Automated Image Segmentation using Deep Learning

A

Minor Project (CC3270)

Report

Submitted in the partial fulfilment of the requirement for the award

ofBachelor of Technology

in

Computer and Communication Engineering

By:

Meghaj Garg

209303226

Under the guidance of:

Prof. Sunil Kumar

MANIPAL UNIVERSITY
JAIPUR

April,2023

Department of Computer and Communication EngineeringSchool of Computer and Communication Engineering Manipal University Jaipur

VPO. Dehmi Kalan, Jaipur, Rajasthan, India – 303007

STUDENT DECLARATION

I hereby declare that this project Automatic Image Segmentation using Deep Learning is my own work and that, to the bestof my knowledge and belief, it contains no material previously published or written by anotherperson nor material which has been accepted for the award of any other degree or diploma ofthe University or other Institute, except where due acknowledgements has been made in the text.

Place: Manipal University Jaipur

Date: 21 Apr. 2023

Meghaj Garg

209303226

B.Tech (CCE) 6th Semester

CERTIFICATE FROM GUIDE

*This is to certify that the work entitled "**Automated Image Segmentation using Deep Learning"** submitted by **Meghaj Garg** (209303226) to **Manipal University Jaipur** for the award of the degree of **Bachelor of Technology** in **Computer and Communication Engineering** is a Bonafede record of the workcarried out by him/ her under my supervision and guidance from January 2023 to April 2023.*

Prof. Sunil Kumar
Department of Computer and Communication EngineeringManipal University Jaipur

TABLE OF CONTENTS

ACKNOWLEDGEMENT

We would like to sincerely thank Prof. Sunil Kumar for giving us this opportunity and guiding us at every step of the same.

We would also like to thank School of CCE, Manipal University Jaipur for providing us with all the guidance and motivation to complete our tasks on time

Thank you

STATEMENT OF PURPOSE

As a computer science enthusiast, I have always been fascinated by the power of artificial intelligence and its ability to solve complex problems. Among the many applications of AI, computer vision has always stood out to me as an area with enormous potential to transform industries and improve people's lives. In particular, I am intrigued by the prospect of using deep learning to automate image segmentation, a task that has traditionally been both time-consuming and error-prone.

My interest in automatic image segmentation using deep learning stems from my desire to contribute to the development of cutting-edge technologies that can help solve real-world problems. With the increasing availability of large datasets and powerful computing resources, I believe that deep learning techniques can significantly improve the accuracy and speed of image segmentation tasks.

I am excited about the potential applications of automatic image segmentation using deep learning in fields such as medical imaging, autonomous driving, and object recognition. By automating the segmentation process, we can enable faster and more accurate diagnosis of medical conditions, improve the safety and efficiency of transportation systems, and enhance the performance of computer vision systems in a wide range of domains.

In pursuing this topic, I am eager to explore the latest research in deep learning and computer vision, as well as to develop my skills in implementing and optimizing deep learning models. I am confident that by working on automatic image segmentation using deep learning, I can make meaningful contributions to this rapidly evolving field and help to drive innovation anD progress in the broader AI community.

INTRODUCTION

Deep learning has become an active research topic in the field of medical image analysis. In particular, for the automatic segmentation of stemmatological images, great advances have been made in segmentation performance. In this paper, we will systematically review the recent literature on segmentation methods for stemmatological images based on deep learning, and their clinical applications. We will categorize them into different tasks and analyse their advantages and disadvantages. The main categories that we plan to explore are the data sources, backbone network, and task formulation. We categorized data sources into panoramic radiography, dental X-rays, cone-beam computed tomography, multi-slice spiral computed tomography, and methods based on intraoral scan images.

Imaging examinations, intraoral scanning, and other technologies are often required to assist diagnosis and treatment of diseases because of the complex structure of the oral and maxillofacial region and various types of diseases. Imaging examinations use dental X-rays, panoramic radiography, cone-beam computed tomography (CBCT), and multi-slice spiral computed tomography (MSCT). These are widely used in stomatology and produce large amounts of medical image data. Efficient and accurate processing of medical images is essential for the development of stomatology. The key task is image segmentation, which can realize the localization and the (qualitative and quantitative) analysis of lesions, help to design a treatment plan, and analyse the efficacy of the treatment. The traditional manual segmentation method is time-consuming, and the segmentation effect depends on the experience of the doctor, leading to an unsatisfactory result. Therefore, the application of modern image segmentation technology to stomatology is very important.

