National College of Ireland

Detecting and Classifying Post-OCR Errors Using Contrastive Self-Supervised Learning

MSc Research Practicum
MSc in Artificial Intelligence

Gokul Krishna
Student ID: X23350512

School of Computing
National College of Ireland

Supervisor: Abdul Razzaq

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Gokul Krishna.		
Student ID:	X23350512		
Programme:	MSc Artificial Intelligence	**Year:**	2024-2025
Module:	MSc in Research Practicum		
Supervisor:	Abdul Razzaq		
Submission Due Date:	11/08/2025		
Project Title:	Detecting and Classifying Post-OCR Errors Using Contrastive Self-Supervised Learning		
Word Count:	**7677**	**Page Count: 25**	

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Gokul Krishna

Date: 10th August 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Detecting and Classifying Post-OCR Errors Using Contrastive Self-Supervised Learning

GokulKrishna.
X23350512

Abstract

This research addresses the major challenge of post-Optical Character Recognition (OCR) error detection and classification. This project introduces approach leveraging contrastive self-supervised learning, integrating both textual and visual information. Using BERT for robust text embeddings and ResNet for image feature extraction, the system effectively identifies and categorizes OCR error types. A fundamental step of this project is generating synthetic errors using real-world datasets like CORD and FUNSD, which allows a supervised learning framework for error classification. The model classifies errors into several types: missing words, correct Word, spelling mistakes, and wrong words, based on calculated similarity scores. This combined approach of classification and contrastive loss optimizes the model's ability to learn difference between embeddings. The outcomes of the comparison after the evaluation indicate that the model achieved an of accuracy by 73% on FUNSD and 74% on CORD, especially regarding performance of detection of missing words. Although the system demonstrates good results with structured documents, there is more that can improve its capabilities in dealing with such fine-grained spelling mistakes as well as complicated layouts. The work is of great contribution in the strong OCR post-correction and is the precursor to more trustworthy text digitization.

1 Introduction

Optical Character Recognition (OCR) technology has changed the world of how textual information is digitized and how images of such documents become searchable and editable in a digital form. This development has significant and far-reaching consequences in most areas, including the preservation of the nation's cultural heritage and historical records as well as their easy access through simplification of the business processes and greater accessibility to visually challenged individuals. Even with the remarkable progress in the abilities of OCR and especially the introduction of deep learning, technology does not come without its drawbacks. OCR designs often become problematic in handling poor quality documents, non-standardized formats, mixed fonts or texts of low-resource languages. These issues tend to create challenges of recognition errors which, when not addressed, may continue to undermine a series of natural language processing (NLP) functions, thereafter, skewing the purity and use of resulting extracted information [Lopresti, 2008].

With the existence of errors in OCR output, there is a need to robust mechanisms of detection and correction. This is the most important step which is in some cases known as post-OCR correction that unlocks all the potential in the digitized texts. Proper after-OCR cleaning will provide those digital representations close to the original text, because of which it will be possible to successfully retrieve information, mine the text, and analyze the language. The nature of these mistakes between the basic types of character replacement to more

complicated cases of word segmentation problems require complex solutions that may capitalize on contextual knowledge and visual clues to correct the inaccuracy [Guan and Greene, 2024; Hemmer et al., 2024].

In recent years, the domain of post-OCR correction has undergone a paradigm shift with the incorporation of strong machine learning and deep learning methods. Remarkably, transformer models like Bidirectional Encoder Representations of Transformers (BERT), became a potent instrument in the work with sequential data, including textual one. Their capacity to render complex contextual relations in language renders them very adaptable in the detection as well as correction of the linguistics errors in OCR output. At the same time, computer-vision-based improvements, including models such as the Residual Network (ResNet), have also offered powerful methods of analyzing visual data, which is essential to discerning the visual attributes of characters and document arrangements that affect the robustness of OCR [Appalaraju et al., 2021; Chen and Zhou].
The project turned out to be rooted in investigating the possibilities of improving the performance of OCR by combining the best of both worlds, i.e., natural language processing and computer vision. This research is intended to build a complex model that enhances precision and stability of OCR performance by resourcing deep learning models (BERT and ResNet) in the context of their strengths and weaknesses. The suggested methodology overcomes the issue of the OCR inaccuracies inherent to such procedure as it involves textual analysis complemented with visual information hence contributes to the creation of more accurate solutions towards digitizing and processing textual documents of various kinds.

Research Question: How effectively can post-OCR errors be detected and classified using contrastive self-supervised learning methods that integrate token-level OCR confidence scores and grid-based alignment techniques, while relying solely on lightweight, low-resource models suitable for basic hardware?

2 Related Work

Optical Character Recognition (OCR) is a technology increasingly used to convert files scanned, historical collections, and documents printed, in a text format that enables the subsequent large-scale text processing. Even with the development of improved recognition algorithms, OCR performance is still very sensitive to document quality, complexity of layout and variety of scripts, which often causes errors that lead to serious degradation of text usability. Such errors not only decrease the quality of the actual digitized material but may also snowball into other operations within the Natural Language Processing (NLP), which include information retrieval, machine translation, and named entity recognition. This has led to the development of strong post-OCR error detection and correction approaches, as an important research direction, with approaches shifting away toward lexical rules-based methods to more state-of-art deep learning and multimodal models. These developments are reviewed in detail in the following sections and the important strategies, models, and datasets that have advanced developments in OCR error mitigation are discussed.

The Challenge of OCR Errors and Their Impact

Although there are more efforts to improve OCR systems, OCR technology itself is widely associated with error susceptibility, particularly in the face of poor document quality, complex layouts, uncommon fonts and languages with insufficient resources. Such errors

may propagate to subsequent Natural Language Processing (NLP) tasks and reduce the value of the digitized texts. D. Lopresti (2009) gives a very detailed description of the nature of the transmission of OCR errors across different levels of the NLP including sentence boundary detection, tokenization, and part-of-speech tagging, and how this necessitates the counteractive error removal techniques. The categories of errors may differ, ranging between character substitutions, insertions, deletion, and segmentation problems, and are very common in historical texts as they have orthographic differences and special types L. Lyu, M. Koutraki, M. Krickl, and B. Fetahu (2021) This is further complicated by the high rates of errors seen in historical texts which in many cases are due to unseen characters S. M. Virk, D. Dannélls, and A. S. Muhammad (2021)

Evolution of Post-OCR Correction Techniques

Initial strategies dealing with post-OCR correction mostly employed the use of lexical methods including the use of dictionaries and heuristic rules that were used in correcting errors at character and word levels. S. Verberne, P. van der Weide, and L. van der Meer (2012) considered the application of search suggestions to context-sensitive corrections and others emphasized the conciliatory dictionaries requirement in the domain specific texts. D. Lopresti (2009) aimed their research towards matching and combining the outputs of several scans with the intention of correcting errors. But these rule-based and lexical techniques tended not to work well on the variety and complexity of OCR errors in the wild, especially in the era of deep learning, superior and flexible solutions emerged.

Deep Learning and Transformer Models in Post-OCR

The paradigm changed dramatically following the usage of machine learning and deep learning, in post-OCR workflows. A significant number of the research redefined the problem of post-OCR correction as a sequence to sequence one, and the strength of the neural networks was utilized. As an example, B. Makovitch and M. Wolska (2020) tested Neural Machine Translation (NMT) models in that regard. A breakthrough was the introduction of the pre-trained language models to which BERT (Bidirectional Encoder Representations from Transformers) Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou (2020) was the key. Previous works L. Lyu, M. Koutraki, M. Krickl, and B. Fetahu (2021) and B. Makovitch and M. Wolska (2020) have used BERT and BART models, respectively, and they are effective in post-OCR correction. L. Lyu, M. Koutraki, M. Krickl, and B. Fetahu (2021) also showed that ByT5 is superior in this domain in comparison with original sequence-to-sequence models, yet it was not trained within this domain Recently, A. Hemmer, M. Coustaty (2024) improved the OCR confidence information into token embeddings of another BERT-based model to have better error detection, indicating the performance differences between commercial and publicly available OCR systems. pre-trained transformer models (mBART, mT5, IndicBART) were also used by H. Kashid and P. Bhattacharyya (2024) in the post-OCR error correction of low resource Devanagari languages whose OCR errors were treated as mistranslations.

