Configuration Manual

MSc Research Project
Data Analytics

Bilal Mustaq Mulani
Student ID: x22212132

School of Computing
National College of Ireland

Supervisor: Dr. Ahmed Makki

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Bilal Mustaq Mulani
Student ID:	x22212132
Programme:	Data Analytics
Year:	2024
Module:	MSc Research Project
Supervisor:	Dr. Ahmed Makki
Submission Due Date:	12/08/2024
Project Title:	Configuration Manual
Word Count:	889
Page Count:	9

Signature:	Bilal Mustaq Mulani
Date:	16th September 2024

Configuration Manual

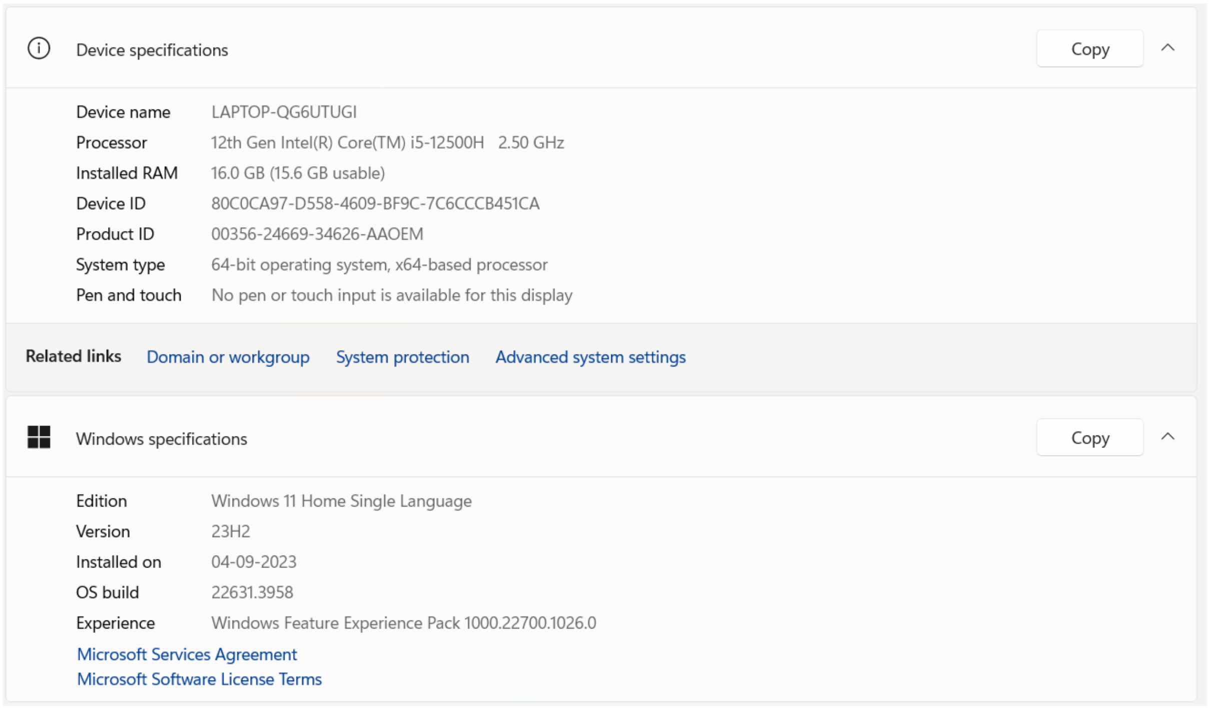
Bilal Mustaq Mulani
x22212132

1 Introduction

The intent of this document is to give the process that was followed during the coding phase for the project. The hardware and software configurations are given in detail so as to enable any future researcher reproduce the research. It includes programming and deployment phases for making code run smoothly and the steps to be followed in executing the code.

