A SEMANTIC SEGMENTATION APPROACH TO RIVER SEDIMENT IDENTIFICATION

BY

AHMAD ZAKY ISWANI

A dissertation submitted in fulfillment of the requirement for the degree of Master of Science (Mechatronics Engineering)

Kuliyyah of Engineering
International Islamic University Malaysia

AUGUST 2023

ABSTRACT

Soil erosion is an ecological hazard that, if left unchecked, poses wider threats to the environment. These threats range from inconveniences such as the ruining of landscapes and the reduction of water quality to hazards such as floods and landslides. This thereby necessitates a method to monitor soil erosion, one of which is by monitoring the formation of river sediments. Computer vision techniques have matured in recent years and have been used in many different fields of applications. One form of computer vision technique is called "semantic segmentation," which is a technique that seeks to associate every pixel of an image with its own object class. This presented an opportunity where images of river sedimentation could be analysed and identified accurately to the pixel. In examining further the use of semantic segmentation for river sedimentation purposes, this project looked at three publicly available network architectures: Unet, Linknet, and Feature Pyramid Network (FPN). All these three networks belong to a type of architecture called fully convolutional networks. Three prediction models, one from each architecture, were trained and tested against 100 images of various river sediment formations along the course of the IIUM river. The images are divided into 75 images for training and 25 images for validation. Meanwhile, the model is assessed both quantitatively by Intesection over Union (IoU), and label predictions assessed qualitatively. After training, the sediment IoU scores obtained were as follows: 0.83446103 for Unet, 0.8188789 for Linknet, and 0.20392573 for FPN. The qualitative results outputted however were mixed: the architectures are able to perform somewhat well in identifying sedimentation when the formation of those sediments is uniform, with Unet performing the best, followed by Linknet and then FPN. However, all the architectures struggled in identifying the sediment when non-uniform sedimentation formations are present. One recommendation proposed is to add object classes to reduce intraclass differences and hopefully reduce class confusion by the prediction models. Another recommendation is to develop novel architectures that are able to accommodate intraclass differences while still producing accurate sediment identification.

خلاصة البحث

يعدّ تآكل التربة خطراً بيئياً، بحيث إذا ترك دون معالجة فإنه يشكل تهديدات أوسع للبيئة. وتمتدّ هذه التهديدات من مظاهر مزعجة، مثل تدمير المناظر الطبيعية وتقليل جودة المياه، إلى مخاطر أكبر مثل الفيضانات والانهيارات الأرضية. ممّا يتطلب بالتالي استخدام طريقة لرصد تآكل التربة، ومن ذلك مراقبة تكوين الرواسب النهرية. لقد نضجت تقنيات الرؤية الحاسوبية في السنوات الأخيرة وتم استخدامها في العديد من المجالات المختلفة للتطبيقات. يُطلق على أحد أشكال تقنية الرؤية الحاسوبية اسم "التجزئة الدلالية"، وهي تقنية تسعى إلى ربط كل بكسل في الصورة بفئة الكائن الخاصة بها. وقد أتاح هذا فرصة حيث يمكن تحليل صور الرواسب النهرية وتحديدها بدقة على مستوى البكسل. وعند إجراء مزيد من الدراسة لاستخدام التجزئة الدلالية لأغراض الرواسب النهرية، نظر هذا المشروع في ثلاثة تصاميم للشبكات متاحة للجمهور: (Unet) و (Linknet) وشبكة هرم الخصائص (FPN). تنتمي جميع هذه الشبكات الثلاث إلى نوع من التصميم يسمى الشبكات كاملة الالتفاف. تم تدريب واختبار ثلاثة نماذج تنبؤ، واحد من كل تصميم، باستخدام 100 صورة للتشكيلات المختلفة للرواسب النهرية على طول مجرى نهر (IIUM). حيث تم تقسيم الصور إلى 75 صورة للتدريب و25 صورة للتحقق. وفي الوقت نفسه، تم تقييم النموذج كمياً عن طريق التداخل عبر الاتحاد (IoU)، وتقييم تنبؤات التسميات نوعياً. بعد التدريب، تم الحصول على درجات (IoU) للرواسب على النحو الآتي: 0.83446103 لـ (Unet)، و 0.8188789 لـ (Linknet)، و 0.20392573 لـ (FPN). ومع ذلك، كانت النتائج النوعية التي تم الحصول عليها مختلطة؛ فقد كانت التصاميم قادرة على تقديم أداء جيد إلى حد ما في تحديد الترسيب عندما يكون تشكيل تلك الرواسب موحداً، حيث كان أداء (Unet) هو الأفضل، يليه (Linknet) ثم (FPN). ومع ذلك، فقد واجهت جميع التصاميم صعوبة في تحديد الرواسب عند وجود تشكيلات ترسيبية غير منتظمة. إحدى التوصيات المقترحة هي إضافة فئات الكائنات لتقليل الاختلافات داخل الفئة أملاً في تقليل ارتباك نماذج التنبؤ في تحديد الفئة. وهناك توصية أخرى، تتمثل في تطوير تصاميم جديدة قادرة على استيعاب الاختلافات داخل الفئة مع القدرة على التحديد الدقيق للرواسب.

APPROVAL PAGE

I certify that I have supervised and read this study and that in my opinion, it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Master of Science (Mechatronics Engineering)

…….
Yasir bin Mohd Mustafah
Supervisor

…….
Azhar bin Mohd Ibrahim
Co-Supervisor

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a dissertation for the degree of Master of Science (Mechatronics Engineering)

…….
Dwi Pebrianti
Examiner

This dissertation was submitted to the Department of Mechatronics Engineering and is accepted as a fulfilment of the requirement for the degree of Master of Science (Mechatronics Engineering)

…….
Ali Sophian
Head, Department of Mechatronics Engineering

This dissertation was submitted to the Kulliyyah of Engineering and is accepted as a fulfillment of the requirement for the degree of Master of Science (Mechatronics Engineering)

…….
Sany Izan Ihsan
Dean, Kulliyyah of Engineering

DECLARATION

I hereby declare that this dissertation is the result of my own investigations, except where otherwise stated. I also declare that it has not been previously or concurrently submitted as a whole for any other degrees at IIUM or other institutions.

Signature……  Date…15 August 2023…

INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA

DECLARATION OF COPYRIGHT AND AFFIRMATION OF FAIR USE OF UNPUBLISHED RESEARCH

A SEMANTIC SEGMENTATION APPROACH TO RIVER SEDIMENT IDENTIFICATION

I declare that the copyright holder of this thesis/dissertation are jointly owned by the student and IIUM.

No part of this unpublished research may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without prior written permission of the copyright holder except as provided below

1. Any material contained in or derived from this unpublished research may only be used by others in their writing with due acknowledgement.

2. IIUM or its library will have the right to make and transmit copies (print or electronic) for institutional and academic purpose.

3. The IIUM library will have the right to make, store in a retrieval system and supply copies of this unpublished research if requested by other universities and research libraries.

By signing this form, I acknowledged that I have read and understand the IIUM Intellectual Property Right and Commercialization policy.

Affirmed by Ahmad Zaky Iswani

…… 15 August 2023 ……..

Signature Date

ACKNOWLEDGEMENTS

All praise is due to Allah, the All-Powerful, whose Grace and Blessings have accompanied me throughout the duration of my programme. His Mercies and Blessings on me alleviate the titanic work of completing this thesis, despite its difficulty.

A massive gratitude to my supervisor, Assoc. Prof. Yasir bin Mohd Mustafah, whose persistent disposition, friendliness, promptness, thoroughness, and friendship were crucial in the effective completion of my assignment. I appreciate his extensive remarks, valuable recommendations, and insightful questions, which significantly improved this thesis. His intelligent remarks, recommendations, and questions really aided me due to his exceptional understanding of the purpose and subject of my work. Despite his obligations, he always made time to listen and assist me when I requested it. His moral support was unquestionably a boost that aided in the development and writing of this research paper's draught. I am also thankful to my co-supervisor, Azhar bin Mohd Ibrahim, whose assistance and collaboration contributed to the success of this project.

Lastly but not least, I am tremendously grateful to both of my adored parents for their prayers, patience, and understanding during my absence from them.

Once more, we laud Allah for His inexhaustible mercy on us, one of which is allowing us to complete our dissertation with success. Alhamdulillah

TABLE OF CONTENTS

Abstract i
Abstract in Arabic ii
Approval Page iii
Declaration v
Copyright vi
Acknowledgements vii
Table of Contents viii
List of Tables x
List of Equations xi
List of Figures xii
List of Abbreviations xiv

CHAPTER ONE: INTRODUCTION 1
1.1 Background 1
1.2 Problem Statement 2
1.3 Research Objectives 3
1.4 Research Methodology 3
1.5 Research Scope 5
1.6 Outline 6

CHAPTER TWO: LITERATURE REVIEW 8
2.1 Introduction 8
2.1.1 Convolution Operation 8
2.1.2 Activation Functions 10
2.1.3 Downsampling 12
2.2 Convolutional Neural Networks 13
2.1.2 Advantages of CNNs 16
2.1.3 Drawbacks of CNNs 17
2.3 Fully Convolutional Networks 19
2.3.1 Upsampling 21
2.3.2 Skip Connections 22
2.4 Unet 24
2.5 Linknet 26
2.6 Feature Pyramid Network 28
2.7 Sediment Identification by Other Techniques 29
2.7 Recent Research Works 30
2.8 Issue of Small Dataset 34
2.9 Chapter Summary 35

CHAPTER THREE: EXPERIMENTAL SETUP 36
3.1 Introduction 36

3.2 Data Collection 38
3.3 Dataset Preparation 39
3.4 Parameter Setting 42
3.5 Model Training 45
3.5.1 Network Architecture Implementations 45
3.5.1.1 Backbone Structure 45
3.5.1.2 ResNet-50 Network 47
3.5.1.3 Implemented Architectures 49
3.5.1.3.1 Implemented Unet 49
3.5.1.3.2 Implemented Linknet 50
3.5.1.3.3 Implemented FPN 52
3.5.1.3.4 Comparison of Implemented Architectures 53
3.5.2 Evaluation Metric and Loss Function 54
3.5.3 Code Implementation 55
3.6 Chapter Summary 57

CHAPTER FOUR: RESULTS AND DISCUSSIONS 58
4.1 Introduction 58
4.2 Training and Validation Results 58
4.3 IoU Scores 61
4.4 Segmentation Prediction Comparisons 62
4.5 Discussions 67
4.6 Chapter Summary 70

CHAPTER FIVE: CONCLUSION AND RECOMMENDATIONS 71
5.1 Conclusion 71
5.2 Contributions 72
5.3 Recommendations 72

REFERENCES 73

LIST OF TABLES

Table 1	Training hyperparameters.	43
Table 2	ResNet-50 internal structure (He et al., 2015).	49
Table 3	Table of comparison of the implemented structures.	53
Table 4	Computer used for model training.	57
Table 5	Implemented architecture IoU scores.	62
Table 6	Summary table of implemented architecture performances.	70

LIST OF EQUATIONS

Equation 1	ReLU equation.	11
Equation 2	Softmax equation.	12
Equation 3	IoU equation.	54

LIST OF FIGURES

Figure 1	Research methodology flowchart.	4
Figure 2	Convolution operation flow.	9
Figure 3	Convolution operation with added zero padding.	10
Figure 4	ReLU function (Es-Sabery et al., 2021).	11
Figure 5	Softmax function (Es-Sabery et al., 2021).	12
Figure 6	Downsampling operation by maxpooling.	13
Figure 7	Basic convolutional neural network (Șerban et al., 2014)	15
Figure 8	Loss value fed back to update and improve the model.	16
Figure 9	Fully connected network (Long et al., 2014).	21
Figure 10	Nearest neighbour upsampling.	21
Figure 11	Transpose convolution.	22
Figure 12	Fully convolutional network (FCN) flow (Long et al., 2014).	23
Figure 13	Results of FCNs with varying skip connections (Long et al., 2014).	23
Figure 14	Unet architecture (Ronneberger et al., 2015).	26
Figure 15	Linknet architecture (Chaurasia & Culurciello, 2017).	27
Figure 16	FPN architecture (Kirillov et al., n.d.).	29
Figure 17	FPN skip connection flow (Kirillov et al., n.d.).	29
Figure 18	Experimental setup flowchart.	37
Figure 19	(Left) an example of a dataset image and (right) its corresponding ground truth mask. The ground truth masks are marked in grayscale values where 1 is river body, 2 is the river sediment, and 3 is the background.	39
Figure 20	ResNet residual block.	47
Figure 21	Schematic of ResNet-50.	48
Figure 22	Schematic of implemented Unet with Resnet-50 backbone.	50
Figure 23	Schematic of implemented Linknet with Resnet-50 backbone.	51

Figure 24	Schematic of implemented FPN with Resnet-50 backbone.	53
Figure 25	Unet epoch progression charts.	58
Figure 26	Linknet epoch progression charts.	59
Figure 27	FPN Epoch progression charts.	60
Figure 28	Original image, ground truth mask, and predicted masks for image #1.	63
Figure 29	Original image, ground truth mask, and predicted masks for image #2.	65
Figure 30	Original image, ground truth mask, and predicted masks for image #3.	65
Figure 31	Original image, ground truth mask, and predicted masks for image #4.	66

LIST OF ABBREVIATIONS

CNN – Convolutional neural network

CPU – Central Processing Unit

ESA – European Space Agency

FCN – Fully convolutional network

FPN – Feature Pyramid Network

GPU – Graphics Processing Unit

GPR – Gaussian Projection Regression

HOG – Histogram of Oriented Gradients

IIUM – International Islamic University Malaysia

ILSVRC – ImageNet Large Scale Visual Recognition Competition

IoU – Intersection over Union

ISBI – IEEE (Institute of Electrical and Electronics Engineers) International Symposium on Biomedical Imaging

KNN – K-Nearest Neighbour

NOAA – National Oceanic and Atmospheric Administration

PLDM – Power Line Dataset of Mountain Scene

PLDU – Power Line Dataset of Urban Scene

PNG – Portable Network Graphics

RAM – Random-Access Memory

R-CNN – Region-based Convolutional Neural Network

ReLU – Rectified Linear Unit

RF – Random Forest

RGB – Red, Green, Blue

SAR – Synthetic Aperture Radar

SIFT – Scale-Invariant Feature Transform

SVM – Support Vector Machine

UAV – Unmanned Aerial Vehicle

USGS – United States Geological Survey

VGG – Visual Geometry Group

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND

Due to the ever-increasing activities in human activity, coupled with the boom in human population especially in the second half of the twentieth century, issues regarding the health of river systems have become more and more prominent. One of these issues is soil erosion that occurs in and around the body of river flow. River erosion presents a threat since it often brings negative impacts towards the surroundings. For example, the European Union noted ecological damages caused by soil erosion as decreases in crop yields, siltation of rivers, loss of water surface quality, and muddy floods (Pineux et al., 2017). Such problems serve as pollution to rivers, thereby increasing the cost of river clean-up and also the cost of water tariffs (Elijah et al., 2018). More serious consequences of unmonitored erosion may lead to landslides that may result in human fatalities if there were any residential areas built around the river. Although these examples are mostly true for large, natural rivers, the effects of erosion in small rivers or creeks are still often undesirable. In an example close to us, the river at the IIUM Gombak Campus has been suffering from erosion, with the effects being clearly visible in the sedimentation formed on the river itself. For one, sand mounds have crept up into the middle of the river body itself, providing an eyesore for pedestrians. Moreover, it has caused non-optimal water flow, sometimes causing flooding along some sections of heavily built-up areas of the campus.

Computer vision techniques have long been applied to applications involving object identification and classification of aerial imagery. Applications of them have included the classification of aerospatial objects, automobiles, buildings, and vegetation cover (Manso-Callejo et al., 2020), thereby showing computer vision's usefulness in a wide variety of applications. In recent years, the field of semantic segmentation has been

conceived and has continued to grow. Semantic segmentation is a technique that associates every pixel of an image to a unique, predefined label or object class. It involves the training of a neural network model, and then having that model output of the same height and width dimensions of the input image that contains a dense prediction map of the different object classes of the input image (Hussein & Ali, 2022). The technique differs from non-semantic segmentation which only clusters pixels together based on the general characteristics of single objects and therefore is not well-defined as many different segmentations might be acceptable (Thoma, 2016).

1.2 PROBLEM STATEMENT

Semantic segmentation has been utilised in a variety of geographical surveillance applications, from tracking built-up areas in cities to monitoring river water levels during floods. However, from the best understanding of the literature, semantic segmentation has not been utilised explicitly to identify nor monitor river sedimentation formations. Inspired by this research gap, this project aims to study the feasibility of different architectures of semantic segmentation for river sedimentation identification.

Semantic segmentation works by predicting a class to every single pixel of an image based on user-given ground truths. This is different from an object detection task, whereby the algorithm predicts an object by assigning a bounding box to where that object is in the image. In object detection, the algorithm makes predictions based on a collection of learned features of an object, such as distinct edges or shapes the object may have. Semantically segmenting river sediments, however, is more complex because unlike the discrete objects on which object detection is usually applied on, river sediment formations are rarely fixed or shape-distinct. Hence, the semantic segmentation algorithm should be robust enough to identify the different forms the river sediments may come in. Training a robust segmentation algorithm conventionally involves a huge dataset collection, however the personal nature of this project means that collecting a vast number of images to make up the dataset, along with manually annotating them, is highly

unlikely. Instead, different network architectures will be trained, and their performances will be compared to see which outputs the best predictions of the river sediments.