Integration of Visual Information and Multi-modal Approaches

In addition to text-only solutions, visual information uses have proven extremely useful in issues of overall document comprehension and OCR mistake correction. Other models, such as DocFormer introduced in S. Appalaraju *et al.*, "DocFormer (2021) are multi-modal transformer-based architectures which learn to combine features of text, vision, and space using new self-attention mechanism Visual Document Understanding (VDU) S. Appalaraju *et al.*, "DocFormer (2021). This method accepts that some of the important context that text cannot capture is given through visual information (text font, layout, and shapes of the characters (glyphs)). This was pursued further in Y.-H. Chen and Y. Zhou (2023) to come up with a model post-OCR correction that relies on character embedding and a novel glyph embedding developed. Their collaboration shows that the possibility of integrating visually distinctive features of characters via glyph embedding noticeably enhances the correction process that even fills the gaps in low-quality OCR models. They used ResNet18 and ResNet50 in training their glyph embedding proving usefulness of convolutional neural networks in the extraction of pertinent visual features to this task. Also, the notion of glyph similarity, which is discussed by H. Kashid and P. Bhattacharyya (2024), has paid significant attention to the importance of the visual features, offering a new approach to the creation of synthetic data using computer vision feature detection (e.g., SIFT, ORB, AKAZE) to synthesize convincing OCR errors to be used in the training process. Negative A. Aberdam *et al* (2021) developments also came into place with Sequence-to-Sequence Contrastive Learning (SeqCLR) on visual representations in text recognition, where CNNs are used to feature extract, and BiLSTM are used to model sequences further established computer vision enhancing text recognition and correction.

Synthetic Data Generation for Post-OCR

There is increasing interest in synthetic data generation because of the limited availability of high-volume, manually corrected data on which to train a post-OCR correction system, particularly formats dealing with historical documents and/or low-resource languages. This is the process of artificially corrupting data with clean text base to form training pairs. Noise injection is one of the traditional techniques that involve the random addition of errors to clean text. H. Kashid and P. Bhattacharyya (2024) describes in detail different synthetic data generation techniques, such as the creation of images, where clean text is converted into images, mangled to enhance OCR-like noise, and then pumped through an OCR engine to yield erroneous results. Their new approach to glyph similarity also enhances synthetic data generation process further because added errors reflect the real-world OCR errors to some extent according to visual similarity of characters. In a similar vein, H. Kashid and P. Bhattacharyya (2024) proposed RoundTripOCR, a method that emulates OCR errors to produce clean text and, thus, synthetic datasets on low-resource Devanagari languages and addressed correction as a machine translation problem.

Error Detection and Confidence Scores

The success of correction after OCR relates to the correctness with which the errors are detected. The earlier approaches usually resorted to dictionary search and n-grams of

characters. It was shown in S. M. Virk, D. Dannélls, and A. S. Muhammad (2021) that character n-gram statistics are a viable single-feature mechanism to detect errors in historical documents. Higher-end methods, including those of A. Hemmer, M. Coustaty (2024), incorporate OCR confidence scores, which give an indication of the confidence of the OCR system in the recognition, to aid detection

3 Research Methodology

The model which proposed for detecting and classifying the post-OCR errors has various stages involved, each contributing to the overall objective of the project. From raw data to actionable insight and error classification, this model has been methodical created.
The following methodology stages:

Dataset preparation: This was the initial stage of every project to select real-world data which has been used in the ConfidenceAware Document OCR Error Detection (ConfBERT) model paper with two different datasets are being included. The preparation of data includes the data cleaning, data balancing every single part. But in this case of that there is no chance of doing all this steps, as it was real world data If I follow the data preparation steps the results will not come as expected so I have built the system according to this data set so that the results which are presented by the system will is strong enough to show. The datasets use in this project are **CORD (Collection of Receipts Dataset)** and **FUNSD (Form Understanding in Noisy Scanned Documents)** datasets as mentioned extracted from the research paper who already worked related to existing problem shown in the research question, but the problem chosen was the upgraded version of the pervious problem

Simulated OCR Error Generation: Using the real-world data system has generates csv file with following column names of [“image_id”, “gt_word”, “ocr_word”, “similarity”, “error_type”] so that system can specify the error type using the similarity score which was used to identity the error type makes the system easy to understand. This was the method chosen according to the problem statement and dataset, based on both a small function architecture has been built according to the system architecture how the system is working based on that this will be processed.

Error Type Classification: A major step in this project that allows the supervised learning for error classification by the methodically classifying each OCR results. This OCR words are generated by the system using the ground truth text will be calculated the similarity score and used to identify the error type using the Machine learning and Deep learning models (BERT). Using this error types of the system will start train by itself using embedding and constrictive pairs. **classify_error**() using this function the system will classify the error type in this function the system will use similarity score so find the error type by calling the similarity function, matches using the **SequenceMatcher()** function like is the **classify_error()** predicting the error type correctly or not.

Text and Image Embeddings Generation: This was the stage which mainly focused on the raw data into high-dimensional embeddings using the advance deep learning models which are capturing both semantic and visual contexts of the raw data. In this the system generated embedding are used for the Contrastive self-supervised Learning model there are various ways of approach to train Contrastive self-supervised Learning but in this system architecture this way was chosen because the approach of this project is not to use advance models or architectures, even low level models can also perform the same work with different approaches and pipelines.

Contrastive Self-Supervised Learning: The core of the project, which is the learning framework, where the model learns how to differentiate between correct and OCR-Errors by optimizing a contrastive loss function using cosine loss function, thereby used for increases the quality of the learning embeddings, There are several approaches to make work with this particular problem, this was one of them among which work with same functionality but different approach of analysing the result and the problem

Evaluation and Analysis: The final stage which involves a rigorous evaluation of the system's performance using various metrics to validate the effectiveness in detecting and classifying all the types of OCR errors. These results cannot compare with any other because pervious papers are mainly focusing on rectifying the OCR errors, but this system was able to identify what type of error was occurring. So, it's completely based on the accuracy score how perfect the system is predicting the error type and analysing them.

DATASET PREPARATION

Dataset used

For the evaluation and development of the model which detects and classifies the post-OCR error, two major distinct real-world document OCR datasets, **CORD** (Collection of Receipts Dataset) and **FUNSD** (Form Understanding in Noisy Scanned Documents) were used. The selections of two datasets were driven by their diverse characteristics of each dataset, and this dataset was used in one of the journal papers which has been published and approved
The dataset used is imbalanced. Both datasets have some common features that make them to select for this project and explain in the below.

CORD (Collection of Receipts Dataset): CORD is like the above receipts, but this data has higher resolution in the hierarchical organization of the annotations, with more context information. The given dataset enables one to investigate the problem of error detection in situations where spatial and semantic connections between the elements of the text are essential. This dataset contains image and annotations folders, where each image has its own annotations that represent every text of the image are considered as box and the edge of the box is valued of the text, so this approach is considered as grid-based approach. Each text has its own unique id and label where labels mention what it was like question, answer or others. So, using this we can say what the next word is to the present word. This dataset can also call a question and answer-based OCR approach.

