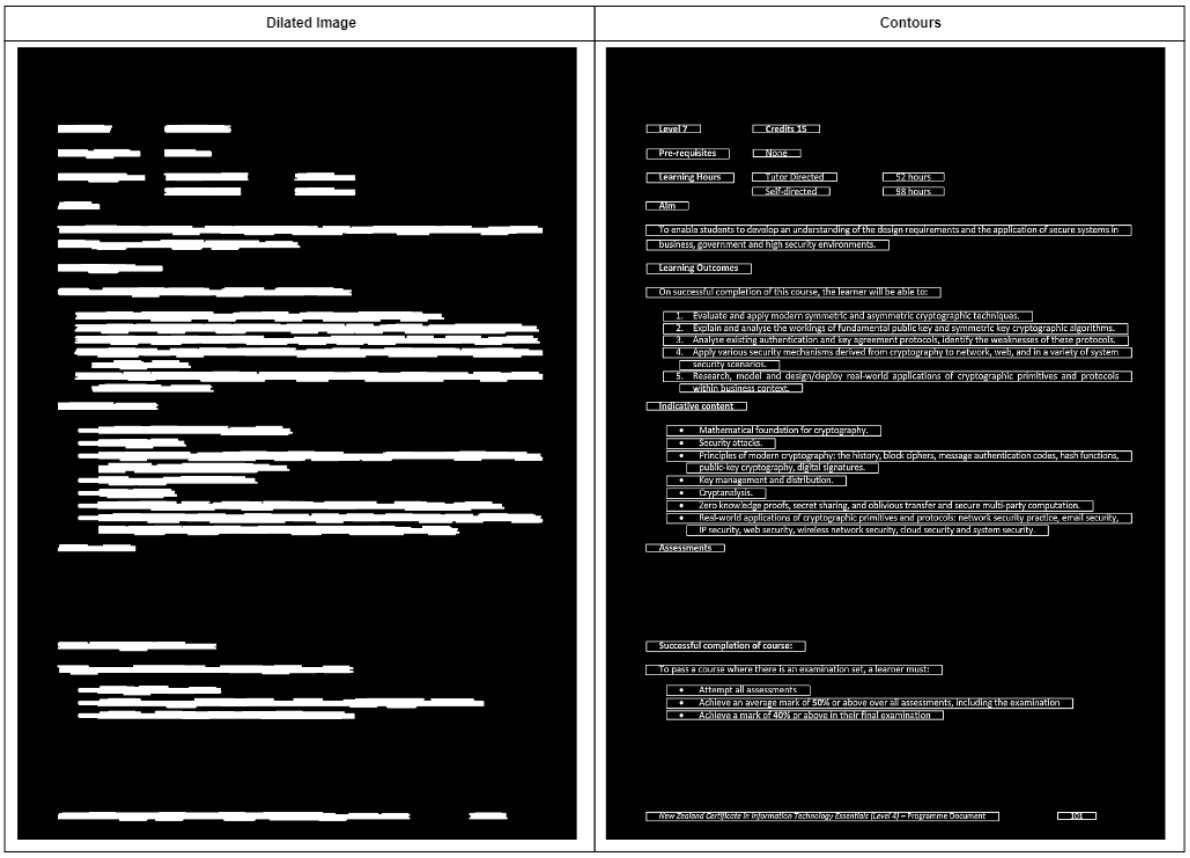


Figure 40: Line Segmentation.

Upon identifying contours, sorting is required based on the y-coordinate to accurately gather lines from top to bottom. This sorting step ensures that the subsequent processing considers the spatial arrangement of lines in the correct sequence. The effective use of '**cv2.dilate**' and the specific kernel in this context enhances the pipeline's ability to discern and isolate lines of text within the image, contributing to the overall precision of the OWR system. The adaptability of these parameters allows for the versatility of the function, facilitating its application to different documents with varying layouts and structures.

3.4.13 Word Segmentation

Word segmentation represents the concluding phase in the segmentation process, wherein the segmented lines undergo processing to identify individual words. From an implementation standpoint, both line and word segmentation follow identical procedures, differing only in the

kernel utilized. For this research, a specifically tuned (6, 5) kernel is employed, fine-tuned to suit the characteristics of CD documents. It is crucial to note that adjustments to the word segmentation kernel become necessary when dealing with images of varying resolutions [72].

```
def calculate_area(coord):
    # Calculate the area of a rectangle defined by [x, y, w, h]
    return (coord[2] - coord[0]) * (coord[3] - coord[1])

def is_overlapping(coord1, coord2):
    buffer = 10
    # Check if two rectangles defined by [x, y, w, h] are overlapping
    x1, y1, x12, y12 = coord1
    x2, y2, x22, y22 = coord2
    #return (x1 < x2 + w2 and x1 + w1 > x2 and y1 < y2 + h2 and y1 + h1 > y2)
    return ((x1 - buffer <= x2 <= x12 + buffer and x1 - buffer <= x22 <= x12 + buffer
             and y1 - buffer <= y2 <= y12 + buffer and y1 - buffer <= y22 <= y12 +buffer)
           or (x2 - buffer <= x1 <= x22 + buffer and x2 - buffer <= x12 <= x22 + buffer
               and y2 - buffer <= y1 <= y22 + buffer and y2 - buffer <= y12 <= y22 + buffer))

def remove_overlapping(rectangles):
    # Sort the rectangles by area (largest first)
    rectangles.sort(key=calculate_area, reverse=True)

    # Initialize a list to store non-overlapping rectangles
    non_overlapping_rectangles = []

    for rect in rectangles:
        if all(not is_overlapping(rect, existing_rect) for existing_rect in non_overlapping_rectangles):
            non_overlapping_rectangles.append(rect)

    non_overlapping_rectangles = sorted(non_overlapping_rectangles, key=lambda rect : rect[0])
    # Sort the outer rectangles based on their top-left coordinates (y, x)
    #outer_rectangles = sorted(outer_rectangles, key=lambda rect: (rect[1], rect[0]))

    return non_overlapping_rectangles
```

Figure 41: Remove Overlapping.

Given that words possess different widths based on the length of the characters they contain, the chosen kernel must effectively identify these variations. In this context, the width of the kernel is set to approximately one character length. Consequently, the '**findContours**' function may return multiple contours for the same word, potentially fragmenting the word into several parts. The function depicted in Figure **41** has been implemented to address this. This function determines whether the contours refer to the same area. If they do, it retains the contour covering the largest area, discarding smaller contours that overlap with that region. The final effect of the word segmentation function is illustrated in Figure **42**.