1.3 RESEARCH OBJECTIVES

The objectives of this research have been set as follows:

- To train different semantic segmentation network architectures to perform IIUM river sediment identification
- To evaluate and compare the performances of the trained semantic segmentation architectures in identifying river sediment

1.4 RESEARCH METHODOLOGY

This project aims to seek the feasibility of semantic segmentation for river sediment identification by comparing the performances of three different popular segmentation network architectures, which are Unet, Linknet, and Feature Pyramid Networks (FPN). The research methodology is as described in Figure 1 below:

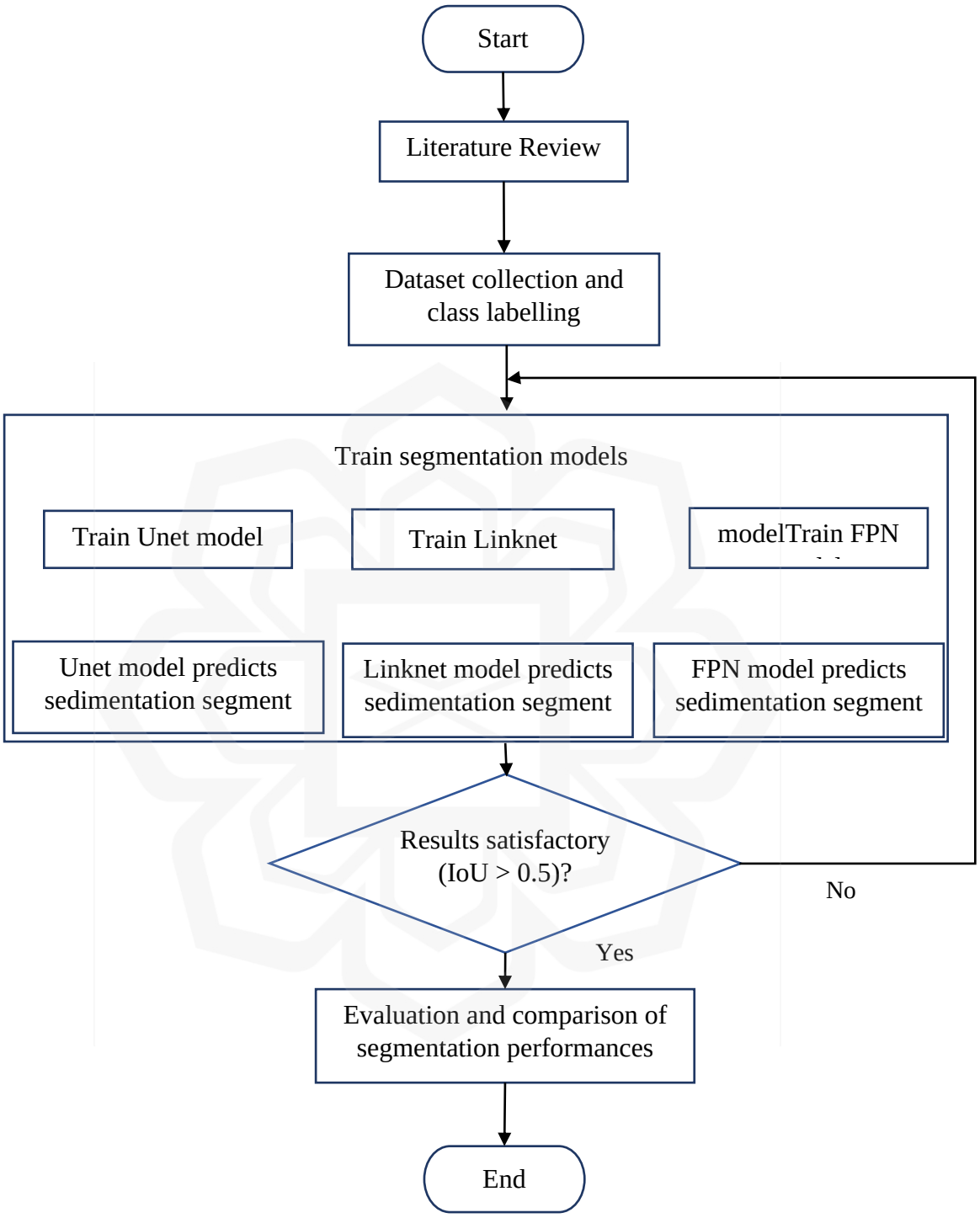


Figure 1 Research methodology flowchart.

1. Literature review
    - Literature review on existing sediment identification techniques as well as applications of semantic segmentation in aerial imagery contexts to gauge the effectiveness of the method for river sediment identification purposes.
2. Dataset collection and class labelling
    - Dataset is comprised of images of the IIUM river along different sections. Images were captured by personal phone camera. The images are then labelled/annotated into object classes of river body, river sediment, and background.
3. Train segmentation models
    - Three open-source architecture models – Unet, Linknet, and FPN – were trained by Tensorflow Keras using collected dataset.
4. Sedimentation segment prediction
    - The three architecture models perform segmentation prediction of sedimentation of the dataset river images.
5. Evaluation and comparison of segmentation performances
    - The performances of the three architecture models were evaluated and compared with each other by evaluating quantitively their epoch progression charts and Intersection over Union (IoU) scores, as well as qualitatively from the models' mask predictions on select test images. If the IoU scores achieved are below 0.5, the models are retrained.

1.5 RESEARCH SCOPE

The scope of this research is laid as follows:

- Collection and annotation of a dataset focusing on the IIUM river and is limited to only 100 images. To portray the sedimentation conditions of the river more accurately, these images were taken at various locations along the course of the

river. Moreover, they were taken at elevated positions for clarity. More specifically, the camera is to be held at an arm's length above head level and angled in such a way that all the riverbed features sought (i.e. river body, river sediment, and background) are captured. The images are to be taken using a smartphone that produces 4032x2268 dimension images in PNG format.

- The semantic segmentation algorithms that will be used are three different open-source available network architectures – Unet, Linknet, and Feature Pyramid Networks (FPN). The training of these models is to be done on a laptop with an AMD Ryzen 9 5900HX CPU and Nvidia GeForce RTX 3070 GPU.
- Evaluation and comparison will be done on the performances of the trained algorithm models. It must be noted that only the results of the prediction on the sedimentation will be evaluated, not on the river body nor on any other objects.

1.6 OUTLINE

An overview of the contents of subsequent chapters is provided below:

1. Chapter One
    - Chapter One introduces the background, problem statement, research objectives, research methodology, and research scope of this project.
2. Chapter Two
    - Chapter Two firstly introduces the underlying concepts that form the backbone of the various processes that are to be used in the project such as the convolution operation, activation functions and downsampling. Next, various neural network architectures, such as the Convolution Neural Network, Fully Convolutional Networks, Unet, Linknet, and Feature Pyramid Networks are acquainted with. The chapter then reviews literature on previous research works done on various semantic segmentation applications. The issue of small dataset was discussed at the end of the chapter.

3. Chapter Three
    - Chapter Three contains discussions on the data collection, dataset preparation, parameter setting, and dataset training of this project. Specifically in the dataset training subchapter, the implemented network architectures are described in further detail and also the evaluation metric and loss function utilised.
4. Chapter Four
    - Chapter Four elaborates the results obtained in the project. The epoch progression charts of the three architectures were analysed along with their evaluation metric scores. The segmentation predictions of the architectures were also qualitatively examined. This is followed by a discussion of the results obtained.
5. Chapter Five
    - Chapter Five serves as the conclusion of the project whereby the objectives of the project were revisited and improvements to the research conducted were made.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

Before delving deeper into the architectures for semantic segmentation, it is worthwhile to gain an understanding of the concepts that are fundamental to the operation of these networks. Such concepts that are to be understood are the convolution operation, the pooling operation, and the sampling operations. Following this, the network architectures and the subsequent literature review can be understood better.

2.1.1 Convolution Operation

The convolution operation is a defining characteristic of contemporary machine learning, and since its inception, it has gained prominence in a wide range of applications where neural networks are utilised (Yamashita et al., 2018). Convolution is a special type of linear operation performed on a tensor, i.e. an array of numbers. This tensor is applied by means of a smaller array of numbers known as a kernel. In computer vision applications, this input tensor would consist of the input image's 2D vector. In greater detail, the operation is defined as the calculation of the element-wise product of each element of the kernel and the input tensor at each position of the tensor, followed by the summation of the element-wise product. The obtained summation is then assigned as the output value at the corresponding position of a tensor output called a feature map. Figure 2 depicts an example of a convolution operation that can help clarify this process. In this instance, the input tensor is a 5x5 2D vector while the kernel is a 3x3 2D vector. The kernel begins its position at the top-left corner of the input tensor such that it covers a 3x3 "shadow" or nine elements total on the input tensor, covering the element values 1, 2, 1, 2, 0, 0, 1, 0, 2, 2. (counting the elements from left-to-right and then down). The values of each element

value on the tensor are then multiplied by the element value on the kernel that corresponds to it. After multiplying each element by itself, the nine resulting products are added together, resulting in the mathematical expression (1*1) + (2*0) + (1*1) + (2*0) + (0*1) + (0*0) + (1*1) + (0*0) + (2*1) Therefore, the obtained sum, i.e. 5, would represent the value of the first element of the feature map. To obtain the value of the next element of the feature map, the kernel is shifted to the right by one position, shadowing a new set of 3x3 elements, this time 2, 1, 0, 0, 0, 1, 0, 2, 1. A stride is the movement of the kernel from one position to another. The same operation of element-wise product and summation with the kernel is performed once more, with the mathematical operation this time being (2*1) + (1*0) + (0*1) + (0*0) + (0*1) + (1*0) + (0*1) + (2*0) + (1*1). Therefore, the resultant sum of 3 represents the value of the second element of the feature map. This procedure is then repeated until all of the feature map's elements are filled. Typically, the kernel is shifted one position, adjacent to the next element column of the input tensor; this represents a stride of 1. A stride greater than 1 - i.e., when the kernel is shifted more than one element column away - can however sometimes be used, such as when downsampling is desired.

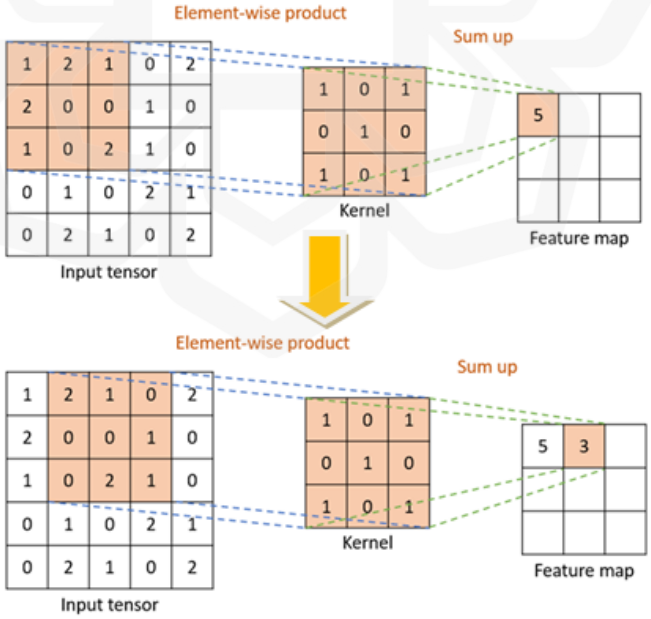


Figure 2 Convolution operation flow.

The convolutional process in Figure 2, however, produces a feature map that has dimensions that are smaller than that of the input tensor. This is due to the fact that the kernel cannot be positioned so that its centre element corresponds to any of the elements on the edges of the input tensor. Typically, a technique known as zero padding is used to address this change in dimension issue. As depicted in Figure 3, this method involves the addition of zeros that surround the edges of the input tensor. With this technique, the kernel's centre can shadow the input tensor's edge elements, while the kernel's edge elements now shadow the added zero "pads," allowing the feature map to maintain the input tensor's dimensions. Without this method, as more and more convolution operations are performed, the resulting feature maps will become smaller and smaller, which reduces the information of the feature maps.

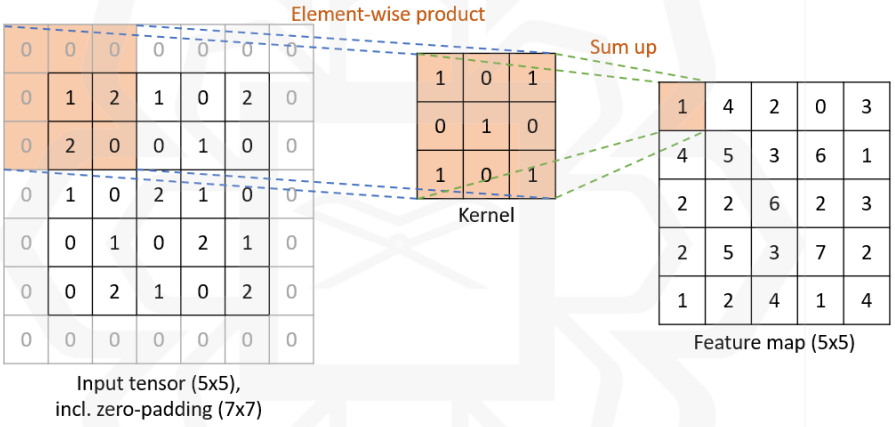


Figure 3 Convolution operation with added zero padding.

2.1.2 Activation Functions

Activation functions are in essence functions that transform an input signal into a different form of output signal. In the context of neural networks, activation functions in neural networks are frequently of the nonlinear variety. These functions, as their name suggests, are used to introduce nonlinearity into the network. A neural network lacking nonlinearity will only generate output signal functions with a polynomial degree of one, i.e. linear functions. Although linear functions are simple and easy to solve, they lack

complexity and are therefore not well suited to learn and recognise the complexities in given data, which is a necessary ability for object detection because different objects within an input image must be identified and classified. Therefore, nonlinear activation functions are utilised so that networks can comprehend and make sense of complex, high-dimensional datasets and architectures.

One very popular form of a non-linear activation function is the Rectified Linear Unit, or ReLU, function. The function operates by deactivating outputs with values less than zero and activating only outputs with positive values (Sharma et al., 2020). Figure 4 depicts a diagrammatic illustration of ReLU. Mathematically, ReLU is defined as follows:

$$f(x) = \max(0, x) \tag{1}$$

ReLU is an efficient activation function because only a certain subset of neurons are activated at any given time, as opposed to all of them.

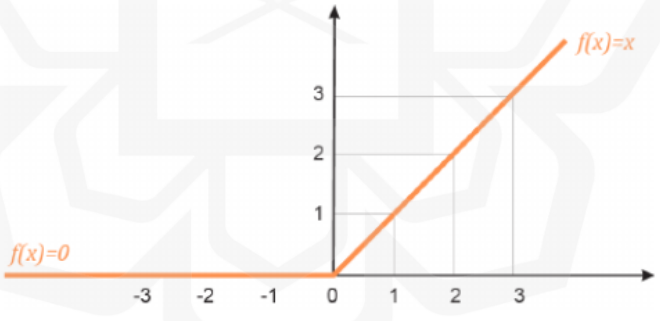


Figure 4 ReLU function (Es-Sabery et al., 2021).

Another popular form of a non-linear activation function is the softmax. The softmax function returns values between 0 and 1, so its outputs can be interpreted as the probabilities of an object class's data points. An advantage of softmax is that it is able to be utilised for multiclass classification tasks (Sharma et al., 2020). Figure 5 shows the graphical function shape of softmax, while mathematically it is defined as follows:

$$\sigma(\boldsymbol{z}_j) = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \text{for } j = 1, \ldots, K. \quad (2)$$

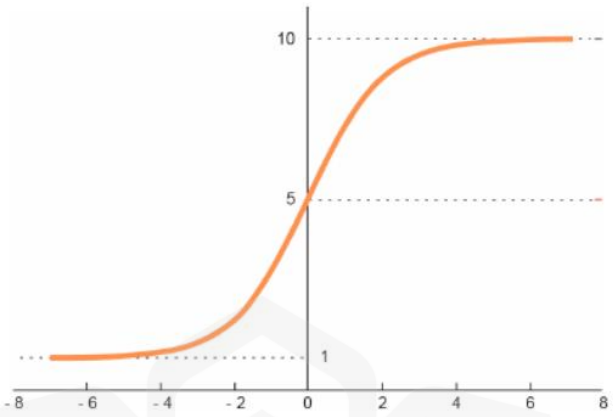


Figure 5 Softmax function (Es-Sabery et al., 2021).

2.1.3 Downsampling

Downsampling is the process of reducing the dimensions of feature maps. As previously mentioned, inputs may be downsampled by increasing the stride during convolution operations to a value greater than one. Another common technique for downsampling is by using pooling operations. The most prevalent form of this technique is the max pooling technique. In contrast to convolution, where various mathematical operations are required to obtain the output values, max pooling simply takes the maximum value of an input sensor's patch. Figure 6 illustrates an example of a max pooling operation. In the example, the dimensions of the input tensor are 4x4 while those of the filter are 2x2. The filter thereby does a total of four patches on the input tensor. The maximum value of the elements in the first patch is 8, in the second patch it is 9, in the third patch it is 7, and in the fourth patch it is 6. Therefore, the output of the feature map following max pooling is 8, 9, 7, 6.

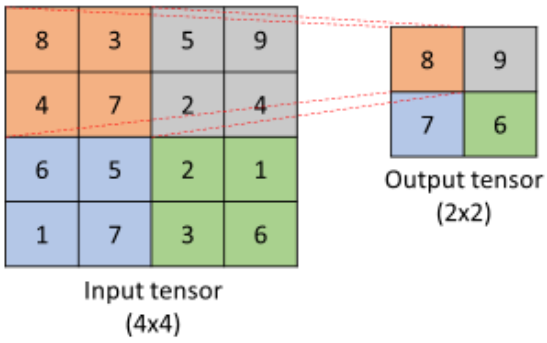


Figure 6 Downsampling operation by maxpooling.

2.2 CONVOLUTIONAL NEURAL NETWORKS

Convolutional neural networks, or CNNs, have become one of the most prevalent types of artificial neural networks over the past decade. Its popularity skyrocketed after its remarkable performance in the 2012 ImageNet Large Scale Visual Recognition Competition, or ILSVRC, which is a significant, annual, large-scale image object recognition competition due to the vast volume of data involved. The ILSVRC contained 1431167 annotated images with 1000 distinct object classes in 2012 (Russakovsky et al., 2015). The name for the particular CNN architecture is AlexNet and was concepted by Khrizhevsky et al. Since then, numerous researchers have endeavoured to develop novel architectures to extract even greater performance from the CNN concept. The VGG16 model is one of the most established and extensively used of these architectures. The model first ascended to prominence after achieving 92.7% top-five test accuracy during the 2014 ILSVRC. In order to better comprehend the functioning of CNNs in general, the VGG16 architecture is discussed in greater detail below.

A CNN architecture can be divided into two main blocks, the first block consisting of the convolutional and pooling layers, and the second block comprising of the fully-connected layers. When an input data or tensor, – in computer vision applications this would be an image of specified dimensions – is transmitted through a CNN, it would first undergo a series of convolution and pooling operations before being

fed into the fully-connected neural network. In the VGG16 example in Figure 7, an image first travels through two convolution layers. In these two layers, 64 distinct kernels of dimensions 3x3 operate on an input tensor of dimensions 224x224x3, the dimension of 3 denoting each of the three RGB channels. In addition, a stride of 1 was set for the kernels, padding was applied to preserve the dimensions from the input tensor, and the ReLU activation function was applied. This yields a first convolutional layer with dimensions of 224x224x64. The resulting feature maps are then passed through a second set of 64 kernels and ReLU operation to generate a second set of 224x224x64 feature maps, thereby forming the second convolutional layer. Of particular note, the number of kernels corresponds to and determines the depth of the resulting feature maps; in this case, 64 kernels generate a depth of 64. On the second convolutional layer, max pooling is then performed. A 2x2 pixel window with a stride of 2 halves the height and width to 112x112. The tensor then traverses an additional pair of convolution layers. This process is analogous to the first two layers — convolution followed by ReLU — but here there are 128 kernels in this layer instead. After the fourth convolutional layer, max pooling is implemented to reduce the dimensions to 56x56. This procedure is repeated three times with three sets of convolutional layers followed by maximum pooling until the tensor's dimensions are reduced to 7x7x512.

After passing through the convolutional and pooling layers block, the image tensor is first flattened and one-dimensionalized. This process of flattening will prepare the tensor for input into the neurons of the fully-connected layer block. There are three fully-connected layers in VGG16, with the first two containing 4096 channels and the third containing 1000 channels, which corresponds to the number of classes from ILSVRC. The final layer is a softmax activation layer. As previously discussed, softmax permits the generation of probabilities for multiclass problems; therefore, it is employed here.