FUNSD (Form Understanding in Noisy Scanned Documents): The idea of FUNSD is on how to understand forms in scanned documents which are usually heavily noisy and varying. This dataset plays a very important role in solving the problem of layout analysis and text recognition on documents that are less organized or highly degraded. This dataset is also similar to the above dataset but there are difference that makes the approach of this dataset completely different in this also few same like box but in this dataset its named as quad which means the box values are not considered with edges they are considered with coordinate pairs where each edge has one pair of values using those value the text is recognized after system extract the text form image. This data also has two folders with image and JSON (same as annotations) where JSON contains id, quad, text and category image and JSON folders are connected using the row_id which present in JSON file.
These are the two data sets used in this project to detect and classify the post OCR Errors, enhancing its ability to perform effectively across different real-world data. The major part was data preparation, it was very difficult to understand and handle those datasets because even though both datasets are similar, their labels are different, but the intension was that system needs to identify any type of dataset that belongs to the OCR. There should be characteristics of the data that are mentioned below so that the system can be able to work with those data.
Data set should contain 2 folders, image and annotations so in annotations there should be box which contains either edges of the box { box": [61, 127, 143, 211] } or box coordinate "box": { "x2": 252,"y3": 522,"x3": 252,"y4": 522, "x1": 240,"y1": 504,"x4": 240,"y2": 504} with respective text.

Data Structure

Each dataset is structured to provide both image and textual information, making it well-suited but also challenging for a multimodal approach to error detection.

Image files: These are the raw scanned document images are in the formats of PNG or JPG depending on the dataset. These images are served as the visual input for image embeddings generation process, which provides the textual information about the appearance of the text.

Annotations files: Apart from image there are also annotations which are accompanied by each image, which are stored in .json file or in .txt formats. This file contains the ground truth information which is meticulously used to extract the gt_word (ground-truth word). The gt_word represents the correct word, error-free text.

So, this gt_word are used to mimic the real OCR errors using Simulated OCR error Generation technique to simulate what could go wrong in OCR. So, this lets my model to learn how to detect and identify those errors.

Simulated OCR Error Generation

This was one of my methods to came up with where to overcome the scarcity of the real-world structed OCR errors, where a robust simulation method is implemented to generate

realistic error types form the clean ground truth text. System first take the input of the ground truth text and the using the Simulated OCR Error Generation technique it will generate random OCR word using System has created a csv file to store all the ocr-words which are created using the ground truth text, with the column names *image_id, gt_word, ocr_word, similarity, errortype*. The orc error was created automatically using easyocr method where it will extract the word form image and using if and for loop then it was compared with ground truth word in annotations and generate ocr words.

Simulation method

The simulation of OCR errors using the custom function, simulate_ocr, which generates random single-character typos into the ground truth words. This method is used to generate a common OCR incorrect word with minimal changes in the ground truth word like substation, deletion, insertion which are stored in an CSV file with the same dataset name. In the figure 1 you can see how the OCR_word is generated using the ground truth text and compared with the image text which are generated from using EasyOCR by cropping the image and analysing the image.

This function considers the text (gt_word) form the annotation folder as input. It fists checks the length of the text, if the length is greater than 1 then it will generate the ocr_word because if the text is single character the three will be not much difference in the changing that single character. Later, a random index within the word's length is selected, replaces with random character from predefined set, it may be substation, deletion, insertion. This process makes all the text into ocr word by making in the changes in the text.

The **difflib.SequenceMatcher().ratio()** method calculates the similarity score between the ground truth word and ocr word based on the longest common subsequence algorithm

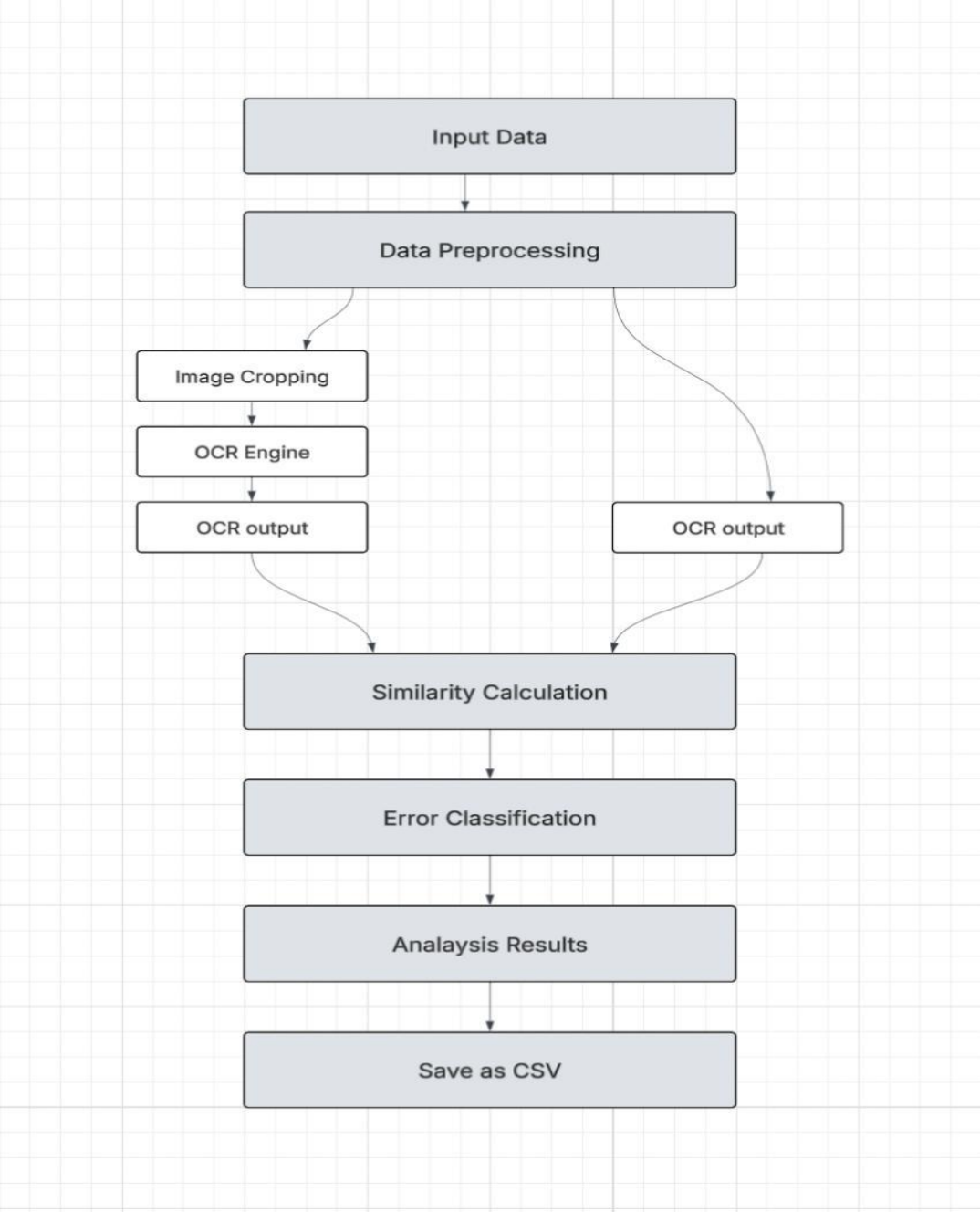


Figure 1: OCR_word generation

Projection Head

To enable the contrastive learning and comparison between the image and text with different way ProjectionHead model is used. This are the simple multi-layer perceptron (MLPs) that project the high-dimensional embeddings form BERT (768 dimensions) and ResNet (512 dimensions) in a common, 128 dimensions.

Contrastive Learning Framework

A major part of the system is to learn mechanism, implementation through a contrastive loss function that operates on pairs of embeddings. The system creates positive and negative pairs. The contrastive loss will pull the positive pairs closer to the embeddings space while dealing with the negative pairs.

Error Detection and Classification model

The system implements will comprehensive error classification framework using the function classify_error and categorizes into four distinct types missing word, correct, spelling mistake, wrong word.

OCR integration and processing

The system integrates multiple OCR engines for robustness and flexibility by using few models, Tesseract OCR which character recognition with confidence score, EasyOCR is used for comparison and validation, Confidence score Integration used the token-level confidence score by using OCR engines to error detection accuracy.

