Diese Arbeit wurde vorgelegt am
Lehr- und Forschungsgebiet Theorie der hybriden Systeme

Datenwissenschaftlicher Ansatz zur Lösung von Okklusion in AR

Data Scientific Approach to Solve Occlusion in AR

Masterarbeit
Informatik

September, 2023

Vorgelegt von Presented by	Rishabh Saxena Matrikelnummer: 416770 rishabh.saxena@rwth-aachen.de
Erstprüfer First examiner	Prof. Dr. rer. nat. Erika Ábrahám Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University
Zweitprüfer Second examiner	Prof. Dr. rer. nat. Thomas Noll Lehr- und Forschungsgebiet: Software Modellierung und Verifikation RWTH Aachen University
Betreuer Supervisor	Dr. Henning Petzka Lehr- und Forschungsgebiet: Theorie der hybriden Systeme RWTH Aachen University

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Die Stellen meiner Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen sind, habe ich in jedem Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht. Dasselbe gilt sinngemäß für Tabellen und Abbildungen. Diese Arbeit hat in dieser oder einer ähnlichen Form noch nicht im Rahmen einer anderen Prüfung vorgelegen.

Aachen, im September, 2023

Rishabh Saxena

Overview

Wind energy has been on the rise since the climate crisis, and there is ample need to provide better tools to plan new installations. A good use case for augmented reality and deep learning-based occlusion in augmented reality deals with the problem of how to accurately occlude virtual wind turbines in a smartphone. Using computer vision, one can programmatically derive these occlusion maps, using both non-neural-network and neural-network approaches, both of which are compared in this thesis. Depth estimation is an equivalently hard challenge, but efforts are made to solve this problem for a better final result as well. This thesis looks at two methods to provide sky segmentation, and a third method to provide depth estimation on the ground, finally combining to give an occlusion map that is used to place virtual objects. It finally concludes that a mixture of methods can be used to realize an implementable occlusion map.

Contents

List of Figures

List of Tables

List of Abbreviations

1 Introduction

Due to the increasing effects of climate change, a huge proportion of resources are being funneled into renewable resources. Here, in Germany, the renewable of choice for large power-generating units have become wind power[22], being the majority electricity provider in the country compared to other renewables. Still, many challenges remain, one of which deals with the accurate planning of the wind turbines that must be built. To facilitate this plan, this thesis looks at ways to augment the planning process using new technologies, particularly augmented reality and computer vision. The overall objective of this work is to contribute to the growing pool of solutions regarding accurate augmented reality applications for wind turbine placements. To that end, we look at how computer vision techniques can help solve the problem of the placement of virtual wind turbines in the viewfinder of a regular smartphone. Due to the complicated factors of the performance of the device, the infeasible task of accurately measuring the distance with a single, or monocular, image, and finally the task of deploying the methods at scale, we can consider this task to be non-trivial and worth looking at.

In particular, we look at the problem of occlusion, that is, the ability of an algorithm to accurately predict which pixel should be behind a given virtual object, and which pixel should be ahead of the given object. We first start with the formulation of the problem and previous literature in the domain, and then jump into looking at how to solve the problem. Throughout this thesis, the mention of "author" corresponds to the author of this thesis, unless stated otherwise.

1.1 Problem Formulation

We verbally define the problem as follows: From an input image, assuming that the given image is of an outdoor landscape, return a mask that provides per-pixel information on whether this pixel should be behind the object that is intended to be placed, or in front of this said object, by either performing this calculation locally or using a tiered server system that sends and receives the data via a network.

Refining this problem statement, we want to judge the occlusion of a virtual wind turbine when placed in the viewfinder of a smartphone camera, computing the depth information for the pixel and returning the information to a comparator algorithm to compare with the intended placement depth of the virtual wind turbine. Since the problem requires multiple solutions at multiple levels, we focus on a specific subset of tasks that should be solved in this case. Particularly, we look at receiving an input image and running it through an algorithm that provides an estimation of the depth, and finally providing this "mask" for further processing. We ignore the problems of actual deployment on devices using Unity, the lighting estimation, and the overall engineering challenges at scale, instead focusing on simply providing a comparison of possible solutions to provide the most efficient depth estimation for this image.

To achieve this result, we can divide the occlusion task further into two broad subtasks: sky segmentation, i.e. differentiating the sky from the ground since the sky will always be behind the virtual object, and ground depth estimation, i.e. deciding the occlusion of each pixel on the ground. We start with sky segmentation, first looking at a data-specific method to segment the sky from the ground using Mean Shift Clustering[12], followed by a deep learning method using UNets[50] and finally comparing the two methods for efficiency.

We then look at the ground depth estimation using pre-trained openly available models to provide relative depth information on the image. We then finally combine these approaches to provide as accurate a depth map as possible and list all the possible further challenges, improvements, and overall goals for this task.

For the sky segmentation subtask, we can define the problem more formally as: Given an input image X, label each pixel as the sky, which we shall consider as 1 throughout this thesis, and ground, which we shall consider as 0 throughout this thesis, based on a function $F(X, \theta)$, where θ is the model parameters or algorithm being used. The metrics we can define to evaluate the final result $Y_\theta = F(X, \theta)$ is precision, which is formulated as $precision_X(Y) = true\_positives_X(Y)/(true\_positives_X(Y) + false\_positives_X(Y))$, where true positives are correctly classified sky pixels, and false positives are incorrectly classified sky pixels. Similarly, for recall, which is formulated as $recall_X(Y) = true\_positives_X(Y)/(true\_positives_X(Y) + false\_negatives_X(Y))$, where false negatives are incorrectly classified ground pixels. These metrics will be used throughout this thesis.

1.2 Previous Literature

In this section, we look at the existing literature related to various aspects of augmented reality, including occlusion, depth estimation, neural networks for computer vision, and ARCore.

1.2.1 Augmented Reality

Augmented Reality (AR) has emerged as a significant technology that holds promise across various domains. By seamlessly merging digital elements with the physical world, AR enhances users' perceptual experiences, changing the way we interact with our environment. This dynamic integration of virtual objects into real surroundings has sparked interest due to its potential to bridge the gap between the digital and physical realms.

In recent years, AR has witnessed a surge in research and application across diverse fields including education, healthcare, entertainment, and industrial training. Its educational potential is demonstrated by its ability to create interactive learning environments, making complex concepts more accessible through interactive visualizations [32]. In the healthcare sector, AR has been explored for medical training, surgery simulations, and aiding in diagnostics by projecting medical imagery directly onto patients' bodies [5]. The gaming industry has harnessed AR's interactive capabilities to create immersive and engaging gaming experiences, blurring the lines between the virtual and real worlds [60]. Furthermore, AR has found utility in industrial training, facilitating the acquisition of complex skills through interactive simulations in sectors such as manufacturing and maintenance [5].

One of the key challenges in AR technology is the precise positioning and alignment of virtual objects in the real world. Researchers have diligently explored tracking techniques to accurately anchor virtual objects within physical environments. Marker-based tracking methods involve placing recognizable markers in the real world, which can be detected by AR systems to establish reference points [5]. Markerless tracking methods, on the other hand, seek to identify features in the real world without the need for predefined markers [60]. These methods have significantly contributed to enhancing the spatial accuracy and realism of AR applications, enabling users to interact with virtual objects as if they were part of their immediate surroundings.

While the field of AR is rapidly evolving, these developments underscore the transformative potential of this technology across diverse domains, offering a glimpse into a future where the virtual and real worlds interact seamlessly.

1.2.2 Occlusion in Augmented Reality

Creating convincing occlusion effects, where virtual objects realistically appear behind real-world elements, is a complex and ongoing challenge in the field of AR. The ability to seamlessly integrate virtual objects within the physical environment is essential for creating immersive and believable AR experiences.

To address this challenge, researchers have explored various techniques for handling occlusion in AR applications. These techniques aim to accurately depict the relationships between virtual and real objects in terms of depth and occlusion effects. By simulating how virtual elements interact with the surrounding environment, occlusion techniques enhance the sense of realism and immersion [8].

One common approach involves depth-based methods. These techniques rely on accurate depth information to determine the relative positions of virtual and real objects in the scene [26]. Depth maps, acquired through methods like stereo vision or depth sensors, provide valuable insights into the scene's geometry. By utilizing this depth information, AR systems can intelligently render virtual objects in a manner that respects the occlusion interactions with real objects.

Another avenue of research involves multi-view stereo techniques. These methods leverage multiple viewpoints of the same scene to reconstruct the underlying 3D geometry [26]. By analyzing the scene from different perspectives, multi-view stereo techniques enhance the accuracy of occlusion effects, ensuring that virtual objects align seamlessly with real-world elements.

Environment reconstruction is yet another strategy to handle occlusion. By creating a detailed 3D representation of the physical environment, AR systems can better understand the scene's layout and occlusion relationships [26]. This understanding enables more accurate rendering of virtual objects within the context of the real world.

Researchers have proposed innovative solutions to overcome occlusion challenges in AR applications. Depth image-based rendering involves generating images from depth maps, allowing virtual objects to be properly occluded by natural objects. Depth-aware compositing techniques enable the seamless fusion of virtual and real elements by taking into account depth information during the compositing process [26, 8].

These approaches collectively contribute to achieving more convincing occlusion effects in AR, enhancing users' perception of the virtual elements' integration within the real-world context. As AR technology continues to evolve, the ongoing advancements in occlusion handling techniques promise to elevate the quality and authenticity of AR experiences.

1.2.3 Depth Estimation

Accurate depth estimation plays a vital role in enhancing the realism and authenticity of AR experiences. Depth information offers critical context for the proper positioning of virtual objects within the physical environment and facilitates the accurate simulation of occlusion interactions between virtual and real elements. By accurately capturing the spatial relationships between objects, depth estimation ensures that virtual elements align seamlessly with

their real counterparts, contributing to a more immersive and believable AR environment.

Various techniques have been explored for depth estimation in AR applications. These techniques encompass a range of methodologies, each leveraging distinct principles and technologies to infer depth information.

Stereo matching is a fundamental technique that relies on the disparity between corresponding points in stereo images to estimate depth [24, 53]. By comparing the differences in pixel positions between left and right views, stereo matching can deduce the relative distances of objects from the camera.

Structured light involves projecting patterns of light onto a scene and analyzing the deformation of these patterns as they interact with the scene's geometry. This deformation provides valuable cues for depth estimation, enabling the reconstruction of a scene's 3D structure. [6]

Time-of-flight (ToF) sensors measure the time it takes for light to travel to an object and back to the sensor, allowing for the calculation of distance. ToF technology facilitates real-time depth measurements, making it suitable for dynamic AR environments. [23]

Monocular depth estimation leverages a single camera to infer depth information from 2D images. While traditional monocular techniques often rely on hand-crafted features and heuristics, recent advancements in deep learning, particularly Convolutional Neural Networks (CNN), have revolutionized this field. CNNs can learn complex depth cues from large datasets, enabling them to accurately estimate depth solely from a single image.

Recent developments in CNN-based monocular depth estimation techniques have led to remarkable improvements in accuracy. Models such as those introduced by Eigen et al. [16], Liu et al. [42], and Fu et al. [17] have demonstrated the potential of deep learning in capturing intricate depth patterns and translating them into accurate depth maps.

Collectively, these depth estimation techniques contribute to the foundational aspects of AR, enabling virtual content to be integrated with real-world scenes. As the field of depth estimation continues to evolve, these techniques promise to play a crucial role in shaping the future of augmented reality experiences.

1.2.4 Neural Networks for Computer Vision

Neural networks have played a significant role in computer vision, revolutionizing how tasks such as object recognition, image segmentation, and object tracking are approached. Among a variety of neural network architectures, convolutional neural networks (CNNs) have shown significant promise, showcasing remarkable efficacy across a spectrum of computer vision challenges.

CNNs excel in learning spatial hierarchies of features, making them particularly suited for tasks involving image data. These networks are structured to automatically capture and hierarchically process features, learning to discern essential patterns in raw images without explicit feature engineering. As a result, CNNs have demonstrated exceptional proficiency in tasks like image classification, where they outperform traditional methods by a considerable margin.

The efficacy of CNNs can be seen by their notable success in various computer vision competitions, including the ImageNet Large Scale Visual Recognition Challenge (ILSVRC)). Pioneering CNN architectures, such as AlexNet [36], VGG [55], ResNet [25], and DenseNet [27], have set new benchmarks in terms of accuracy and performance on challenging datasets.

Researchers have extensively explored these architectures for a range of applications, adapting them to specific domains including augmented reality. By integrating these advanced neural networks into AR systems, researchers have fortified the capabilities of object recognition and scene understanding. The robust feature extraction capabilities of CNNs enable accurate identification of objects within real-world scenes, while their hierarchical structure aids in comprehending the contextual information present in the environment.

Incorporating CNN-based models, such as those inspired by VGG, ResNet, and DenseNet, has elevated the quality and accuracy of object recognition and scene understanding in AR applications. These architectures' innate ability to capture intricate visual patterns and their potential for transfer learning has empowered AR systems to better perceive and interpret the visual information present in the user's surroundings.

The combination of neural networks and AR represents a significant advancement in the field. It utilizes state-of-the-art computer vision techniques to improve how we perceive augmented reality.

1.2.5 ARCore

ARCore, developed by Google, leads the way in enhancing AR experiences on Android devices. This advanced platform empowers developers with a wide range of tools and features, simplifying the creation of immersive and interactive AR apps.

ARCore's foundation relies on cutting-edge technologies to connect the digital and physical worlds. It achieves this through three essential features that transform how users interact with their environment:

1. **Motion Tracking:** ARCore uses advanced motion tracking algorithms to closely track the device's position and orientation in the real world. This enables a seamless blend of virtual objects into the user's surroundings, ensuring these objects move and interact naturally with real-world elements.

2. **Environmental Understanding:** ARCore excels in its capability to understand the spatial context of the environment. It achieves this by mapping the physical world, identifying flat surfaces, and recognizing features. This empowers developers to accurately place virtual content onto real-world objects, seamlessly merging the two realms.

3. **Light Estimation:** ARCore doesn't stop at spatial alignment; it also considers lighting conditions. It measures the intensity and direction of lighting in the environment, enabling virtual objects to cast shadows and be illuminated realistically, just like in the real world. This adds an exceptional level of visual detail and authenticity to AR experiences.

ARCore's impressive features have captivated the developer community, drawing considerable interest. Its versatility serves as an excellent starting point for creating a wide range of AR applications, spanning immersive games to useful tools for real-world tasks. By simplifying the intricacies of AR technology, ARCore frees developers to channel their creativity and innovation, allowing them to craft intricate and captivating digital experiences within the physical environments users inhabit.

ARCore has become a preferred choice for developers eager to explore the possibilities of AR [59]. By making these potent AR-building tools accessible to a wider audience, ARCore

drives the adoption of augmented reality experiences, fundamentally reshaping our interactions with the world.

In the following section, we look at the first of the two subtasks, namely the sky segmentation using mean-shift clustering, and then sky segmentation using UNets.

2 Mean-Shift Segmentation

As a baseline, we look at a clustering algorithm called mean-shift clustering, and an accompanying classification condition to segment the clustered image. We start off with a basic explanation of Image Segmentation and proceed to give an in-depth explanation of the algorithm itself, followed by an implementation.

2.1 Image Segmentation

Image segmentation is the process of dividing an image into different regions based on a certain feature or set of features in order to separate salient regions that might be of interest in solving a particular problem. Each region is homogeneous and this process is used in many applications such as object tracking and recognition[10], image retrieval[7], and volumetric reconstruction[11]. There are three main categories of image segmentation algorithms: feature-based clustering, spatial segmentation, and graph-based approaches[14]. Feature-based clustering uses the characteristics of the image through extraction and selection schemes. Spatial segmentation methods are called region-based when derived from region entities. Graph-based approaches combine feature and spatial information to group and organize images.

Alongside these three categories, there are also three levels of resolution for color image segmentation:

1. **Undersegmentation** is the case where the tolerance margin is very high, leading to multiple groups of pixels being assigned together as a single segment. This means that the most dominant color segments are considered, hence it has very low resolution.

2. **Oversegmentation** is the case when the feature palette is wide enough for the color segments to be significantly more than the undersgemented case. In this case, the tolerance is lower, the algorithm is more sensitive to changes in the color and the resolution is higher. This case is most favored for object recognition.

3. **Quantization** is the most sensitive case, where even small changes in color are significant for the algorithm to create separate segments, and the resolution is the highest. This means that the image can be quantized into very small sections, which are good for small object detection.

We shall utilize the undersegmentation case for the sky-segmentation algorithm since we are looking to segment the image into ”Sky” and ”Ground”, which are very broad segmentation labels. The goal of this work is to use color and spatial segmentation to provide accurate background and foreground in the context of the sky, in order to place a virtual object at the optimum depth. One algorithm that the author used to segment the sky region from the ground region is the ”Mean-Shift Clustering” algorithm to cluster the images and an

additional classification condition to classify the clusters into their respective segments. The following sub-section details more on the same.

2.2 Mean-Shift Clustering

Mean-Shift Sky Segmentation derives its inner workings from the original algorithm of Mean-Shift Clustering. We can elaborate on the Mean-Shift Clustering algorithm as follows:

Mean-shift Clustering is an iterative algorithm used for cluster analysis and mode-seeking in a dataset. In each iteration, it shifts each data point toward the weighted mean of its neighbors within a given window size (bandwidth). The algorithm continues to update the positions of the data points until convergence, where each data point reaches a local mode in the underlying data distribution.[12]. The algorithm starts by assigning each data point in the n-dimensional space to its own cluster. Then, for each data point, it computes the mean of all points within the window and shifts the data point to this mean. This process is repeated until convergence, which is defined as the iteration after which there are no changes to the mean value of the points in the window, or a certain maximum number of iterations have been reached.

Once all points have been processed by the mean-shift algorithm and their values have converged, clusters are formed by grouping together points that have converged to the same final mean value. The final step involves filtering out near-duplicate clusters and clusters with less than a minimum defined number of points within, to form the final set of centroids.

In the context of mean-shift clustering, the key parameter is the bandwidth (window size), which controls the number of points being processed in a single iteration and thus determines the smoothness of the resulting density estimate. A smaller bandwidth will result in more detailed clusters since the window size is smaller. The algorithm focuses on a smaller region around each data point which allows it to capture more fine-grained patterns and local variations in the data. As a result, the clusters formed will be more detailed, representing smaller regions with distinct characteristics. with more peaks and valleys. A larger window size leads to a broader consideration of neighboring points. As a result, data points from a larger region influence the mean shift of each data point. This tends to smooth out the cluster boundaries, as points from different regions may have similar mean values and become part of the same cluster.

We can elaborate this method mathematically by starting with Kernel Density Estimation, which is used to identify the clusters in a non-parametric way for Mean-Shift Clustering.

2.2.1 Kernel Density Estimation

Kernel density estimation [62] is a non-parametric way to estimate the density function of a random vector. Since this work deals with multivariate random variables (due to the fact that each pixel has a multivariate feature space), we can directly look at the mathematical definition of the multivariate density estimation function. We have X_1, X_2, $X_3 \dots X_N$ d-variate random vectors which are also independent and identically distributed, whose density function is defined by f. Given a kernel K, which is a symmetric multivariate density function, bandwidth parameter H which is a $d \times d$ symmetric, positive-definite matrix, the Kernel Density Estimator for a given set of N-dimensional points is:

$$\hat{f}_H(X) := \frac{1}{N}\sum_{i=1}^{N} K_H(X - X_i) \tag{1}$$

Where K_H is defined as

$$K_H(X) := |H|^{-\frac{1}{2}} K(H^{-\frac{1}{2}}X)$$

Kernels follow the additive property, i.e., if $K(X)$ and $L(X)$ are kernels as defined above, then $(K + L)(X)$ is also a kernel, defined as a symmetric multivariate density. This is used for the clustering algorithm since we add the kernels of individual points to create the kernel density.