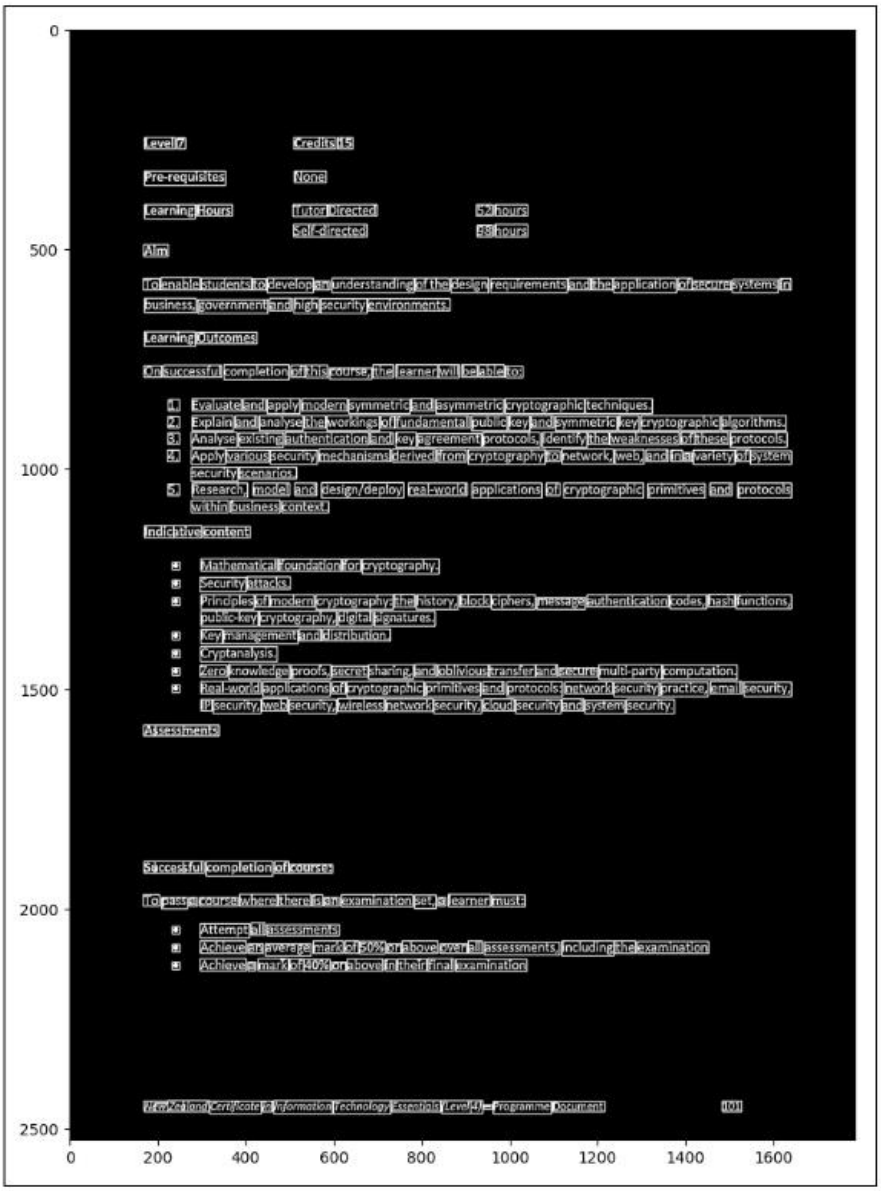


Figure 42: Word Segmentation.

In the OWR pipeline, word segmentation occurs after line segmentation for each segmented line. In Figure **42**, the word segmentation process is applied to the entire document for demonstration purposes, showcasing the function's effectiveness in identifying and isolating individual words within the image. This approach accurately recognises and processes text elements in the OWR system.

3.4.14 OWR Model

The OWR Model is the cornerstone of the OWR pipeline, constituting the final phase in text recognition. When approaching text recognition, two viable options presented themselves. The first involves recognizing individual characters within a word through an OCR model, subsequently constructing complete words. The second option involves recognizing entire words using a sophisticated model equipped with a sequence model, such as an RNN.

The OCR model, in essence, operates as an image classification neural network. Its functionality pivots on recognising characters by leveraging features extracted from the image, with a heightened emphasis on preprocessing, where the image quality assumes a pivotal role. In contrast, an OWR model backed by an RNN goes beyond feature extraction from images. It incorporates a sequence model, enabling the RNN to learn word formations, thereby increasing the likelihood of success. Consequently, for this research, the decision was made to deploy an RNN-based model for word recognition.

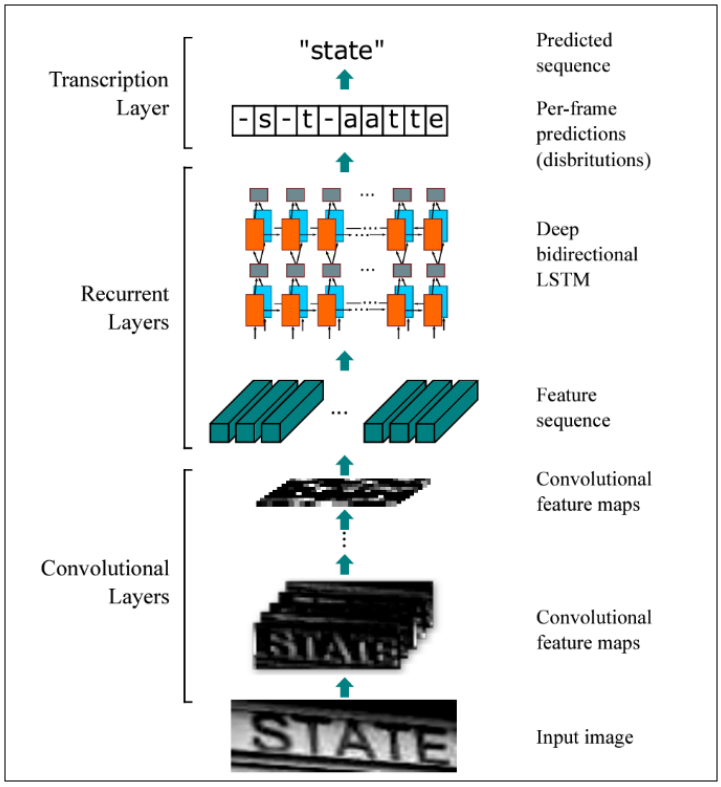


Figure 43: OWR Network Architecture, as illustrated in [14]

The *Convolution Recurrent Neural Network* (CRNN) proposed by Shi et al. 2017 [14] is a highly effective solution to recognise words, and its high-level architecture is visualized in

Figure **43**. This research deploys the same as the OWR model in the text detection and recognition pipeline. It comprises three essential components: the convolutional layer, the recurrent layer, and the transcription layer. The convolutional layer automatically extracts a feature sequence from each input image. The recurrent network is designed to make recognitions for each frame of the feature sequence output by the convolutional layer. Finally, the transcription layer translates per-frame recognitions made by the recurrent layers into a label sequence. CRNN represents a fusion of CNN and RNN architectures, allowing both networks to be jointly trained with a single loss function. This integrated approach harnesses the strengths of both CNN and RNN, enhancing the model's capability to recognize and interpret words in image-based documents.

3.4.14.1 Convolution Layer

The convolutional layers play a pivotal role in feature extraction within the CRNN architecture, involving a sequence of operations, including convolution, max pooling, and normalization. Notably, incorporating the normalization layer is a distinctive feature specific to the AlexNet architecture [73]. This sequence of operations is designed to derive a feature representation sequentially from the input image.

Before inputting into the network, all images are scaled to the same height, ensuring uniformity in the data fed to the subsequent layers. The CNN layers generate feature maps, from which a sequence of feature vectors is extracted. These feature vectors form the input for the subsequent RNN layer. Each feature vector in the sequence is formed column-wise from the feature maps, following a left-to-right generation approach. For instance, the second feature vector is a concatenation of the second columns from all the maps, where the width of each column is consistently fixed at a single pixel.

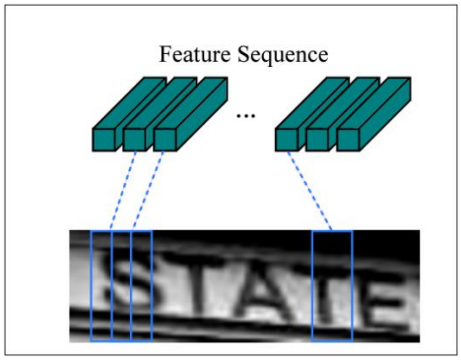


Figure 44: Feature Vector, as illustrated in [14]

Localization is a fundamental concept embedded in CNN, wherein feature and element-wise activation functions operate on local regions. Each column in the feature maps corresponds to a rectangle region of the original image. These rectangles are arranged in the same order as their corresponding columns on the feature maps, proceeding from left to right. Figure **44** illustrates this association, demonstrating how each vector in the feature sequence is linked to a specific field [74].

In the context of CNN, input images are required to be scaled to a fixed size to align with the network's predetermined input dimensions. However, this fixed-size constraint is not suitable for sequence-like objects characterized by significant length variations. CRNN addresses this limitation by transforming deep features into sequential representations, ensuring invariance to the length variations observed in sequence-like objects.

3.4.14.2 Recurrent Layer

The CRNN model architecture integrates a deep bi-directional RNN layer, specifically using LSTM units over the CNN layer. This configuration is designed to predict a sequence of labels, focusing on character recognition in this particular application.

The significance of the RNN layer lies in its multifaceted advantages. Firstly, it exhibits a robust ability to capture contextual information within a sequence, introducing stability and aiding in recognising image-based sequences. This is a departure from traditional OCR models, which treat each symbol independently. By considering the contextual information, the RNN contributes to resolving ambiguities associated with certain characters, enhancing the overall accuracy of the recognition process. Secondly, the RNN layer shares the valuable characteristic of back-propagating errors with CNN layers. This cooperative functionality allows for the joint training of both the RNN and CNN within a unified network. Lastly, the RNN layer demonstrates remarkable flexibility by operating effectively on sequences of arbitrary length, a critical feature for processing words in diverse contexts [75].

