Road Segmentation in Single View Images

Cong Thuan Nguyen

Master's thesis
2024

Tomas Bata University in Zlín
Faculty of Applied Informatics

Tomas Bata University in Zlín
Faculty of Applied Informatics
Department of Informatics and Artificial Intelligence

Academic year: 2023/2024

ASSIGNMENT OF DIPLOMA THESIS

(project, art work, art performance)

Name and surname:	**Cong Thuan Nguyen**
Personal number:	**A22758**
Study programme:	**N0613A140023 Information Technologies**
Specialization:	**Software Engineering**
Type of Study:	**Full-time**
Work topic:	**Segmentace silnic z jednopohledových obrázků**
Work topic in English:	**Road Segmentation in Single View Images**

Theses guidelines

1. Make a literature review of state-of-the-art methods for segmentation.
2. Evaluate limitations of the existing road segmentation methods in single-view images.
3. Choose deep learning or transfer learning models and design strategies to improve the robustness of road segmentation algorithms in diverse environmental conditions.
4. Collect, add or propose a suitable dataset to test the proposed strategies.
5. Evaluate the achieved results in quantitative and qualitative way.

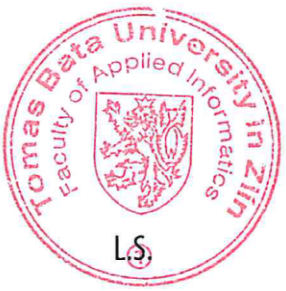
Form processing of diploma thesis: **printed/electronic**
Language of elaboration: **English**

Recommended resources:

1. GOLLAPUDI, Sunila; GOLLAPUDI, Sunila. OpenCV with Python. *Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs*, 2019, 31-50.
2. ADRIAN, Rosebrock. Practical Python and OpenCV: An Introductory, Example Driven Guide to Image Processing and Computer Vision. 2016.
3. LI, Ke, et al. Bird's-Eye View Semantic Segmentation for Autonomous Driving through the Large Kernel Attention Encoder and Bilinear-Attention Transform Module. *World Electric Vehicle Journal*, 2023, 14.9: 239.
4. ALOKASI, Haneen; AHMAD, Muhammad Bilal. Deep learning-based frameworks for semantic segmentation of road scenes. *Electronics*, 2022, 11.12: 1884.
5. QASEEM GHADI, Maen; TÖRÖK, Árpád. Comparison of different road segmentation methods. *Promet-Traffic&Transportation*, 2019, 31.2: 163-172.
6. GONG, Shi, et al. Fastroadseg: Fast monocular road segmentation network. *IEEE Transactions on Intelligent Transportation Systems*, 2022, 23.11: 21505-21514.

Supervisors of diploma thesis: **prof. Ing. Zuzana Komínková Oplatková, Ph.D.**
Department of Informatics and Artificial Intelligence

Date of assignment of diploma thesis: **November 5, 2023**
Submission deadline of diploma thesis: **May 13, 2024**



doc. Ing. Jiří Vojtěšek, Ph.D. m.p.
Dean

prof. Mgr. Roman Jašek, Ph.D., DBA m.p.
Head of Department

In Zlín January 5, 2024

I hereby declare that:

- I understand that by submitting my Master´s thesis, I agree to the publication of my work according to Law No. 111/1998, Coll., On Universities and on changes and amendments to other acts (e.g. the Universities Act), as amended by subsequent legislation, without regard to the results of the defence of the thesis.
- I understand that my Master´s Thesis will be stored electronically in the university information system and be made available for on-site inspection, and that a copy of the Master´s Thesis will be stored in the Reference Library of the Faculty of Applied Informatics, Tomas Bata University in Zlín, and that a copy shall be deposited with my Supervisor.
- I am aware of the fact that my Master´s Thesis is fully covered by Act No. 121/2000 Coll. On Copyright, and Rights Related to Copyright, as amended by some other laws (e.g. the Copyright Act), as amended by subsequent legislation; and especially, by §35, Para. 3.
- I understand that, according to §60, Para. 1 of the Copyright Act, TBU in Zlín has the right to conclude licensing agreements relating to the use of scholastic work within the full extent of §12, Para. 4, of the Copyright Act.
- I understand that, according to §60, Para. 2, and Para. 3, of the Copyright Act, I may use my work - Master´s Thesis, or grant a license for its use, only if permitted by the licensing agreement concluded between myself and Tomas Bata University in Zlín with a view to the fact that Tomas Bata University in Zlín must be compensated for any reasonable contribution to covering such expenses/costs as invested by them in the creation of the thesis (up until the full actual amount) shall also be a subject of this licensing agreement.
- I understand that, should the elaboration of the Master´s Thesis include the use of software provided by Tomas Bata University in Zlín or other such entities strictly for study and research purposes (i.e. only for non-commercial use), the results of my Master´s Thesis cannot be used for commercial purposes.
- I understand that, if the output of my Master´s Thesis is any software product(s), this/these shall equally be considered as part of the thesis, as well as any source codes, or files from which the project is composed. Not submitting any part of this/these component(s) may be a reason for the non-defence of my thesis.

I herewith declare that:

- I have worked on my thesis alone and duly cited any literature I have used. In the case of the publication of the results of my thesis, I shall be listed as co-author.
- That the submitted version of the thesis and its electronic version uploaded to IS/STAG are both identical.

In Zlín; dated: .....

Student´s Signature

ABSTRAKT

Tato práce se zaměřuje na vylepšení segmentace silnic v jednopohledových snímcích s využitím omezení existujících metod prostřednictvím adaptace metod hlubokého učení a transferového učení. Provádí analýzu nejmodernějších technik segmentace a zdůrazňuje aktuální výzvy v různých podmínkách prostředí. Strategie jsou navrženy tak, aby poskytovaly přehled o robustnosti algoritmu a využívaly schopnosti hlubokého učení pro extrakci funkcí a rozpoznávání vzorů. Ke komplexnímu testování algoritmů za určitých povětrnostních podmínek se používají nejmodernější datové sady. Výsledky jsou hodnoceny pomocí přesnosti, vyvolání, skóre F1 a výsledku segmentace vizuálních kontrol. To má přispět k náročné doméně funkcí bezpečného řízení a nahlédnutí do reálné efektivity modelů pro segmentaci silnic.

Klíčová slova: Segmentace silnic, umělá inteligence, strojové učení, neuronová síť, segmentace obrazu

ABSTRACT

This thesis focuses on enhancing road segmentation in single-view images using the limitations of existing methods through the adaptation of deep learning and transfer learning methods. It conducts to analyse state-of-the-art segmentation techniques, highlighting current challenges in diverse environmental conditions. The strategies are designed to give insights about the algorithm robustness, leveraging the capabilities of deep learning for feature extraction and pattern recognition. State-of-the-art datasets are used to test the algorithms comprehensively in certain weather conditions. Results are evaluated using precision, recall, F1 score, and the segmentation outcome of visual inspections. This is to contribute to the challenging domain of safety driving functionalities and insights into the real-world effectiveness of the models for road segmentation.

Keywords: Road segmentation, AI, Artificial Intelligence, Machine Learning, Neural Network, Image Segmentation

ACKNOWLEDGEMENTS

Acknowledgements, motto and a declaration of honour saying that the print version of the Bachelor's/Master's thesis and the electronic version of the thesis deposited in the IS/STAG system are identical, worded as follows:

I hereby declare that the print version of my Bachelor's/Master's thesis and the electronic version of my thesis deposited in the IS/STAG system are identical.

CONTENTS

INTRODUCTION

Road segmentation in single-view photos is important in many real-world applications, including autonomous driving, traffic monitoring, and urban planning. Road borders must be accurately delineated in order for cars and pedestrians to navigate safely and efficiently. However, creating strong and accurate road segmentation remains a difficult challenge due to a variety of environmental elements such as illumination, weather, and occlusions.

This thesis aims to address the limitations of existing road segmentation methods in single-view images by leveraging advancements in deep learning and transfer learning techniques. The primary motivation behind this research is to enhance the robustness and accuracy of road segmentation models, thereby improving their performance across diverse environmental conditions that some original model such as Urban scene segmentation using U-Net model cannot offer the robustness and accuracy of segmentation solutions. Many solutions with strong accuracy are also kept confidential so they are less accessible to researchers.

This thesis will talk about the principles of Artificial Intelligence and the related topics such as Deep learning algorithms and Neural Networks and then give insights into the models which are used in segmentation and their limitations. A brief introduction to the tools which are used to aid the research of this topic will be stated.

In summary, this thesis endeavours to advance the state-of-the-art in road segmentation by improving and advancing deep learning-based approaches using U-Net with VGG16 and to compare with state-of-the-art FCN and regular U-Net segmentation models that show the improvement of robustness and accuracy in single-view images, aiming to contribute towards safer and more efficient navigation systems in real-world environments through meticulous experimentation and evaluation.

I THEORY

1 ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) is a multifaceted technology that is revolutionising various fields by allowing people to reconsider how we combine data, evaluate information, and use the resultant insights to make better decisions. AI is nowadays the most emerging tool that it can aid to our daily lives. In this section, a brief introduction of AI will be discussed and how AI models can be constructed [1].

1.1 History of AI

Artificial intelligence was first introduced by computer scientists Marvin Minsky and John McCarthy in 1956. It was giving a breeze to gather scientists in order to research a new method of the interaction between humans and machines where machines could mimic human's action [2].

AI started to evolve even bigger which first integrated ruled-base systems using predefined rules to make decisions. It kept growing more complex with the desire of scientists that they wanted to expand the system into knowledge-based models [2].

Neural networks were later on getting attention from scientists which led to a significant role of machine learning algorithms in AI domain [2].

There was a decline state of AI but then it resurged and became more powerful in computation and machine learning. Deep learning or deep neural networks became dominance of research fields. A huge number of new tools based on AI models were developed and introduced to the society thanks to the computational power keeping increasing and improving year by year [2].

Nowadays AI have been presenting in every corner of the world. It is a helpful tool to ease manual and time-consuming tasks of our daily activities. It will more and more become powerful with endless research in the field around the world. The challenges of AI are motivating to improve and expand it [2].

1.2 Machine learning

The study of developing algorithms and statistical models that allow computer systems to learn from and make predictions or judgments based on data is the focus of the artificial intelligence (AI) subfield of machine learning (ML). The main goal of machine learning is

to give computers the capacity to gradually get better at a given task without having to be explicitly programmed [3].

Machine learning techniques can be broadly categorized into three main types based on the learning approach: supervised learning, unsupervised learning, and reinforcement learning. Each type serves different purposes and is suitable for specific types of tasks [4].

1. Supervised Learning:

In supervised learning, the model is trained on a labelled dataset, where the input data is paired with corresponding target outputs. The goal is to learn a mapping from input features to the target outputs, allowing the model to make predictions on new, unseen data [3].

Common supervised learning tasks include:

- Classification: Assigning labels to inputs (e.g., spam or not spam, image recognition) [3].
- Regression: Predicting a continuous value (e.g., house prices, stock prices) [3].

2. Unsupervised Learning:

Unsupervised learning involves training a model on an unlabelled dataset without explicit target outputs. The model discovers patterns, structures, or relationships within the data. This type of learning is often used for exploratory data analysis and extracting insights from data [3].

Common unsupervised learning tasks include:

- Clustering: Grouping similar data points based on patterns (e.g., customer segmentation) [3].
- Dimensionality Reduction: Reducing the number of features in a dataset (e.g., principal component analysis) [3].

3. Reinforcement Learning:

Reinforcement learning involves training a model to make sequences of decisions by interacting with an environment. The model receives feedback in the form of rewards or penalties based on its actions, allowing it to learn optimal strategies. Reinforcement learning is often used in scenarios where an agent needs to make a series of decisions over time. Common

reinforcement learning applications include robotics, game playing, and autonomous systems [3].

Additionally, within these main categories, there are several specific machine learning techniques and algorithms. Here are some notable ones:

Supervised Learning Algorithms: Linear Regression, Support Vector Machines (SVM), Decision Trees, Random Forest, k-Nearest Neighbors (k-NN), Neural Networks [4].

Unsupervised Learning Algorithms: K-Means Clustering, Hierarchical Clustering, Principal Component Analysis (PCA), t-Distributed Stochastic Neighbor Embedding (t-SNE), Association Rule Learning (e.g., Apriori algorithm) [4].

Reinforcement Learning Algorithms: Q-Learning, Deep Q Network (DQN), Policy Gradient Methods, Actor-Critic Models [4].

These algorithms can be applied to various domains and tasks, depending on the nature of the data and the goals of the machine learning application. The choice of technique depends on factors such as the availability of labelled data, the nature of the problem, and the desired outcomes [4].

1.3 Data modelling

AI is a wide area of research. However, it is mainly focused on solving real-world problems in result of creating automated and smart systems in various application areas. There are many techniques classified for example machine learning, deep learning and neural networks, data mining, ruled-base knowledge, or fuzzy logic approaches [5].

AI models rely on training data to recognise the patterns and make decisions or predictions based on a chosen problem. In that statement, a structured dataset is required to train an AI model and AI algorithms are used to extract, realise, and parameterise the patterns in the dataset [5].

1.3.1 Datasets

Datasets are collections of data that are used to train and test algorithms and models. These datasets are specifically applied to AI fields and they serve for certain purposes for example

the dataset of iris flower is used to classify three species of Iris (Iris setosa, Iris virginica and Iris versicolor).

1.3.2 Data transformation

Data transformation is a task that converts raw and unorganised data from various sources to a usable and structured data. It is mandatory to convert non-compatible data to usable data such as converting string value to numeric value because algorithms are operations on matrices so a string cannot operate with mathematic operations [6].

Since the size of data is various, data need to be resized to a fixed size because linear models, for example, have fixed number of input nodes, therefore, it is expected to have same size on the dataset.

It is necessary to visualise data. Visualisation can help to find anormal data fractals in the dataset. By that means, noise can be reduced during training and improving outcoming results [6].

1.4 Optimizer

The precision of a model's forecasts is assessed using a loss function. For every training sample, it computes the discrepancy between the expected and actual output.

Minimizing the loss function is the model's objective. The most effective way to determine which combination of parameters will yield the most accurate forecasts is to minimize the loss function [7].

1.4.1 Mean squared error (MSE)

MSE is a widely used loss function in regression issues. It calculates the average squared difference between the projected and actual outputs.

This loss function is sensitive to outliers, which implies that a few extremely big mistakes can have a significant impact on the overall value of the loss function. However, MSE is a common choice due to its differentiability and computational efficiency.

A Python function to calculate the mean squared error (MSE) for a given set of predicted values and actual values is shown as below [7]:

```
def mean_squared_error(predicted_values, actual_values):
    # calculate the squared difference between predicted and actual val-
ues
    squared_differences = [(pred - act) ** 2 for pred, act in zip(pre-
dicted_values, actual_values)]
    # calculate the mean of the squared differences
    mse = sum(squared_differences) / len(squared_differences)
    return mse
```

1.4.2 Mean Absolute Error (MAE)

MAE is another popular loss function for regression problems. MAE calculates the average absolute difference between anticipated and actual values. It responds less to outliers than MSE [7].

```
def mean_absolute_error(predicted_values, actual_values):

    # calculate the absolute difference between predicted and actual val-
ues
    absolute_differences = [abs(pred - act) for pred, act in zip(pre-
dicted_values, actual_values)]
    # calculate the mean of the absolute differences
    mae = sum(absolute_differences) / len(absolute_differences)
    return mae
```

1.4.3 Cross-entropy

Cross-entropy loss is a commonly used loss function in classification issues. It quantifies the difference between the expected and actual probability distributions.

This loss function is especially valuable when the classes are unbalanced, since it can assist to balance the mistakes caused in each class. Depending on the data, you can utilize either Binary or Categorical Cross-entropy [8].

```
def cross_entropy(y_pred, y_true):
    # ensure that the predicted probabilities are in the range [0, 1]
    y_pred = np.clip(y_pred, 1e-15, 1 - 1e-15)

    # calculate the cross-entropy loss
    loss = -np.mean(y_true * np.log(y_pred) + (1 - y_true) * np.log(1 -
y_pred))

    return loss
```

It is important to note that this function assumes that the input arrays have the same number of elements and that the true labels are binary (i.e., either 0 or 1). If the labels are multi-class, one-hot encoding should be applied before calling this function.

1.5 Loss function

Once the loss function has been created, an optimizer is used to modify the model's parameters in order to minimize the loss function. It's also worth noting that these optimizers may be fine-tuned using various variables or hyperparameters like learning rate, momentum, decay rate, and so on.

These optimizers can also be used with other strategies, such as learning rate scheduling, to make the model perform even better.

Below are the three most commonly used optimizers. [9]

1.5.1 Gradient Descent

Gradient descent is one of the most widely used optimizers. It adjusts the model's parameters by taking the derivative of the loss function with respect to the parameters and updating the parameters in the direction of the negative gradient. Gradient descent is simple to implement, but it can be slow to converge when the loss function has many local minima [9].

```
def gradient_descent(model, X, y, learning_rate, num_iterations):
    # obtain the number of training examples
    m = X.shape[0]

    for i in range(num_iterations):
        # make predictions using the current model parameters
        y_pred = model.predict(X)

        # calculate gradients
        grads = model.gradient(X, y, y_pred)

        # update model parameters
        for j in range(len(model.params)):
            model.params[j] = model.params[j] - learning_rate * grads[j]
/ m

    return model
```

It first obtain the number of training examples and then it iterates over the number of iterations. At each iteration it makes predictions using the current model parameters and then it calculates gradients using the provided model.gradient function. Finally, it updates the

model parameters by subtracting the product of learning rate and gradients by the number of examples [9].