2.2.2 Mean-Shift

In the case of mean-shift clustering, we can take the defined kernel density estimation function and use it to find the mean of the kernel with bandwidth H. The mathematical description follows:

$$\hat{m}(X) := \frac{\sum_{X_i \in N(X)} K_H(X - X_i)X_i}{\sum_{X_i \in N(X)} K_H(X - X_i)} - X \tag{2}$$

where $\hat{m}(X)$ is the mean-shift vector, X is the centroid at which the bandwidth is centered, and $N(X)$ is the set of points in the neighborhood of X, i.e. the set of points for which $K_H(X - X_i) \neq 0$. At each iteration, we have $X \longleftarrow X + \hat{m}(X)$, that is, $X + \hat{m}(X)$ is assigned to X for all $X \in \mathbb{X}, \mathbb{X} \subset \mathbb{R}^{(n \times d)}$ ($\mathbb{X}$ being the set of all data points, which is initially the set of centroids as well), and then we calculate the mean-shift vector again until convergence, which is when the L2 distance between X and $X + \hat{m}(X)$ is a very small quantity ϵ, or we reach the maximum number of iterations T. The final $\hat{X}$ value is now considered as the label of the initial X and is assigned to the cluster centered at $\hat{X}$. This process is further referred to as `mean-shift-clustering`, which is only the clustering algorithm.

For each data point $X_j \in \mathbb{X}$, we perform this mean-shift clustering and obtain a set of centroids belonging to each point. This method is also considered a gradient ascent approach since we obtain the gradient estimate of the density function for each point and shift the mean vector of the cluster of points to a position that has a higher function value than the current mean value, which is also known as a gradient clustering approach[18].

2.3 Sky Segmentation: Algorithm and Implementation

To perform mean-shift clustering-based sky segmentation, we shall perform mean-shift clustering on a feature space of an image in a multidimensional setting, then take the most dominant cluster in the top half of the image to be the sky. We can restrict the parameters (i.e. size of the bandwidth and the kernel function) of Mean-Shift and optimize the final segmentation using knowledge control i.e. knowing that the sky must be the most dominant cluster in the upper half, which we define to be as the cluster with the largest number of pixels. The algorithm for mean-shift clustering-based sky segmentation, named as `baseline-mean-shift` from further on here (since we consider this algorithm as our baseline to compare all other algorithms to), is as follows

Algorithm 1 `mean-shift-clustering`

Let $\mathbb{X}$ be the set of all 5-dimensional data points of the image (RGB values and XY coordinates)
Let $X_j \in \mathbb{X}$ be a pixel.
Let T be the maximum number of iterations allowed.
Let τ be the current iteration number, initialized as 0.
Let $\hat{m}(\cdot)$ be the mean-shift function.
Let K_H be a chosen kernel function with bandwidth H
Let $w(\cdot)$ be 1
$\bar{X}_j = X_j$
while $\hat{m}(X_j) > \epsilon$ or $\tau \leq T$ **do**
    $\bar{X}_j \longleftarrow \bar{X}_j + \hat{m}(X_j)$
end while
Let $\hat{X}_j$ be the final value of X_j
Assign X_j to the cluster whose centroid is $\hat{X}_j$.

We perform Algorithm 1 on each pixel of the image $\mathbb{X}$ to get the cluster of each pixel. We then eliminate all clusters that contain less than a certain number of points, which we term as *min-density*. We then assign the remaining centroids $\mathbb{X}$‘ to the closest cluster centroid using the L2 distance between them. The bandwidth consists of two radii, namely the Spatial Radius (consisting of x position and y position) and the Color Range Radius (consisting of R, G, and B values). In terms of matrices, these radii are the diagonal elements of the bandwidth matrix H, which are pre-defined before the experiments.

2.3.1 Implementation

The algorithm was implemented by the author in Python 3.9.7 for testing and benchmarking purposes. The external libraries used were OpenCV[9], SciPy[58], MatPlotLib[28] and PyMeanShift[30]. PyMeanShift is a Python package that implements the `mean-shift-clustering` algorithm using the Blepo Computer Vision Library, which is a C++ implementation of some of the most commonly used computer vision algorithms. Particularly for `mean-shift-clustering`, the library uses the Uniform Kernel, which is defined as follows:

$$K_{h_{i,i}}(u_i) := \mathbb{1}_{\{|u_i| \leq h_i\}} | u_i \in U, h_{i,i} \in H, U \in \mathbb{X} \tag{3}$$

Here, $K_{h_{i,i}}(u_i)$ is the kernel function on the ith feature u_i of a random vector U, which is a point in the 5-dimensional pixel space, and $h_{i,i}$ is the bandwidth for the ith feature, and the diagonal element (i, i) of H, and $\mathbb{X}$ is the set of all vectors. The implementation details are presented at this[1] GitHub repository.

Before segmentation using mean-shift-clustering, we must first resize the images to 256×256 in order to reduce computation. After segmenting the image into its mean-shifted components, we can perform a simple approximation of the sky label by splitting the image in half and taking the most dominant cluster on the top half of the image, as performed in Nice et al. [44]. The author performed their analysis on three combinations of parameters, mainly

[1] https://github.com/cjblackout/masterthesis

Figure 1: Example of `baseline-mean-shift` for Sky Segmentation

Original Image (left), Segmented Image (middle), and Final Result (right). This example uses the `baseline-mean-shift` algorithm with parameters ($sr = 2, rr = 8, min = 200$), yielding a precision of 0.97 and recall of 1.0.

($sr = 7, rr = 7, min = 200$), ($sr = 5, rr = 6, min = 100$), and ($sr = 5, rr = 8, min = 300$), referenced from Nice et al. [43] (see Section 2.3.2 for more details on this dataset), where sr is the Spatial Radius (SR) parameter for the x and y coordinates of a pixel, rr is the Color Range Radius (RR) parameter for the Red-Green-Blue (RGB) values, and min is the minimum density parameter, equivalent to *min-density* defined previously. An internal parameter, 'speedup' was set to default as 'SPEEDUP_HIGH', which was not controlled during experiments.

An example of the Mean-Shift Clustering-based Sky Segmentation algorithm, `baseline-mean-shift` is presented in Figure 1.

2.3.2 Dataset

This small section deals with listing the main dataset used in this thesis for evaluation of the current algorithm, and training and evaluation of all neural network-based algorithms that are mentioned later.

The input image presented in Figure 1 is taken from the dataset presented by Nice et al. [43], which is described as follows: The dataset consists of two directories, `OriginalImages` and `ValidationImages`. The former directory contains the input images which are taken from 39 locations, each location having been photographed multiple times during a year for multiple years. These images are images of landscapes from various regions, covering a variety of possible geographies. These images are of type JPEG and are in the RGB space. The validation images directory contains PNG images, where each pixel is either 0 or 1, 1 representing a sky pixel, and 0 representing a ground pixel. These pixels create a mask, for each corresponding image in the original images directory, that segments the sky from the ground. The total number of images sums up to be around 38,000, excluding partially corrupted images that may exist. In all following sections, we use the term "dataset" to refer to this dataset or `original-dataset`, unless mentioned otherwise.

Another, smaller, dataset used in later sections of this thesis, named as `bachelor-dataset` is taken from the work of [20] who provide a dataset similar to `original-dataset`, but much smaller in the number of images (It contains only 62 images) taken around the city of Aachen, Germany. It has been modified to have the same directory structure as the `original-dataset` and contains the same RGB feature images, and label images that provide per-pixel labels for sky (label 1) and ground (label 0). The author specifies the usage of this dataset wherever used, but otherwise mean `original-dataset` whenever they reference "dataset".

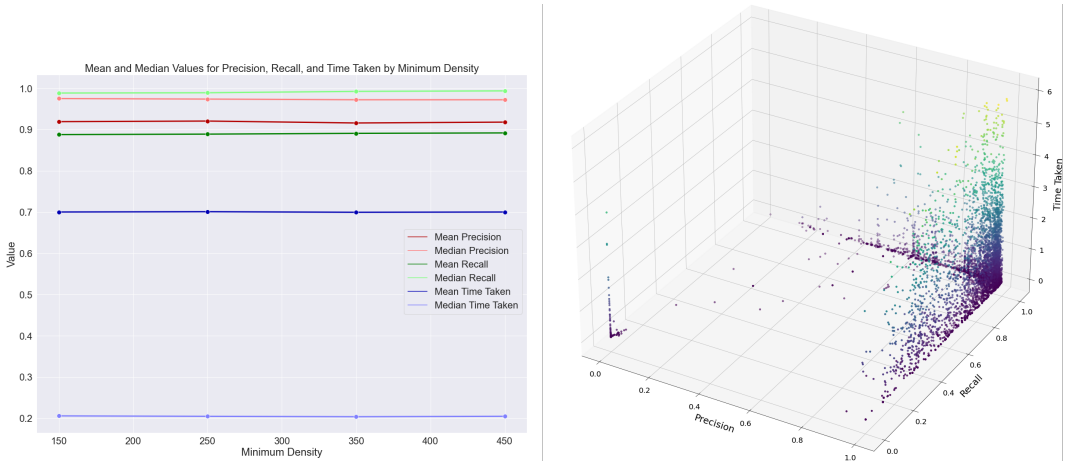

Figure 2: Change in Precision, Recall, and Time-Taken on varying Minimum Density

(Left) X-Axis is the min_density value, Y-axis is the mean and median of precision, recall, and time taken respectively. (Right) Distribution of the images based on their precision, recall, and time, for a minimum density value of 150. Note that change in minimum density does not affect the precision, recall, or time taken, hence we take 150 based on the score in Figure 5

In the next section, we look at the results of the evaluation done on `original-dataset`.

2.4 Image Analysis Results

In this section, the author provides some results regarding different parameters of the algorithm that have an impact on the performance of the algorithm. To define performance, they show the most important metrics that we need to quantify a successful detection of the sky. Since we must optimize the algorithm to work for video input in real-time, we must take into account the speed i.e. the time taken for one frame to process in a running video format should be equal to the framerate of the input stream or video, alongside our previous metrics of precision and recall. Since this condition is extremely strict considering varying hardware and software limitations, we instead look at a unified score that combines precision, recall, and time taken per frame (also referred to as just time taken), in order to select the most optimized configuration of the algorithm.

For their analysis, the author provides an evaluation of the combination of parameters that affect the algorithm. These are the spatial radius, the color range radius, and the minimum density of a cluster, plotted in Figures 2, 3, and 4.

To evaluate all metrics using a level playing field, the author defined a score function which is

$$score = \frac{precision + recall}{time\_taken + 1}$$

This function takes the precision and recall of each predicted value and divides it by the time taken to process the image from start to end. They added 1 to time_taken in order to scale the denominator better with the numerator since, in practice, the value of time_taken can vary a lot depending on the parameters. It can range from 10 seconds to 0.01 seconds, and hence, to visualize the score better, they chose to scale the lower values of time. Using this

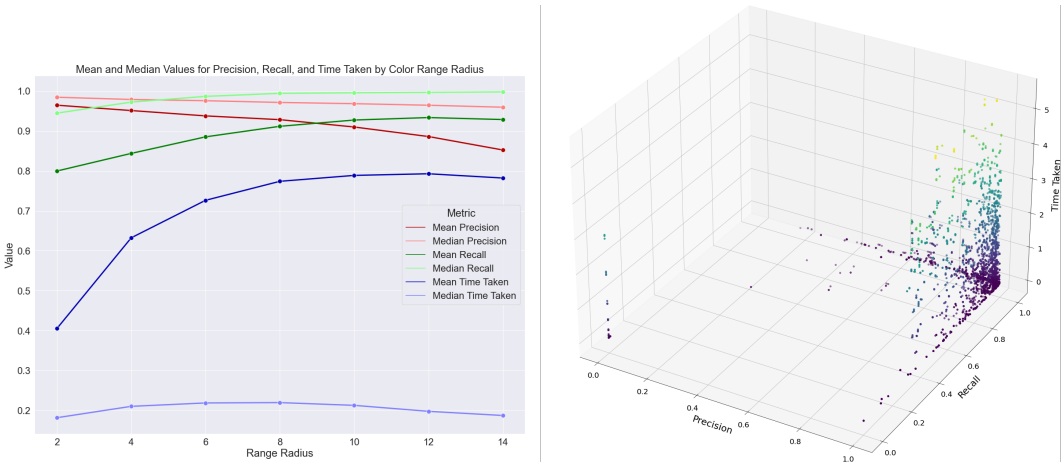

Figure 3: Change in Precision, Recall, and Time-Taken on varying Color Range Radius

(Left) X-Axis is the color range radius value, Y-axis is the mean and median of precision, recall, and time taken respectively. (Right) Distribution of the images based on their precision, recall, and time, for a color range of 8. Note that there is a trade-off between the precision and recall, and the time taken, for lower values of color range radius. Hence, we choose the optimum values based on the results of Figure 5.

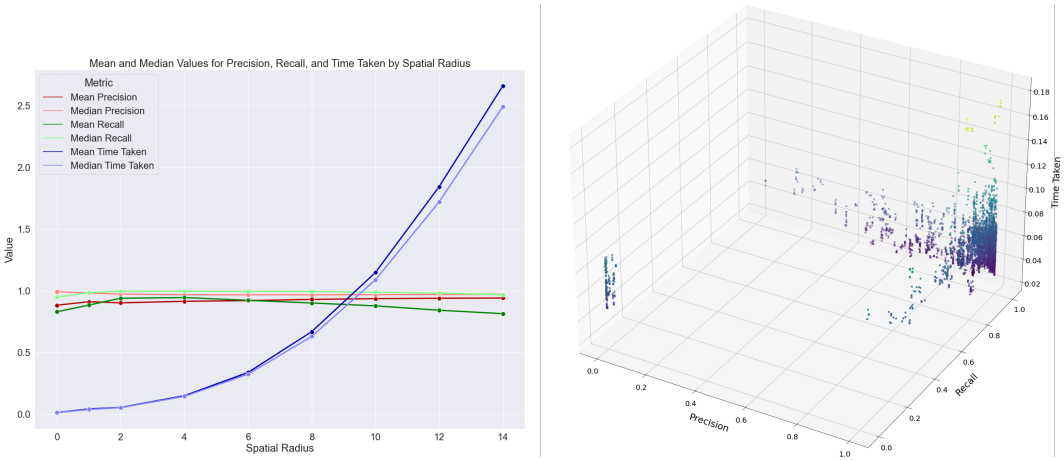

Figure 4: Change in Precision, Recall, and Time-Taken on varying Spatial Radius

(Left) X-Axis is the spatial radius value, Y-axis is the mean and median of precision, recall, and time taken respectively. (Right) Distribution of the images based on their precision, recall, and time, for a spatial radius of 2. Note that spatial radius is significant in the time taken, but less significant for the precision and recall values. Hence, we take 2 as the optimum value, also proven by Figure 5.

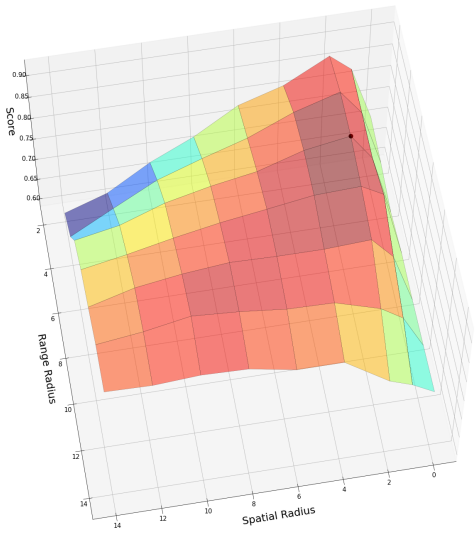

Figure 5: 3D Plane of the Distribution of Score Values on varying Spatial Radius, Color Range Radius, and Minimum Density

X-axis is the Spatial Radius, Y-Axis is the Color Range Radius, Z-Axis is the Score. The color variation goes from bluer to redder as the score increases. A black point is placed on the highest score combination of metrics. It can be observed that lower spatial radii values produce better scores, while color range radii minimally affect the score value.

score, we can plot a score graph to account for the best value to use in real-time applications, which can be seen in Figure 5

Using this analysis, it is clear that certain values of spatial and color range radii affect the performance of the algorithm, with respect to the time taken. We can take the best value for spatial range since the relationship is inverse i.e. higher values of spatial radius yield worse scores. An exception is also the fact that values below 2 also yield worse scores, hence peak score is achieved at spatial radius 2. Hence, we take the value of 2 for the spatial radius for our optimal algorithm while taking the color range radius as 8. We keep the minimum density at 150 since it doesn't affect the overall performance.

In conclusion, a certain set of parameters, $(sr = 2, rr = 6, min = 150)$, give the most optimum performance, with the average time taken to detect sky being 0.052 seconds, albeit on high-end hardware(AMD Ryzen 9 5900HX, RTX3070 Mobile GPU, 16GB of DDR4 RAM and 1TB of NVMe SSD, using Windows 11 64-bit). This gives us an average framerate (without any additional functions) of 19.23 frames or 19 frames as a rounded figure.

In the next section, we shall look at an in-depth analysis of each cluster computed by the `mean-shift` clustering algorithm and the classification paradigm, with an analysis of failure cases and further ways to improve the performance.

