National College of Ireland

Configuration Manual

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:	**623**	**Page Count: 6**	

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):	

Configuration Manual

GokulKrishna.

Student ID: x23350512

Introduction:

This manual provides a comprehensive guide to how to set up my project environment, by installing all the required packages, and ensuring a smooth execution workflow of the project. It shows the essential components and dependencies necessary to successfully run the project and get results.

The purpose of this manual is to support the configuration-related packages, since the original project was developed and tested in Visual Studio Code (VS Code). Therefore, the instructions are to be mentioned to set up environment.

System Specification

Hardware Specification:

Following are the hardware specifications of the system that was used to develop the project.

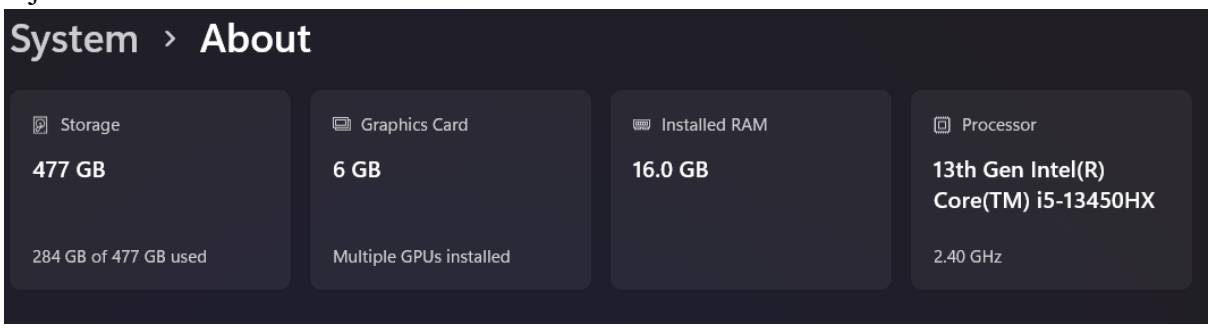


Table 1: System Specifications

Software Specification

The software specification of the system is.

Software	Specifications
Operating System (OS) IDE Scripting Language	Windows 11 (64 bit) Visual Studio Code (VS Code) Python 3.10.11

Table 2: Software Specifications

Software Tools

Following are the software tools that are used to implement the project:

Python

This project was developed with the help of Python (3.10.11) programming language. Packages contented in this programming language are the reason to choose this language which are very useful for the data preprocessing and implementing of the project. To install the specific version of the Python just use the link [1].

Visual Studio Code

Visual Studio Code was used as a compiler to run the code, there are different options as well to compile the code, but It's up to the users as the code was written in .ipynb file which can be complied in VS Code and in Jupyter Notebook.

Implementation

Set of libraries are being installed for this project which are essential to this project in VS Code using **pip install function** which it will set up the libraries automatically. So, to install all the libraries just run this commend **"pip install requirements.txt"** as only particular version of every library so important to run the above commend.

Global configuration and Tesseract setup

This was the major step in this project, which was used in project to run the model in local PC, so the major aim in the project is to be used low level models with minimal resources and get the result so this is the reason behand using CUDA and Tesseract.

CUDA is a computing platform that makes the model run on GUP instead of CPU which makes model smoother like same how we see in google colab. To install the CUDA, follow the link [2].

```
# --- Global Configurations ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Tesseract CMD (if needed, adjust path for your system)
pytesseract.pytesseract.tesseract_cmd = r"C:\Program Files\Tesseract-OCR\tesseract.exe"

# EasyOCR Reader
reader = easyocr.Reader(["en"])
```

Data Loading

```
def load_cord_data(json_dir, image_dir):
    data = []
    for file in os.listdir(json_dir):
        if not file.endswith(".json"):
            continue
```

```
def load_funsd_data(split="train", base_path="dataset"):
    if split == "train":
        split_path = os.path.join(base_path, "training_data")
    elif split == "test":
        split_path = os.path.join(base_path, "testing_data")
    else:
        raise ValueError(

    image_dir = os.path.j
    ann_dir = os.path.joi
    image_ids = [os.path.splitext(f)[0] for f in os.listdir(image_dir) if f.endswith(".png")]

    data = []
    for image_id in image_ids:
        img_path = os.path.join(image_dir, f"{image_id}.png")
        ann_path = os.path.join(ann_dir, f"{image_id}.json")
        if not os.path.exists(img_path) or not os.path.exists(ann_path):
            continue
```

```
(function) def splitext(p: AnyOrLiteralStr@splitext) -> tuple[AnyOrLiteralStr@splitext, AnyOrLiteralStr@splitext]

Split the extension from a pathname.

Extension is everything from the last dot to the end, ignoring leading dots. Returns "(root, ext)"; ext may be empty.
```

