EXPLORING MONOCULAR DEPTH ESTIMATION FOR AUTONOMOUS VEHICLES UTILISING A SELF-SUPERVISED LEARNING-BASED ARTIFICIAL INTELLIGENCE MODEL IN A LIMITED-HARDWARE ENVIRONMENT

Calum Murphy
Newcastle University
School of Computing
Degree Course - G405 MComp (Hons). Computer Science
C.Murphy1@newcastle.ac.uk

ABSTRACT

Supervised-Learning is currently the dominant paradigm in machine learning, with notable usage in Autonomous Vehicles for tasks such as Monocular Depth Estimation, however, it requires large amounts of labelled data. This paper aims to explore Self-Supervised Learning as a potential alternative to the Supervised-Learning paradigm, whereby instead of using a labelled dataset, unlabelled data is instead processed in other ways, allowing for the learning of useful representations. The self-supervised learning solution employed leverages the standard-quality computer hardware available to explore the solution to the problem with the lens of a more real-world environment, compared to the extremely powerful hardware used as standard when training models - the intent of which was to produce a realistic benchmark of the state of Self-Supervised Learning. The obtained results suggest that even within a limited-hardware environment, Self-Supervised Learning is a paradigm suited for the application and shows promise for more widespread application in future.

Declaration

I declare that this dissertation represents my own work except where otherwise stated.

Acknowledgements

I would like to thank my supervisor, Dr Varun Ojha, for his guidence throughout this project, and Michael Tweddle MSc (School of Mathematics Alumni) for assistance with LaTeX formatting.

Contents

Figures

1 Introduction

This section serves to give an overview of the Structure of the paper, briefly introduce the Context of the project, discuss the obstacles present in the field, and finally, outline the objectives it is aiming to achieve.

1.1 Structure

- Introduction - A brief overview of the project (see above)
- Planning - An outline of the Project's plan
- Technical Background - A discussion and evaluation of the myriad background material researched for the project
- Design - A conceptual outline of the overall Design of the project
- Implementation - A high level overview of the Implementation and Application of the methods and tools constructed in the Design phase
- Testing - A showcase of the Testing that was undertaken upon the Implementation
- Results & Evaluation - A thorough review of the results collected during the project, including critique and defence of the results themselves and the methods used to collect them.
- Conclusions - Discussing the development of the initial goal of the project into the end result; conceptualising novel plans for future research, and drawing conclusions.

1.2 Purpose

One of the most promising and well-publicised aspects of current AI development, is that of the field of Autonomous Vehicles (AVs). Autonomous Vehicles that are currently on the market make heavy use of the established "Supervised-Learning" (SL) paradigm. With SL, the model is fed significant amounts of labelled data - this is data that has been meticulously processed by a human engineer in order to explicitly define specific properties of it (for example, the exact depth from the camera displayed in an image). This labelled data is then used to accomplish specific tasks such as Object Detection, Semantic Segmentation and - relevant to this project - Depth Estimation. In future, however, a different paradigm known as "Self-Supervised Learning" that does not require labelled data could be used as an attractive, and potentially more robust, alternative. This project will endeavour to explore a Self-Supervised Learning-based solution to Depth Estimation: one of the vital elements of Autonomous Vehicles, serving as a proof-of-concept for the potential of the approach when compared to the established Supervised approach.

1.3 Obstacles

In recent years, it has been widely reported that the Artificial Intelligence systems used in order to allow for vehicle autonomy have encountered significant setbacks[1]. Though the issues encountered by contemporary AI systems have a plethora of causes, it would be valid to state that among these failings is that of the Artificial Intelligence model itself - that is, Supervised Learning. In the context of Autonomous Vehicles, Supervised Learning has been branded "notoriously data-hungry", requiring "extensive annotation"[2] of the provided data. This is an inherent issue in SL, and results in a colossal amount of man-hours being required to process the sheer amount of raw data provided, in order to train the AI to a good standard. Furthermore, this requirement for large-scale labelled data means that when confronted with the incalculable mass of probable scenarios that arise from even the safest of drives, any abnormalities have the potential to cause unwanted behaviour, as the Supervised model has no hard-annotated data to "fall back" on. As alluded to previously, this has the potential to cause serious harm to both the driver, and other potential road users, and thus a solution is required. While there are many alternative paradigms to Supervised Learning, each with their own promise (such as Reinforcement Learning or Transfer Learning), this work aims to explore Self-Supervised Learning. Self-Supervised Learning differs from the established Supervised paradigm as it allows for training on no labelled data whatsoever. The immediate boon of this being that the time that would have to be spent by human engineers meticulously preparing the dataset is immediately erased. The caveat is that an additional section must be added to the model - referred to as a pretext task (to be discussed in the Design section).

1.4 Objectives

The key objective of this project is to create a model that, given a relevant unlabelled data set, can estimate (to a suitable degree) the required depth in the image, using a variant of the ResNet-18 neural network. The model utilises a designated "pretext task" - a special training method used in Self-Supervised Learning; training the model in order to

allow it to learn useful representations and generalisations inherent in the data itself. This is in contrast to training an AI on labelled data where the depth inputs and outputs are already known and provided. Should the model successfully be able to estimate the depth to a suitable degree, it will provide credence to the notion that unlabelled data can - and should - be used (where relevant) in the field of Autonomous Vehicles.

Since the project proposal, the overall aim of the project has somewhat shifted due to understanding gained from further research into the field - namely that training a model that would be on par with that of an industry standard is unlikely due to the limitations of available hardware. It should therefore be stated that the aim of this project is not to create a necessarily superior model to that of existing supervised models, as these tools are often trained using vast amounts of computational power, colossal datasets, and thousands of hours of training time. Rather, this project's aim is to ultimately serve as a proof of concept and benchmark of the current power and capability of Self-Supervised Learning, within the scope of standard desktop hardware. While the end results will likely not eclipse these industry-standard supervised models, the ability to get remotely close to the same level of success with a fraction of the power and time should serve as a clear indication to the potential of Self-Supervised Learning as the dominant paradigm of the future.

2 Planning

This section serves to give an overview of the Planning Stage of the project that preceded the Design and Implementation. What follows is a display of the work done that ensured realistic progress markers were established and met along an established timeframe.

2.1 Development Model

In order to accomplish the intended goals of the project, a sound development methodology was needed - through this, progress at a reasonable pace could be assured. Considering a myriad of available methodologies, two appeared to be the most workable - Agile and Waterfall.

Agile (or a related derivative methodology) is ubiquitous in the modern landscape of professional software development[3], and as such was the first to be considered for this project. However, after considering that due to a lack of base knowledge about the subject matter at the beginning of the project, it was apparent that an entire phase of development had to be dedicated to appropriate research. Evident from this was that the project would instead need to take on a far more rigid structure. Therefore the decision was made to use the arguably antiquated Waterfall methodology[4]. Despite its reputation, the clear drawbacks are exactly what forged it into an asset for this project - that is to say its use of rigid deadlines and necessary vision of the end goal.

2.2 Project Plan

Following from the decision to employ the Waterfall method, a project plan was devised according to that structure. This project plan was represented as a Gantt Chart, and contains representations for Hard Deadlines and Deliverables. At the point at which planning was undertaken, several risk factors were also taken into account and managed. These included contingencies for illness or other indisposal. Indeed, as a result of events outside of the realm of forsight occuring during the course of this project, some of these elements of contingency had to be considered. However, this was largely mitigated by the fact that, for the majority of the duration, the project was running on schedule, only encountering challenges during the final weeks. These were met head on, and the project's results are proof of this.

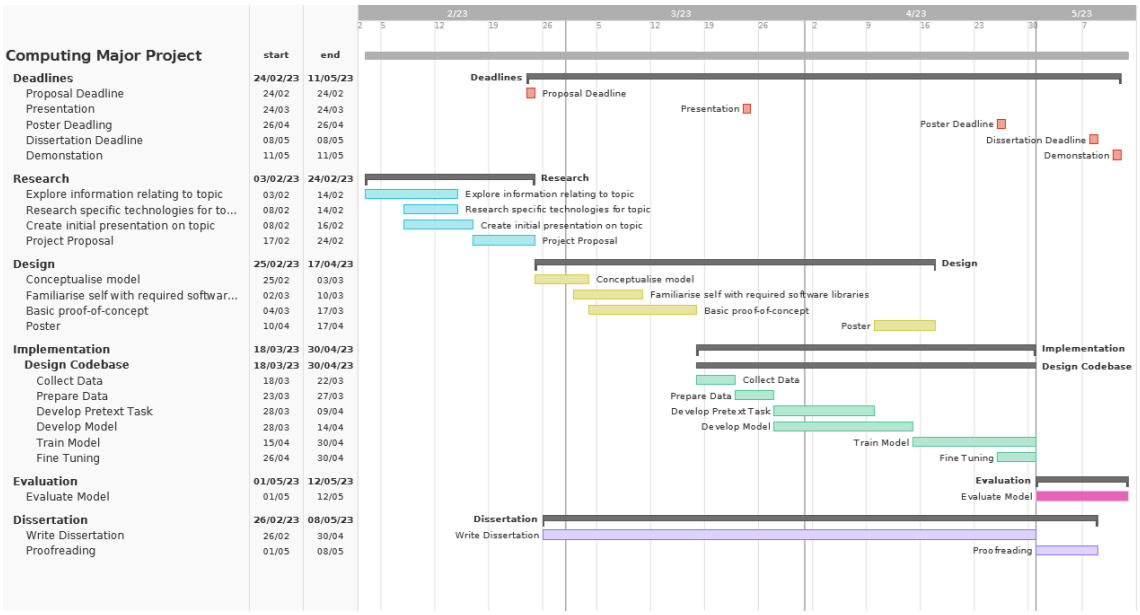


Figure 1: Gantt Chart for project, detailing stages of the Waterfall Development Method, along with Hard Deadlines and Deliverables.

3 Technical Background

This section serves to outline the project's Technical Background. This is comprised of evaluative research into a diverse range of sources, in order to draw conclusions that will be made manifest in the project's Design and Implementation.

3.1 Artificial Intelligence

The notion of Artificial Intelligence itself has been around for millennia, being an ever-present fascination in the collective consciousness of mankind[5], however it was not until the advent of digital computing that truly intelligent automata[6] became a possibility. The first notion of what today we would refer to as a "neural network" was devised by Warren McCulloch and Walter Pitts in 1943, with their model being based on the behaviour of real neurons present in human brains. More specifically, they proposed that a neuron could be modelled as a "binary switch", with the individual cells then being connected to synthesise more complex computations[7]. In the following decades, researchers such as Frank Rosenblatt would attempt to create Neural Networks for pattern recognition[8], however the computational limitations of the time, paired with the rise of preferred alternatives (such as "decision trees"), lead to declining interest by the 1980s and 1990s.