2.5 Cluster Analysis Results

In this section, we look at the `original-dataset` from the lens of each cluster. We first start by evaluating the clustering algorithm itself by defining "ground_truth" metrics and taking

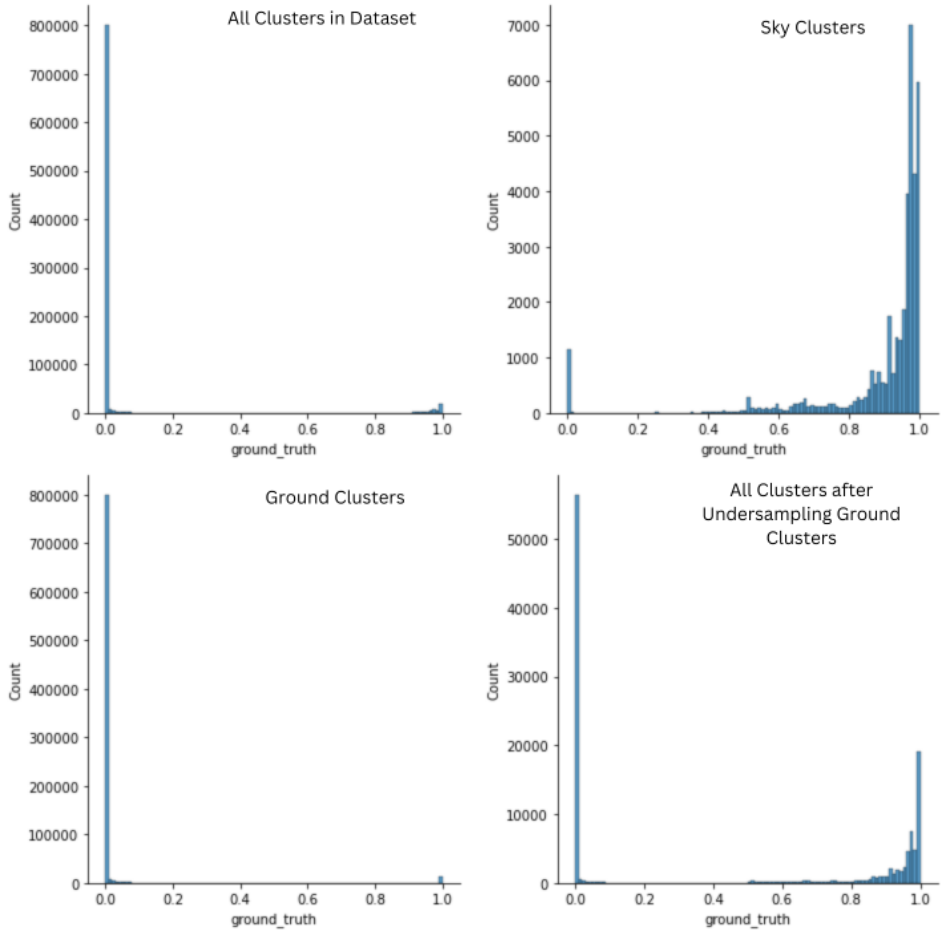

Figure 6: Distribution of Clusters against ground_truth values under different conditions

(Top Left) Distribution of the cluster *ground_truth* value for each cluster, as a count. (Top Right) Distribution of the cluster *ground_truth* for each cluster, as a count, where the prediction made by `baseline-mean-shift` is the sky. (Bottom Left) Distribution of the cluster *ground_truth* values against the number of clusters, where the prediction made by `baseline-mean-shift` is 0. (Bottom Right) Distribution of the cluster *ground_truth* values against the number of clusters such that the number of ground and sky clusters are not imbalanced.

an ideal classification condition that classifies the clusters based on a threshold value for the "ground_truth". After evaluating the maximum, possible performance that can be achieved through the `mean-shift-clustering`, we then look at where the `baseline-mean-shift` algorithm fails by looking at the false positives and false negatives generated by it.

2.5.1 Ideal Classification Condition

In order to evaluate the performance of the overall `mean-shift-clustering` algorithm, we must first look at each cluster, and whether the clusters encapsulate the sky and ground effectively. In simpler terms, we look at if all the pixels in a given cluster are either completely sky or completely ground, and there exists no cluster where there is a mixture of the two. Since this condition is too strict, we instead look at the percentage of pixels in a cluster that are sky and define a threshold value for the percentage to classify these clusters as sky clusters.

The author first generated a per-cluster CSV file of the entire `original-dataset`, in order to infer the performance of the algorithm on a per-cluster basis. This was done in order to better understand the shortfalls of the `mean-shift-clustering` algorithm by analyzing how well the algorithm can create the clusters, and whether the clusters themselves are well-divided into sky and ground regions, with minimal overlap. We can do this by first running the `mean-shift-clustering` algorithm on each image of the `original-dataset` and then taking the metadata produced (average color of cluster, average position of centroid, etc.) of each cluster as the data points for our CSV. The data points include the label of the cluster given by the `mean-shift-clustering` algorithm, mean R, G, B, the number of pixels, the X coordinate of the centroid, and the Y coordinate of the centroid. We also produce two artificial metrics, "ground_truth", i.e. the percentage of pixels that are the sky in a cluster, "predicted_value", i.e. the label given to the cluster by `baseline-mean-shift`. For evaluating the `mean-shift-clustering` algorithm, we only use the ground_truth metrics, while the predicted_value metric is used to see the false positives and false negatives produced by `baseline-mean-shift`.

For each cluster, we can compute a percentage value, which we name "ground_truth", representing the percentage of pixels in any cluster that belongs to the sky. We get this value by taking the cluster in question and masking the rest of the image, so as to isolate the clusters in the image, then masking the validation image with the same mask, and finally comparing the remaining regions to calculate the percentage of sky pixels. We can then calculate the number of sky pixels in this isolated-masked image and divide it by the total number of pixels in the cluster. Note, ground_truth is a metric derived from the `original-dataset` itself. To visualize the performance of the `mean-shift-clustering`, we can plot the distribution of all clusters with respect to ground_truth, which is shown in Figure 6, top-left. Since the number of clusters in the bottom half of the image is more than the top half, owing to the fact that the ground in any landscape image is always more complex, a large number of clusters have no sky pixels at all, which can be seen in the high count for pixels with almost 0 ground_truth.

The author computed the maximal precision, recall, and F1 score that can be achieved by the clustering algorithm, by computing the number of pixels that are correctly classified as the sky using a "threshold" value for each cluster. Using experimental trial-and-error, they came to a threshold value of 0.6 for the best possible values of the intended metrics, therefore saying that if a cluster has a "ground_truth" value of more than 0.6, it is classified

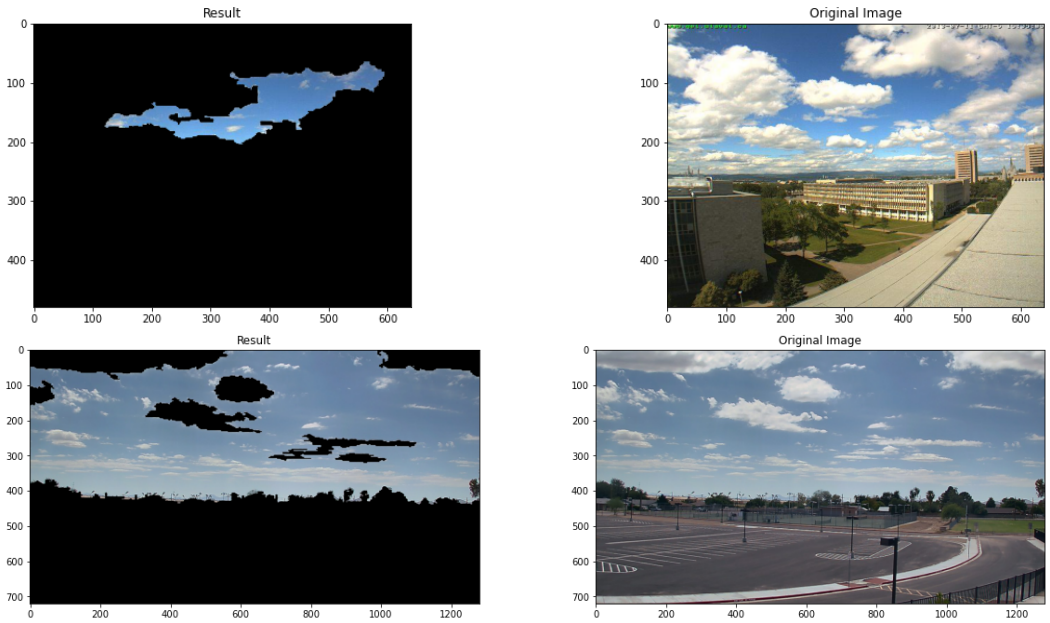

Figure 7: False Negative Cases for `baseline-mean-shift`

Two samples of images where there are clusters with predicted_value as ground but they contain "ground_truth" of more than the threshold, i.e. they contain a majority of sky pixels. It can be seen that the algorithm classifies the sky region as the ground in both cases.

as the sky, otherwise it is classified as ground. Using this "threshold" classification condition yields the following results: True Positive Pixels were 1,215,772,930, False Positive Pixels were 34,256,824, True Negative Pixels were 1,211,930,278, and False Negative Pixels were 33,692,969. Using this information, we calculate the total Precision as 0.972, Recall as 0.973, and the F1 score as 0.973. We can consider this to be the maximum possible performance that can be delivered by the `mean-shift-clustering` algorithm due to its inherent clustering nature. We shall use this result in the next section during the improvement phase of the `baseline-mean-shift`.

2.5.2 False Positives and False Negatives of `baseline-mean-shift`

For evaluating the `baseline-mean-shift` algorithm, we can define a second artificial metric, "predicted_value", which is the label computed by the algorithm, which in short, takes the largest cluster by the number of pixels in the upper half of an image, and calls it the sky cluster.

We can finally plot the distribution of clusters with respect to "ground_truth" that have "predicted_value" (using `baseline-mean-shift`) as 1, as shown in Figure 6, top-right, and "predicted_value" (using `baseline-mean-shift`) as 0, shown in Figure 6, bottom-left. From these graphs, we can see that there are some clusters that are classified as the sky (prediction 1), but contain no sky pixels at all, as plotted in Figure 8, and there are some clusters classified as ground, but they contain a majority of sky pixels, as plotted in Figure 7. We also plot a "resampled" distribution of the in order to better visualize the distribution of clusters, plotted in Figure 6, bottom-right.

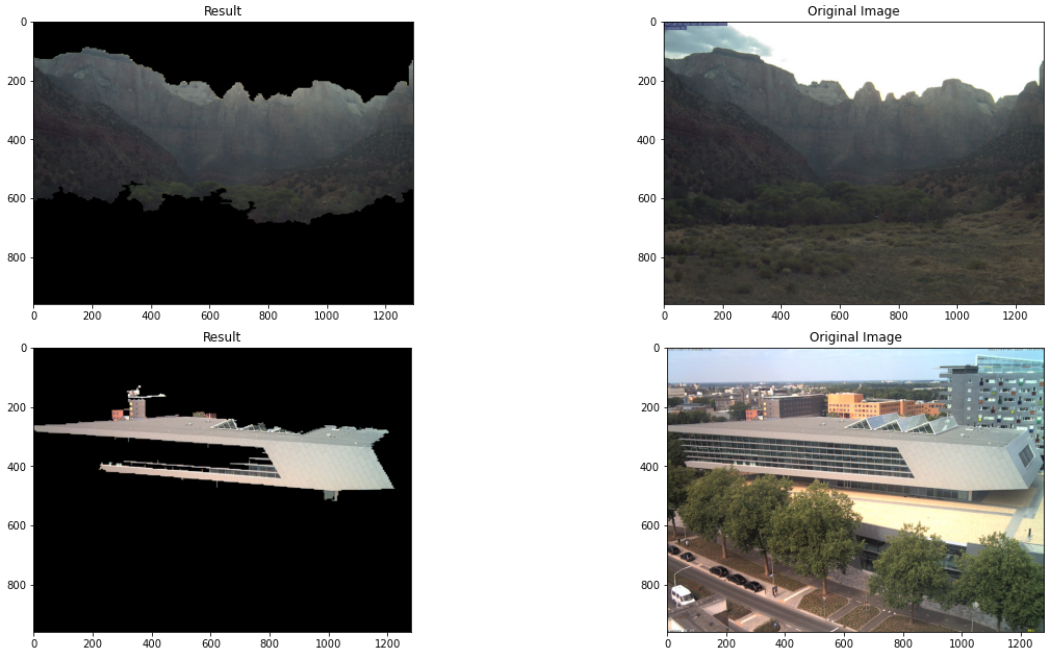

Figure 8: False Positive Cases for `baseline-mean-shift`

Two samples of images where there are clusters with predicted_value as the sky but they contain ground_truth of less than the threshold, i.e. almost no sky pixels in them. It can be seen that the algorithm classifies the wrong region as the sky.

In the next section, we use these results to formulate improved variations of the `baseline-mean-shift`.

2.6 Variations of `baseline-mean-shift`

In this section, we look at the various modified versions of the `baseline-mean-shift` algorithm based on insights from the cluster analysis of the original algorithm. Particularly, we look at how we can improve performance using different classification conditions.

We observe that in some cases, the clustering algorithm produces clusters that might not represent the complete sky, or "Fragmentation" (i.e. there are cloud clusters, or the sky is divided into different portions) or the dominant cluster (i.e. the cluster with the largest number of pixels) in the top half of the image may not be a sky cluster at all. There is also the problem of "border regions", or the border between the sky and ground where sometimes the clustering algorithm creates clusters that are overlapping i.e. the cluster contains a significant number of both sky and ground pixels, and these border regions are not counted towards the sky.

In order to solve the first problem of fragmentation, we can take each cluster and calculate certain features that are inherent to this cluster, in order to find a range of values for these features that correspond to whether that cluster belongs to sky or ground. We can take the following features: Mean R, G, and B values of the cluster, X and Y coordinates of the cluster, and the Label of the clusters. The mean R, G, and B values are the mean RGB values of all the pixels in a cluster, the X and Y coordinate is the pixel number, and the

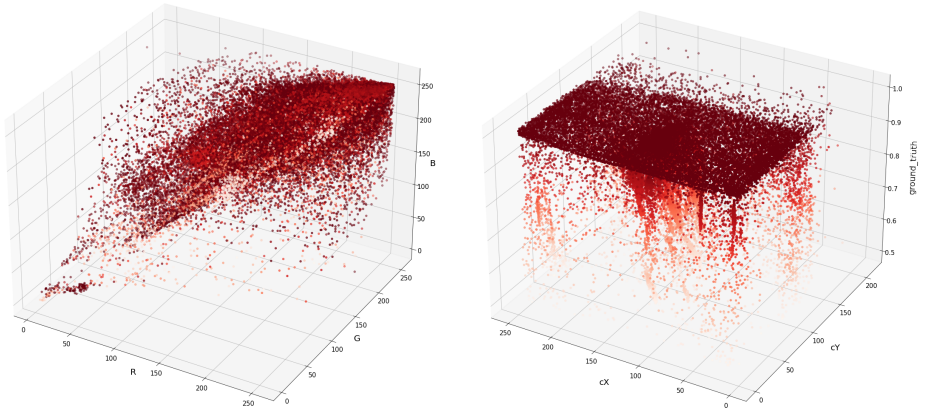

Figure 9: 3D Distribution of Clusters in RGB(left) and XY(right) Feature Space

(left) Clusters plotted in 3D Feature space based on their mean RGB values, with the axis being R, G, and B values that are above the 0.6 threshold value for ground_truth, (right) Clusters plotted in a 3D Feature space based on their XY coordinated, and the ground_truth value, with the axis being X, Y, and ground_truth values that are above the 0.6 threshold value for ground_truth

Label is the label given by the algorithm to the clusters starting from the top left. Each new cluster is encountered from the top left going right, and then each row below is assigned this label, starting from 0 to the number of clusters in the image. The intuition of this approach follows from Figures 9

Using these features, we can calculate the range of the metadata features (mean RGB, XY, etc.) of the clusters that are classified as sky using the threshold value of 0.6 for the ground truth. This means that we can calculate the range of mean R, G, B, X coordinate, Y coordinate, and Label which cover 96% of all clusters that are classified as Sky. The author arrived at this number since taking 100% of the cluster lead to more false positives than with just 96%, but any lower, and the precision suffered again. We name this the optimal range, calculated based on trial-and-error to figure out which range of values for these novel features gave the author the least possible error (i.e., using this range, what percentage of clusters that are classified as sky are not actually sky). After calculating the range of values for these novel features, we get an error percentage of 4.9% error, which is the lowest in any configuration of percentage and threshold.

Using these optimal ranges, we can give the initial `baseline-mean-shift` a new condition check, stating that if a cluster's features fall within this optimal range, then classify this cluster as the sky. The augmented `baseline-mean-shift` algorithm, which we call "`baseline-mean-shift-augmented`" in Algorithm 2:

Another variation that we can compare with the original `baseline-mean-shift` algorithm is a modified version of the algorithm that takes the clusters on the top third of the image instead of the top half. This tackles the problem of clusters that are originally sky, but don't occupy a dominant space in the upper half of the image, but rather the upper one-third of the image. We can assume this does not hamper performance since the dominant cluster in the upper one-third of the sky will still be sky if it is also the dominant cluster in the upper half. We name this algorithm "`baseline-mean-shift-top-third`".

Algorithm 2 `baseline-mean-shift-augmented`

Perform Mean-Shift Segmentation 1 on image $\hat{X}$
Calculate the Mean R, G, B, X coordinate, Y coordinate, and Label of X
Assume optimal ranges are R_{lower}, R_{upper}, G_{lower}, G_{upper}, B_{lower}, B_{upper}, X_{lower}, X_{upper}, Y_{lower}, Y_{upper}, $Label_{lower}$, $Label_{upper}$
if $(R_{lower} \leq R \leq R_{upper})$ AND $(G_{lower} \leq G \leq G_{upper})$ AND $(B_{lower} \leq B \leq B_{upper})$ AND $(X_{lower} \leq X \leq X_{upper})$ AND $(Y_{lower} \leq Y \leq Y_{upper})$ AND $(Label_{lower} \leq Label \leq Label_{upper})$ **then**
    Prediction = Sky
else
    Prediction = Ground
end if

	Mean Precision	Median Precision	Precision SD	Mean Recall	Median Recall	Recall SD
`baseline-mean-shift`	0.959	**0.983**	0.095	0.894	0.982	0.197
`baseline-mean-shift-augmented`	0.913	0.969	0.143	0.946	**0.994**	**0.150**
`baseline-mean-shift-top-third`	**0.964**	**0.983**	**0.070**	0.896	0.982	0.193
`baseline-mean-shift-augmented-top-third`	0.920	0.971	0.132	**0.968**	0.997	0.109

Table 1: Performance Comparison of Variations of `baseline-mean-shift`

Performance comparison of different variations of `baseline-mean-shift` model in terms of Mean Precision, Median Precision, Precision Standard Deviation, Mean Recall, Median Recall, and Recall Standard Deviation. The variations include `baseline-mean-shift`, `baseline-mean-shift-augemented`, and `baseline-mean-shift-top-third`. Note the abbreviation for Standard Deviation (SD).

The author ran the analysis of all the images on all three algorithms again and plotted the outcomes in Table 1

From this table, we can infer that, out of the three methods, `baseline-mean-shift-top-third` performs marginally better than the original `baseline-mean-shift` algorithm in terms of precision, but performs similarly in terms of recall. On the other hand, our `baseline-mean-shift-augmented` algorithm performs much better in terms of recall but fails to perform well in terms of precision. We attribute this behavior to the "conservative" nature of the `baseline-mean-shift-augmented` algorithm since it correctly classifies the ground but makes mistakes classifying the sky, hence being conservative towards classifying the sky. The opposite is true when we look at the other two algorithms. The inferred reason for this is that when we look at the ranges, the calculated range for the novel features provides a good range of values for the cluster features, but not all values can be included (e.g. night sky would have different features than the optimal range), hence it performs poorly in terms of precision. We plot the precision and recall graphs for each of the algorithms in Figure 10

We finally combine the three approaches to try and incorporate two different kinds of classifications: the largest cluster is classified as the sky on the top third of the image, while in the region between the top third and bottom half, we can use the range method to classify small regions as the sky that might be missed by the clustering algorithm. We can consider there to be no explicit classification mechanism for the bottom half of the image, considering most clusters to be of the ground class here. We name this algorithm `baseline-mean-shift-augmented-top-third`.