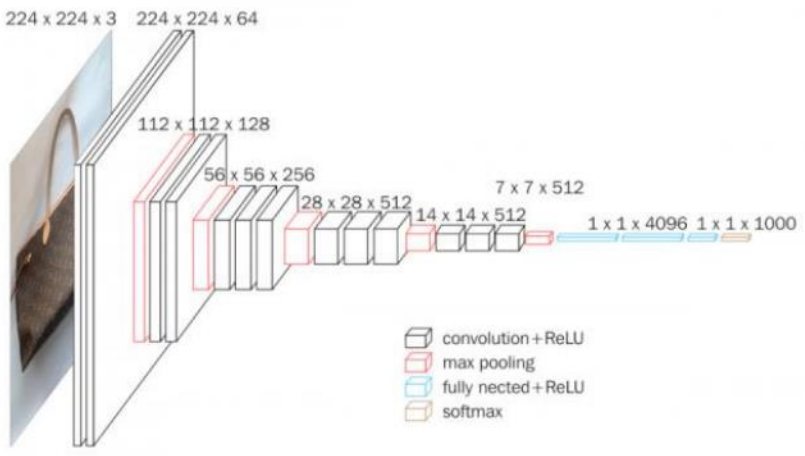


Figure 7 Basic convolutional neural network (Șerban et al., 2014).

The forward transmission of an input tensor through the convolution and pooling layers block and the fully-connected block is referred to as forward propagation. In order to provide the most accurate predictions, a CNN would need to self-train and adjust its parameters appropriately. Therefore, a loss function is used to measure the compatibility between the forward propagation output predictions and the given ground truth labels, which represent the correct and desired predictions. Obtaining a loss value or a measure of the "wrongness" of the CNN model, this value is then fed backwards – in the direction from the fully-connected block to the convolution and pooling block – and is referred to as back propagation; the flowchart for this process is depicted in Figure 8. By back propagating the loss value, it functions as a benchmark for the CNN model to determine the extent to which its parameters must be modified. Modifiable parameters include the weights of the fully-connected (FC) layers and the values of the convolution layers' kernels. Note that the dimensions of the kernels, as well as the dimensions and values of the pooling filters, remain unaltered following back propagation and must be predetermined by the user.

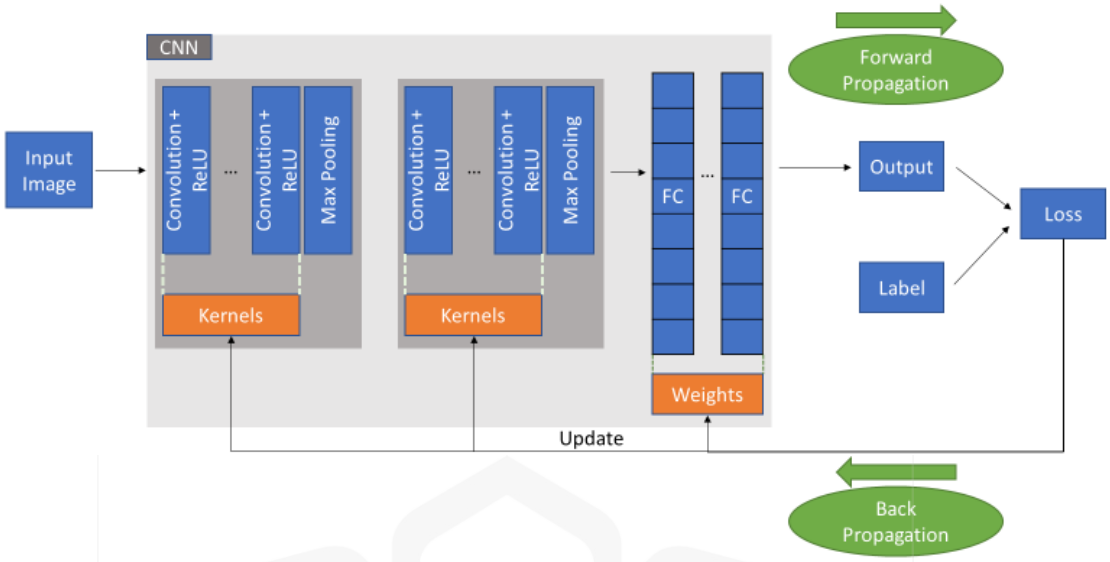


Figure 8 Loss value fed back to update and improve the model.

2.2.1 Advantages of CNNs

CNN has become a powerful tool for object detection applications due to its simplicity in comparison to earlier machine learning techniques. Previously, features of the object had to be manually crafted, typically by detecting the features using algorithms such as SIFT and Hough transforms, then extracting the features using extractors such as HOG or LBP, and finally running them through shallow machine learning classifiers such as KNN or SVM (Wahid et al., 2021). Using CNN does not necessitate the handcrafting of features. In CNN, feature extraction is performed by the convolution and pooling layers, with shallower layers extracting low-level features and deeper layers extracting high-level features. In particular, the convolutions layers detect conjunctions of features from the previous layer, whereas the pooling layers combine semantically similar features into a single feature (Lecun et al., 2015). In addition, continuous learning via back propagation enables these layers to automatically and adaptively learn the spatial hierarchies of features and optimise accordingly.

A further advantage of CNNs is their robustness to variances as a result of translations. This is a result of weight sharing from the convolution (Albawi et al., 2018). During convolution, the kernel is shared across all of the image tensor's positions, enabling the learning of local patterns regardless of their spatial characteristics. The fully connected layers contribute to the robustness against translation variances. They do it by their implementation of extremely intricate functions of their inputs which makes a CNN very sensitive to minute details, such as distinguishing between different breeds of the same animal species, while remaining insensitive to large, irrelevant variations such as background, pose, lighting, and surrounding objects. In shallow machine learning methods meanwhile, it is more challenging to accomplish variance-resilience without sacrificing selectivity. (i.e. the ability to select important features for recognition). Generic kernels such as Gaussian kernels typically do not permit generalisation for examples far beyond the training data, even though they are non-linear in nature. Feature extractors previously therefore needed to be hand-designed, which requires a considerable amount of application domain expertise and skill (Lecun et al., 2015).

2.2.2 Drawbacks of CNNs

CNNs do have drawbacks, however. Firstly, although CNNs are robust to variances from translations, they appear to be less so against variances from transformations. Hosseini et al. (2017) noted that CNNs have sufficient memory to be able to memorise the entire training dataset and, as a result, can generally correctly classify training samples with high confidence. Even if the input sample is randomly perturbed, their ability to memorise even large datasets allow them to typically associate a test sample with one or more training samples of the same class with high accuracy. However, CNNs may not be able to correctly classify a sample if it has undergone a significant pixel-wise change relative to the original sample. The authors observe that CNN test samples are typically collected from the same distribution as training samples, resulting in the majority of test data points occurring close to training data points. This suggests that CNNs have a

difficult time learning samples with large transformation variances, as the test data points are now significantly more dispersed than the training data points.

Secondly, because traditional CNNs contain fully connected layers, tensors that are to travel through them must first be flattened into a one-dimensional array so that their values can be input into the neuron nodes. This results in the loss of spatial information from the original image input. In fact, the vast majority of applications of traditional CNNs involve classification problems, i.e., given an image sample, the network must predict the class of object in the image. Nevertheless, this classification applies to the entire image. For instance, a CNN model may be fed an image of a dog and then predict that the image is indeed a dog. It does not, however, predict the other events occurring in this dog image. For instance, the model will not specify the position and/or size of background objects in the dog image. In other words, while CNNs can reliably perform image classification, they cannot detect objects. Most notably, Girshick et al. (2013) proposed the region-based convolutional neural network, or R-CNN, which is a modification to the CNN architecture that enables it to perform elementary object detection. The model works much like a traditional CNN, however before the convolution layers are created, the input image is divided and grouped hierarchically into several regions based on pixel intensities, and then "suggestions" of specific objects are made based on these regions. R-CNN is able to generate boundary rectangles indicating the location of specified objects within an image. It is however not comprehensive enough to generate precise boundaries between objects as the model does not have enough spatial information. In addition, the fact that each region proposal passes through the network increases computational effort and duration.

Thirdly, conventional CNNs lack input size flexibility. Consider again the VGG16 illustration in Figure 7. If the model was trained with an input image of 224x224, the fully-connected layers will receive an input of 25088 from the final max pooling layer (7x7x512=25088). If, for instance, a 448x448 sample image is transmitted into the network, the fully-connected layers will be unable to receive the final max pooling layer because its size has increased by two orders of magnitude (14x14x512).

2.3 FULLY CONVOLUTIONAL NETWORKS

Due to the shortcomings of traditional CNNs, researchers have developed novel neural network designs that are capable of semantic segmentation. Nonetheless, CNN's dependable performance in object classification applications, researchers continue to accept the usefulness of various CNN components. Therefore, efforts were made to embrace the positive parts of CNN while rejecting its negative aspects. One such effort came from the work of Long et al. (2014) came up with the concept of fully convolutional networks, or FCNs.

FCNs are configured in two main phases, the encoder phase and the decoder phase. The encoder phase consists of layers of convolution and pooling. This phase is almost conceptually equivalent and can be nearly identical to the convolution block seen in traditional CNNs. The phase is referred known as "encoder" since its job is to encode or convert the input data into another format, which is also the role of the convolution block in conventional CNNs. In both architectures, the input image tensor is subjected to convolution processes in order to extract features of the input image.

The major distinction between an FCN and a CNN is the decoder phase of the FCN. In traditional CNN models, the convolution block is followed by fully-connected layers, whose job is to translate the extracted features from the convolutional layers to the network's final output, which in classification tasks are the probabilities for each class. Meanwhile, since FCNs are increasingly employed for segmentation purposes, spatial information and context are essential and the dimensions of the initial input must be maintained at the output. Therefore, in FCNs, following convolutions the tensor is consequently incrementally upsampled, therefore gradually regaining its original dimensions, as opposed to being gradually flattened as in a CNN.

To understand FCNs better, consider Figure 9, which depicts two simplified schematic diagrams of a typical CNN architecture and the FCN architecture developed by Long et al. The schematic on top is of the CNN architecture made up of its expected components and workflow, whereby an input image is converted into an input tensor,

which is then fed through consecutive convolution and pooling operations and gradually downsized before being flattened and fed into the network's fully-connected layers, of which they then output a class prediction of the image, which in this case is a tabby cat. (The three horizontal lines that are respectively underlined with the numbers 4096, 4096 and 1000, represent three fully connected layers with 4096, 4096 and 1000 channels respectively. Meanwhile, the last layer is the prediction output layer; this information was not explicitly described by the authors.)

Looking at the FCN architecture schematic in Figure 9, the input picture undergoes the same convolution and pooling stages as in the CNN architecture, as can be seen by examining the initial few steps of this algorithm. This is defined as the encoder phase of the FCN. The subsequent steps illustrate the difference in paradigm between FCNs and CNNs. Whereas at the next stage in the CNN the tensor would have been flattened thereby losing its spatial information, in FCN the tensors instead undergo a 1x1 convolution, i.e., convolving the tensor with a 1x1 dimension kernel. Therefore, in contrast to CNN where flattening converts the tensor to a 1D array, the tensor in FCN retains its three-dimensionality, although with 1x1xd dimensions (where "d" denotes the depth or number of feature mappings of the tensor).

In the example, the 1x1 convolution transforms the dimensions of the tensor into 1x1x4096. According to Long et al., fully-connected CNN layers can be viewed as experiencing convolutions with kernels that cover their entire input regions. This paradigm is known as convolutionalisation. Utilising the new paradigm thus transforms these layers into fully convolutional networks that can now accept inputs of any size and output classification maps. Of interesting note is the findings by the authors that computation of an FCN is substantially faster - more than five times faster - than the equivalent CNN counterpart due to high ammortisation.

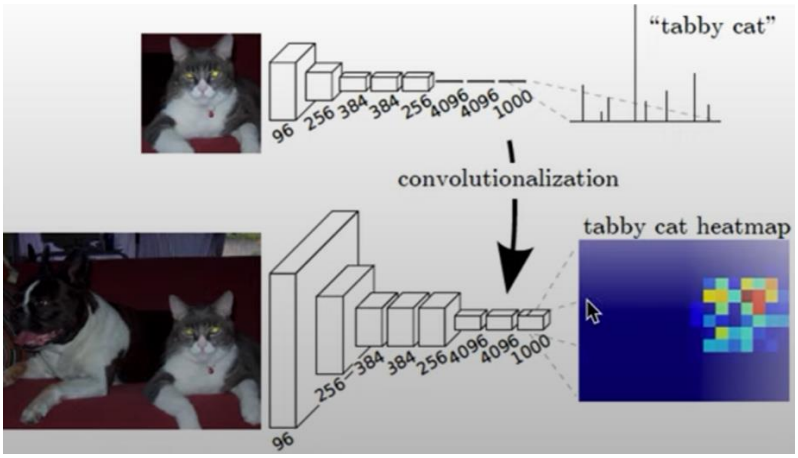


Figure 9 Fully convolutional network (Long et al., 2014).

2.3.1 Upsampling

Since the tensor should eventually return to its initial dimensions, the 1x1xd feature maps generated by a 1x1 convolution operation are insufficient. Upsampling is a technique used to increase the dimensions of a dataset. The nearest neighbour operation is a straightforward upsampling technique (Rukundo & Cao, 2012). In this method, an input tensor element is replicated in adjacent element positions of the output tensor, as depicted in Figure 10.

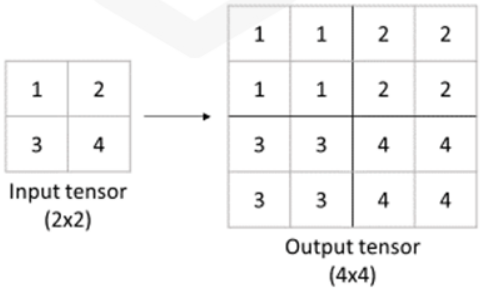


Figure 10 Nearest neighbour upsampling.

A slightly more complex technique for upsampling is known as transpose convolution (Jin et al., 2020). To comprehend the operation better, consider a 2x2 input and a 2x2 kernel, as depicted in Figure 11 below. With those input and kernel dimensions and a stride of 1, the output will have dimensions of 3x3. The sequence of the operation is such that each element of the input is multiplied by the kernel, and then the outputs from each input element's product is added together to generate the output.

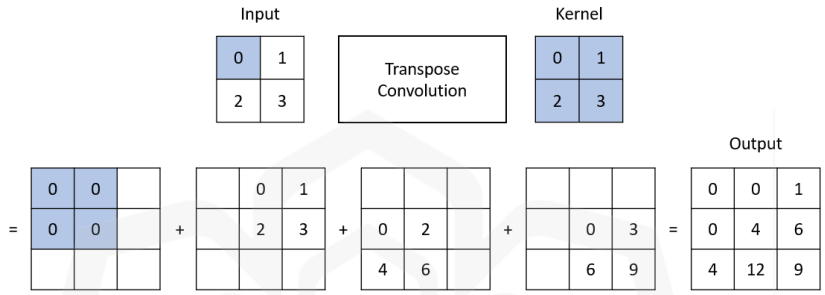


Figure 11 Transpose convolution.

2.3.2 Skip Connections

Upsampling restores the input tensor's lost dimensions during convolution operations. However, upsampling is unable to recover lost information that was contained before convolution, namely the coarse and local information from earlier layers. To recoup these information, what Long et al. termed skip connections were thereby conceived. Skip connections create links between the convoluted layers (i.e., the encoder phase) and upsampled layers of the FCN. Figure 12 below provides a depiction of these skip connections. In the figure, an input image is passed through a series of five convolutional and pooling layers. Each layer divides the input image tensor's dimensions twice, thus by the fifth layer, the tensor has been divided 32 times. Theoretically, the tensor can be directly upsampled 32 times to return it to its original dimensions and then the prediction can be made, resulting in a 32-times upsampled prediction (FCN-32). However, the only readily available information at this point is from the extracted features, not the spatial positions of those features, which may result in an erroneous prediction. As a result, as

the image is gradually upsampled, the tensor is linked back with the earlier pooled layers by way of skip connections. For instance, upsampling the pool5 layer twice doubles its dimensions, making it have the same number of dimensions as pool4. However, pool4 contains more spatial information than the 2x upsampled pool5, so the two are linked through a skip connection and then summed together. If this is followed by an immediate 16x upsampling to the original dimensions (FCN-16), the prediction results will be finer and more precise than those of the FCN-32. Instead of being upsampled 16 times, the 2x upsampled pool5 can receive even greater spatial information by upsampling itself twice more, then linking and summing with pool3, and finally upsampling itself 8 times (FCN-8) to return to its original dimensions. Hence, FCN-8's predictions will be even more precise than those of FCN-16 and FCN-32.

To summarise the previous paragraph, the more intermediate upsampling operations and skip connections there are, the clearer the output will become. The bicycle rider example in Figure 13 demonstrates this; after FCN-32, the predicted masks consist of indistinguishable blobs, but after FCN-16, the blobs become more distinguishable shapes, and after FCN-8, the mask shapes more closely resemble the original rider and bicycle masks of the ground truth. Hence, the skip connection-linked upsampled layers are collectively referred to as the decoder phase, as they are able to decode the various object segments of the image.

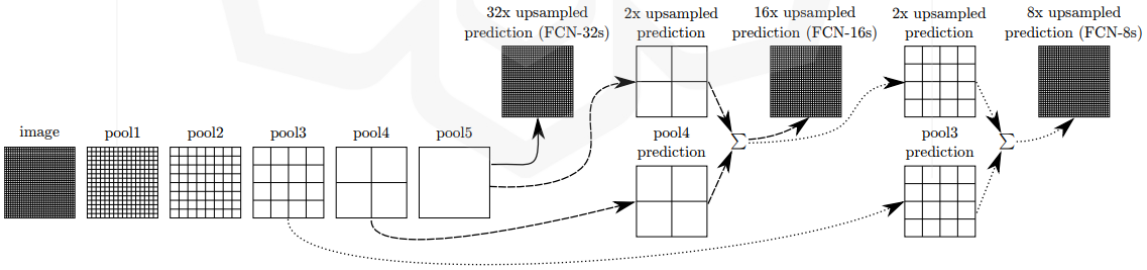


Figure 12 Fully convolutional network (FCN) flow (Long et al., 2014).

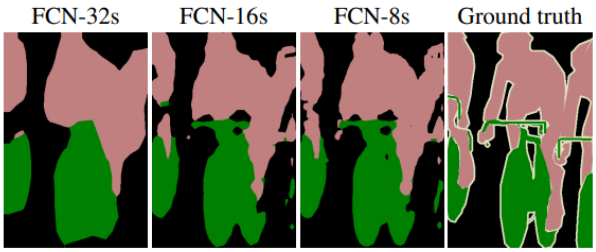


Figure 13 Results of FCNs with varying skip connections (Long et al., 2014).

2.4 UNET

Unet was proposed by Ronneberger et al. (2015) initially for the context of biomedical image segmentation, specifically to segment and classify biological cells in microscopy images. The authors noted a preexisting approach which is a sliding-window network setup that involves "sliding" and then classifying each pixel of an image by using local regions, or patches, around the pixel being analysed as inputs. Although this approach provided good localisation, the entire network needed to be run every time for each patch thus making the approach very computationally extensive and slow. Another drawback to the approach is that it trades off localisation and overall context of the image being analysed. To mitigate these drawbacks, the authors developed the Unet which is a variant of the FCN. The network architecture follows a unique U-shape structure, hence its moniker. Moreover, there are five "levels" of both encoder and decoder layers, with the encoder layers "contracting" downwards and the decoder layers "expanding" upwards.