1.5.2 Stochastic Gradient Descent (SGD)

SGD is an extension of gradient descent. It updates the model's parameters after each training sample, rather than after each epoch. This makes it faster to converge, but it can also make the optimization process more unstable. Stochastic gradient descent is often used for problems with a large amount of data [9].

```
def stochastic_gradient_descent(model, X, y, learning_rate, num_itera-
tions):

    # obtain the number of training examples
    m = X.shape[0]

    for i in range(num_iterations):
        # shuffle the training data
        X, y = shuffle(X, y)

        # iterate over each training example
        for j in range(m):
            # make predictions using the current model parameters
            y_pred = model.predict(X[j])

            # calculate gradients
            grads = model.gradient(X[j], y[j], y_pred)

            # update model parameters
            for k in range(len(model.params)):
```

1.5.3 Adaptive Moment Estimation (Adam)

Adam is an optimizer that combines the advantages of gradient descent and SGD. It uses the first and second moments of the gradients to adjust the learning rate adaptively. Adam is generally considered to be one of the best optimizers for deep learning [10].

The Adam optimizer is often a good choice for problems with a large number of parameters.

```
import numpy as np

def Adam(params, grads, learning_rate=0.001, beta1=0.9, beta2=0.999, ep-
silon=1e-8):
    # initialize the first and second moment estimates
    m = [np.zeros_like(p) for p in params]
    v = [np.zeros_like(p) for p in params]

    # initialize the time step
    t = 0
```

```
# update the parameters
for p, g, m_, v_ in zip(params, grads, m, v):
    t += 1
    m_ = beta1 * m_ + (1 - beta1) * g
    v_ = beta2 * v_ + (1 - beta2) * np.power(g, 2)
    m_hat = m_ / (1 - beta1 ** t)
    v_hat = v_ / (1 - beta2 ** t)
    p
```

1.6 Training and Testing

In machine learning field or in artificial intelligence field in general, training and testing are essential components that help algorithms recognise the patterns of existing data and make decisions and possibly improve accuracy over time [11].

1.6.1 Training

Training refers to providing an algorithm the data usually labelled or categorised to guide it recognising the patterns inside the data. The dataset can be various from text to image, plain numbers to documents [11].

To reveal the accuracy of the algorithm, testing is performed to see how accurate and understandable the algorithm is to the provided data [11].

1.6.2 Testing

Testing is performed after training finished. Testing data will try to use the data in the testing set in the algorithm that is trained previously. The output will be compared to the result of the training set to create a representation based on input-output correlations, allowing it to generate accurate predictions when exposed to new, unseen data [12].

The concept of testing in machine learning involves evaluating the model's generalization ability, assessing its accuracy, and identifying potential issues. Here are key components of testing in machine learning [11]:

1. **Training and Testing Data Split:**
    - The dataset is typically divided into two subsets: a training set used to train the model and a testing (or validation) set used to assess its performance on unseen data.

- The goal is to simulate the model's performance on real-world data it has not encountered during training [11].

2. **Cross-Validation:**
   - To address concerns about the randomness of data splitting, cross-validation techniques are often employed. Common methods include k-fold cross-validation, where the dataset is divided into k subsets, and the model is trained and evaluated k times, using a different subset as the test set in each iteration [12].
3. **Performance Metrics:**
   - Various metrics are used to evaluate the model's performance, depending on the nature of the problem. Common metrics include accuracy, precision, recall, F1 score, and area under the receiver operating characteristic curve (AUC-ROC) for classification tasks. Mean Squared Error (MSE) and R-squared are commonly used for regression tasks [12].
4. **Overfitting and Underfitting:**
   - Testing helps identify issues of overfitting (model performs well on training data but poorly on new data) and underfitting (model fails to capture patterns in the training data) [12].
   - Regularization techniques and hyperparameter tuning are often applied based on testing results to address these issues [12].
5. **Confusion Matrix and Error Analysis:**
   - A confusion matrix provides a detailed breakdown of the model's performance, showing true positive, true negative, false positive, and false negative counts [12].
   - Error analysis involves investigating specific instances where the model makes mistakes, providing insights into potential improvements [12].
6. **A/B Testing (Deployment Testing):**
   - In deployment, A/B testing can be used to compare the performance of the machine learning model with other models or baseline approaches.
   - Continuous monitoring and testing are essential to ensure the model's effectiveness over time, considering changing data distributions [12].
7. **Ethical and Bias Testing:**

- Testing should also include ethical considerations, checking for biases in the model's predictions and ensuring fairness across different demographic groups [12].
- Regular audits and fairness assessments help mitigate bias-related issues [12].

8. **Robustness Testing:**
   - Assessing the model's robustness involves testing its performance under various conditions, such as different input data distributions, noise, or adversarial attacks [12].

1.7 Measurement

To measure the accuracy of trained models, there are many methods to show the result. Along with the output results of each training epoch, some formal methods are used to measure the accuracy and the loss, for example F1 scores or Confusion Matrix. This chapter will briefly introduce two measurement methods: F1 Scores and Confusion Matrix.

1.7.1 Confusion Matrix

Confusion Matrix, as its name implies, provides us with a matrix as an output and details the model's overall performance.

Assume for a moment that we have a binary classification issue. We have a few examples that fall into either the YES or NO categories. Additionally, we have an in-house classifier that assigns a class to an input sample that is given. We obtain the following outcome when 165 samples are used to test our model [14].

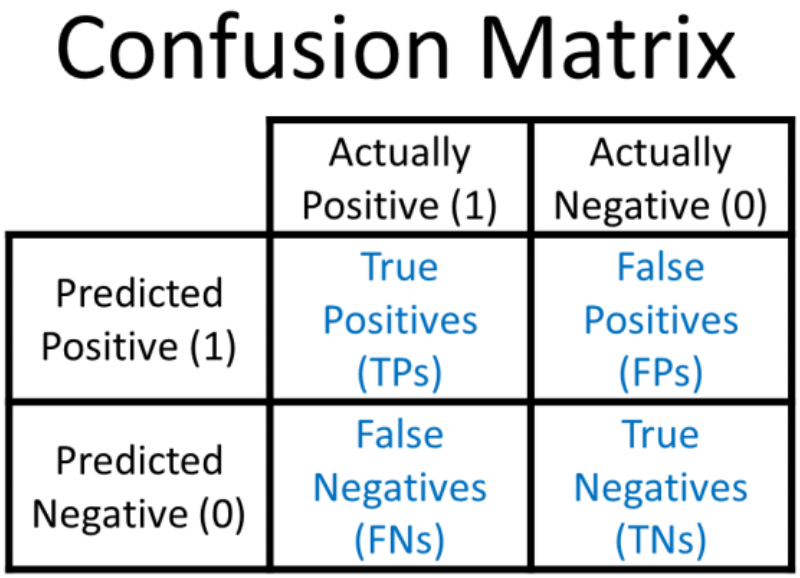


Figure 1 Confusion Matrix [14]

There are 4 important terms of Figure 1 [14]:

- **True Positives**: The cases in which we predicted YES and the actual output was also YES.
- **True Negatives**: The cases in which we predicted NO and the actual output was NO.
- **False Positives**: The cases in which we predicted YES and the actual output was NO.
- **False Negatives**: The cases in which we predicted NO and the actual output was YES.

Accuracy for the matrix can be calculated by taking average of the values lying across the **"main diagonal"** [14]:

$$Precision = \frac{TruePositive + TrueNegative}{TotalSample} \quad (2)$$

1.7.2 F1 Score

F1 Score is the Harmonic Mean of accuracy and recall. The range of F1 Score is [0, 1]. It indicates a classifier's precision (the number of cases properly classified) and robustness.

High accuracy but poor recall produces incredibly precise results, but it also misses a huge number of occurrences that are difficult to identify. Our model's performance improves as the F1 Score increases. It may be mathematically represented as follows [15]:

$$Precision = \frac{TruePositive}{TruePositive + FalsePositive} \quad (1)$$

Precision: It is the number of correct positive results divided by the number of positive results predicted by the classifier [15].

Recall: It is the number of correct positive results divided by the number of all relevant samples (all samples that should have been identified as positive).

The F1 score outperforms basic accuracy in situations where an unbalanced dataset may distort the accuracy statistic. The F1 score makes up for this by taking into consideration both accuracy and recall, giving a more complete picture of the model's performance. However, this measurement is appropriate for binary classification solutions rather than sophisticated classification and segmentation [15].

1.7.3 AUC ROC Curve

The AUC ROC curve is a performance indicator in machine learning that assesses the effectiveness of a binary classification model. Here's an explanation of what it stands for:

Receiver Operating Characteristics (ROC): This is a graphical display that depicts a binary classifier system's diagnostic capacity when the discrimination threshold is changed. It compares the True Positive Rate (TPR) against the False Positive Rate (FPR) at different thresholds [16].

AUC (region Under the ROC Curve): This metric captures the complete two-dimensional region beneath the ROC curve. It calculates an aggregate measure of performance across all categorization criteria. The AUC value is between zero and one. A model with flawless predictions has an AUC of 1, whereas a model with all wrong predictions has an AUC of zero. A higher AUC value implies that the model performs better in differentiating between positive and negative classifications [16].

The ROC curve is especially valuable since it is independent of the classification threshold and provides an indication of how effectively the model separates the two classes as in the Figure 2. The AUC provides a single numerical assessment of the model's performance by

taking into account all conceivable thresholds. It is especially effective when dealing with unbalanced datasets in which one class far outnumbers the other [16].

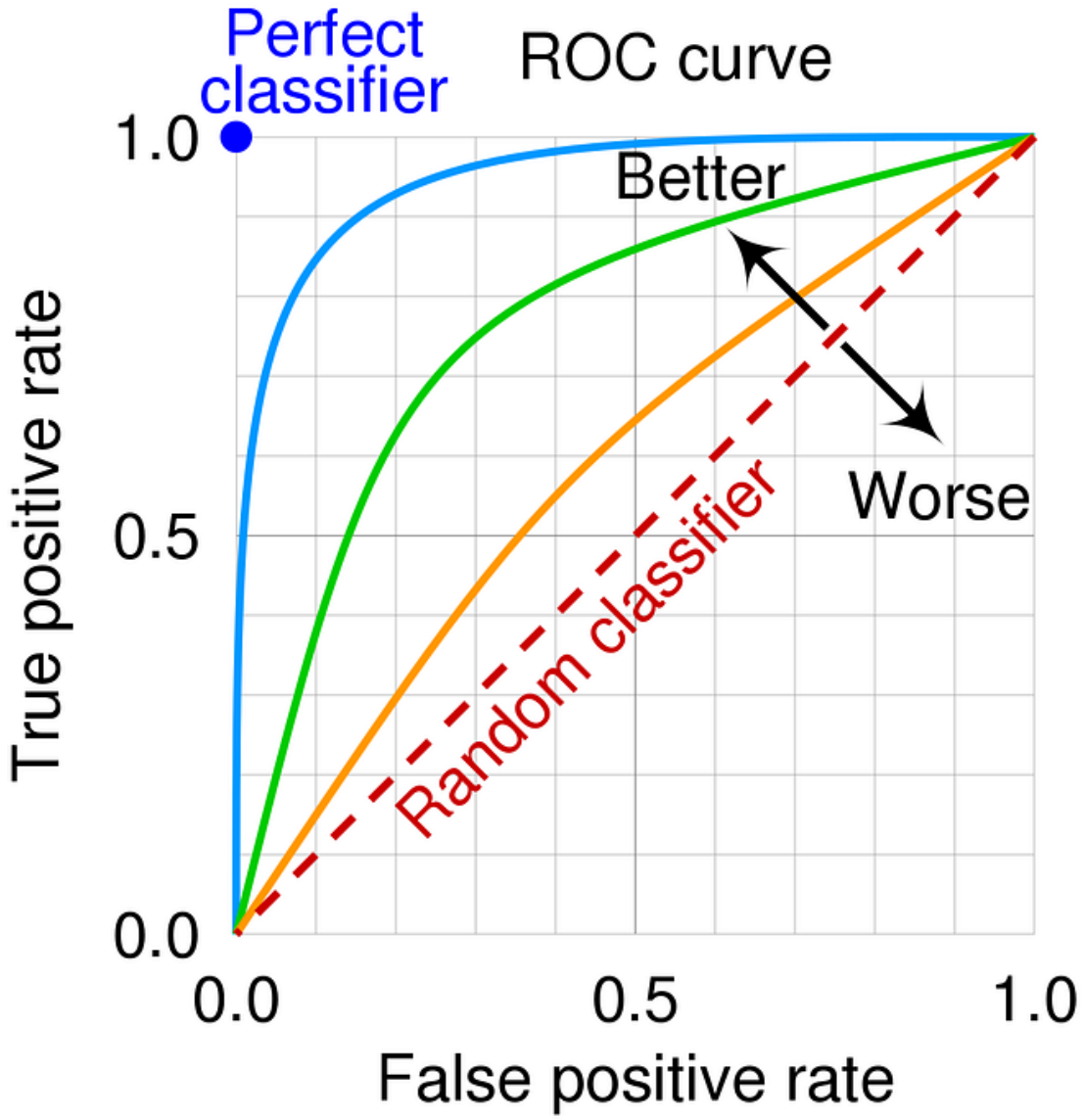


Figure 2 Perfection classifier based on the curve [17]

A machine learning classification model may be used to predict the data point's actual class or the chance of belonging to distinct classes, with an AUC-ROC curve used for assessment. The latter provides us greater influence over the outcome. We may choose our own threshold for interpreting the classifier's results, which is useful when examining the intricacies of the ROC curve. This method is sometimes more wise than developing a whole new model [18].

Changing the criteria for categorizing positive data points will accidentally modify the model's sensitivity and specificity. Depending on whether we want to reduce the amount of

False Negatives or False Positives, one of these criteria will most likely perform better than the others [18].

In an AUC-ROC curve, a higher X-axis value suggests a greater number of false positives than true negatives. A higher Y-axis value suggests a greater number of true positives than false negatives. So, the threshold is determined by the capacity to naturally balance false positives and false negatives [18].

2 NEURAL NETWORKS

Neural networks are conceptualized after the human brain to process information. While they draw inspiration from biological processes, they don't precisely mimic the actual workings of the brain. There are many varieties of neural networks, some are listed as below [13]:

Artificial Neural Networks (ANNs), which are effective for tackling intricate challenges [13].

Convolutional Neural Networks (CNNs), which excel in addressing problems related to computer vision [13].

Recurrent Neural Networks (RNNs), which are adept at handling tasks in the realm of natural language processing [13].

2.1 Fundamental of Neural Networks

Neural networks are a branch of machine learning algorithms. They have been used widely in data mining for various subjects. Before we explore the neural networks, machine learning is revived to express the certainty of this field in order to give a fundamental understanding of artificial intelligence concept [4].

2.1.1 Neural network's structure

A typical neural network consists of three main types of layers: the input layer, one or more hidden layers, and the output layer. Each layer contains nodes, also known as neurons or units. The connections between nodes are associated with weights, which are adjusted during the training process to enable the network to learn from data. Here's a general structure of a feedforward neural network, which is one of the most common types [13]:

- Input Layer:

The input layer receives the features or input values of the dataset. Each node in this layer represents a feature, and the number of nodes corresponds to the number of input features [13].

- Hidden Layers:

Between the input and output layers, there can be one or more hidden layers. These layers help the neural network learn complex patterns and representations in the data [13].

Each node in a hidden layer is connected to every node in the previous and next layers, and each connection has an associated weight [13].

- Output Layer:

The output layer produces the final predictions or classifications. The number of nodes in this layer depends on the nature of the task (e.g., binary classification, multi-class classification, regression) [13].

The output layer's activation function is often chosen based on the task, such as sigmoid for binary classification or softmax for multi-class classification [19].

- Connections, Weights, and bias:

Each connection between nodes has a weight associated with it. During training, these weights are adjusted to minimize the difference between the predicted output and the true target values [19].

The weighted sum of inputs to a node, along with a bias term, is passed through an activation function to determine the node's output [19].

Bias nodes in neural networks serve a similar purpose to the intercept in linear regression, represented as ($y = ax + b$), where “a” is the slope and “b” is the intercept. The primary role of a bias is to add a trainable constant value to the node, aside from the standard inputs it receives. Crucially, the bias allows for shifting the activation function left or right, which is a key factor in the successful training of Artificial Neural Networks [19].

- Activation Function:

Each node, or neuron, typically has an activation function that introduces non-linearity to the model in the Figure 3. Common activation functions include sigmoid, hyperbolic tangent (tanh), and rectified linear unit (ReLU) [19].