Within the CRNN model, the RNN component is specifically implemented using bi-directional LSTM units, as elucidated in Figure 43. Notably, Vanilla RNN units face challenges such as the vanishing gradient problem, limiting their ability to store contextual information over longer sequences and complicating the training process. The LSTM architecture overcomes these challenges by incorporating a memory cell and three multiplicative gates - the input,

output, and forget gates. Conceptually, the memory cell stores past contextual information, while the gates control the storage and removal of this information over extended periods. This unique design empowers LSTM to effectively capture long-range dependencies often present in image-based sequences.

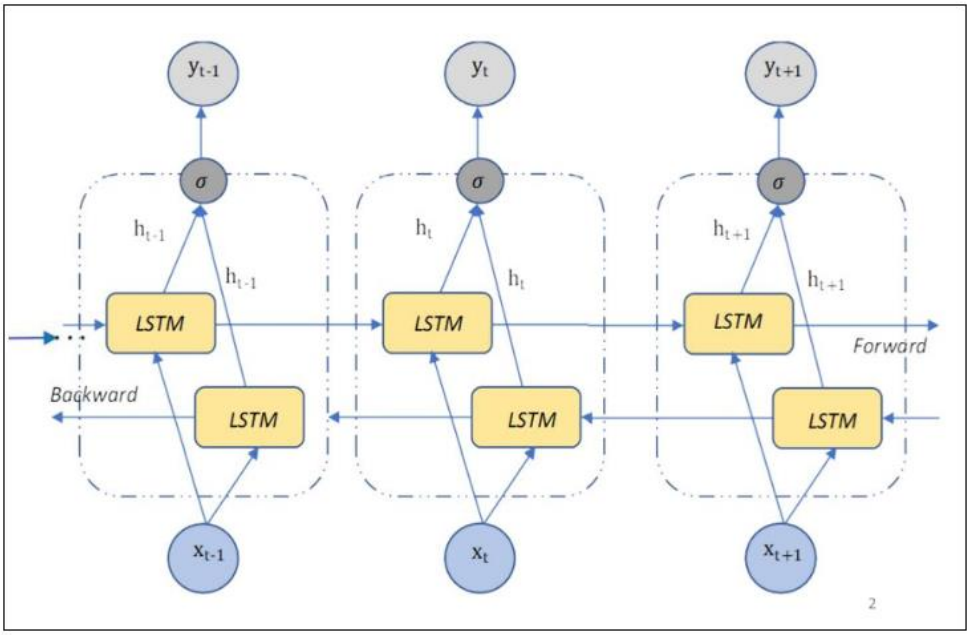


Figure 45: Bi-Directional LSTM Units.

LSTM's directional nature relies on past contexts; however, in the context of image-based sequences, information from both directions is often beneficial and complementary. To capitalize on this, the CRNN model adopts a bidirectional LSTM configuration, combining two LSTMs, one operating in the forward direction and the other in the backward direction. This bidirectional LSTM architecture, as depicted in Figure **45**, enables error back-propagation in both directions, thereby refining the prediction sequence and enhancing the overall model performance. This comprehensive approach ensures that the CRNN model is adept at handling the complexities of image-based sequence recognition, offering superior accuracy and contextual understanding [76].

3.4.14.3 Transcription Layer

The final layer in the CRNN model architecture is the transcription layer, tasked with transforming the per-frame predictions generated by the RNN into a meaningful label sequence. This process, known as transcription, is essentially the derivation of a label sequence

with the highest probability, considering the conditions set by the per-frame predictions. In mathematical terms, transcription involves finding the label sequence 'l' that maximizes the probability, given the per-frame predictions 'y = y1, ..., yT.' To achieve this, the model employs the concept of Conditional Probability, mainly as defined in the *Connectionist Temporal Classification* (CTC) layer [77].

```
def Image_text_recogniser_model_1(stage,drop_out_rate=0.35):

    if K.image_data_format() == 'channels_first':
        input_shape = (1, img_w, img_h)
    else:
        input_shape = (img_w, img_h, 1)

    model_input=Input(shape=input_shape,name='img_input',dtype='float32')

    # Convolution layer
    model = Conv2D(64, (3, 3), padding='same', name='conv1', kernel_initializer='he_normal')(model_input)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = MaxPooling2D(pool_size=(2, 2), name='max1')(model)

    model = Conv2D(128, (3, 3), padding='same', name='conv2', kernel_initializer='he_normal')(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = MaxPooling2D(pool_size=(2, 2), name='max2')(model)

    model = Conv2D(256, (3, 3), padding='same', name='conv3', kernel_initializer='he_normal')(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = Conv2D(256, (3, 3), padding='same', name='conv4', kernel_initializer='he_normal')(model)
    model=Dropout(drop_out_rate)(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = MaxPooling2D(pool_size=(1, 2), name='max3')(model)

    model = Conv2D(512, (3, 3), padding='same', name='conv5', kernel_initializer='he_normal')(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = Conv2D(512, (3, 3), padding='same', name='conv6')(model)
    model=Dropout(drop_out_rate)(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)
    model = MaxPooling2D(pool_size=(1, 2), name='max4')(model)

    model = Conv2D(512, (2, 2), padding='same', kernel_initializer='he_normal', name='con7')(model)
    model=Dropout(0.25)(model)
    model = BatchNormalization()(model)
    model = Activation('relu')(model)

    # CNN to RNN
    model = Reshape(target_shape=((42, 1024)), name='reshape')(model)
    model = Dense(64, activation='relu', kernel_initializer='he_normal', name='dense1')(model)

    # RNN layer
    model=Bidirectional(LSTM(256, return_sequences=True, kernel_initializer='he_normal'), merge_mode='sum')(model)
    model=Bidirectional(LSTM(256, return_sequences=True, kernel_initializer='he_normal'), merge_mode='concat')(model)

    # transforms RNN output to character activations:
    model = Dense(num_classes, kernel_initializer='he_normal',name='dense2')(model)
    y_pred = Activation('softmax', name='softmax')(model)


    labels = Input(name='ground_truth_labels', shape=[max_length], dtype='float32')
    input_length = Input(name='input_length', shape=[1], dtype='int64')
    label_length = Input(name='label_length', shape=[1], dtype='int64')

    #CTC loss function
    loss_out = Lambda(ctc_loss_function, output_shape=(1,),name='ctc')([y_pred, labels, input_length, label_length]) #(None, 1)

    if stage=='train':
        return model_input,y_pred,Model(inputs=[model_input, labels, input_length, label_length], outputs=loss_out)
    else:
        return Model(inputs=[model_input], outputs=y_pred)
```

Figure 46: OWR Model Implementation.

The CTC layer introduces a unique approach to defining probability for a label sequence ‘l’, conditioned on the per-frame predictions ‘y = y1, ..., yT.’ Notably, this definition abstracts away from the specific positions of individual labels within the sequence, focusing solely on the sequence itself. Applying the negative log-likelihood of this probability serves as the objective for training the CRNN network. An intriguing advantage of utilizing CTC lies in its ability to explicitly eliminate the need to label individual characters’ positions during training. Instead, the model can be trained effectively using only images and corresponding label sequences. This circumvents the labour-intensive task of manually annotating the position of each character in the training data.

For a comprehensive understanding, Figure **46** visually represents the implemented CRNN model, encompassing all the layers elucidated in the preceding discussion. This visualization offers a clear overview of the interconnected components, showcasing the integration of convolutional, recurrent, and transcription layers, each playing a crucial role in the model's capability for sequence-based image recognition.

3.4.14.4 Dataset

The RCNN model undergoes training using the “MJ Synthetic Word Dataset” [78], a substantial dataset containing approximately 9 million grayscale images encompassing 90,000 English words. This dataset is accompanied by three text files, namely annotation_train.txt, annotation_val.txt, and annotation_test.txt, which catalogue the images and their respective text labels directly within the filenames. A snippet from the train_annotation text file is visually represented in Figure **47**, offering a glimpse into the organization of values within.