2 System Configuration

2.1 Hardware Configuration



Device specifications

Device name	LAPTOP-QG6UTUGI
Processor	12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz
Installed RAM	16.0 GB (15.6 GB usable)
Device ID	80C0CA97-D558-4609-BF9C-7C6CCCB451CA
Product ID	00356-24669-34626-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Windows specifications

Edition	Windows 11 Home Single Language
Version	23H2
Installed on	04-09-2023
OS build	22631.3958
Experience	Windows Feature Experience Pack 1000.22700.1026.0

Figure 1: Attention UNet Model Trends

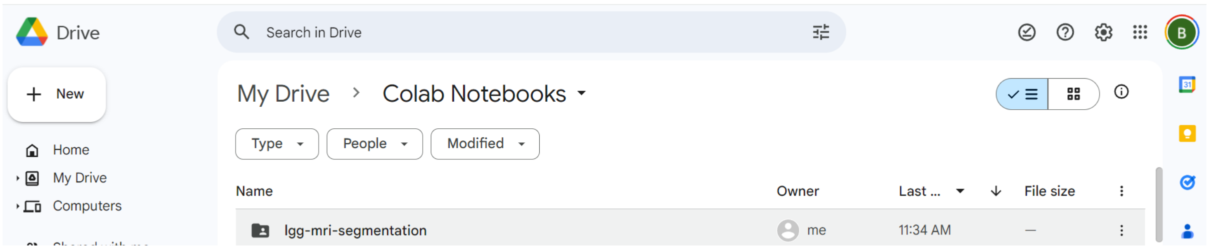
2.2 Software Configuration

In this part, you will find all the details about the software that was used, as well as its specifications. The right configuration of software and tools is crucial not only to make models run smoothly but also to ensure optimal performance during training and evaluation. In this regard, there are several subsections that will help you set up Google Drive and Google Colab, install required libraries, and configure data.

2.2.1 Google Drive

Google Drive stores input files such as model files, historical records for training, among others. To ease access while training, consider uploading your zipped input data file to Google Drive while noting down its path. Besides, it is important to use the same Google Drive account which is linked with your Google Colab Pro subscription so that no accessing issues arise. Proper filing within Google Drive can also make it easier for it to be integrated smoothly with Google Colab.

NOTE:You need to manually upload the file in Google Drive.

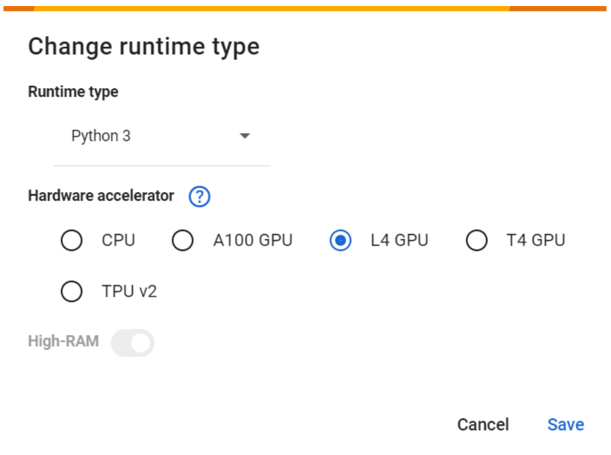


Name	Owner	Last ...	File size
lgg-mri-segmentation	me	11:34 AM	—

Figure 2: Google Drive File Upload

2.2.2 Google Colab

Because of GPUs being such a scarce resource, this study was carried out on a Google Colab Pro subscription that leveraged the 'L4 GPU' runtime. This configuration is sufficient for efficiently handling deep learning tasks. The following are the steps to mount Google Drive in Google Colab, load data, and start running the code.



Change runtime type

Runtime type: Python 3

Hardware accelerator: CPU A100 GPU [✓ L4 GPU] T4 GPU TPU v2

High-RAM

Figure 3: Colab Runtime Setup

2.2.3 Python 3.10

For this research, the Python version used was 3.10, and a `requirements.txt` file is in the artifacts folder, which lists all dependencies needed by the environment. You need to manually install the `albumentations` 0.4.6 library.