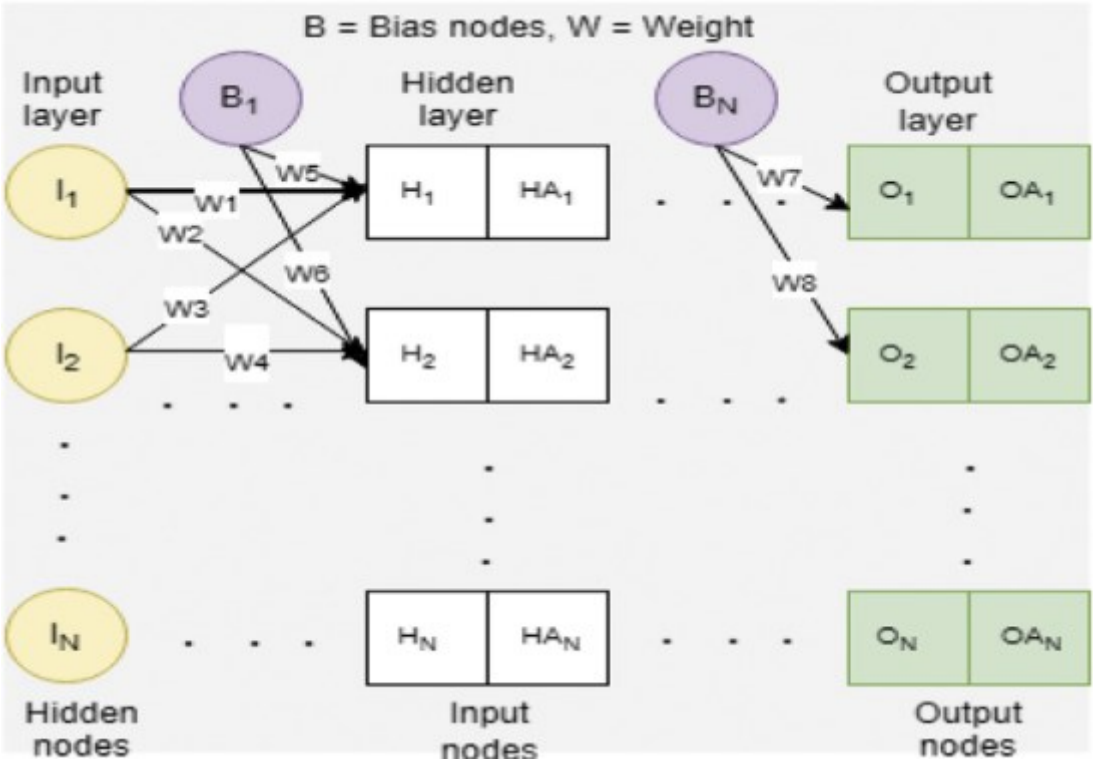


Figure 3 A structure of a neural network [19]

2.2 Layers and Activation Functions

The net inputs of a neural network are the most important components of its architecture. They are processed and transformed into an output result known as the activation of the unit using a function known as the activation function, threshold function, or transfer function—a scalar to scalar transformation [20].

Squashing functions allow a neuron to output in a limited range and at a constrained amplitude. A squashing function limits the amplitude of the output signal [20].

2.2.1 Layers

The convolutional layer is the most commonly utilized, if not the sole, layer in image-related models. Convolutional neural networks have the capacity to identify and extract features independent of their position in provided pictures; this is a required attribute in urban and road scenes with stochastic components such as automobiles, pedestrians, and pavement [20].

ConvLayer

The Convolutional Layer is the fundamental layer of Convolutional Neural Networks and may be thought of as the neural networks' 'eyes'. A convolutional layer's neurons look for specific traits. The input to a convolutional layer is simply a two-dimensional array that might represent the network's input image or the output of a previous layer as shown in the Figure 4. The first convolutional layer receives data from the input image. Typically, there are two types of input pictures: single-channel grayscale images and three-channel colour images [20].

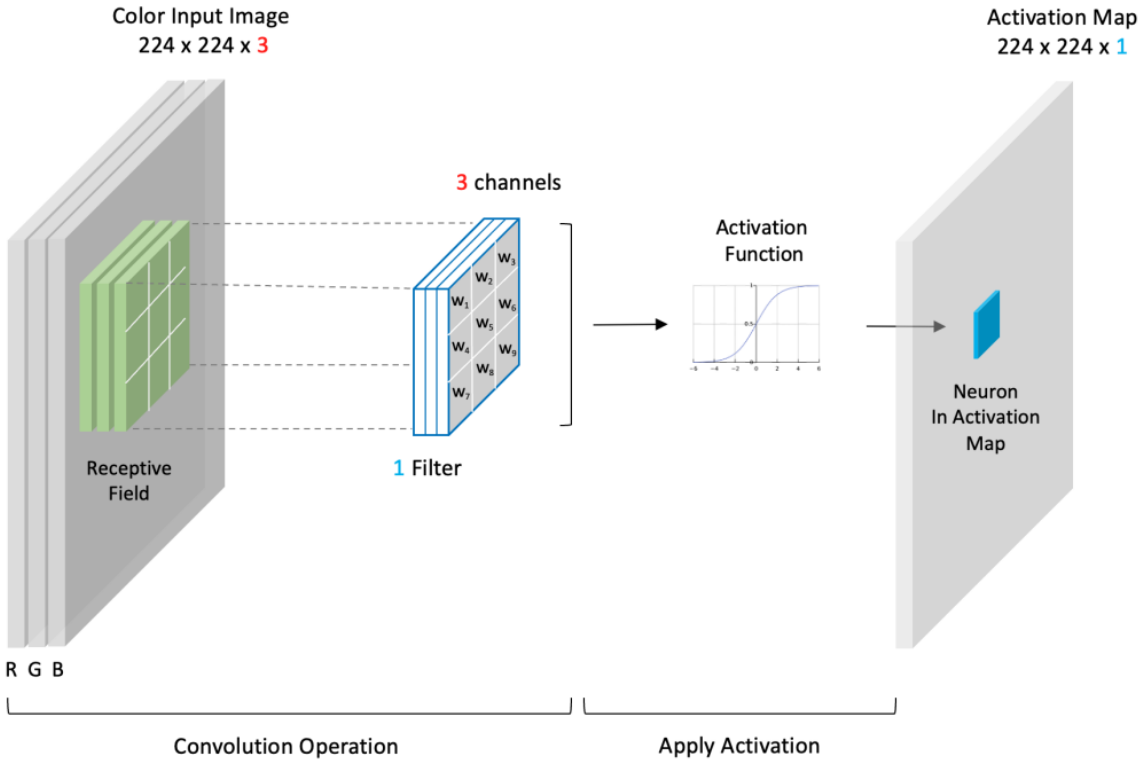


Figure 4 A Convolutional layer setup with three-channel colour images [20]

In a CNN, activation functions are typically applied after each convolutional layer and fully connected layer. However, they are not applied after pooling layers [20].

The math behind is basically described as following Figure 5:

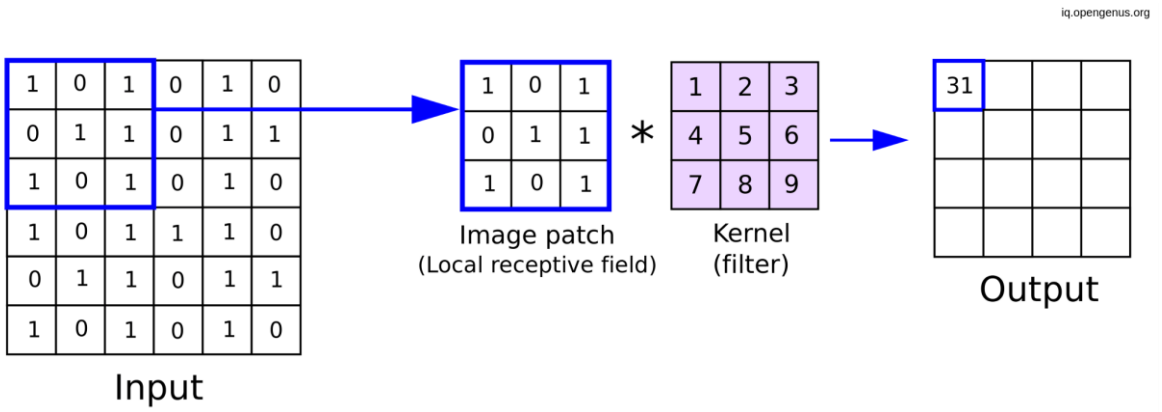


Figure 5 Convolutional layer operation [24]

Consider a set of K learnable filters, each with dimensions and channels distinct from the input, and an input image with dimensions H x W x C, such as an RGB image with three channels. During 2D convolution, each filter glides across the input, multiplying components with pixels that match. The findings are then combined to get a single output value for each point. Similar to matrix multiplication, this operation distributes weights throughout the filter, providing the impression of a fully connected layer. Notably, convolutional layers do not make a direct link between the input and output pixels. The convolution technique is repeated K times, once for each filter, to produce the output feature map. The resulting feature maps are then layered along the depth axis. The size of the feature map produced by the process is determined by the input dimensions, filter size, stride, and padding [24].

Pooling layer - MaxPooling

The output of the max-pooling layer is given by the maximum activation over non-overlapping rectangular regions of size (K_x,K_y). Max-pooling creates position invariance over larger local regions and down-samples the input image by a factor of K_x and K_y along each direction. Max-pooling leads to faster convergence rate by selecting superior invariant features which improves generalization performance [21].

As shown in Figure 6, the algorithm will choose the biggest number in the regions of size (K_x,K_y) and put it in the output matrix.

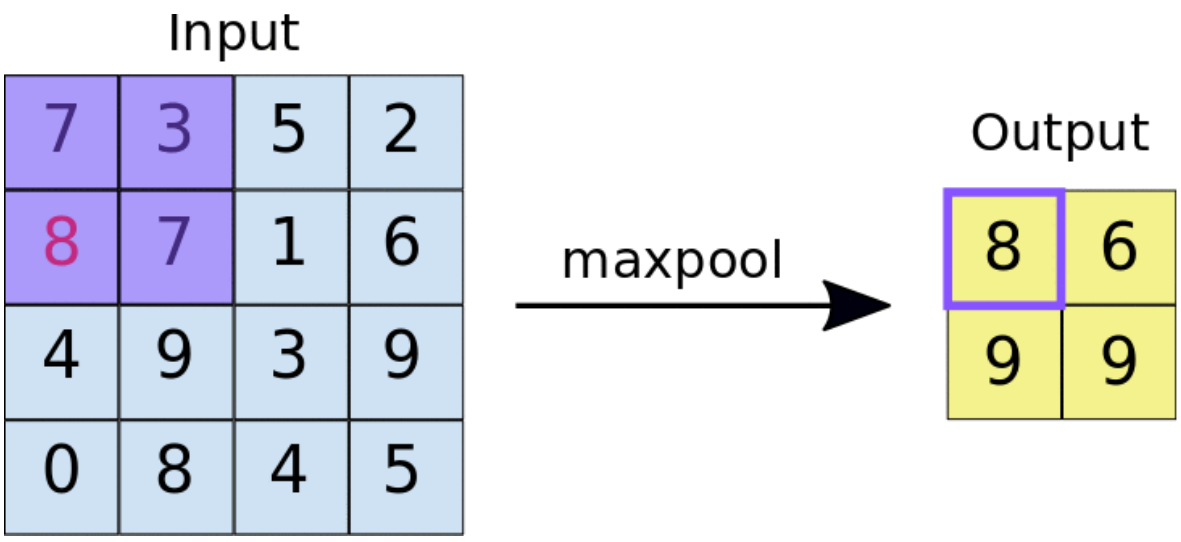


Figure 6 MaxPooling operation [22]

Dropout

Dropout is a regularization method used in neural networks, specifically deep learning models, to minimize overfitting. During training, a subset of neurons in a layer are deactivated at random. This mechanism reduces the dependency of neurons and drives the network to learn more robust properties [23].

1. **Regularization**: Dropout is a regularization strategy that prevents overfitting by minimizing the complicated co-adaptations of neurons [23].
2. **Improved Generalization**: Dropout allows the model to acquire more generalizable properties by preventing it from being overly reliant on specific neurons [23].
3. **Reduction of Model Complexity**: Dropout can simplify the model by lowering the effective network size, resulting in quicker training and lower computing costs [23].

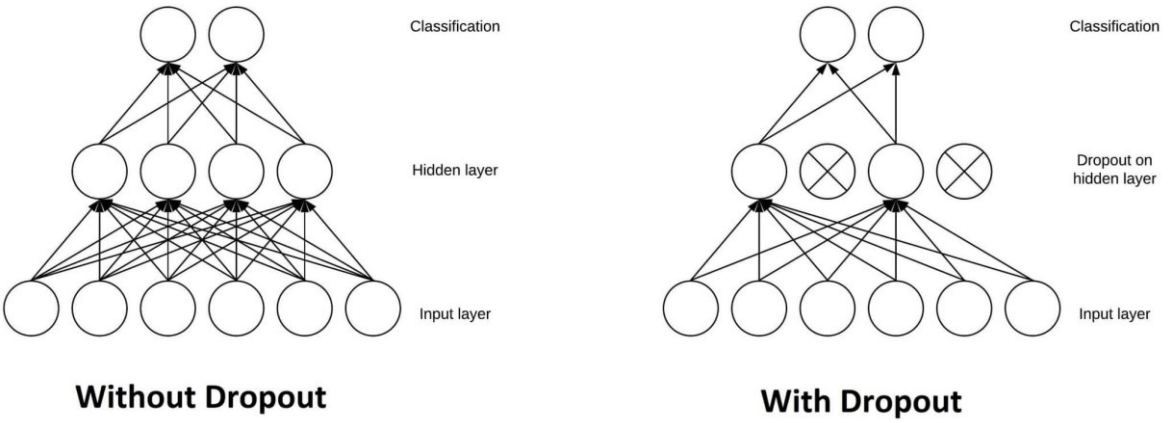


Figure 7 Dropout during training [23]

During training as in Figure 7, dropout randomly sets a fraction of the input units to zero with a probability pp, typically chosen between 0.2 and 0.5. This means that the output of these units is temporarily ignored during the forward and backward passes of training. Dropout is only applied during training, not during inference or evaluation [23].

2.2.2 Activation functions

The activation function defines an artificial neuron's activity range. This is applied to the neuron's entire weighted input data. An activation function is nonlinear. If an activation function is not employed, the only operations involved in calculating the output of a multilayer perceptron are the linear products between the weights and the input values [25].

One linear operation may be conceived of as a collection of successive linear operations. Using a non-linear activation function, on the other hand, leads the artificial neural network to become nonlinear, causing the function that approximates the neural network to become nonlinear as well. According to the approximation theorem, a universal function approximator is a multilayer perceptron that has one hidden layer and a nonlinear activation function [25].

Sigmoid

The sigmoid function is one of the most often used activation functions. It is commonly understood that there are two stages to modeling and training a multi-layer neural network: forward propagation and reverse propagation. In addition, during backpropagation, the derivatives of the activation functions must be computed in each layer. Because the sigmoid function is continuous, it may be differentiated anywhere [25].

$$g(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

In neural networks, the sigmoid function serves as an activation function. To refresh your memory on what an activation function is, consider the function that an activation function plays in one layer of a neural network in the image below. An activation function is applied to a weighted sum of inputs, and the resultant output is used as an input for the subsequent layer [25].

It is guaranteed that the output of a neuron will always be between 0 and 1 as in the Figure 8 when the neuron's activation function is a sigmoid function. The output of this unit would also be a non-linear function of the weighted sum of inputs because the sigmoid is a non-linear function. A sigmoid unit is a kind of neuron that uses the sigmoid function as an activation function [25].

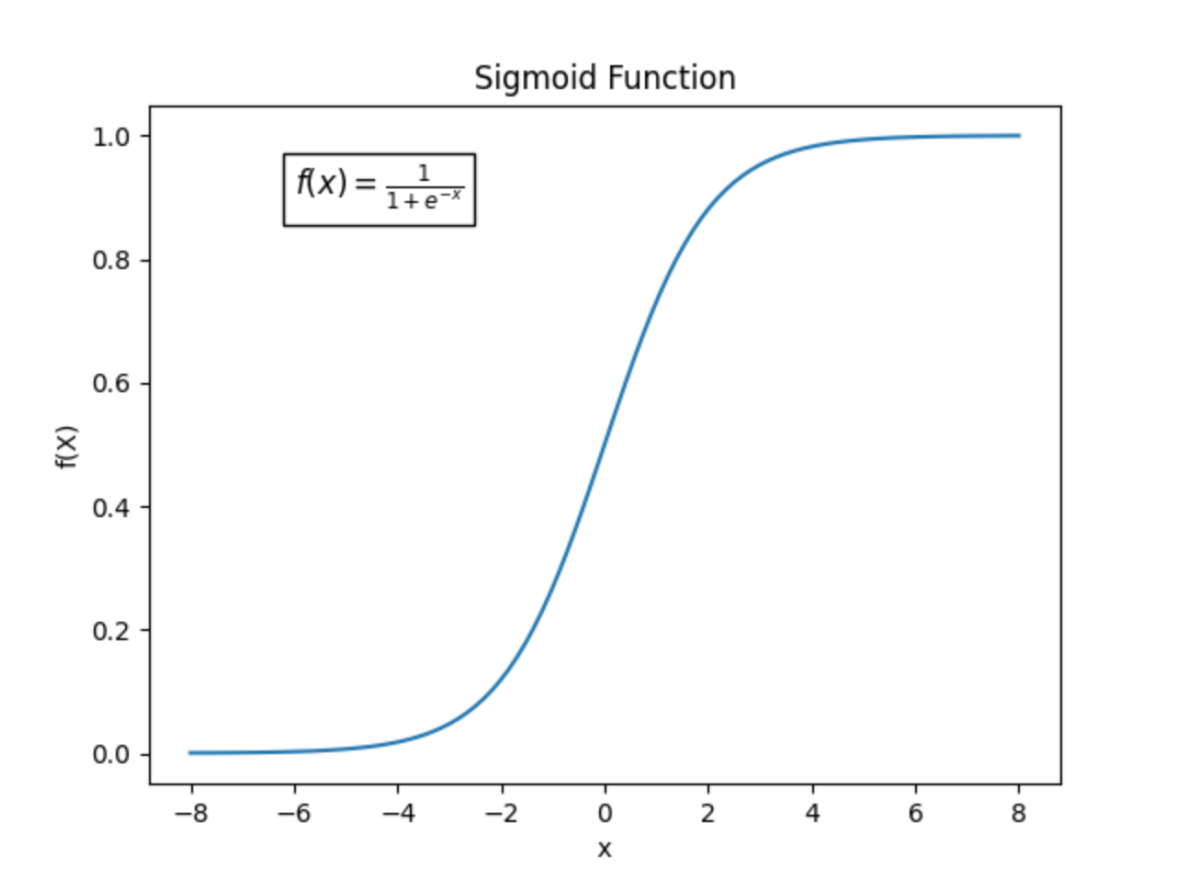


Figure 8 Sigmoid function [25]

ReLU

In the realm of Convolutional Neural Network, the Rectified Linear Unit (ReLU) is a cornerstone that gives a simple solution to some of the industry's most persistent problems. ReLU is just a simple but strong activation function that returns the input when it is positive and zero otherwise. Its simplicity belies its potency, as it provides critical nonlinearity into neural network design, fundamentally altering neural networks. ReLU paves the way for more accurate and nuanced learning by allowing the network to detect more complex linkages within data [25].

$$f(x) = \max(0, x) \tag{3}$$

Saturated functions like sigmoid and tanh pose complications with back propagation. As the neural network construction progresses, the gradient signal begins to disappear, a phenomenon known as the "vanishing gradient". This occurs because the gradient of such functions is usually always close to zero, except in the center. However, the ReLU has a constant gradient for positive input. Although the function is not differentiable, it can be disregarded during implementation [25].