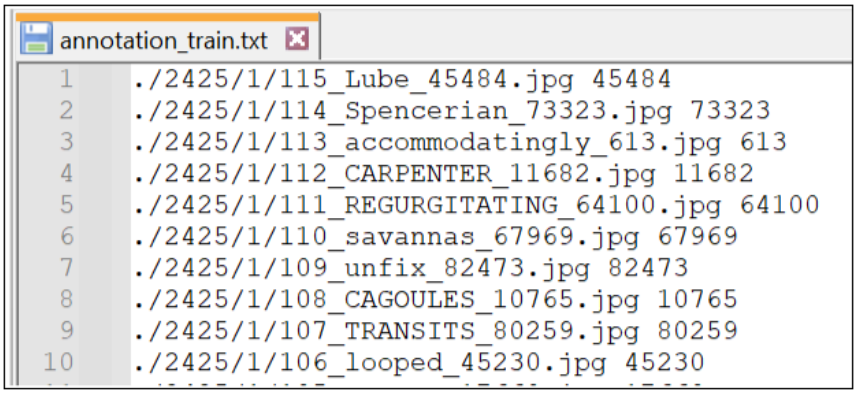


Figure 47: Train Annotation Values.

When compressed, the “MJ Synthetic Word Dataset” occupies an estimated size of 10 GB. Figure **47** depicts the images systematically arranged across multiple folders and subfolders, totalling in the thousands. Given that the research development environment operates within Google Colab, data must be accessible either directly on the Colab platform or via Google Drive, which can be mounted onto the Colab environment. While transferring a 10 GB file to Google Drive or Colab is manageable, the intricate folder structure poses a challenge when try to unzip the data. Unfortunately, due to its volume of folders, neither a tool for programmatically unzipping the dataset nor the ability for Google Colab to unzip it programmatically is available. Consequently, the only viable option was to manually transfer and unzip all files from a local PC to Google Drive, a task that spanned nearly two weeks of continuous processing. This effort was deemed essential to ensure the availability of all images for thorough *exploratory data analysis* (EDA) and to capture a diverse array of word sequences.

```python
def createUniqueDataset(df, number_of_records, no_of_samples, destination_folder):

  # Return if dataframe got less num of records
  if len(df) < number_of_records:
    return None

  # Get unique values
  unique_values = df['Labels'].unique()

  # Create New Dataframe
  new_df = pd.DataFrame()
  added_paths = set()
  destination_list=[]
  height_list = []
  width_list = []
  label_list = []
  count = 140000
  print_count = 70

  while count < number_of_records + 1:
    for value in unique_values:
      if len(df[df['Labels'] == value]) < no_of_samples:
        samples = len(df[df['Labels'] == value])
      else: samples = no_of_samples

      # Sample one row for each unique value
      sampled_row = df[df['Labels'] == value].sample(n=samples)

      for index, row in sampled_row.iterrows():

        # Check same value alredy been added to dataset
        imagePath = row['ImagePath']
        if imagePath in added_paths:
          continue

        #Removing the extra folder structures
        _,_,_,_,_,_,_,Name = imagePath.split('/')
        _,img,_ = Name.split('_')
        sub_folder = str(((10000 + count)//10000 )* 10000)
        destination= f'{destination_folder}/{sub_folder}/{count}_{img}.jpg'

        cv_img=cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)
```

Continue...

```python
        if cv_img is not None:

          #create folder if not exists
          if not os.path.exists(f'{destination_folder}/{sub_folder}'):
            os.makedirs(f'{destination_folder}/{sub_folder}')

          cv2.imwrite(destination,cv_img)
          destination_list.append(destination)
          height_list.append(cv_img.shape[0])
          width_list.append(cv_img.shape[1])
          label_list.append(row['Labels'])

          count += 1

          # Append the sampled row to the new DataFrame
          # new_df = pd.concat([new_df,row])

          # Add the value to the set of added values
          added_paths.add(imagePath)

        # Progress
        if (count//2000 > print_count):
          print(f'Current records: {count} - Time: {time.time()}')
          print_count += 1

        # Break the loop if the desired number of records is reached
        if count > number_of_records :
          break
      # Break the loop if the desired number of records is reached
      if count > number_of_records :
        break

  new_df['ImagePath'] = destination_list
  new_df['Labels'] = label_list
  new_df['Height'] = height_list
  new_df['Width'] = width_list

  # Reset the index of the new DataFrame
  #new_df = new_df.reset_index(drop=True)

  return new_df
```

Figure 48: Implementation of Data Creation.

During the training (fit) processes, the model encountered significant delays while accessing images directly from Google Drive. To mitigate this, a decision was made to relocate the data to the Colab environment. However, copying the multitude of folders directly from Google

Drive posed the same folder-related challenges, necessitating a programmatic approach to move images individually. A representative sample of 200,000 images from the training set, 12,000 images from the validation set, and 15,000 images from the test annotations were selected to expedite this process. Figure 48 illustrates the code implementation devised to facilitate the movement of images. A specialized code was also implemented to extract at least one image for each unique word in the training dataset. Constructing the three datasets required close to 48 hours of continuous activity.

3.4.14.5 Exploratory Data Analysis

The initial phase involves the creation of labels (ground truth) for the images, as illustrated in Figure 47, where all labels are integrated into the image path. Consequently, new files containing the image path and its corresponding label are generated. Notably, all labels are standardized to uppercase letters, regardless of the case used in the images. This decision streamlines the complexity of letter recognition, as the model only needs to anticipate 26 letters rather than 52. Model recognition includes upper case letters and numeric values; hence, the complete list of recognition values is '0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ'. Figure **49** visually represents the newly constructed dataset with labels.

	ImagePath	Labels
0	/content/gdrive/MyDrive/90K_SynthImageDataset/...	SPENCERIAN
1	/content/gdrive/MyDrive/90K_SynthImageDataset/...	ACCOMMODATINGLY
2	/content/gdrive/MyDrive/90K_SynthImageDataset/...	CARPENTER
3	/content/gdrive/MyDrive/90K_SynthImageDataset/...	REGURGITATING
4	/content/gdrive/MyDrive/90K_SynthImageDataset/...	SAVANNAS

Figure 49: Labelled Data.

Analysis of Image Height and Width

Cumulative distribution function plots for image height and width are depicted in Figure **50**. Notably, most images fall within the height range of 29 to 32. Similarly, more than 95% of the images exhibit a width of 200 or less.

Further insights are gained through percentile analysis, as illustrated in Figure 51, focusing on the 90th and 99th percentiles for both image height and width within the training sample. While there is minimal variation in the 90th and 99th percentile values for image height, a substantial

difference is observed in image width. Specifically, the 90th percentile registers at 172, whereas the 99th percentile is 251.

