Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

Master's Thesis

Multi-View Temporal Fusion in Semantic Segmentation

Manoj Kolpe Lingappa

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Nico Hochgeschwender
Prof. Dr. Sebastian Houben
M.Sc. Deebul Sivarajan Nair

January 2023

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Manoj Kolpe Lingappa

Abstract

Semantic segmentation is a technique used in computer vision to assign labels to each pixel in an image or video. It is often used in safety-critical applications, where the efficiency of the segmentation model is important. In video segmentation, the task is typically performed on key frames or all frames in a sequence. Overlapping information between consecutive frames can be used to improve the performance of semantic segmentation through temporal fusion. In this study, two separate temporal fusion methods using Gaussian Process and Long Short Term Memory (LSTM) were developed and applied to the latent space embedding of an encoder-decoder model, and their results were compared to a baseline model. The LSTM model performed well on the scannet and vkitti datasets, achieving mIoU scores of 0.65 and 0.72, respectively, and accuracy scores of 0.85 and 0.94, respectively.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Prof. Dr. Nico Hochgeschwender, Prof. Dr. Sebastian Houben and M.Sc. Deebul Sivarajan Nair for the continuous support of my Master Thesis project for their guidance, encouragement, enthusiasm, and gracious support throughout my work.

I would like to express my gratitude to my colleagues at Hochschule Bonn-Rhein-Sieg and my family members for their support.

Contents

List of Figures

List of Tables

1

Introduction

1.1 Motivation

Any task to predict by combining data from different sources and temporal fusion use data fusion. Data fusion combines information from multiple sources to achieve improved performance and inferences. According to Hall and Llinas [1], data fusion can be defined as "data fusion techniques combine data from multiple sensors and related information from associated databases to achieve improved accuracy and more specific inferences than could be achieved by the use of a single sensor alone." The living organisms fuse information from various sources and past data to make an informed decision [10]. Data fusion aims to reduce the prediction error probability and improve the model's reliability. Data from multiple sources can be fused at different levels, such as raw data fusion, feature fusion, or decision levels. The data sources are from different fields of varying data types. The most common areas include decision fusion and multisensor data fusion [11].

Data fusion is divided into temporal and multi-sensor data fusion based on the timestamp factor. Temporal data are the data collected over time and fusing them together, resulting in temporal data fusion. In the multi-sensor data fusion approach, the data from different sensors are collected for a given time and fused for improved prediction. In the hybrid approach, temporal data and multi-sensor data are fused. Information fusion is applied in different fields such as time series prediction [12], video-based depth estimation [13], and segmentation [14].

Understanding the surrounding regions and the decision-making of a human is based on the signals obtained from different sensors. However, knowing the past helps to recognize the nearby activity better or make an educated choice. Fusing the information from different sources and the former data adds to an improved outcome. Temporal fusion fuses the information to the current step to improve the prediction at each timestamp. Common temporal data types include weather data, video sequence frames, and sensor data. Semantic segmentation takes advantage of temporal fusion to make a better decision.

1.1.1 Temporal fusion

In a general setting, the previous data is not utilized to make a current prediction, resulting in information loss. The rich features from the past can be utilized in the current step, thereby making a

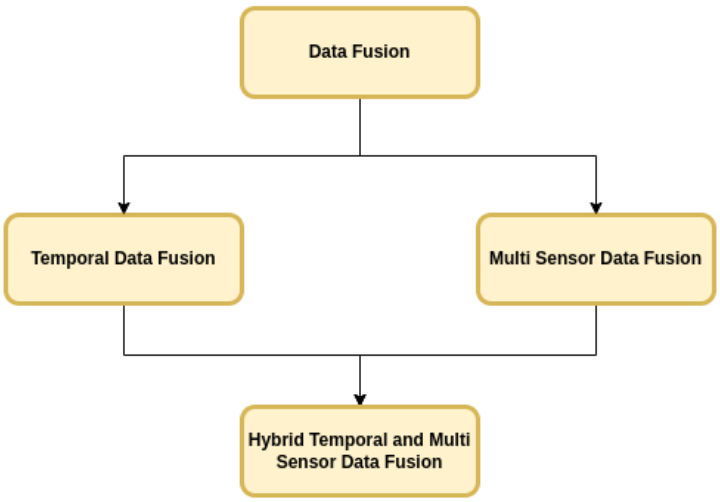


Figure 1.1: Data fusion categories based on the type of fusion. Temporal data fusion fuse the temporal data from past sequences, and multi-sensor data fuse the data from multiple sensors collected at the moment. Generated with Diagrams.net

robust and efficient model prediction. Temporal fusion is one of the dimensions of data fusion, and different data sources collected over time are fused for improvement in the prediction [15]. Classical multi-data fusion finds application in automatic target tracking, autonomous vehicle detection, surveillance systems, robotics, wearable devices, and manufacturing monitoring. The data is collected over time and contains essential features in all areas mentioned earlier. These temporal features are combined to make an efficient prediction.

The temporal fusion can model the behavioral aspect of the collected data rather than just the current timestep data. The temporal fusion extracts the relationship between contextual and temporal proximity [16]. The temporal arrangements of events are captured, thereby incorporating the cause and effect phenomenon [16]. Temporal fusion can be commonly observed in human activity detection [17], context-aware mobile phones [19], and online batch process monitoring [18].

A 3D object detection approach on two popular datasets, KITTI[4] and nuScenes[5] takes single-time step LIDAR data for prediction, resulting in the loss of valuable forecast and features computed during the previous step [3]. Using the rich features present in the successive frames to accommodate the past data at a time is extensively studied in neural network-based action recognition [19] [20] [21] [22] and video object detection [23] [24] [25] methods. The fusion of features from the previous step to the current step to improve the 3D object detection is studied in the Temp-Frustum Net architecture [3]. A Temporal Fusion Module (TFM) is proposed to combine the object-specific features. The temporal fusion method improved the average result by 6% [26]. Depth maps using dynamic MRFs fuse Time-of-Flight (TOF) and passive stereo to get an enhanced depth map. The depth map estimation is extended to the temporal domain resulting in accurate depth maps [27]. Multi-camera video surveillance fuses the spatiotemporal

frames from different sources to reliably find the motion trajectories [28]. A moving object is detected and segmented with the unmanned aerial vehicle (UAV) data by stacking the consecutive frames containing objects of interest resulting in constant object position and moving background, thereby improving the segmentation efficiency [29].