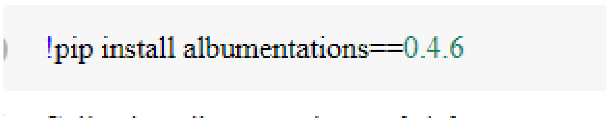
All the other packages, such as CUDA 12.2, TensorFlow 2.17.0, Keras 3.4.1, OpenCV, scikit-learn, and matplotlib, are pre-installed. It's just required to import them. The `requirements.txt` file is in the artifacts folder, which lists all dependencies needed by the environment.

3 Execution of Python Files

NOTE: you can only run one file at a time in Google since it is computer intensive.

3.1 Installing Libraries

Install albumentation package for data augmentation

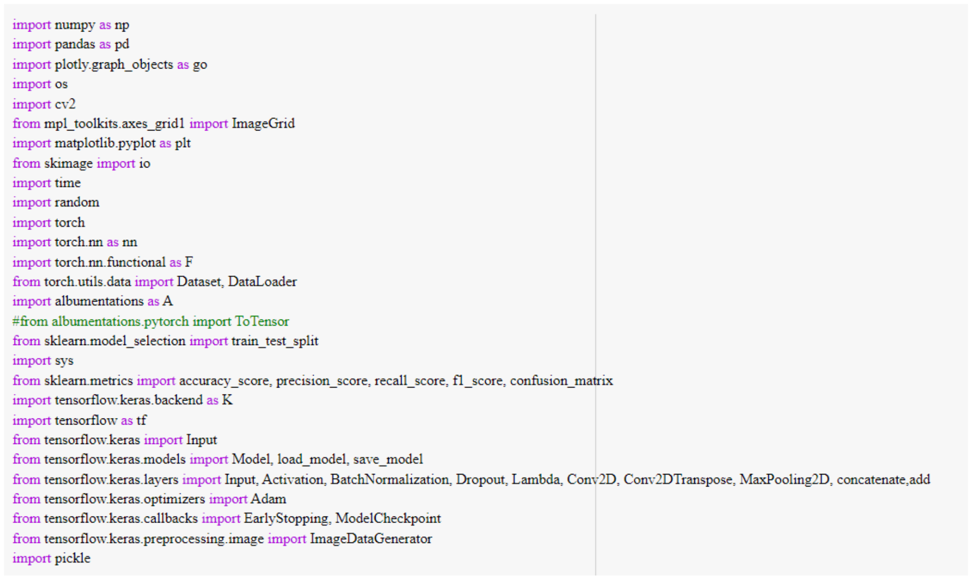


```
!pip install albumentations==0.4.6
```