Unfortunately, we see a decrease in the performance metric, as shown in table 1, with

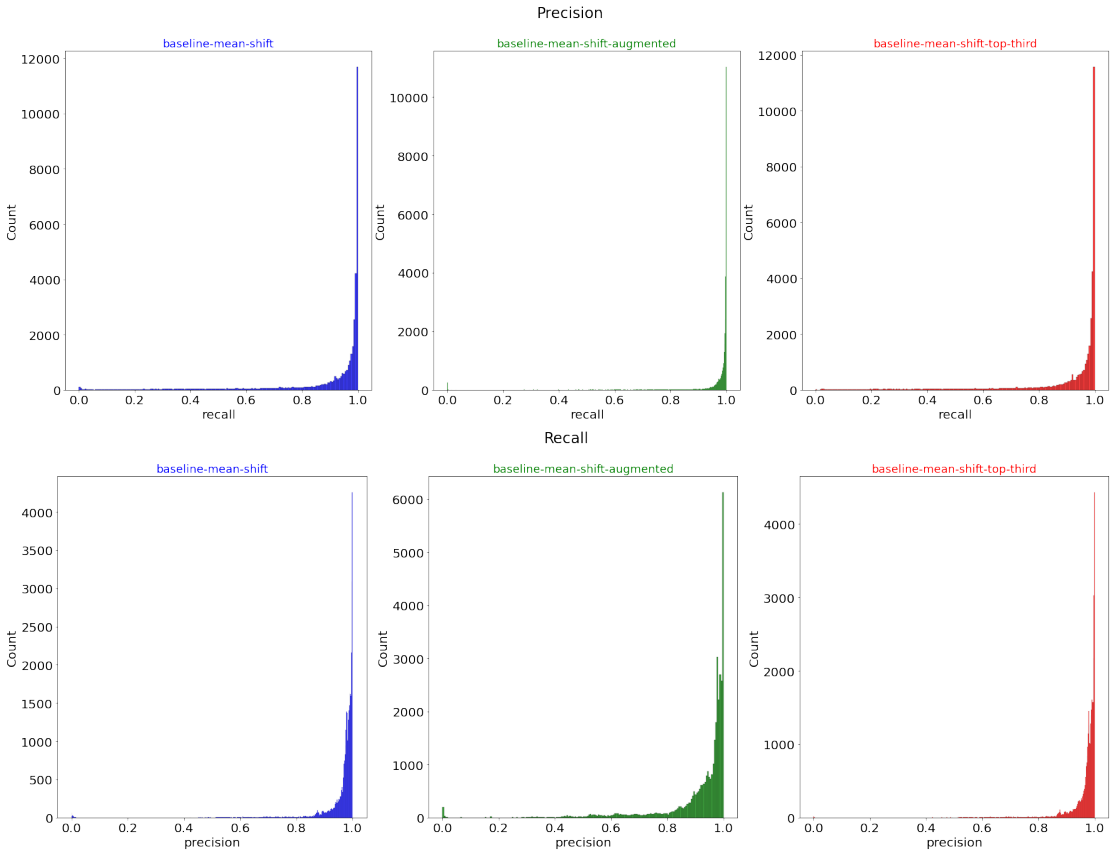

Figure 10: Distribution of metrics on `original-dataset` for variations of `baseline-mean-shift`

A plot of precision and recall for each image in `original-dataset`, for each variation of the `baseline-mean-shift` algorithm

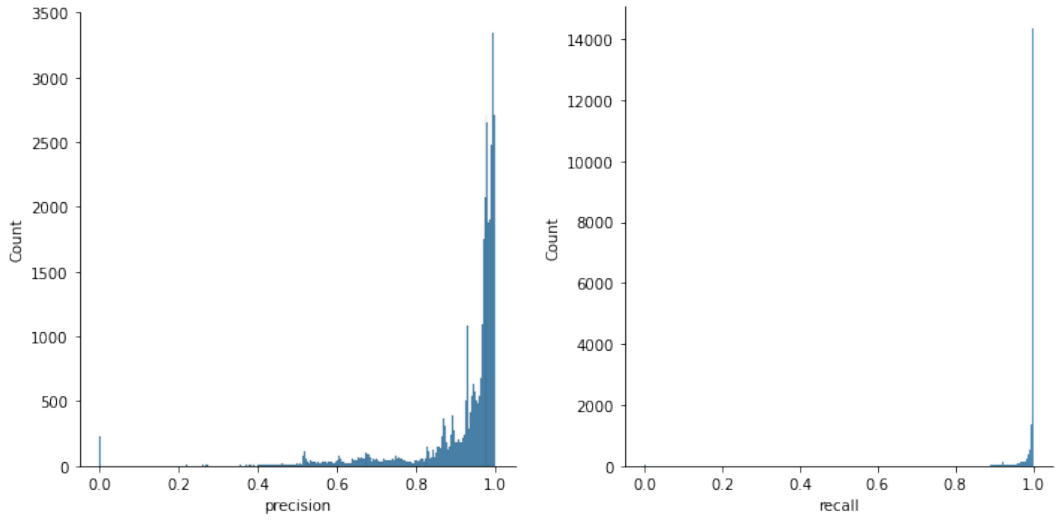

Figure 11: Distribution of metrics on `original-dataset` for `baseline-mean-shift-augmented-top-third`

(left) Precision of the `baseline-mean-shift-augmented-top-third` for each image, (right) Recall of the `baseline-mean-shift-augmented-top-third` for each image.

the mean precision being 0.920, the median precision being 0.971 and the precision standard deviation being 0.132. On the other hand, our mean recall was 0.968, our median recall was 0.997 and our Recall Standard Deviation was 0.109. We can see from the Figure 11 that we have precision lesser than `baseline-mean-shift-top-third` and recall lesser than `baseline-mean-shift-augmented`. Based on Figure 11, we can see that the number of images in the `original-dataset` that produce precision values lower than 0.9 increases significantly, while the number of images with recall values below 0.9 decreases significantly. This can be attributed to the fact that the false positive classifications in the middle one-sixth region of the image produce significantly worse performance in terms of precision, but the recall of the images improves since the number of clusters correctly classified as ground increases in the same region.

The author recommends more investigation into improving the classification conditions in future work regarding this algorithm. In the next section, we look at deep learning methods to directly classify each pixel using image processing neural networks.

3 Deep Learning-based Sky Detection

In this section, we look at the same problem of detecting the sky, but with a different approach. We can use deep learning methods, UNet models specifically, to segment an image into sky and ground segments. We then look at the performance of this method, compare it to the `baseline-mean-shift` method and its variants, and finally look at how this method can solve the original problem.

3.1 Deep Learning and U-Net Models

Deep learning techniques, particularly CNN [38], have revolutionized image segmentation[2] by providing highly accurate and efficient solutions. Deep learning is a subset of machine learning, which is a branch of artificial intelligence that configures computers to perform tasks through experience. It is based on artificial neural networks with representation learning and can be supervised, semi-supervised, or unsupervised. Deep learning algorithms use multiple layers to progressively extract higher-level features from raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify concepts relevant to a human such as digits, letters, or faces. Deep learning has been applied to various fields including computer vision, speech recognition, natural language processing, machine translation, bioinformatics, drug design, medical image analysis, and more. In many cases, it has produced results comparable to and even surpassing human expert performance[19].

UNet[50] is a very common model architecture used for pix-to-pix tasks such as image segmentation. The UNet architecture consists of a contracting path and an expansive path. The contracting path follows the typical architecture of a convolutional network, with repeated application of convolutions, Rectified Linear Unit (ReLU)[1] activations, and max pooling operations for downsampling. The expansive path consists of layers of upsampling of the feature map followed by a convolution that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two more convolutions each followed by a ReLU.

A key feature of the UNet architecture is that in the decoder part of the model, the inputs are not only the feature map from the previous layer but also a ”sideways” input from the encoder module. This allows the network to use information from high-level features, such as edges, angles, etc., and low-level features, such as larger objects, to make more accurate predictions. We can use this advantage to combine the utility of multiple features in an image to create a mask, that separates the sky from the ground, directly by first decoding the image into deeper layers, and then rebuilding the image as a mask separating the sky from the ground. In the next section, we detail the architecture used and the mathematical description of this method.

3.2 Architecture

The proposed UNet architecture consists of two main parts: the encoder (down-sampling path) and the decoder (up-sampling path). The encoder extracts features from the input image, while the decoder reconstructs the mask used to segment the image into sky and ground regions, using the learned features.

1. **Encoder** (Down-sampling Path): The first stage of the UNet model is the encoder, responsible for reducing the spatial dimensions of the input image. It comprises three down-sampling blocks, each containing a 2D convolutional layer, batch normalization[29], and ReLU activation. The number of filters in these blocks increases from `filter` to `filter*2` and then to `filter*4`, where `filter` is the number of filters of the first convolution layer that is given as input. This allows the model to capture features at different scales. Max-pooling[21] layers are employed to further down-sample the feature maps. Additionally, the feature maps obtained at each down-sampling stage are stored in the ”skips” list to be used during the up-sampling phase, which is the ”sideways” connection previously mentioned.

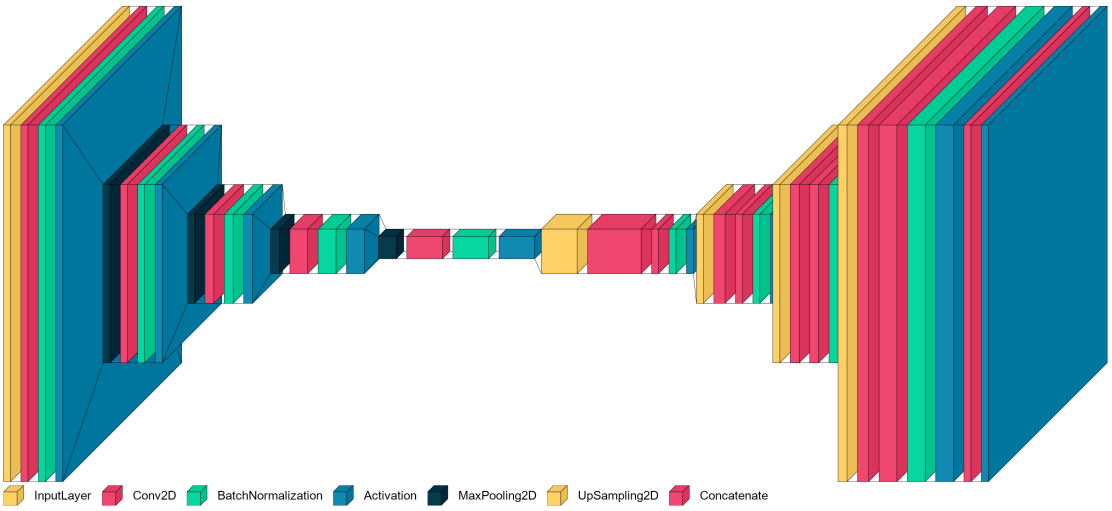

Figure 12: Architecture of the U-Net model

A diagrammatic representation of the architecture of the UNet model. Skip connections have not been visualized here due to limitations of `visual-keras`.

2. **Bottleneck**: Following the encoder, there is a bottleneck layer that aims to compress and capture essential features from the down-sampled data. This layer is composed of a single 2D convolutional layer with `(filters * 8)`, enabling it to represent crucial information in a more compact form.

3. **Decoder** (Up-sampling Path): The decoder stage is designed to recover the spatial information lost during the down-sampling process. It consists of three up-sampling blocks, with each block performing the reverse operation of the corresponding down-sampling block. The filter sizes, or the number of filters, decrease in the opposite manner `(filter*4, filter*2, filter)` as the decoder constructs the mask from the feature maps. For this purpose, up-sampling is done by a factor of 2 using up-sampling layers, and feature maps from the "skips" list, containing the output of the encoder "sideways" connections, are concatenated to the current feature maps, facilitating the recovery of fine-grained details. Each up-sampling block also includes a 2D convolutional layer, batch normalization, and ReLU activation.

4. **Final Layer**: The last part of the UNet model is the final layer, which produces the segmentation mask. It is implemented as a 2D convolutional layer with a kernel size of 1. The output of this layer undergoes a sigmoid activation function[46], generating pixel-wise probabilities representing the likelihood of each pixel belonging to the target class.

The diagrammatic representation of the architecture is depicted in Figure 12:

The optimal hyperparameter configuration for this UNet model was fine-tuned using the results of an extensive hyperparameter grid search[39], which is detailed in the following subsection. The author chose grid search as it was the least resource-intensive for the architecture used here. By systematically exploring various combinations of filters, kernel sizes, learning

rates, loss functions, and optimizers, they identified the hyperparameters that yielded the highest precision and recall.

3.3 Grid Search for Hyperparameter Tuning

3.3.1 Setup

Grid search in hyperparameter tuning is an exhaustive search technique used to find the optimal combination of hyperparameters for a machine learning model. It involves specifying a range of values for each hyperparameter and systematically evaluating all possible combinations. The approach aids in identifying the best configuration to enhance model performance and generalization. The author used a Python library called Tensorcross[52] to perform a grid search on the hyperparameters of the model, the results of which are presented as follows.

Following is a list of hyperparameters that affect the model performance:

1. **Filters**: Filters[38], or the filter size (`filter` as defined before) in a convolutional layer determine the number of feature maps produced by each convolutional layer when the kernel slides across the input image. Increasing the number of filters allows the model to learn more complex and diverse patterns in the data, potentially improving its ability to capture intricate features. However, a higher number of filters also increases the computational cost. During testing, the author used two filter sizes, 32 and 64.

2. **Kernel Sizes**: Kernel sizes[38] define the size of the filter that slides over the input data during the convolution operation. Larger kernel sizes can capture more global patterns, while smaller kernel sizes focus on local features. The author uses multiple kernel sizes, such as $[3, 5, 7]$, that allow the model to learn from different scales of information.

3. **Paddings**: Padding[38] determines how the input data is padded with zeros around its edges during convolution. 'Same' padding ensures that the output size remains the same as the input size, while 'valid' padding reduces the output size. 'Same' padding can help retain more spatial information during convolution. During testing, the author only used 'same' since the decoder requires the size of the output to be the same at each level.

4. **Losses**: Loss functions quantify the dissimilarity between predicted values and actual labels during training. The two loss functions we shall use are:

   - **Binary Crossentropy (BC) Loss ($L_{\mathbf{BCE}}$)[64]**:

     The proposed Binary Crossentropy loss function is defined as follows:

     $$L_{\text{BCE}} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{256}\sum_{k=1}^{256}\left[y_{i,j,k}\log(p_{i,j,k}) + (1-y_{i,j,k})\log(1-p_{i,j,k})\right]$$

     Where:

     - N is the total number of samples in the dataset.
     - $y_{i,j,k}$ represents the true binary label (0 or 1) for the i-th sample and spatial position (j, k).

– $p_{i,j,k}$ represents the predicted probability of the label of the pixel for the i-th sample and spatial position (j, k).

The loss function computes the negative average log-likelihood of the true labels under the predicted probabilities. It penalizes the model when the predicted probability is far from the true label (i.e., $p_{i,j,k}$ close to 0 when $y_{i,j,k} = 1$) and the true negative class (i.e., $p_{i,j,k}$ close to 1 when $y_{i,j,k} = 0$). This is possible since the segmentation only provides us with two labels for each pixel.

- **Mean Squared Error (MSE) Loss (L_{MSE})[61]**:

  The proposed Mean Squared Error loss function is defined as follows:

$$L_{\text{MSE}} = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{256} \sum_{k=1}^{256} (y_{i,j,k} - \hat{y}_{i,j,k})^2$$

  Where:

  – N is the total number of samples in the dataset.

  – $y_{i,j,k}$ represents the true label for the i-th sample and spatial position (j, k).

  – $\hat{y}_{i,j,k}$ represents the predicted label for the i-th sample and spatial position (j, k).

  The loss function computes the squared difference between the true value $y_{i,j,k}$ and the predicted value $\hat{y}_{i,j,k}$ for each pixel position. It aims to minimize the average number of misclassifications, encouraging the model to produce more correct classification values.

5. **Optimizers**: The optimizer aims to find the optimal combination of model parameters to minimize the chosen loss function and improve model performance. Two commonly used optimizers are:

   - **Adam Optimizer**[35]:

     The Adam optimizer adapts the learning rate for each parameter based on the first ($m_{t,i,j,k}$) and second ($v_{t,i,j,k}$) moments of the gradients for the model parameters $\theta_{i,j,k}$ at position (i, j, k). The use of adaptive learning rates in the Adam optimizer allows for faster convergence and more robust training, making it a popular choice for deep learning tasks.

     These moments are defined as follows:

$$m_{t,i,j,k} = \beta_1 \cdot m_{t-1,i,j,k} + (1 - \beta_1) \cdot \nabla\theta_{i,j,k}$$

$$v_{t,i,j,k} = \beta_2 \cdot v_{t-1,i,j,k} + (1 - \beta_2) \cdot (\nabla\theta_{i,j,k})^2$$

     Where $\nabla\theta_{i,j,k}$ represents the gradient of the loss with respect to the parameter $\theta_{i,j,k}$, and β_1 and β_2 are hyperparameters controlling the exponential decay rates of the moment estimates. The Adam optimizer then uses these moment estimates to update the learning rate for each parameter, resulting in efficient and adaptive gradient-based optimization.

The update rule for the model parameters using the Adam optimizer is defined as follows:

$$\theta_{t+1,i,j,k} = \theta_{t,i,j,k} - \frac{\text{learning\_rate}}{\sqrt{v_{t,i,j,k}} + \epsilon} m_{t,i,j,k}$$

Where:

- $m_{t,i,j,k}$ represents the first moment of the gradient at time step t for the model parameters $\theta_{i,j,k}$.
- $v_{t,i,j,k}$ represents the second moment of the gradient at time step t for the model parameters $\theta_{i,j,k}$.
- $\theta_{t,i,j,k}$ are the model parameters at at time t.
- ϵ is a small constant for numerical stability.

The algorithm adapts the learning rate individually for each parameter, allowing it to converge faster compared to traditional gradient descent methods.

- **Stochastic Gradient Descent (SGD)**[51]:

  The Stochastic Gradient Descent optimizer updates model parameters based on the average gradient of a small batch of data points. The learning rate, denoted as learning_rate, controls the step size during parameter updates.

  The update rule for the model parameters using SGD is defined as follows:

$$\theta_{t+1,i,j,k} = \theta_{t,i,j,k} - \text{learning\_rate} \cdot \text{gradient}_{i,j,k}$$

  Where $\text{gradient}_{i,j,k}$ represents the gradient of the loss with respect to the model parameters $\theta_{i,j,k}$ at position (i, j, k).

  SGD updates the parameters in the direction that reduces the loss, iteratively moving towards the optimal solution. The learning rate learning_rate controls the step size, determining the magnitude of parameter updates at each iteration.

By systematically exploring the hyperparameter space, encompassing filters [32, 64], kernel sizes [3, 5, 7], paddings ['same'], learning rates [0.001, 0.0001], losses ['binary_crossentropy', 'mean_squared_error'], and optimizers [Adam, SGD], we can ascertain the most effective combination for optimizing model performance and generalization. The empirical evaluation of the grid search algorithm gave the author the optimal configuration for the UNet model, as presented in the following section.