The ReLU generates a sparse representation in the Figure 9. Because the zero in the gradient results in a full zero. However, sigmoid and tanh always provide non-zero outcomes from the gradient, which may not be advantageous for training [25].

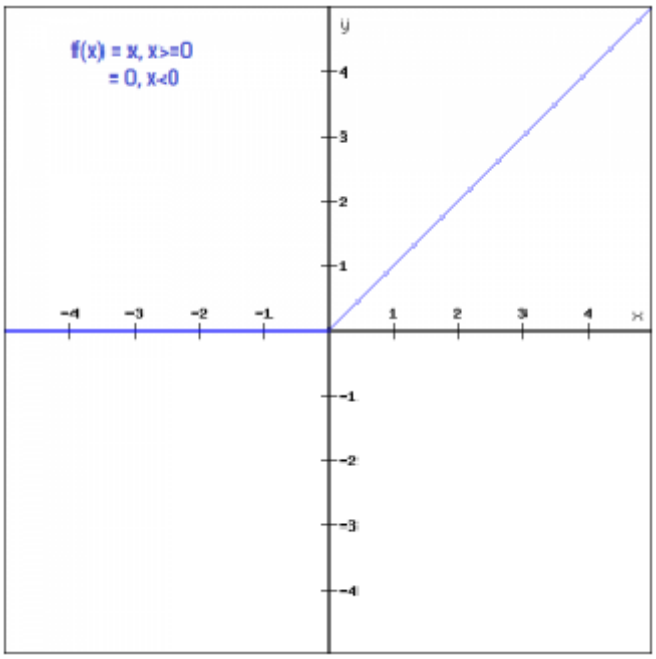


Figure 9 Graph of ReLU

One of ReLU's unique qualities is its ability to effectively handle the vanishing gradient problem, which has long been a hurdle in neural network training. ReLU solves this problem, allowing for quicker learning and better performance than its predecessors, such as the sigmoid and hyperbolic tangent functions. This feature not only reduces training time, but also enhances the network's capacity to learn complex patterns from data. Furthermore, ReLU's fundamental nature makes it easier to train models, speeding up development and usually generating superior results in a wide range of applications [25].

Many neural network topologies, including CNNs and multilayer perceptrons, use ReLU as their default activation function. Its broad use arises from its ability to permit sparse activations in networks and expedite training procedures. This widespread adoption emphasizes the importance of ReLU in modern deep learning, as it serves as the foundation for a wide range of models and applications. ReLU remains a trusty companion as the field evolves, allowing neural networks to manage the complexities of real-world data with unparalleled efficacy and efficiency [25].

LeakyReLU

Leaky ReLU is an improvised version of ReLU function where for negative values of x, instead of defining the ReLU functions' value as zero in the Figure 10, it is defined as extremely small linear component of x. It can be expressed mathematically as [26]:

$$f(x) = \begin{cases} 0.01x, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{4}$$

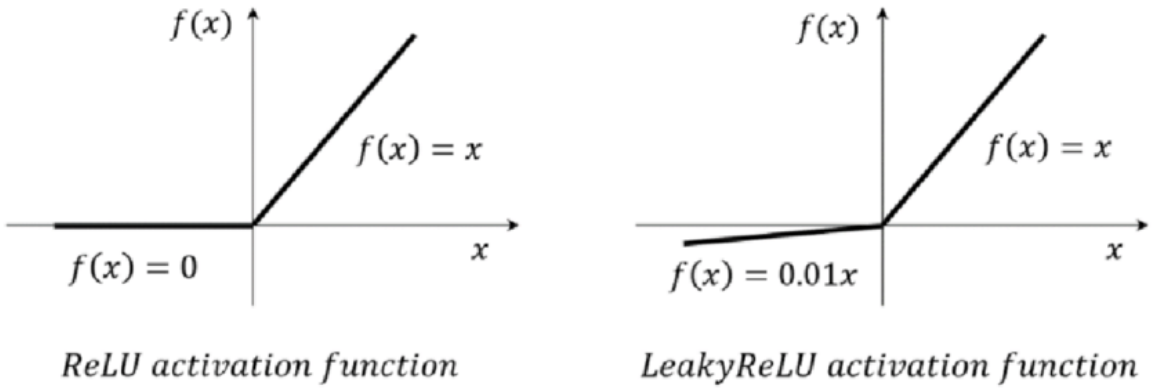


Figure 10 LeakyReLU compared to ReLU [26]

BatchNormalization

Batch normalization (BN) is a technique to normalize activations in intermediate layers of deep neural networks. Its tendency to improve accuracy and speed up training have established BN as a favourite technique in deep learning. BN is an indispensable component in many deep neural networks. BN has been widely used in various areas such as machine vision, speech and natural language processing [27].

When feeding data into a deep learning model, it is common practice to normalize it to zero mean and unit variance. Assume the input data has several features (x1, x2, …xn) , as shown in the Figure 11. Each characteristic may have a separate range of values. For example, values for feature x1 may vary from 1 to 5, but values for feature x2 may range from 1000 to 99999. We compute the mean and variance for each feature column independently, using the values from all samples in the dataset. Next, use the formula in Figure 11 to normalize the data [27].

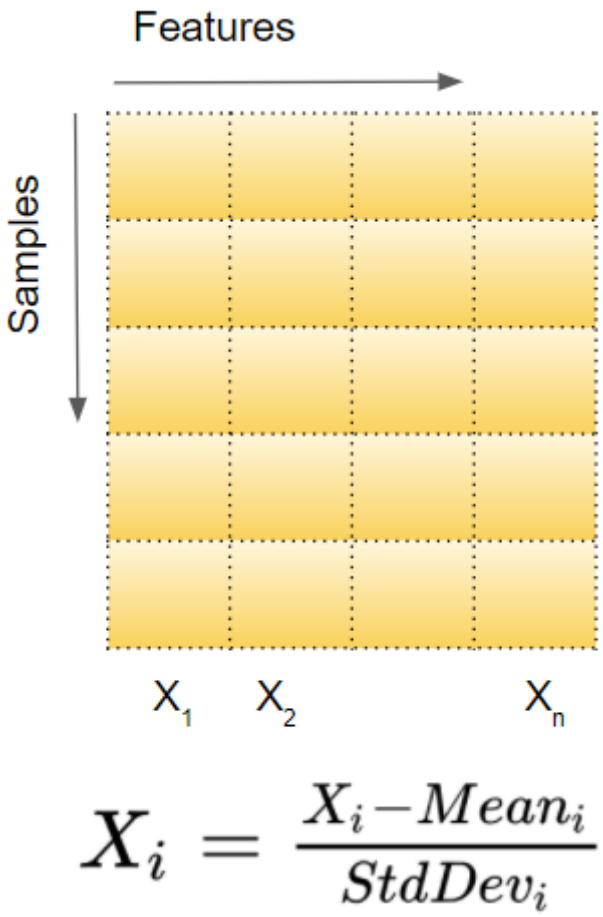


Figure 11 How a batch is normalized [27]

Normalizing data has an effect. The original values (blue) are now centered around zero (red). This guarantees that all feature values are now on a consistent scale as shown in Figure 12 [27].

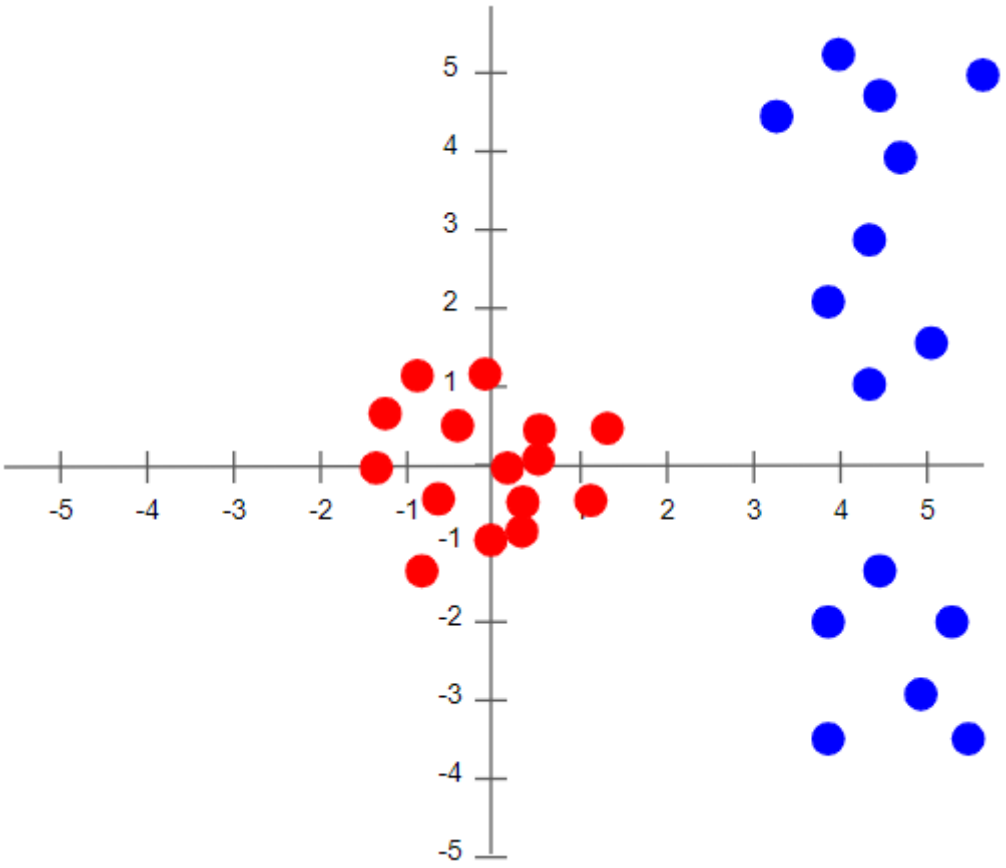


Figure 12 How batch normalization looks like [27]

Softmax

Softmax is a mathematical function that converts a vector of real numbers into a probability distribution. It's commonly used as the activation function in the output layer of a neural network when the task involves multi-class classification. Softmax ensures that the output probabilities sum up to 1, making it suitable for modeling the probability distribution over multiple classes [29].

Given an input vector $z = [z1, z2, \ldots, zn]$, the softmax function computes the probability distribution $\sigma(z) = [p1, p2, \ldots, pn]$ as follows [29]:

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{4}$$

where:

- e is the base of the natural logarithm (Euler's number).
- z_i is the i^{th} element of the input vector.
- $\sum_{j=1}^{n} e^{z_j}$ is the sum of the exponentiated elements of the input vector.

The softmax function effectively "squashes" the input values into the range (0, 1) and ensures that the resulting values sum up to 1, thus representing a probability distribution as Figure 13 [29].

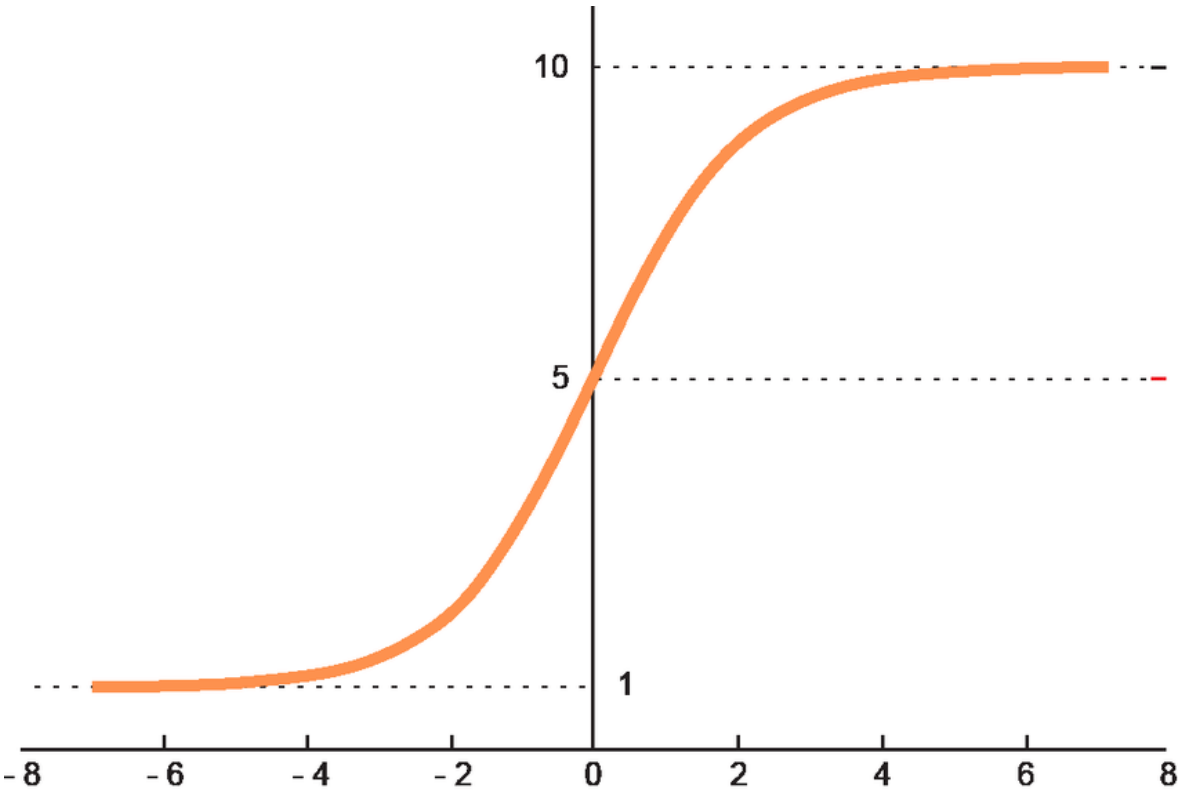


Figure 13 Softmax graph [28]

2.3 Pre-trained models and transfer learning

Transfer learning is a potent machine learning technique that makes use of insights from the resolution of one problem to improve performance on a related but different task. Transfer learning enables the model to profit from prior information acquired during training on a different source task, as opposed to starting from scratch when training for a particular target job. This method is especially useful when obtaining labelled data for the target job is costly or scarce. Transfer learning has shown to be a useful tactic in a variety of fields, allowing models to perform better on tasks with little data available, generalize more effectively, and

achieve faster convergence by transferring acquired representations, patterns, and insights [30].

2.3.1 Pre-trained models

Pre-trained models are neural network designs that have been trained on large datasets, usually for image classification, object detection, natural language processing, or other machine learning applications. These models are trained on massive volumes of data to produce useful representations of the original data. They initially assign a set of weights and bias which can be fine-tuned for a specific task. PyTorch gives an option to load the pre-trained weights to the models from their libraries. The following section will describe the pre-trained models utilised and their attributes in relation to the datasets [31].

2.3.2 Transfer learning

Transfer learning is a machine learning technique in which a model learned on one job is modified or fine-tuned to perform a different but related task. Transfer learning is the process by which knowledge obtained from addressing one problem is transferred and applied to another, but usually similar problem. This is especially advantageous when the task at hand has limited labelled data, because the pre-trained model can apply knowledge gained from a bigger dataset to boost performance on the new task [32].

The following steps are usually involved in the process:

Pre-training: Use a sizable dataset to train a CNN model for a task such as image categorization. The model is able to extract broad characteristics and trends from the data through this method [32].

Transfer Learning: Adapt the previously trained CNN to the semantic segmentation task by making changes to it. This usually entails changing the network's fully linked layers or replacing them to meet segmentation's output needs [32].

Fine-tuning: You can choose to use the new dataset with labeled segmentation masks to test the adjusted model. The model can now modify its learned features to better suit the new dataset thanks to this phase [32].

2.4 Segmentation

Image segmentation is a method of segregating different objects inside an image into different regions. Convolutional neural networks are a part of deep learning models that shows outstanding outcomes of different computer vision tasks including segmentation [33].

This chapter will introduce about Convolutional neural networks and image segmentation, and how CNNs are used to train in image segmentation. Next transfer learning method will be presented to express the efficiency of the method in image segmentation.

