Configuration Manual

MSc Research Project
Data Analytics

Shashank Sanjay Tomar
Student ID: x19213280

School of Computing
National College of Ireland

Supervisor: Dr. Paul Stynes & Dr. Pramod Pathak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Shashank Sanjay Tomar
Student ID:	x19213280
Programme:	Data Analytics
Year:	2021-2022
Module:	MSc Research Project
Supervisor:	Dr. Paul Stynes & Dr. Pramod Pathak
Submission Due Date:	16/12/2021
Project Title:	Summarizing Newspaper Articles using Optical Character Recognition and Natural Language Processing
Word Count:	2415
Page Count:	17

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:	Shashank Sanjay Tomar
Date:	27th January 2022

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

Shashank Sanjay Tomar
x19213280

1 Introduction

This configuration manual presents a step-by-step walkthrough of the research, as well as information on the hardware and software used to implement it. By following this guide, any user can replicate the conducted research.

2 Hardware and Software Specification

2.1 Hardware Specifications

There were primarily two instances of hardware used, one a local machine equipped with a GPU, and the other a cloud-based IDE (Google Colab PRO). By doing this, this research could train the models simultaneously, reducing implementation time and enhancing efficiency. The table 1 describing their specifications is provided below.

Table 1: Hardware Specifications

Name	**Description**
Local Machine	Asus G14
OS	Windows 11 (V. 21H2)
CPU	AMD Ryzen 4900H
RAM	16 GB DDR4
GPU	RTX 2060 MaxQ (6 GB)
Google Colab PRO	32GB of RAM, Tesla P100 GPU (16 GB)

NOTE: One needs to subscribe to Google colab Pro in order to use the higher GPUs listed in the table 1. Also, the specifications of the hardware provided by colab pro are dynamic and can change according to the user's usage.

2.2 Software Specifications

A number of software tools, IDEs, frameworks, and libraries were used in addition to the hardware mentioned above. Detailed information is provided in the following table 2.

Table 2: Software Specifications

Name	Description
Language	Python 3.7
IDEs	Jupyter Notebook & Google Colab Pro
Image Data Annotation	VGG Image Annotator
Base Models and Weights	MaskRCNN (COCO Dataset) & bert-base-uncased
Text Extraction Engine	Tesseract.exe
Spell-Grammar Check	Microsoft Bing API
Text-To-Speech	gTTS (Google)
Model Creation	Tensorflow, Keras, Pytorch
Evaluation	Tensorboard, ROUGE etc.
Miscellaneous Tasks	Libraries like Matplotlib, NumPy, JSON, Requests, Regex etc.

3 Data Collection and Transformation

3.1 Dataset 1: Newspaper Images

For the first dataset, the research acquires images from the "Times of India" newspaper for the January 18 issue. The steps are as follows:

Step 1: Data Download

This step began with downloading a zip file of dataset from the Archive website [1]. This folder contained three different versions of each newspaper image ("-C": Complete Image, "-P": Only Pictures, "-T": Only Text). Next, the images were moved from separate date publication folders to one in a single folder. File names that ended in "-P" or "-T" were ignored and deleted since they exclusively contained only the "Picture" or "Text" data of the newspaper. The reason for doing this was that we wanted our model to be trained on real-life situations where there are both pictures and text in a newspaper.

Step 2: Data Selection & Cleaning

After obtaining images of the complete newspaper pages containing, both text and pictures, the researchers had to manually go through each one of them to remove any images that contained only advertisements since the goal of the research was to summarize news articles. Having done so, the dataset was left with 182 images, each representing a newspaper page. These images were then split into 70:20:10 ratios for train, validation, and test data.

Step 3: Data Annotation

The research uses VGG image Annotator to manually annotate a boundary box around various articles after the dataset was split into three separate sets.

- Images from the train dataset were imported into the VGG annotator tool by clicking on the "Add Files" button, and two classes (Rectangle Article and Non Rectangle Article) were added in the "Region Attributes" section as can be seen in Figure 1.
- Using the "Polygon" region shape, a boundary box around a news article was made and then a corresponding class was selected through a dropdown menu, as shown in the figure 2.

[1]Times of India Jan-18 Dataset: `https://archive.org/details/TOIDELJAN18`

Figure 1: Importing Images and adding Annotation Classes

Figure 2: Bounding Box Class Selection

- After all the article boundary boxes were annotated with the corresponding class (Rectangle/Non-Rectangle), a JSON file containing all these annotations was downloaded using "Annotation"-"Export Annotations (as JSON)", as shown in the figure 3 below.

Figure 3: Downloading JSON File

- The same annotation steps were performed on the validation dataset, and each JSON file was saved in its respective folder (Train Annotation JSON in the Train Image Folder, Validation Annotation JSON in the Validation Image Folder).

3.2 Dataset 2: CNN-DailyMail Dataset

Huggingface repository provides an easy method to access this article-summary pairs dataset[2] using its "dataset" python package. The dataset is downloaded via the following script (Fig. 4) and then stored in the local cache. As it is downloaded in the cache, it is directly accessible from the cache for subsequent dataset requests. Additionally, the train, validation, and test data splits were already available in the huggingface repository.

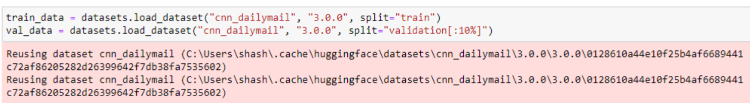


Figure 4: Downloading CNN-DailyMail Dataset

[2]CNN-DailyMail Dataset: `https://huggingface.co/datasets/cnn_dailymail`

4 Experiment Setup

4.1 Experiment 1 : Article & Column Segmentation using MaskR-CNN

This experiment was implemented to segment the article and column images from the newspaper images. The steps followed for this experiment were:

Training Phase

Step 1: The MaskRCNN base model by Matterport Inc. also used by Almutairi and Almashan (2019) was downloaded from Github [3]. Since, this research used transfer learning, the base MaskRCNN (COCO Dataset) weight[4] was also downloaded. Both the "maskrcnn" (Github) folder and the "mask_rcnn_coco.h5" weight were kept in the same folder as the jupyter notebook.