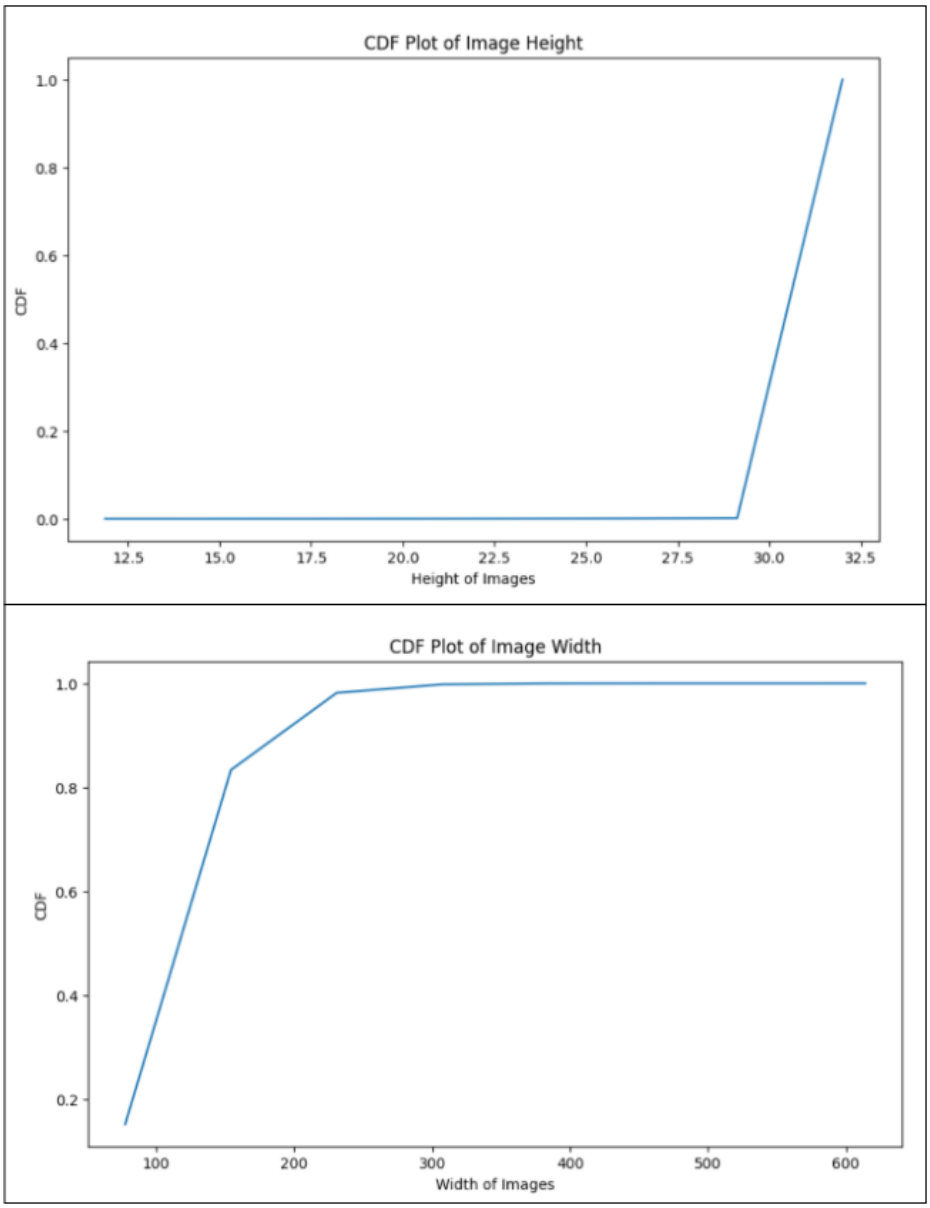


Figure 50: CDF Plots for Height and Width.

Consequently, a more detailed examination is conducted between the 90th and 99th percentiles for width, as illustrated in Figure 51. The distinct values identified across percentiles for input image width led to select the width of 180, representing the 92nd percentile, given its prevalence among images. For input image height, the value of 32 is chosen, as it signifies the maximum height with minimal variability observed among images.

	Height	Width
count	200000.000000	200000.000000
mean	31.037255	117.404805
std	0.354059	42.709331
min	9.000000	1.000000
25%	31.000000	88.000000
50%	31.000000	111.000000
75%	31.000000	139.000000
max	32.000000	614.000000

```
Train Images Height 90 percentile : 31.0
Train Images Height 99 percentile : 32.0
Train Images Width 90 percentile : 172.0
Train Images Width 99 percentile : 251.0
```

```
Train Images Width 90 percentile : 172.0
Train Images Width 91 percentile : 176.0
Train Images Width 92 percentile : 180.0
Train Images Width 93 percentile : 185.0
Train Images Width 94 percentile : 190.0
Train Images Width 95 percentile : 196.0
Train Images Width 96 percentile : 204.0
Train Images Width 97 percentile : 214.0
Train Images Width 98 percentile : 227.0
Train Images Width 99 percentile : 251.0
```

Figure 51: Height & Width Percentile Analysis.

3.5 EVALUATION

3.5.1 Overview

The primary aim of the evaluation stage is to gauge the efficacy of the developed solution in alignment with the predefined research objectives. This evaluation entails systematically applying predefined criteria to the various artefacts created. The normalized ER schema undergoes evaluation based on its ability to persist all extracted information without redundancy. Assessment of the GT capture artefact is a manual task, necessitating a comparison of the information extracted by the artefact with the content in the CD. To ensure accuracy, randomly selected documents are manually verified and examined for parsing errors and attribute mismatches between the schema and the extracted information based on logged errors into the exception table by the artefact.

The artefacts constituting the OWR pipeline undergo evaluation at multiple levels. Models tailored for specific tasks, such as skewness and tilt correction, along with the performance of the OWR models, are initially assessed using training and validation data. Hyperparameters are fine-tuned during this stage to optimize performance for the specific tasks and datasets they are trained on. Ultimately, the best models are evaluated collectively within the OWR pipeline to ensure seamless end-to-end text detection and recognition. A comprehensive testing strategy is employed, involving permutations of data and models, as outlined in Table 2.

Ref.	Images with Skew & Tilt	Skew & Tilt Correction	Identification of Tabular Info.	Line Segmentation	Word Segmentation	OWR Model
Config 1	No	No	Yes	Yes	Yes	Yes
Config 2	Yes	No	Yes	Yes	Yes	Yes
Config 3	Yes	OpenCV	Yes	Yes	Yes	Yes
Config 4	Yes	CNN	Yes	Yes	Yes	Yes

Table 2: Evaluation Parameters.

Table 2 delineates the diverse evaluations conducted on artefacts, illustrating that components like tabular area identification, line and word segmentation, and the OWR model collectively form the backbone of the OWR pipeline as each image traverses through these crucial artefacts. Moreover, the selection and deployment of models to correct skewness and tilt introduce varied configurations. Configuration 1 aims to verify pipeline performance on CD images without tilt or skew. Configuration 2 involves pipeline verification with tilted and skewed data without skewness or tilt correction. Configuration 3 uses OpenCV Hough transformation to correct skewness and tilt in tilted and skewed images. Lastly, Configuration 4 employs CNN to correct skewness and tilt in tilted and skewed images.

3.5.2 Evaluation Criteria

In machine learning, the selection of evaluation metrics is contingent upon the specific characteristics of the task. Classification tasks typically leverage metrics like accuracy, precision, recall, and F1 score, whereas regression tasks commonly employ MSE. Precision gauges the ratio of true positives to all detections, while recall measures the ratio of true positives to all true instances that need to be identified. The F1 score, serving as an overarching, unitary indicator of algorithm performance, is the harmonic mean of recall and precision. The confusion matrix emerges as a vital tool, furnishing a detailed breakdown of a model's performance and facilitating the computation of precision, recall, and F1 score [79]. The chosen classification tasks' metrics for evaluating the performance of the models are explained as follows.

$$Accuracy = \frac{Tp + Tn}{(Tp + Fp + Fn + Tn)}$$

$$Precision = \frac{Tp}{(Tp + Fp)}$$

$$Recall = \frac{Tp}{(Tp + Fn)}$$

$$f1\ score = 2 * \frac{Recall * Precision}{(Recall + Precision)}$$

Where *Tp = True positive, Tn = True negative, Fp = False positive* and *Fn = False negative*.

MSE, a specialized loss function employed primarily in regression tasks, quantifies the average squared difference between predicted and actual values. It calculates the mean of the squared residuals, providing a measure of the model's accuracy in predicting continuous numerical outcomes. A lower MSE signifies superior model performance, indicating smaller errors in predictions.

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - Yhat_i)^2$$

Where *n = Number of data points,* Y_i *= Observed values,* $Yhat_i$ *= Predicted values*.

The loss factor also plays a significant role in the training and validation of models. The loss represents a model's performance during training, reflecting the error between predicted and actual values in the training dataset. The objective during training is to minimize this loss. Different loss functions, such as cross-entropy for classification tasks or MSE for regression tasks, are applied based on the nature of the problem. Monitoring the loss over epochs facilitates the optimization of model parameters, ensuring convergence towards an optimal solution.