2.4.1 Convolutional Neural Networks (CNNs)

Like any standard neural network model, Convolutional Neural Networks (CNNs) follow a structure where neurons are arranged in layers, enabling the learning of hierarchical representations. Connections between neurons across layers involve weights and biases. The first layer serves as the input layer, such as remote sensing data, while the final layer produces the output, like a classification prediction for plant species. Hidden layers in between modify the feature space of the input to align with the desired output. CNNs specifically incorporate a convolutional layer among the hidden layers to capture patterns, particularly spatial patterns in the context of this review which can be seen in Figure 14 [34].

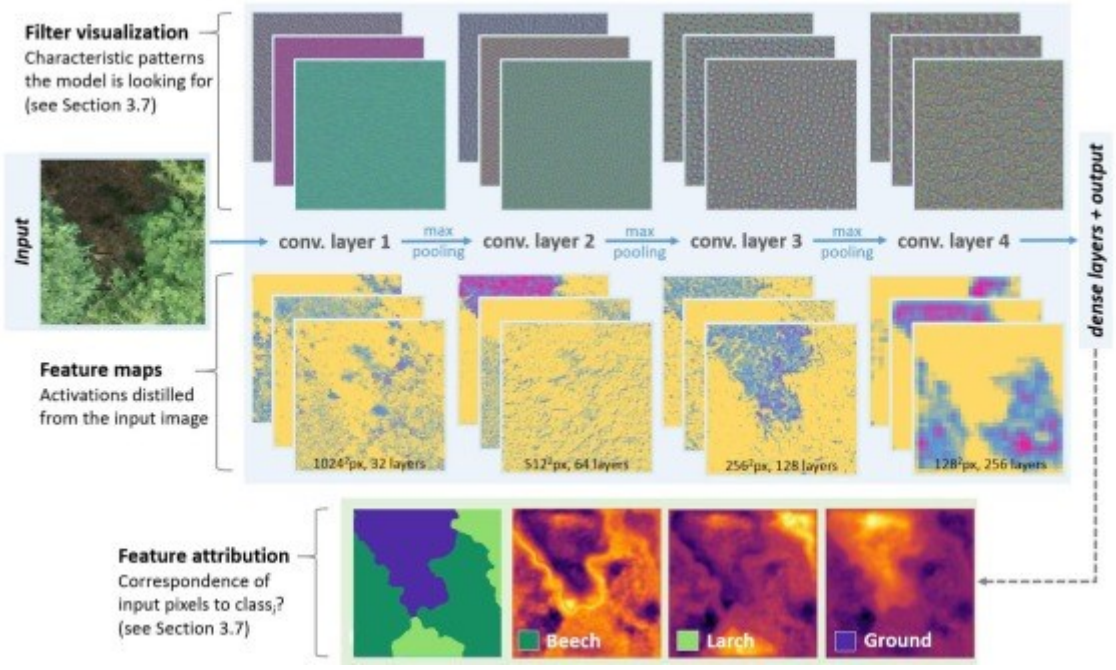


Figure 14 Typical CNN structure [34]

2.5 Semantic segmentation models

Semantic segmentation models play a vital role in computer vision field. They take care of pixel-level analysis precisely. There are many famous semantic segmentation models to increase the accuracy and the comprehensiveness towards the datasets.

2.5.1 SegNet

The architecture of SegNet is encoder-decoder based. Its purpose is to take an image as input and generate a pixel-by-pixel label map. Through the use of many convolutional and pooling layers, the encoder is able to extract high-level information. SegNet's decoder network, which upsamples and creates a segmentation map using the spatial pooling indices created during max-pooling in the encoder phase, is the main innovation in the system. This type of upsampling helps to maintain the output's fine-grained features while still being memory-efficient [35].

2.5.2 Fully Convolutional Network

Fully Convolutional Network is referred to as FCN. This kind of neural network design is mostly applied to computer vision applications involving semantic segmentation. In order to divide objects or areas of interest, semantic segmentation entails dividing a picture into many segments and giving a class name to each pixel [37].

The term "fully convolutional" refers to FCNs as in Figure 15 since they only include convolutional layers and no fully linked layers. Because of this, they can process input of any size and generate output with comparable spatial dimensions, which makes them appropriate for jobs where the output must be spatially matched with the input, such as semantic segmentation [37].

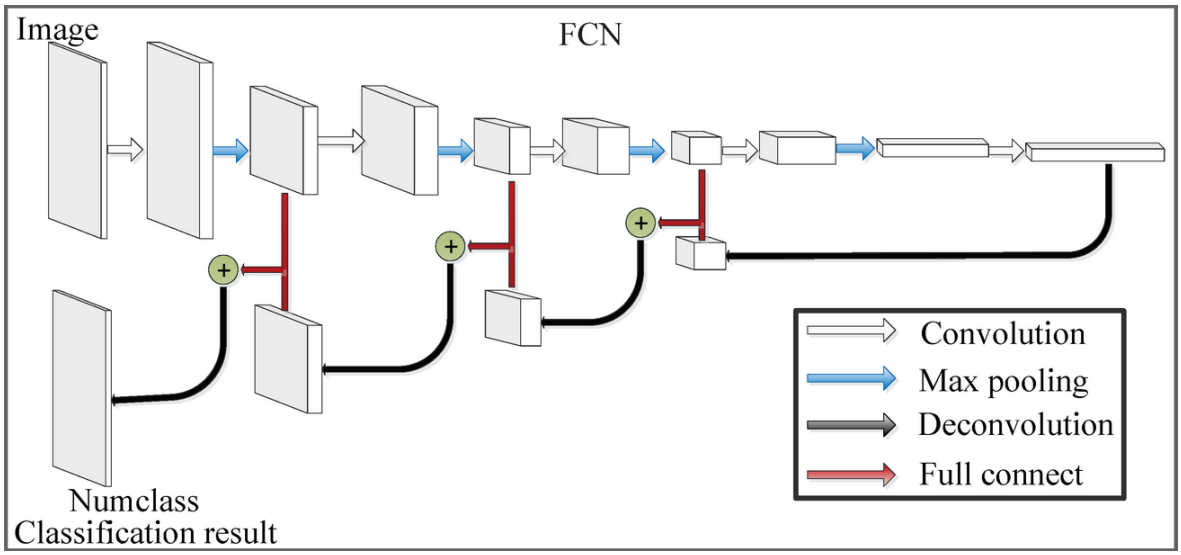


Figure 15 FCN basic structure [37]

Convolutional layers are created by FCN from fully linked layers, allowing for an effective classification net for end-to-end dense learning. By this means, they can enable precise classification as well as segmentation for images [37].

2.5.3 U-Net

Olaf Ronneberger et al. created the U-Net, a convolutional network architecture, for biomedical image segmentation at the University of Freiburg in Germany in 2015. It allows for accurate and speedy image segmentation [38].

With four encoder blocks and four decoder blocks joined by a bridge, U-Net is a U-shaped encoder-decoder network design in Figure 16. At each encoder block, the number of filters (feature channels) is doubled and the spatial dimensions are cut in half by the encoder network (tracing path). Similarly, the number of feature channels is cut in half and the spatial dimensions are doubled by the decoder network [38].

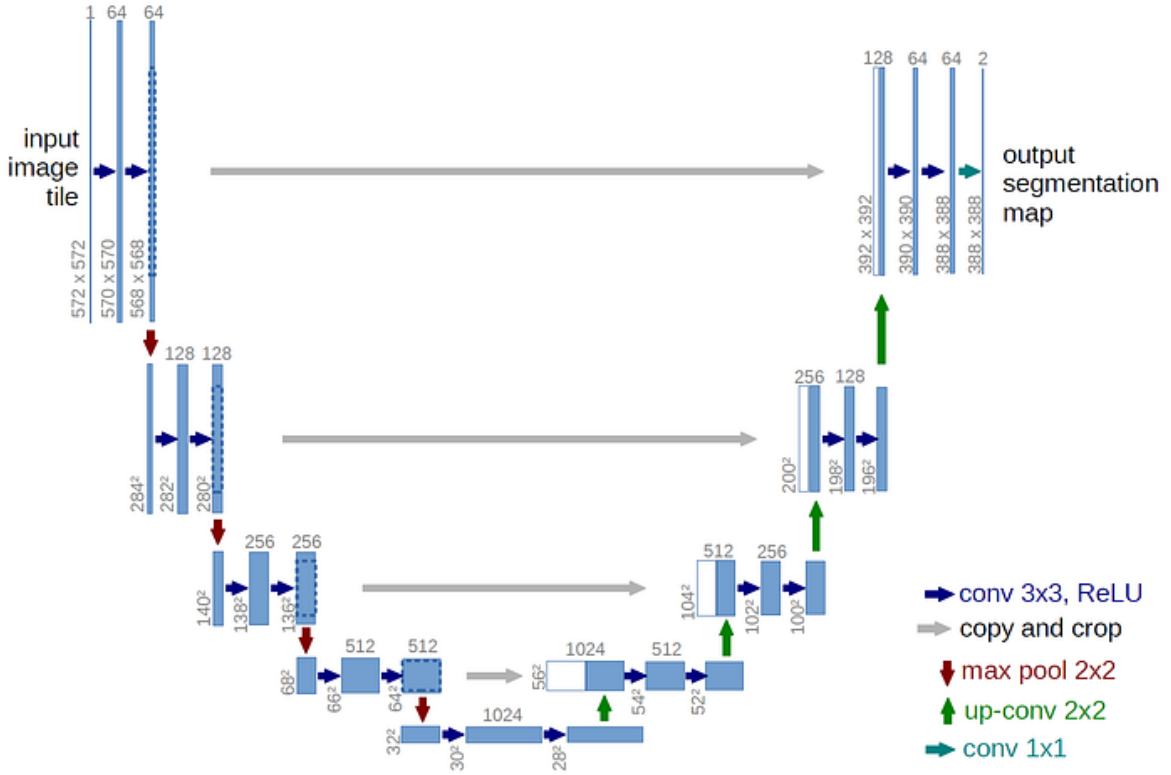


Figure 16 U-Net architecture [38]

2.6 VGG16

VGG16 is developed by Visual Geometry Group, is a type of CNN (Convolutional Neural Network) that is considered to be one of the best computer vision models to date. The creators of this model evaluated the networks and increased the depth using an architecture with very small (3 × 3) convolution filters, which showed a significant improvement on the prior-art configurations. They pushed the depth to 16–19 weight layers making it approximate — 138 trainable parameters [39].

VGG16 is object detection and classification algorithm which is able to classify 1000 images of 1000 different categories with 92.7% accuracy. It is one of the popular algorithms for image classification and is easy to use with transfer learning. By this means, this model can be classified among the classes in the road segmentation solution. It observes and distinguishes classes and outputs large quantities of parameters for segmentation in U-Net variants [39].

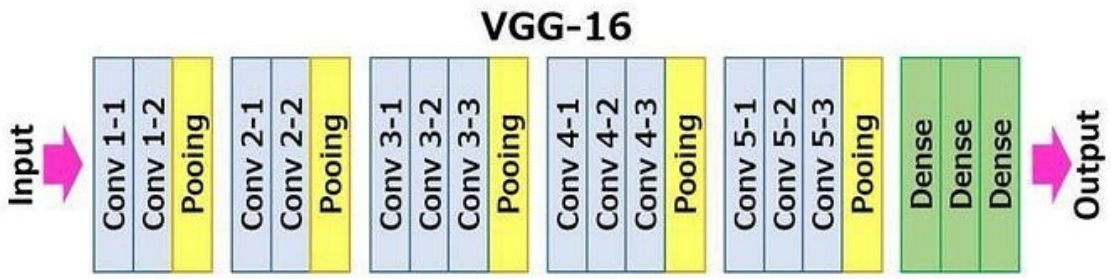


Figure 17 Example architecture of VGG-16 [32]

The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv〈receptive field size〉 - 〈number of channels〉". Configurations are following with the patterns below [39]:

Table 1 Parameter are following these, with, A, A-LRN, B, C, D, and E configurations [39]

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
ma pool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
ma pool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
ma pool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
ma pool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

To follow the table, the size of a layer is stated after the layer name. For example, if the configuration A is used, Conv3 means the output of the Layer, block3_pool (MaxPooling2D) and its size of the output is 64. The next layer is max pool layer, then comes next with conv3-128 and so on. In the output layer, it will be connected to Fully Connected layer with size 4096 (FC-4096) and finally with softmax layer. Based on which configuration, the more parameters it will generate, the larger the model it is [39].

3 MACHINE LEARNING TOOLS AND LIBRARIES

There are several popular machine learning libraries and tools that are widely used in the field which can be told as Tensorflow, Keras, PyTorch, or OpenCV. Every library has its own advantages such as PyTorch has its simplicity, ease of use, flexibility, efficient memory usage, and dynamic computational graphs, but it cannot display the deep details as Tensorflow. Tensorflow also supports Keras library to keep the explicit of model construction so that it can be tracked down on every layer. Therefore, Tensorflow is chosen for creating and measuring models.

Tensorflow Keras is, briefly described, used for establishing models, training, and testing. It provides detailed insights about layers, shapes, and model architecture.

OpenCV is a library to load and use the image. Because its ease of use, I use this for easy access to an image and load it. OpenCV is used for evaluating trained models and to import and display images, as well as try trained models with videos. By this means, it can mimic a real-life application for road segmentation solutions.

3.1 Tensorflow.Keras

In this project, TensorFlow is chosen to measure, train, evaluate, and construct models for segmentation. TensorFlow Keras is a high-level neural networks API written in Python and integrated into TensorFlow. It serves as TensorFlow's official high-level API for building and training deep learning models. Keras provides a user-friendly interface that allows for easy and fast prototyping of neural networks, making it suitable for both beginners and experts in machine learning [40].

TensorFlow Keras offers several key features and benefits:

- *Modularity:* Keras makes it simple to create complicated structures by enabling users to construct neural networks by stacking modular building pieces (layers) together [40].
- *User-friendly:* Keras has an easy-to-understand interface that is straightforward, making it a great choice for individuals who are new to deep learning [40].
- *Flexibility:* Keras makes it simple to experiment with various network architectures, optimizers, loss functions, and other hyperparameters [40].

- *Compatibility:* TensorFlow Keras works in unison with other TensorFlow components, including TensorFlow Es-timators for distributed training and TensorFlow Datasets for data loading [40].
- *Performance:* Keras leverages the computational power of TensorFlow backend for efficient training and inference, while prioritizing simplicity and ease of use. This results in excellent performance [40].
- *Community and Resources:* As one of the most widely used deep learning libraries, Keras boasts a sizable user and contributor community that offers a wealth of resources, including pre-trained models, documentation, and tutorials [40].

Therefore, TensorFlow Keras is a powerful tool for building and training deep learning models, offering a balance between simplicity and flexibility, and it can show in detail about models and give messages about faults during training [32].

To define a model using Keras, there is many ways to define one, it depends on user coding style.

```
inputs = keras.Input(shape=(37,))
x = keras.layers.Dense(32, activation="relu")(inputs)
outputs = keras.layers.Dense(5, activation="softmax")(x)
model = keras.Model(inputs=inputs, outputs=outputs)
```

The model to define should have inputs as input layer using keras.Input, it will have multiple x variables as hidden layers which is provided by Keras in keras.layers library that will link the input layers and be the output for output layer. Then it will define an output layer. Eventually, the model is declared with inputs and outputs which are the parameters inputs and outputs. A model can be defined as a class with inheritance as keras.Model. To overview the model just created, Keras provides summary method from the keras.Model. With this method, a summary of the model with shapes and parameter sizes is shown.

3.2 OpenCV

In this project, real-time applications for assessing and measuring the performance of the trained models are provided by using OpenCV to analyze photos, as well as input and output images from the models. An open-source software library for computer vision and machine learning is called OpenCV, short for Open Source Computer Vision Library. Originally created by Intel, it is currently maintained by the OpenCV Foundation, a community of developers [41].

The library is renowned for its real-time vision applications and computational efficiency, supporting interfaces in C++, Python, Java, and MATLAB, and is compatible with Windows, Linux, Android, and Mac OS. Therefore, OpenCV is suitable for implementing evaluation application with high performance by using native application like C++. OpenCV's algorithms are designed to be highly optimized, taking advantage of hardware acceleration where available. Its extensive use in various industries, from Google and Microsoft to startups, highlights its versatility in applications like surveillance, robotics, interactive art, and automated inspection [41].

II ANALYSIS

4 DATASETS

The algorithms are benchmarked and evaluated with dataset Cityscapes contains single view images and ground truth images for training. The datasets can be downloaded via their official website Cityscapes [43].

The Datasets contain single view images from an interior camera that it captures urban street scenes. They possess many different sets of datasets. In this project, the datasets originally have 30 classes but simplify into 13 classes. This is to reduce the resources needed to train and validate.

Alongside with the official datasets, paired image datasets are used to simplify the data processing and data structure for training and testing [45].