Step 2: As the next step, a few libraries were installed, custom configurations such as setting the ROOT Dierectory, Dataset path, base weight path, confidence detection level etc. were set as can be seen in the figure 5.

```
# !pip install keras==2.4.3
# !pip install tensorflow==2.3.1
# !pip install -U scikit-image==0.16.2
```

```
import os
import sys
import json
import datetime
import numpy as np
import skimage.draw
import cv2
from mrcnn.visualize import display_instances
import matplotlib.pyplot as plt
from mrcnn.config import Config
from mrcnn import model as modellib, utils
```

Setting Configs

```
ROOT_DIR = os.getcwd()
#Mask RCNN Training for Stage 1 (Identifying Articles from a Newspaper Page)
DataSet_DIR= os.path.join(ROOT_DIR,"Dataset")

#Mask RCNN Training for Stage 2 (Identifying Columns from various Articles)
#DataSet_DIR=os.path.join(ROOT_DIR,"Dataset_Stage2")
```

```
# Import Mask RCNN
sys.path.append(ROOT_DIR)  # To find local version of the library
# Path to trained weights file of mask_rcnn_coco as the Base Model
COCO_WEIGHTS_PATH = os.path.join(ROOT_DIR, "mask_rcnn_coco.h5")
# Directory to save logs
DEFAULT_LOGS_DIR = os.path.join(ROOT_DIR, "logs")
```

```
class CustomConfig(Config):
    """Configuration for training on the custom  dataset.
    Derives from the base Config class and overrides some values.
    """
    # Give the configuration a recognizable name
    NAME = "object"
    #Setting Images per GPU
    IMAGES_PER_GPU = 2
    # Number of classes
    NUM_CLASSES = 1 + 2   # Background + Rectangle Article, Non-Rectangle Article

    #Set the Steps as 131 (Stage 1 Training) and 52(Stage 2 Training)
    # Number of training steps per epoch
    STEPS_PER_EPOCH = 131
    # Skip detections with < 85% confidence
    DETECTION_MIN_CONFIDENCE = 0.85
```

Figure 5: Configuration Settings

[3] Matterport Inc. MaskRCNN base: https://github.com/matterport/Mask_RCNN

[4] MaskRCNN trained on COCO Dataset Weight: https://github.com/matterport/Mask_RCNN/releases/download/v2.0/mask_rcnn_coco.h5

Step 3: Next, a custom class was defined, containing methods to load the custom article annotations, masks as can be seen from the figure 6.

```
class CustomDataset(utils.Dataset):

    def load_custom(self, dataset_dir, subset,annotationFilePath):
        # Adding two classes
        self.add_class("object", 1, "RectangleArticle")
        self.add_class("object", 2, "NonRectangleArticle")
        # Train or validation dataset
        assert subset in ["train", "val"]
        dataset_dir = os.path.join(dataset_dir, subset)
        # We mostly care about the x and y coordinates of each region
        annotations1 = json.load(open(annotationFilePath))
        # print(annotations1)
        annotations = list(annotations1.values())
        annotations = [a for a in annotations if a['regions']]
        # Adding images
        for a in annotations:
            polygons = [r['shape_attributes'] for r in a['regions']]
            objects = [s['region_attributes']['Names'] for s in a['regions']]
            print("objects:",objects)
            name_dict = {"RectangleArticle": 1,"NonRectangleArticle": 2}
            # key = tuple(name_dict)
            num_ids = [name_dict[a] for a in objects]
            print("numids",num_ids)
            image_path = os.path.join(dataset_dir, a['filename'])
            image = skimage.io.imread(image_path)
            height, width = image.shape[:2]

            self.add_image(
                "object",
                image_id=a['filename'],
                path=image_path,
                width=width, height=height,
                polygons=polygons,
                num_ids=num_ids
                )

    def load_mask(self, image_id):
        image_info = self.image_info[image_id]
        if image_info["source"] != "object":
            return super(self.__class__, self).load_mask(image_id)
        info = self.image_info[image_id]
        if info["source"] != "object":
            return super(self.__class__, self).load_mask(image_id)
        num_ids = info['num_ids']
        mask = np.zeros([info["height"], info["width"], len(info["polygons"])],
                        dtype=np.uint8)
        for i, p in enumerate(info["polygons"]):
            rr, cc = skimage.draw.polygon(p['all_points_y'], p['all_points_x'])
            mask[rr, cc, i] = 1
        num_ids = np.array(num_ids, dtype=np.int32)
        return mask, num_ids

    def image_reference(self, image_id):
        """Return the path of the image."""
        info = self.image_info[image_id]
        if info["source"] == "object":
            return info["path"]
        else:
            super(self.__class__, self).image_reference(image_id)
```

Figure 6: Loading Custom Dataset and Mask

Step 4: Once, the custom annotation and mask loading methods were defined, a Mask RCNN model training method was created, using the specifications seen in the figure 7. This trained a custom MaskRCNN Model based on the newspaper image data using transfer learning on MaskRCNN (COCO Dataset) weight.

```python
def train(model,DataSet_DIR):
    # Training dataset.
    dataset_train = CustomDataset()
    dataset_train.load_custom(DataSet_DIR, "train",os.path.join(DataSet_DIR,"train\demo_json.json"))
    dataset_train.prepare()

    # Validation dataset
    dataset_val = CustomDataset()
    dataset_val.load_custom(DataSet_DIR, "val",os.path.join(DataSet_DIR,"val\demo_val_json.json"))
    dataset_val.prepare()
    print("Training network heads")
    model.train(dataset_train, dataset_val,
                learning_rate=config.LEARNING_RATE,
                epochs=20,
                layers='heads')
```

```python
config = CustomConfig()
model = modellib.MaskRCNN(mode="training", config=config,
                                  model_dir=DEFAULT_LOGS_DIR)
weights_path = COCO_WEIGHTS_PATH
if not os.path.exists(weights_path):
  utils.download_trained_weights(weights_path)
model.load_weights(weights_path, by_name=True, exclude=[
            "mrcnn_class_logits", "mrcnn_bbox_fc",
            "mrcnn_bbox", "mrcnn_mask"])
#Training the Model
train(model,DataSet_DIR)
```

Figure 7: MaskRCNN Training

Step 5: Once, the article segmentation model was trained. It was time to move on to training a second Mask RCNN model to segment the columns from those identified article segmentation. This was done by training yet another model on Stage 2 Dataset, by performing annotations on article images by following similar annotation steps as mentioned in the subsection 3.1 . Figure 8 showcases an example of the same.

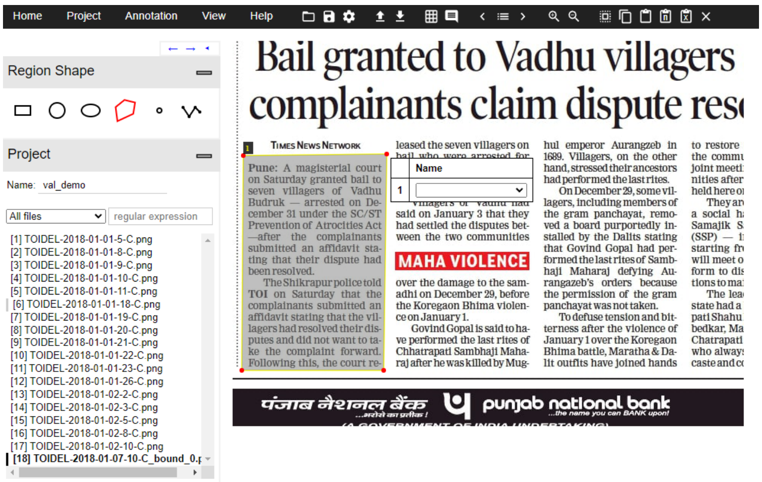


Figure 8: Stage 2 Dataset Annotation for Column Segmentation MaskRCNN Model

Step 6: After the creation of the Stage 2 dataset with annotated article images, with column annotations. Aforementioned Step 1 to Step 4 were implemented again, to produce Stage 2 MaskRCNN model to identify the columns after the Stage 1 MaskRCNN model would identify the articles from newspaper. However, this was done after making sure, the dataset directory was changed to "Stage 2 Dataset" as can be seen in the figure 9.