Deep learning (DL) is a branch of machine learning and is a promising method of achieving artificial intelligence. Owing to the availability of large-scale annotated data and powerful computing resources, DL-based medical image segmentation algorithms have achieved excellent performance. These methods have successfully assisted the accurate diagnosis and minimally invasive treatment of brain tumours, retinal vessels, pulmonary nodules, cartilage, and bon. This paper reviews current DL-based medical image segmentation methods and their applications in stomatology. Existing automatic segmentation algorithms are classified according to the data source, the form of the automatic segmentation task, and the structure of the backbone network of the algorithm. The feasibility, accuracy, and application prospects of these algorithms are comprehensively analysed, and their future research prospects are discussed.

METHODOLOGY/ PLANNIG OF WORK

- **Data collection and preparation:** This dataset should cover a wide range of object types, sizes, and orientations, to ensure that the trained model can generalize well to new images.

- **Model selection:** Once the dataset is prepared, the next step is to select an appropriate deep learning model for the task. This will depend on factors such as the size of the dataset, the complexity of the images, and the desired level of accuracy

- **Model training**: With the dataset and model selected, the next step is to train the model on the dataset. This involves feeding the images and corresponding segmentation masks into the network, adjusting the network's weights based on the difference between the predicted segmentation and the ground truth, and iteratively repeating the process until the network reaches a satisfactory level of accuracy.

- **Model evaluation**: Once the model is trained, the next step is to evaluate its performance on a separate test set of images. This will help to determine how well the model generalizes to new images and whether any adjustments need to be made to improve its performance.

- **Model deployment**: Once the model has been trained and evaluated, the final step is to deploy it in a production environment. This may involve integrating it into a larger computer vision system or developing a standalone application that can perform image segmentation on new images in real-time.

FACILITIES REQUIRED

- A computer with a dedicated GPU (Graphics Processing Unit) with at least 4 GB of VRAM (Video Random Access Memory) for faster training and inference of deep learning models
- Sufficient RAM (Random Access Memory) to accommodate large image datasets and deep learning models
- Sufficient storage space for storing image datasets and trained models

Software Requirements:

- A deep learning framework such as TensorFlow, PyTorch, or Keras for implementing deep learning models
- Python programming language for scripting and running deep learning models
- A text editor or integrated development environment (IDE) for writing and debugging code
- Image processing libraries such as OpenCV for loading and preprocessing image data
- A software package for visualization of segmentation results, such as Matplotlib or Pillow