Assessing the OWR pipeline's performance necessitates an analysis of sentence similarity. As mentioned above, traditional metrics fall short in this context, as the sequence of words in a sentence becomes a crucial factor. To address this, this research employs the '**SequenceMatcher**' from the Python '**difflib**' library [80]. Specifically, the '**SequenceMatcher**' class compares two strings and yields a similarity score ranging from 0 (no match) to 1 (complete match). The ratio calculation in the '**SequenceMatcher**' class is determined through the following process. Additionally, the class provides a list of mismatched words in both sentences. To comprehend the behaviour of the SequenceMatcher class, I examine several examples, as illustrated in Figure **52**, demonstrating the implementation of the '**difflib.SequenceMatcher**' class.

```python
def compare_statements(statement1, statement2):

    if statement1 is None:
      statement1 = ''
    if statement2 is None:
      statement2 = ''
    # Tokenize statements into words
    words1 = text_cleanUp(statement1).split()
    words2 = text_cleanUp(statement2).split()

    # Calculate matching ratio using SequenceMatcher
    matcher = difflib.SequenceMatcher(None, words1, words2)
    matching_ratio = matcher.ratio()
    matched_words1 = []
    matched_words2 = []
    # Get the list of matching and mismatching words
    matching_words = [word for word in matcher.get_matching_blocks()]
    for matching_words_row in matching_words:
      matched_words1.append(words1[matching_words_row[0]:matching_words_row[0] + matching_words_row[2]])
      matched_words2.append(words2[matching_words_row[1]:matching_words_row[1] + matching_words_row[2]])

    matched_words1 = [word for sublist in matched_words1 for word in sublist]
    matched_words2 = [word for sublist in matched_words2 for word in sublist]


    mismatched_words1 = list(set(words1) - set(matched_words1))
    mismatched_words2 = list(set(words2) - set(matched_words2))

    return matching_ratio, mismatched_words1, mismatched_words2
```

Figure 52: Implementation of difflib.

The '**ratio**' function outputs the similarity score for the two sentences provided to the class, while the '**get_matching_blocks**' function identifies matching words in each sentence and highlights the missing words in either sentence.

$$\text{ratio} = \frac{2 * matches}{(len\_seq1 \ + len\_seq2)}$$

The variable '**matches**' represents the length of the matches, where the '**SequenceMatcher**' identifies the longest contiguous matching subsequence between the two sequences under consideration. Subsequently, it computes the total length of the two compared sequences ('**len_seq1**' and '**len_seq2**'). The ratio is then derived by evaluating the above expression.

```
statement1 = 'HELLO'
statement2 = 'HELLOW'


Matching Ratio: 0.00%
Mismatched Words in Statement 1: ['HELLO']
Mismatched Words in Statement 2: ['HELLOW']
```

Example 1: Statement Matching 1.

In Example 1, the comparison of ‘HELLO’ and ‘HELLOW’ results in a matching ratio of 0%.

```
statement1 = 'FINE WORD'
statement2 = 'WORD'

Matching Ratio: 66.67%
Mismatched Words in Statement 1: ['FINE']
Mismatched Words in Statement 2: []
```

Example 2: Statement Matching 2.

Example 2 compares ‘FINE WORD’ and ‘WORD’, yielding a matching ratio of 66.67%.

```
statement1 = 'FINE WORD'
statement2 = 'FINE WORDS'


Matching Ratio: 50.00%
Mismatched Words in Statement 1: ['WORD']
Mismatched Words in Statement 2: ['WORDS']
```

Example 3: Statement Matching 3.

Example 3 compares ‘FINE WORD’ and ‘FINE WORDS’, resulting in a matching ratio of 50%.

```
statement1 = 'FINE WORD'
statement2 = 'FINE WORD '


Matching Ratio: 100.00%
Mismatched Words in Statement 1: []
Mismatched Words in Statement 2: []
```

Example 4: Statement Matching 4.

Example 4 compares ‘FINE WORD’ and ‘FINE WORD ’ where the second statement ends with additional space but still results in a 100% matching ratio. Example 5 presents statements from a CD document. The first statement is from the GT: ‘DESCRIBE THE ROLE OF NETWORK DEVICES AND DATA COMMUNICATIONS PROTOCOL ***MODEL*** LAYERS’. The second statement is from the OWR pipeline output: ‘DESCRIBE THE ROLE OF NETWORK DEVICES AND DATA COMMUNICATIONS PROTOCOL ***MODAL*** LAYERS’. The matching ratio, in this case, is 91.67%.

```
statement1 = 'DESCRIBE THE ROLE OF NETWORK DEVICES AND DATA COMMUNICATIONS PROTOCOL MODEL LAYERS'
statement2 = 'DESCRIBE THE ROLE OF NETWORK DEVICES AND DATA COMMUNICATIONS PROTOCOL MODAL LAYERS'


Matching Ratio: 91.67%
Mismatched Words in Statement 1: ['MODEL']
Mismatched Words in Statement 2: ['MODAL']
```

Example 5: Statement Matching 5.

These examples show that the ‘**difflib.SequenceMatcher**’ class adeptly identifies matching ratios, a methodology other researchers employ to assess documents and source codes [81]. In this project, I extend this approach to compare each attribute of entities for matching ratios, focusing on statements from the GT and those fetched from the OWR pipeline for evaluation.

CHAPTER 4

CHAPTER 5 REFERENCES

[1] T. Loughran and B. McDonald, 'Textual analysis in accounting and finance: A survey', *Journal of Accounting Research*, vol. 54, no. 4, pp. 1187–1230, Sep. 2016, doi: 10.1111/1475-679X.12123.

[2] H. T. Ha, M. Medved', Z. Nevěřilová, and A. Horák, 'Recognition of OCR invoice metadata block types', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 11107 LNAI, pp. 304–312, 2018, doi: 10.1007/978-3-030-00794-2_33/TABLES/2.

[3] C. Bardelli, A. Rondinelli, R. Vecchio, and S. Figini, 'Automatic electronic invoice classification using machine learning models', *Machine Learning and Knowledge Extraction 2020, Vol. 2, Pages 617-629*, vol. 2, no. 4, pp. 617–629, Nov. 2020, doi: 10.3390/MAKE2040033.

[4] School of Innovation Design and Technology, 'Course descriptors (postgraduate diploma in IT)', Wellington, 2023. Accessed: Apr. 17, 2023. [Online]. Available: https://moodle.whitireia.ac.nz/pluginfile.php/1569273/mod_resource/content/1/Course%20Descriptors-%20Postgraduate%20Diploma%20in%20IT.pdf

[5] School of Innovation Design and Technology, 'Master of information technology handbook', Wellington, 2023. Accessed: Apr. 17, 2023. [Online]. Available: https://moodle.whitireia.ac.nz/pluginfile.php/1615815/mod_resource/content/1/2023%20MIT%20Handbook.pdf

[6] M. Delakis and C. Garcia, 'Text detection with convolutional neural networks', *In VISAPP*, pp. 290–294, Jan. 2008, Accessed: Mar. 23, 2023. [Online]. Available: http://www.imagemagick.org

[7] H. Xu and F. Su, 'Robust seed localization and growing with deep convolutional features for scene text detection', *ICMR 2015 - Proceedings of the 2015 ACM International*

Conference on Multimedia Retrieval, pp. 387–394, Jun. 2015, doi: 10.1145/2671188.2749370.

[8] W. Huang, Y. Qiao, and X. Tang, 'Robust scene text detection with convolution neural network induced MSER trees', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8692 LNCS, no. PART 4, pp. 497–511, 2014, doi: 10.1007/978-3-319-10593-2_33/COVER.

[9] C. Zhang, C. Yao, B. Shi, and X. Bai, 'Automatic discrimination of text and non-text natural images', *Proceedings of the International Conference on Document Analysis and Recognition, ICDAR*, vol. 2015-November, pp. 886–890, Nov. 2015, doi: 10.1109/ICDAR.2015.7333889.

[10] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, 'Multi-digit number recognition from street view imagery using deep convolutional neural networks', *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, Dec. 2013, Accessed: Mar. 23, 2023. [Online]. Available: https://arxiv.org/abs/1312.6082v4

[11] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, 'Deep structured output learning for unconstrained text recognition', *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, Dec. 2014, Accessed: Mar. 23, 2023. [Online]. Available: https://arxiv.org/abs/1412.5903v5

[12] F. A. Gers, J. Schmidhuber, and F. Cummins, 'Learning to forget: Continual prediction with LSTM', *Neural Comput*, vol. 12, no. 10, pp. 2451–2471, 2000, doi: 10.1162/089976600300015015.

[13] P. He, W. Huang, Y. Qiao, C. C. Loy, and X. Tang, 'Reading scene text in deep convolutional sequences', *Proceedings of the AAAI Conference on Artificial*

Intelligence, vol. 30, no. 1, pp. 3501–3508, Mar. 2016, doi: 10.1609/AAAI.V30I1.10465.

[14] B. Shi, X. Bai, and C. Yao, 'An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition', *IEEE Trans Pattern Anal Mach Intell*, vol. 39, no. 11, pp. 2298–2304, Nov. 2017, doi: 10.1109/TPAMI.2016.2646371.