However, by the 2000s, advances in computational power and new techniques spurred a renaissance in the field - with one of the key advances being that of "Deep Learning"; a technique whereby a neural network is constructed using multiple "layers", with each layer transforming the data to produce further abstract representations. The result of this process is that the model learns increasingly abstract representations of data.

The year in which this dissertation has been penned represents a globally significant time for Artificial Intelligence. While in the minds of those outside of academic circles, AI has been previously something reserved only for fiction, the widespread popularity (and controversy) of tools such as "Stable Diffusion" and OpenAI's "ChatGPT"[9] have brought the reality of the field into the public eye and into popular culture. What has been known for a long time in academia, however, is that this miraculous "deep learning" breakthrough is not some sort of "silver bullet". Indeed, in order for these kinds of neural networks to operate, they must be trained on massive amounts of data.

3.2 Self-Supervised Learning

Whilst the idea of "self-supervised learning", at least in the philosophical sense of self-taught automata, has been around for centuries, the term itself in the field of artificial intelligence is relatively new, prior to that being referred to as "unsupervised learning". Regardless of shifting nomenclature, however, the basic idea is the same - forge a way for Machines to learn without the need of explicitly labelled data[10].

LeCun draws attention to the way in which babies learn - observing the world and forming generalised and predictive mental models, such as the concept of object permanence or gravity. Then, as we grow and learn more about the world, we alter these general models based on trial and error. We call these generalisations "common sense", and it has become regarded as the "dark matter of artificial intelligence"[11]. When we take standard Supervised Models, their understandings are rigid, and are often unable to perform their intended function when unexpected alterations are made to the input (e.g. an Animal Image Recognition model being unable to recognise a cat as it is lying upside down). It is believed that SSL is the way in which a machine can be imbued with this capability to learn generalised understandings instead of rigid knowledge. This is done through the use of a pretext task, mentioned earlier in the Introduction.

Self-Supervised Learning can be thought of in terms of a so-called "energy based model" (EBM). This EBM represents a trainable system that, given two inputs, returns how incompatible they are. For example, if a model was designed to pick an appropriate jigsaw piece to connect to an already placed piece, the model would tell the user to what extent the unplaced piece (X) was an appropriate fit for the piece-in-situ (Y). This value of incompatibility is called "energy". It must be kept in mind however that this is just a theoretical model. In practice, as is demonstrated in the next chapter, the "energy" can be thought of as the loss of the pretext task.[11]

Whilst the energy model is a useful theoretical tool for exploring the way in which a pretext task operates, it is unfortunately just that - theoretical. In practice, the notion that the task compares the compatibility of two inputs is self-evident. Furthermore, the idea of comparing the energy to the pretext task's loss opens the possibility for unnecessary confusion, as the latter corresponds explicitly to a datapoint, whereas the former is an abstract concept used to aid human visualisation.

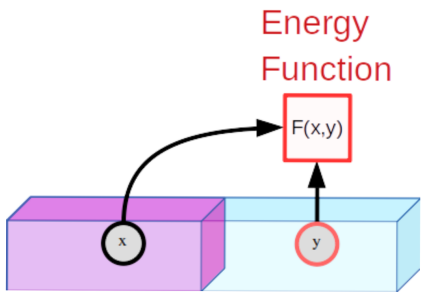


Figure 2: Overview of the Energy-Based Model (Image credit: Lecun, Y.[11])

3.3 Pretext Task

A pretext task is a construct created specifically for a self-supervised model that allows for the model to send itself training signals, rather than getting them from labelled data - the end goal of this is to learn useful representations of the data. In other words, rather than learning explicit aspects of the data to attempt to judge, as in the case of Supervised Learning, the system is pushed to learn more subtle, underlying structures in the data.

To give an example - in a labelled depth-estimation scenario, the model currently being trained would make its "guess" on the depth of a section of the image relative to the camera. It would then cross reference this with the depth that had been labelled. It would then make note of the difference by which its guess was incorrect (or "loss"). This serves as the training signal, which then amends the model itself before making the next guess.

Now in a self-supervised scenario, the model makes its guess, and then refers to a pretext task, which forces the model to apply its current understanding of the problem, or representation, to the task. The model will make its "guess" on the task, and then be given its loss, amending the model appropriately. Whilst this appears to be a small difference, it is a fundamental change in the way in which the model learns. The representation obtained through use of the pretext task is then used for "downstream" tasks, or tasks that use this representation as their input.

3.4 Monocular Depth Estimation

"Monocular Depth Estimation" (MDE) refers to the task of predicting the "depth map" of a given RGB image - with the depth map being a description of the distance from each "point" in the scene to the camera. The objective of this is to provide information about the three-dimensional structure of the scene[12].

Prior to the advent of Deep Learning, techniques based around extracting the edges and textures of the image to infer information about the depth were utilised, however these methods often made assumptions about the scene[13].

With the aforementioned advances in artificial intelligence-based methods, Monocular Depth Estimation has seen a new wave of interest in recent years. The basic principle behind deep-learning based MDE is to use a neural network to learn mappings from the input image to the depth map. In standard SL implementations, the network is trained on large datasets of pairs of images and corresponding depth-maps, gradually minimising the difference between the predicted depth map and the "ground-truth" (real) depth map[14].

There are multiple approaches within the realm of deep learning that are used when tackling this task, however the two major standards are "Convolutional Neural Networks" (CNNs) and "Fully Convolutional Networks" (FCNs). While these two types of neural network are similar, the main principle of separation is that an FCN is a CNN that only performs convolution (covered below) - it should be noted therefore that while all FCNs are CNNs, this relationship is not the case vice-versa. Many prominent Neural Networks exist following both standards, and experimentation was done with the implementation of the UNet FCN, however after re-evaluating its large performance requirements, and examining the MiDaS supervised monocular depth estimation implementation (covered below), which uses a bespoke variation of the ResNet18 CNN, it was decided that an implementation of ResNet18 would be more appropriate.

In standard SL Implementations of Monocular Depth Estimation, the availability of good-quality ground-truth depth maps for the corresponding images in the dataset is a vital requirement. Datasets such as NYUv2, KITTI, and Make3D, aimed at Monocular Depth Estimation, feature these depth maps as well as their raw data to allow for the training of supervised networks. While the Self-Supervised Implementation does not require the depth map, the raw version of the dataset still presented an important decision (discussed below).

Whilst Monocular Depth Estimation is not necessarily the only method of determining depth for autonomous vehicles (the primary method being Depth Sensors), it is a vital requirement in cases where depth sensing is not readily available (which can occur for a plethora of reasons)[15]. As such, reliable methods for determining depth in real-time are vital for the success of Autonomous Vehicles.

3.5 Convolution

In order for Monocular Depth Estimation to be possible, the Convolutional Neural Networks used must perform the process known as Convolution. This is a mathematical operation that extracts features from the input image, which involves applying "learnable filters" - also referred to as "kernels" - to the input image to produce "feature maps". The kernels slide over the image, with the convolution operation computing the dot product at each location between the filter and the input image's respective pixel. The scalar result of this computation is then used to populate the corresponding area of the feature map. This process is then repeated for all other locations on the input image resulting in a set of feature maps that all encode different aspects of the input.

Convolution is suited for Monocular Depth Estimations for a variety of reasons. Firstly, convolution is translation invariant, meaning that it can detect features of the image regardless of their position in it. For Monocular Depth Estimation, this is ideal as the depth of an object does not necessarily relate to its position on the image. Secondly, convolution is parameterized, meaning that the CNN can be trained to optimise the filters and extract only the relevant features for depth estimation. Finally, convolution as an operation is computationally efficient, which allows for fast processing of datasets.[16]

3.6 Machine Learning Frameworks

Two of the major Machine Learning Frameworks existing today are for use with the Python programming language - that being Tensorflow (specifically the Keras module) and PyTorch. Both libraries are highly regarded and have both seen heavy use in industry and academia.

Statistics suggest[17] that in recent years, the number of available models for PyTorch has grown significantly, whereas those for Tensorflow have gradually reduced by comparison. Despite this however, Tensorflow does have an active community, and should not be regarded as a relic of antiquity. Indeed, the fact that the core of this project relies on designing a model - in contrast to utilising a pre-trained one - makes the issue of "available models" far less relevant.

Therefore, rather than selecting one or the other based on pure capability, the consideration became wholly practical - whether or not Tensorflow or PyTorch would work with the specific requirements. This would be determined in the Design and Implementation.

3.7 Supervised Implementation

Clear from the outset of the project was the need for a Supervised Learning implementation in order to have something to compare the eventual Self-Supervised output to. Referred to as "pretrained models", these are not pieces of complex code that have to be meticulously implemented, but generally just a file to be loaded by a relatively simple piece of operational code.

The reason for this is that these models have already been heavily trained on large amounts of data for significant amounts of time[18][19]. The exact nature of this training has been discussed earlier on within this chapter, and so will not be retreaded.

There are a myriad of examples of possible pretrained models to use, but as the exact specificity of results of these models are not the key piece of information being evaluated for this project, the deciding factor was predominantly the ease-of-implementation.

3.8 Dataset Choice

Unlike the choice of Model, wherein the choice of dataset has already been made by the engineers who have trained it, there is a free choice for that of the Self-Supervised system's model. In the case of Monocular Depth Estimation, the model is required to be one that is reflective of real-world situations, as well as reflecting diversity in lighting conditions (such as those that will come from weather, direct sunlight, shade, etc).

Some of the major datasets specifically for Monocular Depth Estimation are that of NYU Depth V2, Make3D, Cityscapes, and KITTI - however, the most cited datasets are KITTI and NYUv2[20].

These two datasets both contain a large amount of high quality images, however KITTI is specifically designed for use with Autonomous Vehicles, with the images being captured from a moving vehicle - whereas NYUv2 is interior environments. As a result, KITTI was chosen.

4 Design

This section outlines the project's Design phase. In particular, it details how appropriate methods and tools were derived from conclusions obtained from the research detailed in the prior section, and logical design processes were then undertaken to reach a state where a realistic Implementation would be possible.