1.1.2 Semantic segmentation

Segmentation of images is an essential task of visual understanding systems. It involves dividing the image into multiple segments. Image segmentation can be framed as classifying the individual pixels into a particular class or semantic labels. Segmentation can be classified as a semantic, instance, and panoptic. Segmentation of images finds a broad range of application areas [30], such as medical for boundary extraction and tissue volume estimation, autonomous systems for detecting a boundary for path planning, and surveillance to track objects. Semantic segmentation is not only about the data but the problem segmentation addresses. For example, in a pedestrian detection system, pixels belonging to a person are categorized into a single class; however, for action recognition, the different parts of the body are classified into other classes. Instance segmentation [31] solves the problem of counting unique objects present in an image and is a common task in image retrieval tasks. Many traditional techniques have been developed to solve the segmentation problem [32]. For specialized tasks, different algorithms are developed [33]. Work by Shervin surveyed the various state-of-the-art segmentation algorithms [34]. However, many algorithms are developed, but only some works are proposed for multi-view semantic segmentation. From the previous work, it is evident that temporal fusion improves the model's performance. This work aims to study the impact of the temporal fusion of information in the latent space and cross-transfer the technology to the semantic segmentation task.

1.2 Challenges and Difficulties

Deep learning methods have contributed to the improvement of semantic segmentation. Building a temporal fusion for a semantic segmentation model is challenging due several factors involved such as temporal variations, handling large data volumes, noise and error corrections, robustness to different scenarios. Common challenges involved are the

- Datasets
- Fusion architecture
- Computation cost
- Application areas

1.2.1 Dataset

The deep learning model can be trained from scratch in many application areas, given that we have large datasets. However, there are not enough datasets available for a new domain to train the model; in

such cases, transfer learning can be applied. In the transfer learning approach, a model is trained on some data, and the part of trained model weights are used for building a new application areas architecture. Many deep learning-based models are trained on the ImageNet datasets and take the pretrained encoder weights, which capture the features needed to do the segmentation, thereby reducing the dependency on the requirement of large datasets. Image augmentation is another approach to increase the number of data points. Data augmentation helps to create more data by applying a transformation to the existing small datasets so that a variety of input data is generated from the existing small datasets. Some typical transformations on the input images are translation, reflection, rotation, warping, scaling, color space shifting, and projecting onto the principal component. It helps to faster convergence, reduce the over-fitting probability, and improve the model's generalization capability. For some tasks, data augmentation showed improvement in the model's performance. For temporal fusion, there is a need for 2D datasets along with the camera's pose. Pose information can be fused at the latent space to improve the prediction efficiency of the model. There is a need for different kinds of datasets, such as still images, navigation datasets, and Unmanned aerial vehicle (UAV) datasets, to validate the model in a different environment, and helps to evaluate the model performance.

1.2.2 Fusion architecture

With the advancement of deep learning, more segmentation models are developed with improved efficiency and a variety of fusion architecture. The fusing of features is commonly used in the segmentation task. Adapting fusion features in the increased depth deep learning model showed significant improvement in the prediction. U-net [35] model effectively use the already learned features by fusing the information from the encoder to the decoder. To tackle the decrease of initial image resolution at the output, a RefineNet [36] network was proposed. Deeper layers capture the high-level semantic features refined by fusing the fine-grained features from the earlier convolutions. Dense connection is employed in many of the recent neural network architectures [37], [38], [39]. Choosing the appropriate fusion architecture depends on the problem and the available resources to solve the problem.

1.2.3 Computation cost

Many state-of-the-art segmentation networks require high computation costs during training and inference time. So the recent research is focused on decreasing the computation cost and keeping the model's accuracy high. To deploy the model in low computational mobile devices, simpler models need to be developed that fit the device's computation cost. This can be done by compressing the model or using the knowledge distillation techniques to build the low computational model [34].

1.2.4 Real time inference for various application areas

Most of the recent top-performing semantic segmentation models are based on the fully convolutional network [40]. A real-time application or the camera's frame rate needs to be high, and the model has

reasonable accuracy and prediction speed. Real-time prediction is highly critical in the autonomous driving and medical fields. However, most of the fully convolutional network needs to be better with respect to the maximum requirements defined by application areas. Models with dilated convolution improved the performance of the model. However, the benchmark can still be improved. ICNet takes multiple input sizes to capture objects of varying sizes to tackle the real-time deployment [41].

1.3 Use cases

Semantic segmentation finds application in many areas of computer vision. Some of them are listed below,

1.3.1 Autonomous driving and Robotics

Essential components of autonomous driving systems are object recognition, object localization, and segmentation. Semantic segmentation classifies each image pixel into a particular class, thereby identifying different classes such as street, traffic signs, trees, cars, sky, pedestrians, or sidewalks. Due to safety concerns, it is critical to classify each pixel with high accuracy. The rich information captured in the last step can be used in the current step calculation to make a better prediction at the current computational step. With the development of the robotics system to perform complex tasks, the interaction with the environment also increased. So, there is a need to develop a robust system to understand the knowledge about the workspace.

1.3.2 Weed mapping using Unmanned Aerial Vehicle (UAV)

Mapping of the fields is essential for weed control and spraying applications. The presence of the weed can be mapped by unmanned aerial vehicle remote sensing technology. The targeted spraying of the weed area helps to curb weed growth by inspecting the weed map obtained from the UAV. The entire process involves real-time image processing hardware that integrates map visualization, flight control; image collection [42]. Semantic segmentation can be employed with good performance and real-time capability to build a weed map.

1.3.3 Real-Time Hand Gesture Recognition

Hand Gesture Recognition (HGR) is an essential component in human-computer interactions. With the advancement of vision-based HGR systems, HGR is widely used in the automotive sector, consumer electronics, home automation, etc. An essential feature of the HGR is real-time performance, and HGR should perform without lag to control the cursor's location. HGR is based on the semantic segmentation method to locate the hand's position; therefore, an efficient real-time segmentation network needs to be developed [43].

1.4 Problem Statement and Contribution

Research question answered and contribution in the thesis work is listed below

1.4.1 Research question

RQ1 What are the works on state-of-the-art temporal fusion?

RQ2 How are the results from RQ1 compared with each other to perform temporal fusion?

RQ3 How to cross-transfer the depth estimation temporal fusion technique to semantic segmentation?