3.3.2 Results

The author varied the hyperparameters, including filters, kernel sizes, learning rates, loss functions, and optimizers, to identify the optimal configuration for model performance. Due to resource constraints, they only tested for one epoch with 10,000 shuffled images from the `original-dataset`. The images were kept the same for each iteration of the grid search algorithm. Since the comparison for each configuration was performed with the same samples, we can compare the metrics for each configuration. We can skip the padding parameter since all test instances utilize the *same* padding parameter, owing to the fact that other padding

Filters	Kernel Size	LR	Loss Function	Optimizer	Loss	Precision	Recall
32	7	10^{-3}	BC	SGD	0.117	**0.979**	0.929
64	5	10^{-4}	MSE	Adam	0.019	0.977	0.971
64	3	10^{-3}	BC	Adam	0.104	0.971	0.955
32	3	10^{-3}	BC	Adam	0.097	0.971	0.960
64	3	10^{-3}	MSE	Adam	0.025	0.971	0.960
64	7	10^{-4}	BC	Adam	0.078	0.969	**0.973**
32	7	10^{-3}	MSE	Adam	0.032	0.968	0.942
64	5	10^{-3}	BC	SGD	0.090	0.967	0.967
64	5	10^{-3}	BC	Adam	0.173	0.967	0.898
32	5	10^{-4}	MSE	Adam	**0.018**	0.967	0.984

Table 2: Results of Hyperparameter Grid Search

Performance comparison of various hyperparameter configurations of the UNet model. The table highlights the influence of filters, kernel sizes, learning rates, loss functions, and optimizers on key metrics, including loss, precision, and recall. Bold values in 'Loss,' 'Precision,' and 'Recall' represent the highest performance in each respective column. Note the abbreviation Learning Rate (LR).

types interfere with the skip connections during execution. The results are presented in table 2

From the results, we observe that the configuration with 32 filters, a kernel size of 7, and a learning rate of 0.001 achieved the highest precision of 0.979 and a recall of 0.929. On the other hand, the configuration with 64 filters, a kernel size of 5, and a learning rate of 0.0001 achieved the highest recall of 0.973. The loss function and optimizer choice also influenced the model's performance, with binary cross-entropy and SGD leading to the highest loss value. At the same time, "mean squared error" and Adam optimizer resulted in a lower loss and better precision and recall. We can take the configuration with the higher precision value since the original problem of sky segmentation requires precisely classifying the sky pixels and reducing the false positive numbers.

In the next section, we look at using the gained insights ascertained from the optimal hyperparameters for the training of the UNet model and the final results for the sky segmentation problem.

3.4 Training

During the training process, the author used the `original-dataset` of 38,000 samples to train a deep learning model for sky segmentation[43]. The `original-dataset` was divided into three subsets: a training set containing 30,000 samples, a validation set with 2,000 samples, and a test set with 6,000 samples. We assume the objective to be the same as before: we feed an input image to a model that produces an output 2D mask where 1 corresponds to a sky pixel and 0 corresponds to a ground pixel. We define the precision and recall for an image as described in the previous section.

The model was trained using the fit method from Keras, with a total of 5 epochs. These particular number of epochs were chosen after some trial-and-error runs, showing that beyond 5 epochs, the model tends to overfit. After each epoch, the model's progress was saved using

a callback (`ModelCheckpoint`) to ensure that the best-performing model was retained. The training process took advantage of multiprocessing by setting `use_multiprocessing=True` and employing four workers for enhanced efficiency. Finally, the author utilized the Keras GPU training methods to train the model on an RTX3070 GPU with CUDA[45] and CuDNN[13] for faster train times.

Throughout the training, the model's performance was evaluated on both the training and validation datasets. During the first epoch, the model achieved a training loss of 0.0467, a precision of 0.9787, and a recall of 0.9851. On the validation set, the corresponding metrics were 0.0867 for loss, 0.9665 for precision, and 0.9776 for recall. Over subsequent epochs, the model continued to improve its performance, reducing the training loss to a minimum value of 0.0063, achieving a maximum training precision of 0.9972, and a maximum recall of 0.9976. However, it is worth noting that the validation loss increased slightly to 0.1391, with precision at 0.9512 and recall at 0.9865 during the final epoch, indicating slight overfitting.

The results indicate that the model exhibits strong predictive capabilities, as evidenced by high precision and recall values on both training and validation datasets. However, the increasing validation loss during the training process may indicate the potential for overfitting. This occurs since the dataset being used is a repeated set of images of the same location and perspective during different times, creating less variation, which reduces the overall information the model can infer from the images.

3.5 Testing

The author tested the model on a dataset of 6000 images that were a subset of the `original-dataset`, where they achieved a mean precision of 0.964 and a mean recall of 0.987. The mean time taken for each image to process through the model was 0.072 seconds on the same hardware[2], and the number of images with either very low precision or very low recall was only 6. The author also tested the same test set on the `baseline-mean-shift-top-third` algorithm to compare the results, which give us a mean precision of 0.962, a mean recall of 0.899, and a mean time taken of 0.72 as well. We infer from this that the model performs similarly in terms of precision but significantly better in terms of recall. We can see the plot of the distribution of precision and recall for both, the UNet model and the `baseline-mean-shift-top-third` algorithm on the same test dataset to compare their performance. The plotted results are present in Figure 13 and table 3.

We look at some real-world examples and compare the two algorithms visually in order to see how well they perform side-by-side in Figure 14. Since the images being used for this result are not part of the original training or test dataset, we can say that the model can perform well on completely unknown data as well.

We also look at the 6 failure cases, i.e., where the recall values are below 0.2, in order to visually explore the reasons for the low performance. We infer from Figure 15 that the failure cases consist of images that are either extremely noisy, are of nighttime, or do not contain a clearly defined sky region for the model to classify.

[2]AMD Ryzen 9 5900HX, RTX3070 Mobile GPU, 16GB of DDR4 RAM and 1TB of NVMe SSD, using Windows 11 64-bit

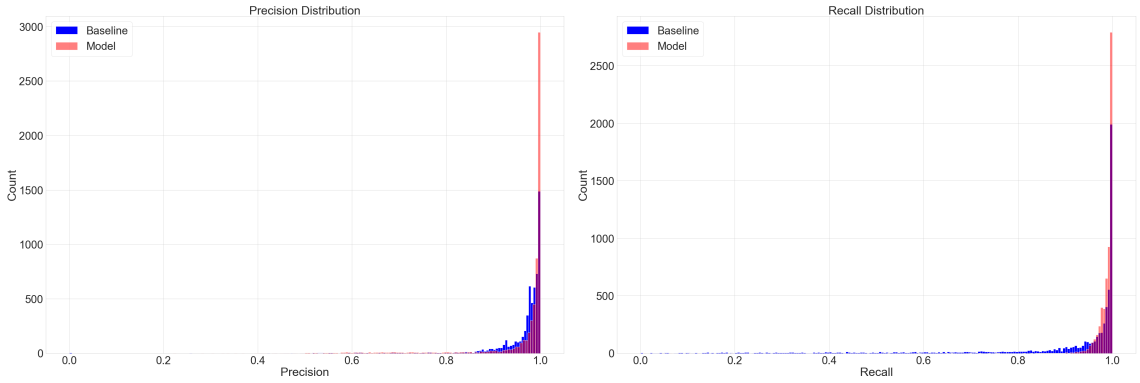

Figure 13: Comparison of metrics between `baseline-mean-shift-top-third` and UNet

Distribution of precision and recall for UNet model and `baseline-mean-shift-top-third` algorithm for 6000 images testing sample of `original-dataset`. The X-axis represents the metric and the Y-axis represents the number of values in a bin of width 0.005.

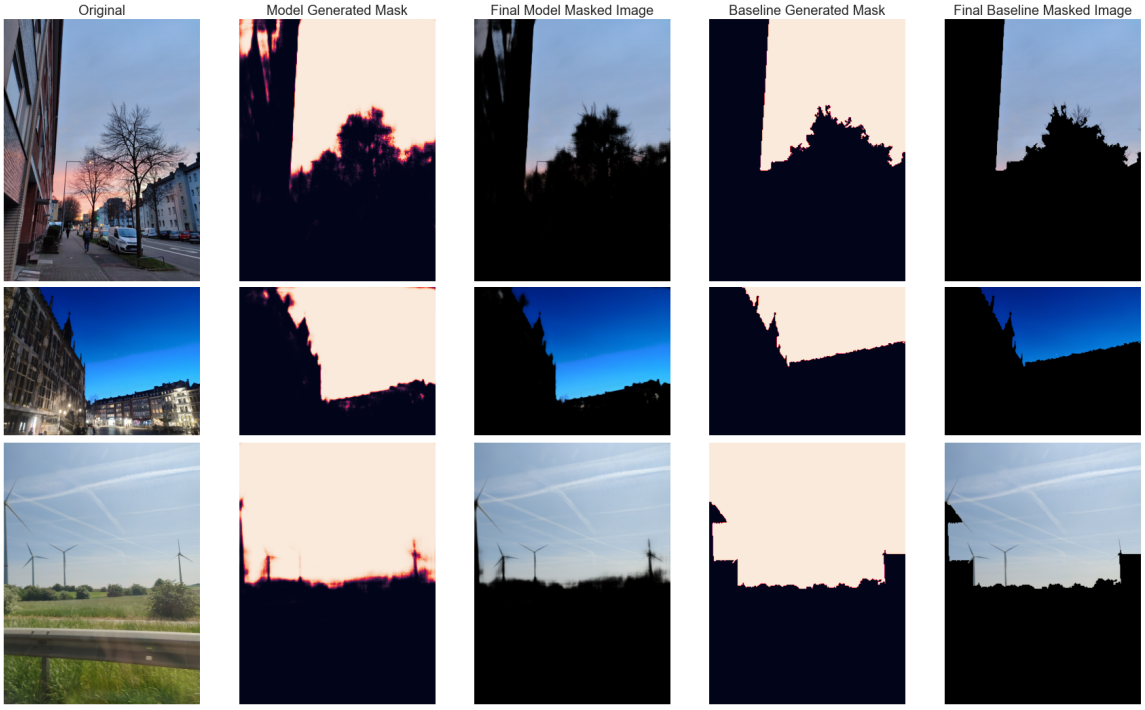

Figure 14: Comparison of Segmentation Output of `baseline-mean-shift-top-third` and UNet

Three examples of mask generation by the `baseline-mean-shift-top-third` algorithm and the UNet model. Note that these images are not part of `original-dataset` and were captured by the author.

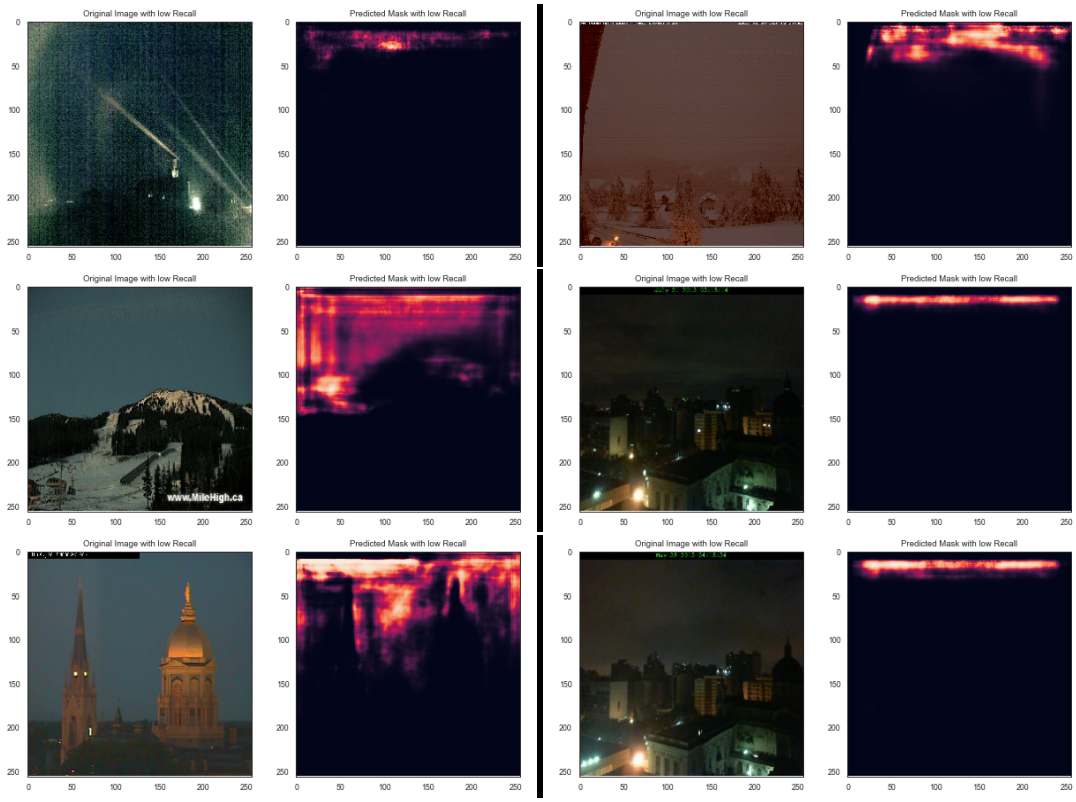

Figure 15: Failure Cases for UNet Model

All failure cases that give recall values below 0.2, tested on a 6000 sample from `original-dataset`. Note that both these images are extremely noisy and do not contain a clearly defined sky region.

	Mean Precision	Mean Recall
`baseline-mean-shift`	0.955	0.891
`baseline-mean-shift-augmented`	0.911	0.940
`baseline-mean-shift-top-third`	0.962	0.899
`baseline-mean-shift-augmented-top-third`	0.919	0.961
UNet Model	**0.964**	**0.987**

Table 3: Comparison of Variations of `baseline-mean-shift` Algorithm with UNet Model

Performance comparison of different variations of the `baseline-mean-shift` segmentation algorithm with the UNet model on 6000 images from `original-dataset` in terms of Mean Precision and Mean Recall. The variations include "`baseline-mean-shift`," "`baseline-mean-shift-augmented`", "`baseline-mean-shift-top-third`", and "UNet Model"

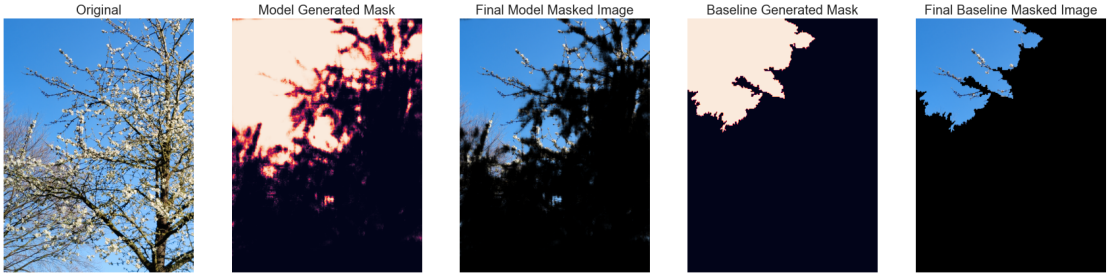

Figure 16: UNet Model and `baseline-mean-shift-top-third` on trees

Comparison of UNet model and `baseline-mean-shift-top-third` on images that have trees in them.

3.6 Fine Tuning

We now have a model that generalizes well on landscape data to segment the sky from the ground. However, upon further analysis of `original-dataset`, we can see that the dataset lacked samples that represent certain objects that people encounter in their daily lives, particularly trees. This can be seen in Figure 16. Since trees possess minute details and are intricate in design, the model is unable to differentiate well between the sky and ground pixels. Another reason for this is that the `original-dataset` does not contain any trees in its images, hence the trained model does not understand the representation of trees. To solve this problem, the author looked at previous work done on this topic by Geuenich et al. [20]. In particular, they borrowed the dataset from their work, which contains 62 labeled images of landscapes around Aachen, Germany, and contains trees as part of the landscape. We shall name this dataset `bachelor-dataset`, as mentioned in Section 2.3.2, since this dataset belongs to the work done during their bachelor thesis.

Since the amount of data available to the author was small compared to the 38,000 images of `original-dataset`, the author decided to use Fine Tuning[4] as a method to introduce this data into the pre-trained model. In particular, they took the entire network and trained it on this new dataset for multiple epochs to fine-tune the model toward samples that were missing in the `original-dataset`.

We look at three different epoch levels for training the model, starting from 1, 5, and then 10. We can use the model that was pre-trained using the optimized hyperparameters. The

results of this training give us a training loss of 0.351, a mean training precision of 0.934, and a mean training recall of 0.910 for the 1 epoch variant.

Due to the paucity of data in this case, the author chose to feed the entire dataset to the model instead, without separating any training, validation, or test sets. They re-evaluated the 6000 image dataset to be able to compare our results with the original non-fine-tuned models, in order to compare the difference between the mask generation capabilities of both models. We also look at a comparison of the performance of the non-fine-tuned model, as well as the fine-tuned models on the `bachelor-dataset`. The results of the 1, 5, and 10 epoch variants on the 6000 image testing sample of `original-dataset` are plotted in Figure 17 and table 4. The table also contains a comparison of the variants of the fine-tuned model with the original non-fine-tuned model. A visual example is also available in Figure 18. It can be observed that the model still makes some errors in the images, particularly in the bottom region of the image.

Model	`original-dataset` **6000**		**bachelor-dataset**	
	Mean Precision	**Mean Recall**	**Mean Precision**	**Mean Recall**
1 Epoch Fine-Tuned	0.868	**0.994**	0.930	0.860
5 Epoch Fine-Tuned	0.853	0.989	0.934	0.928
10 Epoch Fine-Tuned	0.862	0.988	**0.941**	**0.937**
non-fine-tuned	**0.964**	0.987	0.925	0.680

Table 4: Mean Precision and Recall for Fine-Tuned Model using 1, 5 and 10 Epochs along with non-fine-tuned model

Mean Precision and Mean Recall for Fine-Tuned Models for 1, 5, and 10 Epochs on the 6000 image test set of `original-dataset` and complete `bachelor-dataset`

Inferring from the data, it's clearly visible that a degree of Catastrophic Forgetting[33] takes place on the pre-trained neural network. The network weights change beyond the capacity of the original network to understand the `original-dataset`, creating an overall worse solution. However, it can be noted that the fine-tuned network does perform well on the task of segmented images with trees, as seen in Figure 18. In this scenario, the author chose to keep the network as it is, since the overall metrics were better without the new dataset. Although the models presented in this section can be switched later depending on the task at hand, the author recommends future investigation for better fine-tuning methods for improved performance.