4.1 Architecture

The overarching architecture of the project is relatively straightforward. As this project is not intended for commercial use, a Command Line Interface was deemed suitable. From there, two considerations were necessary: how would the

project allow a user to choose between SL or SSL systems; and how would the project handle these two - diametrically opposed paradigms?

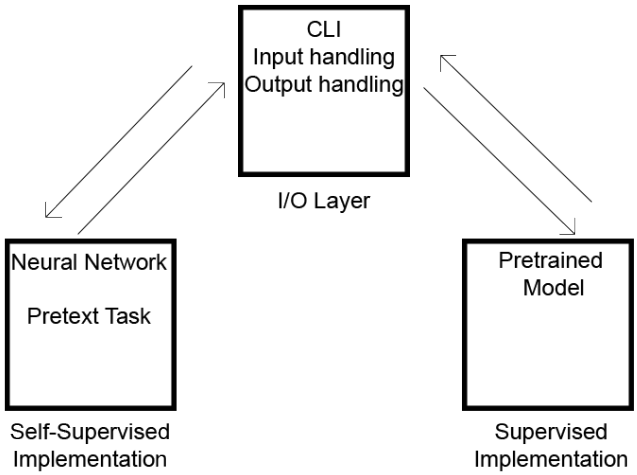


Figure 3: High level design of proposed architecture for the system - featuring the three core components

4.2 Technology Used

4.2.1 Hardware

- AMD Ryzen 5 2600 CPU
- EVGA GeForce GTX 960 (2 GB V-RAM)
- 16 GB DDR4 RAM
- 640x480 resolution webcam (for depth estimation)

4.2.2 Software

Python In order to implement the conceptual ideas above, a suitable language was required. The clear choice for this was Python. Due to its platform independence and relative ease-of-use, Python has already become the de facto language of choice for Machine Learning Implementation[21]. With this fact in mind, a python programmer has a plethora of purpose-built Machine Learning software-libraries at their disposal; therefore to choose anything else would have created a large amount of extra work.

PyTorch While PyTorch is the fastest growing Machine Learning framework[22], it was ultimately chosen because of the ease of use when compared to TensorFlow's Keras. When experimenting with early implementations near the project's beginning, TensorFlow proved to be unintuitive, with the lack of easy readability of its syntax being a barrier to entry. Moving to PyTorch from there was a logical step and progress resumed far quicker after this. PyTorch's readability - while subjective to the individual programmer - proved to be a benefit as the project moved to its final stages as it allowed for far easier maintenance than a hypothetical TensorFlow project. Additionally, after discovering the MiDaS supervised model and deeming it ideal for the Supervised implementation, the move to PyTorch garnered further value, as the aforementioned model is built for this framework.

MiDaS MiDaS (Mixed Depth Network for Single Image Depth Estimation) is a Supervised Learning model, designed specifically for Monocular Depth Estimation. It is currently regarded as having so-called "state-of-the-art" performance[18] on commonly used benchmark datasets. The MiDaS model is trained on a bespoke system, based on the ResNet18 CNN that employs additional features. MiDaS' stand-out feature however is that it is trained on a variety of datasets - comprising 72,000 images, which is then repeated for 60 epochs.

The MiDaS model is included as a model within the *torch.hub*[23] package from PyTorch, where it can be imported to the project in a mere line of code. As a result of this ease-of-implemntation, the MiDaS model was selected.

OpenCV OpenCV (Open Source Computer Vision) is a Python Library designed for real-time computer vision tasks. In this project, it is utilised primarily on the I/O layer of the project in order to handle real-time webcam input.

KITTI The KITTI (Karlsruhe Institute of Technology and Toyota Institute) Dataset is a high-quality image dataset designed for use with Autonomous Vehicles, containing 7,481 raw images. Whilst for standard SL implementations, the image-depth-map pairs are the most important aspect, pre-calculated depth maps are irrelevant for this Self-Supervised Implementation.

Matplotlib Matplotlib is a standard library for scientific plotting in Python. For this project, it serves as the output for real-time depth-maps in the I/O layer.

4.3 I/O layer and Supervised Model

From the inception of the project, it was known that an I/O layer would have to be implemented. This would have to fulfil several goals:

- Serve as a "hub" for all testing, allowing for multiple different models to be used with relative ease
- Take real-time input from a webcam and return the output
- Have some form of basic interface
- Allow for utilisation of Self-Supervised Model, to be loaded from an external file

Through rudimentary research, using relevant PyTorch and MiDaS documentations, it became clear that the design of this surface level of the system would be relatively simple. Additionally, this layer should be able to load individual images instead of video, for demonstration and testing purposes.

4.4 Choosing the Neural Network

Once the design phase of this project had been reached, it was known that a bespoke Neural Network solution was required in order to address the specific task of Monocular Depth Estimation. The first step in this was choosing (based on the research done) what form of Neural network would be appropriate. Before choosing the network, however, it was important to understand some common aspects of the majority of networks under consideration.

4.5 Layer

A Layer is a collection of interconnected neurons that performs specific operations on inputted data. Layers can be considered as the building blocks of neural networks, and are combined in series (or "stacked"). Each layer then passes its information to the next layer. An example of a type of layer discussed in Section 2 is that of the convolutional layer.

4.6 Channels

A standard RGB Image is - as the name suggests - composed of three colours, Red, Green, and Blue - this means that the image has a Red channel, Green channel, and Blue channel. If passed to a convolutional layer with 16 filters, the output of the layer would be 16 feature maps, each with 3 channels, corresponding to the three in the input image.

4.7 ReLU

ReLU, or Rectified Linear Unit is a type of commonly used activation function (or defined output given a set of inputs) in the field of deep learning. It is defined by the function:

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

Figure 4: The function utilised for ReLU

ReLU applies the function to each element of an input tensor. The result of this is that if an element of the input tensor is less than zero, it is replaced by zero. This aids in reducing the "vanishing gradients". Whilst other activation functions exist and are used in appropriate circumstances, such as Sigmoid or Hyperbolic Tangent, ReLU is generally preferred as it trains the Neural Network significantly faster without a penalty to the accuracy of its generalisation[24].

4.8 Max Pooling

A Pooling Layer, or in this particular case, Max Pooling is a technique used to Downsample feature maps by preserving the maximum value of a defined rectangular subregion. This is done by sliding a rectangular "window" over the feature map and taking the maximum value in said window. This is to reduce the dimensionality of a given feature map, whilst still preserving vital features - the aim of this being to prevent overfitting and improve network efficiency.

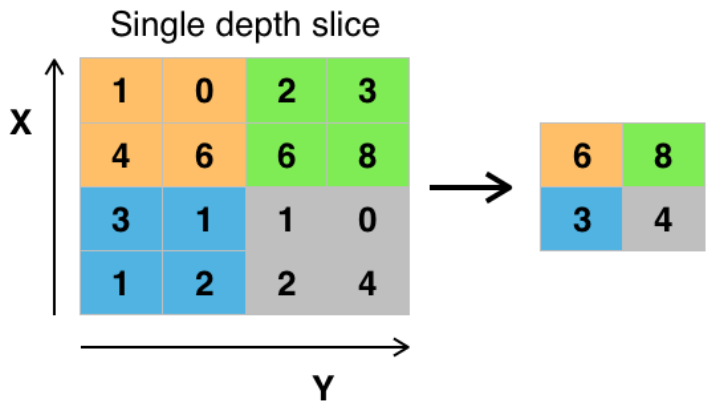


Figure 5: Overview of the Max-Pooling layer (Image credit: Aphex34 - Own work, CC BY-SA 4.0, `https://commons.wikimedia.org/w/index.php?curid=45673581`)

4.9 Fully Connected Layer

Generally the final layer in a series of Max Pooling and ReLU layers, a Fully Connected Layer has connections to all activations in the prior layer in the structure.

4.10 Encoder and Decoder

In some CNNs, an encoder-decoder structure is used. The purpose of this is to strip an image down to an essential low-dimensional representation to extract necessary features, then reconstruct the original data.

4.11 Researched Neural Networks

4.11.1 U-Net

U-Net is a Fully Convolutional Network, with its name coming from the "shape" of its architecture - being visually similar to the letter "U". The network is made up of a contracting path and an expanding path, with these paths being connected by a "bottleneck" layer. The contracting path is responsible for the capture of the input's context and the reduction of the spatial dimensions of the input image. The expanding path is then responsible for producing the final output by increasing the spatial dimensions and recovering the information "lost" during the operations of the contracting path.

The contracting path is made up of a series of convolutional layers, each followed by a ReLU and a Max Pooling layer. The convolutional layers extract features from the input image, with the pooling layers reducing the spatial dimensions of the produced feature maps - allowing for the network to capture a wider context. The contracting path gradually reduces the number of spatial dimensions of the input image (contracting), while increasing the number of channels in feature maps.

Once the Bottleneck Layer is reached - which is typically a convolutional layer, the network begins on the expanding path, which consists of a series of up-convolutional layers, each followed by a concatenation with the respective feature map from the contracting path, and then a convolutional layer. The up-convolutional layers increase the spatial dimensions of the feature maps, whereas the concatenation with the corresponding feature maps allows the network to recover the "lost" spatial information. The final convolutional layer produces the final output.

U-Net utilises "skip-connections" in its architecture, which allows the network to recover the aforementioned lost spatial data. By concatenating the respective features from the contracting path to the expanding path, the network

recovers the data lost during the Max Pooling layers, allowing for more accurate results. However, U-Net's robustness comes at a cost - that being computational power. Given the aim of this project is specifically geared towards common commercial hardware, U-Net's performance makes it undesirable. Furthermore, while it could be made to work for Monocular Depth Estimation, U-Net's primary aim is at Image Segmentation, and therefore may not be operationally ideal for the task at hand.[25]

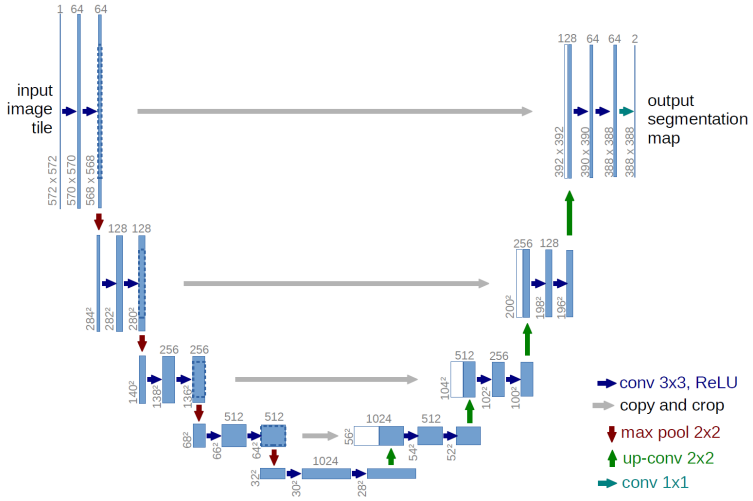
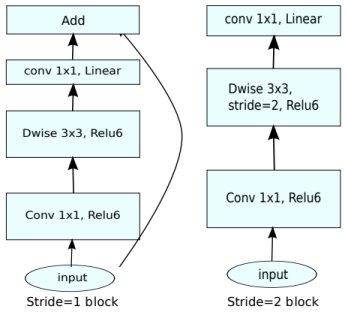


Figure 6: Representation of the U-Net Architecture (Image credit: University of Freiburg[25])

4.11.2 MobileNet

In contrast to U-Net's performance demands, MobileNet is a (family of) Neural Network(s) designed for Mobile and Embedded Devices, and therefore is optimised for limited computational resources.