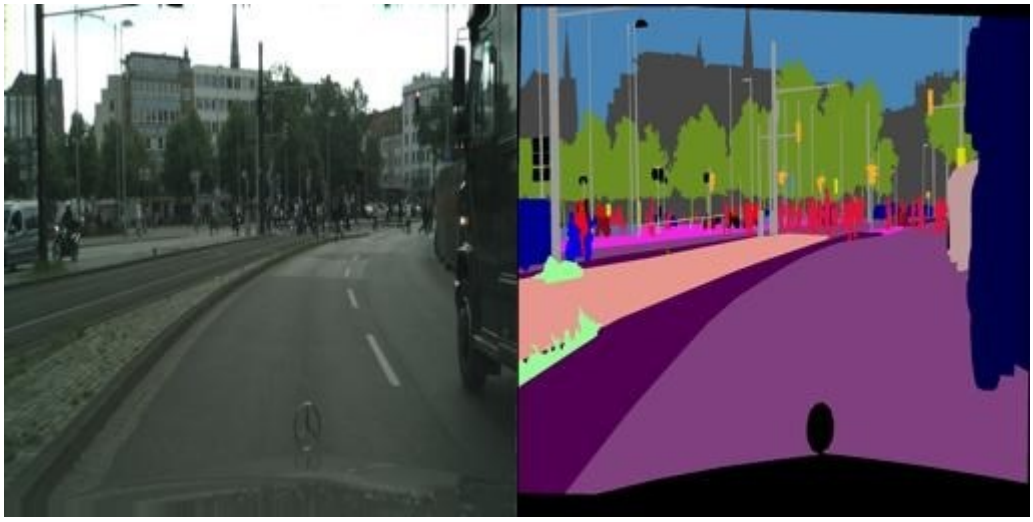


Figure 18 Image display example of the Cityscapes dataset [45]

The classes in the datasets are described in their official website as below:

Table 2 Classes in the datasets [43]

Group	Classes
flat	Road, sidewalk, parking, rail track
human	Person, rider
vehicle	Car, truck, bus, on rails, motorcycle, bicycle, caravan, trailer
construction	Building, wall, fence, guard rail, bridge, tunnel
object	Pole, pole group, traffic sign, traffic light
nature	Vegetation, terrain
sky	sky
void	Ground, dynamic, static

The chosen classes in this project are for obstacles as road, sidewalk, person, car, bus, building, pole, traffic sign, traffic light, vegetation, terrain, and sky; and static as car itself which is displayed as black in the ground truth image. Each class is coloured with an RBG code and to display it for demonstration and prediction.

4.1 Data Generator

Data generator is a function to extract images and ground truth images; and put them separately into variables.

The datasets is delivered without pre-made trainsets and testsets so it has to be done by separating all the images into to set. In this project, they are named as 'train' and 'val' with proportion 80% for trainset and 20% for validation sets.

```
for class_folder in ["train", "mask"]:
    #Get a generator expression for all file names that ends with jpg
    file_names = [f for f in os.listdir(os.path.join(path, class_folder)) if f.endswith(".jpg")]
    #Shuffle the filenames
    file_names = random.sample(list(file_names), len(file_names))

    split_index = int(len(file_names) * (1 - test_size))
    #move the files to train folder
    for file_name in file_names[:split_index]:
        src_path = os.path.join(path, class_folder, file_name)
        dest_path = os.path.join(train_folder, class_folder, file_name)
        os.makedirs(os.path.dirname(dest_path), exist_ok = True)
        shutil.copy2(src_path, dest_path)
        print(file_name)

    #move the files to test folder
    for file_name in file_names[split_index:]:
        src_path = os.path.join(path, class_folder, file_name)
        dest_path = os.path.join(test_folder, class_folder, file_name)
        os.makedirs(os.path.dirname(dest_path), exist_ok = True)
        shutil.copy2(src_path, dest_path)
```

Figure 19 Splitting train sets and test sets

The code snippet can be broken down as follows:

1. **Data Preparation**: The code prepares the dataset for training and testing by iterating through each class folder.
2. **File Selection**: It selects only files with the ".jpg" extension for further processing.
3. **Shuffling**: The file names are randomly shuffled to ensure that the data is not biased during training and testing.
4. **Splitting**: The shuffled file names are split into training and testing sets based on the specified test size ratio.

5. **Moving Files**: Files are moved from their original location to either the train or test folder based on the split index.
6. **Directory Creation**: Directories for the train and test folders are created if they don't already exist.
7. **Copying Files**: Files are copied from the source path to the destination path using shutil.copy2().
8. **Feedback**: Progress is printed to the console indicating which files are being moved to the train and test folders.

After the trainsets and testsets are partitioned and structured in separate folders, the images are loaded to variables for training and testing during fitting the model.

The function DataGenerator gets all the images in both folder, load into a variable, and split them into batches. The data is converted to NumPy array to fit with Keras format.

```
def DataGenerator(path, batch_size=BATCH_SIZE, classes=N_CLASSES):
    files = os.listdir(path)
    while True:
        for i in range(0, len(files), batch_size):
            batch_files = files[i : i+batch_size]
            imgs=[]
            segs=[]
            for file in batch_files:
                image, mask = LoadImage(file, path)
                mask_binned = bin_image(mask)
                labels = getSegmentationArr(mask_binned, classes)

                imgs.append(image)
                segs.append(labels)

            yield np.array(imgs), np.array(segs)
```

Figure 20 DataGenerator code snippet

To access to an image in the output of DataGenerator, it can be queried by using next() function.

```
train_gen = DataGenerator(train_folder, batch_size=BATCH_SIZE)
val_gen = DataGenerator(valid_folder, batch_size=BATCH_SIZE)
```

```
imgs, segs = next(train_gen)
imgs.shape, segs.shape
```

Firstly, DataGenerator has to be defined and assigned to a variable train_set or val_set. To access to an element in train_set or val_set, next() function is used as in above code snippets.

5 TRAINING AND MODELS

The algorithms are verified by models U-Net, FCN, and a use of U-Net structure with pre-trained model VGG16 for classification then combine with U-Net structure.

The U-Net has its specifically designed advantages in image segmentation due to its efficiency and good handling of multi-class tasks. FCN is chosen to measure the performance of the chosen dataset because the model has no restrictions from the full connection layer, so the size of the input can be flexible. U-Net with VGG16 is to improve the disadvantages of both models that the U-Net is more prone to overfitting, especially when working with small datasets and FCN is relatively big and slow even if it is with small datasets [46][47]. VGG16 is the state-of-the-art pre-trained model in segmentation solutions fine-tuned for specific tasks, reducing the need for large annotated datasets. It is famous for its high accuracy and well-trained weights. [38]

5.1 Training

By training, loss function and some parameters are used for training. It is necessary to declare an optimizer and a loss function to validate the model. It can be declared in the compile method to compile the model.

In this project, some state-of-the-art algorithms are used for optimizer and loss function. Each model is trained using different algorithms. Initially all models were used the same algorithms. After each training and testing phase, I adjusted the algorithms to find which models were performed the best and keep them for application evaluation.

5.2 Models for training

In this project, FCN, U-Net, and pre-trained U-Net (using VGG16) are used to validate the dataset and test the performance of the models.

5.2.1 FCN

Fully Convolutional Networks are used to measure the accuracy of segmentation in this project. This model is simply constructed with a straightforward architecture which include five convolutional blocks with ReLU as activation function and then flattens out the data. The

Lambda layer is to normalise the input data into a range of 0-1 from RGB values whose data is from 0 to 255.

Because the architecture of FCN is quite big, a table will describe its architecture as below:

Table 3 Architecture of FCN model

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 256, 256, 3)]	0	[]
lambda	(None, 256, 256, 3)	0	['input_1[0][0]']
block1_conv1 (Conv2D)	(None, 256, 256, 64)	1792	['lambda[0][0]']
block1_conv2 (Conv2D)	(None, 256, 256, 64)	36928	['block1_conv1[0][0]']
block1_pool (Max-Pooling2D)	(None, 128, 128, 64)	0	['block1_conv2[0][0]']
block2_conv1 (Conv2D)	(None, 128, 128, 128)	73856	['block1_pool[0][0]']
block2_conv2 (Conv2D)	(None, 128, 128, 128)	147584	['block2_conv1[0][0]']
block2_pool (Max-Pooling2D)	(None, 64, 64, 128)	0	['block2_conv2[0][0]']
block3_conv1 (Conv2D)	(None, 64, 64, 256)	295168	['block2_pool[0][0]']
block3_conv2 (Conv2D)	(None, 64, 64, 256)	590080	['block3_conv1[0][0]']
block3_conv3 (Conv2D)	(None, 64, 64, 256)	590080	['block3_conv2[0][0]']
block3_pool (Max-Pooling2D)	(None, 32, 32, 256)	0	['block3_conv3[0][0]']
block4_conv1 (Conv2D)	(None, 32, 32, 512)	1180160	['block3_pool[0][0]']
block4_conv2 (Conv2D)	(None, 32, 32, 512)	2359808	['block4_conv1[0][0]']
block4_conv3 (Conv2D)	(None, 32, 32, 512)	2359808	['block4_conv2[0][0]']
block4_pool (Max-Pooling2D)	(None, 16, 16, 512)	0	['block4_conv3[0][0]']
block5_conv1 (Conv2D)	(None, 16, 16, 512)	2359808	['block4_pool[0][0]']
block5_conv2 (Conv2D)	(None, 16, 16, 512)	2359808	['block5_conv1[0][0]']
block5_conv3 (Conv2D)	(None, 16, 16, 512)	2359808	['block5_conv2[0][0]']
block5_pool (Max-Pooling2D)	(None, 8, 8, 512)	0	['block5_conv3[0][0]']

conv2d_transpose	(None, 512, 512, 13)	851968	['block3_pool[0][0]']
conv2d_transpose_1	(None, 512, 512, 13)	6815744	['block4_pool[0][0]']
conv2d_transpose_2	(None, 512, 512, 13)	2726297	['block5_pool[0][0]']
add	(None, 512, 512, 13)	0	['conv2d_transpose[0][0]', 'conv2d_transpose_1[0][0]', 'conv2d_transpose_2[0][0]']
final (MaxPooling2D)	(None, 256, 256, 13)	0	['add[0][0]']
conv2d	(None, 256, 256, 13)	182	['final[0][0]']
activation	(None, 256, 256, 13)	0	['conv2d[0][0]']

This model takes approximately longer time to train and more memory to store the parameters during training sessions. Every epoch takes about 160s which is around 550ms/step. The accuracy is saturated around 0.8, and the loss is around 0.6.

```
298/298 [==============================] - 166s 558ms/step - loss: 0.6785
- acc: 0.8050 - val_loss: 0.7261 - val_acc: 0.7917
```

A trained model is exported for testing. The model is quite big due to the transformation of pixel-to-pixel classes. The outcome of the model is not high in accuracy because it cannot be addressed well in the contextual information. [48]

Therefore, this model is suitable for being used for measurement and experimentation to show the early stage of semantic segmentation model experiment.

Figure 21 Prediction of FCN

The prediction image with overlay masks can show that most of details in the original masks are not predicted, for example the traffic signs and the traffic poles. However, it does well predicting vehicles (cars).

5.2.2 U-Net

The U-Net model in this project is constructed with two blocks, down block and up block, and a bottle neck with minimal convolutional shape.

The input layer accepts 256x256 image as default. Then it feeds to Conv2D layer to 16 output classes.

Down block contains the two layers of convolutional computation in between of MaxPooling layers. The shape of sample is reduced by 2 by each MaxPooling layer.

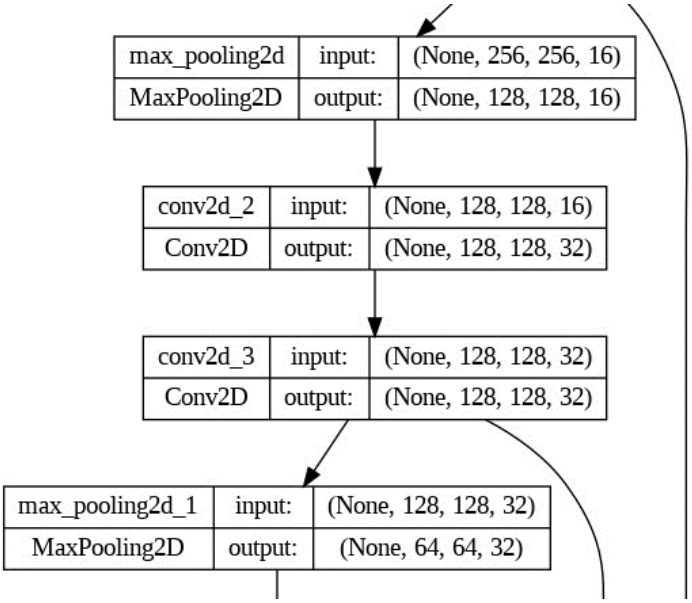


Figure 22 Sample of Up block in U-Net

The Neck bottle is the bottom of the U-Shape. It is the smallest shape of the samples. And it is the transition to start concatenating the samples together, which is reconstructing the image and its shape.

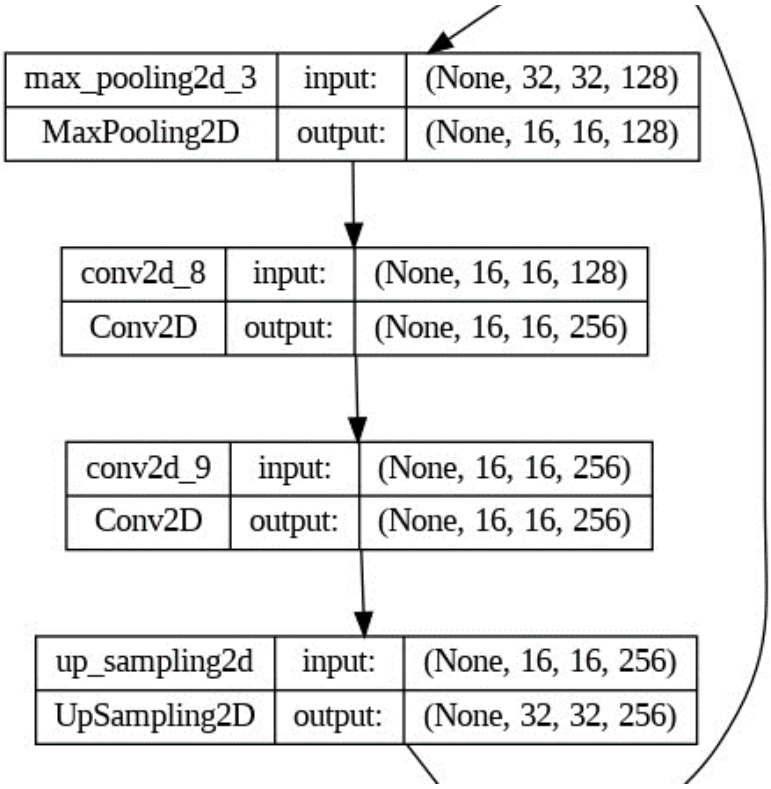


Figure 23 Bottleneck part of U-Net

The Up block contains concatenate layers in between of Convolutional layer with the same depth as Down block. And the output of each Concatenate layer is multiplied by 2.

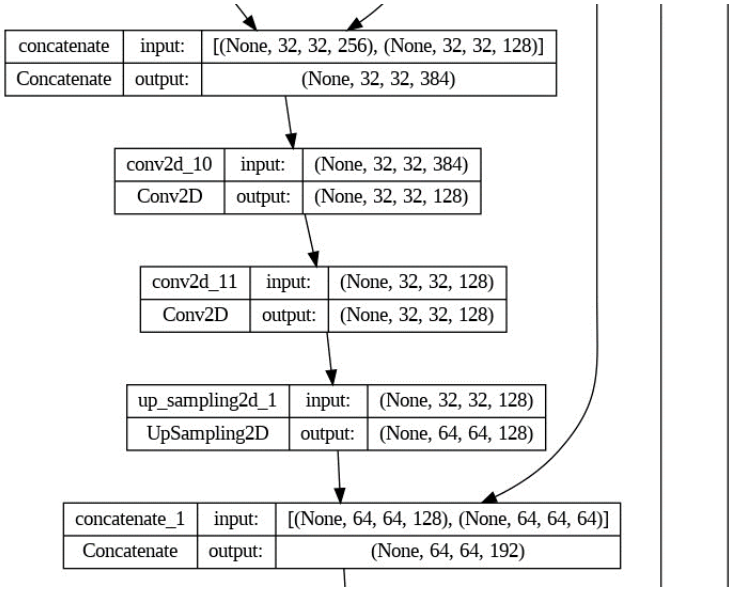


Figure 24 Up block architecture of the U-Net

There are connections between layers of convolutional layer with the same shape output of Concatenate layer. For example, Layer *Conv2D* will connect with layer *concatenate_1*.

Output layer will convolute the image to match the number of configured classes.

The U-Net can predict small details in an image better than the FCN and has smaller parameter size. The predict time is faster compared to FCN, which can be used for systems requiring faster performance instead of quality performance.