In the following description of Unet, the levels of the architecture shall be numbered according to the forward flow of the network. Encoder levels will be numbered as it goes down, while decoder levels will be numbered as it goes up. Figure 14 below shows the schematic flow of Unet for further clarity. An input image is first fed into the first encoder level of the network, where it undergoes two 3x3 convolution and ReLU operations to generate 64 feature maps. The image tensor is then moved to the second level by a 2x2 max pooling operation, and it is subjected to a second series of two 3x3 convolution and ReLU processes, resulting in 128 feature maps. Similar procedures are

performed on the tensor at the third, fourth, and fifth levels, creating 256, 512, and 1024 feature maps, respectively. The first decoder level shares the same "height" as the fourth encoder level, as well as the second decoder level with the third encoder level, the fourth decoder level with the first encoder level, and so on. Skip connections are then employed to link each of the encoder-decoder levels of the same height. As a result, when the tensor enters the first decoder layer, it experiences a 2x2 transpose convolution upsampling operation as well as concatenation from the fourth encoder level. Hence, half of the feature maps result from upsampling, while the other half result from concatenation. The tensor would then be subjected to a sequence of two 3x3 convolutions and ReLU operations before ascending to the next decoder level, where it would be subjected to another round of upsampling and concatenation. This process would continue till the final decoder level. The very last step of the final decoder level is a 1x1 convolution operation that reduces the channel to two - the number of segment classes as utilised by the authors - and thereby outputs the predicted segmentation masks.

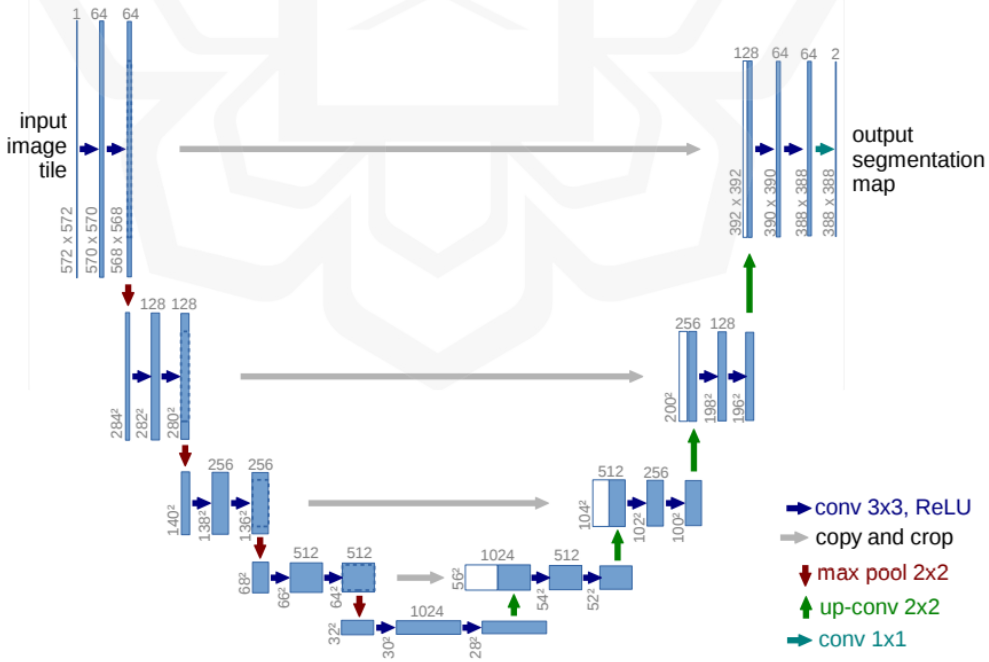


Figure 14 Unet architecture (Ronneberger et al., 2015).

One particular advantage that the authors note of their architecture is its ability to seamlessly segment arbitrarily large images. They attribute this to the large number of feature channels (i.e. feature maps) in the decoder phase, the use of only valid convolutions (i.e. convolutions without prior padding), and the absence of fully-connected layers. Indeed, the authors dataset consisted of 512x512 pixel images of biomedical cells; traditional CNNs input dimensions are usually only around 100 pixels height and width. The only padding present is at the input image itself, where the edges are mirrorly padded to preserve context of the border regions of the image. This is why the output segmentation mask is smaller than the input image.

The fact that the input and output images had slightly different dimensions did not matter to the authors, however. In the 2015 ISBI cell-tracking challenge, Unet performed remarkably. It achieved first place as it was able to segment PhC-U373 and DIC-HeLa cells with IoU scores of 0.9203 and 0.7756 respectively (Ronneberger et al., 2015), suggesting that the architecture has enough spatial information at hands for the tasks given.

2.5 LINKNET

Another FCN-based design is Linknet (Chaurasia & Culurciello, 2017). While in Unet convolutional and pooling layers (in the encoder phase) and upsampling and convolutional layers (in the decoder phase) were divided into different levels, the authors of Linknet use the term "blocks" instead. In addition, the architectural structure of Linknet resembles an inverted U, as depicted in Figure 15. The specifics of the operations within a Linknet block differ slightly from those of a Unet level. For example, in a Unet decoder level, the operational sequence starts off with an upsampling followed by two rounds of convolution, whereas in a Linknet decoder block, the sequence starts off with a 1x1 convolutional and ReLU operation followed by a 3x3 transpose convolution upsampling (Chaurasia & Culurciello denotes this operation as a "full-convolution") and finally by another 1x1 convolutional and ReLU operation. Yet, the general functioning ideas of

Unet and Linknet are very similar. Similar to how encoder-decoder levels of the same height are linked via skip connections in Unet, encoder-decoder blocks of the same height are likewise linked via skip connections in Linknet. Due to this similarity, Linknet can be thought of as an inverted version of Unet. The primary distinction between the two architectures lie in the skip connection operation. At the Unet skip connection, a decoder level would receive concatenation from its corresponding encoder level. In Linknet, a summing operation is conducted between a decoder block and its matching encoder block.

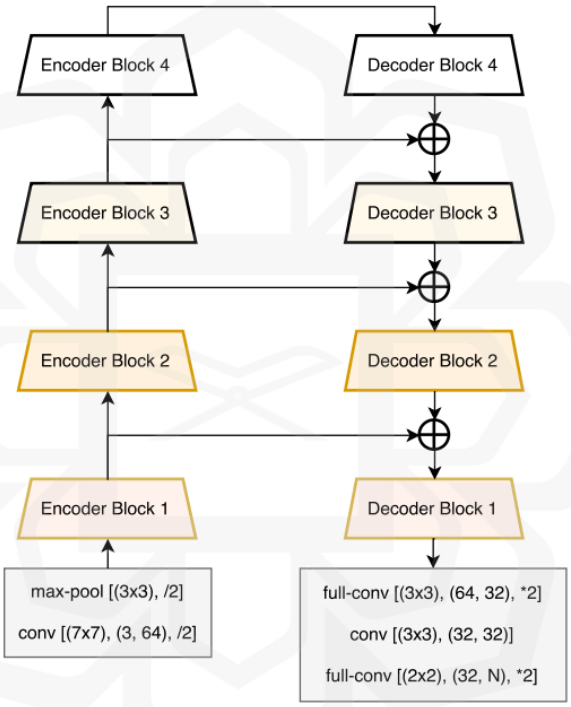


Figure 15 Linknet architecture (Chaurasia & Culurciello, 2017).

The authors tested the architecture on the Cityscapes dataset which consists of 5000 fine-annotated images of various views of a city, and 19 classes. Linknet performed amicably, scoring an IoU of 76.4. Moreover, the authors also noted of Linknet's efficiency in that it is able to process a very high-resolution single input frame of 1920x1080 pixels at 8.5 frames per second when using the Nvidia Titan X GPU card.

2.6 FEATURE PYRAMID NETWORK

Feature pyramid networks, or FPNs, are described by their authors, Lin et al. (2016), as a top-down lateral architecture with lateral connections developed for building high-level semantic feature maps at all scales. The additional column of feature map levels to the right of the decoder phase distinguishes the FPN's architectural structure from that of an Unet or Linknet, as depicted in the FPN diagram in Figure 16. Encoder and decoder phases function identically to any FCN variation, i.e., the encoder phase contains the convolution and pooling layers, whilst the decoder phase contains the upsampling layers. The skip connections between the encoder and decoder phases are similarly comparable, though FPNs use a summation skip connection similar to Linknet, as depicted in Figure 17. Thus, the most distinguishing characteristic is the aforementioned extra feature maps column. The operations from the decoder phase to this new third column are not skip connections, despite being depicted laterally. Rather, each decoder level is subjected to two consecutive 3x3 convolutional operations. These operations are done to reduce aliasing, which is a side-effect of sampling procedures that causes high-frequency components of the original signal to be indistinguishable from its low-frequency components (Ribeiro & Schön, 2021). Aliasing causes the folding of higher frequency signals onto the low-frequency component of the original signal, resulting in distortions and artefacts (Uzun & Temizel, 2022); FPN's anti-aliasing block aims to mitigate these effects. Herein lies a further distinction between FPN and FCN variations; whilst the final output of the latter relies on the feature maps of the final decoder level, the final output of the former is based on the concatenation of all the feature maps of the third column. Concatenation demands that the feature maps have the same dimensions, therefore the top level of the new column is upsampled eight times, the next level is upsampled four times, and the level below that is upsampled twice. After concatenated feature maps undergo a round of 3x3 convolution, they are then utilised to generate the output segmentation mask.

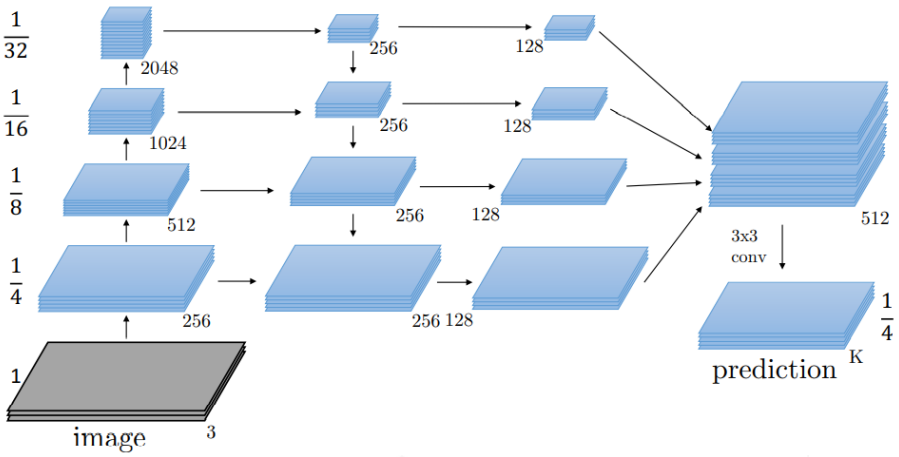


Figure 16 FPN architecture (Kirillov et al., n.d.).

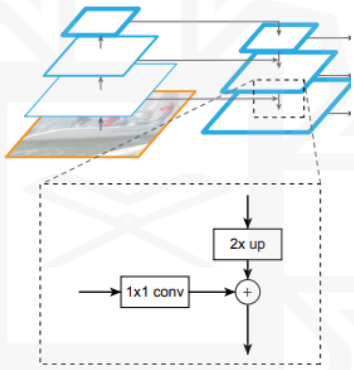


Figure 17 FPN skip connection flow (Kirillov et al., n.d.).

2.7 SEDIMENT IDENTIFICATION BY OTHER TECHNIQUES

Much of the existing research endeavours on monitoring and identifying river identification that employ machine learning have been conducted on other methods. This section will explore some of them below.

One example of such research was done by Liu et al (2019), which employed machine learning classifiers to identify sediment in a mixed-texture dredge pit on the Louisiana shelf. High-resolution bathymetry and side-sonar backscatter data of a seabed

area of 6.3 square kilometers were taken as the primary data, of which then secondary data which include the bathymetric position index, seabed curvature, and terrain variability were derived from by use of a mapping software. These data were then run through three supervised classification methods, which were Decision Trees, Random Forest, and Regularised Logistic Regression. It was found by the researchers that the Random Forest model provided the best average accuracy figures, although they also noted that the model performed poorly in specific conditions where the surficial sediment samples are mixed (such as mud and sand).

Another research conducted by Roushangar and Shahnazi (2020) predicted sediment transport rates in gravel-bed rivers. In the study, 19 various gravel-bed rivers in the United States which was collected by the US Forest Service. An array of river characteristics was available in the dataset, which included the drainage area, river slope, and the range of discharge. The pair of researchers employed two techniques in their study which were Gaussian Process Regression (GPR) and Support Vector Machine (SVM). These techniques were investigated in two scenarios, the first being on hydraulic properties only while the second being on both hydraulic and sediment properties. The study concluded that the GPR method performed better than the SVM, and that scenario with both hydraulic and sediment properties investigated generated more accurate forecast rates.

2.8 RECENT RESEARCH WORKS

Recalling Chapter 1.5, the project's dataset is to be collected from elevated positions overlooking the IIUM river. However, much of the literature concerning semantic segmentation applications for geographic surveillance – not just on rivers – have been conducted by way of aerial image datasets. Some of the literature will hence be explored below.

The bulk of the published applications of semantic segmentation of aerial images have been for satellite photos. For instance, Zhu et al. (2019) examined multi-class semantic segmentation of high-resolution satellite pictures. The authors utilised the Potsdam dataset published by the International Society of Photogrammetry and Remote Sensing. In the dataset there are 38 urban aerial photos with a resolution of 6000x6000 and six ground truth classifications, including impervious surfaces, building, low vegetation, tree, automobile, and clutter/background. 24 images were designated for training, while the remaining images were designated for testing. FCN with a VGG-19 backbone, Linknet with a ResNet-50 backbone, EncNet with a ResNet-101 backbone, and D-Linknet with a ResNet-50 backbone were the implemented architectures. The order of the architectures from best to worst were found to be as follows: EncNet, D-Linknet, FCN, Linknet. This is based on their respective overall accuracy rates of 86.4%, 86.1%, 85.0%, and 84.1%. The authors however noticed that EncNet has significantly more parameters than D-Linknet, which it outperforms by only 0.3%. It is also noticed that D-Linknet has fewer false positives for the background class than the others.

A similar study was conducted by Wu et al. (2021) in which high-resolution satellite imagery was segmented to recover buildings. The study utilised the WHU AERIAL Building dataset, which was separated into 4736 training photos, 1036 validation images, and 2416 testing images. Unet, Linknet, DLinknet, SegNet, RefineNet, DeepLabV3+, HRLinkNetv1, and HRLinkNetv2 were among the eight architectures evaluated. The training hyperparameters included a batch size of 6, a learning rate of 0.01, and a training epoch of 30. From best to worst performing, the order of the architectures were HRLinkNetv2, HRLinknetv1, DeeplabV3+, RefineNet, Unet, DLinknet, SegNet, and Linknet, with their IoU scores being 89.53%, 89.40%, 87.31%, 86.93%, 86.20%, 85.73%, 85.56%, and 84.91% respectively.

Meanwhile, Manohara Pai et al. (2019) used semantic segmentation to segment between river and land in synthetic aperture radar (SAR) satellite images. In their study, 40 SAR images of several coastal areas of India from the European Copernicus Satellite mission was utilised, 30 of which were used for training and 10 for testing. The original images had a resolution of 5x20m, thus they were rescaled to 512x512. A vanilla Unet

and a Unet with pretrained weights on the ISBI 2015 Cell Tracking dataset were utilised for the architecture. As training hyperparameters, a batch size of 4, steps of 2500, and an epoch of 5 were applied. The study produced a vanilla Unet model with mIoU (Mean Intersection over Union) 0.9551 and a pretrained Unet model with mIoU 0.9512; however, the authors found that the pretrained Unet could segment small-width rivers whereas the vanilla Unet could not.

Bai et al. (2022) conducted a spatial-temporal analysis of the urban built-up area in a number of Chinese cities. Particularly, the study intends to extract characteristics of cities' built-up regions across different time periods, and then utilise those characteristics to determine the behaviour of expansion of cities. However, the segmentation process only occurs during feature extraction; hence, just this portion of the study will be described. For the raw dataset, the authors collected composite nighttime and vegetation photos of various parts of China from 2012 and 2021, obtained from the National Oceanic and Atmospheric Administration (NOAA) and the United States Geological Survey (USGS), respectively. In the meantime, ground truth labels were gathered using European Space Agency landcover pictures (ESA). Both the raw and label photos were then cropped to 128x128, and 1294 images were designated as training images and 142 as validation images. The primary architecture studied is the CBAM_Unet model with differing hyperparameters. The learning rates employed were 0.1, 0.01, 0.001, and 0.0001. Moreover, the batch sizes were 8 and 16. The authors observed that greater learning rates resulted in faster convergence times but higher converged loss values, concluding that the optimal learning rate was 0.001. Regarding the performance of batch sizes, batch size 16 has fewer local changes in classification accuracy and was therefore the ideal batch size. This ideal CBAM Unet model is subsequently compared to a baseline Unet and two non-convolutional networks techniques, the Support Vector Machine (SVM) and Random Forest (RF). The results indicated that convolutional networks were the most accurate, with CBAM Unet and vanilla Unet achieving mIoU scores of 0.7480 and 0.7342, respectively, while SVM and RF achieved scores of 0.6648 and 0.6643.

However, a growing number of studies have also been done for closer-to-ground images, such as from above-ground cameras or UAVs. For example, Muhadi et al. (2021) examined the implementation of semantic segmentation to estimate the water levels of rivers from images obtained by surveillance cameras. In their study, data was first collected from the European Flood 2013 dataset compiled by Computer Vision Group of the University of Jena, and images collected by the Department of Irrigation and Drainage, Malaysia. Only clear photos were used, and so 710 manually sorted images of dimensions 533x800 were used as the study dataset, which was then further split into 60% training, 20% validation and 20% testing images. The architectures used were the DeepLabV3+ network with a pretrained ResNet-18 backbone, and the SegNet network with a VGG-16 backbone. The segmentation conducted by the authors was binary, i.e. consisting of two classes, with the first class being the river water and the second being the background. The images were trained for up to 30 epochs, and the hyperparameters set were the stochastic gradient descent momentum at 0.9, the batch sizes (8 for DeepLabV3+, 4 for SegNet), L2 regularisations (0.005 for DeepLabV3+, 0.0005 for SegNet), and learning rates (0.003 for DeepLabV3+, 0.001 for SegNet). It was found that DeepLabV3+ outperformed SegNet with the former's IoU score of 94.37% compared to the latter's score of 91.05%. The exact approach the authors used to then estimate the water level is not related to segmentation, and therefore will not be discussed in detail. Of particular interest however is that the estimated water levels based on data from the segmented images were largely able to water levels – along with the fluctuations of them – recorded by dedicated water level sensors, with the maximum difference in measurements between the two at around 1 metre.