U NET

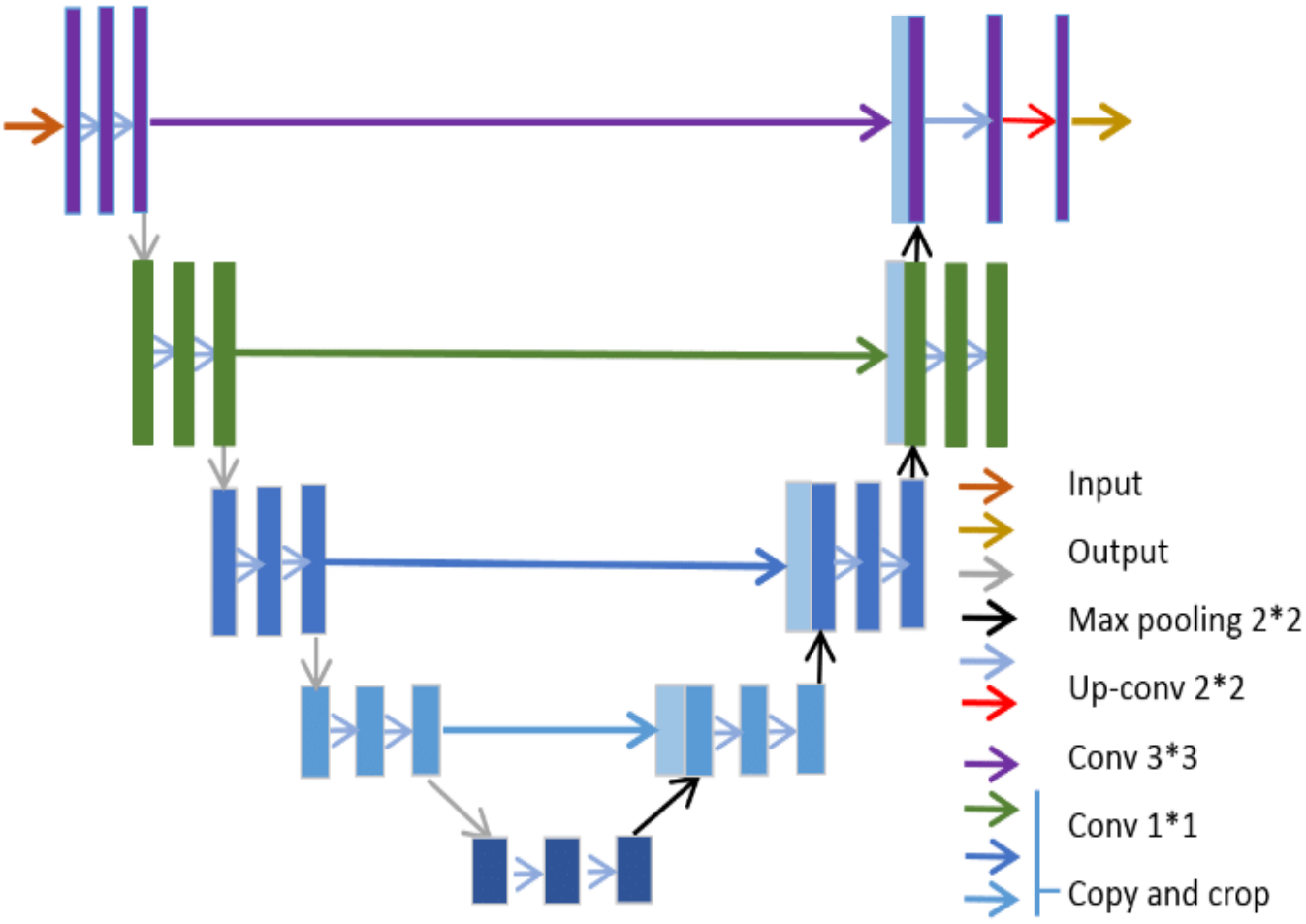
U-Net is a convolutional neural network that was developed for biomedical image segmentation. The network is based on a fully convolutional network whose architecture was modified and extended to work with fewer training images and yield more precise segmentation.

U-net architecture is symmetric and consists of two major parts: * The left part is called the contracting path, constituted by the general convolutional process.

- The right part is an expansive path, constituted by transposed 2D convolutional layers.
- The image is input at the beginning of the network (left-top). The data is then propagated through all possible paths and, in the end, the segmentation map comes out.
- The main idea is to supplement a usual contracting network by successive layers, where up sampling operators replace pooling operations. Hence these layers increase the resolution of the output. What's more, a successive convolutional layer can then learn to assemble a precise output based on this information.

Key feature of U Net

- U-Net learns segmentation in an end-to-end setting.
- U-Net is able to precisely localize and distinguish borders.
- U-Net uses very few annotated images.



