National College of Ireland

Configuration Manual

Microvascular Structure Segmentation in Human Tissue Using Deep Learning
Data Analytics

Sarvochch Balwant Singh
Student ID: 22163891

School of Computing
National College of Ireland

Supervisor: Prof. Noel Cosgrave

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sarvochch Balwant Singh
Student ID:	22163891
Programme:	Data Analytics
Year:	2023
Module:	Microvascular Structure Segmentation in Human Tissue Using Deep Learning
Supervisor:	Prof. Noel Cosgrave
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	774
Page Count:	10

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:	
Date:	14th December 2023

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

Sarvochch Balwant Singh
22163891

1 Local Machine System Configuration

Device name DESKTOP-QV0O42F
Processor Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz 2.20 GHz
Installed RAM 8.00 GB (7.85 GB usable)
System type 64-bit operating system, x64-based processor
Pen and touch No pen or touch input is available for this display

Os Specification
Edition Windows 11 Home Single Language
Version 22H2
OS build 22621.2861
Experience Windows Feature Experience Pack 1000.22681.1000.0

2 Kaggle notebook Specifications

Kaggle's notebook specification, data retrieved on 14/12/2023 from
https://www.kaggle.com/docs/notebooks
12 hours execution time for CPU and GPU notebook sessions and 9 hours for TPU notebook sessions
20 Gigabytes of auto-saved disk space (/kaggle/working)
Additional scratchpad disk space (outside /kaggle/working) that will not be saved outside of the current session
CPU Specifications
4 CPU cores
30 Gigabytes of RAM
P100 GPU Specifications
1 Nvidia Tesla P100 GPU
4 CPU cores
29 Gigabytes of RAM

3 Data Collection

Data has been collected from kaggle, the name of the dataset is - HuBMAP - Hacking the Human Vasculature, its a open source dataset anyone can access and use it.

1

4 Importing Libraries

Figure 1 Importing the necessary libraries, If any libraries are not installed on the given environment please write "pip install 'library_name'"

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import shutil
from collections import Counter
from typing import Callable
import random

from PIL import Image
import imageio
import json
import cv2
import ipywidgets as widgets
import IPython.display as ipd
import glob
import logging
import functools

import base64
import numpy as np
import typing as t
import zlib

import tensorflow as tf
from keras import backend as K
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Conv2D, Conv2DTranspose, MaxPooling2D, Dropout, concatenate
from tensorflow import keras
from tensorflow.keras import layers
from scipy.ndimage import label
from scipy.ndimage import find_objects

import warnings
warnings.simplefilter("ignore")

import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
import cv2
from scipy.ndimage import label
from scipy.ndimage import find_objects
```

Figure 1: Image of result generated by the model on validation set

5 Data Visualization

Figure 2 shows the function to create masks for visualization

```
def create_mask(image_id, shape, polygons):

    masks[image_id] = {annotation_type: {'count': 0, 'area': 0, 'sum_sq_area': 0, 'image':np.zeros(shape, dtype=np.uint8)} for annotation_type in annotation_types}
    # For each polygon
    for polygon in polygons:
        annotation_type = polygon['type']
        lines = np.array(polygon['coordinates'])
        lines = lines.reshape(-1, 1, 2)
        # Draw the polygon on the mask
        mask = masks[image_id][annotation_type]['image']
        cv2.fillPoly(mask, [lines], 1)

    for t in annotation_types:
        mask = masks[image_id][t]['image']
        labels, n_labels = label(mask)
        f = find_objects(labels)
        total_area = 0
        instances = []
        for i in range(n_labels):
            instance_area = np.sum(labels[f[i]]//(i+1))
            instances.append(instance_area)
            total_area += instance_area

        masks[image_id][t]['count'] = n_labels
        masks[image_id][t]['area'] = total_area
        masks[image_id][t]['sum_sq_area'] = np.sum(np.array(instances)**2)
```

Figure 2: Image of function to create masks for visualization

Figure 3 shows the function to get annotations from the jsonl file.
Figure 4 shows the function plot the mask, image and overlay.

[1]Dataset `https://www.kaggle.com/competitions/hubmap-hacking-the-human-vasculature/data`

```
import json

annotated_images = list(images.keys())
train_images_path = os.path.join(BASE_DIR, 'train')
with open(os.path.join(BASE_DIR, 'polygons.jsonl'), 'r') as file:
    for line in file:
        json_data = json.loads(line)
        image_id = json_data['id']

        # Load image
        fn = os.path.join(train_images_path, image_id + '.tif')

        if not os.path.exists(fn):
            print(f"Error: File not found - {fn}")
            continue

        img = cv2.imread(fn, cv2.IMREAD_UNCHANGED)
        images[image_id] = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        shape = img.shape[:2]