1.4.2 Contribution

- Literature review on the temporal fusion in the context of depth estimation and semantic segmentation
- Analysis of the state-of-the-art temporal fusion architectures
- Create a baseline of temporal fusion with sequence images
- Compare performances of state-of-the-art temporal fusion techniques with different error metrics
- Cross-transfer the temporal fusion architecture to the segmentation task

1.5 Report outline

The theoretical background of deep learning, semantic segmentation, temporal fusion, and their limitations is discussed in Chapter 2. Datasets, preprocessing steps, experimental designs, training procedures, and hardware configuration used for training and inferences are listed down in Chapter 3. Evaluation of the temporal fusion architecture with different experimental settings, metrics, and research questions are discussed in Chapter 4. Deployment of the model in the android is described in the android deployment Chapter 5. Finally contribution of the thesis work, lessons learned, and future work is explained in conclusion chapter 6.

2

State of the Art

Introduction to modern deep learning and their impact on the various vision tasks are described in the Deep Learning section. Information fusion in the temporal domain to fuse information is explained in Temporal Fusion. State-of-the-art segmentation of the input images, in particular, the semantic segmentation task, is illustrated in the Semantic Segmentation section. State-of-the-art segmentation in the classical era and modern deep learning plays a crucial role in temporally fused semantic segmentation. However, there is very little work in fusing the camera pose into the segmentation task in a temporal fashion. More details are discussed in the Semantic Segmentation. Finally, chapter 2 is ended with a discussion on the limitations of the previous work concerning temporal fusion.

2.1 Deep Learning

Deep learning is a subfield of machine learning that aims to learn the features present in the data by utilizing hierarchical architectures. The deep area learning falls in artificial intelligence is depicted in the picture 2.1

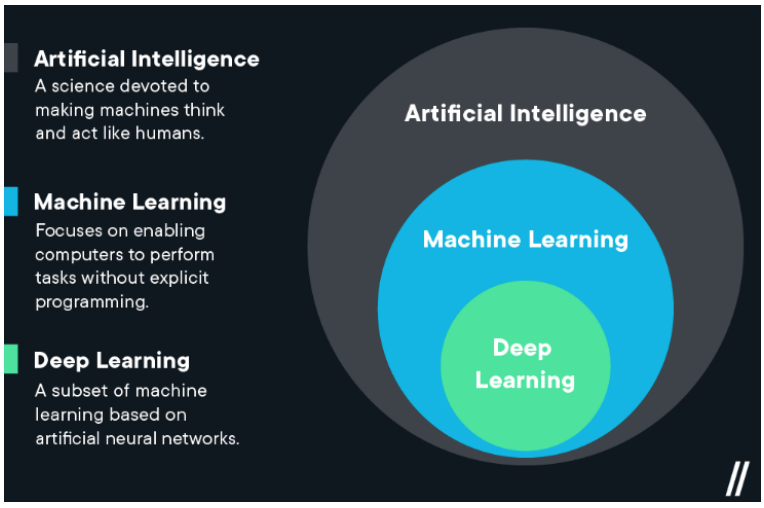


Figure 2.1: Deep learning in the artificial intelligence domain. Courtesy of [1]

Classical machine learning system uses the raw input, and domain experts carefully represent the data as a feature vector from which the data is fed to the models to learn the patterns and classify them into appropriate classes [44]. Deep learning is a representation learning that takes raw data and finds the patterns in the data with different levels of representation in the multiple layers [44]. Deep learning can learn any complex representation of the data. For example, an image is represented as pixels and is fed to the neural network; at each layer of the network, a different feature is learned. Higher-level features, such as edges at a specific orientation and location, are determined in the first layer. In the second layer, motifs are learned, and so on. The vital aspect of deep learning is that the features are not designed by the field expert but instead learned from the data with a specific set of learning procedures [44].

Many current state-of-the-art learning models use the deep learning approach to learn a complex function from data. Currently, deep learning methods can be found in image recognition [45], speech technologies [46], the discovery of drug molecule [47], understanding the particle accelerator data [48], DNA sequencing [49], and natural language processing [50].

Computer vision is the field of computer science that deals with replicating the functionalities of the human visual system. Traditionally computer vision solved the vision problem by finding hand-crafted features. However, the performance of the classical approach is outperformed by the advancement of deep learning-based methods. Hand-crafted feature descriptors such as Speeded Up Robust Features (SURF) and Hough Transforms are used as feature vectors for the classical machine learning methods for learning [51]. Deep learning methods automatically learn the patterns from the data. Computer vision solves a wide variety of problems in the perception domain. Latest approaches helps to solve the detection [52], [53], classification [54], image synthesis and segmentation tasks [55].

Temporal data are time-varying information and can be commonly observed in financial portfolio management, accounting, medical records, inventory management, and data from the airline, hotel; train industries contain time components with it [56]. Video data are constructed by combining time variant frames and are a typical example of temporal data. Temporal fusion deals with combining past information into the current step computation with the aim of improved performance.

In general, approach segmentation is done frame by frame or by skipping between frames and computing the segmentation on the nth frame. Temporal fusion can be applied in these settings to improve segmentation by combining the past rich information in the current step.

2.2 Temporal Fusion

Temporal fusion can be defined as the process of fusing the temporal data onto the current step to improve the model's performance. Temporal data can be observed in many fields, such as social media, healthcare, accounting, agriculture, transportation, physics, crime data, traffic dynamics, and climate science [57]. Temporal data can be encountered with different data types: video, audio, tabular data, and sensor data. Forecasting is a typical application of temporal fusion. Multi-horizon forecasting is an important problem in the domain of time series. Multi-horizon forecasts allow the user to optimize the process across the entire path. A novel Temporal fusion Transformer (TFT) [58] is an attention-based DNN architecture for forecasting by fusing the important past features into the current step. Temporal fusion