Traditional CNNs, as discussed in Section 2, are highly parameterized, which increases computational expense - the obvious effect of this is meaning they are not suited for scenarios with exceedingly limited resources, such as Mobile or Embedded Devices. MobileNet's solution to this issue is to replace standard convolutions with "depthwise separable convolutions". These convolutions are constructed from two layers - a depthwise convolution, which applies a filter to each channel - and a pointwise convolution, which applies a convolution of size 1x1 to combine the previous output channels. This reduces computation cost by using fewer parameters, while still allowing for complex representations to be learned.[26]



(d) Mobilenet V2

Figure 7: Representation of the MobileNetvV2 Architecture (Image credit: Sandler, M. et al)[27])

The MobileNet family's greatest asset, by far, is its computational advantage over more demanding networks, and indeed for certain tasks it is a viable competitor. Unfortunately, the obvious caveat is that - should one have the necessary

resources at their disposal, it is therefore potentially outclassed by networks that do not have the limitation of having to run on an embedded system.

4.11.3 ResNet18

ResNet18 - short for "Residual Network 18" is a Convolutional Neural Neural network that is composed of multiple residual blocks that allow the network to learn residuals - or rather, the difference between the output of a layer and its input[28]. The residual block is constructed from two convolutional layers with a kernel size of 3x3, batch normalisation, and ReLU activation functions. The first convolutional layer contains filters that match the number of input channels, with the second convolutional layer then having filters that match the number of output channels. The output of the residual block is obtained by adding that input to the output of the second convolutional layer, which is then followed by an additional ReLU activation.

The ResNet18 Neural Network additionally includes a series of convolutional and max-pooling layers at the start of the network, serving to downsample the input and increase the number of channels. These layers are then followed by a series of residual blocks and an additional pooling layer that takes the average of the last residual block across all spatial dimensions. Finally, a fully connected layer with a softmax activation function is used to produce the output.

Thanks to its reliance on residual blocks, this ResNet18 is able to mitigate the issue of so-called "vanishing gradients"; a common issue affecting deep neural networks and hampering efficient training. By learning residuals - as opposed to raw features - the network learns more effective representations of the data, thus improving the performance[28].

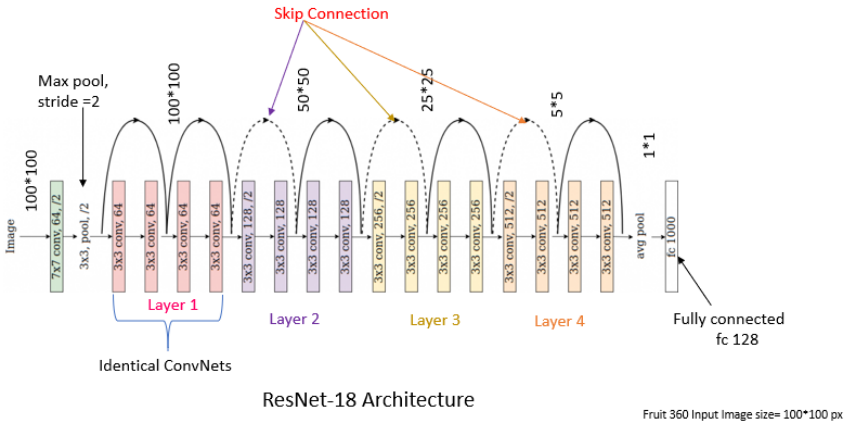


Figure 8: Representation of the ResNet18 Architecture (Image credit: `https://www.pluralsight.com/guides/introduction-to-resnet`))

One of the most attractive features of ResNet18 in the design considerations is its balance between performance and power. Whilst U-Net and MobileNet were determined to not be appropriate for the project, ResNet18 appeared to be a good compromise between the two - still performing exceptionally well on standard hardware, but additionally offers workable performance. Furthermore, given the MiDaS model utilises a modified version of ResNet18, the choice of ResNet18 architecture for the Self-Supervised implementation ensures some level of parity between the two systems.

4.12 Pretext Task

Given the notion of the pretext task, as per the researched literature, it was incredibly important that a pretext task was designed that would not only perform, but would suit the specific needs of the project. While many potential concepts existed, the unique nature of this particular project meant a bespoke task would have to be conceptualised so that said specific needs were fulfilled.

In order to be both within the realms of available computational power, and easy to implement, it was decided that the basic idea of the pretext task is to take the image currently being processed, create a copy of the image - creating a pseudo-stereo pair. From there, the copy could be "augmented" - or have some aspect of it be modified. The aim of this is to increase the diversity of provided training data and allows for the model to learn more useful representations.

4.13 Forward Pass

One of the key processes that will have to be implemented is that of the Forward Pass, which is the process of passing the input image through the Neural Network, thereby producing the resultant depth-map.

4.14 Image Augmentation

Some of the few mentions about the specifics of Pretext Tasks relate to the Augmentation of the Image Data. As Zhang writes, many recent attempts for SSL Monocular Depth Estimation have transformed the task "from a depth prediction task to an image synthesis task"[29]. For a slightly lower-level description, this could involve mapping an image to a 2D plane on a 3D coordinate grid, then warping the image along the Z axis. This would create an effect similar to that of moving the camera to a different position in the scene, allowing for a far greater diversity in terms of training data. This, however, was deemed inappropriate for this particular task - largely due to the additional computational power required in order to achieve this.

However, simply using no augmentation whatsoever would lead to a poor model, and so a compromise had to be made - one where one image in the pseudo-stereo pair would be altered to a significant degree, forcing the model to learn more useful representations, but simultaneously not so much as to adversely affect computational requirements. Therefore, an Augmentation function was conceptualised that would take one of the images, then apply certain alterations to it (rotational position, flips, brightness adjustments, and colour adjustments) at a frequency and severity to be determined by random number generation.

4.15 Loss Function

The Loss for a system such as this is generally calculated after the Forward Pass. For calculating the loss, two main strategies exist - "Mean Absolute Error"(MAE, also known as L1 Error) and "Mean Squared Error" (MSE, also known as L2 Error). MAE is designed to measure the average absolute difference between given values (in this case, the prediction versus the actual value). MAE can prove a more accurate estimate of error as it is less-sensitive to outliers. MSE, comparatively, measures the average squared difference between the given values. For tasks that outliers may prove detrimental to, MSE is more appropriate as it is exceptionally sensitive to them. For this project, MAE was chosen due to the "unsatisfactory" results often produced by MSE[30].

4.16 Optimizer

The purpose of the Optimizer is to take the resultant loss from the Loss step and back-propagate that through the Neural Network, allowing the Network to hone its accuracy over time. In SL models, the Optimizer updates the Neural Net's parameters, which in turn minimises the discrepancy between the predicted depth and the ground-truth depth. In SSL models, the Optimizer's input and target depend upon the designated Pretext Task; in this instance, that means the Optimizer takes the pseudo-stereo pair as the input and target. As per the already cited literature in this paper[18][19][31], the Adam optimizer is generally the most commonly used, and therefore was chosen for this, however its specific parameters would have to be tweaked as per results garnered from testing.

4.17 Test Plan

The overall test plan for the project, due to the nature of its design, was required to be split into two sections - one for the Supervised Model, and one for the I/O layer. Furthermore, the Test Plan would have to be somewhat rigid when compared to other elements of the design, as the implementation may be required to change in terms of its scope.

4.17.1 Supervised Model Test Plan

- Ensure Dataset can be properly loaded, fail appropriately if not
- Ensure Data can be properly augmented
- Ensure training occurs correctly
- Ensure loss is back-propagated correctly
- Ensure model is correctly saved

4.17.2 I/O Layer Test Plan

- Ensure that both Supervised and Self-Supervised models are supported, with any unsupported inputs by the user being redirected back to the initial input statement
- Ensure that both webcam and folder input are supported, with unsupported inputs being redirected back to the initial input statement
- Test both the MiDaS model and custom model, ensuring both have proper behaviour

5 Implementation

This section details how the conceptualised Design, consisting of appropriate Methods and Tools, were implemented, with respect given to their potential strengths and limitations.

5.1 Self-Supervised System

The bulk of the implementation of this project is in the form of the Self-Supervised system. At a high level, this system begins with a Custom Collation function, that obtains the raw KITTI images and applies appropriate transformations to ensure compatibility with the rest of the system, before returning them. With the specified number of Epochs taken into account, these images, being passed through the data loader, are then taken and given to the device (either the CPU or CUDA enabled GPU), and finally, passed through the Network. However, for this project, due to the system used not possessing a CUDA enabled GPU, all work was done on the CPU. From there, a copy of the images is then passed through the Image Augmentation function, with these images then too being passed to the device and passed through the model. From here, the L1 loss is calculated, between the Network's output of the Augmented and Unaugmented Images. Finally, the optimizer's gradient is Zeroed, with the loss then back-propagated through the model via the optimizer. This loop then continues until the Dataset is exhausted, and this is further repeated for the desired number of Epochs, saving several checkpoints each epoch for testing and monitoring purposes, and a final "main" model at the conclusion of each epoch.

The structure of this network is in a single python script, "self-supervised.py". More modularity could have been employed in order to separate the respective classes into different files, however this seemed ultimately unnecessary as the file does not exceed 200 lines of code, and therefore it would have more likely deteriorated code readability.