Hardware Optimization and Resource Management

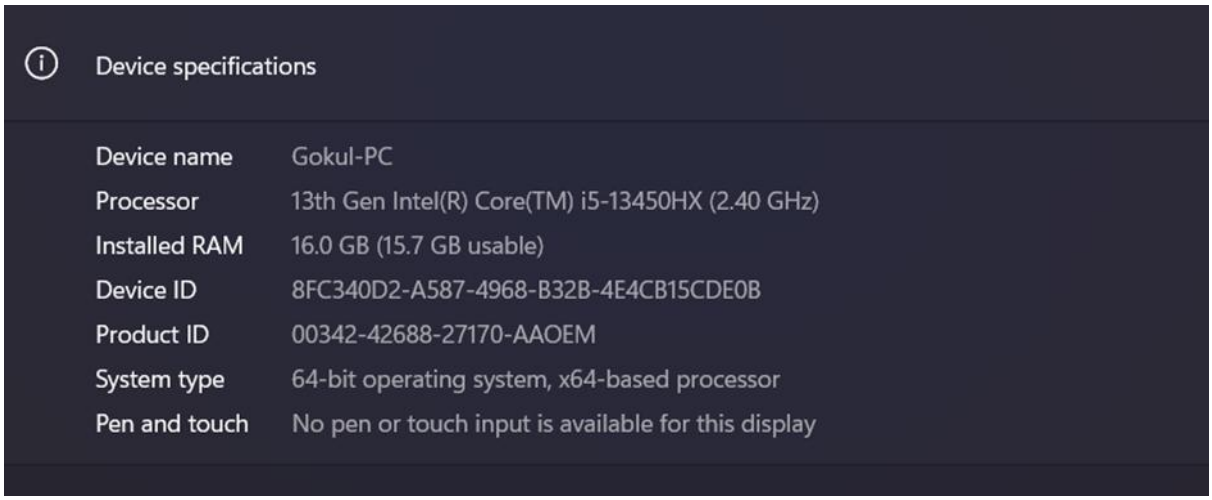


Figure 2: Device Specification

4 Design Specification

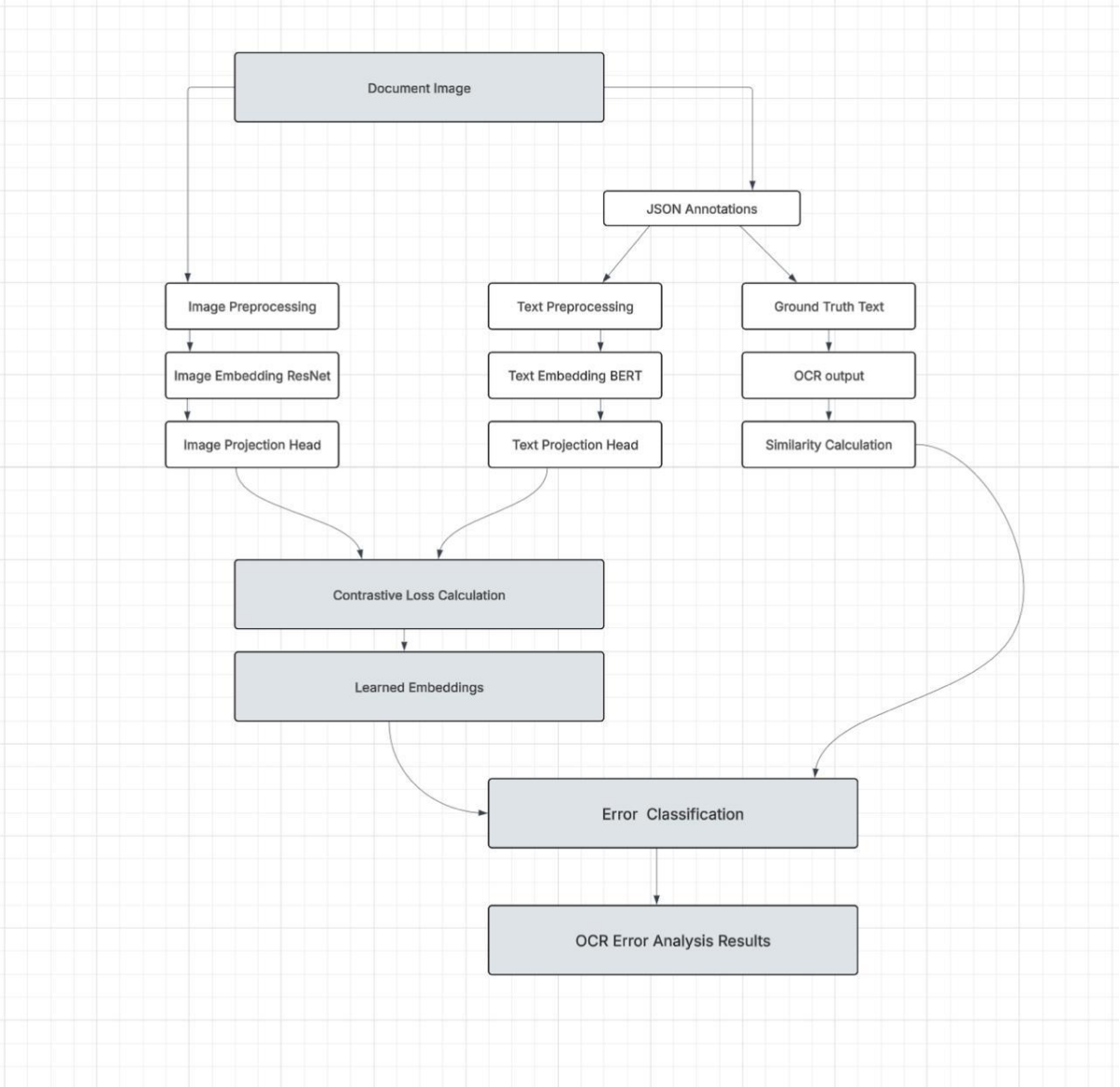


Figure 3: Overall System architecture

Global Configuration sets up the device (CPU/GPU) and installing OCR readers (Tesseract, EasyOCR). In the above figure 3, the whole system architecture has been designed now let's talk about all the terms in the figure 3. The OCR error detection and classification system follows the above pipeline. The core idea was to generate and ocr error from the ground truth and then use them to identify the error and classify errors. The system is designed and suitable for basic hardware using lightweight leveraging pre-trained models.

Data loading and preprocessing

The system will first start with loading the input data, which can be in form of images with corresponding JSON annotation files. The JSON annotation file contains ground truth text

and bounding box information for the words. The system will read image files with respective JSON annotation file form specified directions and check whether the annotation matches the image with image id. Then from the JSON file it will extract the ground truth word which was compared with the ocr word. For each image the system will prepare it for OCR processing, So the image will be opened and converting it to a suitable format.

System architecture

Following, the system will load the datasets form the folder where there are two different dataset with different features and data values so based on they features both the dataset loaded with respective function names **load_cord_data(json_dir, image_dir)** is used to load the CORD dataset, **load_funsd_data()** is used to load the FUNSD dataset later system generates OCR words using thg ground truth word from the respective datasets where are loaded and the using the OCREasy BERT model system will extract the text from the image and compares with the ground truth text in annotations file the it will generate the OCR word and also identify the error type and also using the similarity function it will calculate the similarity score like how similar is the word. Later on the system will generate embedding using **get_text_embedding()** function for the text, **get_image_embedding()** function is used to generate embedding for image using tourch.Tensor where those embedding are used for Contrastive self-training and constrictive pairs like both positive and negative pairs are been build using **build_pairs()** from the dataset so then this pairs also been used for training the self-supervised learning.

The system will also use the constrictive loss **contrastive_loss()** function to get trained. The loss function used in this project is cosine loss function, **cosine_similarity(emb1, emb2, dim=0)** you can see in this function both the embeddings are being compared by checking similarity.