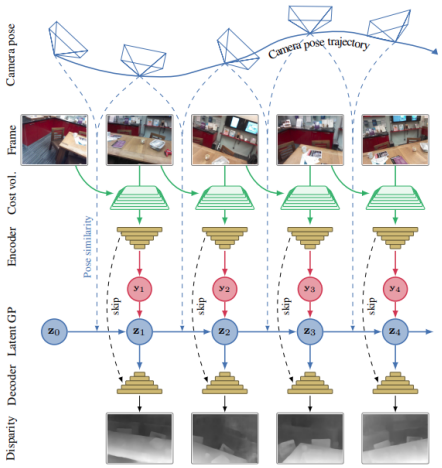


Figure 2.2: Multi-View Stereo by Temporal Nonparametric Fusion. Two consecutive frames are used to construct the cost volume and the output is passed onto the encoder to generate latent space encoding. The frames coordinates are transferred to the Gaussian Process to generate the updated latent space encoding. The Gaussian Process take encoder output, frame coordinates, and the propagated latent space encoding. The output of Gaussian Process is forwarded to decoder to generate the depth map. Courtesy of [2]

plays a significant role in improved video action recognition. Temporal fusion helps in two ways; firstly, by understanding the temporal data, the accuracy of the recognition for the dynamic action is improved; secondly, removing the redundant temporal data saves the computation overhead. A temporal fusion network known as the AdaFuse fuses the current and past features with a goal of improved accuracy and efficiency [59]. A temporal nonparametric fusion aims to fuse the temporal pose data to the computation of the depth map, thereby improving accuracy and efficiency [2]. The architecture of the multi-view stereo can be depicted in Fig 2.2. An online multi-view depth prediction approach where the depth estimated in the previous step is fused onto the current step in a sensible manner. The network is named DeepVideoMVS, and it is based on the encoder-decoder architecture. A ConvLSTM is placed at the latent space to fuse the information from the previous step. The proposed approach outperformed all the existing state-of-the-art multi-view stereo methods evaluated on the standard metrics [60]. A Multiple Fusion Adaptation (MFA) method improves the segmentation accuracy on an unlabeled dataset. Three fusion approach was proposed under MFA, cross-model fusion, temporal fusion, and novel online-offline pseudo labels. The MFA produced improved semantic segmentation results of 58.2% and 62.5% on GTA5-to-Cityscapes and SYNTHIA-to-Cityscapes, respectively [61].

2.3 Semantic Segmentation

Humans can perceive the surrounding environment and make sense of it with high accuracy. Due to the advancement of computer vision, these capabilities are transferred to machines, performing even better than humans. Today, we have computer vision models that can detect objects, find shapes, track object movement and perform an action based on the data. Computer vision is most commonly used in

autonomous driving cars, aerial mapping, surveillance applications, virtual and augmented reality, etc. One common problem in computer vision is labeling each pixel of the image to a particular category. Also known as segmentation. Mathematically image segmentation can be defined as

If I is a set of all image pixels of an image, then segmentation generates unique regions $S_1, S_2, S_3, S_4, ...S_n$ such that combining all these regions will return I.

Image segmentation can be classified into three categories Semantic segmentation, Instance segmentation, and Panoptic segmentation. Semantic segmentation finds the objects' shape, size, and form in addition to their location. Instance segmentation finds one more parameter of a number of unique objects in the image. Panoptic segmentation is the combination of semantic and instance segmentation. The difference between all the types of semantic segmentation can be observed in Fig 2.3.

Figure 2.3: Picture depicting the semantic, instance and panoptic segmentation examples. Courtesy of [3]

2.3.1 Classical Semantic Segmentation

Most commonly used traditional segmentation techniques are threshold-based technique [62], histogram-based bundling, region-growing [63], k-means clustering, watersheds, active contours, graph cuts, conditional and Markov random fields [64], sparsity-based methods [65]. However, deep learning (DL) in recent years yielded a new generation of image segmentation models with state-of-the-art performance.

2.3.2 Deep Learning based Semantic Segmentation

Deep learning based segmentation network can be classified into following categories [4]

- Fully convolutional networks
- Convolutional models with graphical models
- Encoder-decoder-based models
- Multi-scale and pyramid network-based models
- R-CNN-based models (for instance segmentation)
- Dilated convolutional models and DeepLab family
- Recurrent neural network-based models
- Attention-based models
- Generative models and adversarial training
- Convolutional models with active contour models

Deep learning-based computer vision models most commonly use the convolutional neural network [66], recurrent neural network (RNNs), and Long short term memory (LSTM), encoder-decoder [6]. Generative adversarial networks (GANs) based networks [4]. The master thesis work is concentrated on the encoder-decoder-based deep learning models. The encoder-Decoder-based network is a two-stage network that learns to map from the input point to the output point. In the encoder stage, the input data is compressed into a latent space representation $z = f(x)$ and the decoder decompresses the latent space representation to the output $a = g(z)$ [67]. Latent representation of the input data in compressed form. It can be commonly observed in image-to-image translation problems and in sequence-to-sequence models in NLP. A reconstruction loss $L(y, \hat{y})$ is defined at the output that measures the differences between the ground truth output y and corresponding reconstruction $\hat{y}$. Autoencoders are the particular version of the encoder-decoder models with similar input and output.

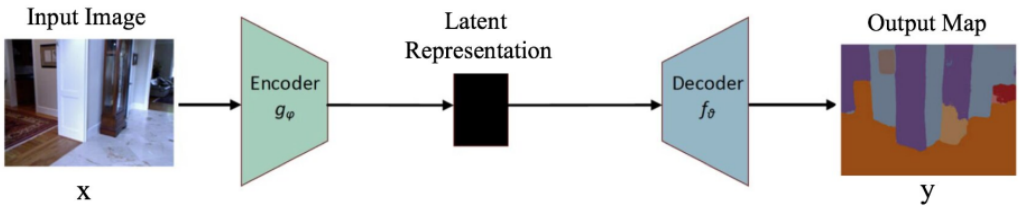


Figure 2.4: Plain encoder-decoder architecture with latent space encoding. Courtesy of [4]

Most of the segmentation networks are encoder-decoder-based architecture. A novel semantic segmentation network was proposed by Noh et al. [5]. The network is based on deconvolution. The encoder network is based on the VGG 16-layer network, and the decoder network takes the latent space encoding and outputs the pixel-wise class probabilities. The segmentation mask and pixel-wise class labels are

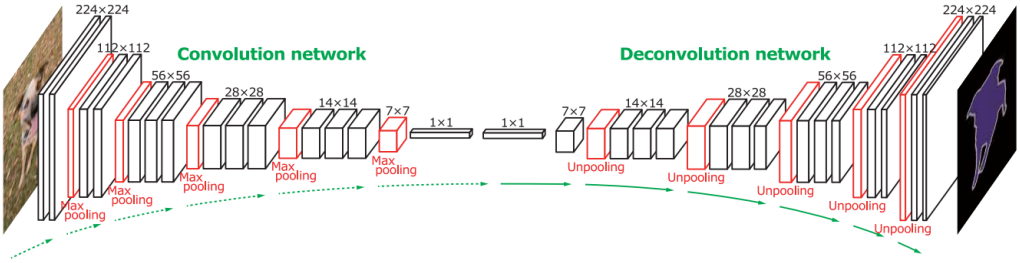


Figure 2.5: Simple encoder-decoder architecture with convolution and deconvolution network. Courtesy of [5]

predicted by the deconvolutional and unpooling layers. The network generated an accuracy of 72.5 % on the PASCAL VOC 2012 dataset.