U Net Architecture

IMPLEMENTATION

U Net implementation

```python
from tensorflow.keras.layers import Conv2D, BatchNormalization, Activation, MaxPool2D, Conv2DTranspose, Concatenate, Input
from tensorflow.keras.models import Model

def conv_block(inputs, num_filters):
    x = Conv2D(num_filters, 3, padding="same")(inputs)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    x = Conv2D(num_filters, 3, padding="same")(x)
    x = BatchNormalization()(x)
    x = Activation("relu")(x)

    return x

def encoder_block(inputs, num_filters):
    x = conv_block(inputs, num_filters)
    p = MaxPool2D((2, 2))(x)
    return x, p

def decoder_block(inputs, skip_features, num_filters):
    x = Conv2DTranspose(num_filters, 2, strides=2, padding="same")(inputs)
    x = Concatenate()([x, skip_features])
    x = conv_block(x, num_filters)
    return x

def build_unet(input_shape):
    inputs = Input(input_shape)
```

```python

def build_unet(input_shape):
    inputs = Input(input_shape)

    s1, p1 = encoder_block(inputs, 64)
    s2, p2 = encoder_block(p1, 128)
    s3, p3 = encoder_block(p2, 256)
    s4, p4 = encoder_block(p3, 512)

    # print(s1.shape, s2.shape, s3.shape, s4.shape)
    # print(p1.shape, p2.shape, p3.shape, p4.shape)

    b1 = conv_block(p4, 1024)

    d1 = decoder_block(b1, s4, 512)
    d2 = decoder_block(d1, s3, 256)
    d3 = decoder_block(d2, s2, 128)
    d4 = decoder_block(d3, s1, 64)

    outputs = Conv2D(1, 1, padding="same", activation="sigmoid")(d4)

    model = Model(inputs, outputs, name="UNET")
    return model

if __name__ == "__main__":
    input_shape = (256, 256, 3)
    model = build_unet(input_shape)
    model.summary()
```

```
Model: "UNET"
```

```
Model: "UNET"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 256, 256, 3)]     0

 conv2d_38 (Conv2D)          (None, 256, 256, 64)      1792

 batch_normalization_36 (BatchNormalization)  (None, 256, 256, 64)  256

 activation_36 (Activation)  (None, 256, 256, 64)      0

 conv2d_39 (Conv2D)          (None, 256, 256, 64)      36928

 batch_normalization_37 (BatchNormalization)  (None, 256, 256, 64)  256

 activation_37 (Activation)  (None, 256, 256, 64)      0

 max_pooling2d_8 (MaxPooling2D)  (None, 128, 128, 64)  0

 conv2d_40 (Conv2D)          (None, 128, 128, 128)     73856
```