Setting Configs

```
ROOT_DIR = os.getcwd()
#Mask RCNN Training for Stage 1 (Identifying Articles from a Newspaper Page)
#DataSet_DIR= os.path.join(ROOT_DIR,"Dataset")

#Mask RCNN Training for Stage 2 (Identifying Columns from various Articles)
DataSet_DIR=os.path.join(ROOT_DIR,"Dataset_Stage2")
```

Figure 9: Dataset directory: Stage 2 Dataset (Column Segmentation)

Testing/Inference Phase

Step 1: Once, both the Stage 1(Article Segmentation) and Stage 2(Column Segmentation) MaskRCNN Models were trained, an inference was made by using our custom trained model weights and boundary boxes and masks were displayed on the test newspaper images as can be seen in Figure 10(Article Segmentation) and Figure 11(Column Segmentation).

Figure 10: Stage 1 Model: Article Segmentation


Predictions1

'Illegal occupants have no right over govt land'

New Delhi: The Delhi high court on Wednesday asked a group of visually impaired students, whose hostel was recently demolished, to move to an alternative accommodation given to them, saying they were illegal occupants of government land and cannot claim legal right over it.

A bench of acting Chief Justice Gita Mittal and Justice C Hari Shankar, however, asked the agencies concerned to ensure that the visually impaired students, who were left without a roof in winter, are lodged in proper regular hostels.

The court was hearing a PIL by nine visually impaired students, who alleged that DDA acted illegally in forcibly evicting them from the hostel in southwest Delhi where they were residing since 2000.

The petitioners submitted that a direction be issued to the authorities not to remove them from the site, were they were staying in temporary structures.

"You will have to move out of that place as the agencies have also given you an alternative temporary accommodation," the bench said and asked the counsel for the petitioners "with what authority the students are staying there". The court has fixed the matter for January 16. PTI

Figure 11: Stage 2 Model: Column Segmentation

Step 2: The Stage 1 model inference was run on all the test images of newspaper to segment articles and then Stage 2 model inference was run on those article images to segment them into column images. The cropped Article and Column Images were stored within distinct folders(combined on the basis of newspaper print date), using the code in figure 12.

```
#Running Detection on All Images under test folder and Cropping the Identified Article Blocks into results Folder
for file in os.listdir(test_folder):
    path_to_new_image = os.path.join(test_folder, file)
    image1 = mpimg.imread(path_to_new_image)
    # Run object detection
    print(len([image1]))
    results1 = model.detect([image1], verbose=1)
    # Display results
    ax = get_ax(1)
    r1 = results1[0]
    visualize.display_instances(image1, r1['rois'], r1['masks'], r1['class_ids'],
    dataset.class_names, r1['scores'], ax=ax, title="Predictions for "+file)
    #Cropping Boundaries of the Article Blocks
    imageFile=path_to_new_image
    i=0
    for r in r1['rois']:
        x = r[0]-15
        y = r[1]-15
        if x<0:
            x=0
        if y<0:
            y=0
        width = r[2]+30
        height = r[3]+30
        image = cv2.imread(imageFile)
        crop_img = image[x:width, y:height]
        segmentFileName = r"Article_%d.png"%i
        temp=file.split('TOIDEL')[1]
        temp=temp.split('.')[0]
        temp='TOIDEL'+temp
        segmentFileName=temp+"_"+segmentFileName
        if not os.path.isdir(os.path.join(ROOT_DIR,"Articles",temp)):
            os.makedirs(os.path.join(ROOT_DIR,"Articles",temp))
        cv2.imwrite(os.path.join(ROOT_DIR,"Articles",temp,segmentFileName),crop_img)
        i+=1
```

Figure 12: Cropping Segmented Article/Column Images

4.2 Experiment 2 : Text Extraction using Tesseract

Step 1: The researchers installed libraries such as OpenCV and PyTesseract and changed the path of PyTesseract command to point to the installed executable of Tesseract.exe. In addition to that, a few set of image pre-processing methods were defined to grayscale and threshold an image, which can be seen in the figure 13.

```
# conda install -c conda-forge opencv
# conda install -c conda-forge pytesseract
# pip install neattext
```

```
import cv2
import pytesseract
import numpy as np
import os
import neattext as nt
import re
import requests
import json
from statistics import mean
```

```
#SET THE TESSERACT EXE PATH
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
```

Image Processing Methods

```
# get grayscale image
def get_grayscale(image):
    return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# noise removal
def remove_noise(image):
    return cv2.medianBlur(image,1)

#thresholding
def thresholding(image):
    return cv2.threshold(image, 80, 255, cv2.THRESH_TOZERO)[1]
```

Figure 13: Installing PyTesseract,OpenCV, NeatText and Image Pre-Processing Methods

Step 2: The next step was to have all the extracted column images go through the tesseract engine in order to extract the text and then concatenate to form the textual content on a single article. This was done using the code shown in the figure 14.

```
for subFolder in os.listdir(articles_folder):
    for subArticleFolder in os.listdir(os.path.join(articles_folder,subFolder)):
        if not ("jpg" in subArticleFolder) or ("png" in subArticleFolder):
            text=""
            if os.path.isfile(os.path.join(articles_folder,subFolder,subArticleFolder,"OCRedText.txt")):
                os.remove(os.path.join(articles_folder,subFolder,subArticleFolder,"OCRedText.txt"))
            for columnFile in os.listdir(os.path.join(articles_folder,subFolder,subArticleFolder)):
                OCR_img=cv2.imread(os.path.join(articles_folder,subFolder,subArticleFolder,columnFile))
                #Image Processing
                OCR_img=processImage(OCR_img)
                extracted_text=pytesseract.image_to_string(OCR_img)
                text=text+" "+extracted_text
                #Getting the Confidence Score Data
                tesseractData=pytesseract.image_to_data(OCR_img,output_type=pytesseract.Output.DICT)
                confScore=list(np.float_(tesseractData['conf']))
                confScore=list(filter((-1.0).__ne__, confScore))
                confidenceScoreList.append(mean(confScore))

            with open(os.path.join(articles_folder,subFolder,subArticleFolder,"OCRedText.txt"), 'w') as f:
                f.write(text)
```