Badrinarayanan et al proposed a convolutional encoder-decoder architecture for image segmentation called as SegNet [6]. The architecture of the SegNet described in the Figure 2.6

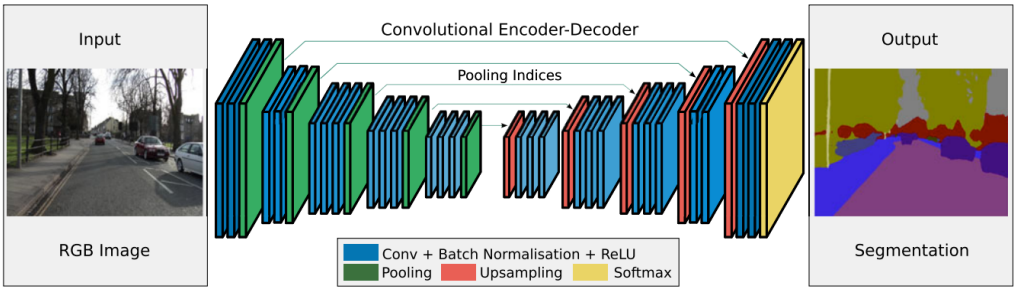


Figure 2.6: SegNet architecture with novel upsampling of latent space encoding. Courtesy of [6]

The encoder part of the SegNet consists of 13 convolutional layers in the VGG16 network, followed by the pixel-wise classification layer. Decoder uniquely upsamples the low-resolution feature maps. Non-linear upsampling is performed by using the pooling indices computed in the max-pooling step of the encoder. This process of reusing the encoder output helps to eliminate the need for learning to up-sample. Dense feature maps are generated by convolving with the trainable filters. To account for the uncertainty involved with the encoder-decoder network, scene segmentation is proposed [68]. HRNet [69] is the recently developed high-resolution network that connects the high to low-resolution convolutions streams in parallel and exchanges information between different resolutions. HRNet maintains high-resolution representation through the encoding process. Many recent architectures use HRNet as the backbone. Other encoder decoder segmentation models are Stacked Deconvolutional Network [70], Linknet [71], W-net [72].

Many segmentation models are developed for medical applications, and among those, U-Net [7], and V-Net [73] are the famous architecture. These architectures are now used outside of the medical domain.

Ronneberger et al. [7] proposed a segmentation model to perform semantic segmentation on medical microscopy images. The architecture of the U-Net is described in Fig 2.7. The contracting part captures

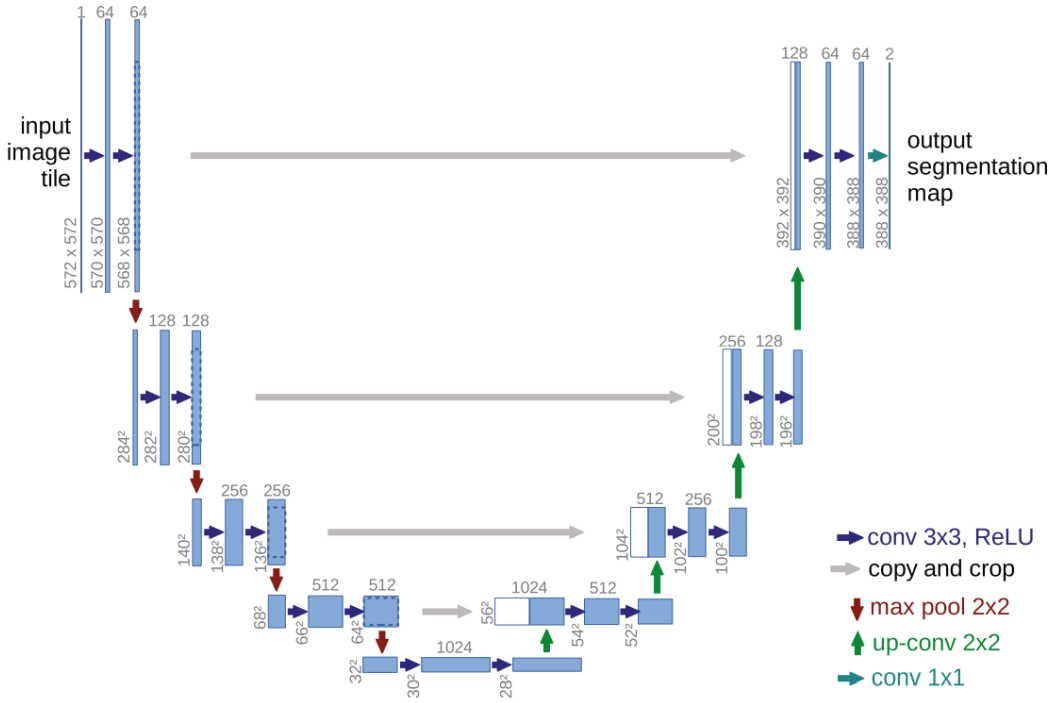


Figure 2.7: Unet architecture. A semantic segmentation network for biomedical applications with the strong use of augmentation. The network consist of encoder that down samples the features and a decoder for up sampling. Courtesy of [7]

the context, and the expanding decoder path identifies the localization of the target area. The network heavily dependent on the annotated images efficiently. The encoder part has a 3x3 convolution features extractor, similar to the FCN-like architecture. The decoder part increases the dimensions and reduces the number of feature maps. The feature map from the encoder is mapped to the upscaled decoder to retain the pattern information. A 1x1 convolution at the output process the feature maps to generate segmentation output by categorizing each pixel of the input image to a particular class. Original U-Net was trained on the electronic microscopic images and outperformed by a large margin on the ISBI challenge. The network is fast and produces a result on 512x512 images in less than a second on the modern GPU [7], [4]. In sequence data, the information from the previous frames can be utilized to segment the current frame with the aim of improved performance compared to the segmentation without the temporal fusion.