[15] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, 'End-to-end text recognition with convolutional neural networks', *In Proceedings of the 21st international conference on pattern recognition (ICPR2012), IEEE*, pp. 3304–3308, Nov. 2012, Accessed: Mar. 23, 2023. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6460871?casa_token=XxesCzuwOw8AAAAA:4_QqL7d--4Pn79c_2ShfxwapK3fbXg1QYHh6EihCM_vNMlrYXD69F0WPWXaOSaHCEuar4YlvzPii

[16] M. Jaderberg, A. Vedaldi, and A. Zisserman, 'Deep features for text spotting', *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8692 LNCS, no. PART 4, pp. 512–528, 2014, doi: 10.1007/978-3-319-10593-2_34/COVER.

[17] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, 'Reading text in the wild with convolutional neural networks', *Int J Comput Vis*, vol. 116, no. 1, pp. 1–20, Jan. 2016, doi: 10.1007/S11263-015-0823-Z/FIGURES/13.

[18] 'tika · PyPI'. Accessed: Mar. 30, 2023. [Online]. Available: https://pypi.org/project/tika/

[19] 'Extract text from a PDF — PyPDF2 documentation'. Accessed: Apr. 07, 2023. [Online]. Available: https://pypdf2.readthedocs.io/en/3.0.0/user/extract-text.html

[20] ‘docx2python · PyPI’. Accessed: Apr. 07, 2023. [Online]. Available: https://pypi.org/project/docx2python/

[21] S. Eskenazi, P. Gomez-Krämer, and J. M. Ogier, ‘A comprehensive survey of mostly textual document segmentation algorithms since 2008’, *Pattern Recognit*, vol. 64, pp. 1–14, Apr. 2017, doi: 10.1016/J.PATCOG.2016.10.023.

[22] K. Jung, K. I. Kim, and A. K. Jain, ‘Text information extraction in images and video: a survey’, *Pattern Recognit*, vol. 37, no. 5, pp. 977–997, May 2004, doi: 10.1016/J.PATCOG.2003.10.012.

[23] A. Vinciarelli, ‘A survey on off-line cursive word recognition’, *Pattern Recognit*, vol. 35, no. 7, pp. 1433–1446, Jul. 2002, doi: 10.1016/S0031-3203(01)00129-7.

[24] Y. Y. Tang, S. W. Lee, and C. Y. Suen, ‘Automatic document processing: A survey’, *Pattern Recognit*, vol. 29, no. 12, pp. 1931–1952, Dec. 1996, doi: 10.1016/S0031-3203(96)00044-1.

[25] A. Gonzalez, L. M. Bergasa, and J. J. Yebes, ‘Text detection and recognition on traffic panels from street-level imagery using visual appearance’, *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 1, pp. 228–238, Feb. 2014, doi: 10.1109/TITS.2013.2277662.

[26] C. Indravadanbhai Patel, D. Patel, C. Patel Smt Chandaben Mohanbhai, A. Patel, S. Chandaben Mohanbhai, and D. Patel Smt Chandaben Mohanbhai, ‘Optical character recognition by open source OCR Tool Tesseract: A case study’, *Article in International Journal of Computer Applications*, vol. 55, no. 10, pp. 975–8887, 2012, doi: 10.5120/8794-2784.

[27] S. Manolache, A. Boiangiu, C. Avatavului, and M. Prodan, ‘Combining Tesseract and Asprise results to improve OCR text detection accuracy’, *Article in Journal of*

Information Systems Management, 2019, Accessed: Apr. 07, 2023. [Online]. Available: https://www.researchgate.net/publication/333968802

[28] Z. Xia, W. Zhang, W. Dou, and Z. Weng, 'Research on verification code recognition based on improved CRNN network model', *Lecture Notes on Data Engineering and Communications Technologies*, vol. 88, pp. 1328–1336, 2021, doi: 10.1007/978-3-030-70665-4_145/TABLES/2.

[29] X. Bai, B. Shi, C. Zhang, X. Cai, and L. Qi, 'Text/non-text image classification in the wild with convolutional neural networks', *Pattern Recognit*, vol. 66, pp. 437–446, Jun. 2017, doi: 10.1016/J.PATCOG.2016.12.005.

[30] L. Gomez, A. Nicolaou, and D. Karatzas, 'Improving patch-based scene text script identification with ensembles of conjoined networks', *Pattern Recognit*, vol. 67, pp. 85–96, Jul. 2017, doi: 10.1016/J.PATCOG.2017.01.032.

[31] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, 'End-to-end text recognition with convolutional neural networks', *IEEE*, pp. 3304–3308, Nov. 2012.

[32] H. A. Simon, 'The Sciences of the Artificial, 3rd Edition (Google eBook)', p. 247, 1996, Accessed: Jan. 11, 2024. [Online]. Available: http://books.google.com/books?id=k5Sr0nFw7psC&pgis=1

[33] A. Dresch, D. P. Lacerda, and J. A. V. Antunes, 'Design Science Research', *Design Science Research*, pp. 67–102, 2015, doi: 10.1007/978-3-319-07374-3_4.

[34] Vaishnavi V and Kuechler B, 'Design research in information systems', Dec. 2011, Accessed: Jan. 11, 2024. [Online]. Available: http://desrist.org/design-research-in-information-systems

[35] N. J. Manson, 'Is operations research really research?', *ORiON*, vol. 22, no. 2, pp. 155–180, 2006, Accessed: Jan. 11, 2024. [Online]. Available: https://www.ajol.info/index.php/orion/article/view/34262

[36] H. C. Shin *et al.*, ‘Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning’, *IEEE Trans Med Imaging*, vol. 35, no. 5, pp. 1285–1298, May 2016, doi: 10.1109/TMI.2016.2528162.

[37] N. Jatana, S. Puri, M. Ahuja, I. Kathuria, and D. Gosain, ‘A Survey and Comparison of Relational and Non-Relational Database’, Accessed: Feb. 10, 2024. [Online]. Available: www.ijert.org

[38] M. Gabryel, ‘The bag-of-features algorithm for practical applications using the MySQL database’, *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9693, pp. 635–646, 2016, doi: 10.1007/978-3-319-39384-1_56/COVER.

[39] S. A. Jowan, R. Faraj Swese, A. Yousf Aldabrzi, and M. Saad Shertil, ‘TRADITIONAL RDBMS TO NOSQL DATABASE: NEW ERA OF DATABASES FOR BIG DATA’, *Journal of Humanities and Applied Science*, no. 29, 2016, Accessed: Feb. 10, 2024. [Online]. Available: https://www.researchgate.net/publication/355165835

[40] ‘db4free.net - MySQL Database for free’. Accessed: Jan. 13, 2024. [Online]. Available: https://db4free.net/

[41] ‘Welcome to Colaboratory - Colaboratory’. Accessed: Jan. 13, 2024. [Online]. Available: https://colab.research.google.com/

[42] ‘Design elements - ER diagram (Chen notation) | Chen Notation | ERD, Chen’s notation - Vector stencils library | Chen Notation’. Accessed: Mar. 31, 2023. [Online]. Available: https://www.conceptdraw.com/examples/chen-notation

[43] ‘Apache Tika – Apache Tika’. Accessed: Feb. 10, 2024. [Online]. Available: https://tika.apache.org/

[44] ‘Welcome to PyPDF2 — PyPDF2 documentation’. Accessed: Feb. 10, 2024. [Online]. Available: https://pypdf2.readthedocs.io/en/3.0.0/

[45] ‘docx2python · PyPI’. Accessed: Feb. 10, 2024. [Online]. Available: https://pypi.org/project/docx2python/

[46] J. A. Bakar, K. Omar, M. F. Nasrudin, and M. Z. Murah, ‘Tokenizer for the Malay language using pattern matching’, *International Conference on Intelligent Systems Design and Applications, ISDA*, vol. 2015-January, pp. 140–144, Mar. 2015, doi: 10.1109/ISDA.2014.7066258.