Figure 4: Package Installation

3.2 Importing Libraries and setting up global varibles

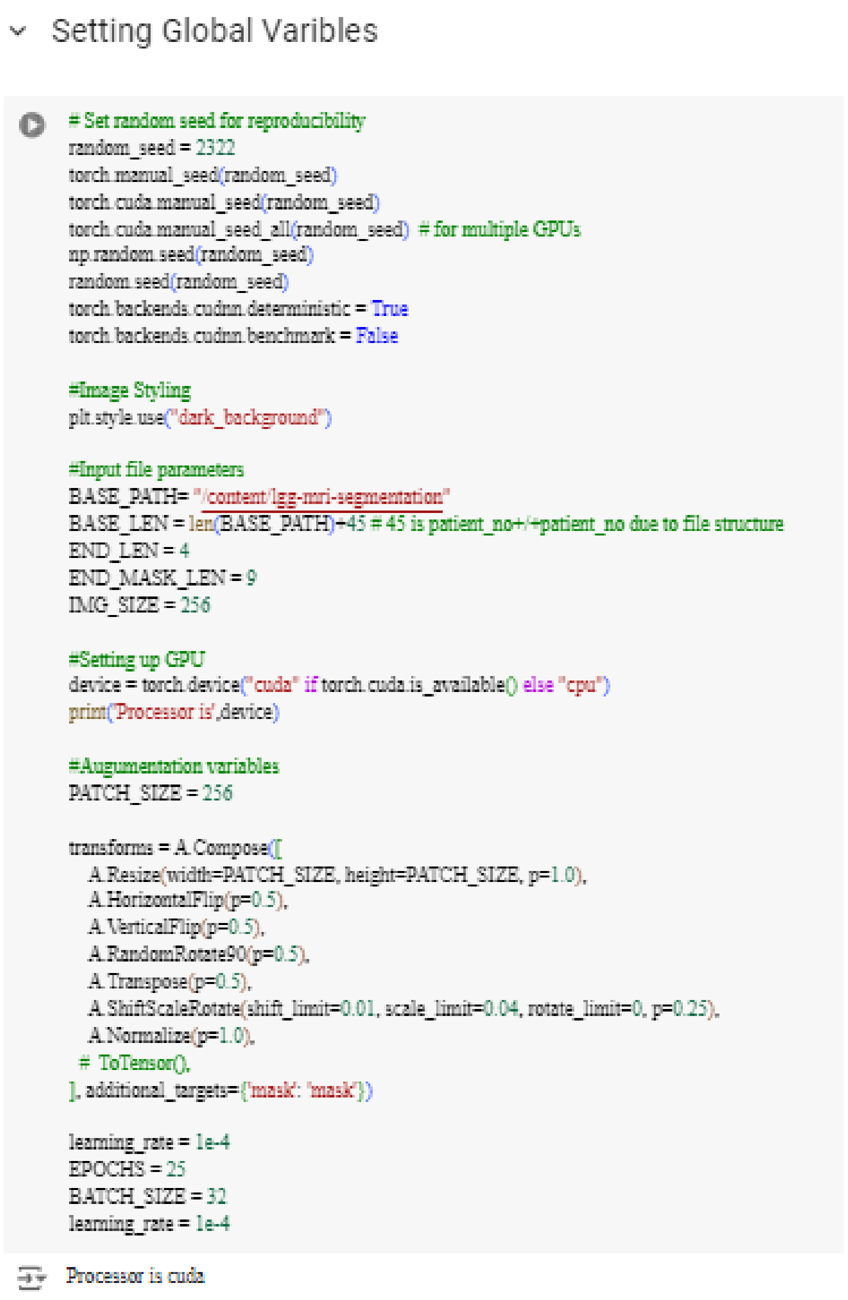
Most of the libraries are preinstalled hence, as required importing them based on the dependencies



```
import numpy as np
import pandas as pd
import plotly.graph_objects as go
import os
import cv2
from mpl_toolkits.axes_grid1 import ImageGrid
import matplotlib.pyplot as plt
from skimage import io
import time
import random
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
import albumentations as A
#from albumentations.pytorch import ToTensor
from sklearn.model_selection import train_test_split
import sys
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
import tensorflow.keras.backend as K
import tensorflow as tf
from tensorflow.keras import Input
from tensorflow.keras.models import Model, load_model, save_model
from tensorflow.keras.layers import Input, Activation, BatchNormalization, Dropout, Lambda, Conv2D, Conv2DTranspose, MaxPooling2D, concatenate,add
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import pickle
```