2.4 Temporal Fusion in Semantic Segmentation

Semantic segmentation of sequence data aims to assign pixel-wise semantic labels to the video frames. It is an essential task in visual understanding [74]. A strong representation of the feature map is essential for the segmentation task. One common video segmentation approach is performing the image segmentation to each frame independently. However, this approach needs to capture the temporal information of the

dynamic scenes. A standard solution to the problem is to apply semantic segmentation to every frame and add a layer on top to capture the temporal data to extract the better features [75], [76], [77]. However, such an approach does not help improve the performance as the feature needs to be computed at every frame. So a good approach is to apply the segmentation at keyframes and reuse the already computed features for the other frames [78], [79]. A new highly efficient and accuracy neural network-based model is developed for semantic video segmentation called Temporally Distributed Network (TDNet) [8]. In the TDNet feature, extraction is distributed evenly across the sequential frames to eliminate the re-computation. Then these features are combined using the Attention Propagation Module (APM) to get the solid features for accurate segmentation [8]. The pictorial representation of the same is described in Fig 2.8.

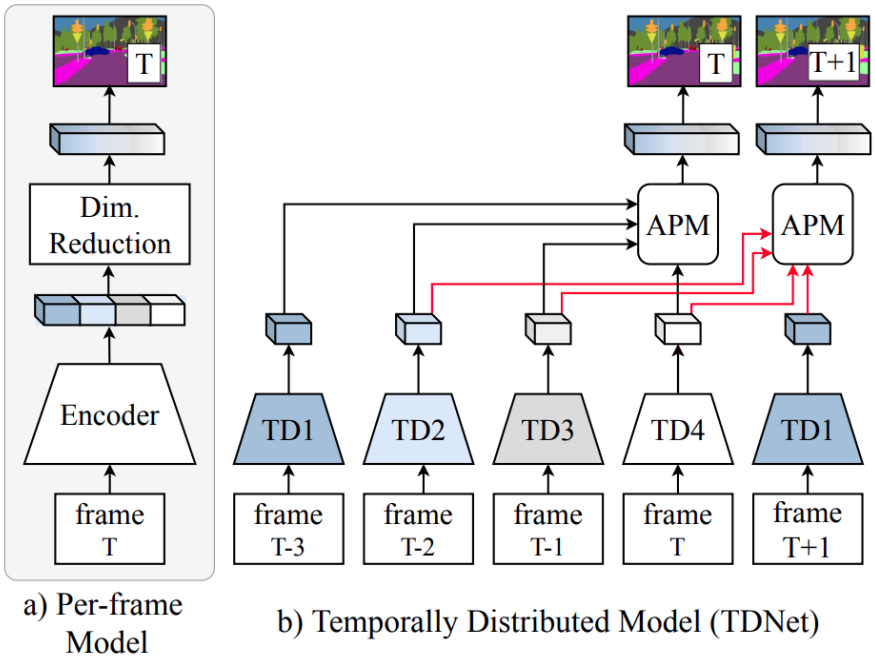


Figure 2.8: TDNet is a efficient and accurate video segmentation framework. Courtesy of [8]

2.5 RQ1: What are the works on state-of-the-art temporal fusion in semantic segmentation?

Segmentation is the process of assigning pixel labels for a given image. Segmentation can be observed in different contexts, such as Video instance segmentation (VIS), Semantic segmentation of the key frames of a video sequence, Video panoptic segmentation, MRI image segmentation, and autonomous driving scene understanding. Youtube - Video Instance Segmentation (VOS) and Scannet are the standard datasets in the semantic segmentation domain.

Youtube VIS data Youtube - Video Instance Segmentation (VIS) is a dataset that extends the image instance segmentation from the image to the video domain. The dataset aims to solve the problem

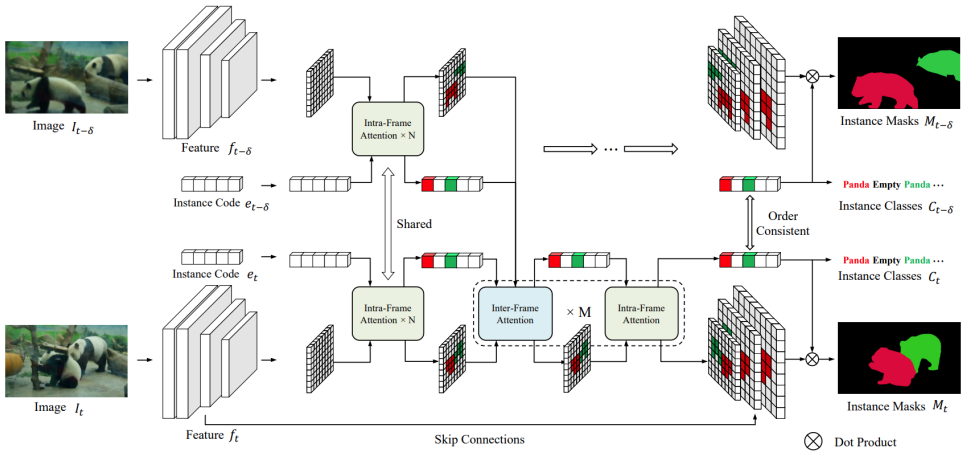


Figure 2.9: VIS proposed framework.An online video instance segmentation framework with instance aware temporal fusion method. Intra frame attention framework for combining instance code and feature map. Inter-frame attention for fusing the hybrid temporal information from previous prediction stage. Courtesy of [87]

of simultaneous detection, segmentation, and tracking problem. Scannet is an RGB-D video dataset containing 2.5 million views with more than 1500 scans. A paper by Xiang Li et al. [87] explains the temporal fusion for online video instance segmentation. The author introduced the concept of an online video framework with a novel, aware temporal fusion method. A cropping-free temporal fusion approach to model the temporal consistency between video frames. A bottom-up online transformer-based network is used to solve the VIS problem. A transformer layer is introduced in the convolutional neural network (CNN) to include the instance information. An attention layer between the frames is used to extract the instance code for the current frame. A skip connection uses low-level contextual information and dynamic convolution to generate the segmentation map—the CNN feature map and the latent code help to represent instance-aware features jointly. An instance code is an LxD vector to the VIS task, where L is the maximum detected instances number in a frame D is the feature instance for each instance. Inter and Intra frame attention module for fusing the temporal information. The network architecture is presented in Fig 2.9. Three types of attention are code-to-code(c2c), code-to-pixel(c2p), and pixel-to-code(p2c), used to construct the relationship between the instance code and feature map. Interframes are used to build the temporal correlation and combine contextual features across frames. Interframe c2c, c2p, and p2c are used to construct the temporal information.