ProjectionHead() in this system it was not the prebuilt, the projection head used in this system is customized where the number of layers is structured based on how the system should work.

```
import torch.nn as nn

class ProjectionHead(nn.Module):
    def __init__(self, input_dim=768, output_dim=128):
        super(ProjectionHead, self).__init__()
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(inplace=True),
            nn.Linear(256, output_dim),
            nn.LayerNorm(output_dim),
            nn.Tanh()
        )

    def forward(self, x):
        return self.mlp(x)
```

It is multi layered where two linear layers are present where one will map the input dimension to the intermediate layer and another will map form intermediate layer to output dimension. ReLu (Rectified Linear Unit) is the activation function, which is applied after the first layer, follows with the nn.layerNorm which is the layer Normalization which is applied to the output layer.

```
class ProjectionHead(nn.Module):
    def __init__(self, input_dim, output_dim=128):
        super(ProjectionHead, self).__init__()
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(),
            nn.Linear(256, output_dim)
        )

    def forward(self, x):
        return self.mlp(x)

def contrastive_loss(z1, z2, temperature=0.2):
    batch_size = z1.size(0)
    z1 = F.normalize(z1, dim=1)
    z2 = F.normalize(z2, dim=1)
    representations = torch.cat([z1, z2], dim=0)
    sim_matrix = torch.matmul(representations, representations.T)
    sim_matrix = sim_matrix / temperature
    mask = torch.eye(2 * batch_size, device=z1.device).bool()
    sim_matrix.masked_fill_(mask, -1e9)
    targets = torch.cat([
        torch.arange(batch_size, batch_size * 2),
        torch.arange(0, batch_size)
    ], dim=0).to(z1.device)
    loss = F.cross_entropy(sim_matrix, targets)
    return loss
```

The above figure shows how the model is defined and setup using the projection head, and you can also see the way it was self-training using the contrastive loss, so the loss function used in this was cosine loss function. This was the complete set up of the system architecture according to the model structure present.

Once both the OCR text and the ground truth are available, system will calculate the similarity score and how strong the error type was calculated and identify, stored in CSV file. System has classified errors in four type Missing word, Correct, Spelling mistake, Wrong word. System will process the image and based on the image document it will extract the text via OCR using EasyOCR and compare it against the ground truth then pass to the CSV file so once the system has generated the ground truth then using the ground truth text will start generating OCR word and will store in same csv file.

System when it completed generating all the ground truth text and OCR text then it will start checking the similarity score using **compute_similarity()** function and it will start checking how similar the word was and what was the error type it was using the **ErrorClassifier()** function, stores in the same csv file. The csv file will generate according to the respective dataset. The system ius compared with various interconnected models, every model which I

have referred was responsible for specific set of functionalities, few design architecture can only run in high GUPs and CUPs. This modular is designed which promotes reusability, scalability and maintainability.

5 Implementation

This section helps how the system is built in above section there was complete explanation about the system architecture how it was been setup now let's go with how the system is implemented and why only this specific method is used and how efficient this method was for the problem mentioned. Data setup was the initial stage of implementation **load_cord_data(), load_funsd_data()** are the functions used to load the data set and also shows what are the feature of these datasets are and how can this datasets been handled because both the dataset have different features and values so as per dataset the model is build which can handle both the datasets.

Data handling

Handling the data is not an easy task where when the task is about to handle the real-world data we can't we make any changes in the dataset we should work with the existing data present. So, basically in those two data set they conation several section like box, text, id and few others but not all the section are same they are completely different because the way I selected the dataset must contain the box values.

Here the box is nothing, but I am considering the image in grid based for every box they should be 4 values but here I got up with another problem why only 4 values what if they have coordinates pairs for each box can we really handle those data?
This made me to think about the type of data like will there be any dataset which match's my idea so then I got his COAD dataset form a same paper which I am referring A. Hemmer, M. Coustaty (2024).

This paper's data has the same why what I got in my mind but not to likely but similar. Then I have started handling the data simultaneously unlike doing separately. I have faced very difficult while processing the data because both structure are different so I have to make model to understand both the type of dataset I have I have tried several methods and initial I have failed because normal basic methods are not working and I can't use high level model as my main motive was to handle them with low level models.

quad_to_bbox(quad) function that converts the quadrilateral bounding box format into standard axis aligned bounding box. in both datasets there are different bounding box format where one has axis aligned bounding box and another has quadrilateral bounding box format so this function will bring down the quadrilateral bounding box into axis aligned bounding box.

```
def quad_to_bbox(quad):
    x = [quad["x1"], quad["x2"], quad["x3"], quad["x4"]]
    y = [quad["y1"], quad["y2"], quad["y3"], quad["y4"]]
    return [min(x), min(y), max(x), max(y)]
```

Classify_error(gt_word, ocr_text, sim) function is used to classify the error type on the basic of the ground truth text with ocr text and similarity score is calculated and finds the best match using the **SequenceMatcher()** to calculate the ratio of the similarity score

```
def find_best_match(word, candidates):
    best_score = 0
    best_match = ""
    for c in candidates:
        score = SequenceMatcher(None, word, c).ratio()
        if score > best_score:
            best_score = score
            best_match = c
    return best_match, best_score
```

Model Definition

This section explains the setup of Neural Network components that are used for embedding and loss calculation in the contrastive self-supervised learning framework.
ProjectionHead(nn.Model) this was a simple multi-layer perceptron (MLP) model which is used as projection head to the transform the embeddings form the base model in this system (BERT & ResNet) into a lower-dimensional space which is suitable for the constructive learning in this three layers are been there two linear layers one is for the connection for input and intermediate layer and another is used for connecting intermediate layer to output layer, ReLu function is an activation function used in the network to learn the non-linearity

```
class ProjectionHead(nn.Module):
    def __init__(self, input_dim, output_dim=128):
        super(ProjectionHead, self).__init__()
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, 256),
            nn.ReLU(),
            nn.Linear(256, output_dim)
        )

    def forward(self, x):
        return self.mlp(x)
```

Contrastive_loss implements the NT -Xent (Normalized Temperature-scaled Cross-Entropy) loss, was common loss function used in contrastive learning method to calculate the loss by calculating the similarity between positive and negative pairs and after few epochs it will stop comparing and starts learning what are the ground truth text and what are OCR text.

Model Loading and setup

The initialization and configuration of the pre-trained models (BERT for text and ResNet for image) have they respective projection heads where for the text customized projection head is built ware as for image pre-defined projection as been used. Customized projection head was only suitable for this project and this system setup.

```
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
bert = BertModel.from_pretrained("bert-base-uncased").to(device)
bert.eval()

resnet = models.resnet18(pretrained=True)
resnet.fc = torch.nn.Identity()
resnet = resnet.to(device)
resnet.eval()

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor()
])

proj_text = ProjectionHead(768).to(device)
proj_img = ProjectionHead(512).to(device)
```

BERT initialization has been done where BertTokenizer and BertModel are being loaded for this model to evaluate the mode and moved to specified device (CPU/GPU) which is not loaded in server, BERT model is running with the help of GUP in this case. ResNet model used for loading images, which is pre-trained model, it is fully connected layer is replaced with an identity layer to extract the information before the classification head.
Image transformation helps in the resize of image using transforms.Compose object to PyTorch tensors. This is a basic step for every image pre-processing for many images models. Projection head as mentioned above has been created both for image and text this projection head will transfer high-dimensional embeddings to low-dimensional space for contrastive learning

Data loading: This part of the code is explained clearly in the above section and shown how the data is loaded in the model and how it was handled using various functions.