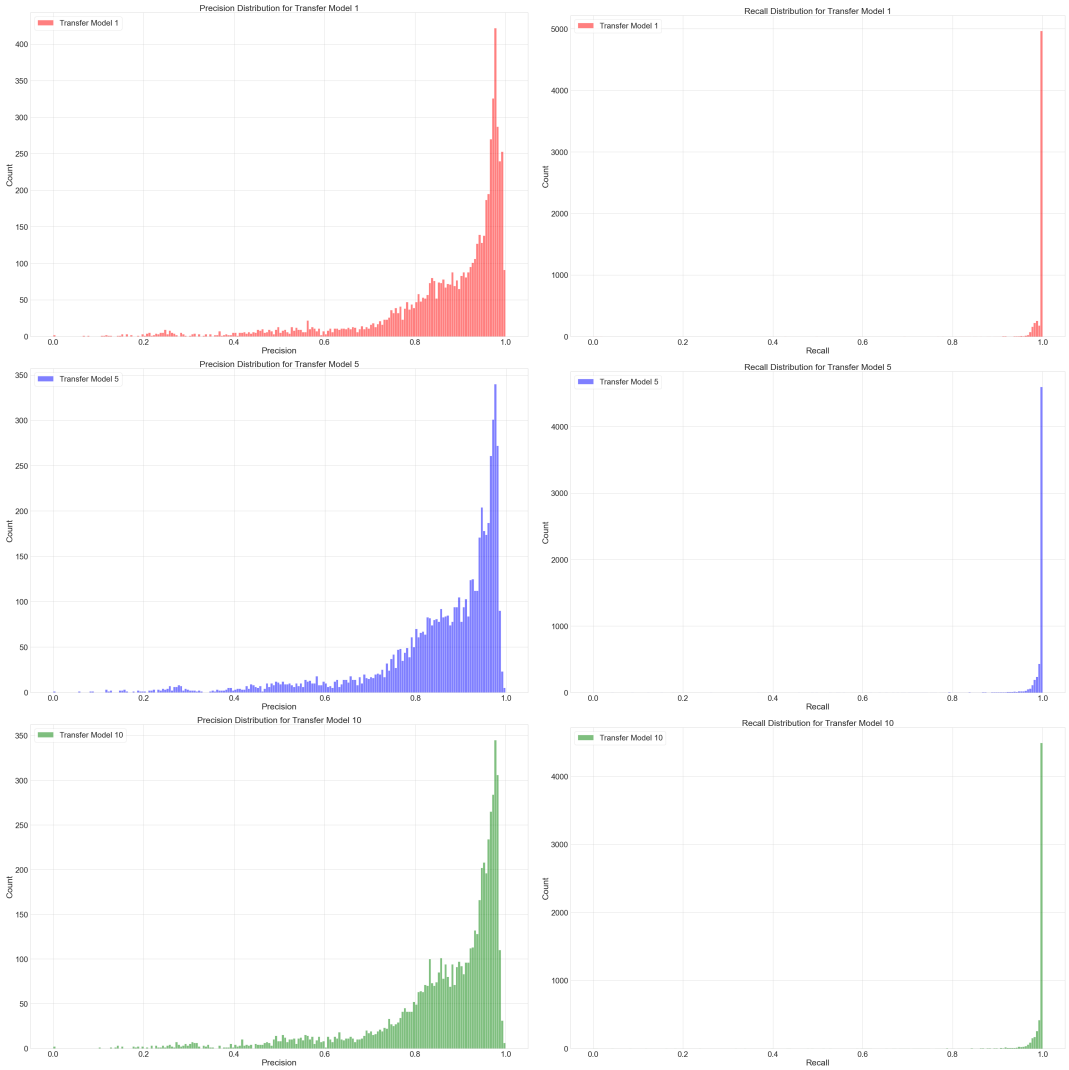

Figure 17: Comparison of Distribution of Precision and Recall for Fine-Tuned Model using 1, 5 and 10 Epochs

Distribution of precision and recall for fine-tuned UNet model for 1, 5, and 10 epochs of training on the 6000 image test dataset of `original-dataset`. The X-Axis represents the metric and the Y-Axis represents the number of values in a bin of width 0.005.

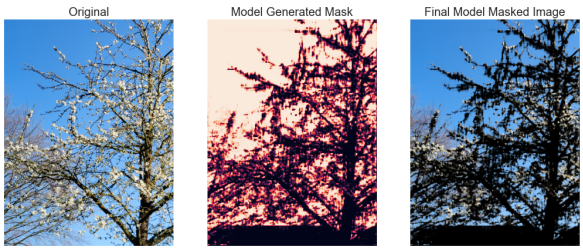

Figure 18: Example of UNet Model after Fine-Tuning

An example of the UNet model trained for 1 epoch on the `bachelor-dataset`.

4 Depth Estimation

In this section, we look into the depth estimation problem to produce a depth map to combine with the initial sky segmentation approaches. We can consider the two sub-types of depth estimation, that is, absolute depth estimation and relative depth estimation. We respectively look at these two subtypes and then try to solve the challenge in the context of the occlusion map creation. Since we have already looked at solutions for the sky segmentation, in this section we look at the remaining "ground" image in order to provide the aforementioned occlusion map. The author asserts that the sky region that was computed in the previous sections will always be behind any object placed in the viewfinder of a device, hence we now only consider the differentiation of the ground pixels into foreground and background with respect to the placed virtual object. In this sense, we can use depth estimation to calculate the depth of each pixel in the ground region, which we can use to estimate the position of these pixels with respect to the virtual object, thereby producing the occlusion map by simply labeling the image's foreground or background relative to the said object. We first start by looking at absolute depth estimation and then move on to relative depth estimation.

4.1 Absolute Depth Estimation

Absolute depth estimation involves the precise determination of distances from the camera to objects in a scene. Absolute depth estimation necessitates establishing real-world measurements, which inherently encapsulate challenges arising from scene scale, camera calibration, and varying lighting conditions[3]. This complex problem is compounded by the fact that images alone lack sufficient cues for direct depth inference, leading to an ill-posed and highly under-constrained task.

Monocular Absolute depth estimation is exceedingly intricate due to its reliance on establishing physical measurements from singular 2D images. Numerous factors, including occlusions, ambiguous textures, and perspective variations, contribute to the inherent difficulty of this problem, which are hard to resolve without additional information in the form of binocular images or non-image distance measure. Since our original problem statement must deal with deriving occlusion maps using edge devices, particularly smartphones, we are constrained to only have monocular images. Traditional methodologies, while valuable, struggle to capture the nuanced relationships between image features and depth information using singular images.[47].

For our problem of providing an occlusion map, we have an even more specific set of constraints we must look at, namely outdoor landscape absolute depth estimation. This provides more challenges since depth information at a distance can become inaccurate[47]. This poses problems for any network to accurately estimate the depth for all distances. At the time of writing, most available solutions to the problem implement absolute depth estimation for indoor scenes[63], [31] or short distance (distance less than 400m) absolute depth estimation [41].

One of the solutions for long-range monocular absolute depth estimation was proposed by Polasek et al [47], which uses a mix of 10 datasets containing 1 million images to train a modified Vision Transformer[15] to predict the absolute depth of long-distance images. Due to the sheer size of the dataset and computational resource requirements stated in the original paper, the author of this thesis was unable to replicate their solution on a local machine and hence leaves the application of their model as a future task. At the time of writing, the author

was also unable to contact Polasek et al. for requisitioning their model for the purposes of this thesis.

Since direct absolute depth estimation can be challenging, as described above, we rather move towards producing the occlusion map using a different method: we first produce a relative depth map and try to infer the absolute depth of the objects using additional knowledge. The concept, implementation, and results are explained in the following sections.

4.2 Relative Depth Estimation

We move on to solve the problem of monocular relative depth estimation instead, which has a wider variety and quantity of literature and contains more data to work with. Relative Depth Estimation is the problem of producing pixel-level depth information for an image based on either just the image or additional information, where the pixels are ranked based on their relative distance to each other[34]. Monocular Relative Depth Estimation introduces the challenge of doing relative depth estimation using single images, thereby reducing the amount of information at hand to calculate the relative distances.

We look at relative depth estimation as a solution to providing the occlusion map since we are trying to solve the problem of computing which pixels are foreground and which ones are background. We assume that if we can obtain the relation between pixels, and then obtain the absolute measure for certain pixels, we can combine this information to produce an absolute depth map. This assumption comes from the work done by Masoumian et al[41] who also look at producing relative depth maps and then estimating the position of objects using deep learning-based object detection, and then combining this information to provide absolute distance measure. They performed this task for indoor or short-distance scenes, hence this approach for longer distances is still unproven. The author of this thesis only looked at the relative depth estimation part of this problem, leaving the absolute distance of objects as a future proposed task for further investigation. To solve the problem of depth predictions, we look at a pre-existing solution called Depth Prediction Transformer (DPT) which uses vision transformer[15] to predict the relative depth map. We can utilize an existing solution since the computational constraints of training a similar model yielded no immediate benefits over the use of pre-trained work in this case. We first start by explaining transformers, and then look at it's usage in the context of depth prediction.

4.3 Depth Prediction using Transformers

In this section, we look at a pre-packaged solution to the depth estimation problem using Depth Prediction Transformers[49]. We first start with explaining transformers, then processing to vision transformers, and finally how we can use vision transformers for depth estimation using DPT.

4.3.1 Transformers

The Transformer[57] architecture, initially introduced in the context of natural language processing, has garnered widespread recognition as a pivotal milestone in representation learning. Since natural language processing requires sequences of words, these models can be referred to as sequence-to-X, where X can be any desired outcome. In the context of transformers, we want the outcome to be another sequence, since we are trying to generate output

that is similar to input or trained data[56], which are traditionally computed using recurrent layers[54]. The architectural paradigm of transformers, founded on the self-attention mechanism, fundamentally diverges from conventional sequence-to-sequence models by dispensing with recurrent or convolutional layers. In our case, we are interested in utilizing the approach of transformers by using patches of a single image as the input sequence, while producing an output sequence to solve the desired task.

At the heart of the Transformer are Multi-Headed Self-Attention (MHSA) modules[57], which serve as the core part of the approach for capturing relationships between elements within a sequence. We look at the computation at the self-attention layer in the following paragraphs. We can then use this information to understand how transformers work in the later context of images and then combine this approach with the sky segmentation algorithms. We shall start by looking at the explanation of MHSA using the general term "token" first.

Let $\boldsymbol{X}$ be a sequence of tokens (which are later replaced with patches of an image), where each token is represented as a vector. Therefore, the shape of $\boldsymbol{X}$ is (n, d), where: Let n be the number of tokens in the sequence. Let d be the dimensionality of each token vector.

The self-attention mechanism computes a weighted sum of values based on attention scores. Given an input sequence $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$, where the shape of $\boldsymbol{X}$ is (n, d), n is the number of tokens in the sequence, d is the dimensionality of each token vector. the self-attention mechanism computes attended output $\boldsymbol{Y}$ based on attention scores $\boldsymbol{A}$.
Linear projections yield $\boldsymbol{Q}$, $\boldsymbol{K}$, and $\boldsymbol{V}$, which are obtained by projecting the input sequence $\boldsymbol{X}$ using learnable weight matrices $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$, and $\boldsymbol{W}_V$:

$$\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W}_Q, \quad \boldsymbol{K} = \boldsymbol{X}\boldsymbol{W}_K, \quad \boldsymbol{V} = \boldsymbol{X}\boldsymbol{W}_V$$

Where the shape of each element is as follows: $\boldsymbol{W}_Q$: (d, d_q), $\boldsymbol{Q}$: (n, d_q), $\boldsymbol{W}_K$: (d, d_k), $\boldsymbol{K}$: (n, d_k), $\boldsymbol{W}_V$: (d, d_v), and $\boldsymbol{V}$: (n, d_v). The specific values of d_q, d_k, and d_v depend on the design of the self-attention mechanism and the model architecture.

(d_q, d_k, d_v) refer to the dimensions of the transformed representations of tokens within the input sequence. These dimensions play a crucial role in determining how the attention mechanism operates and captures relationships between tokens.

1. **Query**: The query Q is a transformed representation of an element in the input sequence. Informally, it's used to compare against the other elements to assess their relevance, and in simpler terms, it provides weight to tokens that might be "relevant". d_q represents the dimensionality of the transformed query representations. When the input tokens $\boldsymbol{X}$ are projected using the weight matrix $\boldsymbol{W}_Q$, the resulting $\boldsymbol{Q}$ matrix has a shape of (n, d_q), where n is the number of tokens in the sequence.

2. **Key**: The key K is another transformed representation of an element in the input sequence. Informally, it's used to determine the similarity or compatibility between elements, and in simpler terms, it provides information on the salience of a token. d_k denotes the dimensionality of the transformed key representations. After projecting the input tokens $\boldsymbol{X}$ using the weight matrix $\boldsymbol{W}_K$, the resulting $\boldsymbol{K}$ matrix has a shape of (n, d_k).

3. **Value**: The value V is yet another transformed representation of an element in the input sequence. Informally, it holds the information that we want to use when combining elements based on their attention scores, and in simpler terms, it contains the semantic

information about the query against the keys. d_v represents the dimensionality of the transformed value representations. Projecting the input tokens $\boldsymbol{X}$ using the weight matrix $\boldsymbol{W}_V$ yields the $\boldsymbol{V}$ matrix with a shape of (n, d_v).

This mechanism allows the transformer to map the queries to the corresponding values based on the keys provided. Hyperparameter tuning techniques can be employed to find optimal values of (d_q, d_k, d_v) that yield the best performance on a validation set.
Attention scores $\boldsymbol{A}$ are derived from the dot product of $\boldsymbol{Q}$ and $\boldsymbol{K}$, scaled by the reciprocal of the square root of the dimension of key vectors d_k:

$$\boldsymbol{A} = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)$$

Here, $\boldsymbol{Q}\boldsymbol{K}^T$ represents the dot product between $\boldsymbol{Q}$ and the transpose of $\boldsymbol{K}$, and the shape of $\boldsymbol{A}$ is (n, n).
Attended output $\boldsymbol{Y}$ is obtained by taking the weighted sum of values $\boldsymbol{V}$, where the weights are determined by the attention scores $\boldsymbol{A}$:

$$\boldsymbol{Y} = \boldsymbol{A}\boldsymbol{V}$$

This operation computes a weighted average of values, where the weights are determined by the attention scores. The resulting $\boldsymbol{Y}$, of shape (n, d_v), encodes contextual information from the entire input sequence, facilitating the model's ability to capture relationships and dependencies.

The final training optimization equation for the attention layer comes out to be:

$$\text{minimize} \quad \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}(\boldsymbol{A}, \boldsymbol{V}, \boldsymbol{Y})$$

Where $\mathcal{L}(\boldsymbol{A}, \boldsymbol{V}, \boldsymbol{Y})$ is the task-specific loss function, comparing the attended output $\boldsymbol{A}\boldsymbol{V}$ to the ground truth label $\boldsymbol{Y}$. Finally, the concatenation layer in the multi-headed attention module is defined as:

$$\boldsymbol{MultiHead(Q, K, V)} = \boldsymbol{Concat}(\boldsymbol{H_1}, \boldsymbol{H_2}, \ldots, \boldsymbol{H_h}) \times \boldsymbol{W_o}$$

Where the shape of the final output is of the (n, n), $\boldsymbol{W_o}$ is the weight matrix for the final linear layer of shape (hd_v, n), $\boldsymbol{H_j}$ is the output of the scaled dot-product attention for the jth layer, and the shape of the final output $\boldsymbol{MultiHead(Q, K, V)}$ is (n, n).

The overall structure of the Transformer can be described in Figure 19, where each component is described as follows: Inputs consist of sequences of data, patches of images in our case, represented as tokens. Input embeddings convert these tokens into continuous vectors to capture semantic relationships. Positional embeddings are added to account for token positions. The Add & Norm operation, applied after sub-layers, adds residual connections and normalizes intermediate values. Feed-forward networks capture complex token interactions. Multi-Head Attention allows simultaneous focus on different parts of the input sequence. Masked Multi-Head Attention prevents attending to future positions during training. Output embeddings transform decoder outputs into the input embedding dimension. During training, outputs are shifted right to predict the next token based on previous ones. Softmax

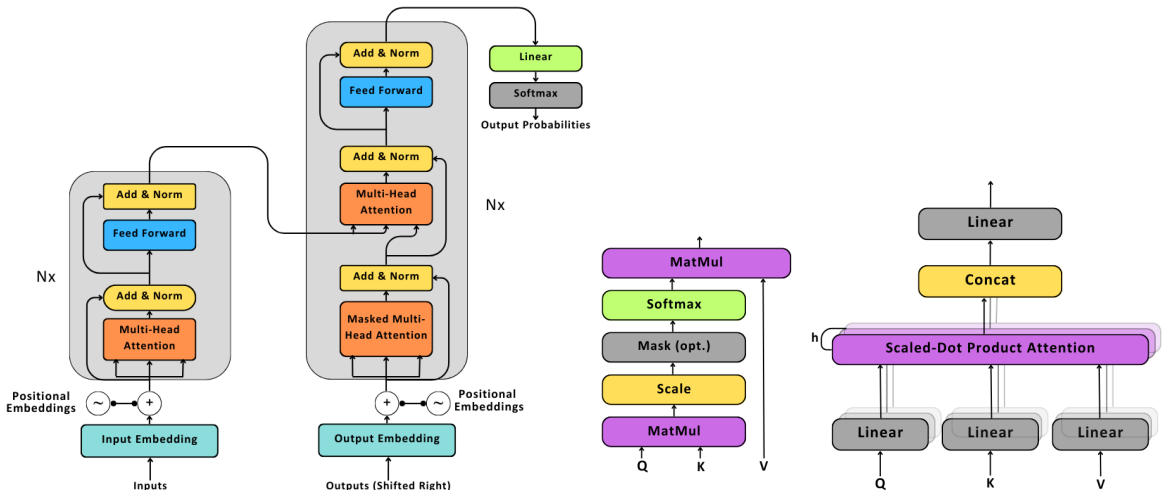

Figure 19: Diagrammatic Explanation of Transformer Attention Mechanism

(Left) The architecture of a single transformer layer, (middle) Scaled Dot-Product Attention Mechanism, (Right) Multi-Head Attention [57]. Note, that N is the number of transformer modules, and h is the number of layers.

computes token probabilities, forming a distribution. The output probabilities assign likelihoods to each token in the vocabulary, with the highest probability indicating the predicted next token.

Additionally, we can also see from Figure 19 that multi-head attention is comprised of h layers of scaled-dot product attention, which has been defined as the attention mechanism. Here, h is the number of Scaled-Dot-Product attention layers in a Multi-head Attention layer.

In summary, the Transformer architecture is made up of encoder and decoder components, each containing multiple self-attention layers. These layers include position-wise feedforward neural networks and layer normalization. Importantly, the Transformer allows each token to have a wide context range across the entire input sequence. The decoder adds masked self-attention for autoregressive generation during inference. This explanation is brief, and the author would recommend the original work for a more detailed explanation of the workings of a transformer.

4.3.2 Vision Transformers

The application of Transformers in computer vision has resulted in the Vision Transformer (ViT) [15]. ViT converts 2D image data into sequences of "visual tokens," providing tokenized representations for image patches. These representations are projected into embeddings and processed through self-attention layers. ViT uses patch embeddings to organize images into structured visual tokens. Importantly, ViT maintains token counts across transformations, preserving spatial information across layers. This leads to a wide-reaching global perspective, allowing ViT to understand the context better.

Additionally, ViT incorporates position embeddings, combined with image embeddings, enhancing token representations with spatial awareness. This combined embedding imparts positional understanding to tokens, enabling the model to perceive spatial relationships in images. The overview of the architecture can be seen in Figure 20

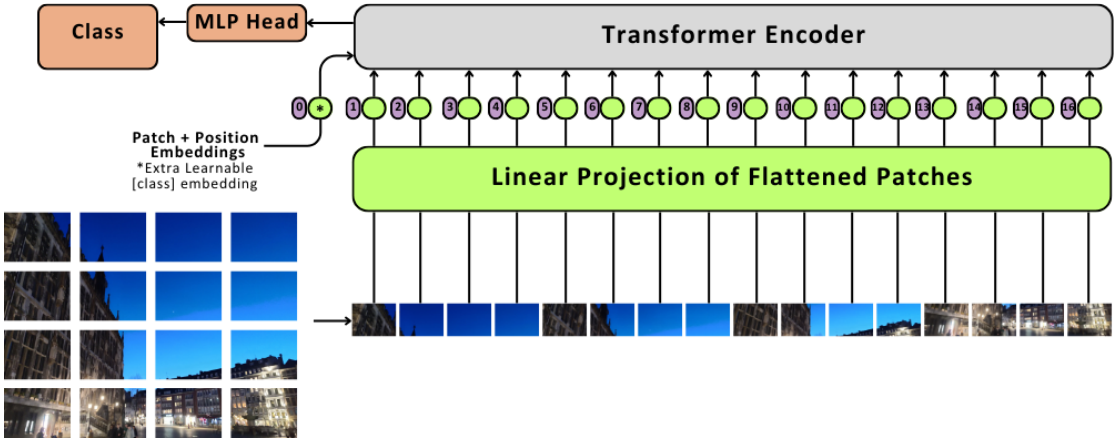

Figure 20: Diagrammatic Explanation of the Vision Transformer

Model overview of the ViT architecture as described by the paper by Dosovitskiy et al [15]. The transformer encoder works the same as the original transformer[57] encoder.