Figure 5: Import Libraries

Setting up configuration parameters or variables


Setting Global Varibles

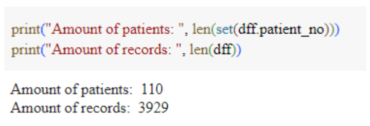
```
# Set random seed for reproducibility
random_seed = 2322
torch.manual_seed(random_seed)
torch.cuda.manual_seed(random_seed)
torch.cuda.manual_seed_all(random_seed)  # for multiple GPUs
np.random.seed(random_seed)
random.seed(random_seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False

#Image Styling
plt.style.use("dark_background")

#Input file parameters
BASE_PATH= "/content/lgg-mri-segmentation"
BASE_LEN = len(BASE_PATH)+45 # 45 is patient_no+/+patient_no due to file structure
END_LEN = 4
END_MASK_LEN = 9
IMG_SIZE = 256

#Setting up GPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print('Processor is',device)

#Augumentation variables
PATCH_SIZE = 256

transforms = A.Compose([
   A.Resize(width=PATCH_SIZE, height=PATCH_SIZE, p=1.0),
   A.HorizontalFlip(p=0.5),
   A.VerticalFlip(p=0.5),
   A.RandomRotate90(p=0.5),
   A.Transpose(p=0.5),
   A.ShiftScaleRotate(shift_limit=0.01, scale_limit=0.04, rotate_limit=0, p=0.25),
   A.Normalize(p=1.0),
  # ToTensor(),
], additional_targets={'mask': 'mask'})

learning_rate = 1e-4
EPOCHS = 25
BATCH_SIZE = 32
learning_rate = 1e-4
```

```
Processor is cuda
```

Figure 6: Global variable definition

3.3 Data Preprocessing

Performing the initial data loading cleaning sanity cheques data analysis and augmentation in order to prepare them for modelling



```
print("Amount of patients: ", len(set(dff.patient_no)))
print("Amount of records: ", len(dff))
```

```
Amount of patients:  110
Amount of records:  3929
```

Figure 7: Record Validation

Before performing data split, we need to cheque the distribution of images in order to avoid bias

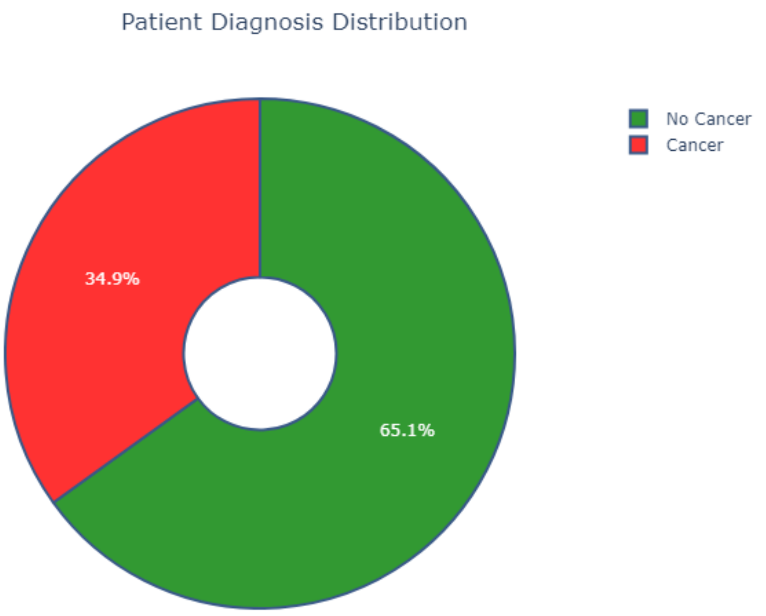
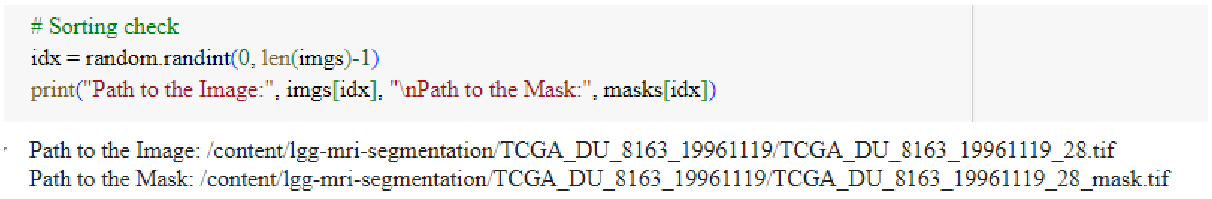


Figure 8: Patient Distribution

Performing data sanity cheque by checking if all images have their corresponding mask present.