        # Check image shape
        if shape != (IMG_SIZE, IMG_SIZE):
            print(f"Error: Image shape mismatch for {image_id}. Expected ({IMG_SIZE}, {IMG_SIZE}), got {shape}.")
            continue  # Skip to the next iteration if the image shape is incorrect

        del img
        create_mask(image_id, shape, json_data['annotations'])

annotated_images = list(images.keys())
```

Figure 3: Image of function to create masks for visualization

```
import matplotlib.patches as mpatches

def display_image_and_masks(image_id):
    if image_id not in images.keys():
        return

    image = images[image_id]
    mask = masks[image_id]
    combined_mask = np.zeros(image.shape[:2])
    overlay = image.copy()
    colors = [(255, 0, 0), (0, 0, 255), (0, 206, 209)]
    colorsRBG = [tuple(c / 255 for c in tup) for tup in colors]

    annotation_counts = {}
    for i, a_t in enumerate(mask.keys()):
        annotation_counts[a_t] = mask[a_t]['count']
        overlay[mask[a_t]['image'] > 0] = colors[i]
        combined_mask += mask[a_t]['image'] * (i+1)

    print(f"Annotation counts for image {image_id}: {annotation_counts}")

    plt.figure(figsize=(25,25))

    # original image
    plt.subplot(1,3,1)
    plt.imshow(image)
    plt.title('Image' + f' {image_id}')
    plt.axis('off')
    # mask
    plt.subplot(1,3,2)
    plt.imshow(combined_mask, cmap='bone')
    plt.title('All annotations mask')
    plt.axis('off')
    # overlay
    plt.subplot(1,3,3)
    plt.imshow(image)
    plt.imshow(overlay, alpha=0.4)

    patches = [mpatches.Patch(color=color, label=label) for color, label in zip(colorsRBG, list(annotation_counts.keys()))]
    legend = plt.legend(handles=patches)

    plt.title('Overlay')
    plt.axis('off')
    plt.show()

image_ids = ['097dd2ed6c14', '075d12b2917', '0065bc6f2698']
for im_id in image_ids:
    display_image_and_masks(im_id)
```

Figure 4: Image of function plot the mask, image and overlay.

6 Data Preprocessing

Figure 5 shows the function to load file path and function to split the data in train and validation.

```
def load_file_paths(directory_path, examples, file_extension = '.tif'):
    file_paths = [directory_path + '/' + id + file_extension for id in examples]
    return file_paths

def get_train_valid_pats(directory_path, examples, train_split_size=0.8):
    file_paths = load_file_paths(directory_path, examples)

    split_index = int(len(file_paths) * train_split_size)

    # Split the list into training and testing sets
    train_files = file_paths[:split_index]
    valid_files = file_paths[split_index:]

    return train_files, valid_files

train_directory_path = '/kaggle/input/hubmap-hacking-the-human-vasculature/train'
train_file_paths, validation_file_paths = get_train_valid_pats(train_directory_path, training_examples)
```

Figure 5: function to load file path and function to split the data in train and validation.

Figure 6 shows the function to get the image gets it's id and mask.

```
def get_image(path):
    image = cv2.imread(path)
    image = image/255
    image = image.astype(np.float32)
    return image
```

```
def get_id(string_path):
    parts = string_path.split("/")
    file_name = parts[-1]
    identity = file_name.split(".")[0]
    return identity
```

```
def get_mask(path):
    image_id = get_id(path)

    mask = np.zeros((512, 512, 1), dtype=np.uint8)
    annots = polygons_df.loc[polygons_df["id"] == image_id, 'annotations'].iloc[0]
    for annot in annots:
        annot_type = annot['type']
        coordinates = annot['coordinates']
        if annot_type == 'blood_vessel':
            cv2.fillPoly(mask[:, :, 0], [np.array(coordinates)], (255,))
    mask = mask/255
    mask = mask.astype(np.float32)
    return mask
```

Figure 6: Image of the function to get the image gets it's id and mask.

Figure 7 shows the function map image and mask.

```
def map_function(path):
    def preprocess(path):
        path = path.decode()

        image = get_image(path)
        mask = get_mask(path)

        return image, mask

    image, mask = tf.numpy_function(preprocess, [path], [tf.float32, tf.float32])
    image.set_shape([512,512,3])
    mask.set_shape([512,512,1])

    return image, mask
```

Figure 7: Image of the function map image and mask.

Figure 8 shows the function to crop flip and rotate.