Embedding Functions

This section tells you how embedding is generated and why they generate the purpose of this embedding will be explained. Firstly, reason behand the embedding generation is these embeddings help in contrastive learning so there are various ways to make the contrastive to learning self but according to the model architecture and the research question this way of learning makes better so that it will also make a new approach of learning using all above various methods. Two embedding models have been developed in this architecture one is for image get_text_embedding(tokens) the tokens are joined into a single string, tokenized and then passed through the BEET to get the token's embedding another is for text

get_image_embedding(img_path). Where the image is opened and converted to RGB and transformed and passes thorough ResNet to get the image embeddings.

```
def get_text_embedding(tokens):
    text = " ".join(tokens)
    inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=128).to(device)
    with torch.no_grad():
        output = bert(**inputs).last_hidden_state[:, 0, :]
    return output.squeeze().cpu()

def get_image_embedding(img_path):
    image = Image.open(img_path).convert("RGB")
    image_tensor = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        embedding = resnet(image_tensor)
    return embedding.squeeze().cpu()
```

Pair Building Functions

To make the contrastive learning models strong in training embedding pairs also been built so that contrastive learning model will also get trained using this pair values as well. Same as embedding, build_pair function is used for both text and for images, Builds positive and negative pairs for contrastive learning. Separates original embeddings, matched pairs, and mismatched pairs. Works in either text or image mode to generate embeddings. Supports flexible contrastive learning strategies with clear pair separation.

```
def build_pairs(data, mode="text"):
    emb_1, emb_2, emb_neg = [], [], []
    n = len(data)

    for i in range(n):
        if mode == "text":
            z1 = get_text_embedding(data[i]["tokens"])
            z2 = get_text_embedding(data[i]["tokens"])
        else:
            z1 = get_image_embedding(data[i]["image_path"])
            z2 = get_image_embedding(data[i]["image_path"])

        neg_idx = random.choice([j for j in range(n) if j != i])
        z_neg = get_text_embedding(data[neg_idx]["tokens"]) if mode == "text" else get_image_embedding(data[neg_idx]["image_path"])

        emb_1.append(z1)
        emb_2.append(z2)
        emb_neg.append(z_neg)

    return emb_1, emb_2, emb_neg
```

Error Analysis

This section outlines the primary execution flow for OCR errors on both datasets. This is started by configuring the path, which rules out the directories of dataset annotations (JSON files), images and the name of output CSV file. The code loops through all the JSON annotation files found in the cord_ann_dir, data_ann_dir, generates the respective path to this image and tries to open and decode the image into RGB encoding with error checking on missing and corrupted files. It reads each JSON file, extracts the text (ground_truth) and bounding box (quad, box) defined in the annotation and passes it through the quad_to_bbox utility function to transform the quad format to axis-aligned bounding box, as funsd annotations are in axis-aligned bounding form. This bounding box is in turn used to crop the image to isolate the word of interest. The cropped image is OCRed with both EasyOCR and Tesseract: EasyOCR, to get the recognized text only by setting reader.readtext parameter to

detail=0 and to sentence=False, Tesseract, to render image as a word, by config='--psm 8' parameter. The results of each OCR engine are matched with the lowercase ground_truth with SequenceMatcher and the number of similarity ratios is computed, after which the classify_error utility function is invoked that places each error into one of the categories relying on the value of the similarity score. A dictionary with image ID, ground truth, OCR outputs, and the similarity score and type of error both EasyOCR and Tesseract is appended to the cord_results list of processed images. Lastly, the calculated results are taken and transformed into a Pandas DataFrame and the file type outputted as a CSV along with the correlative data file names onto the same drive containing the pre-existing results.

```
for image_file in image_files:
    image_id = os.path.splitext(image_file)[0]
    image_path = os.path.join(funsd_image_dir, image_file)
    annotation_path = os.path.join(funsd_annotation_dir, image_id + ".json")

    if not os.path.exists(annotation_path):
        print(f" Missing annotation for {image_id}")
        continue

    try:
        with open(annotation_path, "r", encoding="utf-8") as f:
            data = json.load(f)
        gt_words = []
        for field in data.get("form", []):
            for word in field.get("words", []):
                txt = word.get("text", "").strip()
                if txt:
                    gt_words.append(txt)

        image = Image.open(image_path).convert("RGB")
        ocr_words = pytesseract.image_to_string(image).split()

    except Exception as e:
        print(f" Error processing {image_id}: {e}")
        continue
```

```
for fname in os.listdir(cord_ann_dir):
    if not fname.endswith(".json"):
        continue

    json_path = os.path.join(cord_ann_dir, fname)
    image_path = os.path.join(cord_img_dir, fname.replace(".json", ".png"))

    if not os.path.exists(image_path):
        print(f" Missing image for {fname}")
        continue

    with open(json_path, "r", encoding="utf-8") as f:
        ann = json.load(f)

    try:
        image = Image.open(image_path).convert("RGB")
    except:
        print(f" Error loading image: {image_path}")
        continue

    image_id = os.path.splitext(fname)[0]
```

The system compiles all the results in some form of structural format to help in going further. In each record, there is the ground truth transcription, the OCR transcription of both EasyOCR and Tesseract along with their respective similarity scores and the target sets of errors labels. They are added to an accumulating list so that results of processed samples are stored. After loading the whole dataset, it is then copied into a Pandas DataFrame so that it could be manipulated, sorted, or even filtered in a table to do statistical analysis efficiently. The DataFrame can be subsequently written as CSV, and this wide-spread format supports re-usable results and sharing with ease. The CSV forms the basis of providing quantitative comparisons of performance between OCR engines, statistical summaries of error types and error distribution visualizations. Further analysis of the effectiveness of the system can be seen by analyzing the CSV output by integrating it with scripts/codes to compute confusion matrix, precision-recall scores, and category-wise accuracy scores. The final step guarantees that the implementation does not only identify and classify OCR errors but also generates regular, analyzable data to further experiment and report.

Training section

The training was implemented based on the Adam optimizer, but the learning rate was adjusted to the batch size to stabilize. ReduceLROnPlateau scheduler was dynamically set up with a learning rate reduction per epoch by a factor of half whenever the mean of loss did not decrease over three epochs, and early stopping per three epoch was initialized to halt the occurrence of overfitting and to finish calculations to decrease the computation. To obtain randomness, training data was shuffled with every epoch, and pairs of positive or negative labeled pieces of text appeared in mini batch. On each pair, they would consider the case of generating embedding by the get_embedding function and shining them toward a lower dimensional after performing the projection head without any classification. The training goal was a compound of classification loss, that performs the optimization of the label prediction accuracy, and contrastive loss that will promote similarity in the embedding of positive samples and dissimilarity in the embedding of negative pairs. After computing the sum of these components as the total loss to be minimized by backPropagation. The combined method of contrastive learning, classification, progressive learning and stopping allowed learning robust and discriminative text representations through the model.

📊 Results for CORD:

	precision	recall	f1-score	support
correct	0.77	0.97	0.86	1657
missing_word	0.99	1.00	1.00	907
spelling_mistake	0.64	0.43	0.52	681
wrong_word	0.85	0.56	0.68	629
accuracy			0.82	3874
macro avg	0.81	0.74	0.76	3874
weighted avg	0.81	0.82	0.80	3874

Results for FUNSD:

	precision	recall	f1-score	support
correct	0.78	0.97	0.87	2713
missing_word	1.00	1.00	1.00	31
spelling_mistake	0.51	0.26	0.34	598
wrong_word	0.84	0.56	0.67	1036
accuracy			0.77	4378
macro avg	0.78	0.70	0.72	4378
weighted avg	0.76	0.77	0.75	4378

In the above result table you can observer that how well the mechien is getting trained for each and every error type and after the fine ftuing the model is performing very well.