We shall keep this explanation brief since the core concepts of Vision Transformers work on the original Transformer are explained in detail in the previous section. We look at a pre-trained ViT model in the next section, since our final approach uses DPT directly.

4.3.3 Depth Prediction Transformer

The Depth Prediction Transformer (DPT) [49] offers a solution for relative depth estimation in computer vision. It builds upon the dense vision transformer framework, incorporating both encoders and decoders to achieve accurate dense predictions in a pix-to-pix methodology.

DPT relies on a vision transformer (ViT) at its core, which processes structured image segments, similar to how words are handled in text. These segments, whether individual patches or deep features are transformed into a feature space, resembling the meanings of words. Through sequential multi-headed self-attention (MHSA) operations [57], the relationships between these segments are clarified, turning image patches into geometrically intuitive representations suitable for relative depth estimation. Unlike traditional convolutional architectures, where receptive fields grow hierarchically through the layers, transformers maintain a consistent way of understanding tokens, preserving spatial information.

ViT's essence lies in its embedding of discerned image patches, projecting them into essential "tokens." This procedure maintains spatial resolvability over epochs, which is crucial for precise depth estimation. The global MHSA mechanism extends the receptive field beyond spatial limits. DPT's architecture draws heavily from ViT's but diverges at a crucial point since it must produce a depth map, which is also an image. Hence, it uses the encoder-decoder structure similar to that of a UNet.

- **Transformer Encoder:** The core of DPT's architecture is the Transformer Encoder. It employs the ViT framework to process input image data. ViT transforms image patches into structured "visual tokens" through linear embeddings, allowing them to be treated as sequential data. The process involves projecting input sequences $\boldsymbol{X} = (x_1, x_2, \ldots, x_n)$ into query ($\boldsymbol{Q}$), key ($\boldsymbol{K}$), and value ($\boldsymbol{V}$) matrices using linear

transformations $\boldsymbol{W}_Q$, $\boldsymbol{W}_K$, and $\boldsymbol{W}_V$. Attention scores ($\boldsymbol{A}$) are then calculated based on the dot product of $\boldsymbol{Q}$ and $\boldsymbol{K}$, normalized by $\frac{1}{\sqrt{d_k}}$ where d_k is the dimension of the key vectors. These scores guide the weighted aggregation of values, resulting in attended outputs ($\boldsymbol{Y}$) for each example.

After each transformer stage, the (n, n) input is flattened and then processed by the transformer encoder. Each transformer layer produces "readout tokens" which serve as distinct tokens that are not grounded in the input image data but are added as an integral part of the Transformer Encoder. This token passes through the transformer layers, passing information from the upper layers to the lower layers. Its purpose is to capture and distribute global contextual information across different parts of the sequence. More information can be seen in the original work by Ranftl et al. [49]

- **Reassemble:** The Reassemble step bridges the Transformer Encoder to the Convolutional Decoder. It involves assembling tokens (not the readout tokens) into contiguous image-like features. This is achieved through a series of operations, including concatenation of token outputs (***Concat***), linear transformations (***Linear***), and resampling (***Resamples***). These operations culminate in refined feature representations that retain spatial and contextual information.

  1. **Resample Operation (*Resamples*):**

     The Resample operation involves spatially resizing the tokenized representations to match the desired output resolution. It is a combination of convolutional operations that involve down-sampling or up-sampling the feature maps. Mathematically, the ***Resamples*** operation can be defined as:

     $$\boldsymbol{Resamples} : R_{H_p \times W_p \times D} \rightarrow R_{H_s \times W_s \times D'}$$

     Here, $H_p \times W_p$ represents the input spatial resolution, $H_s \times W_s$ represents the output spatial resolution, and D and D' represent the feature dimensions before and after resampling.

  2. **Concatenate Operation (*Concatenate*):**

     The Concatenate operation combines token representations $\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_h$ from different layers of the Transformer Encoder into a single tensor. This tensor is then transformed using linear projections and activation functions to enhance its expressiveness.

  3. **Read Operation (*Read*):**

     The Read operation involves mapping the concatenated tensor into a tensor amenable to spatial concatenation, and then reshaping it to match the spatial layout of the original input image. This operation ensures that the tokenized representations are correctly positioned in the final image-like feature representation.

     Mathematically, the Reassemble operation can be described as follows:

     $$\boldsymbol{Reassemble}(\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_h) = \boldsymbol{Resamples}(\boldsymbol{Concatenate}(\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_h))$$

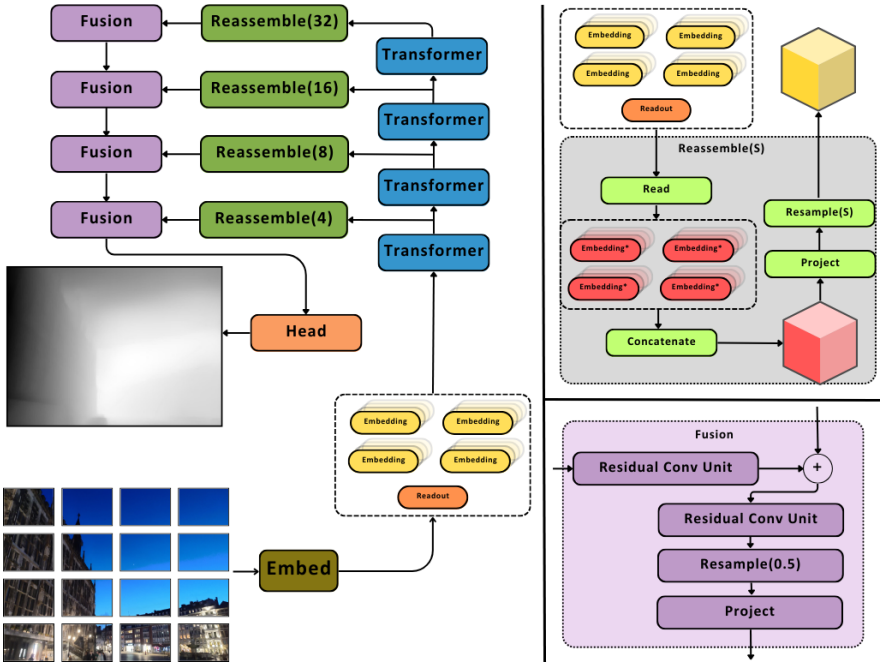

Figure 21: Depth Prediction Transformer Architecture

Model overview of the DPT architecture as described by the paper by Ranft et al [49]. (Left) The complete overview of the architecture, where the image patches are converted into embeddings by either flattening or a ResNet50 feature extractor[25]. A readout token is added, shown in red. Multiple transfer layers are used, the tokens being fed to the reassemble layer, which produces an image-like representation. Finally, fusion layers add the reassembled features and upsampled features from the previous layer using residual convolutional units[40]. (top-right) Reassemble operation. (bottom-right) Fusion operation, the convolutional decoder.

The token representations $\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_h$ from each layer of the Transformer Encoder are concatenated, linearly transformed, and then spatially resampled to form a coherent image-like feature representation. This representation serves as the input to the Convolutional Decoder for generating final predictions.

- **Convolutional Decoder:** The Convolutional Decoder integrates the refined feature representations from the Reassemble stage. It fuses these features using a RefineNet-based feature fusion block[40], gradually upsampling the representation. A task-specific output head is attached to produce the final dense prediction. This decoder architecture ensures that the information captured by the Transformer Encoder is effectively transformed into accurate predictions.

A detailed overview of the architecture is described in Figure 21

DPT utilizes convolutional decoder principles, computing dense predictions using upsampling. The Reassemble operation—Resamples, Concatenate, and Read—rebuilds tokens into coherent image-like features. Hierarchical concatenation with readout tokens ensures spatial coherence while resampling strategies control geometric shifts. By merging transformer-encoded nuances with convolutional syntax, DPT performs well for depth estimation tasks[49].

4.3.4 DPT Implementation and Final Occlusion Map

We can incorporate the DPT mechanism into the final occlusion task by using the following hybrid system:

1. **Sky Segmentation**: By using the previously described sky segmentation algorithm, either the mean-shift-top-third algorithm or the UNet operation, we can segregate the sky efficiently. This helps us provide a concrete background value for the mask.

2. **Depth estimation for Ground**: By using the DPT algorithm, we can provide a relative depth estimation value of the ground, giving up a rough idea of the distances between objects.

In this section, we look at the implementation and results of using the DPT model. We use the pre-trained model provided by HuggingFace called `Intel/dpt-large`. We use the `DPTForDepthEstimation` class from the HuggingFace python library as the model. A key thing to note is that the author did not train the model further for downstream tasks.

The author provided the previously trained DPT model with the original image and also provided the same image to the `baseline-mean-shift-top-third` algorithm and the UNet model to compute the sky mask. The output of the DPT model assigns each pixel a value using a complex mechanism that is described in the work of Ranft et al. [48] and is out of the scope of this thesis. In short, the output of the DPT model contains 0.0 as the value for the furthest pixel (sky pixel in our case) and an arbitrary floating point value for all other pixels, with higher values assigned to closer pixels.

Since we are interested in the relative depth values, we modify the final output such that the furthest pixel, i.e. the sky, is assigned the value of 0.0, and the closest pixel is assigned the value of 1.0, as described by the official HuggingFace documentation given here[3]. In short, the entire output image is scaled by the maximum value in terms of pixels in the depth image using $depth\_scaled_{(i,j)} = depth_{(i,j)}/max(depth)$ where $depth_{(i,j)}$ is the depth value at the (i, j)th position in the matrix, $depth$ is the matrix containing the depth values outputted by the DPT model, $max(\cdot)$ is the maximum function, and $depth\_scaled_{(i,j)}$ is the scaled depth value at (i, j)th position.

Next, we will multiply the pixels in both, the sky segmentation output and the DPT output, for which, we need to invert the sky mask generated by both the sky segmentation models (Previously, 1 represented sky and 0 represented ground, but we must invert the mask such that 0 represents the sky and 1 represents the ground, in order to align with the DPT output). After getting the depth map from the DPT model and sky mask from both the `baseline-mean-shift-top-third` algorithm and the UNet Model, they combined the final output by pixel-wise multiplication (multiplying the pixels in the same position in both the depth map and the sky mask) to give a final image that can be seen in Figure 22. In Figure 23 we can see the plot of some examples that the author considered challenging for both the DPT and Sky Segmentation algorithms to figure out.

4.3.5 Evaluation on EDEN Dataset

In this section, we look at the statistical evaluation of the two final approaches, DPT + `baseline-mean-shift-top-third` and DPT + UNet model on the Enclosed garDEN

[3]https://huggingface.co/Intel/dpt-large

dataset[37]. We first look at the dataset structure, and then at the evaluation results.

EDEN dataset (further referred to as `eden`) is a dataset comprising synthetic images taken from various angles of a 3D rendered garden, which are considered photorealistic by the author of the original work. An example of the dataset is presented in Figure 24. We take a 12,520 image sample of this dataset, consisting of RGB feature images, and equivalent validation matrices consisting of the metric depth of each pixel in the garden scene. The validation images are encoded as follows: 0.0 is assigned to sky pixels, while all other pixels that are not the sky (i.e. objects on the ground) contain floating-point values of metric depth from the camera. The author of this thesis performed two modifications to scale the final validation matrices between 0.0, representing the furthest object, or the sky, and from the camera 1.0, representing the closest pixel to the camera, since this aligns with the output of the DPT model.

This modification first follows the same procedure mentioned in Section 4.3.4, i.e., scaling the entire depth matrix with the maximum value of elements of the matrix. After the scaling, since the highest value of 1.0 is assigned to the pixels belonging to objects furthest from the camera which are not the sky (i.e. objects that are on the ground, and are furthest away from the camera), we first set all values that are 0.0 (i.e. the sky pixels) as 1.0, and invert the entire matrix such that 0.0 represents the further pixel and 1.0 represents the closest pixel by subtracting each pixel in the depth matrix from 1.0. We can see that this creates some inconsistencies since the sky pixels are assigned the same value as the furthest ground pixel, but since our objective is to find the change in error between DPT and DPT + `baseline-mean-shift-top-third` and UNet, we consider this change irrelevant to the final evaluation, considering all three models will be evaluated against images having the same artifacts.

This was done in order to align the output of the DPT model with that of the EDEN dataset and fit the dataset for the task of relative depth estimation. To the best of the author's knowledge, the original DPT model used in this thesis has not been trained on `eden` dataset, hence this dataset is completely novel and suitable for an independent evaluation.

To evaluate the dataset, the author first restructured the dataset into two directories, "images" containing the RGB images and "labels" containing the depth matrices, and compared the final output of the three models in question, namely DPT, DPT + `baseline-mean-shift-top-third`, and DPT + UNet, using the RGB images as the input. The output was compared to the validation images present in the `eden` dataset, where the evaluation metric used to compare the outputs with the three approaches and the validation images was Mean Squared Error, as mentioned in Section 3.3. The author took the root of this metric, appropriately named as Root Mean Squared Error (RMSE). They collected data for the three models, namely: $RMSE_{DPT}$ which was used to describe the RMSE between the validation images and the original DPT model output, $RMSE_{Baseline}$ which was used to describe the RMSE between the validation images and the DPT + `baseline-mean-shift-top-third` model output, and finally $RMSE_{UNet}$ which was used to describe the RMSE between the validation images and the DPT + UNet model output. Finally, the author noted the absolute and percentage change in RMSE value between $RMSE_{DPT}$ and $RMSE_{Baseline}$, and $RMSE_{DPT}$ and $RMSE_{UNet}$. We can see the result of this evaluation in Table 5.

In conclusion, we can clearly see that the combined models (DPT + `baseline-mean-shift-top-third` and DPT + UNet) give us some percentage points of improved results over using the original DPT model, which is shown by the fact that the DPT + `baseline-mean-shift-top-third` model has a decreased mean $RMSE$ value of 0.0067 and a decreased

	Mean	Median
$RMSE_{DPT}$	0.1929	0.1864
$RMSE_{Baseline}$	**0.1862**	**0.1801**
$RMSE_{UNet}$	0.1892	0.1825
$RMSE_{Baseline} - RMSE_{DPT}$	**-0.0067**	**-0.0063**
$RMSE_{UNet} - RMSE_{DPT}$	-0.0037	-0.0039

Table 5: Evaluation of DPT, DPT + `baseline-mean-shift-top-third` and DPT + UNet on `eden` dataset (Lower is better)

The evaluation was done on a sample of `eden` dataset containing 12,520 images. The RMSE was calculated for each model between their respective output and the validation images. The last two rows show the difference in RMSE values.

median $RMSE$ value of 0.0063. At the same time, the DPT + UNet model shows a decreased mean $RMSE$ value of 0.0037 and a decreased median $RMSE$ value of 0.0039. In terms of percentages, we can say that the DPT + `baseline-mean-shift-top-third` model performs 3.47% better in terms of mean RMSE and 3.37% in terms of median RMSE, while the DPT + UNet model performs 1.92% in terms of mean RMSE and 2.10% in terms of median RMSE against the original model.

From these results, we can infer that the combination of DPT + `baseline-mean-shift-top-third` performs better than the DPT + UNet model when evaluated on the `eden` dataset. We can also infer that both approaches show some improvement over the original DPT model. In the next section, we will look at visually analyzing the results of both approaches and finally infer whether the problem of occlusion map computation can be solved using the proposed methods.

4.3.6 Visual Results and Inference

In this section, we look at visually evaluating the output of both DPT + `baseline-mean-shift-top-third` and DPT + UNet model and intuitively try to ascertain the performance by looking at some examples produced by both approaches, also comparing them to the output of the original DPT model.

In Figure 22, we see the five examples that the author considered to be regular cases, which include (starting from the top):

1. A regular evening with a blue sky and a well-defined border between the ground and the sky. We can see that both `baseline-mean-shift-top-third` and the UNet model perform well on the sky segmentation, but the DPT model fails to predict depth on the top half. We also observe that the `baseline-mean-shift-top-third` segments the sky incorrectly. Hence, the combination of UNet and DPT provides a better depth result than just using DPT.

2. A brightly lit pasture, with blue sky and clouds. Here, the DPT model fails to provide proper depth for the sky, and the `baseline-mean-shift-top-third` fails as well, due to clouds, but the UNet model gives much better results and the overall depth map is much better with the combination.

3. A similar picture as the one before, but with shrubbery in the middle. We can see that

the DPT model provides good depth cues for this image, but the sky segmentation algorithms fail to add to this. The UNet model does perform decently, but the overall image is not any better.

4. Here we have an image of a bright sky and a well-defined border between houses. In this case, the DPT model gives erroneous depth for the sky, the but `baseline-mean-shift-top-third` algorithm corrects this, while the UNet model has some artifacts at the bottom of the picture. Both algorithms improve the overall output.

5. An evening picture with less defined sky colors and a close object. The DPT model produces a good depth map, and both the sky segmentation approaches give good results, which leads to no change in the final output.

Overall, these examples suggest that the combination of DPT with either of the sky segmentation methods produces better results than using just DPT to derive the depth map. The UNet model is more reliable in this case since it learns both the low and high-level features, while the `baseline-mean-shift-top-third` algorithm fails whenever the clustering algorithm makes a mistake in segmenting the sky properly. The DPT model produces depth maps that contain certain artifacts in the sky region that can be countered by the segmentation algorithms, thereby improving the overall quality of the map produced.

Upon looking at the challenging cases (starting from top to bottom) in Figure 22, we find the following:

1. The first image is that of a tree against the sky. As previously mentioned, trees are a vulnerable point for the UNet and `baseline-mean-shift-top-third` is also unable to segment these images properly. We see that the DPT model also produces artifacts between the leaves of the trees. Upon combining as well, the final output is still not very good, suggesting that this example is a hard problem for all three.

2. This image looks at a blue sky with a blue ocean. This image is challenging since the colors can be confusing for the model. The DPT output is very good, apart from some artifacts in the top region, while `baseline-mean-shift-top-third` fails to segment the image properly, and the UNet model also has some artifacts in the top region. The combination, however, yields a slightly better result than just the DPT model.

3. This image is challenging because it deals with a night sky and ground lights. We see that DPT still produces good results on this, but `baseline-mean-shift-top-third` and the UNet fail the segment the image properly, thereby producing low-quality results.

4. This image is originally blurry, is during the night and contains clouds and trees, which combines all challenging environments together. In this case, the DPT and the `baseline-mean-shift-top-third` algorithm combined produce very good results, and also good results just by themselves.

5. This image contains snow and a grey sky, where colors can be deceptive. The DPT model still produces a good depth map, while the `baseline-mean-shift-top-third` and UNet Model produce decent results, each having some artifacts in the form of patches or problem segmenting trees. The combined results are, however, similar to the

original DPT result, which we can infer as being just good, and the two segmentation algorithms do not contribute to the overall outcome.

We can infer from these challenging images that under certain conditions, the three algorithms can perform well, but their combination can still provide a better occlusion result than just using either of the individual algorithms.

With this statement, we can say that for the occlusion problem, a combination of DPT and UNet could produce good results when compared to pure depth prediction models, or sky segmentation models. We can also reiterate the solution presented by Masoumian et al[41], and suggest a similar application for solving the absolute depth estimation problem for producing an occlusion map.