Mahmud et al. (2021) implemented semantic segmentation to perform segmentation of roads in images. Data was collected by capturing aerial images by a UAV flying 30 metres above the ground, capturing features such as buildings, trees and shadows. In total, 300 images with a resolution of 5472x3648 were captured which were then divided into 180 training dataset and the rest as testing dataset; the training and validation dataset were separated by randomly shuffling all the datasets. Like in study by Muhadi et al., the study by Mahmud et al. is a binary segmentation task, of which they

divided the classes into road and background. There is only one architecture used in the study, which is DeepLabV3+, however two backbones were examined which were ResNet-50 and Mobile-NetV2. After training for 30 epochs, it was found that the RestNet-50 backbone (mIoU of 89.79%) outperformed the Mobile-NetV2 backbone (84.87%).

Tsellou et al. (2022) meanwhile studied the segmentation of power lines in Greece from images captured from UAVs. The dataset used were compiled from publicly available datasets, namely the Power Line Dataset of Mountain Scene (PLDU) – which captures urban scenes – and the Power Line Dataset of Urban Scene (PLDM) – which captures mountains scenes – as well as video frames from the Hellenic Electricity Distributor Network Operator S.A., combining to a total of 771 training image samples and 185 testing image samples. The architectures studies were D-Linknet with a ResNet18 backbone, DeepLabV3 with a MobileNet backbone, FCN-8 and Unet; the implemented FCN-8 and Unet architectures had no backbones. For all architectures, a learning rate of 0.001 was used, while for the epoch numbers, Unet was trained up to 10 epochs while the rest up to 6 epochs. From best to worst performing, the order of the architectures is D-Linknet, DeepLabV3, FCN-8, and Unet, with each gaining an F1 score of 0.9829, 0.9285, 0.9238, and 0.7985 respectively.

2.8 ISSUE OF SMALL DATASET

Although the literature shows great promise in semantic segmentation, much of the studies involved have had the luxury of large datasets, due in part to the collaborative nature of them. As this project is conducted independently, access to relevant data is much scarcer, therefore the issue of small dataset is much more prominent.

Some researchers have studied approaches to mitigate the disadvantages that small datasets have. Ma et al. (2019) emphasizes the importance of data augmentation in small dataset conditions. The most popular method of data augmentation is flipping. It is the

most popular due to its simplicity yet effectiveness. Raw dataset images are simply flipped horizontally and vertically, and these flipped images already double the number of dataset images. Images can be flipped multiple times along different axes to create many more images. The authors note that flipping can improve the mIoU of non-augmented dataset from 73.28% to 87.24%.

From the literature that were found, the concern of small datasets seems to be more prevalent in medical applications, where specific medical conditions create more niche needs. Rania et al. (2020) studied semantic segmentation for the purpose of healing assessments for diabetic foot ulcers. The authors note that in their field of work, image collection is not easily accessible and image annotation is a critical task. The paper therefore had only 92 training images and 22 test images. To mitigate overfitting due to this small dataset size, the authors augmented their training images in four different manners, i.e. horizontal flipping, rotations, translations and zooms. This increased their number of training images to 368. As a result of data augmentation, the authors managed to obtain good results for their segmentation purposes, getting an IoU of 94.86% for Unet, 92.17% for VNet, and 87.87% for SegNet.

2.9 CHAPTER SUMMARY

In this chapter, the surrounding concepts behind convolutional networks like the convolution operation, the pooling operation, and the activation function was first discussed. Then, the development of convolution networks – whereby convolutional neural networks were a breakthrough due to their efficiency in classification tasks but were not suited for segmentation tasks – led to the development of the many types of fully convolutional networks. Finally, recent literature in sediment identification in other techniques as well as aerial image segmentation were looked into to determine the feasibility of semantic segmentation for river segmentation purposes.

CHAPTER THREE

EXPERIMENTAL SETUP

3.1 INTRODUCTION

This chapter will discuss in depth the experimental setup of the project, beginning with data collection, data preparation, parameter setting, and concluding with training of the dataset.

In the parameter setting and dataset setup, the segmentation models library created by Iakubovskii (2019) will be used to generate all three Unet, Linknet and FPN architectures in code, as well as their training parameters. These three specific architectures will be examined in this project as they are ones available in the library. A major advantage of using the segmentation models library is its great ease of use; the preferred network architecture can be set by simply typing their name in and thereby doing away with having to construct the architecture in code from scratch. Another advantage is peace of mind; with an MIT license, the library is open source meaning that permissions for private and commercial use by others have been granted.

Following the research methodology in Chapter 1.4, the experimental setup is thereby divided into these following steps in consecutive order: data collection, data preparation, parameter setting, and model training. The last step, i.e. model training, contains further breakdowns into the implemented network architectures, evaluation metric Intersection over Union (IoU) employed, and code execution. Figure 18 below depicts a flowchart of the experimental setup employed.

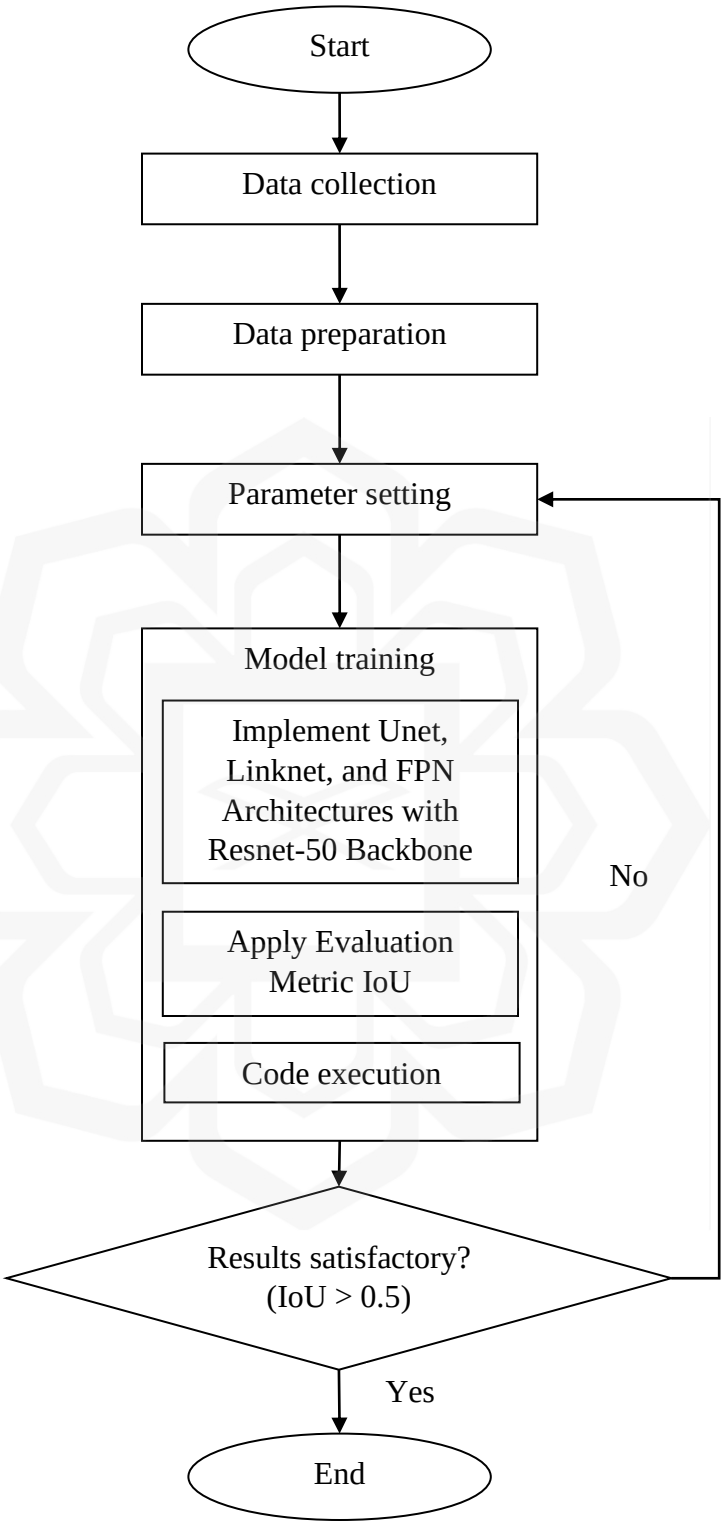


Figure 18 Experimental setup flowchart.

Most of the implemented code in this chapter has been adapted from documentation by Bhattiprolu (2022). Much credit is hence directed to him.

3.2 DATA COLLECTION

Images of the IIUM river were taken along the course of the river. This is so that different types of backgrounds can be captured, thereby better representing the river's actual conditions. A total of 100 images make up the dataset. Afterward, the images are annotated for ground truth masks using an annotation tool available on the website APEER.com. There are four classes labelled in total: unlabeled, river body, river sediment, and background. They are represented by the grayscale values 0, 1, 2, and 3, respectively (shown in Figure 19). The existence of a class for "unlabelled" is a peculiarity of the annotation tool, as the background must be explicitly labelled as a separate class. This “unlabelled” class, however, cannot be removed. Due to the fact that the unlabeled class is irrelevant to this project, only the background, river body, and river sediment classes will be discussed in the chapters that follow.

Figure 19 (Left) an example of a dataset image and (right) its corresponding ground truth mask. The ground truth masks are marked in grayscale values where 1 is river body, 2 is the river sediment, and 3 is the background.

The equipment used to capture these images is a Sony Xperia 1 smartphone which has a 12-megapixel main camera. Moreover, images by the phone camera have the dimensions 4032 pixels in width and 2268 pixels in height; these are hence the dimensions of the raw dataset images. All the images are saved in the PNG format.

3.3 DATASET PREPARATION

Before any training of images could be done, the dataset to be trained must be prepared first. With only 100 images, this project's dataset is significantly smaller than the majority of segmentation applications in the scientific literature. Therefore, according to the

literature reviewed, the images must be subdivided into patches of smaller dimensions, or patchified, to increase the total number of images in the dataset. Since the dimensions of the patchified dataset images will change, the dataset masks should be patchified in the same way so that both the patchified images and masks have the same dimensions.

The following pseudocode carries out the image patchifying process. Note that the root directory and patch size must be defined before the patchifying process can begin.

```
SET root directory

SET patch size TO 512

#Execute patchifying FOR dataset images

image directory = root directory + "images”

from patchify IMPORT patchify #patchify library needs to be imported first

FOR path, subdirectories, and files IN image directory:
   SET subdirectory BY splitting and separating path
   LIST all image names IN this subdirectory
   FOR each image name IN image directory
      IF image name ends with ".JPG":
         OUTPUT image name
         READ image as BGR
         DIVIDE image width to nearest size divisible by patch size
         DIVIDE image height to nearest size divisible by patch size
         CREATE image object from array
         CROP image at origin and at nearest patch size divisible size
         SET image TO Numpy array
         OUTPUT to subdirectory path+image_name
         SET image patchify to 512 width, 512 height, and 512 step

         FOR image width range:
            FOR image height range:

               SET single image patch TO image patch with 512 width and 512 height

               SAVE single image patch to root directory + "512_patches/images/" +
                     “image name” +"patch number” + ".JPG"
```

To patchify the dataset images, the pseudocode is as follows:

```
mask directory = root directory + "masks”

FOR path, subdirectories, and files IN mask directory:
   SET subdirectory BY splitting and separating path
   LIST all mask names IN this subdirectory
   FOR each mask name IN image directory
      IF mask name ends with ".tiff":
         OUTPUT mask name
         READ mask ask grayscale
         DIVIDE mask width to nearest size divisible by patch size
         DIVIDE mask height to nearest size divisible by patch size
         CREATE mask object from array
         CROP mask at origin and at nearest patch size divisible size
         SET mask TO Numpy array
         OUTPUT to subdirectory path + mask_name
         SET mask patchify to 512 width, 512 height, and 512 step

         FOR mask width range:
            FOR mask height range:

               SET single mask patch TO mask patch with 512 width and 512 height

               SAVE single mask patch to root directory + "512_patches/masks/" +
                     “mask name” +"patch number” + ".tiff"
```

For this project, all the dataset images and masks have been patchified to 512 by 512 pixels. This is the optimal patch size for the project, as smaller dimensions would contain less context information for each image patch, and memory constraints of the experimental equipment prevented the use of larger patch dimensions.

After patchification, the dataset will be divided into training and testing sets. The "splitfolders" library performs this division automatically. For the purposes of this project, 75% of the dataset consists of training data, while the remaining 25% consists of validation data. The splitfolders library application is as follows:

```
IMPORT splitfolders

SET INPUT_folder TO 'data/512_patches/i'
SET output_folder TO 'data/data_for_training_and_testing/'
```

```
SPLITFOLDERS from INPUT_folder
OUTPUT to output_folder with ratio 0.75 training and 0.25 validation
```

Following the patchification procedure and the division of the folders into training and validation datasets, 2100 training images and masks and 700 validation images and masks were generated, respectively. For the images to be properly trained, the training and validation datasets must be organised according to the following directory tree:

```
Data/
   train_images/
            train/
               img1, img2, img3, ......

   train_masks/
            train/
               msk1, msk, msk3, ......

   val_images/
            val/
               img1, img2, img3, ......

   val_masks/
            val/
               msk1, msk, msk3, ......
```

3.4 PARAMETER SETTING

Once the dataset is prepared, the parameters and conditions for training the model are then to be readied. Among these include defining the batch size, number of classes, network backbone and preprocessing the dataset.

Batch size is the quantity of samples to be trained in a single iteration. The greater the batch size, the more datapoints there will be, resulting in a more accurate gradient descent. However, due to memory limitations, the maximum batch size could only be set to eight.

As mentioned previously, the project's scope is to segment a river image into three which is the river, the river sediment, and the background. However, due to the peculiarity of the annotation tool used, there are in actual fact 4 classes contained in the mask, containing one extra undefined mask with a value of 0, despite this mask being of no use. Because of this, the number of classes should be set to 4.

Table 1 below summarises the training hyperparameters employed in this project.

Table 1 Training hyperparameters.

Hyperparameter	**Value**
Batch size	8
Number of classes	4

The backbone refers to the type of encoder of the network. As discussed in the literature review, the encoder is the phase in which the input image tensors undergo convolution operations for feature extraction before they are resized back in the decoder phase. For the purposes of this project, the ResNet50 is utilised. A detailed description regarding this backbone structure is available in Chapter 3.5.

The image masks' 0, 1, 2, and 3 grayscale values are unintelligible to the computer in their unprocessed form. Before these grayscale values can be used as class labels, they must undergo preprocessing. This is performed by the Keras function to_categorical. To set the ResNet50 backbone and preprocess the data, the following pseudocode are used:

```
SET BACKBONE TO 'resnet50'
SET preprocess INPUT based on BACKBONE

from Sklearn preprocessing package IMPORT MinMaxScaler
SET scaler TO MinMaxScaler
from Keras utils package IMPORT to_categorical

DEFINE preprocess data
   Reshape image with scaler
   Preprocess image based on BACKBONE
   SET mask TO to_categorical
```

```
RETURN
```

Next, the dataset is to undergo data augmentation. Data augmentation refers to random transformations that are applied to a dataset. The purpose of this is to generate new data points from the existing dataset in order to artificially increase the size of the dataset. Due to the limited number of available dataset image files, this is an important step to take. For the purposes of this project, both the dataset images and masks must be horizontally and vertically flipped. Other forms of augmentation, such as scaling and rotation, exist but are problematic for semantic segmentation applications because they leave gaps that must be interpolated and alter the pixel values of the masks. For data augmentation, the ImageDataGenerator function is utilised, which enables data augmentation to be performed in real-time during training.

```
from Tensorflow Keras IMPORT ImageDataGenerator

DEFINE FUNCTION trainGenerator

    SET image data_generator_arguments TO horizontal flip=True,
                vertical flip=True,
                fill mode=reflect)

    SET image data generation TO ImageDataGenerator based on image data generator arguments
    SET mask data generation TO ImageDataGenerator based on image data generator arguments

    SET image generator TO image datageneration from training image directory
        SET class_mode TO None,
        SET batch_size TO batch_size,
        SET seed TO seed)

    SET mask_generator TO mask datageneration from training mask directory,
        SET class mode TO None,
        SET color mode TO 'grayscale',
        SET batch size TO batch_size,
        SET seed TO seed)

    SET train_generator TO zip based on image_generator and mask_generator

    FOR images and masks IN train generator:
```

```
    SET image and mask TO preprocess data
    YIELD images and masks
```

3.5 MODEL TRAINING

The next step is to conduct model training of the network architectures. The following subchapters further elaborates on the implementation of the architectures, the evaluation metric and loss function utilised, and the implemented code in Tensorflow Keras to train the models.

3.5.1 Network Architecture Implementation

There are some differences between the Unet, Linknet, and FPN network architectures discussed in Chapter 2 and the architectures to be implemented in code. The presence of a backbone structure at the encoder phase is the primary distinction between the "plain" network architectures described in Chapter 2 and those utilised in the experimental setup. The subsections that follow elaborate further on the implemented architectures.

3.5.1.1 Backbone Structure

A backbone structure in the context of deep learning network architectures refers to a modified structure implanted at the encoder phase of an architecture that is comprised of a prior network that was independently pretrained with a large image database in advance. As this modified network is located at the encoder phase of the overall network architecture, its purpose is to perform feature extraction of the images that are passed through the network.

One example of an image database upon which numerous backbone structures have been trained is ImageNet. Although this database was previously mentioned in

Chapter 2.1 in the context of the ILSVRC, the entire database contains a much larger variety of objects, from various species of mammals and flowers to various types of vehicles and musical instruments, totaling 5247 distinct classes of objects (or, as the authors call them, synonym sets or synsets) comprised of 3.2 million images. Backbone structures that have been trained on such a massive image database will have "learned" weights for extracting generic features at multiple levels for a wide range of contexts. In 2D convolutional operations, weights correspond to the element-wise values of a convolutional kernel.

Having a backbone structure in a fully convolutional network architecture allows to take advantage of these "learned" pretrained weights. Having been trained on such a large number of images of diverse objects, it is hoped that the kernels can detect features of the input tensor much more robustly. A pretrained network from a large database such as Imagenet should aid the overall network in learning the lower-level features such as lines, curves, and edges present in an input image, despite the fact that Imagenet does not explicitly contain river images for training and learning its features. By incorporating a pretrained network into their architecture, the overall Unet, Linknet, and FPN networks should experience faster training times and greater training accuracies.

3.5.1.2 ResNet-50 Network

For the purposes of this project, the ResNet-50 backbone network trained on Imagenet is incorporated into all three of the primary network architectures under consideration. ResNet was first introduced by He et al. (2015), which has since become a popular backbone structure in the field of computer vision that employs convolutional networks. The structure is composed of residual blocks arranged in sequences. An input x is propagated through a series of weighted layers (i.e. convolutional processes with varying and activation functions) within a residual block; let $F(x)$ denote the state of the input during this process. Just before the end of the block, a skip connection from the input side adds x, the input "residue," to $F(x)$. Therefore, the output of the residual block, $H(x)$, is

H(x) = F(x) + x. ResNet blocks use skip connections for the same reason they are used in Unet, Linknet, and Linknet: to better understand the "context" of input data. As the ResNet backbone is located in the encoder phase of the overall network, where feature extraction processes are executed, this ensures that the higher-level feature extractors perform no worse than the lower-level ones. In addition, Atliha & Šešok (2020) note that the skip connections in ResNet enable models to learn more efficiently and that its greater depth compared to other backbone networks such as VGG enables it to capture more complex concepts, thereby making it a more effective encoder.