Figure 14: Extracting Text using Tesseract

Step 3: The researchers then cleaned the extracted text using their own code and used the Spelling and Grammar Check API by Microsoft Bing by making a request call using an API_KEY as can be seen in the figure 15. This Microsoft Bing API Key was generated by hosting a service through Azure Portal. Due to the scope of the study not being to demonstrate how to use a service, the Microsoft documentation website [5] can be consulted for that.

```
def cleanText(text):
    #Handling New Line Character
    text=text.lower()
    text=text.replace('-\n','')
    text=text.replace('\n',' ')
    #Replace Characters
    onlyCharacterList=["\\","|","<",">","*","[","]","_","~","x0c","+",",.",".,","§","#","@"]
    for char in onlyCharacterList:
        text=text.replace(char,"")
    while ".." in text:
        text=text.replace("..",".")
    #Replace Special Words
    onlyWordList=["Times News Network","timesgroup.com"]
    for word in onlyWordList:
        text=re.sub(r'\b'+word+r'\b','',text)
    #Remove random Alphabets
    onlyAlphaList=['b','c','d','e','f','g','h','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y','z']
    for char in onlyAlphaList:
        text=re.sub(r'\b'+char+r'\b','',text)
    docx=nt.TextFrame(text)
    docx.remove_multiple_spaces().text
    docx.remove_emails().text
    text=docx.text
    return text
```

Preliminary Bing Spell Check API Call Method

```
numOfReplacements=0
def spellCheck(text):
    global numOfReplacements
    api_key = "[redacted]"
    endpoint = "https://api.bing.microsoft.com/v7.0/SpellCheck"
    example_text = text
    data = {'text': example_text}
    params = {'mkt':'en-us','mode':'proof'}
    headers = {'Content-Type': 'application/x-www-form-urlencoded','Ocp-Apim-Subscription-Key': api_key,}
    response = requests.post(endpoint, headers=headers, params=params, data=data)
    json_response = response.json()
    #Printing JSON Response
    #print(json.dumps(json_response, indent=4))
    if json_response['flaggedTokens']!=[]:
        #Iterating through the JSON Response to replace the flagged Spelling Mistakes
        for item in json_response['flaggedTokens']:
            textToBeReplaced=item['token']
            scoreList=[]
            for suggestion in item['suggestions']:
                scoreList.append(suggestion['score'])
            for suggestion in item['suggestions']:
                if suggestion['score']==max(scoreList):
                    textReplacingWith=suggestion['suggestion']
            #example_text=example_text.replace(textToBeReplaced,textReplacingWith)
            textToBeReplaced=textToBeReplaced.replace("(","")
            textToBeReplaced=textToBeReplaced.replace(")","")
            numOfReplacements+=1
            example_text=re.sub(r'\b'+textToBeReplaced+r'\b',textReplacingWith,example_text)
    return example_text
```

Figure 15: Text Cleaning and Spelling-Grammar Check

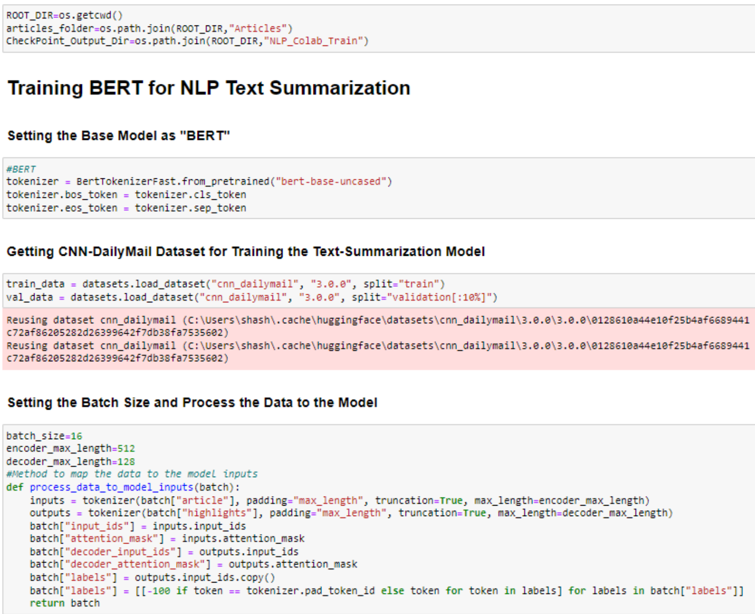
[5] Bing API Documentation: `https://docs.microsoft.com/en-us/azure/cognitive-services/bing-spell-check/overview`

4.3 Experiment 3 : Text Summarization by BERT-NLP

This experiment was implemented to generate an audio and text summary from the extracted article text. The steps followed for this experiment were:

Training Phase

Step 1: The researchers installed libraries like "Datasets", "gTTS", "rouge_score" and "transformers", set a few basic configurations, and then downloaded "bert-base-uncased" base BERT model. In addition, training parameters such as batch size to be 16, maximum encoder length to be 512 etc. were set, as shown in the figure 16.



```
ROOT_DIR=os.getcwd()
articles_folder=os.path.join(ROOT_DIR,"Articles")
CheckPoint_Output_Dir=os.path.join(ROOT_DIR,"NLP_Colab_Train")
```

Training BERT for NLP Text Summarization

Setting the Base Model as "BERT"

```
#BERT
tokenizer = BertTokenizerFast.from_pretrained("bert-base-uncased")
tokenizer.bos_token = tokenizer.cls_token
tokenizer.eos_token = tokenizer.sep_token
```

Getting CNN-DailyMail Dataset for Training the Text-Summarization Model

```
train_data = datasets.load_dataset("cnn_dailymail", "3.0.0", split="train")
val_data = datasets.load_dataset("cnn_dailymail", "3.0.0", split="validation[:10%]")
```

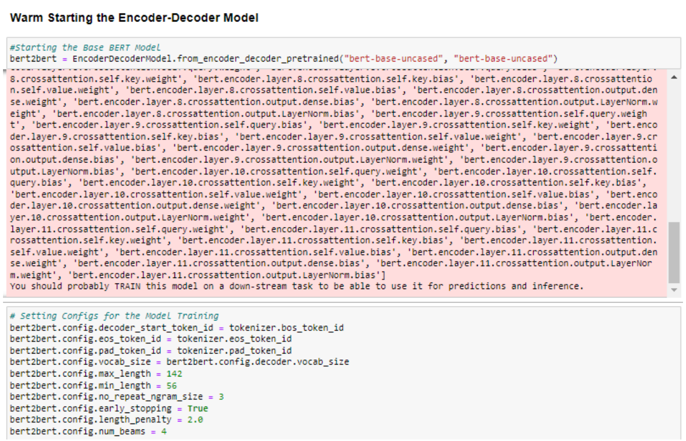
```
Reusing dataset cnn_dailymail (C:\Users\shash\.cache\huggingface\datasets\cnn_dailymail\3.0.0\3.0.0\0128610a44e10f25b4af6689441
c72af86205282d26399642f7db38fa7535602)
Reusing dataset cnn_dailymail (C:\Users\shash\.cache\huggingface\datasets\cnn_dailymail\3.0.0\3.0.0\0128610a44e10f25b4af6689441
c72af86205282d26399642f7db38fa7535602)
```