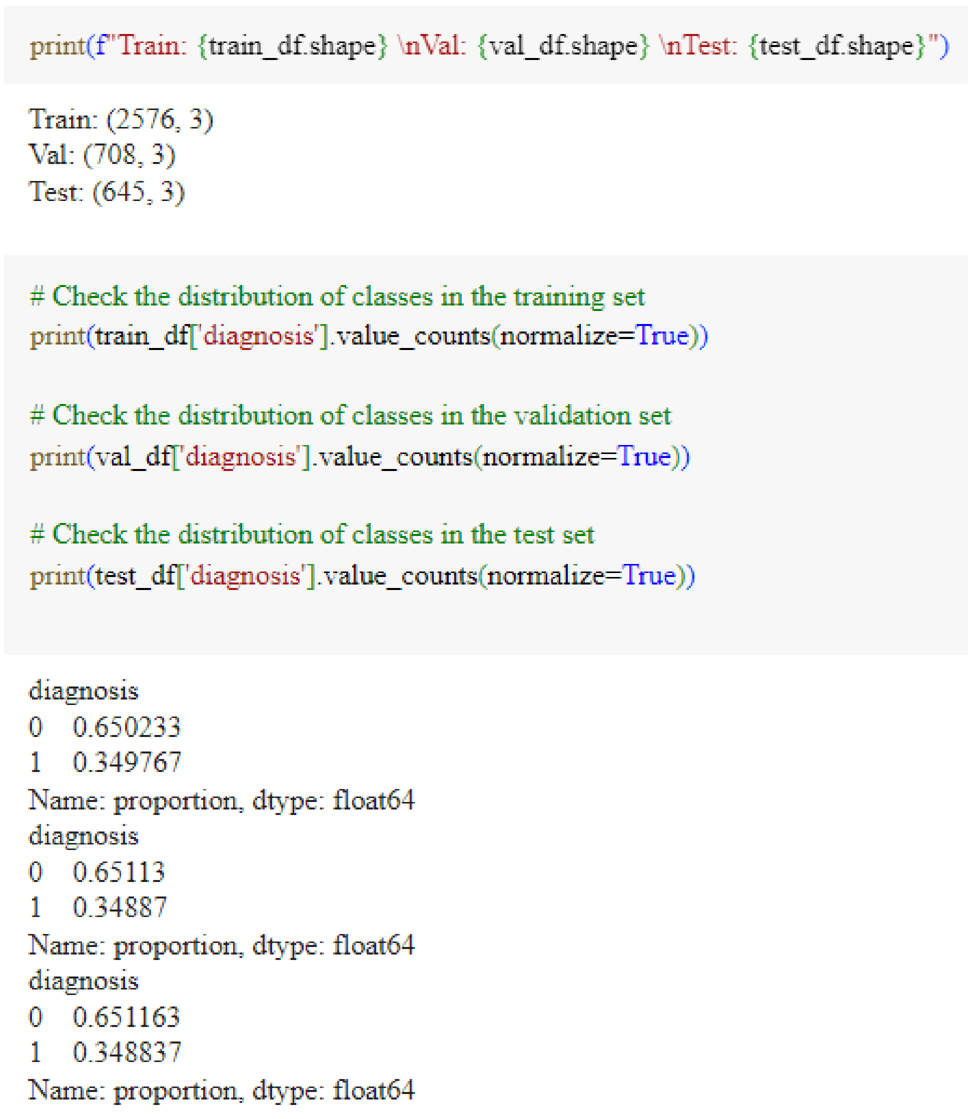
```
# Sorting check
idx = random.randint(0, len(imgs)-1)
print("Path to the Image:", imgs[idx], "\nPath to the Mask:", masks[idx])
```

```
Path to the Image: /content/lgg-mri-segmentation/TCGA_DU_8163_19961119/TCGA_DU_8163_19961119_28.tif
Path to the Mask: /content/lgg-mri-segmentation/TCGA_DU_8163_19961119/TCGA_DU_8163_19961119_28_mask.tif
```