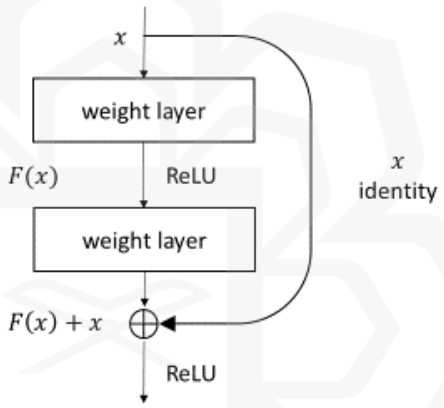


Figure 20 ResNet residual block.

According to the original literature, ResNet should perform better as its structure becomes more complex, with their tests demonstrating that the deepest ResNet layer with 152 layers, also known as ResNet-152, had the lowest top-1 and top-5 training error rates of 21.43% and 5.71% respectively (He et al., 2015). However, the original research was based on tests conducted on the Imagenet database. In the meantime, Zhang et al. (2020) discovered that for small datasets, ResNet networks with deeper layers performed worse than networks with shallower layers. In light of the project's small dataset and Zhang et al.'s findings, Resnet-50 was selected as a compromise for the project's experiments.

The Resnet-50 backbone structure consists of 50 layers, with 49 of them being convolutional layers and the remaining layer being a maxpooling layer (as shown in

Figure 21 and Table 2). For the purpose of clarity, the network is divided into five stages. The input tensor (i.e. the input image's 2D vectors; recall section 2.1.1) is first subjected to a 7x7 convolution with a stride of 2 and 64 kernels, which halves its dimensions. It then enters the second stage, where it first undergoes a 3x3 max pool with stride 2, again halving the dimensions of the tensor, before entering the first series of residual blocks. Note that, contrary to the illustration in Figure 20, the number of layers in a residual block is not always two; in ResNet-50, all residual blocks have three convolutional layers with varying kernel dimensions. Similarly to Figure 20, however, all residual blocks possess a skip connection between their "head" and "tail" ends. In addition, as in Figure 20, a residual block contains a ReLU after all convolutional layers. A residual block comprises a 1x1 convolution of 64 kernels, a 3x3 convolution of 64 kernels, and a 1x1 convolution of 256 kernels in the second stage. A 1x1 convolution of 128 kernels, a 3x3 convolution of 128 kernels, and a 1x1 convolution of 512 kernels are performed in the third stage. A 1x1 convolution of 256 kernels, a 3x3 convolution of 256 kernels, and a 1x1 convolution of 1024 kernels are performed in the fourth stage. A 1x1 convolution of 128 kernels, a 3x3 convolution of 128 kernels, and a 1x1 convolution of 512 kernels are performed in the fifth stage. The residual blocks also occur in different multiples depending on the stage; in the second, third, fourth, and fifth stages, there are 3, 4, 6, and 3 residual blocks, respectively.

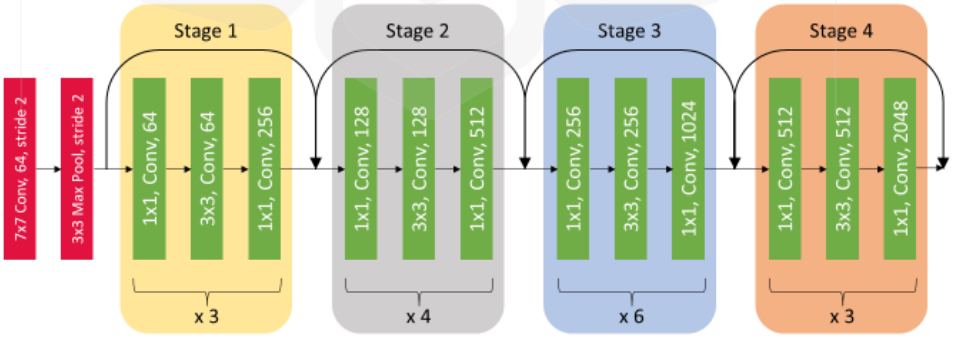


Figure 21 Schematic of ResNet-50.

Table 2 ResNet-50 internal structure (He et al., 2015).

Layer Stage	**ResNet-50**
Conv1	7x7, 64, stride 2
Conv2_x	3x3 max pool, stride 2
	$\begin{bmatrix} 1x1, 64 \\ 3x3, 64 \\ 1x1, 256 \end{bmatrix}$
Conv3_x	$\begin{bmatrix} 1x1, 128 \\ 3x3, 128 \\ 1x1, 512 \end{bmatrix}$
Conv4_x	$\begin{bmatrix} 1x1, 256 \\ 3x3, 256 \\ 1x1, 1024 \end{bmatrix}$
Conv5_x	$\begin{bmatrix} 1x1, 512 \\ 3x3, 512 \\ 1x1, 2048 \end{bmatrix}$

3.5.1.3 Implemented Architectures

With a common Imagenet-pretrained ResNet-50 backbone at the encoder phases of the implemented Unet, Linknet, and FPN, the primary distinctions between the three networks therefore lie at the decoder phase.

3.5.1.3.1 Implemented Unet

After the final ResNet stage, the tensor is upsampled using a 4x4 transpose convolution with a stride of 2. It then enters the first decoder level and is concatenated with the fourth ResNet stage to generate 128 feature maps. Two rounds of 3x3 convolution and ReLU are followed by a 4x4 stride 2 transpose convolution before upsampling. All subsequent decoder levels undergo concatenations with their respective ResNet stage (i.e., decoder level 2 with Resnet stage 3, decoder level 3 with Resnet stage two, and decoder level 4 with ResNet stage 1) and two rounds of 3x3 convolution and ReLU before proceeding to

the next level. The decoder levels 2, 3, and 4 generate 64, 32, and 16 feature maps, respectively. The final step of the decoder phase consists of a 1x1 convolution followed by a softmax activation function that returns the original tensor dimension, i.e. 512x512, and a feature map number corresponding to the number of classes, i.e. 4. Figure 22 is a visual representation of the implemented Unet.

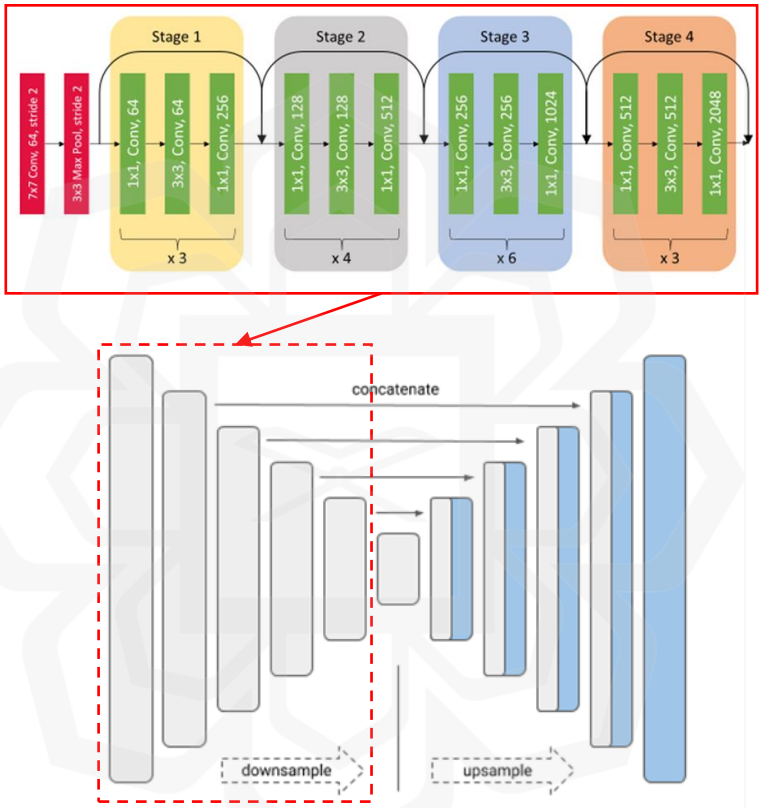


Figure 22 Schematic of implemented Unet with Resnet-50 backbone.

3.5.1.3.2 Implemented Linknet

The implemented Linknet decoder phase bears a striking resemblance to the implemented decoder phase, with the exception of additions at the skip connections as opposed to concatenations. Following the final ResNet stage, the tensor is upsampled via a 4x4

transpose convolution with a stride of 2. It then enters the first decoder level, where it is subjected to summation with the fourth ResNet stage, resulting in 128 feature maps. Two rounds of 3x3 convolution and ReLU are followed by a 4x4 stride 2 transpose convolution before upsampling. Similar to the Unet decoder phase, the subsequent processes at the second through fifth and final decoder levels are fairly comparable, with all levels undergoing additions with their respective ResNet stages (except for the summation operations) and undergoing two rounds of 3x3 convolution and ReLU before proceeding to the next decoder level. The decoder levels 2, 3, and 4 generate 64, 32, and 16 feature maps, respectively. The final step of the decoder phase is a 1x1 convolution followed by a softmax activation function that produces an output with a 512x512 dimension and a feature map number of 4. Below is a visual diagram of the implemented Linknet, depicted in Figure 23.

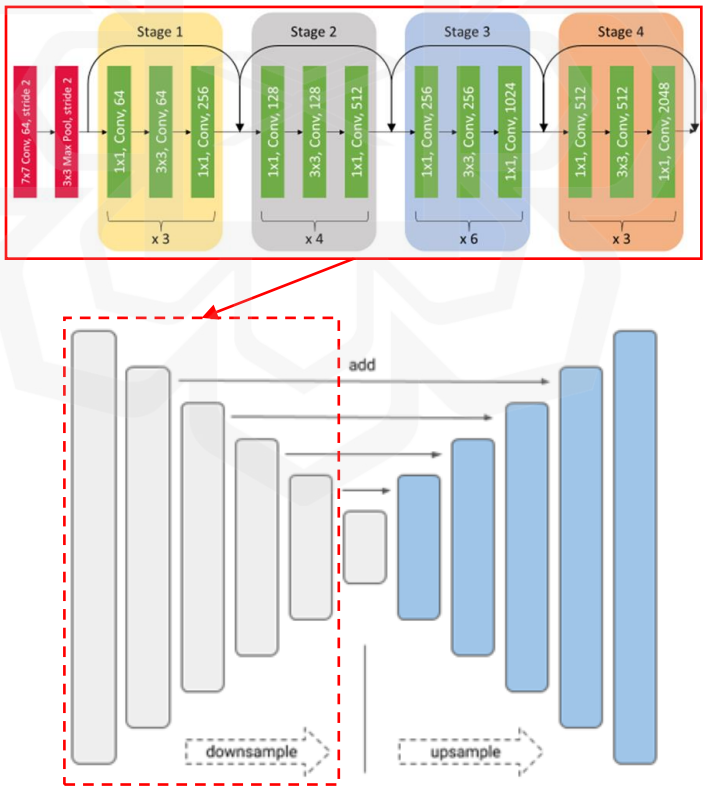


Figure 23 Schematic of implemented Linknet with Resnet-50 backbone.

3.5.1.3.3 Implemented FPN

As FPN's architecture differs significantly from Unet and Linknet, so does its decoder phase process flow. First, unlike the implemented Unet and Linknet networks, only the nearest neighbour algorithm is used for upsampling. After ResNet, the tensor is upsampled three times by 2x nearest neighbour to produce a total of four feature pyramid levels. Similar to Linknet, the operation of the encoder phase's skip connection is summation. There is a minor distinction whereby a 1x1 convolution is performed just before the skip connection reaches the summation node. Each of the four levels of the feature pyramid is then subjected to an aliasing-reduction operation consisting of two 3x3 convolutions in series. Next, the pyramid levels are upsampled individually such that their dimensions become identical. To accomplish this, the top level of the pyramid is upsampled by 8x, the level below it by 6x, the level below that by 4x, and the level below that by 2x. Next is the concatenation of all upsampled pyramid levels, which is not present in either Unet or Linknet. The final step is a 3x3 convolution followed by a softmax activation with a single level output from the aggregate concatenation. Figure 24 is a visual representation of the implemented FPN.

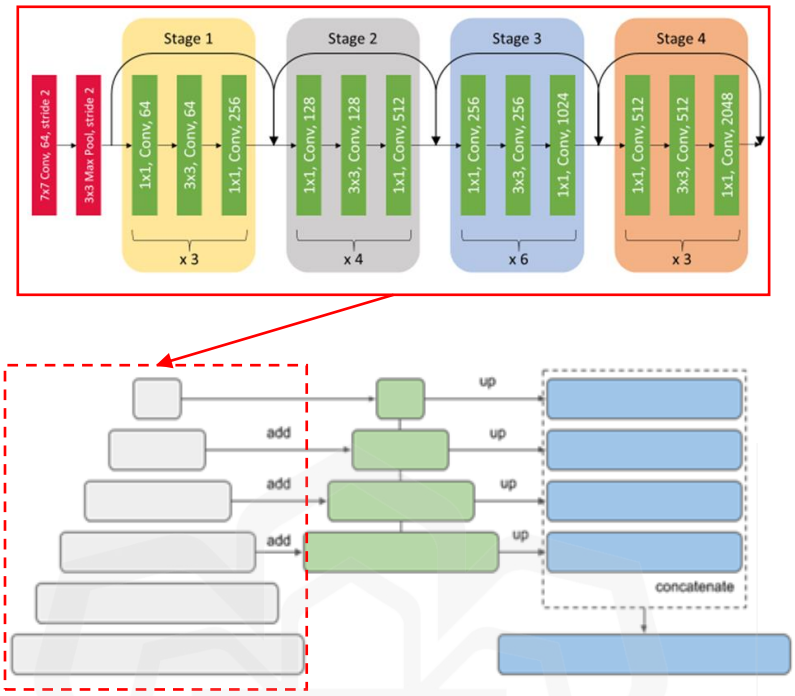


Figure 24 Schematic of implemented FPN with Resnet-50 backbone.

3.5.1.3.4 Comparison of Implemented Architectures

Table 3 below summarises a comparison between the three implemented architectures based on various characteristics, i.e. the shape of the architecture, backbone utilised, upsampling method, and skip connection operation employed.

Table 3 Table of comparison of the implemented architectures.

Characteristics	**Architectures**		
	Unet	**Linknet**	**FPN**
Shape	Generally U-shaped		Pyramidal; multi-column
Backbone	ResNet-50		
Upsampling	Transpose Convolution		Nearest Neighbour

Method			
Skip Connection Operation	Concatenation	Summation	Summation (preceded by 1x1 convolution)

3.5.2 Evaluation Metric and Loss Function

To evaluate the performance of the network architectures, there must be a method for quantitatively judging their results. To achieve this, an evaluation metric is utilised. IoU, also known as intersection over union, is a popular metric for evaluating image segmentation tasks. By definition, intersection over union is the ratio between the area of overlap between the predicted truth and the ground truth and the area of union between the predicted truth and the ground truth. In the context of semantic segmentation, it is the ratio between the number of pixels for which the predicted truth matches the ground truth and the total number of pixels present in both the predicted truth and the ground truth. Another way to describe IoU is that it is the measure of how well a model predicts the true positives relative to all other possibilities. Indeed, Keras's definition includes the following equation for IoU (Tensorflow, 2022):

$$IoU = \frac{TP}{TN+FP+FN} \tag{3}$$

where TP is true positives, TN is true negatives, FP is false positives, and FN is false negatives.

IoU is frequently considered a superior metric of a model's performance than simple accuracy. In an accuracy metric, the extent to which a model correctly identifies a true negative – in semantic segmentation, correctly predicting that a pixel indeed does not belong to an object class – is considered. The issue with this is when a class imbalance exists, the evaluations of the majority class may overshadow those of the minority classes. In contrast, the intersection over union metric is not concerned with evaluating true negatives and is a much better indicator of segmentation task success(Van Beers et al., 2019). In consideration of this, the IoU metric was selected for this project.

With the IoU as the chosen evaluation metric, this means that the most suitable loss function is the Jaccard loss. IoU and Jaccard only constitutes a difference in terminology; both refer to the same thing and their mathematical formulas are identical. However, IoU seem to be the preferred term for the evaluation metric, while Jaccard is preferred to denote the loss function.

3.5.3 Code Implementation

Model metrics and the network model itself must be defined just prior to training. These metrics include the number of sample batches that must be produced before one epoch concludes. The steps per epoch are defined for both training images and masks. The steps per epoch for this project were determined by dividing the number of training images/masks by the batch size.

In addition, the architecture of the network to be trained is also defined at this stage. Furthermore, the types of loss functions and evaluation metrics to be used are specified.

```
Define the model metrics

SET number of training images TO number of images in
        'data/data_for_keras_aug/train_images/train/' directory
SET number of validation images TO number of images in
        'data/data_for_keras_aug/val_images/val/' directory
SET steps per epoch TO number of training images divided by batch size
SET validation steps per epoch TO number of validation images divided by batch_size

SET number of classes to 4

#Define the model

SET model TO Unet with BACKBONE, encoder weights SET TO 'imagenet',
        classes SET TO number of classes, activation SET TO 'softmax')
COMPILE mode with ‘Adam optimiser', loss SET TO ‘jaccard loss’, metrics SET TO
        IoU score
```

In the above commands, the Unet architecture is defined along with the Jaccard loss function and the IoU evaluation metric. To train for other network architectures, the "Unet" text is simply replaced with "Linknet" or "FPN."

The epoch number, 100 in this case, is then set, and training of the network model can begin.

```
Train model with data augmented training images,
        steps per epoch SET TO steps per epoch,
        epochs SET TO 100,
        verbose SET TO 1,
        validation data SET TO data augmented validation images,
        validation steps SET TO validation steps per epoch)

Save model as ‘trainedmodel.hdf5’
```

To obtain the IoU metrics, both the mean IoU and per class IoU, the following pseudocode is run.

```
from Keras models IMPORT load_model

from Tensorflow Keras Metrics IMPORT IoU

SET model TO load_model FROM ‘trainedmodel.hdf5’

SET number of classes TO 4
SET IOU keras TO MeanIoU on number of classes
UDATE IOU keras state
OUTPUT IOU keras result as “Mean IoU”

SET IOU keras perclass TO IoU target class ID 0
UPDATE IOU keras perclass state
OUTPUT IOU keras perclass result as "Class 0 (unlabeled) IoU”

SET IOU_keras_perclass TO IoU target class ID 1
UPDATE IOU keras perclass state
OUTPUT IOU keras perclass result as "Class 1 (river) IoU"

SET IOU_keras_perclass TO IoU target class ID  2
UPDATE IOU keras perclass state
OUTPUT IOU keras perclass state as "Class 2 (sediment) IoU"
```

SET IOU_keras_perclass TO IoU target class ID 3 UPDATE IOU keras perclass state OUTPUT IOU keras perclass state as "Class 3 (background) IoU"

The training of the models was conducted on an Alienware laptop. More detailed specifications of the laptop can be found in Table 4 below.

Table 4 Computer used for model training.

Specifications	Description
Model Name	Alienware m15 Ryzen Ed. 5
CPU	AMD Ryzen 9 5900HX @ 3.30 GHz
RAM	16.0 GB
GPU	Nvidia GeForce RTX 3070 Laptop GPU w/ 8.0 GB dedicated memory

3.6 CHAPTER SUMMARY

In this chapter, the steps to conduct the project's experiment were comprehended. Starting with the river image dataset collection by phone camera, these images are then patchified to divide them up into smaller pieces and thereby increase the dataset size. After being split into training and validation images, various parameters such as batch size, number of classes, backbone used, and data augmentation were set. After completing these steps, the models of the three architectures are finally trained.