Setting the Batch Size and Process the Data to the Model

```
batch_size=16
encoder_max_length=512
decoder_max_length=128
#Method to map the data to the model inputs
def process_data_to_model_inputs(batch):
    inputs = tokenizer(batch["article"], padding="max_length", truncation=True, max_length=encoder_max_length)
    outputs = tokenizer(batch["highlights"], padding="max_length", truncation=True, max_length=decoder_max_length)
    batch["input_ids"] = inputs.input_ids
    batch["attention_mask"] = inputs.attention_mask
    batch["decoder_input_ids"] = outputs.input_ids
    batch["decoder_attention_mask"] = outputs.attention_mask
    batch["labels"] = outputs.input_ids.copy()
    batch["labels"] = [[-100 if token == tokenizer.pad_token_id else token for token in labels] for labels in batch["labels"]]
    return batch
```

Figure 16: Configuration and Parameter Settings

Step 2: After mapping the train and validation data to match the model inputs, researchers moved on to the next step. The "base-bert-uncased" model was warm started and the parameters for a "bert2bert" model were set as can be seen from the figure17.



Warm Starting the Encoder-Decoder Model

```
#Starting the Base BERT Model
bert2bert = EncoderDecoderModel.from_encoder_decoder_pretrained("bert-base-uncased", "bert-base-uncased")
```

```
8.crossattention.self.key.weight', 'bert.encoder.layer.8.crossattention.self.key.bias', 'bert.encoder.layer.8.crossattentio
n.self.value.weight', 'bert.encoder.layer.8.crossattention.self.value.bias', 'bert.encoder.layer.8.crossattention.output.den
se.weight', 'bert.encoder.layer.8.crossattention.output.dense.bias', 'bert.encoder.layer.8.crossattention.output.LayerNorm.w
eight', 'bert.encoder.layer.8.crossattention.output.LayerNorm.bias', 'bert.encoder.layer.9.crossattention.self.query.weigh
t', 'bert.encoder.layer.9.crossattention.self.query.bias', 'bert.encoder.layer.9.crossattention.self.key.weight', 'bert.enco
der.layer.9.crossattention.self.key.bias', 'bert.encoder.layer.9.crossattention.self.value.weight', 'bert.encoder.layer.9.cr
ossattention.self.value.bias', 'bert.encoder.layer.9.crossattention.output.dense.weight', 'bert.encoder.layer.9.crossattenti
on.output.dense.bias', 'bert.encoder.layer.9.crossattention.output.LayerNorm.weight', 'bert.encoder.layer.9.crossattention.o
utput.LayerNorm.bias', 'bert.encoder.layer.10.crossattention.self.query.weight', 'bert.encoder.layer.10.crossattention.self.
query.bias', 'bert.encoder.layer.10.crossattention.self.key.weight', 'bert.encoder.layer.10.crossattention.self.key.bias',
'bert.encoder.layer.10.crossattention.self.value.weight', 'bert.encoder.layer.10.crossattention.self.value.bias', 'bert.enco
der.layer.10.crossattention.output.dense.weight', 'bert.encoder.layer.10.crossattention.output.dense.bias', 'bert.encoder.la
yer.10.crossattention.output.LayerNorm.weight', 'bert.encoder.layer.10.crossattention.output.LayerNorm.bias', 'bert.encoder.
layer.11.crossattention.self.query.weight', 'bert.encoder.layer.11.crossattention.self.query.bias', 'bert.encoder.layer.11.c
rossattention.self.key.weight', 'bert.encoder.layer.11.crossattention.self.key.bias', 'bert.encoder.layer.11.crossattention.
self.value.weight', 'bert.encoder.layer.11.crossattention.self.value.bias', 'bert.encoder.layer.11.crossattention.output.den
se.weight', 'bert.encoder.layer.11.crossattention.output.dense.bias', 'bert.encoder.layer.11.crossattention.output.LayerNor
m.weight', 'bert.encoder.layer.11.crossattention.output.LayerNorm.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
# Setting Configs for the Model Training
bert2bert.config.decoder_start_token_id = tokenizer.bos_token_id
bert2bert.config.eos_token_id = tokenizer.eos_token_id
bert2bert.config.pad_token_id = tokenizer.pad_token_id
bert2bert.config.vocab_size = bert2bert.config.decoder.vocab_size
bert2bert.config.max_length = 142
bert2bert.config.min_length = 56
bert2bert.config.no_repeat_ngram_size = 3
bert2bert.config.early_stopping = True
bert2bert.config.length_penalty = 2.0
bert2bert.config.num_beams = 4
```

Figure 17: Warm Starting the bert-base-uncased Model

Step 3: Moving forward to the stage of tuning and training the custom BERT-NLP Summarization model. As can be seen from the figure 18, the parameters such as logging_step, eval_step, batch size per device, etc. were set and the model was trained.

Fine-Tuning the Encoder-Decoder Model

```
#Setting Training Parameters
training_args = Seq2SeqTrainingArguments(
    output_dir=CheckPoint_Output_Dir,
    evaluation_strategy="steps",
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    predict_with_generate=True,
    logging_steps=1000,
    save_steps=10000,
    eval_steps=8000,
    warmup_steps=2000,
    overwrite_output_dir=True,
    save_total_limit=100,
    fp16=True,
)
```

Training Model

```
trainer = Seq2SeqTrainer(
    model=bert2bert,
    tokenizer=tokenizer,
    args=training_args,
    compute_metrics=compute_metrics,
    train_dataset=train_data,
    eval_dataset=val_data,
)
trainer.train()
```

```
/usr/local/lib/python3.6/dist-packages/datasets/arrow_dataset.py:835: UserWarning: The given NumPy array is not writeable, and
PyTorch does not support non-writeable tensors. This means you can write to the underlying (supposedly non-writeable) NumPy arr
ay using the tensor. You may want to copy the array to protect its data or make it writeable before converting it to a tensor.
This type of warning will be suppressed for the rest of this program. (Triggered internally at  /pytorch/torch/csrc/utils/tenso
r_numpy.cpp:141.)
  return torch.tensor(x, **format_kwargs)
/usr/local/lib/python3.6/dist-packages/torch/optim/lr_scheduler.py:123: UserWarning: Detected call of `lr_scheduler.step()` bef
ore `optimizer.step()`. In PyTorch 1.1.0 and later, you should call them in the opposite order: `optimizer.step()` before `lr_s
cheduler.step()`.  Failure to do this will result in PyTorch skipping the first value of the learning rate schedule. See more d
etails at https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate
  "https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate", UserWarning)
```

[53835/53835 24:11:51, Epoch 3/3]