Figure 9: Sanity Check

3.4 Data Split

In order to train the data and then validate it. Will split the data accordingly.



```
print(f"Train: {train_df.shape} \nVal: {val_df.shape} \nTest: {test_df.shape}")
```

```
Train: (2576, 3)
Val: (708, 3)
Test: (645, 3)
```

```
# Check the distribution of classes in the training set
print(train_df['diagnosis'].value_counts(normalize=True))

# Check the distribution of classes in the validation set
print(val_df['diagnosis'].value_counts(normalize=True))

# Check the distribution of classes in the test set
print(test_df['diagnosis'].value_counts(normalize=True))
```

```
diagnosis
0    0.650233
1    0.349767
Name: proportion, dtype: float64
diagnosis
0    0.65113
1    0.34887
Name: proportion, dtype: float64
diagnosis
0    0.651163
1    0.348837
Name: proportion, dtype: float64
```

Figure 10: Data Splitting

3.5 Model Configuration

It usually involves and entails the specification of deep learning model topology that will be used. This section includes code snippets for setting up the U-Net architecture, Attention U-Net, and ResU-Net. In principle, there are differences between the models, and their characteristics are derived from the structure of the architecture which influences the efficiency of the segmentation tasks. The configuration of these models is important as it determines how well they will be trained and the quality of the segmentation outcomes.

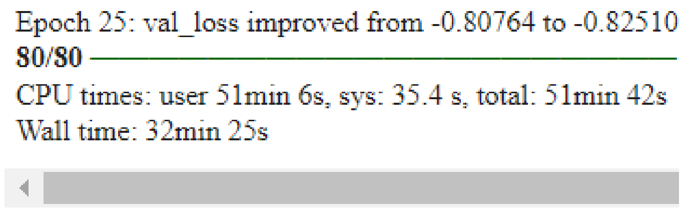
3.5.1 Hyperparameter Settings

- **Learning Rate**: 1e-4
- **Batch Size**: 32
- **Dropout Rate**: 0.3

- **Epochs**: 25
- **Resize Image**: 128*128
- **Optimizer**: Adam
- **Loss Function**: Cross-Entropy Loss

3.5.2 U-Net Model Configuration

U-Net is popular for image segmentation problems because of the encoder-decoder structure connected with skip connections. The following is a list of codes that form the core of the U-Net model and, once compiled, will help in training from your dataset Ronneberger et al. (2015)

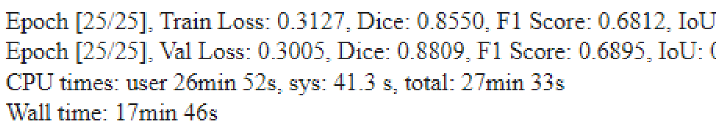


```
Epoch 25: val_loss improved from -0.80764 to -0.82510
80/80 ────────────────────────────────────────
CPU times: user 51min 6s, sys: 35.4 s, total: 51min 42s
Wall time: 32min 25s
```

Figure 11: UNet Model Training Time

3.5.3 Attention U-Net: Model Configuration

Attention U-Net model is an improved version of the U-Net model where attention mechanisms have been applied. They are only able to attend to specific features in the input data, which could enhance the segmentation performance. Below you can see the code to define and compile the Attention U-Net model Oktay et al. (2018)

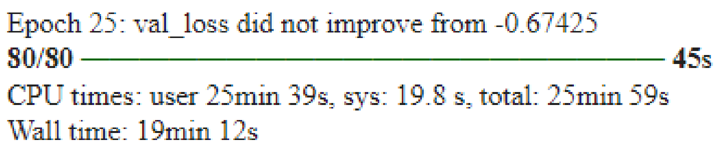


```
Epoch [25/25], Train Loss: 0.3127, Dice: 0.8550, F1 Score: 0.6812, IoU
Epoch [25/25], Val Loss: 0.3005, Dice: 0.8809, F1 Score: 0.6895, IoU: (
CPU times: user 26min 52s, sys: 41.3 s, total: 27min 33s
Wall time: 17min 46s
```