```python
def crop_flip_rotate(image, mask):
    num_channels_image = 3
    num_channels_mask = 1

    combined = tf.concat([image, mask], axis=-1)

    combined = tf.image.random_flip_left_right(combined)
    combined = tf.image.random_flip_up_down(combined)

    combined = tf.image.rot90(combined, tf.random.uniform(shape=[], minval=0, maxval=4, dtype=tf.int32))

    image = combined[:, :, :num_channels_image]
    mask = combined[:, :, num_channels_image:]

    return image, mask
```

Figure 8: Image of the function to crop flip and rotate.

Figure 9 shows the function for mosaic augmentation.

```python
def mosaic_augmentation(image, mask, image_paths=train_file_paths):
    num_images = 4
    output_size = (512, 512)

    random.shuffle(image_paths)

    selected_paths = image_paths[:num_images]

    images = []
    masks = []
    for path in selected_paths:
        image = get_image(path)
        mask = get_mask(path)
        images.append(image)
        masks.append(mask)

    num_channels = images[0].shape[-1]

    mosaic = np.hstack(images[:2]), np.hstack(images[2:])
    mosaic = np.vstack(mosaic)

    mask = np.hstack(masks[:2]), np.hstack(masks[2:])
    mask = np.vstack(mask)

    combined = tf.concat([mosaic, mask], axis=-1)

    cropped_combined = tf.image.random_crop(combined, size=[output_size[0], output_size[1], combined.shape[-1]])

    image = cropped_combined[:, :, :num_channels]
    mask = cropped_combined[:, :, num_channels:]

    image.set_shape([512,512,3])
    mask.set_shape([512,512,1])
    return image, mask
```

Figure 9: Image of the function for mosaic augmentation.

Figure 10 shows the function get the main dataset after all the preprocessing and batching of the dataset.

```python
def dataset(file_paths, batch_size=16, augment=True):
    dataset = tf.data.Dataset.from_tensor_slices(file_paths)
    dataset = dataset.map(map_function, num_parallel_calls=tf.data.AUTOTUNE)

    if augment:
        mosaic_dataset = dataset.map(mosaic_augmentation, num_parallel_calls=tf.data.AUTOTUNE)
        rotated_dataset = dataset.map(crop_flip_rotate, num_parallel_calls=tf.data.AUTOTUNE)

        dataset = dataset.concatenate(mosaic_dataset)
        dataset = dataset.concatenate(rotated_dataset)

    dataset = dataset.batch(batch_size)
    return dataset
```

+ Code + Markdown

```python
BATCH_SIZE= 8
TRAIN_LENGTH = len(train_file_paths)
VAL_LENGTH = len(validation_file_paths)
train_ds = dataset(train_file_paths, BATCH_SIZE, augment=True)
validation_ds = dataset(validation_file_paths, BATCH_SIZE, augment=False)
```

Figure 10: Image of the function get the main dataset after all the preprocessing and batching of the dataset.

Figure 11 shows the function to calculate dice coeff and writing the number of Epochs.

```python
EPOCHS = 50
```

```python
def dice_coeff(y_true, y_pred):
    smooth = 1.0
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2.0 * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smooth)

def dice_loss(y_true, y_pred):
    return 1.0 - dice_coeff(y_true, y_pred)
```

Figure 11: Image of the function to calculate dice coeff and writing the number of Epochs.

Figure 12 shows the code that adjusts the learning rate and implies earlystoping.

```python
import tensorflow as tf
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers.schedules import ExponentialDecay

initial_learning_rate = 0.0001
decay_steps = 1000
decay_rate = 0.9

lr_schedule = ExponentialDecay(
    initial_learning_rate=initial_learning_rate,
    decay_steps=decay_steps,
    decay_rate=decay_rate,
    staircase=True
)

early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=5,
    restore_best_weights=True
)
```

Figure 12: Image of the code that adjusts the learning rate and implies earlystoping.

7 Model DeeplabV3+

Figure 13 shows the Convolutional block of DeeplabV3+ model.

```
def convolution_block(
    block_input,
    num_filters=256,
    kernel_size=3,
    dilation_rate=1,
    padding="same",
    use_bias=False,
):
    x = layers.Conv2D(
        num_filters,
        kernel_size=kernel_size,
        dilation_rate=dilation_rate,
        padding="same",
        use_bias=use_bias,
        kernel_initializer=keras.initializers.HeNormal(),
    )(block_input)
    x = layers.BatchNormalization()(x)
    return tf.nn.relu(x)
```

Figure 13: Image of the Convolutional block of DeeplabV3+ model.

Figure 14 shows the DSPP block of DeeplabV3+ model.