5.1.1 Data Loader

The very core of the system, without which it would be unable to function, is surprisingly represented in the form of only one line of code. The DataLoader, imported from the torch.utils.data Library, allows for the creation of a "python iterable" dataset. The options for the Data Loader are myriad, however four are utilised in this project: "batch_size", "num_workers", "shuffle", and "collate_fn". The batch_size parameter eponymously represents the number of images to be processed in one step of the training loop; increasing the batch size, therefore, creates a training loop with fewer steps, however this also increases the time taken to complete the dataset. The key benefit of batching in the nature of this project is by reducing the total amount of disk read operations necessary - both improving overall efficiency and reducing the chance of disk damage as a result of intensive writing. For project-specific context, a batch size of 8 was used, thereby reducing the total number of operations by 8 times. By default, the Data Loader uses only a single process, however the num_workers parameter allows for the allocations of separate worker processes in order to

hypothetically increase the speed of training, at the cost of additional CPU resources as a result of the increased number of total processes. For this project, a number of 4 workers was found to be appropriate, not increasing CPU usage beyond operational levels.

The shuffle parameter performs the task befitting of its name, shuffling the dataset, thus creating opportunities for better generalisation when training the dataset.

It should be noted that a reference-able instance of the Data Loader is created, allowing it to be called.

Finally, the collate_fn parameter determines how loading and batches of images is handled. By default, this stacks tensors along a new "batch dimension", forming a batch - but this particular behaviour is not necessarily suited for all types of implementations, and furthermore assumes that the loaded data is already in tensor form. In this project's implementation, a Custom Collate Function was used in order to be able to account for the specific requirements of this project.

5.1.2 Collation

One of the first technical problems encountered during the project implementation was that the default Collate Function of the Data Loader was unable to load the raw PNG images forming the KITTI dataset. Therefore a bespoke collation function for the specific needs of this project had to be created.

Extraction from Batch This function first extracts the images from the batch, sent to the function from the Data Loader.

Image Resizing From here, the images are all resized - this is due to the fact that the elements in the tensor that will soon be created need to be of equal size. The size chosen for the images was determined by a rudimentary python script that iterated over the images in the KITTI dataset and obtained the mean average size (as KITTI images are of different sizes within a small range), which came to 1240x374.

Convert to Tensor Each image in the batch was then converted from a PNG image to a four-dimensional Tensor, with shape:

$$[B, C, H, W] \tag{2}$$

Figure 9: Tensor representation of the Image, where B = Batch Size, C = Channels, H = Height, W = Width

Output Finally, these image tensors are "stacked" along the Batch dimension with the torch library function, torch.stack[32], and returned to the Data Loader. While Augmentation of Data could have been performed in the Collation Function, this was saved for a separate function due to the workflow of this system. In particular, augmenting all source images defeats the particular pretext task-goal that this system attempts to accomplish. Were the function to augment all base images, then the loss function, which is intended to take a base image and an augmented image as its inputs, would effectively be calculating based on an identical input and target.

5.1.3 Neural Network

Contained within the Network class, this facilitates the Neural Network that was used in the project, with the Network Architecture itself being defined in the init method. As alluded to in previous sections, the Neural Network employed in this project is that of a ResNet-18 Convolutional Neural Network, due to its balance between power and performance. One key aspect to note is that this implementation of ResNet-18 features the ability to utilise Pre-trained weights as supplied by PyTorch. Through using these pre-trained weights, this gives the model a "head-start" with the learning of its generalised representations, whilst also alleviating some computational time. However, this Pre-trained ResNet18 architecture only serves as the encoder of the image.

The decoder of the image consists of a five-layer architecture, beginning with one Convolutional layer, followed by a ReLU layer - alternating for the remaining three layers. The first Convolutional Layer is of size 1x1, the second of size 3x3, and the third again of 1x1. The convolutional layers downscales the channels present in the encoder's output feature map, eventually returning an image tensor with a single channel.

The forward method, defined in the network class, defines the forward pass through the network. It defines the specific operations performed when the model receives an input, and how this input is processed to eventually produce an output.

In this implementation, the input, defined by the variable x, is passed through operational layers of the encoder, being Convolutional Layers, Batch Normalisation Layers, ReLU layers, and Maxpool layers. From there, the encoder's output is then fed through the decoder. From there, the squeeze PyTorch Library function[33] is applied to x to ensure it is a tensor of proper shape, after which x is finally returned.

The ultimate aim of this implementation is to extract high level features from the image, which can then be used to form useful predictions in regards to the depth of the image.

5.1.4 Augmentation Function

The augmentation of the images is handled by the augment_data function. This function applies rudimentary transformative operations to the inputted images, producing a varied selection of altered images, thereby increasing the diversity of training data. In order to perform these transformations, the torchvision.transforms.functional module is used, in conjunction with Python's random library.

An image passed to the function has the potential to be:

- Flipped directly horizontally (50% chance)
- Rotated by an angle between 345 and 15 degrees (generated randomly)
- Have the image brightness adjusted from 0.8 to 1.2, with 0 being a black image, 1 being the original image, and 2 being a brightness increase by scale factor of 2[34].
- Have the image contrast adjusted from 0.8 to 1.2, with 0 being a solid grey image, 1 being the original image, and 2 being a brightness increase by scale factor of 2[35].
- Have the image saturation adjusted from 0.8 to 1.2, with 0 being a black and white image, 1 being the original image, and 2 being a saturation increase by scale factor of 2[36].
- Have the image hue adjusted from -0.1 to 0.1, with -0.5 and 0.5 being a total hue reversal of the image, and 0 being no shift whatsoever[37].

As a result of the deliberate choice to not include the augmentation in the Collation function, the call for this function has to be done in such a way as to where it iterates through the individual image in a given image batch. This manifests such that per step, there are B number of augmentation operations, where B = batch size. After this, the augmented images are then evaluated, before being compared to the original images by the Loss function

The effect of this augmentation is to simulate real-world scenarios wherein aspects such as light level or colour composition may not be like that of the training data; thereby forcing the model to search for more abstract structures within the data.

5.1.5 Loss Function

Based on the review of literature suggested by the Design section of the paper, the chosen Loss calculation method was that of Mean Absolute Error, or L1Loss (which is what it will be referred to henceforth). L1Loss is defined by the following equation: From there, an instance of the loss function is created, from which the evaluation of both sets of

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n} \tag{3}$$

Figure 10: The equation defining Mean Absolute Error (or L1Loss)

images are passed to it. The loss is calculated, and then this is back-propagated through the model.

This is done through three steps:

1. *optimizer.zero_grad()* - The gradient of the optimizer is zeroed - this is done before back-propagation as without first zeroing the gradient, it would accumulate and begin exponentially distorting the results.
2. *loss.backward()* - The loss is then back-propagated through the model; this computes the model parameters with respect to the loss. The result of this is the gradient required for the weight update
3. *optimizer.step()* - Finally, the optimizer's weights are updated based on the gradient that has been computed. The optimizer is update with the given algorithm used for the model.

5.1.6 Optimizer

The chosen optimizer, Adam, is highly regarded, being used in the vast majority of monocular depth estimation application[18][19][31], and therefore was the obvious choice. Furthermore, Adam is deemed as a "computationally efficient" method with "little memory requirements"[38], making it ideal for the design limitations of this particular project. In this particular implementation of Adam, several hyperparameters were specified - a hyperparameter being a parameter that is defined by the user and generally unaltered as a result of training, as opposed to model parameters, such as weights, that are altered[39].

The hyperparameters used in the Adam Optimizer are:

- *Learning Rate* - Also referred to as "α" or "*lr*", this governs the step size at each parameter update. Smaller learning rates mean slower training, but more precise model convergence - higher rates can lead to faster training but can effectively train "past" the solution, leading to a less-useful model.
- *Betas* - Also referred to as "β_1, β_2", refer to the coefficients used for computing the moving averages for the gradient (β_1), and its square (β_2). These are generally set to values approaching 1, with β_2 being closer.
- *Epsilon* - Also referred to as "ϵ" or "*eps*", is a minute numerical constant added to the denominator, this is in order to prevent accidental division by zero for low denominator values.
- *Weight Decay* - Also referred to as "λ" or "*L2 Regularisation*" is used to prevent overfitting by causing a proportional decay to the learned rates over time.

5.1.7 Saving

In order to access and use the saved model, a PyTorch model file (or a .pt file) was saved after every epoch, this could then be loaded by the I/O layer. As well as this, checkpoints were also added at arbitrary training steps (e.g. step 225, step 749, etc) ad hoc during testing - these files were referred to as *pseudo checkpoints*.

5.2 I/O Layer & Supervised Model

In order to handle the I/O of the system, a small script was written - additionally, it was made capable of loading a pretrained, supervised model for testing and reference purposes.

5.2.1 Loading the Model

The system initially offers the user a choice to load either the MiDaS pretrained model, or a custom model. Should the user choose MiDaS, the model is loaded via the torch.hub module and downloaded locally, should the user not already have the model downloaded. If the user should instead choose to load a custom model, they are asked to specify the file-path to a .pt file. In this case, in order to successfully load the .pt file, the program also imports the network from the self-supervised file. This could be deemed a limitation of this particular implementation, however this hard-coded approach was the most efficient way of retrieving and utilising the desired network architecture.

5.2.2 Input Transforms

The model then loads the MiDaS model's stock "small transform". Initially, an alternate transform was intended, however initial tests of the program showed that the MiDaS transforms also worked for the Self-Supervised Model.

5.2.3 Real-time Depth Estimation

With the model - either SL or SSL - already loaded, the real-time implementation was relatively simple, and makes use of the open-cv library. After specifying the video-capture device (which uses the system default in this implementation), the current frame is obtained, and designated as the image to have depth estimation performed on - additionally, Open-CV's BGR colour-scheme is applied. From there, it is passed through the model, down-scaled as appropriate for output, and finally output. The output consists of matplotlib library plots for each individual frame, as well as a numpy output of the depth values of each pixel to the console.

5.2.4 Single Image Depth Estimation

Depth estimation for a single image was, ostensibly, an almost identical implementation. The change came where, instead of obtaining the user's video capture device and then using the individual frames from said device as the image to be processed, it instead used a singular image from a user-specified folder.