So, the above figure shows how the data is loaded in my model and below links are the path of dataset which has been used in the project

C:\Users\N Gokul Krishna\Desktop\main code\CORD

C:\Users\N Gokul Krishna\Desktop\main code\dataset

Model definition

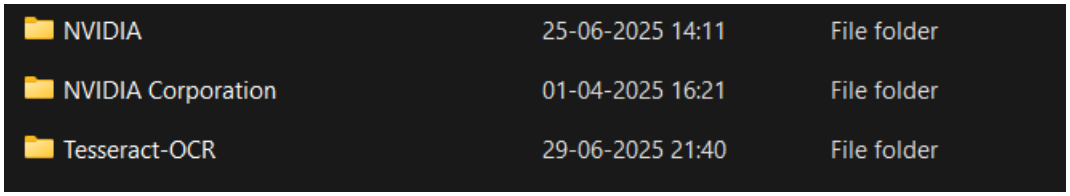
The code below shows how the model is defined to handle the dataset which is used to extracting the ground truth and ocr wors in the respective datasets in this project.

```
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
```

To use CUDA we need to install **NVIDIA** in our local PC as you can see below

C:\Program Files\NVIDIA\CUDNN\v9.10 #NVIDIA path in my system

C:\Program Files\Tesseract-OCR #tesseract path in my system



Model installation

The model utilizes BERT and RESNET models in this project there are different BERT models used in project, easyOCR one of the model which is used in this project to extract the text for an image to generate some ground truth and OCR words form the image and form annotation

```
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)   # BERT output
proj_img = ProjectionHead(512).to(device)    # ResNet output
```

```
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_pa (variable) image: Image
    image_tensor = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        embedding = resnet(image_tensor)
    return embedding.squeeze().cpu()
```

The below figure show that how the training will start

```
Epoch 1 | Batch 1 | Loss: 4.3614
Epoch 1 | Batch 2 | Loss: 4.1757
Epoch 1 | Batch 3 | Loss: 4.3226
Epoch 1 | Batch 4 | Loss: 4.2171
Epoch 1 | Batch 5 | Loss: 4.1964
Epoch 1 | Batch 6 | Loss: 3.8649
Epoch 1 | Batch 7 | Loss: 3.9716
Epoch 1 | Batch 8 | Loss: 3.3793
Epoch 1 | Batch 9 | Loss: 3.4164
Epoch 1 | Batch 10 | Loss: 3.2536
Epoch 1 | Batch 11 | Loss: 3.3725
Epoch 1 | Batch 12 | Loss: 3.5282
Epoch 1 | Batch 13 | Loss: 3.0104
Epoch 1 | Batch 14 | Loss: 2.9858
Epoch 1 | Batch 15 | Loss: 3.0584
Epoch 1 | Batch 16 | Loss: 3.5314
Epoch 1 | Batch 17 | Loss: 3.6941
Epoch 1 | Batch 18 | Loss: 3.1736
Epoch 1 | Batch 19 | Loss: 3.3370
Epoch 1 | Batch 20 | Loss: 2.9887
Epoch 1 | Batch 21 | Loss: 2.9512
```

The selection model trains using training dataset and parameters are adjusted according to the dataset like identifying the ocr_errors and identifying them the model is trained on different datasets and also gets the results how perfect the model is able to identify the ocr error types

```
from torch.optim.lr_scheduler import ReduceLROnPlateau        Chat (CTRL + I) / Share (CTRL

batch_size = 32
epochs = 10

# Adjust learning rate based on batch size
base_lr = 1e-3 * (batch_size / 16)
optimizer = torch.optim.Adam(
    list(proj_head.parameters()) + list(classifier.parameters()), lr=base_lr
)

# Scheduler and early stopping
scheduler = ReduceLROnPlateau(optimizer, mode='min', patience=3, factor=0.5)
best_loss = float('inf')
no_improve = 0
patience = 5

for epoch in range(1, epochs + 1):
    random.shuffle(train_data)
    epoch_loss = 0.0
    num_batches = 0
```

Reference

1) https://www.nvidia.com/en-us/drivers/
2) https://www.python.org/downloads/release/python-31011/