```
def DilatedSpatialPyramidPooling(dspp_input):
    dims = dspp_input.shape
    x = layers.AveragePooling2D(pool_size=(dims[-3], dims[-2]))(dspp_input)
    x = convolution_block(x, kernel_size=1, use_bias=True)
    out_pool = layers.UpSampling2D(
        size=(dims[-3] // x.shape[1], dims[-2] // x.shape[2]), interpolation="bilinear",
    )(x)

    out_1 = convolution_block(dspp_input, kernel_size=1, dilation_rate=1)
    out_6 = convolution_block(dspp_input, kernel_size=3, dilation_rate=6)
    out_12 = convolution_block(dspp_input, kernel_size=3, dilation_rate=12)
    out_18 = convolution_block(dspp_input, kernel_size=3, dilation_rate=18)

    x = layers.Concatenate(axis=-1)([out_pool, out_1, out_6, out_12, out_18])
    output = convolution_block(x, kernel_size=1)
    return output
```

Figure 14: Image of the DSPP block of DeeplabV3+ model.

Figure 15 shows the Deeplabv3plus block of DeeplabV3+ model.

```
def DeeplabV3Plus(image_size, num_classes):
    model_input = keras.Input(shape=(image_size, image_size, 3))
    resnet50 = keras.applications.ResNet50(
        weights= None, include_top=False, input_tensor=model_input
    )
    x = resnet50.get_layer("conv4_block6_2_relu").output
    x = DilatedSpatialPyramidPooling(x)

    input_a = layers.UpSampling2D(
        size=(image_size // 4 // x.shape[1], image_size // 4 // x.shape[2]),
        interpolation="bilinear",
    )(x)
    input_b = resnet50.get_layer("conv2_block3_2_relu").output
    input_b = convolution_block(input_b, num_filters=48, kernel_size=1)

    x = layers.Concatenate(axis=-1)([input_a, input_b])
    x = convolution_block(x)
    x = convolution_block(x)
    x = layers.UpSampling2D(
        size=(image_size // x.shape[1], image_size // x.shape[2]),
        interpolation="bilinear",
    )(x)
    model_output = layers.Conv2D(num_classes, kernel_size=(1, 1), padding="same", activation='sigmoid')(x)
    return keras.Model(inputs=model_input, outputs=model_output)
```

Figure 15: Image of the Deeplabv3plus block of DeeplabV3+ model.

Figure 16 shows the execution of DeeplabV3+ model.

```
model = DeeplabV3Plus(image_size=512, num_classes=1)

learning_rate=0.0001
optimizer = tf.keras.optimizers.Adam(lr_schedule)
loss = tf.keras.losses.BinaryCrossentropy()

model.compile(optimizer=optimizer,
              loss=[dice_loss],
              metrics=['accuracy',dice_coeff])

model_Deeplab = model.fit(train_ds, epochs=EPOCHS, validation_data=validation_ds,callbacks=[early_stopping])
```

Figure 16: Image of the execution block of DeeplabV3+ model.

8 Model U-Net

Figure 17 shows the convolutional, encoder, decoder and Unet block of the of U-Net model.

```
def conv_block(input, num_filters):
    conv = tf.keras.layers.Conv2D(num_filters, 3, padding="same")(input)
    conv = tf.keras.layers.BatchNormalization()(conv)
    conv = tf.keras.layers.Activation("relu")(conv)
    conv = tf.keras.layers.Conv2D(num_filters, 3, padding="same")(conv)
    conv = tf.keras.layers.BatchNormalization()(conv)
    conv = tf.keras.layers.Activation("relu")(conv)
    return conv

def encoder_block(input, num_filters):
    skip = conv_block(input, num_filters)
    pool = tf.keras.layers.MaxPool2D((2, 2))(skip)
    return skip, pool

def decoder_block(input, skip, num_filters):
    up_conv = tf.keras.layers.Conv2DTranspose(num_filters, (2, 2), strides=2, padding="same")(input)
    conv = tf.keras.layers.Concatenate()([up_conv, skip])
    conv = conv_block(conv, num_filters)
    return conv

def Unet(input_shape):
    inputs = tf.keras.layers.Input(input_shape)

    skip1, pool1 = encoder_block(inputs, 64)
    skip2, pool2 = encoder_block(pool1, 128)
    skip3, pool3 = encoder_block(pool2, 256)
    skip4, pool4 = encoder_block(pool3, 512)

    bridge = conv_block(pool4, 1024)

    decode1 = decoder_block(bridge, skip4, 512)
    decode2 = decoder_block(decode1, skip3, 256)
    decode3 = decoder_block(decode2, skip2, 128)
    decode4 = decoder_block(decode3, skip1, 64)

    outputs = tf.keras.layers.Conv2D(1, 1, padding="same", activation="sigmoid")(decode4)

    model = tf.keras.models.Model(inputs, outputs, name="U-Net")
    return model

unet_model = Unet((512, 512, 3))
```

Figure 17: Image of convolutional, encoder, decoder and Unet block of the of U-Net model.

Figure 18 shows the execution block of U-Net model.