6 Testing

6.1 Self-Supervised Model Testing

6.1.1 Ensure Dataset can be properly loaded, fail appropriately if not

Result The program searches for the Dataset in the folder designated as the Data Root, if no dataset is detected, the dataset is downloaded. If no network connection is detected, the program exits with appropriate feedback

Actions Taken N/A - Working as intended

6.1.2 Ensure Data can be properly augmented

Result The some of the augmentations were applied but the angle was seemingly not being adjusted

Actions Taken Angle mistakenly set to -0.15 and 0.15 - human error due to other augmentations being floats with values less than 0. Adjusted to proper values of -15 and 15 respectively.

6.1.3 Ensure training occurs correctly

Result As the batch number is 8, but the dataset is comprised of 7481 images, this was in imperfect division resulting in a total of 935 full batches and 1 batch of 1 image - this was causing crashing upon the end of an epoch.

Actions Taken Training loop structure was adjusted to ensure proper accounting for the uneven batch size.

6.1.4 Ensure loss is back-propagated correctly

Result Loss appeared to be progressing at unexpected rates

Actions Taken The designated target and output in the loss function were set the incorrect way round, this was adjusted.

6.1.5 Ensure model is correctly saved

Result See Test #3.

Actions Taken See Test #3.

6.2 I/O Layer & Supervised Model

6.2.1 Ensure that both Supervised and Self-Supervised models are supported, with any unsupported inputs by the user being redirected back to the initial input statement

Result Due to the "custom" model being selected by an "else" statement, any unsupported input causes the user to be prompted for the SSL model path

Actions Taken User input defined more rigorously

6.2.2 Ensure that both webcam and folder input are supported, with unsupported inputs being redirected back to the initial input statement

Result Unsupported inputs are not handled and cause the program to crash.

Actions Taken Input handling implemented to ensure incorrect inputs cause re-prompting of the statement.

6.2.3 Test both the MiDaS model and custom model, ensuring both have proper behaviour

Result Self-Supervised Model appears pixelated, with improper dimensions

Actions Taken An investigation yielded that this was due to improper structuring of the Self-Supervised Model's network class - specifically, that the resulting tensor x had an additional dimension of value "1", creating a malformed tensor. Therefore, a torch.squeeze function was added to remove this.

7 Results & Evaluation

This section presents an Overview of the Quantitative and Qualitative results collected, with a description and thorough evaluation given to each. This is followed by a final summary of the Evaluation as a whole.

7.1 Overview of Results

As a result of the research behind and implementation of this project, both quantitative and qualitative results were produced. These results consist of loss outputs collected from the console output of the model during training - being the quantitative component; and the depth maps being produced with a control input image; being the qualitative component.

7.2 Quantitative Results - Depth Loss

Describing the Results Collected through computing the L1Loss between the unaugmented image and augmented image, the Depth Loss represents the current disparity in the model's prediction. When viewing the raw data as plotted

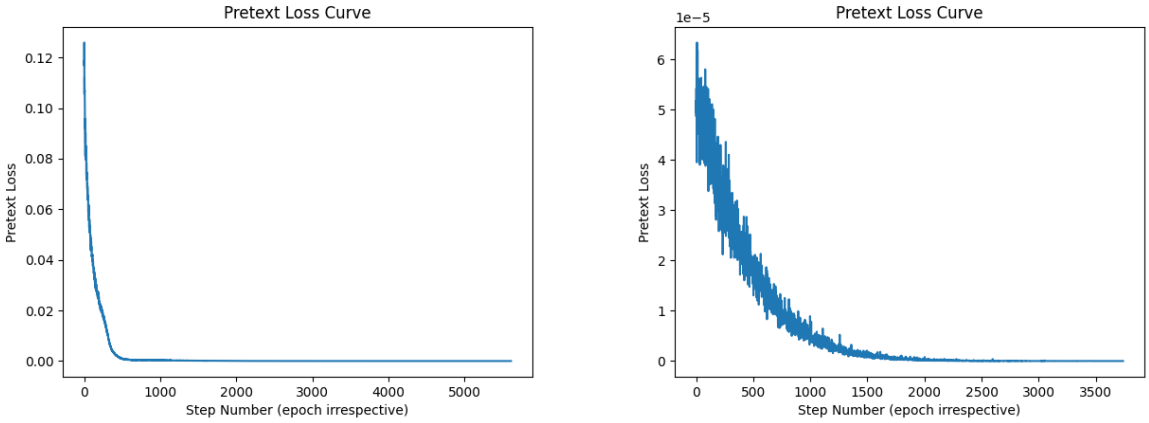


Figure 11: The loss result plotted against step number. Figure 11.1 represents the full breadth of training, Figure 11.2 represents only epochs 4-6

on a curve, the phenomenon known as Convergence is visually illustrated - wherein the loss produced by the model exponentially decreases until plateauing within a small range of its "final value". Once convergence is reached, training the model will not improve it, and may ultimately cause a detriment.

The results detail the convergence of the model over a period of 6 Epochs, or 5,616 total steps. Each step took approximately 15 seconds to complete - giving a total training time of 84,240 seconds or 23 hours and 12 minutes.

An unexpected qualitative result that was also gathered was that of CPU usage (as stated in the Implementation section, all work was performed on the CPU in lieu of a CUDA-enabled GPU). While it was expected that CPU usage would begin high, but exponentially decrease to correspond with the decrease in loss - instead, CPU remained essentially constant. Prior to runtime, CPU usage was at approximately 5%. At runtime, and immediately leading into Epoch 1, CPU usage had repeated spikes to values in excess of 90% at runtime, however after the training loop initiates (indicated by debug Print statements within the code), the CPU usage returns to a base of 60% in between. By Epoch 6, CPU usage was at a constant average of that same 60%, not a value closer to the initial 5%.

What do these findings suggest? Examining the loss, it becomes clear that despite the fact that L1Loss is reputedly less affected by outliers, anomalous data points can - and do - still occur, and are visible to the naked eye when plotted. Furthermore, the results suggest that despite the disparity between the Loss function's target and output decreasing, the computational complexity of the calculation remains identical.

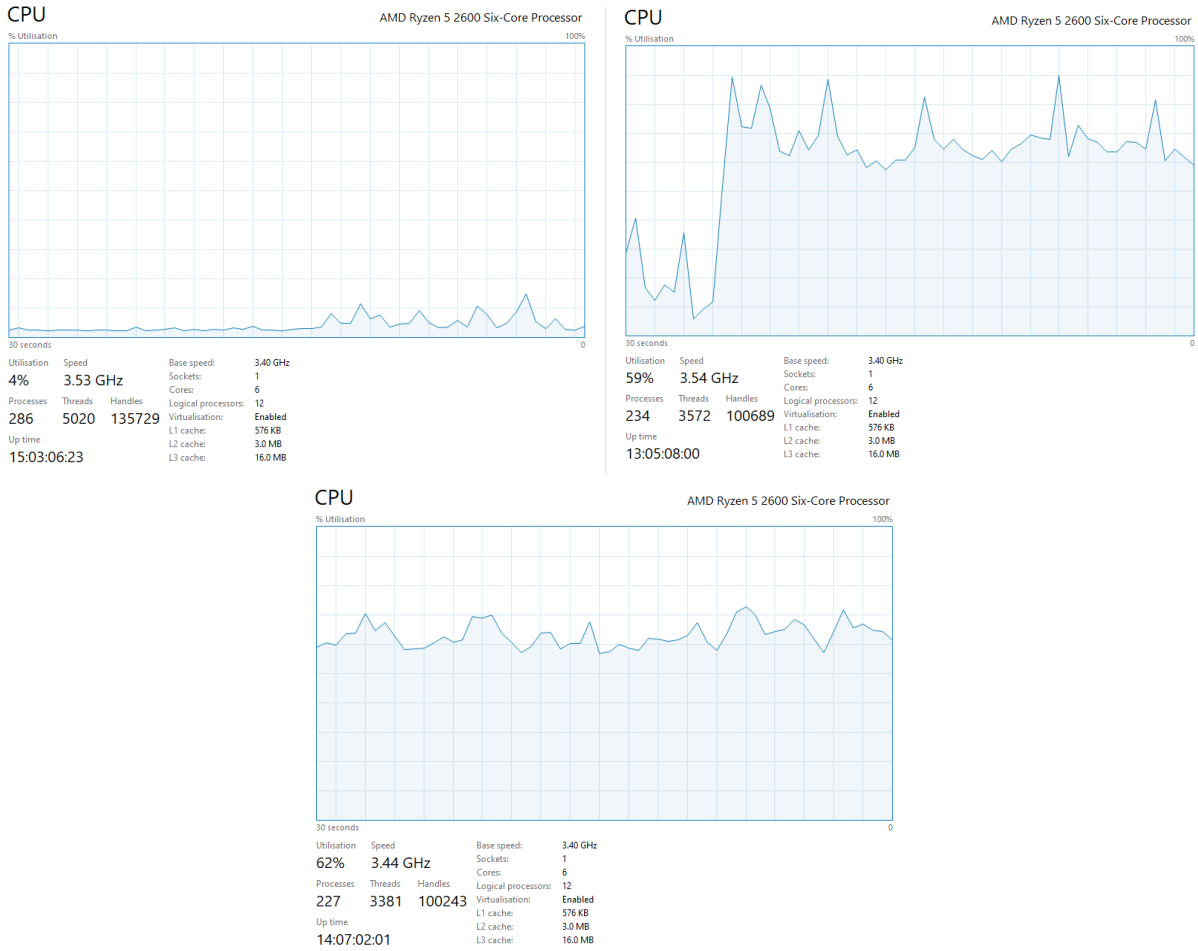


Figure 12: The CPU usage. Figure 12.1 represents idle CPU usage, figure 12.2 represents CPU usage at runtime, figure 12.3 represents CPU usage at Epoch 6

Criticism of these findings While Outliers have been highlighted, whether or not they simply exist is irrelevant without considering their appearance - this notion being composed of several factors. The first is that of the fact that anomalous data can occur at random. While this can eliminate some of the error, it perhaps addresses the symptom, rather than the cause. Indeed, this could be pinned on the Loss Function itself - in the context of its Target and input data.

While commonly used in Depth Estimation tasks, as this project's background research has suggested, it is not a Loss Function explicitly designed for it - and certainly not for Depth Estimation in a Self-Supervised context - wherein rules that are held as standard for SL models are no longer so concrete. Furthermore, this also may shed light on the second core finding from the Quantitative Data - that of the CPU usage. Despite the fact that the computational complexity may seem as if it should be simpler, this is from a human perspective (ironically, based on the same type of generalised "common-sense" thinking that SSL is attempting to imbue Artificial Intelligence with). Mean Absolute Error exists in the abstract context of pure mathematical statistics, and while it has been adapted into a machine learning framework, it is not inherently suited for machine learning. Due to this, it features no analysis for computational efficiency[40]. As a result, a custom loss function, specifically developed for the task of this project, may have performed better.