Step	Training Loss	Validation Loss	Rouge2 Precision	Rouge2 Recall	Rouge2 Fmeasure	Runtime	Samples Per Second
8000	2.959800	2.891456	0.098700	0.145800	0.113700	525.446900	2.545000
16000	2.611700	2.634151	0.104400	0.155300	0.120700	534.901200	2.500000
24000	2.359700	2.511087	0.110000	0.163100	0.127200	544.302700	2.456000
32000	2.293400	2.437275	0.107100	0.159900	0.124100	563.116300	2.374000
40000	2.105600	2.394423	0.110400	0.161700	0.127100	545.211800	2.452000
48000	2.069000	2.363725	0.106700	0.156600	0.122800	557.875000	2.397000

```
TrainOutput(global_step=53835, training_loss=2.5525387110390834, metrics={'train_runtime': 87112.8614, 'train_samples_per_secon
```

Figure 18: BERT-NLP Model Training

Testing/Inference Phase

Step 1: As soon as the BERT-NLP Summarization model was trained and the final checkpoint was achieved, the model was run on the clean extracted text from the articles in the previous experiment and the summary was saved in the corresponding article folder, using the code in the figure 19.

```
def generate_summary(text,model_dir):
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    #CHECK THE CHECKPOINT
    tokenizer = BertTokenizerFast.from_pretrained(model_dir)
    model = EncoderDecoderModel.from_pretrained(model_dir).to(device)
    inputs = tokenizer([text], padding="max_length", truncation=True, max_length=512, return_tensors="pt")
    input_ids = inputs.input_ids.to(device)
    attention_mask = inputs.attention_mask.to(device)
    output = model.generate(input_ids, attention_mask=attention_mask)
    return tokenizer.decode(output[0], skip_special_tokens=True)
```

```
for subFolder in os.listdir(articles_folder):
    for subArticleFolder in os.listdir(os.path.join(articles_folder,subFolder)):
        if not ("jpg" in subArticleFolder) or ("png" in subArticleFolder):
            with open(os.path.join(articles_folder,subFolder,subArticleFolder,"Clean_Spell_Check_Text.txt")) as f:
                text = f.read()
                print("Generating Summary for: ",subFolder,subArticleFolder,"Clean_Spell_Check_Text.txt")
                text=generate_summary(text,os.path.join(CheckPoint_Output_Dir,"checkpoint-FINAL"))
                with open(os.path.join(articles_folder,subFolder,subArticleFolder,"Summary.txt"), 'w') as fWrite:
                    fWrite.write(text)
```

```
Generating Summary for:  TOIDEL-2018-01-01-15-C Article_0 Clean_Spell_Check_Text.txt
Generating Summary for:  TOIDEL-2018-01-01-15-C Article_1 Clean_Spell_Check_Text.txt
Generating Summary for:  TOIDEL-2018-01-01-15-C Article_2 Clean_Spell_Check_Text.txt
Generating Summary for:  TOIDEL-2018-01-01-15-C Article_3 Clean_Spell_Check_Text.txt
Generating Summary for:  TOIDEL-2018-01-01-15-C Article_4 Clean_Spell_Check_Text.txt
Generating Summary for:  TOIDEL-2018-01-02-4-C Article_0 Clean_Spell_Check_Text.txt
```

Figure 19: BERT-NLP Model Inference Generating Summaries

Step 2: By using the Microsoft Bing API, the generated summary was yet again put through spelling and grammar check, using a similar code snippet as shown in the figure 15. The cleaned text summary was saved as a text file in corresponding article folders.

Step 3: To conclude the experiments and generate a final audio summary from the cleaned text summary, python's gTTS (Google text-to-speech interface) package was used, as shown in the figure 20.

```
def generateAudioSummary(text,filePath):
    #Checking if it Exists,Then Delete
    if(os.path.isfile(os.path.join(filePath,"AudioSummary.mp3"))):
        os.remove(os.path.join(filePath,"AudioSummary.mp3"))
    # Language in which you want to convert
    language = 'en'
    myobj = gTTS(text=text, lang=language, slow=False)
    # Saving the converted audio in a mp3 file named
    myobj.save(os.path.join(filePath,"AudioSummary.mp3"))
    # Playing the converted file
    #os.system("mpg321 "+os.path.join(filePath,"AudioSummary.mp3"))
```

```
for subFolder in os.listdir(articles_folder):
    for subArticleFolder in os.listdir(os.path.join(articles_folder,subFolder)):
        if not ("jpg" in subArticleFolder) or ("png" in subArticleFolder):
            with open(os.path.join(articles_folder,subFolder,subArticleFolder,"Clean_Summary.txt")) as f:
                text = f.read()
                print("Generating Audio of Clean Summary for: ",subFolder,subArticleFolder,"Clean_Summary.txt")
                generateAudioSummary(text,os.path.join(articles_folder,subFolder,subArticleFolder))
```

```
Generating Audio of Clean Summary for:  TOIDEL-2018-01-01-15-C Article_0 Clean_Summary.txt
Generating Audio of Clean Summary for:  TOIDEL-2018-01-01-15-C Article_1 Clean_Summary.txt
Generating Audio of Clean Summary for:  TOIDEL-2018-01-01-15-C Article_2 Clean_Summary.txt
Generating Audio of Clean Summary for:  TOIDEL-2018-01-01-15-C Article_3 Clean_Summary.txt
Generating Audio of Clean Summary for:  TOIDEL-2018-01-01-15-C Article_4 Clean_Summary.txt
```

Figure 20: Generating Audio Summaries

5 Evaluation

Since the technologies used in each experiment were different, the evaluation criteria and metrics associated with each experiment were chosen accordingly.

5.1 Evaluation of Experiment 1 : Article & Column Segmentation using MaskRCNN

Tensorboard is one of the most popular tools for evaluating deep learning models. It is a visualisation tool that tracks and plots loss training and validation loss curves. As part of this research, the Bounding Box and Mask losses were analyzed with each epoch. Tensorboard can be started in the following steps:

- As can be seen in the figure 21, enter the command (tensorboard –logdir logs_Directory_path) in the terminal of your environment by replacing the "logs_Directory_path" with the path where the trained weights are stored.

```
(base) C:\Users\shash>tensorboard --logdir C:\Users\shash\Desktop\Project\CodeBackup\logs\object20211127T2208
2021-12-15 19:38:35.595981: W tensorflow/stream_executor/platform/default/dso_loader.cc:59] Could not load dynamic libra
ry 'cudart64_101.dll'; dlerror: cudart64_101.dll not found
2021-12-15 19:38:35.596164: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do n
ot have a GPU set up on your machine.
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.7.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

Figure 21: Environment Terminal

- After this, Tensorboard can be accessed by visiting "http://localhost:6006/" through a browser as seen in the figure 22.

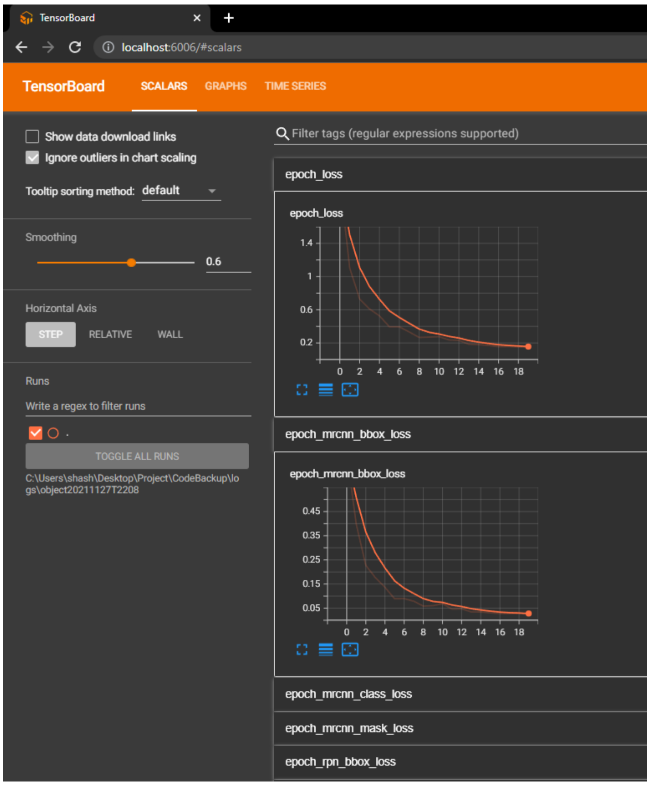


Figure 22: Tensorboard

- In the figure 23, an example of the custom Stage 1(Article Segmentation) MaskRNN model's validation and training loss curves is showcased.

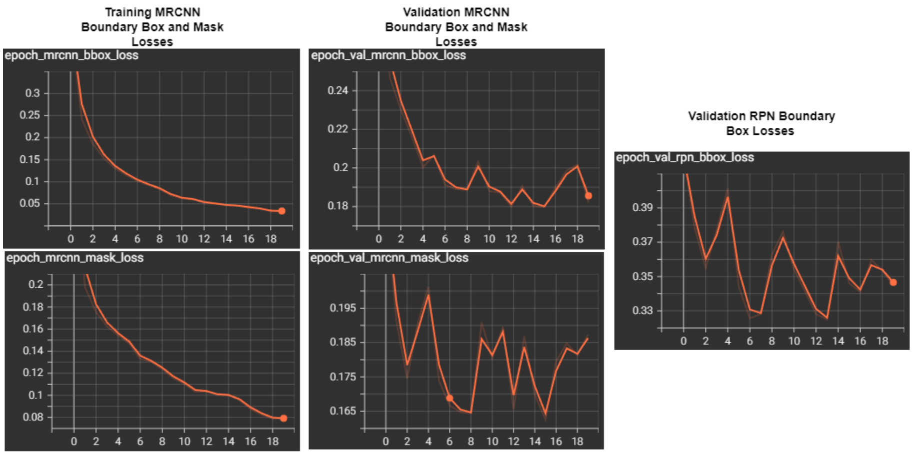


Figure 23: Stage 1(Article Segmentation) MaskRNN model's loss curves

5.2 Evaluation of Experiment 2 : Text Extraction using Tesseract

The average confidence score of every recognition by tesseract was used to evaluate the quality of the text recognition. Using the Bing API, a second evaluation was performed by calculating the number of changes suggested by the spelling and grammar check. The code snippet for these evaluations and the results are shown below in the figure 24.

```
for subFolder in os.listdir(articles_folder):
    for subArticleFolder in os.listdir(os.path.join(articles_folder,subFolder)):
        if not ("jpg" in subArticleFolder) or ("png" in subArticleFolder):
            text=""
            if os.path.isfile(os.path.join(articles_folder,subFolder,subArticleFolder,"OCRedText.txt")):
                os.remove(os.path.join(articles_folder,subFolder,subArticleFolder,"OCRedText.txt"))
            for columnFile in os.listdir(os.path.join(articles_folder,subFolder,subArticleFolder)):
                OCR_img=cv2.imread(os.path.join(articles_folder,subFolder,subArticleFolder,columnFile))
                #Image Processing
                OCR_img=processImage(OCR_img)
                extracted_text=pytesseract.image_to_string(OCR_img)
                text=text+" "+extracted_text
                #Getting the Confidence Score Data
                tesseractData=pytesseract.image_to_data(OCR_img,output_type=pytesseract.Output.DICT)
                confScore=list(np.float_(tesseractData['conf']))
                confScore=list(filter((-1.0).__ne__, confScore))
                confidenceScoreList.append(mean(confScore))

            with open(os.path.join(articles_folder,subFolder,subArticleFolder,"OCRedText.txt"), 'w') as f:
                f.write(text)
```

```
print("The Mean Confidence score of the OCR on Articles with image processing is: ",mean(confidenceScoreList))
```

```
The Mean Confidence score of the OCR on Articles with image processing is:  82.79506867655532
```

```
numOfReplacements=0
def spellCheck(text):
    global numOfReplacements
    api_key = [redacted]
    endpoint = "https://api.bing.microsoft.com/v7.0/SpellCheck"
    example_text = text
    data = {'text': example_text}
    params = {'mkt':'en-us','mode':'proof'}
    headers = {'Content-Type': 'application/x-www-form-urlencoded','Ocp-Apim-Subscription-Key': api_key,}
    response = requests.post(endpoint, headers=headers, params=params, data=data)
    json_response = response.json()
    #Printing JSON Response
    #print(json.dumps(json_response, indent=4))
    if json_response['flaggedTokens']!=[]:
        #Iterating through the JSON Response to replace the flagged Spelling Mistakes
        for item in json_response['flaggedTokens']:
            textToBeReplaced=item['token']
            scoreList=[]
            for suggestion in item['suggestions']:
                scoreList.append(suggestion['score'])
            for suggestion in item['suggestions']:
                if suggestion['score']==max(scoreList):
                    textReplacingWith=suggestion['suggestion']
            #example_text=example_text.replace(textToBeReplaced,textReplacingWith)
            textToBeReplaced=textToBeReplaced.replace("(","")
            textToBeReplaced=textToBeReplaced.replace(")","")
            numOfReplacements+=1
            example_text=re.sub(r'\b'+textToBeReplaced+r'\b',textReplacingWith,example_text)
    return example_text
```

```
print("The total number of Replacements made: ",numOfReplacements)
```

```
The total number of Replacements made:  4802
```

Figure 24: OCR Confidence Score and SpellCheck Recommendation Count

5.3 Evaluation of Experiment 3 : Text Summarization by BERT-NLP

ROUGE has been suggested by many researchers in the previous literature, like Moratanch and Chitrakala (2017) and Allahyari et al. (2017), as an evaluation criterion to assess the quality of a generated summary against a reference summary (human generated). A method for generating ROUGE 2 metrics to be used with the validation data while traning the BERT-NLP model is shown in the figure 25.