Training and testing

```
[ ]
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

import numpy as np
import cv2
from glob import glob
from sklearn.utils import shuffle
import tensorflow as tf
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger, ReduceLROnPlateau, EarlyStopping, TensorBoard
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split


""" Global parameters """
H = 256
W = 256

def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)

def load_dataset(path, split=0.2):
    images = sorted(glob(os.path.join(path, "images", "*.png")))
    masks = sorted(glob(os.path.join(path, "masks", "*.png")))

    split_size = int(len(images) * split)

    train_x, valid_x = train_test_split(images, test_size=split_size, random_state=42)
    train_x, valid_x = train_test_split(images, test_size=split_size, random_state=42)
    train_y, valid_y = train_test_split(masks, test_size=split_size, random_state=42)

    train_x, test_x = train_test_split(train_x, test_size=split_size, random_state=42)
    train_y, test_y = train_test_split(train_y, test_size=split_size, random_state=42)

    return (train_x, train_y), (valid_x, valid_y), (test_x, test_y)

def read_image(path):
    path = path.decode()
    x = cv2.imread(path, cv2.IMREAD_COLOR)
    x = cv2.resize(x, (W, H))
    x = x / 255.0
    x = x.astype(np.float32)
    return x

def read_mask(path):
    path = path.decode()
    x = cv2.imread(path, cv2.IMREAD_GRAYSCALE)  ## (h, w)
    x = cv2.resize(x, (W, H))   ## (h, w)
    x = x / 255.0               ## (h, w)
    x = x.astype(np.float32)    ## (h, w)
    x = np.expand_dims(x, axis=-1)## (h, w, 1)
    return x

def tf_parse(x, y):
    def _parse(x, y):
        x = read_image(x)
        y = read_mask(y)
        return x, y
```

```python

def tf_dataset(X, Y, batch=2):
    dataset = tf.data.Dataset.from_tensor_slices((X, Y))
    dataset = dataset.map(tf_parse)
    dataset = dataset.batch(batch)
    dataset = dataset.prefetch(10)
    return dataset

if __name__ == "__main__":
    """ Seeding """
    np.random.seed(42)
    tf.random.set_seed(42)

    """ Directory for storing files """
    create_dir("files")

    """ Hyperparameters """
    batch_size = 16
    lr = 1e-4
    num_epochs = 500
    model_path = os.path.join("files", "model.h5")
    csv_path = os.path.join("files", "log.csv")

    """ Dataset """
    dataset_path = "/media/nikhil/Seagate Backup Plus Drive/ML_DATASET/brain_tumor_dataset/data"
    (train_x, train_y), (valid_x, valid_y), (test_x, test_y) = load_dataset(dataset_path)

    print(f"Train: {len(train_x)} - {len(train_y)}")
    print(f"Valid: {len(valid_x)} - {len(valid_y)}")
```

```python
import os
os.environ["TF_CPP_MIN_LOG_LEVEL"] = "2"

import numpy as np
import cv2
import pandas as pd
from glob import glob
from tqdm import tqdm
import tensorflow as tf
from tensorflow.keras.utils import CustomObjectScope
from sklearn.metrics import f1_score, jaccard_score, precision_score, recall_score
from sklearn.model_selection import train_test_split


""" Global parameters """
H = 256
W = 256

""" Creating a directory """
def create_dir(path):
    if not os.path.exists(path):
        os.makedirs(path)

def save_results(image, mask, y_pred, save_image_path):
    mask = np.expand_dims(mask, axis=-1)
    mask = np.concatenate([mask, mask, mask], axis=-1)
```

```
    y_pred = np.expand_dims(y_pred, axis=-1)
    y_pred = np.concatenate([y_pred, y_pred, y_pred], a
    y_pred = y_pred * 255

    line = np.ones((H, 10, 3)) * 255

    cat_images = np.concatenate([image, line, mask, lir
    cv2.imwrite(save_image_path, cat_images)


if __name__ == "__main__":
    """ Seeding """
    np.random.seed(42)
    tf.random.set_seed(42)

    """ Directory for storing files """
    create_dir("results")

    """ Load the model """
    with CustomObjectScope({"dice_coef": dice_coef, "di
        model = tf.keras.models.load_model(os.path.joi

    """ Dataset """
```

```
    y_pred = np.expand_dims(y_pred, axis=-1)
    y_pred = np.concatenate([y_pred, y_pred, y_pred], axis=-1)
    y_pred = y_pred * 255

    line = np.ones((H, 10, 3)) * 255

    cat_images = np.concatenate([image, line, mask, line, y_pred], axis=1)
    cv2.imwrite(save_image_path, cat_images)


if __name__ == "__main__":
    """ Seeding """
    np.random.seed(42)
    tf.random.set_seed(42)

    """ Directory for storing files """
    create_dir("results")

    """ Load the model """
    with CustomObjectScope({"dice_coef": dice_coef, "dice_loss": dice_loss}):
        model = tf.keras.models.load_model(os.path.join("files", "model.h5"))

    """ Dataset """
    dataset_path = "/media/nikhil/Seagate Backup Plus Drive/ML_DATASET/brain_tumor_dataset/data"
    (train_x, train_y), (valid_x, valid_y), (test_x, test_y) = load_dataset(dataset_path)

    """ Prediction and Evaluation """
    SCORE = []
    for x, y in tqdm(zip(test_x, test_y), total=len(test_y)):
        """ Extracting the name """
```

```
        y_pred = y_pred >= 0.5
        y_pred = y_pred.astype(np.int32)

        """ Saving the prediction """
        save_image_path = os.path.join("results", name)
        save_results(image, mask, y_pred, save_image_path)

        """ Flatten the array """
        mask = mask/255.0
        mask = (mask > 0.5).astype(np.int32).flatten()
        y_pred = y_pred.flatten()

        """ Calculating the metrics values """
        f1_value = f1_score(mask, y_pred, labels=[0, 1], average="binary")
        jac_value = jaccard_score(mask, y_pred, labels=[0, 1], average="binary")
        recall_value = recall_score(mask, y_pred, labels=[0, 1], average="binary", zero_division=0)
        precision_value = precision_score(mask, y_pred, labels=[0, 1], average="binary", zero_division=0)
        SCORE.append([name, f1_value, jac_value, recall_value, precision_value])

    """ Metrics values """
    score = [s[1:]for s in SCORE]
    score = np.mean(score, axis=0)
    print(f"F1: {score[0]:0.5f}")
    print(f"Jaccard: {score[1]:0.5f}")
    print(f"Recall: {score[2]:0.5f}")
    print(f"Precision: {score[3]:0.5f}")

    df = pd.DataFrame(SCORE, columns=["Image", "F1", "Jaccard", "Recall", "Precision"])
    df.to_csv("files/score.csv")
```

REFERENCES

https://www.kaggle.com/datasets/nikhilroxtomar/brain-tumor-segmentation?resource=download

https://www.educative.io/answers/what-is-u-net