Miran Heo et al. [88] worked on video instance segmentation via Object Token Association (VITA). The work effectively understands the video through the object-centric tokens. The VITA model requires a frame-level detector. The frame-level detector localizes instances using masks without bounding boxes. Two features are generated for the frame-level predictions: a) dynamic 1x1 convolution weight from the frame queries f b) per-pixel embeddings from the pixel decoder. The dot product between the two embeddings is taken in the frame-level predictions. The end-to-end video instance segmentation method VITA is divided into three stages. Firstly, VITA works with the frame-level detector in a frame-independent

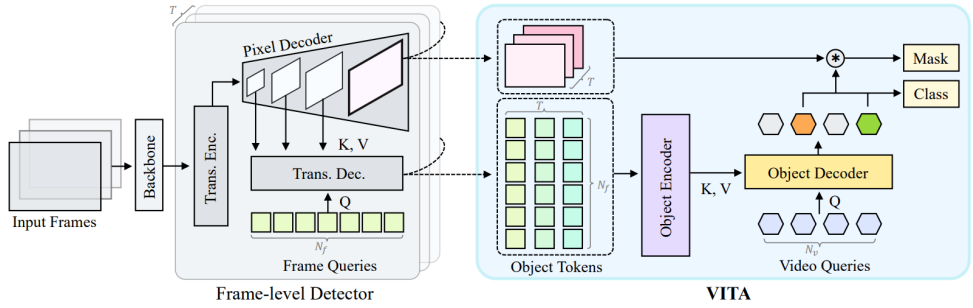


Figure 2.10: The frame level detector takes frame queries and mask features, generates the embeddings, and pass onto the VITA model for mask prediction. Constructing the temporal interactions between the frame queries captures the object-aware knowledge in the spatial scenes. Finally, mask trajectories are obtained from the VITA model. Courtesy of [88]

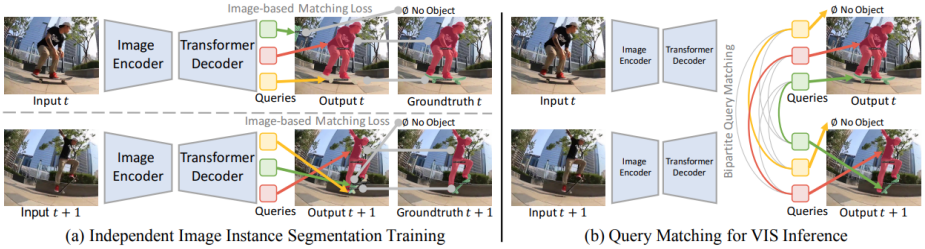


Figure 2.11: (a) MinVIS trained on query-based segmentation individually for every frame. (b) Inference of the video instance segmentation from the segmented image using bipartite matching of the query embeddings. Courtesy of [89]

manner. No computation between the frames is involved. The frame queries that hold the object-centric information are collected throughout the video sequence, and an object encoder is used to build the video-level information between the frames. Moreover, thirdly the decoder combines the information from the frame and video queries which is finally used to find the object mask in the video frames.

A minimal video instance segmentation (MinVIS) by De-An Huang et al. [89] does not require video-based training and can be applied directly to images containing sparse image instance segmentations annotations. MinVIS is a two-stage approach: 1) Independent image instance segmentation on each frame and 2) Instance association between the frames. Image Instance Segmentation has three unique main components a)Encoder to extract features from the image b) Transformer decoder, used to process the output of the image encoder to update the query embeddings c) Prediction head takes the query embeddings to predict the output. Association between the instance segmented frames is calculated by bipartite matching of the respective query embeddings.

The state-of-the-art segmentation models solve mask classification in image segmentation problems. B. Cheng et al. [90] proposed a method to process the video directly rather than the image. This is achieved by feeding the Mask2Former with 3D spatiotemporal features and predicting the 3D volume to track each object instance across time. The Mask2Former works as a general image segmentation model for data

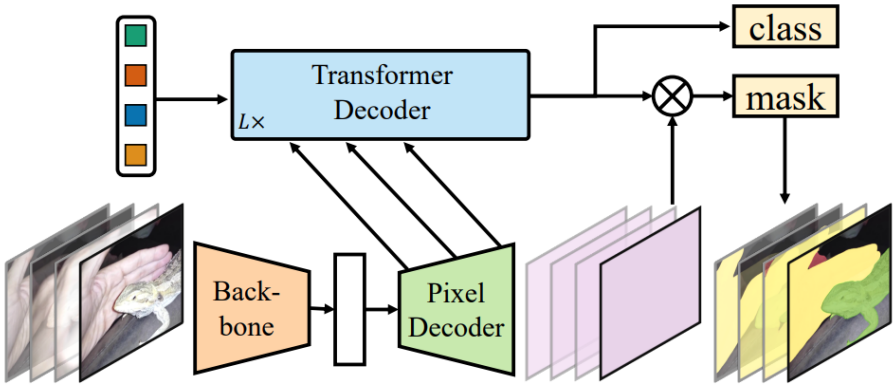


Figure 2.12: A mask2former with video instance segmentation. The architecture can also tackle the semantic and panoptic segmentation task. Courtesy of [90]

with single frames and for data with more than one frame the model segments and tracks instances across frames.

2.6 RQ1 Conclusion

There is much work with respect to temporal fusion. Temporal fusion can be done by taking the previous single feature data or multiple past features and combining them in the current stage. Feature combination can be done at multiple stages of the network. Subjecting the feature can be done in different ways, such as subjecting the feature to Gaussian Process, Long-Short-Term memory, combining the instance code with the encoder features, finding masks in the individual frame and combining them,or taking multiple features from the decoder and passing them to the transformer to model the temporal data. The approaches are made without considering the pose information. Hence, a new temporal fusion approach needs to be developed to model the temporal fusion data to improve performance.