```python
rouge = datasets.load_metric("rouge")
def compute_metrics(pred):
    labels_ids = pred.label_ids
    pred_ids = pred.predictions
    pred_str = tokenizer.batch_decode(pred_ids, skip_special_tokens=True)
    labels_ids[labels_ids == -100] = tokenizer.pad_token_id
    label_str = tokenizer.batch_decode(labels_ids, skip_special_tokens=True)
    rouge_output = rouge.compute(predictions=pred_str, references=label_str, rouge_types=["rouge2"])["rouge2"].mid
    return {
        "rouge2_precision": round(rouge_output.precision, 4),
        "rouge2_recall": round(rouge_output.recall, 4),
        "rouge2_fmeasure": round(rouge_output.fmeasure, 4),
    }
```

Figure 25: Validation ROUGE-2 metrics method

ROUGE-2, ROUGE-1, and ROUGE-L scores were also calculated for the test split of the CNN DailyMail dataset. There were 11,490 article-summary pairs in this split, with the summaries written by professional journalists. The figure 26 illustrates the key results and the code used to create them.

Evaluation of the Trained Model

```python
tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
model = EncoderDecoderModel.from_pretrained(os.path.join(CheckPoint_Output_Dir,"checkpoint-Final"))
model.to("cuda")
#Loading the Test Data
test_data = datasets.load_dataset("cnn_dailymail", "3.0.0", split="test")
batch_size = 32
#Mapping the Data
def generate_summary(batch):
    inputs = tokenizer(batch["article"], padding="max_length", truncation=True, max_length=512, return_tensors="pt")
    input_ids = inputs.input_ids.to("cuda")
    attention_mask = inputs.attention_mask.to("cuda")
    outputs = model.generate(input_ids, attention_mask=attention_mask)
    output_str = tokenizer.batch_decode(outputs, skip_special_tokens=True)
    batch["pred"] = output_str
    return batch
#Getting the Results
results = test_data.map(generate_summary, batched=True, batch_size=batch_size, remove_columns=["article"])
#Getting the Predictions and Actual Summaries
pred_str = results["pred"]
label_str = results["highlights"]
#Computing Rouge Scores
rouge_output = rouge.compute(predictions=pred_str, references=label_str, rouge_types=["rouge2"])["rouge2"].mid
print(rouge_output)
```

Calculating Rouge-2

```python
#Computing Rouge Scores
rouge_output = rouge.compute(predictions=pred_str, references=label_str, rouge_types=["rouge2"])["rouge2"].mid
print(rouge_output)
```

```
Score(precision=0.17675207497792284, recall=0.1990203482945054, fmeasure=0.18215824329209357)
```

Calculating Rouge-1 and Rouge-l

```python
rouge = Rouge()
rouge.get_scores(pred_str, label_str, avg=True)
```

```
{'rouge-1': {'f': 0.2743262484042596,
  'p': 0.26801272128875564,
  'r': 0.29333046589389206},
 'rouge-l': {'f': 0.2578642069309123,
  'p': 0.25183493007656427,
  'r': 0.2758238025053934}}
```

Figure 26: ROUGE-2, ROUGE-1 and ROUGE-l metrics on test data

6 Important Notes

The following section provides a few key points to consider when implementing the research or running the provided code along with the dataset.

6.1 Flow of Jupyter Notebooks

To ensure smooth operation of the entire study, the sequence of the Jupyter Notebooks is imperative. Hence, the notebooks containing Python code should be run in the following order:

1. Train_MaskRCNN_Custom_Dataset.ipynb
2. Test_Mask_RCNN_Stage1_Stage2.ipynb
3. OCR_Clean_Text_Generation.ipynb
4. NLP_Text_Summarization_BERT_Audio_Generation.ipynb

6.2 Ideal Folder Structure

The ideal folder structure would look like the one shown in the Figure 27 after implementing the entire study.

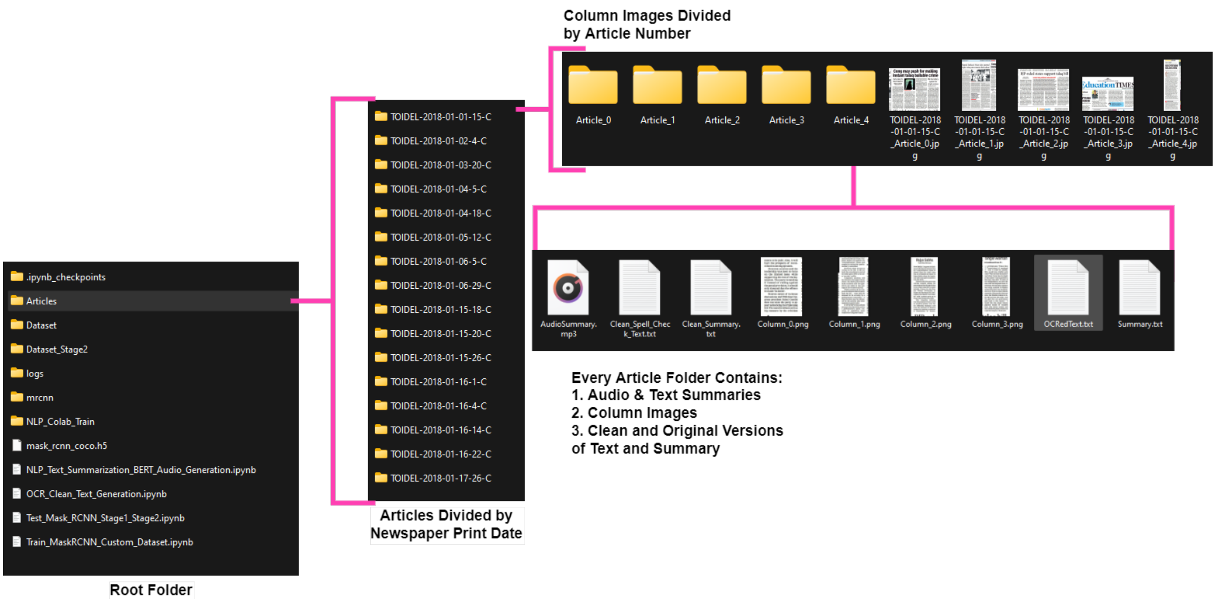


Figure 27: Folder Structure

References

Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E. D., Gutierrez, J. B. and Kochut, K. (2017). Text summarization techniques: a brief survey, *arXiv preprint arXiv:1707.02268* .

Almutairi, A. and Almashan, M. (2019). Instance segmentation of newspaper elements using mask r-cnn, *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, IEEE, pp. 1371–1375.

Moratanch, N. and Chitrakala, S. (2017). A survey on extractive text summarization, *2017 international conference on computer, communication and signal processing (ICCCSP)*, IEEE, pp. 1–6.