[47] G. Kaur, ‘USAGE OF REGULAR EXPRESSIONS IN NLP’, *IJRET: International Journal of Research in Engineering and Technology*, pp. 2321–7308, Accessed: Feb. 10, 2024. [Online]. Available: http://www.ijret.org

[48] ‘Module fitz - PyMuPDF 1.23.21 documentation’. Accessed: Feb. 10, 2024. [Online]. Available: https://pymupdf.readthedocs.io/en/latest/module.html

[49] ‘OpenCV: Geometric Transformations of Images’. Accessed: Feb. 10, 2024. [Online]. Available: https://docs.opencv.org/3.4/da/d6e/tutorial_py_geometric_transformations.html

[50] Y. Huo *et al.*, ‘Grid Row Parameter Identification Using Tensorflow Convolutional Neural Network’, *2021 15th IEEE International Conference on Electronic Measurement and Instruments, ICEMI 2021*, pp. 217–221, 2021, doi: 10.1109/ICEMI52946.2021.9679668.

[51] A. Mordvintsev, ‘OpenCV-Python Tutorials Documentation Release 1’, 2017.

[52] K. Saddami, K. Munadi, Y. Away, and F. Arnia, ‘Improvement of binarization performance using local otsu thresholding’, *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, no. 1, pp. 264–272, 2019, doi: 10.11591/ijece.v9i1.pp264-272.

[53] B. Gatos, I. Pratikakis, and S. J. Perantonis, ‘An adaptive binarization technique for low quality historical documents’, *Lecture Notes in Computer Science (including subseries*

Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 3163, pp. 102–113, 2004, doi: 10.1007/978-3-540-28640-0_10/COVER.

[54] T. R. Singh, S. Roy, O. I. Singh, T. Sinam, and Kh. M. Singh, 'A New Local Adaptive Thresholding Technique in Binarization', Jan. 2012, Accessed: Feb. 10, 2024. [Online]. Available: https://arxiv.org/abs/1201.5227v1

[55] K. Lin, T. H. Li, S. Liu, and G. Li, 'Real photographs denoising with noise domain adaptation and attentive generative adversarial network', *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, vol. 2019-June, pp. 1717–1721, Jun. 2019, doi: 10.1109/CVPRW.2019.00221.

[56] 'Pre-Processing in OCR!!!. A basic explanation of the most widely… | by Susmith Reddy | Towards Data Science'. Accessed: Feb. 11, 2024. [Online]. Available: https://towardsdatascience.com/pre-processing-in-ocr-fc231c6035a7

[57] 'Noisy and Rotated Scanned Documents | Kaggle'. Accessed: Sep. 14, 2023. [Online]. Available: https://www.kaggle.com/datasets/sthabile/noisy-and-rotated-scanned-documents

[58] J. Tong, H. Shi, C. Wu, H. Jiang, and T. Yang, 'Skewness correction and quality evaluation of plug seedling images based on Canny operator and Hough transform', *Comput Electron Agric*, vol. 155, pp. 461–472, Dec. 2018, doi: 10.1016/J.COMPAG.2018.10.035.

[59] A. Amin and S. Fischer, 'A document skew detection method using the Hough transform', *Pattern Analysis and Applications*, vol. 3, no. 3, pp. 243–253, 2000, doi: 10.1007/S100440070009/METRICS.

[60] C. Singh, N. Bhatia, and A. Kaur, 'Hough transform based fast skew detection and accurate skew correction methods', *Pattern Recognit*, vol. 41, no. 12, pp. 3528–3546, Dec. 2008, doi: 10.1016/J.PATCOG.2008.06.002.

[61] W. Rong, Z. Li, W. Zhang, and L. Sun, ‘An improved Canny edge detection algorithm’, *2014 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2014*, pp. 577–582, 2014, doi: 10.1109/ICMA.2014.6885761.

[62] ‘(5) How do I detect a rotated image and fix it back to its proper position using Python/OpenCV? - Quora’. Accessed: Oct. 19, 2023. [Online]. Available: https://www.quora.com/How-do-I-detect-a-rotated-image-and-fix-it-back-to-its-proper-position-using-Python-OpenCV

[63] P. Heckbert, ‘Fourier Transforms and the Fast Fourier Transform (FFT) Algorithm’, *Comput Graph (ACM)*, vol. 3, pp. 15–463, 1995.

[64] H. L. Buijs, A. Pomerleau, M. Fournier, and W. G. Tam, ‘Implementation of a Fast Fourier Transform (FFT) for Image Processing Applications’, *IEEE Trans Acoust*, vol. 22, no. 6, pp. 420–424, 1974, doi: 10.1109/TASSP.1974.1162620.

[65] V. Nair, M. Chatterjee, N. Tavakoli, A. S. Namin, and C. Snoeyink, ‘Fast Fourier Transformation for Optimizing Convolutional Neural Networks in Object Recognition’, Oct. 2020, Accessed: Feb. 11, 2024. [Online]. Available: https://arxiv.org/abs/2010.04257v1

[66] ‘OpenCV: Discrete Fourier Transform’. Accessed: Feb. 11, 2024. [Online]. Available: https://docs.opencv.org/3.4/d8/d01/tutorial_discrete_fourier_transform.html

[67] ‘sklearn.model_selection.train_test_split — scikit-learn 1.4.0 documentation’. Accessed: Feb. 11, 2024. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[68] A. Botalb, M. Moinuddin, U. M. Al-Saggaf, and S. S. A. Ali, ‘Contrasting Convolutional Neural Network (CNN) with Multi-Layer Perceptron (MLP) for Big Data Analysis’, *International Conference on Intelligent and Advanced System, ICIAS 2018*, Nov. 2018, doi: 10.1109/ICIAS.2018.8540626.

[69] D. Chudasama, T. Patel, S. Joshi, and G. I. Prajapati, ‘Image Segmentation using Morphological Operations’, *Int J Comput Appl*, vol. 117, no. 18, pp. 975–8887, 2015.

[70] T. A. Vo-Nguyen, P. Nguyen, and H. S. Le, ‘An Efficient Method to Extract Data from Bank Statements Based on Image-Based Table Detection’, *Proceedings - 2021 15th International Conference on Advanced Computing and Applications, ACOMP 2021*, pp. 186–190, 2021, doi: 10.1109/ACOMP53746.2021.00033.

[71] ‘OpenCV: Finding contours in your image’. Accessed: Feb. 11, 2024. [Online]. Available: https://docs.opencv.org/3.4/df/d0d/tutorial_find_contours.html

[72] V. More, M. Kharat, and S. Gumaste, ‘Segmentation Of Devanagari Handwritten Text Using Thresholding Approach’, Accessed: Feb. 11, 2024. [Online]. Available: www.ijstr.org

[73] H. Eldem, E. Ülker, and O. Y. Işıklı, ‘Alexnet architecture variations with transfer learning for classification of wound images’, *Engineering Science and Technology, an International Journal*, vol. 45, p. 101490, Sep. 2023, doi: 10.1016/J.JESTCH.2023.101490.

[74] B. P. Sowmya and M. C. Supriya, ‘Convolutional Neural Network (CNN) Fundamental Operational Survey’, *Learning and Analytics in Intelligent Systems*, vol. 21, pp. 245–258, 2021, doi: 10.1007/978-3-030-65407-8_21/COVER.

[75] A. Sherstinsky, ‘Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network’, *Physica D*, vol. 404, p. 132306, Mar. 2020, doi: 10.1016/J.PHYSD.2019.132306.

[76] Y. Yu, X. Si, C. Hu, and J. Zhang, ‘A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures’, *Neural Comput*, vol. 31, no. 7, pp. 1235–1270, Jul. 2019, doi: 10.1162/NECO_A_01199.

[77] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, ‘Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks’, *ACM International Conference Proceeding Series*, vol. 148, pp. 369–376, 2006, doi: 10.1145/1143844.1143891.

[78] ‘Visual Geometry Group - University of Oxford’. Accessed: Jan. 17, 2024. [Online]. Available: https://www.robots.ox.ac.uk/~vgg/data/text/#sec-synth

[79] K. Wadhawan and E. Gajendran, ‘Automated recognition of text in images: A survey’, *Int J Comput Appl*, vol. 127, no. 15, pp. 975–8887, 2015.

[80] ‘difflib — Helpers for computing deltas — Python 3.12.1 documentation’. Accessed: Jan. 18, 2024. [Online]. Available: https://docs.python.org/3/library/difflib.html

[81] A. Nagaraj and M. Kejriwal, ‘Robust Quantification of Gender Disparity in Pre-Modern English Literature using Natural Language Processing’, Apr. 2022, Accessed: Jan. 18, 2024. [Online]. Available: https://arxiv.org/abs/2204.05872v1