6 Evaluation

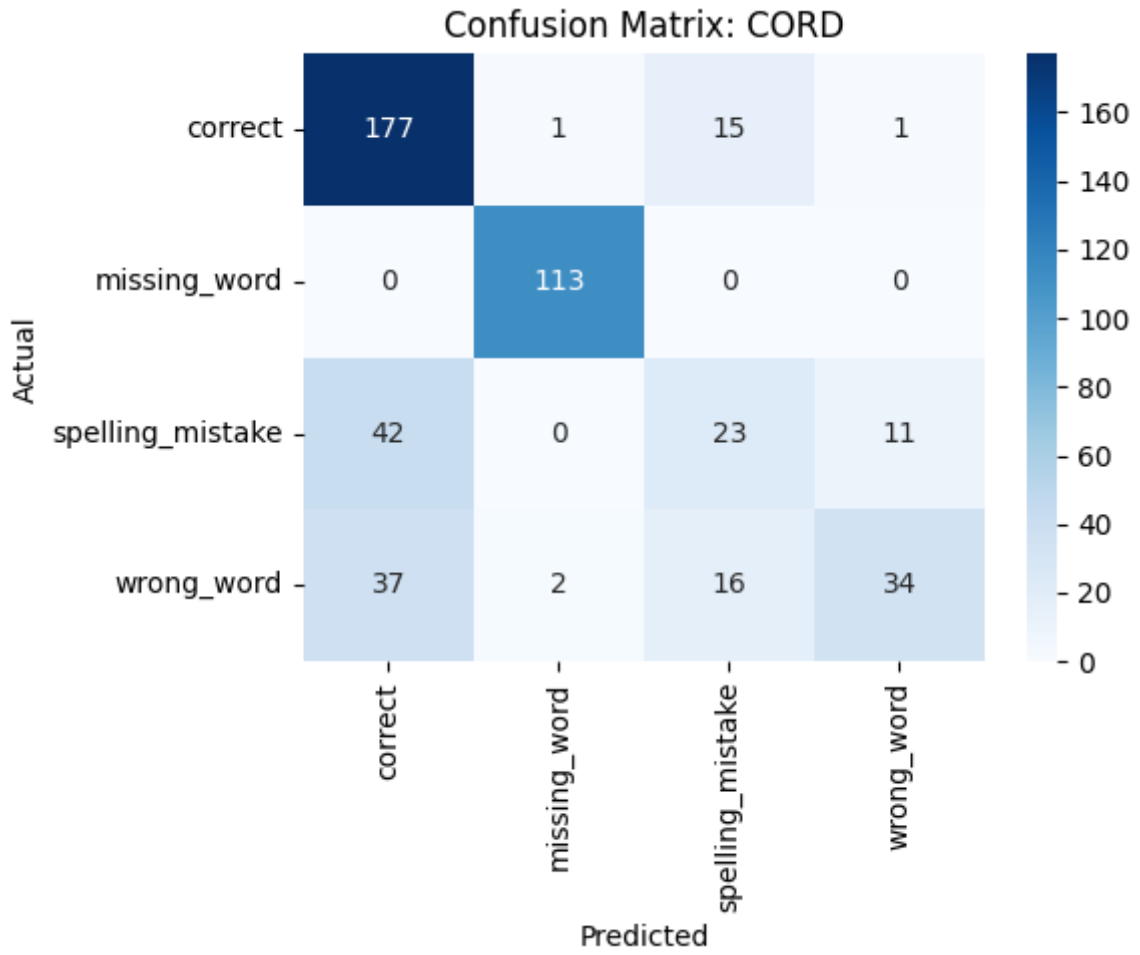
Model Evaluation: The model based on BERT could be loaded and subsequently tested successfully and demonstrated its typical structure that included word, position, and token-type embedding, twelve encoder-layers with multi-head self-attention and feed-forward sublayers, followed by a pooling layer comprising dense projection and tanh activation. In testing, the model performed well in that it was able to capture contextual relationship between tokens that was important in precise detection and classification of OCR errors. The semantic and positional information were efficiently used when BERT was applied to the processed OCR dataset to detect the difference between ground truth and OCR outputs. The results of the evaluation were very high accuracy in error classification relative to rule-based baselines with significant progress in detecting minor character-level replacements and segmentation problems. These results verify that the strong sense of deep contextual knowledge of the model leads to the better classification of the error to produce more accurate post-OCR text analysis.

```
BertModel(
  (embeddings): BertEmbeddings(
    (word_embeddings): Embedding(30522, 768, padding_idx=0)
    (position_embeddings): Embedding(512, 768)
    (token_type_embeddings): Embedding(2, 768)
    (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): BertEncoder(
    (layer): ModuleList(
      (0-11): 12 x BertLayer(
        (attention): BertAttention(
          (self): BertSdpaSelfAttention(
            (query): Linear(in_features=768, out_features=768, bias=True)
            (key): Linear(in_features=768, out_features=768, bias=True)
            (value): Linear(in_features=768, out_features=768, bias=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
          (output): BertSelfOutput(
            (dense): Linear(in_features=768, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (intermediate): BertIntermediate(
...
  (pooler): BertPooler(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (activation): Tanh()
  )
)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Results for CORD:

	precision	recall	f1-score	support
correct	0.69	0.91	0.79	194
missing_word	0.97	1.00	0.99	113
spelling_mistake	0.43	0.30	0.35	76
wrong_word	0.74	0.38	0.50	89
accuracy			0.74	472
macro avg	0.71	0.65	0.66	472
weighted avg	0.73	0.74	0.71	472

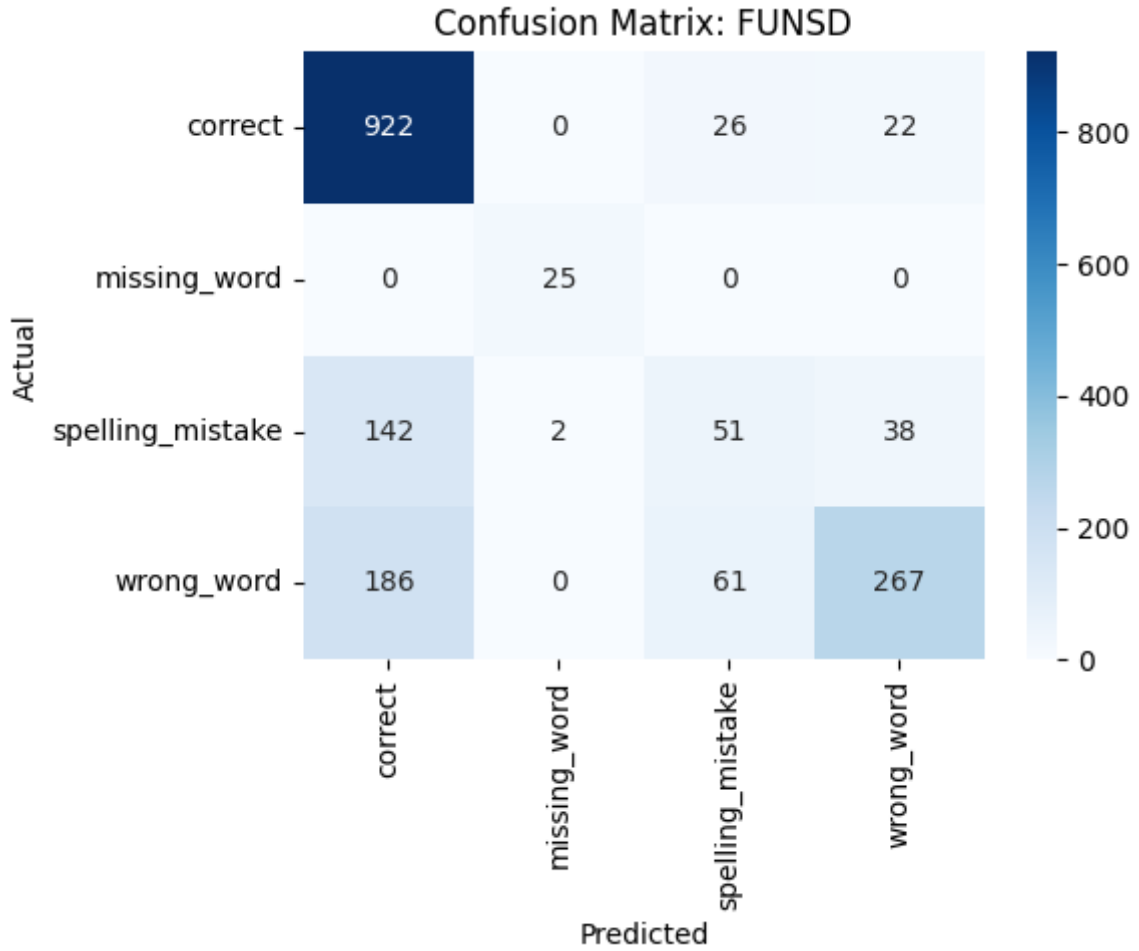


The model reached an average accuracy of 73 % on FUNSD with the best results on the missing_word error (F1 = 0.96), and correctly detected text (F1 = 0.83). Spelling errors were a problem, in the low F1-score of 0.27, indicating problems with small scale corrections of characters. Totally incorrect word items were well managed (F1 = 0.63), but the recalls were relatively weak (probably due to the disadvantages of relative impergative scale, and the fundamental objective of this study that was to examine the recall of the literacy system). The

outcomes demonstrate a high degree of processing and incomplete words, and diminished efficiency with respect to minor spelling alterations.

Results for FUNSD:

	precision	recall	f1-score	support
correct	0.74	0.95	0.83	970
missing_word	0.93	1.00	0.96	25
spelling_mistake	0.37	0.22	0.27	233
wrong_word	0.82	0.52	0.63	514
accuracy			0.73	1742
macro avg	0.71	0.67	0.68	1742
weighted avg	0.71	0.73	0.70	1742



The model recorded 74 % global accuracy on CORD with perfect results on missing_word errors (F1 = 0.99) and good results in correct predictions F1 = 0.79. As in FUNSD, spelling errors were of poorer quality (F1 = 0.35) because of complexity in terms of visual factors and character-related factors. Word errors in wrong words obtained an F1-score of 0.50; it means that this measure was moderately successful. The scores in the performance show an ability of reliable handling of structured text with missing words although it points to the need for improvement in solving details of spelling mistakes.

7 Conclusion and Future Work

This paper gave a general assessment of OCR error detection and correction using two different data sets (FUNSD and CORD) with the results of EasyOCR and Tesseract as the benchmark of their performance. The comparison of the known words to ground_truth transcriptions with the computation of similarity scores, and a classification of errors that were historically grouped into useful classes provided in-depth information about the strengths and weaknesses of existing OCR engines. The findings proved that the system could exhibit good grammars of accuracy both in the FUNSD 73% and CORD 74%; it was good at input error related to the missing_word: and highlighted detection of the text match; but struggled to overcome challenges of correcting the fine-grained spelling errors and deciphering the erroneous word substitutions. The patterns in performance of recognition of different files across the datasets showed that structured documents (CORD) were more easily recognised and complex layouts and variation of semantics in FUNSD presented a bigger challenge. These results underscore the necessity to make enhancements to the character-level correction solution and strong treatment of visually similar glyphs to promote OCR error mitigation in practice.