```
learning_rate=0.001
optimizer = tf.keras.optimizers.Adam(lr_schedule)
loss = tf.keras.losses.BinaryCrossentropy()

unet_model.compile(optimizer=optimizer,
              loss=[dice_loss],
              metrics=['accuracy', dice_coeff])

model_Unet = unet_model.fit(train_ds, epochs=EPOCHS, validation_data=validation_ds,callbacks=[early_stopping])
```

Figure 18: Image of the execution block of U-Net model.

9 Post Processing and Model Testing

Figure 19 shows the script for steps taken for post processing.

```
for image, mask in validation_ds.take(1):
    pass
```

```
CompasableFunction = Callable[[np.array], np.array]

def compose(*functions: CompasableFunction) -> CompasableFunction:
    return functools.reduce(lambda f, g: lambda x: g(f(x)), functions)
```

```
def dilate_image(predictions: np.array, kernel: tuple = (2,2), dilation_iterations: int = 1):
    kernel = np.ones(kernel, np.uint8)
    dilated_images = []

    for prediction in predictions:
        dilated_image = cv2.dilate(prediction, kernel, iterations=dilation_iterations)
        dilated_images.append(dilated_image)

    return dilated_images

def get_prediction(image, model):
    predictions = model.predict(image)
    return predictions

def process_prediction(predictions: np.array, *preprocessing: CompasableFunction) -> np.array:
    prediction_processing = compose(*preprocessing)
    proccessed_pred = prediction_processing(predictions)
    return proccessed_pred
```

Figure 19: Image of script for steps taken for post processing.

Figure 20 shows the functions to calculate the average dice coefficient and function to plot the predection.

```
def calculate_average_Dice(mask, predictions):
    iou = 0
    for i in range(0, len(predictions)):
        iou += dice_coeff(mask[i], predictions[i].astype(np.float32)).numpy()

    return iou/len(predictions)
```

```
def plot_predictions(image, mask, predictions):
    num_images = image.shape[0]
    fig, axs = plt.subplots(num_images, 3, figsize=(12, 3 * num_images))

    for i in range(num_images):

        iou = calculate_average_Dice(mask[i], predictions[i].astype(np.float32))

        axs[i, 0].imshow(predictions[i], cmap='gray')
        axs[i, 0].set_title(f'Prediction, Dice: {iou:.3f}')

        axs[i, 1].imshow(image[i])
        axs[i, 1].set_title('Image')

        axs[i, 2].imshow(mask[i], cmap='gray')
        axs[i, 2].set_title('Mask')

    plt.tight_layout()
    plt.subplots_adjust(hspace=0.4)
    plt.show()
```

Figure 20: Image of the execution block of U-Net model.

Figure 21 shows the functions to process the test image (Resize it) and get the prediction on the image and plot it.

```python
def Test_image(directory, model, plot=True):
    file_paths = [os.path.join(directory, path) for path in os.listdir(directory)]
    predictions = []
    image_ids = []
    for path in file_paths:
        image_id = os.path.splitext(os.path.basename(path))[0]
        image=cv2.imread(path)/255
        resized_image = cv2.resize(image, (512, 512), interpolation=cv2.INTER_AREA)
        if resized_image.shape[2] == 1:
            resized_image = cv2.cvtColor(resized_image, cv2.COLOR_GRAY2BGR)
        prediction = model.predict(np.expand_dims(resized_image, axis=0))[0]

        image_ids.append(image_id)
        predictions.append(prediction)

    if plot:
        fig,axs = plt.subplots(1, 2, figsize=(24, 8))
        axs[0].imshow(resized_image)
        axs[0].set_title(f"Image {image_id}")

        axs[1].imshow(prediction, cmap='gray')
        axs[1].set_title(f"U-net Prediction")

    df = pd.DataFrame({'id': image_ids, 'height': 512, 'width':512 ,'prediction_string': predictions})

    return df
```

Figure 21: Image of the execution block of U-Net model.

Figure 22 shows the functions to get the output of U-net model for the test image same is written for the DeeplabV3+ model .

```python
df = Test_image("/kaggle/input/hubmap-hacking-the-human-vasculature/test", unet_model)
df.head()
```

Figure 22: Image of the execution block of U-Net model.

References