Figure 12: Attention UNet Model Training Time

3.5.4 ResU-Net Model Configuration

The proposed ResU-Net model is based on the pre-existing model of U-Net which contains the idea of residual connections to increase stability in the functioning of neural networks. Essential in this segment is the code that will help in the creation of the ResU-Net model that has been pre-coded and developed to undergo training and testing with your dataset He et al. (2016)

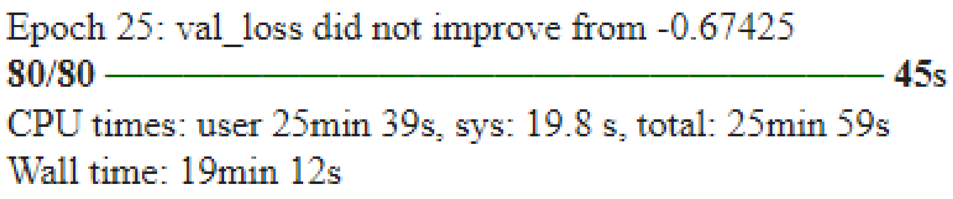


```
Epoch 25: val_loss did not improve from -0.67425
80/80 ──────────────────────────────── 45s
CPU times: user 25min 39s, sys: 19.8 s, total: 25min 59s
Wall time: 19min 12s
```

Figure 13: ResUNet Model Training Time

3.5.5 Retaining Model

Google collapse pro runtime involvements deletes on the same data. Hence, we need to copy data back to drive in order to access the train model to future tests.

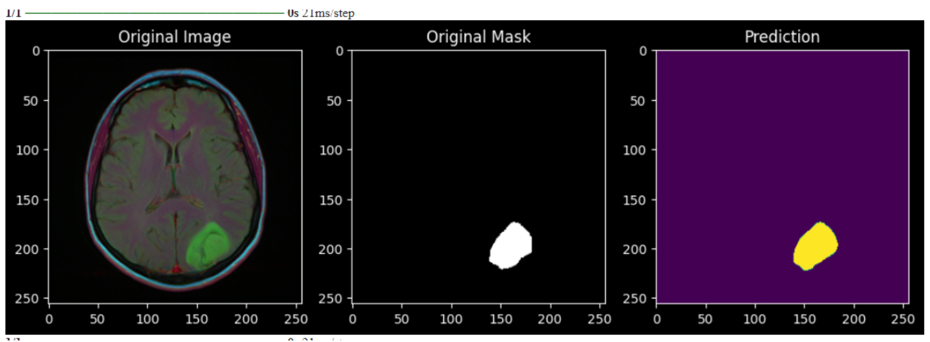


```
Epoch 25: val_loss did not improve from -0.67425
80/80 ──────────────────────────────── 45s
CPU times: user 25min 39s, sys: 19.8 s, total: 25min 59s
Wall time: 19min 12s
```

Figure 14: Saving Model to Google Drive

3.6 Model Testing

After training the model thoroughly in fine tuning the parameters we use test data to validate the output generated by the model.



```
1/1 ──────────────── 0s 21ms/step
```

Figure 15: Model Prediction

References

He, K., Zhang, X., Ren, S. and Sun, J. (2016). Deep residual learning for image recognition, *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.
URL: *https://doi.org/10.1109/CVPR.2016.90*

Oktay, O., Schlemper, J., Folgoc, L. L., Lee, M., Heinrich, M., Misawa, K., Mori, K., McDonagh, S., Hammerla, N. Y., Kainz, B., Glocker, B. and Rueckert, D. (2018). Attention u-net: Learning where to look for the pancreas, *arXiv preprint arXiv:1804.03999*

.
URL: *https://arxiv.org/abs/1804.03999*

Ronneberger, O., Fischer, P. and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation, *arXiv preprint arXiv:1505.04597* .
URL: *https://arxiv.org/abs/1505.04597*