Additionally, it should be noted that the data used to plot the graph is the console output, which produces a value in the format of twelve-decimal places multiplied by ten to the power of whatever the calculated exponent is for a given loss calculation (for example, a value such as 3.050216101110e-02). As a result of this, if the loss value exceeds the bounds of this format, it is simply rendered as 0. This is a significant limitation in the production of this data as it means that certain data points are effectively rendered non-existent.

Defence of these findings Despite the valid criticism levied against the above findings, it should also be argued that the exponentially decreasing loss, eventually culminating in convergence, is the behaviour expected in a model such as this[41], and it would be far more alarming if convergence was not reached.

Furthermore, whilst the idea of a bespoke loss function is intriguing, it was not strictly within the bounds of this research and indeed, the knowledge of possibly desiring or requiring one could not have been known without first extensively utilising Mean Absolute Error.

Finally, while the usage of a rigid decimal form may have potentially adjusted plotted results, it must be said that as the values for these results are both so small, and so numerous, the additional data tending towards zero would have made little difference, especially as convergence is reached either way.

7.3 Qualitative Results - Depth Loss

Describing the Results Obtained through the system's I/O layer, the Qualitative Results consist of BGR Depth-Maps of a given test image (see below figures), produced using the trained Self-Supervised Learning model. Additionally, Real-Time Depth Estimation was utilised ad hoc in order to examine the model's effectiveness for this real-world application. It should be noted that, in the BGR colour-scheme, the darker blue the colour, the further away that particular pixel is purported to be, whereas the lighter green the pixel, the closer it is purported to be.

When viewing the appropriate images, while rudimentary depth estimation has occurred, the result is not to the quality of the MiDaS state-of-the-art model. Basic shapes can be identified, however specific features are not visible. When viewed in Real-time, the effect of the depth estimation is clearer, and while shape mapping is still not of industry-standard quality, the model is aware of a change in depth, as indicated by the vivid colour change in the depth map.

Additionally, during the production and refinement of the model, overfitting issues were encountered, producing unusable depth maps.

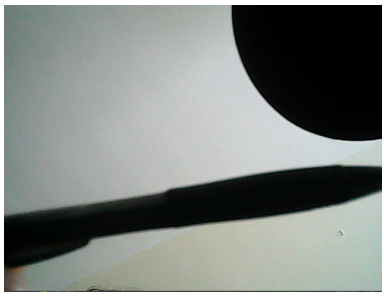


Figure 13: The ad hoc image used to test depth estimation

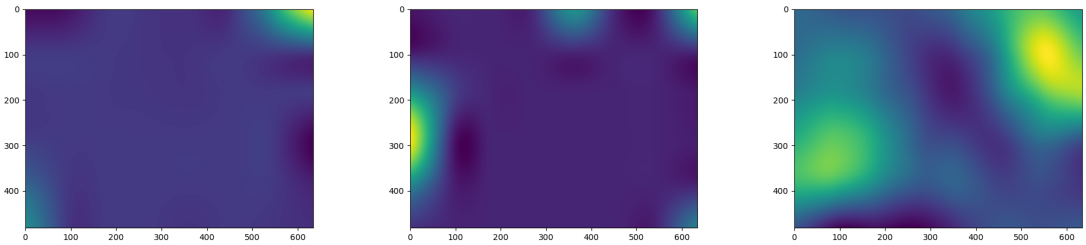


Figure 14: Depth Estimation for Epochs 1-3

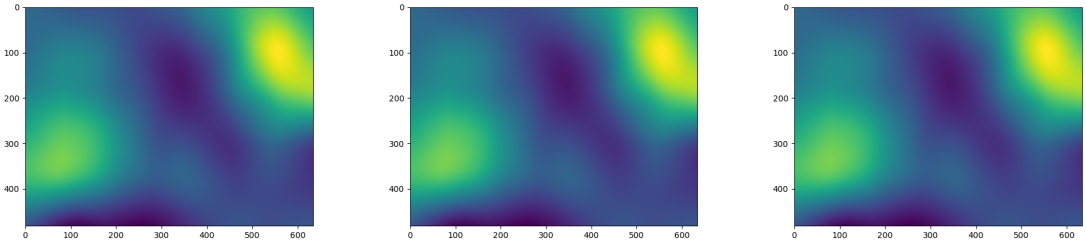


Figure 15: Depth Estimation for Epoch 4-6

What do these findings suggest? These depth maps suggest that whilst Self-Supervised Monocular Depth Estimation is possible on limited hardware, the implementation of it is not necessarily straight forward. Unexpectedly, Real-time Depth Estimation seems to be in more of an operational condition than that of depth estimation for a still image. While this bodes well for the application of the model in the intended hypothetical context of Autonomous Vehicles, it would be expected that a network trained on still images would be far more capable of predicting depth in still images. Additionally, the visual evidence would appear to indicate convergence around the conclusion of Epoch 3, due to the similarities in results for Epochs 4, 5, and 6

Criticism of these findings With Depth Maps being able to be produced with both static images and in real-time by a model trained via SSL, it is clear that implementation and utilisation of the paradigm on limited hardware is possible. However, the quality and usability of these results must be brought into question, as well as the nature of the network by which they were obtained.

First and foremost, the results do not have parity with industry standard models, both in terms of those highest ranked Self-Supervised Models and Supervised Models. Due to the inability to make out specific aspects of shapes during depth estimation, this model would not be appropriate to use on Downstream Tasks, especially in the context of an Autonomous Vehicle where a mistake in depth estimation could result in serious danger.

Secondly, the observation that the Real-time depth is in a more usable state than that of the static image depth is somewhat questionable. While the frame-by-frame plotting of the depth data does indeed suggest this, the lack of object clarity as stated prior means that some of the observed "successful" depth estimation performed in real-time may just be somewhat random depth data, produced by an improperly trained model, misconstrued by the human observer as correlating to the motion currently being captured by the video device. Furthermore, the video device used was a standard webcam of resolution 640x480, somewhat different to that of the images captured by the KITTI team's camera. The impact this could have on the results is unknown, and could only be effectively determined through obtaining an identical camera - which is far beyond the means of this project.

Thirdly, the exact nature of the Neural Network architecture used must be called into question, specifically the use of Pre-trained Resnet18 weights in training. With this, the nature of the project itself could be called into question, with the assistance of pre-trained weights muddying the waters as to what exactly "self-supervised learning" can be defined as.

Finally, despite the visual indication of convergence, further training may have altered these results, yielding change in the next epoch - thus pushing back convergence.

Defence of these findings To begin, it is true that these findings are not in parity with industry standard solutions, but as the introduction states, that is not the intention of this project - instead, the produced results should serve as a benchmark for the capabilities of the technology. To address the criticism factually, the fact that a novice with no knowledge of machine learning whatsoever was able to construct a model that is remotely capable of Self-Supervised Monocular Depth Estimation whatsoever, should speak to the sheer promise of the paradigm. While it is true that the models produced here would be far-from-appropriate for real-world usage, there is no possibility of that whatsoever, and so that aspect of the criticism is made moot.

Secondly, while it is a possibility that the real-time data is inaccurate, observational testing provides significant correlation between movements captured by the camera and respondent depth-map. Indeed, the depth map is designated as qualitative research as, although it would be theoretically possible to measure it quantitatively, that would involve taking the depth prediction of every single pixel and comparing it to the actual ground truth of the frames captured on camera, which would have to be known in advance. At that point, the task is no longer a self-supervised one as you have obtained ground-truth depths (as well as being too far out of the scope of this research).

Thirdly, while the default weights of the Resnet18 network were used, they can be easily justified as they increase the speed at which convergence can occur under the given hyperparameter. When considering this alongside the colossal training time for a mere 6 epochs, the time saved through using the pretrained weights is suddenly appreciated. Additionally, the quantitative loss results would suggest that regardless of the initial weights, the model was nowhere near trained enough to achieve reasonable convergence, given the loss at Step 1, Epoch 1 (1.185677573085e-01) is approximately ten times larger than the loss at Step 936, Epoch 6 (1.345741157399e-10).

Finally, whilst additional training may have yielded a stronger model and more insight on the point of convergence, this is merely an assumption. Due to the sheer amount of time taken to train for six epochs alone, the feasibility of training for longer on the current system is questionable.

7.4 Evaluative Summary

While the obtained results are not without flaw, they clearly suggest valuable information (both quantitative and qualitative) has been obtained. It can be said that limitations in the results may not rest exclusively on the methods in which they were obtained, and indeed the flaws in the project's planning should too be considered. While the project remained on track for the duration, the limitations of the Waterfall model have indeed made themselves known, particularly in the design phase. Had a more optimal design methodology been adopted, implementation could have begun far earlier, simultaneously with the design, and therefore any potential limitations that would manifest in data would have been detected at an earlier date and corrected.

Furthermore, it must be said that due to the almost entirely academic nature of Self-Supervised Learning in its current state, and my own lack of knowledge at the project's inception, gaining the necessary foundation skills within the temporal scope of the project also presented a significant challenge.

Ultimately, a satisfactory range of data was collected that both support and challenge elements of the project's hypothesis.

8 Conclusions

This section presents the conclusions drawn from the project in its finalised state. This consists of a restatement of the project's goal, ruminating on how this goal may have shifted throughout the course of the project, followed by novel proposals for future research, and the final remarks on the project itself.

8.1 Initial Goal

This project's core objectives were to explore a Self-Supervised Learning-based solution to Monocular Depth Estimation - with the inspiration of it being one of the vital elements in use for Autonomous Vehicles. More specifically, this project aimed to produce results that would serve as a "proof-of-concept", taking an appropriate, unlabelled dataset and producing a self-supervised model, that would then be trained on said dataset.

8.2 Achieving the Goal

Categorically, this goal was met. Through use of the PyTorch framework, and the KITTI autonomous vehicle benchmark dataset, a system was created that used the ResNet18 Network and appropriate image augmentation. In turn, this system produced a model that could be used to predict depth, both in real-time and with static images. Though, as discussed in the Evaluation, attention must be drawn to the exact quality of these results. Depending on one's requirements for a produced system to be considered a viable "proof-of-concept", this project could be construed as having fallen short of the goal. Regardless of this subjectivity, however, the fact is that a model was indeed produced that, while making use of hardware that is exceptionally below industry standards, was still able to achieve rudimentary depth estimation.