For this model, I use Adam as optimizer and categorical_crossentropy as loss function. It sets to 10 epochs and 298 steps per epoch.

```
298/298 [==============================] - 66s 223ms/step - loss: 0.6845 -
acc: 0.8029 - val_loss: 0.7902 - val_acc: 0.7783
```

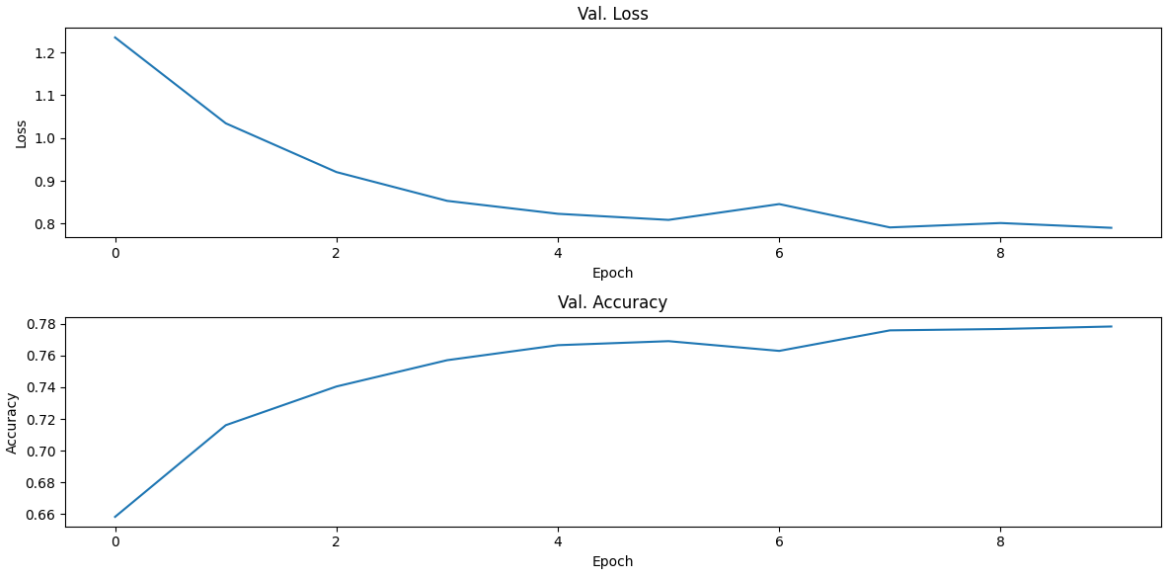


Figure 25 AUC of the model U-Net

As shown in Figure 25, the curve shows that the accuracy of the model is improving each epoch. At epoch 6, the accuracy is slightly decreasing. But it is improved in later epochs.

The Figure 26 shows the masks how they are formed compared to the original ground truth image. The masks are not formed well in shapes with the pavements. However, the poles on the pavement are well predicted.

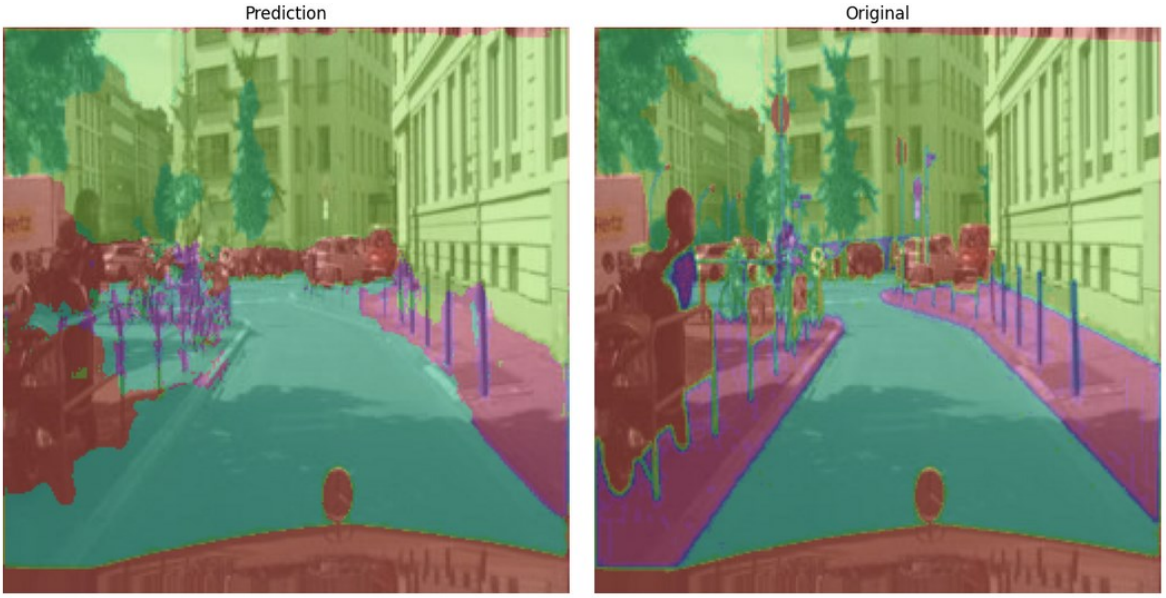


Figure 26 Predict masks using U-Net

5.2.3 Transfer trained weights using VGG16

VGG16 is used as pre-trained model for classification.

In Keras library, a VGG16 model is embedded in its applications class. To load the model to my local, I use below a line of code below to import the VGG16 class from library `keras.applications.vgg16`.

```
from keras.applications.vgg16 import VGG16
```

After the class is imported, it must be declared with weights from the applications. Since the Keras library contains many pre-trained weight sets which can be used for different scenarios, I have to declare which weight I want for my model. For segmentation solutions, I use 'imagenet' as pre-trained weight. It can be set as 'None' to randomize the choice of pre-trained weight, or a customized pre-trained weight can be loaded by setting the path to the weight with parameter 'weights' [49].

The VGG16 consists of 5 blocks of two Conv2D and a MaxPooling2D. By using this, the classification weights need to be kept promoting and taking advantages of pre-trained models. To customize my input, I keep only 'block1_conv1' as trainable, so I can put my images from the dataset.

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 input_1 (InputLayer)         [(None, 256, 256, 3)]     0
```

```
 block1_conv1 (Conv2D)        (None, 256, 256, 64)      1792

 block1_conv2 (Conv2D)        (None, 256, 256, 64)      36928

 block1_pool (MaxPooling2D)   (None, 128, 128, 64)      0

 block2_conv1 (Conv2D)        (None, 128, 128, 128)     73856

 block2_conv2 (Conv2D)        (None, 128, 128, 128)     147584

 block2_pool (MaxPooling2D)   (None, 64, 64, 128)       0

 block3_conv1 (Conv2D)        (None, 64, 64, 256)       295168

 block3_conv2 (Conv2D)        (None, 64, 64, 256)       590080

 block3_conv3 (Conv2D)        (None, 64, 64, 256)       590080

 block3_pool (MaxPooling2D)   (None, 32, 32, 256)       0

 block4_conv1 (Conv2D)        (None, 32, 32, 512)       1180160

 block4_conv2 (Conv2D)        (None, 32, 32, 512)       2359808

 block4_conv3 (Conv2D)        (None, 32, 32, 512)       2359808

 block4_pool (MaxPooling2D)   (None, 16, 16, 512)       0

 block5_conv1 (Conv2D)        (None, 16, 16, 512)       2359808

 block5_conv2 (Conv2D)        (None, 16, 16, 512)       2359808

 block5_conv3 (Conv2D)        (None, 16, 16, 512)       2359808

 block5_pool (MaxPooling2D)   (None, 8, 8, 512)         0

=================================================================
```

List above is the actual structure of VGG16 that is fetched from Keras library. The input layer is defined by users which will adapt to the actual dataset. As we set block1_conv1 as trainable and the rest are untrainable, so the weights stay remained. The output of VGG16 is (None, 8, 8, 512), so bottleneck layer is necessary to merge with the up-block of U-Net.

```
re_lu (ReLU)                (None, 8, 8, 512)            0
['block5_pool[0][0]']

 max_pooling2d (MaxPooling2  (None, 4, 4, 512)            0
['re_lu[0][0]']
 D)

 dropout (Dropout)           (None, 4, 4, 512)            0
['max_pooling2d[0][0]']

 conv_middle (Conv2D)        (None, 4, 4, 256)            1179904
['dropout[0][0]']
```

The bottleneck block consists of ReLU, MaxPooling, Dropout and a Conv2D layer. This is to adapt the output of the VGG16 to up-block of U-Net. The up-block of U-Net is reused from the previous section. Therefore the shape and the parameters are able to be compatible.

For this model, I use Adam as optimizer and categorical_crossentropy as loss function. It sets to 10 epochs and 298 steps per epoch.