CHAPTER FOUR

RESULTS AND DISCUSSIONS

4.1 INTRODUCTION

To evaluate the performances of the trained models, three factors were investigated. The first is the epoch progressions of the training and validation losses and IoU scores of the models. Next, the IoU per class scores of the models were inspected and compared. Finally, a qualitative examination of segmentation predictions of selected dataset images was undertaken.

4.2 TRAINING AND VALIDATION RESULTS

The charts below show the training and validation losses and IoU scores for the three analysed semantic segmentation architectures throughout the 100 epochs.

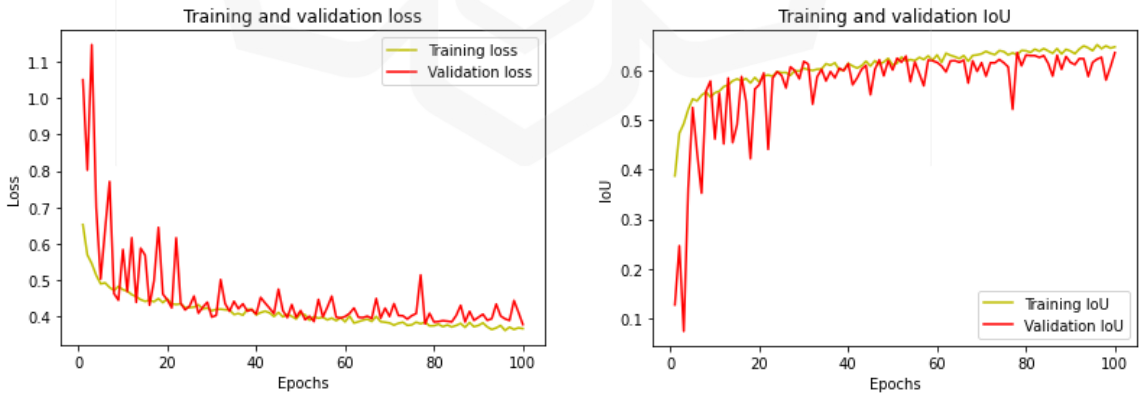


Figure 25 Unet epoch progression charts.

The progression graphs for Unet are depicted in Figure 25. Looking at the Unet loss chart, the training loss begins at approximately 0.66 and decreases sharply to approximately 0.47 after 10 epochs, before becoming gradually linear over the next 100 epochs to reach just under 0.4. The validation loss initially fluctuates between 1.05 and 1.15 before rapidly decreasing to approximately 0.5 after approximately 5 epochs. The progression remains erratic until it stabilises after approximately 22 epochs, at which point the validation loss stabilises just above 0.4 after 100 epochs.

Looking at the Unet IoU scores, after 10 epochs the training score increases sharply from approximately 0.4 to 0.57. After that, the score increases linearly until it reaches around 0.63 after 100 epochs. After around five epochs, the validation score increases sharply to roughly 0.53; however, the progression remains erratic until about 30 epochs, after which it becomes less so. After 100 iterations, the validation score reaches approximately 0.6.

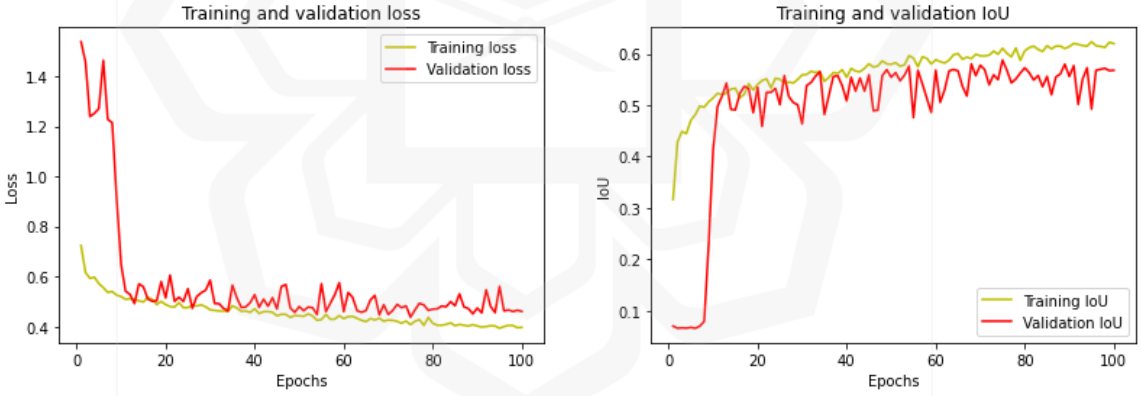


Figure 26 Linknet epoch progression charts.

Figure 26 depicts the Linknet progression charts. After about fifteen epochs, the training loss quickly decreases from around 0.75 to around 0.5 when examining the Linknet losses chart. After 100 epochs, the training loss level is just above 0.4, marking the end of the gradual decline. After about 12 epochs, the validation loss rapidly

decreases from around 1.5 to around 0.5. After 100 epochs, the value fluctuates slightly, but the trend is largely linear, levelling out at roughly 0.5.

Observing the Linknet IoU scores chart, the training IoU score increases sharply from around 0.32 at the start to around 0.55 after approximately 18 epochs. At the end of 100 epochs, the training score is roughly 0.6. Interestingly, the validation IoU score almost does not improve until about 8 epochs, when it suddenly jumps from around 0.05 to 0.55 at around 7 epochs. Up until the end, the score progression for validations remains somewhat erratic, culminating in a score of 0.55.

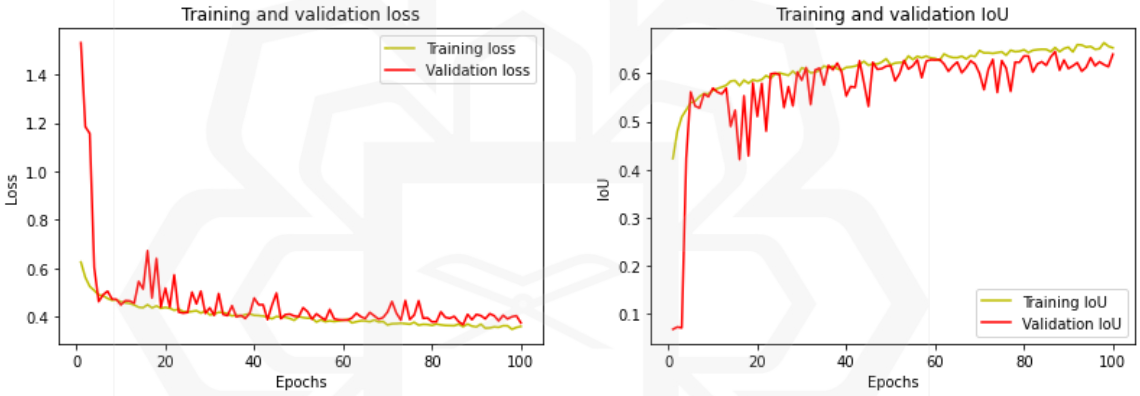


Figure 27 FPN epoch progression charts.

Figure 27 displays the FPN progression charts. Looking at the FPN losses chart, the training loss decreases from an initial value of around 0.62 to around 0.45 after 5 epochs. At the conclusion of 100 epochs, its progression is relatively linear, reaching approximately 0.4. The validation loss decreases sharply from an initial value of about 1.55 to about 0.45. Its behaviour is quite erratic until about 17 epochs, when it spikes briefly to around 0.65, after which it stabilises somewhat, reaching just over 0.4 after 100 epochs.

Observing the FPN IoU scores chart, the training score quickly increases from an initial score of around 0.42 to around 0.55 after about 15 epochs. Its behaviour thereafter follows a gradual positive linear trend until it reaches a final score of roughly 0.63. After

around four epochs, the validation score increases from roughly 0.5 at the start to around 0.57. Its behaviour continues to be quite erratic for the next 20 epochs, with a brief dip to about 0.42 at around 18 epochs, before stabilising to a final score of approximately 0.6.

Inspecting figures 25 to 27, one feature that is common in the progression charts across all architectures is the relatively high levels of jaggedness of the curves, especially the ones for validation, throughout the model training cycle. Moreover, even if subsiding somewhat, the jaggedness in the charts is still noticeable even at the latter epoch stages. This is in contrast to progression charts found in literature, such as by Qian (2019), whereby both the training and validation charts remain relatively smooth throughout all epoch stages. The consistent presence of jaggedness in this project's progression charts might thereby suggest the occurrence of overfitting in all the models trained. With the current dataset used, it seems that during the validation phase of the model training, the model is not able to "lock on" consistently to the river features based on and as described by the ground truth masks, hence the erratic and jagged behavious of the validation curves observed.

Observing the graphs reveals that the behaviour of progression for both the losses and the IoU scores is relatively similar across all architectures. There are slight differences, for example the charts trends for Unet is slightly more fluctuated after the first 20 epochs or so. In spite of this, the final validation losses and IoU scores for all architectures are remarkably similar, averaging between 0.40 and 0.60. On the other hand, the erratic and jagged behaviour of the validation curves in the progression charts suggests that overfitting is present in the trained models. This means that the models may not be able to detect river features in a generalised manner.

4.3 IOU SCORES

The obtained inter-class mean IoU scores as well as per-class IoU scores for each network architecture as obtained by the Keras functions are as displayed in Table 5 below:

Table 5 Implemented architecture IoU scores.

IoU Class	IoU Scores per Architecure		
	Unet	**Linknet**	**FPN**
Mean	0.916234	0.87452227	0.707427
River	0.978273	0.89490145	0.9216773
Sediment	0.83446103	0.8188789	0.20392573
Background	0.93596166	0.9097864	0.9966786

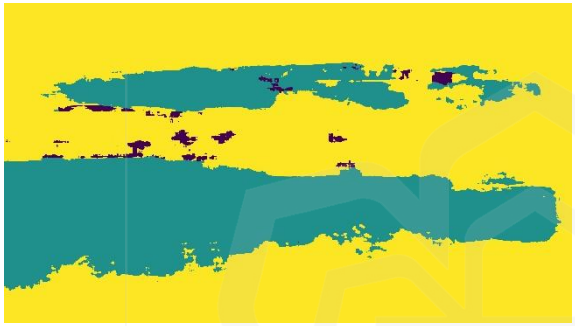
Since the objective of this project is to evaluate river sediment identification performance, only the sediment class IoU scores are considered. Both Unet and Linknet achieve IoU scores of approximately 0.8, with Unet scoring slightly higher at 0.83 compared to 0.81 for Linknet. In contrast, FPN's score is significantly lower at roughly 0.2, well below 50%. According to these results, Unet should be the model with the best performance, followed by Linknet and FPN. In order to further validate the performance of each architecture, a qualitative evaluation of the actual image predictions made by the three network models is presented in the subchapter below.

4.4 SEGMENTATION PREDICTION COMPARISONS

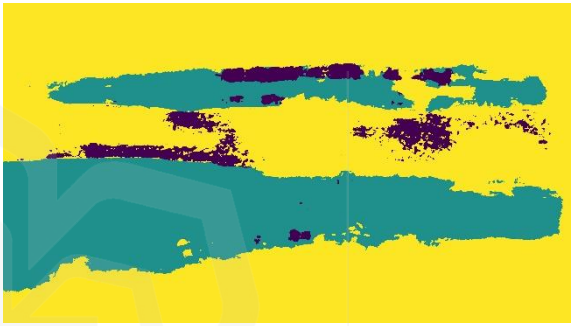
In this image prediction analysis, the ground truth segmentation masks and predicted segmentation masks of the following images are compared. These images were chosen to provide a glimpse of the generalised performance of the three architectures, as they were captured at far away river segments with vastly different river course and sedimentation profile characteristics.

Original image

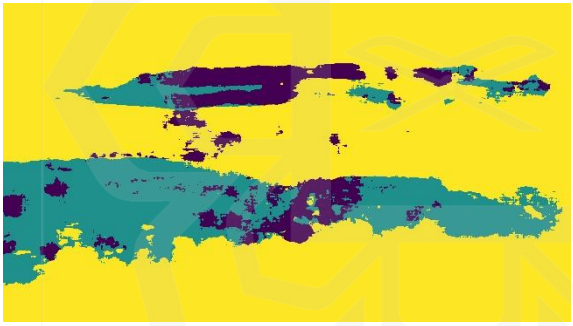
Ground truth mask



Unet



Linknet



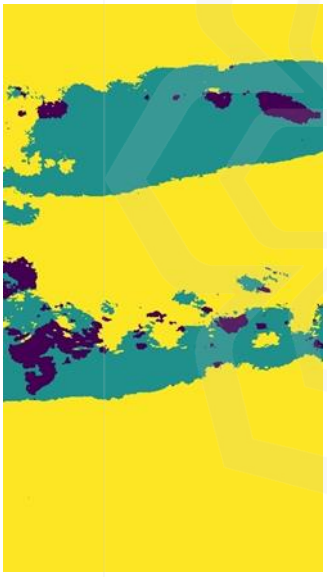
FPN

Figure 28 Original image, ground truth mask, and predicted masks for image #1.

Original image

Ground truth mask

Unet



Linknet

FPN

Figure 29 Original image, ground truth mask, and predicted masks for image #2.

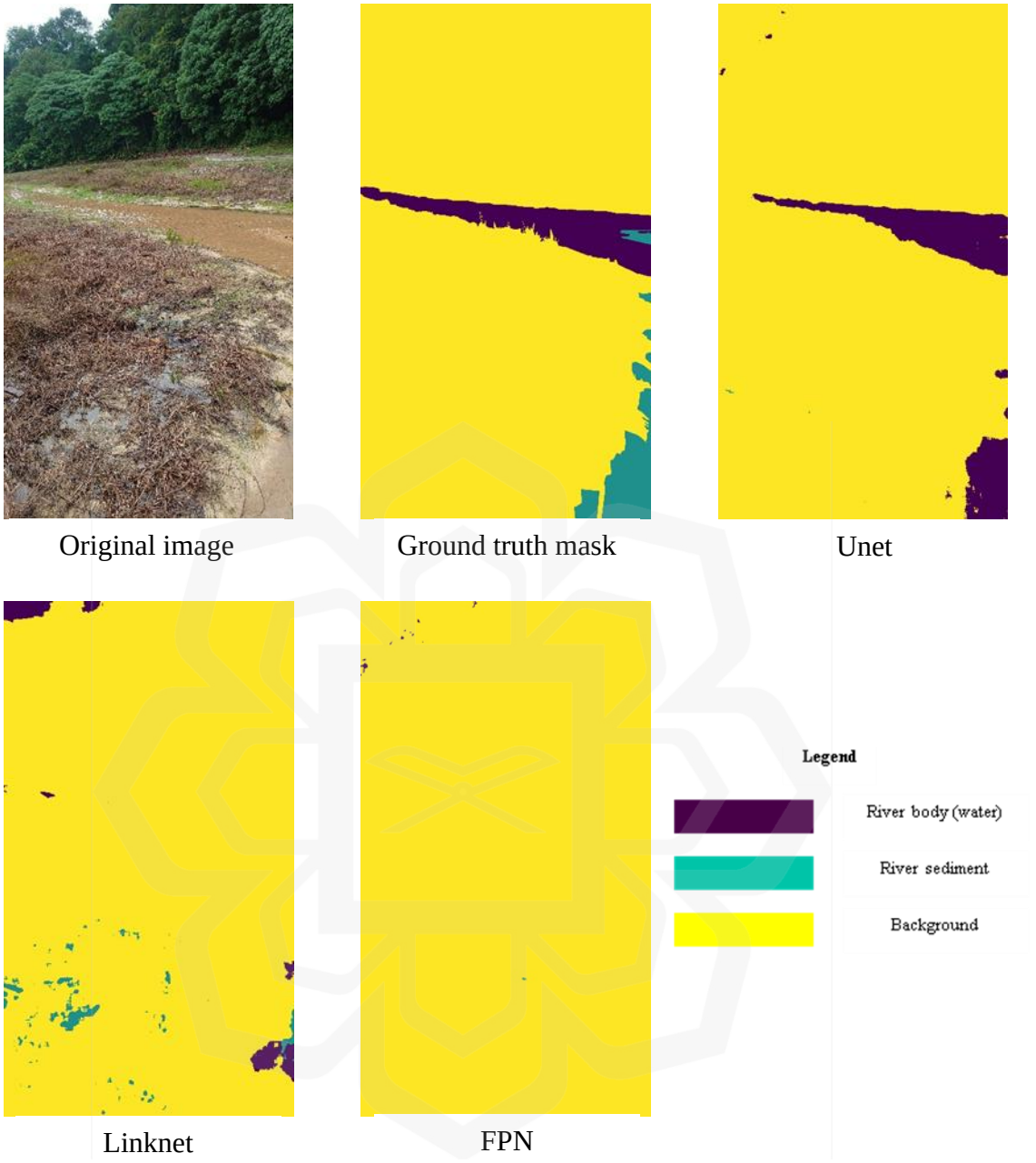


Figure 30 Original image, ground truth mask, and predicted masks for image #3.

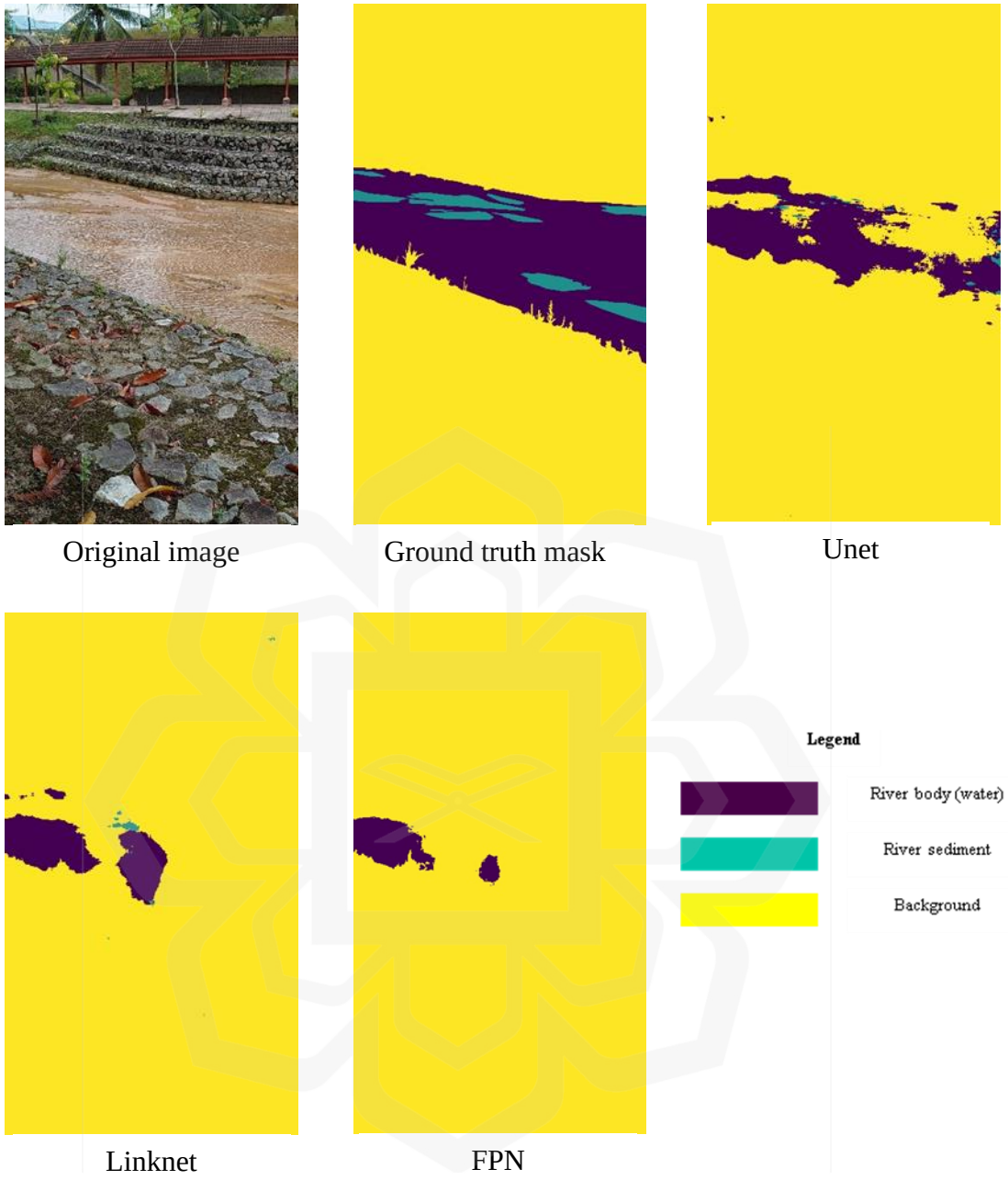


Figure 31 Original image, ground truth mask, and predicted masks for image #4.