8.3 Changes in the Goal

At a very early stage in the project, the goal was far more ambitious - that being the desire to actually produce a model that could rival industry standard models in the established Supervised paradigm, such as MiDaS. With the advent of the initial background reading, though, it quickly became clear that this was simply not possible. MiDaS (and other contemporary Supervised Models), are trained for hundreds of hours, on incredibly optimised hardware[18]. Leading Self-Supervised models are much the same, but have the added challenge of being on the bleeding edge of Machine Learning. This caused documentation on related methodologies to be somewhat obfuscated behind academic jargon and machine learning lingua franca, all of which assumed a certain level of base understanding of machine learning at the outset of the project. As a result, it was realised that this goal would never be met, and so had to be adjusted. This was not necessarily to the project's overall detriment, however, and indeed may have given it a more unique character. Knowing there was no possibility of using advanced hardware to fulfil the intense training that is common in-industry, it was decided instead that the poor hardware that was at the project's disposal should instead be leveraged to create a somewhat different premise - Can Self-Supervised Learning be done to any kind of rudimentary degree on commonly available hardware?

This adjustment in goal ultimately manifested in the form of attempting to make the implementation as computationally efficient as possible, with design decisions such as the basic pretrained network being used to get an effective "head-start" on the already monumental training time. Alongside this was the utilising of the simple augmentation-based pretext task to allow for more general representations to still be learned, but all the while keeping careful control over the amount of resources used.

8.4 Future Research

With these limits in mind, there is potential for future research. The first area, clearly indicated by this research, is the exploration into a bespoke Loss function specifically designed for Self-Supervised Learning - and possibly even tailored to specific hardware requirements. With this, computational needs could be dynamically allocated as per the complexity of an individual loss calculation, as opposed to the current "one-size-fits-all" method. In practice, this could leverage representations already learned to make estimates about the ultimate output of the loss function (this could additionally be used as a secondary training layer).

Another budding area of research is that of Transfer Learning, and in particular its current competition with Self-Supervised Learning. Yang et al concluded in a recent paper that Transfer Learning and Self-Supervised Learning are very much equal in some aspects, and yet for other specific tasks, one clearly exceeds the other. This competition and overlap is ripe for additional exploration.[42]

Finally, the area that most interests me is in the nature of the way Self-Supervised Learning is communicated. In 2023, Artificial Intelligence was thrust into the public consciousness in a way that has arguably not occurred since Deep Blue defeated World Chess Champion Garry Kasparov in 1997. This is in no small part thanks to communities such as HuggingFace, and companies such as StabilityAI and OpenAI making efforts to "democratise"[43] the field, and place development firmly in the hands of people - with the results speaking for themselves. Self-Supervised Learning, however, remains firmly within the grasp of Academia (likely due to its young age compared to other paradigms). I firmly believe that, should SSL too be "democratised", the rate at which high-quality, easy-to-understand information is both communicated and shared, and the rate at which high-quality applications of SSL are implemented, will both grow at an unprecedented rate. If I were to pursue the field in future, I would want the majority of my efforts to be firmly in this effort of democratisation.

8.5 Closing Remarks

While this project has inherent limitations because of all of the factors outlined thus far, these factors unequivocally became the project's strength. More importantly, however, and as was alluded to in the Evaluation Summary, the lens of these limitations highlights in clear detail the true strengths of Self-Supervised Learning as a paradigm. In the span of four months, with no initial experience or background in Machine Learning, I was able to create a system that explored Monocular Depth Estimation utilising Self-Supervised-Learning, all while in a Limited-Hardware environment. Because of this, regardless of how rudimentary the achieved depth-estimation was, this project serves as a thorough exploration and proof-of-concept for the paradigm, and a clear indication of the potential of Self-Supervised Learning as the dominant paradigm of the future.

References

[1] The Associated Press. "Nearly 400 car crashes in 11 months involved automated tech, companies tell regulators", The Associated Press, https://www.npr.org/2022/06/15/1105252793/nearly-400-car-crashes-in-11-months-involved-automated-tech-companies-tell-regul (accessed Feb. 17, 2023).

[2] International Conference on Computer Vision. "2021 Workshop: Self-supervised learning for Next-Generation Industry-level Autonomous Driving, ICCV, https://sslad2021.github.io/ (accessed feb. 20, 2023).

[3] Forbes Council. "13 Tech Leaders Share Their Preferred Software Development Methodologies And Strategies", Forbes, https://www.forbes.com/sites/forbestechcouncil/2022/08/15/13-tech-leaders-share-their-preferred-software-development-methodologies-and-strategies/ (accessed April. 17, 2023.

[4] R. Sherman. "Waterfall Methodology - an overview", Science Direct, https://www.sciencedirect.com/topics/computer-science/waterfall-methodology (accessed April. 5, 2023), 2015.

[5] A. Rhodius and R. C. Seaton (Ed & Translation). *Argonautica*. Harvard University Press, Cambridge MA, 1912.

[6] A. M. Turing. Computing machinery and intelligence. *Mind*, LIX:433–460, 1950.

[7] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5:115–133, 12 1943.

[8] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[9] M. O'Brien. Explainer: What is ChatGPT and why are schools blocking it?, The Associated Press, https://apnews.com/article/what-is-chat-gpt-ac4967a4fb41fda31c4d27f015e32660 (accessed April. 30, 2023).

[10] L. Deng. Deep learning: Methods and applications. *Foundations and Trends® in Signal Processing*, 7:197–387, 2014.

[11] Y. LeCun. "Self-supervised learning: The dark matter of intelligence", Facebook, https://ai.facebook.com/blog/self-supervised-learning-the-dark-matter-of-intelligence (accessed Feb. 6, 2023).

[12] A. Masoumian, H. A. Rashwan, J. Cristiano, M. S. Asif, and D. Puig. Monocular depth estimation using deep learning: A review. *Sensors*, 22:5353, 07 2022.

[13] Y. Furukawa and J. Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32:1362–1376, 08 2010.

[14] Toyota Research Institute. "Monocular Depth in the Real World", Toyota, https://medium.com/toyotaresearch/monocular-depth-in-the-real-world-99c2b287df34 (accessed Feb. 22, 2023).

[15] E. Newton. You Need to Know the Pros and Cons of Self-Supervised Learning, ITchronicles, https://itchronicles.com/artificial-intelligence/you-need-to-know-the-pros-and-cons-of-self-supervised-learning/ (accessed April. 6, 2023).

[16] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521:436–444, 05 2015.

[17] R. O'Connor. "PyTorch vs TensorFlow in 2023", AssemblyAI, https://www.assemblyai.com/blog/pytorch-vs-tensorflow-in-2023/ (accessed May. 4, 2023).

[18] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:1623–1637, 03 2022.

[19] C. Godard, O. Aodha, M. Firman, and G. Brostow. Digging into self-supervised monocular depth estimation. *International Conference on Computer Vision*, 08 2019.

[20] "Papers with Code - Datasets", Papers with Code, https://paperswithcode.com/datasets?task=monocular-depth-estimation (accessed May. 14, 2023).

[21] C. Voskoglou. "What is the best programming language for Machine Learning?", Towards Data Science, https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7 (accessed April. 15, 2023).

[22] L. Qiao. "PyTorch builds the future of AI and machine learning at Facebook", Facebook, https://ai.facebook.com/blog/pytorch-builds-the-future-of-ai-and-machine-learning-at-facebook/ (accessed April. 18, 2023).

[23] Torch hub, Torch v2.0, PyTorch Foundation, https://pytorch.org/docs/stable/hub.html.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Communications of the ACM*, 60:84–90, 05 2012.

[25] University of Freiburg . "U-Net: Convolutional Networks for Biomedical Image Segmentation, University of Freiburg, https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/ (accessed April. 17, 2023).

[26] A. Pujara. "Image Classification with MobileNet", built in, https://builtin.com/machine-learning/mobilenet (accessed April. 4, 2023).

[27] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *Invernational Conference on Computer Vision*, 12 2016.

[29] Y. Zhang, M. Gong, J. Li, M. Zhang, F. Jiang, and H. Zhao. Self-supervised monocular depth estimation with multiscale perception. *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, 31:3251–3266, 2022.

[30] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao. Deep ordinal regression network for monocular depth estimation. *International Conference on Computer Vision and Pattern Recognition*, 06 2018.

[31] M. Fonder, D. Ernst, and M. Droogenbroeck. M4depth: Monocular depth estimation for autonomous vehicles in unseen environments. *Preprint*, 07 2021.

[32] Torch stack, Torch v2.0, PyTorch Foundation, `https://pytorch.org/docs/stable/generated/torch.stack.html`.

[33] Torch squeeze, Torch v2.0, PyTorch Foundation, `https://pytorch.org/docs/stable/generated/torch.squeeze.html`.

[34] Torchvision adjust_brightness, Torchvision v0.15, PyTorch Foundation, `https://pytorch.org/vision/main/generated/torchvision.transforms.functional.adjust_brightness.html`.

[35] Torchvision adjust_contrast, Torchvision v0.15, PyTorch Foundation, `https://pytorch.org/vision/main/generated/torchvision.transforms.functional.adjust_contrast.html`.

[36] Torchvision adjust_saturation, Torchvision v0.15, PyTorch Foundation, `https://pytorch.org/vision/main/generated/torchvision.transforms.functional.adjust_saturation.html`.

[37] Torchvision adjust_hue, Torchvision v0.15, PyTorch Foundation, `https://pytorch.org/vision/main/generated/torchvision.transforms.functional.adjust_hue.html`.

[38] D. Kingma and J. Lei Ba. Adam: A method for stochastic optimization. *International Conference for Learning Representations*, 05 2015.

[39] L. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, 11 2020.

[40] Torch adam, Torch v2.0, PyTorch Foundation, `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`.

[41] Google Developers. "Machine Learning Glossary", Google, `https://developers.google.com/machine-learning/glossary#convergence` accessed May. 14, 2023, 07 2022.

[42] X. Yang, X. He, Y. Liang, Y. Yang, S. Zhang, and P. Xie. Transfer Learning or Self-supervised Learning? A Tale of Two Pretraining Paradigms. *Preprint*, 06 2020.

[43] E. Seger. "What do we mean when we talk about “AI democratisation”?", Centre for the Governance of AI, `https://www.governance.ai/post/what-do-we-mean-when-we-talk-about-ai-democratisation` (accessed May. 7, 2023).