The only downside to this approach is the time taken for one image to be processed by DPT, which can average around 3.7 seconds. Due to these limitations, implementing this method might not be feasible, and further improvements can be made to increase the speed. The `baseline-mean-shift-top-third` algorithm and the UNet model, both take significantly less time than DPT, hence the author suggests some modifications for implementing this setup in the conclusion.

Figure 22: Comparison of DPT + `baseline-mean-shift-top-third` and UNet Model Sky Segmentation, Regular Cases

Final Depth Image using the hybrid approach of adding DPT depth map predictions to the sky segmentation results using both the `baseline-mean-shift-top-third` algorithm and the UNet model. These images are considered regular cases that are easier to predict for the model, due to good camera conditions and distinguishable objects.

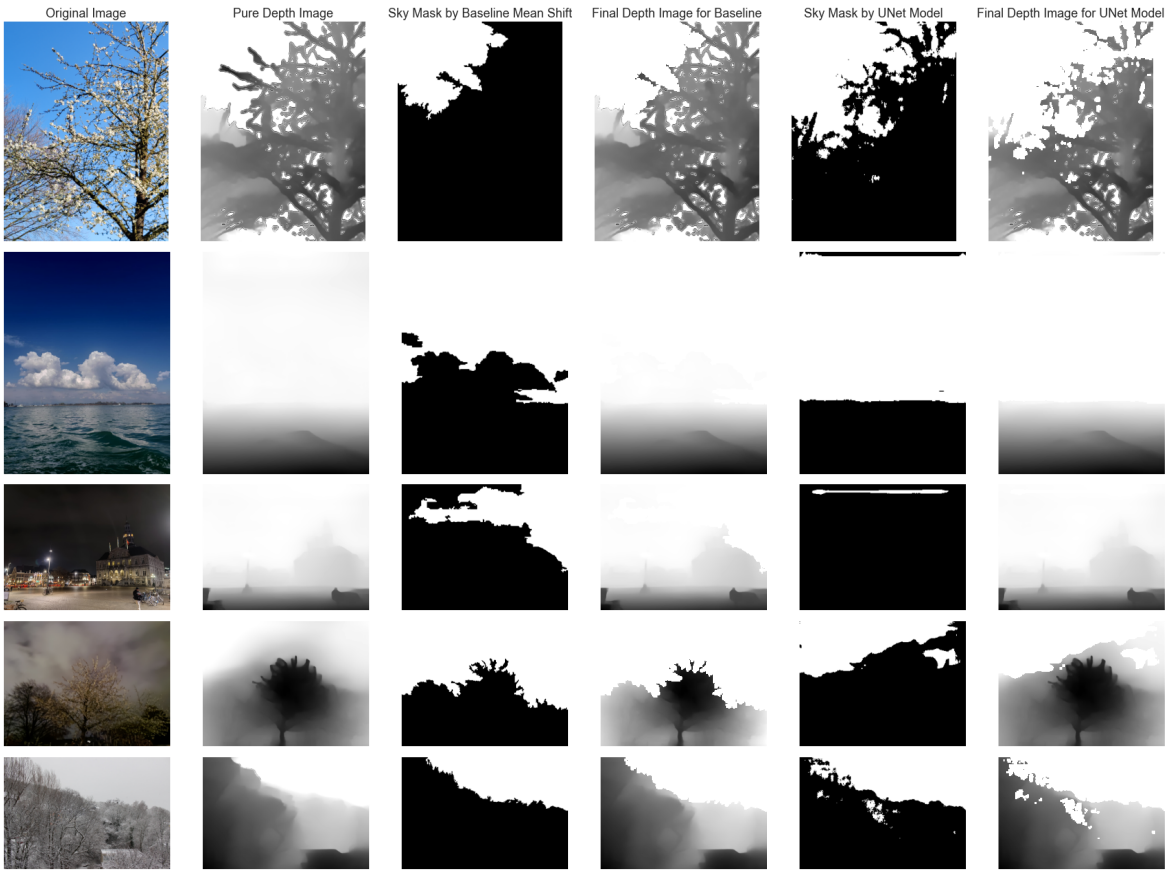

Figure 23: Comparison of DPT + `baseline-mean-shift-top-third` and UNet Model Sky Segmentation, Challenging Cases

Final Depth Image using the hybrid approach of adding DPT depth map predictions to the sky segmentation results using both the `baseline-mean-shift-top-third` algorithm and the UNet model. These cases are considered challenging to predict for the model, due to poor camera conditions and not distinguishable objects.

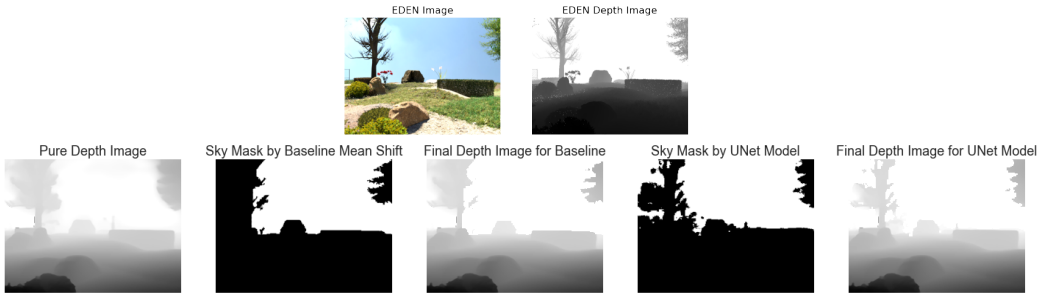

Figure 24: Example of `eden` dataset, RGB Image and Depth Image

(top-row)A random sample taken from the `eden` dataset, where the Features Image is an RGB image, and the final Validation Image is a depth matrix containing values between 0 and 1, shown as an image here. (bottom-row) The output of DPT, `baseline-mean-shift-top-third`, DPT + `baseline-mean-shift-top-third`, UNet, and DPT + UNet respectively, where the input is a sample image from the `eden` dataset

.

5 Conclusion

This thesis deals with multiple solutions to the problem of providing good occlusion in AR. We subdivided the problem into two parts: detecting the sky region, and providing depth for the ground region, and tackled each individual problem, before finally combining our approaches to give an occlusion map.

5.1 Summary

First, we looked at how to efficiently segment the sky from the rest of the image and compared using data analysis, what was going wrong with the images. We used a baseline algorithm that was based on `mean-shift-clustering` algorithm, and then we modified it with a classification condition that assigned the sky region based on the largest cluster in the upper half of the image. We named this algorithm `baseline-mean-shift` and considered it to be a baseline for more advanced solutions. Based on the analysis results, we created two variants of the algorithm, `baseline-mean-shift-augmented` and `baseline-mean-shift-top-third` which used ranges of features of clusters and the top-third of the image instead of the top half respectively. We found that `baseline-mean-shift-top-third` performs the best out of all variations in terms of mean precision and recall.

We then looked at a deep learning method using UNets to segment the images in order to resolve some of the issues that were inherently present due to the clustering approach, such as classifying the wrong cluster as the sky and misclassifying clusters representing clouds as non-sky, after which we looked at a robust comparison of both methods. We showed that the UNet approach yields even better results than `baseline-mean-shift-top-third` since it directly learns the structure of the image.

Since the `original-dataset` did not contain any representation of trees, we then fine-tuned the UNet model using a small dataset (`bachelor-dataset`) to cover up for this. We then compared this fine-tuned model to our original model and saw that the overall performance on the `original-dataset` dropped, but the fine-tuned models did learn the representation of the smaller dataset.

We then looked at how to estimate the depth of the ground, stating that the problem has a harder version called Absolute Depth Estimation, and choosing to solve the easier version first, namely Relative Depth Estimation. Since pre-trained models performed exceptionally well, and due to paucity of time and resources, we looked at a pre-trained vision transformer model called Depth Prediction Transformer, which performed well on outdoor images when visually compared.

We finally combined the results from both these subtasks, using `baseline-mean-shift-top-third` and UNet for sky segmentation, and DPT for depth prediction on the ground images, and provided a mask for occlusion that can be used to solve the original problem, if given some more information on the depth. We compared the results of our two approaches, DPT + `baseline-mean-shift-top-third` and DPT + UNet against the original DPT model on a depth dataset known as `eden` and inferred that the former model performs better than the latter, and both approaches perform better than the original DPT mode. Upon visually inspecting the final results, the author also stated that the use of DPT with a sky segmentation algorithm yields better results than using pure DPT or pure sky segmentation results since the artifacts produced by DPT can be improved upon using the sky-segmentation methods.

In the next section, we look at some of the limitations of this work.

5.2 Limitations and Future Outlook

The biggest drawback of this method is the inability to resolve the actual depth of pixels in the image. We can ascertain the relative depth of each pixel, but to place the final object, we require more information to be able to provide a relationship between the pixels. The author assumes that the given depth map can be used in conjunction with another method to finally give a binary occlusion map, based on the work by Masoumian et al. [41]. The author recommends using object detection to certain the distance of certain pixels or objects in an image and extrapolate it to find out the absolute depth from the relative depth map. This is left as future work.

Considering the intended use case of an AR experience on a tablet or smartphone, another problem identified in this approach pertains to the time of processing each frame. No experiments were done to actually see what the performance would be on an edge device, like a smartphone. The author assumes that the change in performance may render this method inefficient unless it is further optimized or its complexity reduced. Another viable option might be using a hybrid server-local setup where some computation can be done on the local device, while some in a networked location. Further investigation is needed to back this claim as well.

The author hopes to be able to fix these issues in a future endeavor, in order to finally solve the problem of occlusion in AR for virtual object placement on edge devices like smartphones.

References

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018. doi: 10.48550/arXiv:1803.08375.

[2] Hina Ajmal, Saad Rehman, Umar Farooq, Qurrat U Ain, Farhan Riaz, and Ali Hassan. Convolutional neural network based image segmentation: a review. *Pattern Recognition and Tracking XXIX*, 10649:191–203, 2018. doi: 10.1117/12.2304711.

[3] Zeynep Akın and Ahmet Sayar. Challenges in determining the depth in 2-d images. In *2022 International Conference on INnovations in Intelligent SysTems and Applications (INISTA)*, pages 1–6. IEEE, 2022. doi: 10.1109/INISTA55318.2022.9894120.

[4] Mina Amiri, Rupert Brooks, and Hassan Rivaz. Fine tuning u-net for ultrasound image segmentation: Which layers? In *MICCAI Workshop on Domain Adaptation and Representation Transfer*, pages 235–242. Springer, 2019. doi: 10.1007/978-3-030-33391-1_27.

[5] Ronald Azuma, Yohan Baillot, Reinhold Behringer, Steven Feiner, Simon Julier, and Blair MacIntyre. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997. doi: 10.1162/pres.1997.6.4.355.

[6] Tyler Bell, Beiwen Li, and Song Zhang. Structured light techniques and applications. *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 1–24, 1999. doi: 10.1002/047134608X.W8298.

[7] Serge Belongie, Chad Carson, Hayit Greenspan, and Jitendra Malik. Color-and texture-based image segmentation using em and its application to content-based image retrieval. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 675–682. IEEE, 1998. doi: 10.1109/ICCV.1998.710790.

[8] William Benjamin, Andrew Zisserman, and Philip HS Torr. Depth-agnostic supervised learning. *arXiv preprint arXiv:1901.10020*, 2019. doi: 10.48550/arXiv:1901.10020.

[9] Gary Bradski. The opencv library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, 25(11):120–123, 2000.

[10] James Bruce, Tucker Balch, and Manuela Veloso. Fast and inexpensive color image segmentation for interactive robots. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, volume 3, pages 2061–2066. IEEE, 2000. doi: 10.1109/IROS.2000.895274.

[11] Yu-Lin Chang, Chih-Ying Fang, Li-Fu Ding, Shao-Yi Chen, and Liang-Gee Chen. Depth map generation for 2d-to-3d conversion by short-term motion assisted color segmentation. In *2007 IEEE international conference on multimedia and expo*, pages 1958–1961. IEEE, 2007. doi: 10.1109/ICME.2007.4285061.

[12] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on pattern analysis and machine intelligence*, 17(8):790–799, 1995. doi: 10.1109/34.400568.

[13] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. doi: 10.48550/arXiv:1410.0759.

[14] Dorin Comaniciu and Peter Meer. Robust analysis of feature spaces: Color image segmentation. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pages 750–755. IEEE, 1997. doi: 10.1109/CVPR.1997.609410.

[15] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. doi: 10.48550/arXiv:2010.11929.

[16] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *Advances in Neural Information Processing Systems*, 27:2366–2374, 2014. doi: 10.48550/arXiv.1406.2283.

[17] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2002–2011, 2018. doi: 10.1109/CVPR.2018.00214.

[18] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975. doi: 10.1109/TIT.1975.1055330.

[19] Alberto Garcia-Garcia, Sergio Orts-Escolano, Sergiu Oprea, Victor Villena-Martinez, and Jose Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017. doi: 10.48550/arXiv:1704.06857.

[20] Valentino Geuenich. Occlusion for augmented reality windfarms in great distance, 2022. URL `https://ths.rwth-aachen.de/wp-content/uploads/sites/4/Thesis_Geuenich.pdf`.

[21] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020. doi: 10.48550/arXiv:2009.07485.

[22] Stephanie Halbrügge, Hans Ulrich Buhl, Gilbert Fridgen, Paul Schott, Martin Weibelzahl, and Jan Weissflog. How germany achieved a record share of renewables during the covid-19 pandemic while relying on the european interconnected power network. *Energy*, 246:123303, 2022. doi: 10.1016/j.energy.2022.123303.

[23] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Patrice Horaud. *Time-of-flight cameras: principles, methods and applications*. Springer Science & Business Media, 2012. doi: 10.1007/978-1-4471-4658-2.

[24] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003. doi: 10.1017/CBO9780511811685.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[26] David Held and Gal Levin. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(5):1151–1165, 2018. doi: 10.48550/arXiv.1609.05158.

[27] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. doi: 10.1109/CVPR.2017.243.

[28] John D Hunter. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, 9(03):90–95, 2007. doi: 10.1109/MCSE.2007.55.

[29] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[30] Frédéric Jean. Pymeanshift. `https://github.com/fjean/pymeanshift`, 2012.

[31] Dongki Jung, Jaehoon Choi, Yonghan Lee, Deokhwa Kim, Changick Kim, Dinesh Manocha, and Donghwan Lee. Dnd: Dense depth estimation in crowded dynamic indoor scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 12797–12807, 2021. doi: 10.1109/ICCV48922.2021.01256.

[32] Hirokazu Kato and Mark Billinghurst. Markerless tracking for augmented reality. *Communications of the ACM*, 42(8):36–42, 1999. doi: 10.1007/978-1-4614-0064-6_11.

[33] Prakhar Kaushik, Alex Gain, Adam Kortylewski, and Alan Yuille. Understanding catastrophic forgetting and remembering in continual learning with optimal relevance mapping. *arXiv preprint arXiv:2102.11343*, 2021. doi: 10.48550/arXiv:2102.11343.

[34] Faisal Khan, Saqib Salahuddin, and Hossein Javidnia. Deep learning-based monocular depth estimation methods—a state-of-the-art review. *Sensors*, 20(8):2272, 2020. doi: 10.3390/s20082272.

[35] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. doi: 10.48550/arXiv:1412.6980.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012. doi: 10.1145/3065386.

[37] Hoang-An Le, Thomas Mensink, Partha Das, Sezer Karaoglu, and Theo Gevers. Eden: Multimodal synthetic dataset of enclosed garden scenes. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1579–1589, 2021. doi: 10.1109/WACV48630.2021.00162.

[38] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. doi: 10.1109/5.726791.

[39] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*, 2019. doi: 10.48550/arXiv:1912.06059.

[40] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1925–1934, 2017. doi: 10.1109/CVPR.2017.549.

[41] Armin Masoumian, David GF Marei, Saddam Abdulwahab, Julian Cristiano, Domenec Puig, and Hatem A Rashwan. Absolute distance prediction based on deep learning object detection and monocular depth estimation models. In *CCIA*, pages 325–334, 2021. doi: 10.3233/FAIA210151.

[42] Yue Ming, Xuyang Meng, Chunxiao Fan, and Hui Yu. Deep learning for monocular depth estimation: A review. *Neurocomputing*, 438:14–33, 2021. doi: 10.1016/j.neucom.2020.12.089.

[43] Kerry A. Nice and Jasper S. Wijnands. Dataset for: Sky pixel detection in outdoor imagery using an adaptive algorithm and machine learning., February 2019.

[44] Kerry A Nice, Jasper S Wijnands, Ariane Middel, Jingcheng Wang, Yiming Qiu, Nan Zhao, Jason Thompson, Gideon DPA Aschwanden, Haifeng Zhao, and Mark Stevenson. Sky pixel detection in outdoor imagery using an adaptive algorithm and machine learning. *Urban Climate*, 31:100572, 2020. doi: 10.1016/j.uclim.2019.100572.

[45] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. URL `https://developer.nvidia.com/cuda-toolkit`.

[46] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018. doi: 10.48550/arXiv:1811.03378.

[47] Tomas Polasek, Martin Čadík, Yosi Keller, and Bedrich Benes. Vision uformer: Long-range monocular absolute depth estimation. *Computers & Graphics*, 111:180–189, 2023. doi: 10.1016/j.cag.2023.02.003.

[48] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE transactions on pattern analysis and machine intelligence*, 44(3):1623–1637, 2020. doi: 10.48550/arXiv.1907.01341.

[49] René Ranftl, Alexey Bochkovskiy, and Vladlen Koltun. Vision transformers for dense prediction. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 12179–12188, 2021. doi: 10.1109/ICCV48922.2021.01196.

[50] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015. doi: 10.1007/978-3-319-24574-4_28.

[51] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. doi: 10.48550/arXiv:1609.04747.

[52] Jan Schaffranek and Saif Al-Dilaimi. Tensorcross. `https://github.com/franneck94/TensorCross`, 2023.

[53] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47:7–42, 2002. doi: 10.1109/SMBV.2001.988771.

[54] Alex Sherstinsky. Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020. doi: 10.1016/j.physd.2019.132306.

[55] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. doi: 10.48550/arXiv.1409.1556.

[56] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014. doi: 10.48550/arXiv.1409.3215.

[57] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. doi: 10.48550/arXiv.1706.03762.

[58] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272, 2020. doi: 10.1038/s41592-019-0686-2.

[59] Google AR & VR. Arcore: Augmented reality at android scale, 2018. URL `https://developers.google.com/ar/discover/supported-devices`.

[60] Daniel Wagner and Dieter Schmalstieg. A survey of evaluation techniques used in augmented reality studies. *CyberPsychology & Behavior*, 11(2):129–141, 2008. doi: 10.1145/1508044.1508049.

[61] Daniel Wallach and Bruno Goffinet. Mean squared error of prediction as a criterion for evaluating and comparing system models. *Ecological modelling*, 44(3-4):299–306, 1989. doi: 10.1016/0304-3800(89)90035-5.

[62] Stanisław Weglarczyk. Kernel density estimation and its application. In *ITM web of conferences*, volume 23, page 00037. EDP Sciences, 2018. doi: 10.1051/itmconf/20182300037.

[63] Chengrui Wei, Meng Yang, Lei He, and Nanning Zheng. Fs-depth: Focal-and-scale depth estimation from a single image in unseen indoor scene. *arXiv preprint arXiv:2307.14624*, 2023. doi: /10.48550/arXiv.2307.14624.

[64] Zhilu Zhang and Mert Sabuncu. Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, 31, 2018. doi: 10.48550/arXiv.1805.07836.