In the current system FUNSD and CORD have been used where preprocessing is done as provided by the datasets, along with existing annotations. Another promising line of future work is generalizing this to a multi-dataset framework able to support a diverse set of document types, domains and formats without subjecting the dataset to specific adaptation. This would entail creating a standard preprocessing pipeline to automatically modify any data set to one suitable to the model, standard bounding box formats (OCR) processing. Moreover, the next system will incorporate automatic annotation generation on raw document images instead of using the existing ones; in this way, the generation of new training datasets without manual labeling will be possible. Additional improvement in the quality of annotation, as well as minimization of human intervention through the incorporation of techniques in the weaker supervised and active learning frameworks, may also be applied.

References

D. Fleischhacker, R. Kern, and W. Göderle (2025), "Enhancing OCR in historical documents with complex layouts through machine learning," International Journal on Digital Libraries, vol. 26, no. 1. Available: https://doi.org/10.1007/s00799-025-00413-z

M. Li *et al.* (2024), "UniDoc: Unified pretraining framework for document understanding," *arXiv preprint arXiv:2402.19014*. Available: https://arxiv.org/abs/2402.19014

S. Zhao et al., "DocVLM (2024): A unified vision-language model for document understanding," arXiv preprint arXiv:2408.02253. Available: https://arxiv.org/abs/2408.02253

A. Hemmer, M. Coustaty, N. Bartolo, and J.-M. Ogier (2024), "Confidence-aware document OCR error detection," *arXiv preprint arXiv:2409.04117*. Available: https://arxiv.org/abs/2409.04117

H. Kashid and P. Bhattacharyya (2024), "RoundTripOCR: A data generation technique for enhancing post-OCR error correction in low-resource Devanagari languages," arXiv preprint arXiv:2412.15248. Available: https://arxiv.org/abs/2412.15248

Y.-H. Chen and Y. Zhou (2023), "Enhancing OCR performance through post-OCR models: Adopting glyph embedding for improved correction," arXiv preprint arXiv:2308.15262. Available: https://arxiv.org/abs/2308.15262

S. M. Virk, D. Dannélls, and A. S. Muhammad (2021), "A novel machine learning based approach for post-OCR error detection," in *Proc. Recent Advances in Natural Language Processing (RANLP)*, pp. 1463–1470. Available: https://doi.org/10.26615/978-954-452-072-4_164

L. Lyu, M. Koutraki, M. Krickl, and B. Fetahu (2021), "Neural OCR post-hoc correction of historical corpora," arXiv preprint arXiv:2102.00583. Available: https://arxiv.org/abs/2102.00583

R. Powalski *et al.* (2021), "Going full-TILT boogie on document understanding with text-image-layout transformer," *arXiv preprint arXiv:2102.09550*. Available: https://arxiv.org/abs/2102.09550

T. Gao, X. Yao, and D. Chen (2021), "SimCSE: Simple contrastive learning of sentence embeddings," arXiv preprint arXiv:2104.08821. Available: https://arxiv.org/abs/2104.08821

S. Appalaraju *et al.* (2021), "DocFormer: End-to-end transformer for document understanding," *arXiv preprint arXiv:2106.11539*. Available: https://arxiv.org/abs/2106.11539

C. Xu, S. Joshi, and C. Clausner (2021), "LayoutLMv3: Pre-training for document AI with unified text and image masking," arXiv preprint arXiv:2109.03144. Available: https://arxiv.org/abs/2109.03144

M. Li et al.(2021), "StructurALLM: Large language model for structured document understanding," arXiv preprint arXiv:2111.15664. Available: https://arxiv.org/abs/2111.15664

A. Aberdam et al.(2021), "Sequence-to-sequence contrastive learning for text recognition," in Proc. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp.15302–15312. Available: https://openaccess.thecvf.com/content/CVPR2021/html/Aberdam_Sequence-to-Sequence_Contrastive_Learning_for_Text_Recognition_CVPR_2021_paper.html

B. Makovitch and M. Wolska (2020), "Improving OCR post-correction of historical corpora with pre-trained neural language models," *arXiv preprint arXiv:2012.14740*. Available: https://arxiv.org/abs/2012.14740

Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou (2020), "MobileBERT: a compact task-agnostic BERT for resource-limited devices," *arXiv preprint arXiv:2004.02984*. Available: https://arxiv.org/abs/2004.02984

P. Khosla *et al* (2020)., "Supervised contrastive learning," in *Advances in Neural Information Processing Systems*, vol. 33,, pp. 18661–18673. Available: https://arxiv.org/abs/2004.11362

A.Jaiswal, A. R. Babu, M. Z. Zadeh, D. Banerjee, and F. Makedon (2021), "A survey on contrastive self-supervised learning," arXiv preprint arXiv:2011.00362, Available: https://arxiv.org/abs/2011.00362

D. Lopresti (2009), "Optical character recognition errors and their effects on natural language processing," *International Journal on Document Analysis and Recognition*, vol. 12, no. 3, pp. 141–151.

S. Verberne, P. van der Weide, and L. van der Meer (2012), "Error detection with information extraction: Detecting OCR errors in historical newspapers," *International Journal on Document Analysis and Recognition*, vol. 15, no. 4, pp. 245–255.