Observing the mask predictions of the preceding images demonstrates that the models perform differently depending on the sedimentation situation in each image. In images #1 (Figure 28) and #2 (Figure 29), cream-colored sedimentation has formed quite uniformly along the predominantly green and brown riverbank, with sedimentation

forming on both sides of the bank in image #1 and only one side in image #2. Nevertheless, the models appear generally capable of identifying the sedimentation formed in both instances. Unet was able to predict the majority of sedimentation on both sides of the bank, whereas Linknet mislabeled some pixels on the upper bank as the river body and FPN did so on both sides of the bank. There is an interesting observation in image #2 whereby all three models incorrectly predicted the existence of a sediment formation on the right bank, which is not the case. Unet was however able to accurately predict the majority of sedimentation pixels on the left bank, whereas Linknet misclassified more of the sedimentation as background. Intriguingly, FPN almost entirely incorrectly misidentified the true sedimentation as the river's body. In instances where sedimentation is uniform, the results correspond to the obtained IoU metrics, with Unet performing the best, followed by Linknet and FPN.

When the sedimentation formed is not uniform and/or the distinction between the classes is not readily apparent, however, all three models appear to fail. In image #3 (Figure 30), the colour and texture distinctions between the various classes are not as distinct. Although Unet correctly labelled the river body, it incorrectly labelled the sedimentation as part of the river body. Linknet and FPN erroneously labelled the entire image as the background. In image #4 (Figure 31), the colour and texture distinction between the river's body and sediment is also not distinct. In addition, the sedimentation formations are not uniform but rather comprise isolated clusters in the middle of the river. None of the models could correctly classify these sediment clusters.

4.5 DISCUSSIONS

Overall, based on the obtained results, Unet outperforms both Linknet and FPN. Unet has an edge over Linknet because it was able to correctly predict sedimentation with slightly fewer mislabelings than Linknet. Recalling their respective structures, the primary distinction between the two lies in the operation following the skip connection: in Unet, the encoder and decoder layers are concatenated, whereas in Linknet, the operation

is a summation. It is possible that the concatenation operation provides the architecture with a better and more comprehensive context of the spatial information of the object classes than summation does. In summation-formed skip connections, two feature maps from the encoder and decoder levels are combined into a single resultant map. On the other hand, in skip connections by concatenation, feature maps from the encoder level are appended or conjoined to those from the corresponding decoder level. This process entails that the tensors from the encoder and decoder are not "blended" but rather "stuck" onto one another. Therefore, the nature of concatenation may lead to improved retention of encoder-level features, resulting in an overall more accurate predictive model.

In comparison to Unet and Linknet, the FPN model yields noticeably inferior results. FPN has a significantly different structure than the other two architectures, so it is important to discuss it separately. Looking back to the structure of the FPN architecture, a distinguishing feature of it when compared to Unet and Linknet is its anti-aliasing block. To recall the purpose of anti-aliasing blocks, they are designed to prevent the folding of higher frequency signals onto the low-frequency portion of the original signal, which can result in distortions and artefacts. However, they involve two convolution processes in series, each of which halves the dimensions of the feature maps, meaning that some information that was initially present at the encoder may have been lost. Due to the presence of the anti-aliasing block, skip connections designed to preserve encoder context may not be as effective in FPN as they are in Unet and Linknet. Consequently, in the prediction of river sediment images, FPN may have had more difficulty distinguishing subtle differences between object classes (such as a minor colour difference between river body and river sediments) than the other architectures.

On the other hand, it was demonstrated that none of the architecture models perform well when river sedimentation is not formed uniformly or when it is formed in isolated clusters. Indeed, referring back to section 4.2, the jagged progression charts obtained during model training suggest that overfitting may be present in all the models, meaning that they are expected to not perform well in generalised detection of river features. Qualitative analysis conducted in section 4.4 thereby provides evidence for this

in that only uniform sediment clusters were able to be detected and that non-uniform sedimentation is poorly accounted for.

A reason for this non-generalised sediment identification performance may be the large intraclass differences in all the images, and at times a not a distinct enough difference between classes. This intraclass difference can be seen in the backgrounds of all the images, where the stone walls adjacent to the river and the adjacent greenery are considered to belong to the same class. Meanwhile, an example of the small interclass distinction can be seen in image #4, where the colour gradation of the sediment is very similar to that of the river body, with only slight differences in the textures of the two distinguishing classes. Both variables may have contributed to the architectures becoming "confused" regarding which image segment belongs to which class.

According to the conducted literature review, intraclass differences are in fact not uncommon. For instance, in Muhadi et al. (2021), the background class includes various elements such as the sky, surrounding buildings, and vegetation. The study however is a binary segmentation task, so there are only two competing classes. Presumably, the difference between the river body and the various features within the background class was large enough that semantic segmentation was able to be done accurately.

Therefore, in order to mitigate both the problem of large intraclass differences and the problem of small interclass differences, more classes could be assigned to the trained image. For instance, stone walls and riverbank greenery can be divided into distinct classes. This may allow the architecture to distinguish between objects more effectively, since more classes indicate that differences in characteristics such as color, texture, and shape correspond to distinct object classes. This was demonstrated by Zhu et al. (2019), who demonstrate that labelling an image into six distinct classes enables IoU scores to exceed 80%.

Alternatively, semantic segmentation can be conducted using newer, more novel architectures. Although the segmentation models library is extremely user-friendly, it restricts the user to the architectures it contains. In the literature, newer architectures, particularly those employing dilation and atrous convolutions, have consistently

outperformed older architectures. Downsides to using novel architectures may include the fact that writing code to build each architecture is more laborious than the simple "plug-and-play" convenience provided by the segmentation library, along with that there may be usage permission restrictions.

4.6 CHAPTER SUMMARY

This chapter discusses the results of the river segmentation conducted by this project. While the progression patterns of training and validation losses and IoU for all architectures are fairly similar, the final Keras IoU results indicate that Unet is the model with the best performance, followed by Linknet and FPN. The IoU results from Keras are confirmed by qualitatively assessing the predicted masks of select sample images. However, the models only appear to perform reliably when sedimentation formations are uniform; none of the models performed well against non-uniform sedimentation formations.

Table 6 below provides a summary of the performances of all the architecture models based to their epoch progression trends, IoU scores for the river sediment class, uniform sedimentation performances, and non-uniform sedimentation performances;

Table 6 Summary table of implemented architecture performances.

	Architectures		
	Unet	**Linknet**	**FPN**
Epoch progression	Initial fluctuations, stabilises gradually		
Sediment IoU Scores	0.83446103	0.8188789	0.20392573
Sediment Prediction Performance	Good with less mislabeling	Good with slightly more mislabeling	Mediocre; much more mislabeling

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1 CONCLUSION

This project investigated the semantic segmentation approach for river sediment identification. Based on the literature review, the semantic segmentation approach as a whole as well as its underlying concepts were comprehended. Then, the development of convolutional networks and their applications in semantic segmentation applications were investigated. After becoming familiar with semantic segmentation and convolution networks, various network architectures to segment and identify sedimentation formation in the IIUM river were implemented, and then their performances are evaluated.

The first objective was to train different semantic segmentation network architectures to perform IIUM river sediment identification. This objective was achieved by first collecting images of the IIUM river, then labelling them into river body, river sediment, and background classes as ground truths. Next, data preparation, parameter setting, and model training were performed using Python code. The trained models were thus used to finally generate the segmentation masks.

The second objective was to evaluate and compare the performances of the trained semantic segmentation architectures in identifying river sediments. Comparing the architectures of Unet, Linknet and FPN, it was discovered that Unet performed the best, followed by Linknet, and then FPN. In spite of this, the experimental results indicate that the architectures are only useful in specific cases where river sedimentation is uniform, and that none of the examined architectures were robust enough to perform generalised river sediment identification in instances where non-uniform sedimentation formations occur.

Therefore, in conclusion, the project fulfills its objectives and there are key points to be taken. Namely, semantic segmentation for identifying river sediments can be done with relatively minimal resources. However, algorithm models that are trained with minimal resources would only be able to identify case-specific, uniform sedimentation. More robust models would thus need to be trained using more extensive resources, including time, money, and effort.

5.2 CONTRIBUTIONS

- This project shows that with the segmentation models library, training semantic segmentation algorithm models to identify river sediments can be done relatively easily and with minimal resources.
- With minimal resources used in the project, the trained models are however shown to only be effective for identifying case-specific, uniform sedimentation.

5.3 RECOMMENDATIONS

- One recommendation is to increase the number of object classes, such as classifying different objects in the background as their own classes, to provide better distinction of image features and create less confusion for the trained models
- Another recommendation is to utilise more robust architectures such as those that include dilation atrous convolutions as compared to regular convolutions which are able to better take into account large intraclass differences

REFERENCES

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2018). Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, *2018-January*, 1–6.

Atliha, V., & Šešok, D. (2020). Comparison of VGG and ResNet used as Encoders for Image Captioning. *2020 IEEE Open Conference of Electrical, Electronic and Information Sciences (EStream)*, 1–4.

Bai, M., Zhang, S., Wang, X., Feng, Y., Wang, J., & Peng, P. (2022). Deep Semantic Segmentation for Rapid Extraction and Spatial-Temporal Expansion Variation Analysis of China's Urban Built-Up Areas. *Frontiers in Earth Science*, *10*.

Bhattiprolu, S. (2022). *Python for Microscopists*. GitHub. https://github.com/bnsreenu/python_for_microscopists

Chaurasia, A., & Culurciello, E. (2017). *LinkNet: Exploiting Encoder Representations for Efficient Semantic Segmentation*.

Elijah, O., Rahman, T. A., Yeen, H. C., Leow, C. Y., Sarijari, M. A., Aris, A., Salleh, J., & Han, C. T. (2018). Application of UAV and Low Power Wide Area Communication Technology for Monitoring of River Water Quality. *2018 2nd International Conference on Smart Sensors and Application (ICSSA)*, 105–110.

Es-Sabery, F., Hair, A., Qadir, J., Sainz-De-Abajo, B., Garcia-Zapirain, B., & Torre-DIez, I. (2021). Sentence-Level Classification Using Parallel Fuzzy Deep Learning Classifier. *IEEE Access*, *9*, 17943–17985.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2013). *Rich feature hierarchies for accurate object detection and semantic segmentation*. http://arxiv.org/abs/1311.2524

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep Residual Learning for Image Recognition*. http://arxiv.org/abs/1512.03385

Hosseini, H., Xiao, B., Jaiswal, M., & Poovendran, R. (2017). *On the Limitation of Convolutional Neural Networks in Recognizing Negative Images*. http://arxiv.org/abs/1703.06857

Hussein, S., & Ali, K. (2022). Semantic Segmentation of Aerial Images Using U-Net Architecture. *Iraqi Journal for Electrical and Electronic Engineering*, *18*(1), 58–63.

Iakubovskii, P. (2019). *Segmentation Models*. GitHub. https://github.com/qubvel/segmentation_models

Jin, Z., Feng, H., Luo, Q., Li, Y., Wei, J., & Li, J. (2020). Oil Spill Detection in Quad-Polarimetric SAR Images Using an Advanced Convolutional Neural Network Based on SuperPixel Model. *Remote Sensing*, *12*, 944.

Kirillov, A., He, K., Girshick, R., & Dollár, P. (n.d.). *A Unified Architecture for Instance and Semantic Segmentation*.

Lecun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. In *Nature* (Vol. 521, Issue 7553, pp. 436–444). Nature Publishing Group.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., & Belongie, S. (2016). *Feature Pyramid Networks for Object Detection*. http://arxiv.org/abs/1612.03144

Liu, H., Xu, K., Li, B., Han, Y., & Li, G. (2019). Sediment identification using machine learning classifiers in a mixed-texture dredge pit of Louisiana shelf for coastal restoration. *Water (Switzerland)*, *11*(6).

Long, J., Shelhamer, E., & Darrell, T. (2014). *Fully Convolutional Networks for Semantic Segmentation*. http://arxiv.org/abs/1411.4038

Ma, R., Tao, P., & Tang, H. (2019). Optimizing data augmentation for semantic segmentation on small-scale dataset. *ACM International Conference Proceeding Series*, 77–81.

Mahmud, M. N., Osman, M. K., Ismail, A. P., Ahmad, F., Ahmad, K. A., & Ibrahim, A. (2021). Road Image Segmentation using Unmanned Aerial Vehicle Images and DeepLab V3+ Semantic Segmentation Model. *Proceedings - 2021 11th IEEE International Conference on Control System, Computing and Engineering, ICCSCE 2021*, 176–181.

Manohara Pai, M. M., Mehrotra, V., Aiyar, S., Verma, U., & Pai, R. M. (2019). Automatic segmentation of river and land in SAR images: A deep learning approach. *Proceedings - IEEE 2nd International Conference on Artificial Intelligence and Knowledge Engineering, AIKE 2019*, 15–20.

Manso-Callejo, M. Á., Cira, C. I., Alcarria, R., & Arranz-Justel, J. J. (2020). Optimizing the recognition and feature extraction of wind turbines through hybrid semantic segmentation architectures. *Remote Sensing*, *12*(22), 1–16.

Muhadi, N. A., Abdullah, A. F., Bejo, S. K., Mahadi, M. R., & Mijic, A. (2021). Deep learning semantic segmentation for water level estimation using surveillance camera. *Applied Sciences (Switzerland)*, *11*(20).

Pineux, N., Lisein, J., Swerts, G., Bielders, C. L., Lejeune, P., Colinet, G., & Degré, A. (2017). Can DEM time series produced by UAV be used to quantify diffuse erosion in an agricultural watershed? *Geomorphology*, *280*, 122–136.

Qian, K. (2019). Automated Detection of Steel Defects via Machine Learning Based on Real-Time Semantic Segmentation. *Proceedings of the 3rd International Conference on Video and Image Processing*, 42–46.

Rania, N., Douzi, H., Yves, L., & Sylvie, T. (2020). Semantic segmentation of diabetic foot ulcer images: Dealing with small dataset in dl approaches. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12119 LNCS*, 162–169.

Ribeiro, A. H., & Schön, T. B. (2021). *How Convolutional Neural Networks Deal with Aliasing*. http://arxiv.org/abs/2102.07757

Ronneberger, O., Fischer, P., & Brox, T. (2015). *U-Net: Convolutional Networks for Biomedical Image Segmentation*. http://arxiv.org/abs/1505.04597

Roushangar, K., & Shahnazi, S. (2020). Prediction of sediment transport rates in gravel-bed rivers using Gaussian process regression. *Journal of Hydroinformatics*, *22*(2), 249–262.

Rukundo, O., & Cao, H. (2012). Nearest Neighbor Value Interpolation. In *IJACSA) International Journal of Advanced Computer Science and Applications* (Vol. 3, Issue 4). www.ijacsa.thesai.org

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., & Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252.

Șerban, A., Ilaș, G., & Porușniuc, G.-C. (2014). *SpotTheFake: An Initial Report on a New CNN-Enhanced Platform for Counterfeit Goods Detection*.

Sharma, S., Sharma, S., & Athaiya, A. (2020). ACTIVATION FUNCTIONS IN NEURAL NETWORKS. In *International Journal of Engineering Applied Sciences and Technology* (Vol. 4). http://www.ijeast.com

Tensorflow. (2022). *tf.keras.metrics.IoU*. Tensorflow. https://www.tensorflow.org/api_docs/python/tf/keras/metrics/IoU

Thoma, M. (2016). *A Survey of Semantic Segmentation*. http://arxiv.org/abs/1602.06541

Tsellou, A., Moirogiorgou, K., Plokamakis, G., Livanos, G., Kalaitzakis, K., & Zervakis, M. (2022). Aerial video inspection of Greek power lines structures using machine learning techniques. *IST 2022 - IEEE International Conference on Imaging Systems and Techniques, Proceedings*.

Uzun, U., & Temizel, A. (2022). Cycle-Spinning Convolution for Object Detection. *IEEE Access*, *10*, 76340–76350.

Van Beers, F., Lindström, A., Okafor, E., & Wiering, M. A. (2019). Deep neural networks with intersection over union loss for binary image segmentation. *ICPRAM 2019 - Proceedings of the 8th International Conference on Pattern Recognition Applications and Methods*, 438–445.

Wahid, M. F., Shahriar, M. F., & Sobuj, M. S. I. (2021). A Classical Approach to Handcrafted Feature Extraction Techniques for Bangla Handwritten Digit Recognition. *Proceedings of International Conference on Electronics, Communications and Information Technology, ICECIT 2021*.

Wu, M., Shu, Z., Zhang, J., & Hu, X. (2021). HRLINKNET: LINKNET WITH HIGH-RESOLUTION REPRESENTATION FOR HIGH-RESOLUTION SATELLITE IMAGERY. *International Geoscience and Remote Sensing Symposium (IGARSS)*, 2504–2507.

Yamashita, R., Nishio, M., Do, R. K. G., & Togashi, K. (2018). Convolutional neural networks: an overview and application in radiology. In *Insights into Imaging* (Vol. 9, Issue 4, pp. 611–629). Springer Verlag.

Zhang, R., Du, L., Xiao, Q., & Liu, J. (2020). Comparison of Backbones for Semantic Segmentation Network. *Journal of Physics: Conference Series*, *1544*(1).

Zhu, Q., Zheng, Y., Jiang, Y., & Yang, J. (2019). Efficient Multi-Class Semantic Segmentation of High Resolution Aerial Imagery with Dilated LinkNet. *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, 1065–1068.