```
298/298 [==============================] - 131s 439ms/step - loss: 0.6915
- acc: 0.8021 - val_loss: 0.7503 - val_acc: 0.7843
```

The output of accuracy is 0.892 and 0.7877 for validation accuracy. The loss is 0.6542 at rate and validation loss is 0.7254.

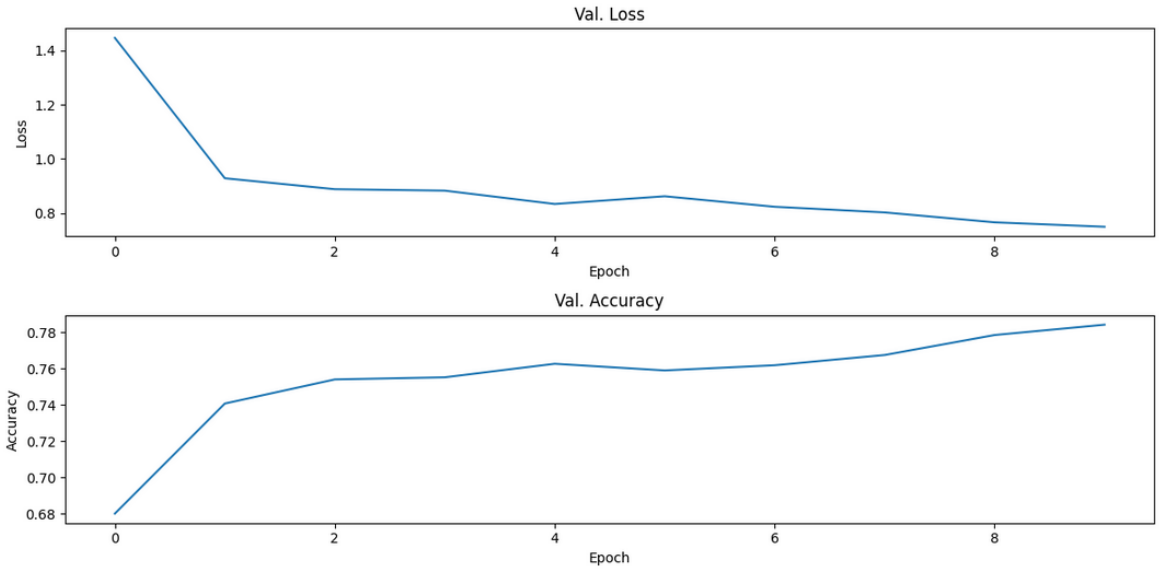


Figure 27 Training curve of 10 epochs for pre-trained model with VGG16

The model is tested with random images in the validation set. The model performs very well in such complex environments that it can predict pedestrians and traffic signs. The areas of objects are not confined well. However, the correct objects within the areas are predicted.

The model can detect pavement with better straight line and find correct pavement areas. The greens and buildings are good defined in such prediction. Vehicles on the road are detected correctly without confusion.

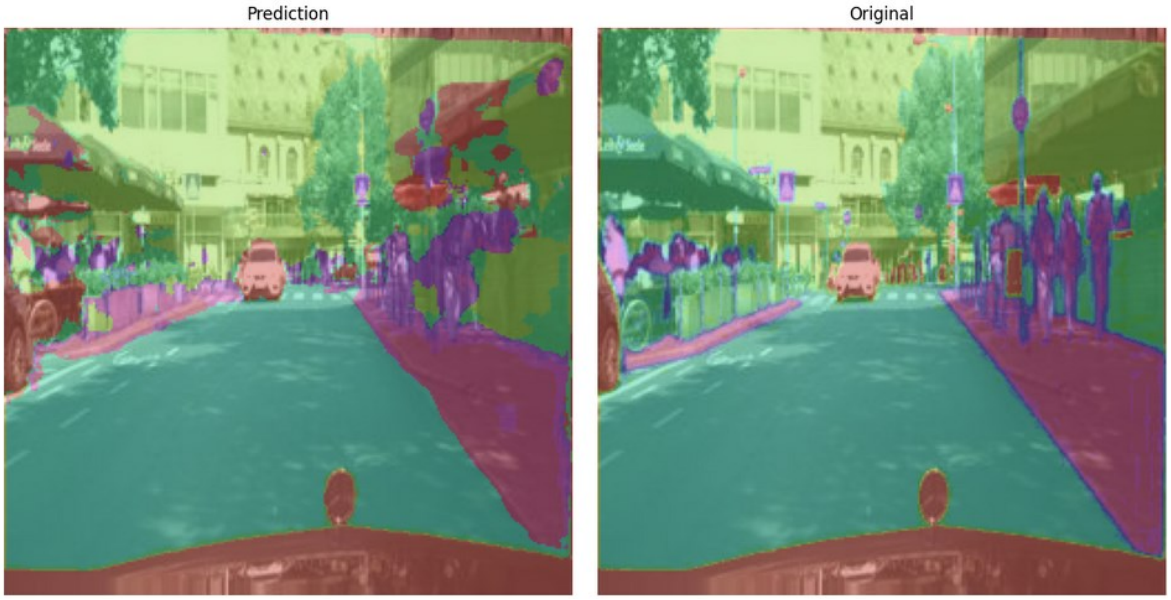


Figure 28 Prediction of complex scene on the road using pre-trained model VGG16

The model performs well on many scenes. Even though the confusion and the misprediction are still there. But the accuracy and the performance of the model increase drastically using pre-trained models.

6 EVALUATION

Three models are taken account to measure based on the accuracy of the output using AUC, the performance of the model based on practical measurement, and the observable details of output predictions.

All models can perform well to detect objects with label Cars, Building, and Green. However, FCNs can segment the road in less complicated scenes. U-Net model can segment smaller objects in an urban scene, and it can predict more various objects in more complex scenes. Even though the accuracy of both models is approximately the same with 0.8050 for FCN and 0.8060 for U-Net, the rapid increase of prediction is observable easily from sample images.

While using the models with real life images, especially images not included in the original train sets and test sets, the results are differed and give outcome with good results.

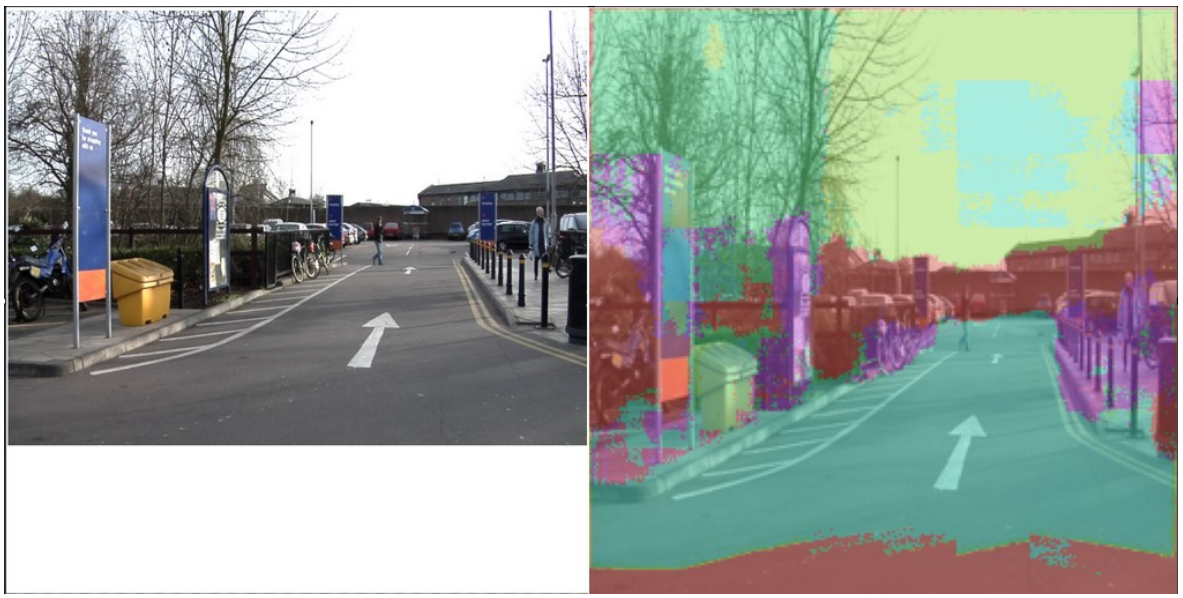


Figure 29 Predict with FCN

The FCN predicts traffic signs and bicycles, poles on pavement, and buildings in good shape and is converged to objects with refined areas. However, the distortion happens on sky area and in small details from far away. A disadvantage of FCN is the parameters and weights size are relative huge and proceed to generate predicted masks fairly slow.

Figure 30 Predict with U-Net

U-Net model loses its performance using real life scenes. It loses its ability to predict traffic signs and objects by the pavement. The pole is surprisingly well detected using U-Net model. It also give bad shapes for roads and pavements. The advantage of the model is the parameters and weights small and very fast generate predicted masks.

Figure 31 Predict with pre-trained U-Net

The pre-trained U-Net remains its accuracy with real life scenes. The traffic signs and the pavements are well segmented. The distortion in the predicted masks of the object is negligible but overall, the correct segmentation is formed. The size of the model is relatively

bigger than U-Net but still small and reliable. It also generates predicted masks fast relatively to U-Net model and faster than FCN.

Below is the summary of all parameters and output values from the models U-Net, FCN, and U-Net with VGG16.

Table 4 Comparison of FCN, U-Net, and U-Net with VGG16

	Total Params (Num - Size)	Loss	Val_loss	Acc	Val_acc	val_time
U-Net	7962829 (7.49 MB)	0.6845	0.7902	0.8029	0.7783	373ms/step
FCN	49645558 (189.38 MB)	0.6785	0.7261	0.805	0.7917	1s/step
U-Net VGG16	20927513 (79.83 MB)	0.6915	0.7503	0.8021	0.7843	745ms/step

7 FUTURE OF THE SOLUTION

The U-Net with pre-trained model potentially improves its accuracy by using different pre-trained models such as DeepLabv3 from Google. DeepLabv3 is well defined model with variety of different set of pre-train weights. The model can be adjusted layers and its algorithms namely activation functions, loss functions, or optimizers.

It can be improved by improving the datasets used. The current solution uses single view images from one camera which lack of information of an image such as depth, objects, or sharpness. There would be a model to extract the depth of an image, to refine blurred objects in far details or in such low quality of camera.

Beside the images from single view camera, there would be an aid from LiDAR camera that capture the real details of the object by objecting millions of dots to front view. Using LiDAR as an aid in trainsets and testsets, the accuracy can be improved.

The solution will be improved in the future with all the potential solutions mentioned above. The next generation of it is a desire to extract and reconstruct the objects from one camera. The motivation for this solution is that the 3D map can be more accuracy and reliable for auto-pilot vehicles. By reconstructing the object, mathematics can be applied to calculate the distance between colliders and can predict and extract more complex scenarios in real life operation.

CONCLUSION

In conclusion, this thesis has aimed to enhance road segmentation from single-view images by refining existing models to achieve superior solutions. The motivation of improving the robustness and accuracy of semantic segmentation for road scene using sigle view images is led to the novel approach of semantic segmentation that combines available solutions and adapts the existing models to it. To summarize the solution, the use of FCN, U-Net, and an upgraded U-Net variation based on VGG16 for road segmentation from single-view photos has shown encouraging results, with each having individual strengths and shortcomings. Both FCN and U-Net recognize common things with impressive precision, such as automobiles, buildings, and plants. FCN thrives in simpler settings, easily segmenting roadways, but U-Net shines in urban areas, successfully collecting tiny objects and managing more complicated scenarios.

Despite their equivalent overall accuracy, FCN has faster prediction capabilities but suffers from distortions in distant features and sky areas due to its considerably greater parameter size and slower processing speed. Conversely, U-Net's reduced parameter set allows for speedier predictions, but with limits in real-world applications, notably when anticipating traffic signs and pavement items. However, its pre-trained version remains accurate and effectively partitions objects with minimal distortion.

Moving forward, integrating the benefits of both models may improve road segmentation performance by exploiting FCN's quick prediction capabilities and U-Net's efficacy in real-world scenarios. Further study might look at solutions to address the observed flaws, such as model tuning or dataset augmentation approaches. Finally, the use of these models has the potential to improve road safety and enable autonomous navigation systems in a variety of environments.

BIBLIOGRAPHY

[1] 1. JACK KARSTEN, Darrell M. West, HENRY-NICKIE, Makada, SUNIL JOHAL, Daniel Araya, WHEELER, Tom, DARRELL M. WEST, Joseph B. Keller and J. SCOTT BABWAH BRENNEN, Matt Perault. How artificial intelligence is transforming the world. *Brookings* [online]. 27 June 2023. [Accessed 26 February 2024]. Available from: https://www.brookings.edu/articles/how-artificial-intelligence-is-transforming-the-world/

[2] HAENLEIN, MICHAEL & KAPLAN, ANDREAS. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. California Management Review. 61. 000812561986492. 10.1177/0008125619864925.

[3] SARKER, I.H. Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN COMPUT. SCI.* **2**, 160 (2021). https://doi.org/10.1007/s42979-021-00592-x

[4] Z.R. YANG, Z. YANG, 6.01 - Artificial Neural Networks, Editor(s): Anders Brahme, Comprehensive Biomedical Physics, Elsevier, 2014, Pages 1-17, ISBN 9780444536334, https://doi.org/10.1016/B978-0-444-53632-7.01101-1.

[5] 3. JACK KARSTEN, Darrell M. West, HENRY-NICKIE, Makada, SUNIL JOHAL, Daniel Araya, WHEELER, Tom, DARRELL M. WEST, Joseph B. Keller and J. SCOTT BABWAH BRENNEN, Matt Perault. How artificial intelligence is transforming the world. *Brookings* [online]. 27 June 2023. [Accessed 26 February 2024]. Available from: https://www.brookings.edu/articles/how-artificial-intelligence-is-transforming-the-world/

[6] 1. Iris. *UCI Machine Learning Repository* [online]. [Accessed 27 February 2024]. Available from: http://archive.ics.uci.edu/dataset/53/iris

[7] PATIL, Shreya U., 2024. Loss Functions and Optimizers in ML models - Geek Culture - Medium. *Medium* [online]. Available at: https://medium.com/geekculture/loss-functions-and-optimizers-in-ml-models-b125871ff0dc

[8] Anon., [b.r.]. Introduction to transforming data. *Google for Developers* [online]. Available at: https://developers.google.com/machine-learning/data-prep/transform/introduction

[9] SHUN-Ichi AMARI, Backpropagation and stochastic gradient descent method, Neurocomputing, Volume 5, Issues 4–5, 1993, Pages 85-196, ISSN 0925-2312, https://doi.org/10.1016/0925-2312(93)90006-O.

[10] WHITNEY K. Newey, Adaptive estimation of regression models via moment restrictions, Journal of Econometrics, Volume 38, Issue 3, 1988, Pages 301-339, ISSN 0304-4076, https://doi.org/10.1016/0304-4076(88)90048-6.

[11] 1. ROLLER, Joshua. A practical guide to working with testing and training data in ML Projects. *IEEE Computer Society* [online]. 26 June 2023. [Accessed 29 February 2024]. Available from: https://www.computer.org/publications/tech-news/trends/machine-learning-projects-training-testing

[12] KUZNETSOV, S.O. (2004). Machine Learning and Formal Concept Analysis. In: Eklund, P. (eds) Concept Lattices. ICFCA 2004. Lecture Notes in Computer Science(), vol 2961. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-24651-0_25

[13] MOHAIMINUL Islam, GUORONG Chen, SHANGZHU Jin. An Overview of Neural Network. American Journal of Neural Networks and Applications. Vol. 5, No. 1, 2019, pp. 7-11. doi: 10.11648/j.ajnna.20190501.12

[14] DRAELOS, Rachel, 2019. Measuring Performance: The Confusion Matrix. *Glass Box* [online]. Dostupné z: https://glassboxmedicine.com/2019/02/17/measuring-performance-the-confusion-matrix/

[15] KUNDU, Rohit, 2024. F1 Score in Machine Learning: Intro & Calculation. *V7* [online]. Available at: https://www.v7labs.com/blog/f1-score-guide

[16] BHANDARI, Aniruddha, 2024. Guide to AUC ROC Curve in Machine Learning : What is specificity? *Analytics Vidhya* [online]. Available at: https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/

[17] KILIÇ, İlyurek, 2023. ROC Curve and AUC: Evaluating Model Performance - İlyurek Kılıç - Medium. *Medium* [online]. Available at: https://medium.com/@ilyurek/roc-curve-and-auc-evaluating-model-performance-c2178008b02

[18] JIN HUANG and C. X. Ling, "Using AUC and accuracy in evaluating learning algorithms," in *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299-310, March 2005, doi: 10.1109/TKDE.2005.50.

[19] OLUDARE ISAAC ABIODUN, AMAN JANTAN, ABIODUN ESTHER OMOLARA, KEMI VICTORIA DADA, NACHAAT ABDELATIF MOHAMED, HUMAIRA ARSHAD. State-of-the-art in artificial neural network applications: A survey. Heliyon 4 (2018) e00938. doi: 10.1016/j.heliyon.2018. e00938

[20] KROMYDAS, Bill and Bill KROMYDAS, 2024. Convolutional Neural Network: A complete guide. *LearnOpenCV – Learn OpenCV, PyTorch, Keras, Tensorflow With Code, & Tutorials* [online]. Available at: https://learnopencv.com/understanding-convolutional-neural-networks-cnn/

[21] D. SCHERER, A. MULLER AND S. BEHNKE, "Evaluation of pooling operations in convolutional architectures for object recognition", *Proc. of the Intl. Conf. on Artificial Neural Networks*, pp. 92-101, 2010.

[22] Anon., [b.r.]. Max pooling layer. *NumPyNet* [online]. Available at: https://nico-curti.github.io/NumPyNet/NumPyNet/layers/maxpool_layer.html

[23] SCIENCE, Baeldung on Computer and Baeldung on Computer SCIENCE, 2024. How ReLU and dropout layers work in CNNs | Baeldung on Computer Science. *Baeldung on Computer Science* [online]. Available at: https://www.baeldung.com/cs/ml-relu-dropout-layers

[24] SUPERANNOTATE AI INC., [b.r.]. Convolutional Neural Networks: 1998-2023 Overview | SuperAnnotate. *SuperAnnotate AI Inc.* [online]. Available at: https://www.superannotate.com/blog/guide-to-convolutional-neural-networks

[25] B. DING, H. QIAN AND J. ZHOU, "Activation functions and their characteristics in deep neural networks," *2018 Chinese Control And Decision Conference (CCDC)*, Shenyang, China, 2018, pp. 1836-1841, doi: 10.1109/CCDC.2018.8407425.

[26] LI, Z. & NASH, W. & BRIEN, S. & QIU, Y. & GUPTA, RAJEEV & BIRBILIS, NICK. (2022). cardiGAN: A Generative Adversarial Network Model for Design and Discovery of Multi Principal Element Alloys.

[27] DOSHI, KETAN, 2022. Batch Norm Explained Visually — How it works, and why neural networks need it. *Medium* [online]. Available at: https://towardsdatascience.com/batch-norm-explained-visually-how-it-works-and-why-neural-networks-need-it-b18919692739

[28] ES-SABERY, FATIMA & HAIR, ABDELLATIF & QADIR, JUNAID & SAINZ DE ABAJO, BEATRIZ & GARCIA-ZAPIRAIN, BEGONA & DE LA TORRE

DÍEZ, ISABEL. (2021). Sentence-Level Classification Using Parallel Fuzzy Deep Learning Classifier. IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3053917.

[29] M. WANG, S. LU, D. ZHU, J. LIN AND Z. WANG, "A High-Speed and Low-Complexity Architecture for Softmax Function in Deep Learning," *2018 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, Chengdu, China, 2018, pp. 223-226, doi: 10.1109/APCCAS.2018.8605654.

[30] GRÉGOIRE MESNIL, YANN DAUPHIN, XAVIER GLOROT, SALAH RIFAI, YOSHUA BENGIO, IAN GOODFELLOW, ERICK LAVOIE, XAVIER MULLER, GUILLAUME DESJARDINS, DAVID WARDE-FARLEY, PASCAL VINCENT, AARON COURVILLE, JAMES BERGSTRA, Proceedings of ICML Workshop on Unsupervised and Transfer Learning, PMLR 27:97-110, 2012.

[31] XU HAN, ZHENGYAN ZHANG, NING DING, YUXIAN GU, XIAO LIU, YUQI HUO, JIEZHONG QIU, YUAN YAO, AO ZHANG, LIANG ZHANG, WENTAO HAN, MINLIE HUANG, QIN JIN, YANYAN LAN, YANG LIU, ZHIYUAN LIU, ZHIWU LU, XIPENG QIU, RUIHUA SONG, JIE TANG, JI-RONG WEN, JINHUI YUAN, WAYNE XIN ZHAO, JUN ZHU, Pre-trained models: Past, present and future, AI Open, Volume 2, 2021, Pages 225-250, ISSN 2666-6510, https://doi.org/10.1016/j.aiopen.2021.08.002.

[32] RIJWAN KHAN, MOHAMMAD AYOUB KHAN, MOHAMMAD ASLAM ANSARI, NIHARIKA DHINGRA, NEHA BHATI, Chapter 1 - Machine learning-based agriculture, Editor(s): Mohammad Ayoub Khan, Rijwan Khan, Mohammad Aslam Ansari, Application of Machine Learning in Agriculture, Academic Press, 2022, Pages 3-27, ISBN 9780323905503, https://doi.org/10.1016/B978-0-323-90550-3.00003-5.

[33] FARHANA SULTANA, ABU SUFIAN, PARAMARTHA DUTTA, Evolution of Image Segmentation using Deep Convolutional Neural Network: A Survey, Knowledge-Based Systems, Volumes 201–202, 2020, 106062, ISSN 0950-7051, https://doi.org/10.1016/j.knosys.2020.106062.

[34] TEJA KATTENBORN, JENS LEITLOFF, FELIX SCHIEFER, STEFAN HINZ, Review on Convolutional Neural Networks (CNN) in vegetation remote sensing, ISPRS Journal of Photogrammetry and Remote Sensing, Volume 173, 2021, Pages 24-49, ISSN 0924-2716, https://doi.org/10.1016/j.isprsjprs.2020.12.010.

[35] BADRINARAYANAN, VIJAY, et al. *SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation*. 3, arXiv:1511.00561, arXiv, 10 Oct. 2016. *arXiv.org*, https://doi.org/10.48550/arXiv.1511.00561.

[36] EVAN SHELHAMER, JONATHAN LONG, AND TREVOR DARRELL, Member, Fully Convolutional Networks for Semantic Segmentation, IEEE, arXiv:1605.06211v1, https://arxiv.org/pdf/1605.06211v1.pdf

[37] LI, Y.; CHEN, Y.; LIU, G.; JIAO, L. A Novel Deep Fully Convolutional Network for PolSAR Image Classification. *Remote Sens.* 2018, *10*, 1984. https://doi.org/10.3390/rs10121984

[38] OLAF RONNEBERGER, PHILIPP FISCHER, AND THOMAS BROX, Computer Science Department and BIOSS Centre for Biological Signalling Studies, University of Freiburg, Germany, arXiv:1505.04597

[39] KAREN SIMONYAN & ANDREW ZISSERMAN, Visual Geometry Group, Department of Engineering Science, University of Oxford, {karen,az}@robots.ox.ac.uk, arXiv:1409.1556v6

[40] ABADI, MARTÍN; BARHAM, PAUL; CHEN, JIANMIN; CHEN, ZHIFENG; DAVIS, ANDY; DEAN, JEFFREY; DEVIN, MATTHIEU; GHEMAWAT, SANJAY; IRVING, GEOFFREY; ISARD, MICHAEL; KUDLUR, MANJUNATH; LEVENBERG, JOSH; MONGA, RAJAT; MOORE, SHERRY; MURRAY, DEREK G.; STEINER, BENOIT; TUCKER, PAUL; VASUDEVAN, VIJAY; WARDEN, PETE; WICKE, MARTIN; YU, YUAN; ZHENG, XIAOQIANG (2016). TensorFlow: A System for Large-Scale Machine Learning. Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). arXiv:1605.08695.

[41] PULLI, KARI; BAKSHEEV, ANATOLY; KORNYAKOV, KIRILL; ERUHIMOV, VICTOR (1 April 2012). "Realtime Computer Vision with OpenCV". doi:10.1145/2181796.2206309

[42] Anon., Models and pre-trained weights — Torchvision 0.17 documentation. (n.d.). https://pytorch.org/vision/stable/models.html

[43] M. CORDTS, M. OMRAN, S. RAMOS, T. REHFELD, M. ENZWEILER, R. BENENSON, U. FRANKE, S. ROTH, AND B. SCHIELE, "The Cityscapes Dataset

for Semantic Urban Scene Understanding," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[44] N. GÄHLERT, N. JOURDAN, M. CORDTS, U. FRANKE, and J. DENZLER, "Cityscapes 3D: Dataset and Benchmark for 9 DoF Vehicle Detection", 3D bounding box annotations of vehicles for train and val sets (3475 annotated images). CVPRW 2020.

[45] Isola, Phillip, et al. *Image-to-Image Translation with Conditional Adversarial Networks*. arXiv:1611.07004, arXiv, 26 Nov. 2018. *arXiv.org*, http://arxiv.org/abs/1611.07004.

[46] HUANG SY, HSU WL, HSU RJ, LIU DW. Fully Convolutional Network for the Semantic Segmentation of Medical Images: A Survey. Diagnostics (Basel). 2022 Nov 11;12(11):2765. doi: 10.3390/diagnostics12112765. PMID: 36428824; PMCID: PMC9689961.

[47] S, PREMANAND, 2023. Top 8 interview questions on UNET architecture. *Analytics Vidhya* [online]. Available at: https://www.analyticsvidhya.com/blog/2023/01/top-8-interview-questions-on-unet-architecture/

[48] HAOFU LIAO, S. KEVIN ZHOU, JIEBO LUO, Chapter 5 - Segmentation: intracardiac echocardiography contouring, Editor(s): Haofu Liao, S. Kevin Zhou, Jiebo Luo, In The MICCAI Society book Series, Deep Network Design for Medical Image Computing, Academic Press, 2023, Pages 89-107, ISBN 9780128243831, https://doi.org/10.1016/B978-0-12-824383-1.00013-7.

[49] TEAM, K. (n.d.). *Keras documentation: VGG16 and VGG19*. https://keras.io/api/applications/vgg/#vgg16-function

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
SVM	Support Vector Machine
k-NN	k-nearest neighbors
NN	Neural Network
PCA	Principal component analysis
t-SNE	t-distributed stochastic neighbor embedding
DQN	Deep Q-Network
MSE	Mean squared error
MAE	Mean absolute error
SGD	Stochastic gradient descent
Adam	Adaptive Moment Estimation
AUC-ROC	“Area Under the Curve” of the “Receiver Operating Characteristic”
ANN	Artificial Neural Networks
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
tanh	Hyperbolic Tangent
ReLU	Rectified Linear Unit
H x W x C	Height x Width x Channel as in RGB image
BN	Batch Normalization
VGG16	Visual Geometry Group, VGG model with 16 layers
FCN	Fully Convolutional Network

LIST OF FIGURES

LIST OF TABLES

APPENDICES

https://www.cityscapes-dataset.com/