2.7 RQ2: How are the results from RQ1 compared with each other to perform temporal fusion?

All of the above mentioned approaches are trained and tested on the Youtube VIS datasets. The performance of the models are tabulated in table 2.1.

2.8 RQ2 Conclusion

It is evident from the table 2.1 that temporal fusion with the VITA network outperformed the rest of the model performance with Swin-L as the backbone. The comparison is made with the average precision metric. Swin-L transformer is a general-purpose backbone for computer vision tasks. The representation is computed with the transformer. The comparison is made with the Youtube VIS dataset. The dataset does not contain the pose information. VIS network is a temporal fusion architecture containing inter and intra-frame fusion for a fusion of temporal data. From the result table 2.1. The method of temporal

Method	AP	AP50	AP75	AR1	AR10
HIATF for VIS 2019 [87]	41.3	61.5	43.5	42.7	47.8
VITA ResNet-50 2019 [88]	49.8	72.6	54.5	49.4	61.0
VITA ResNet-101 2019 [88]	51.9	75.4	57.0	49.6	59.1
VITA Swin-L 2019 [88]	**63.0**	**86.9**	**67.9**	**56.3**	**68.1**
VITA 2021 [88]	45.7	67.4	49.5	40.9	53.6
Min VIS 2019 [89]	61.6	83.3	68.6	54.8	66.6
Min VIS 2021 [89]	55.3	76.6	62.0	45.9	60.8
Mask2former 2019 [90]	60.4	84.4	67.0	-	-
Mask2former 2021 [90]	52.6	76.4	57.2	-	-

Table 2.1: Comparison of temporal fusion model performance with respect to Youtube VIS dataset

fusion by taking only a single frame data from the previous frame is a better approach than computing the temporal fusion of all the data at a time.

2.9 Limitations of Previous Work

The perception system of the modern ADAS uses segmentation to understand the surrounding environment by capturing the environment with the help of modern cameras. The high FPS data collected by the camera are in a continuous sequence where every frame is related to its previous frames. In general, setting segmentation is performed on these frames to understand the object and its boundaries, the number of objects, types of objects present in the frame. A work by Hou et al. [2] integrates the camera pose data into the computation of the depth maps; however, a similar strategy is not studied for a segmentation task. Additionally, the study of using temporal data fusion in the encoder and decoder based network latent space with LSTM for semantic segmentation represents a novel approach for incorporating past frame information in the current step. This thesis aims to study the impact of temporal pose data on the computation of the semantic segmentation and to take the previous frame data onto the current frame semantic segmentation task using the LSTM network is studied.

3

Methodology

The methodology section details the dataset used for performing the experiments, preprocessing of the dataset, and experimental design.

3.1 Dataset

To conduct the experiments, Scannet and Virtual KITTI 2 are used. Scannet is an indoor scene dataset, and Virtual KITTI 2 is a synthetic dataset. This dataset contains the continuous video sequence data and is explained in detail in the below section.

3.1.1 ScanNet [91]

ScanNet is a video sequence dataset with 1513 scenes captured in 707 distinct spaces. The dataset contains a total of 2.5M RGB-D images. The dataset was initially designed for the task of indoor scenes 3D reconstruction, including 3D object classification, semantic voxel labeling, and CAD model retrieval [91], [92]. The dataset contains the details of the 3D camera poses, surface reconstruction, and instance-level semantic segmentation. One sample of the data is shown in the figure

The 3D representation of the sample data is presented in the figure3.2. There are 40 classes in the dataset, with different varieties under each category. The complete list of the classes is presented in Table B.1, B.4, and B.3. Twenty users collect datasets at different countries' locations. Since the dataset is vast, a subset is taken for performing the experiments. Totally 186 video sequence data is taken for conducting the experiments. These video sequence data are further split into 149 sequences for training the model and 36 sequences for testing. The distribution of the scannet dataset labels is shown in figure 3.3. There is a high number of pixels in the dataset belonging to the Wall, Other, Floor, and Chair categories. The dataset represents indoor scenes with different varieties. Hence experiments are conducted by combining the low pixel distribution class into single categories to balance the pixel distribution.

3.1.2 Virtual KITTI 2 [93]

The virtual KITTI dataset is the first released to the public domain with driving applications. It is a synthetic dataset representing the simulation of the vehicle in different environmental conditions. And it

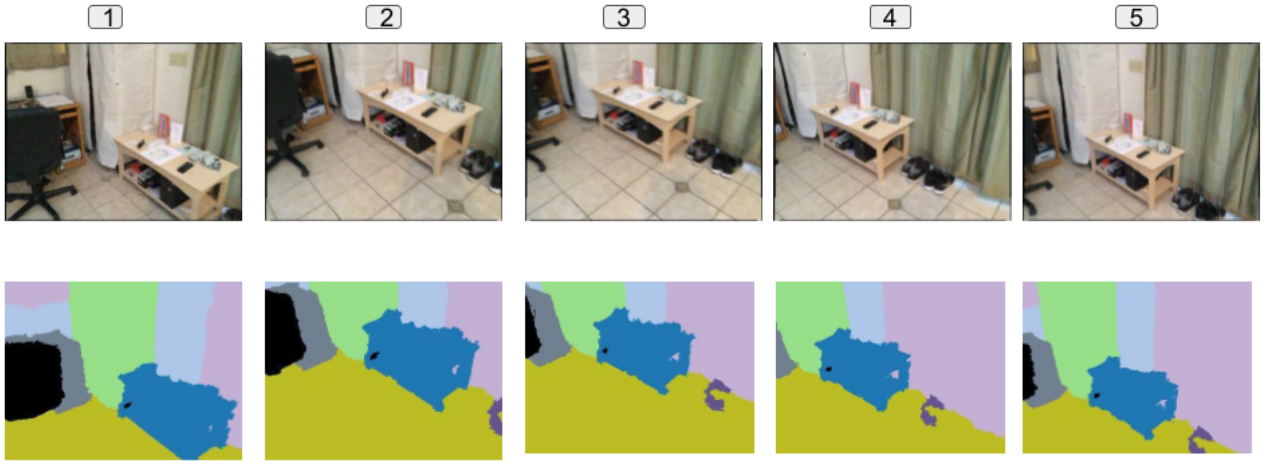


Figure 3.1: Sample of Scannet dataset continuous frame rgb and semantic label. Courtesy of **??**

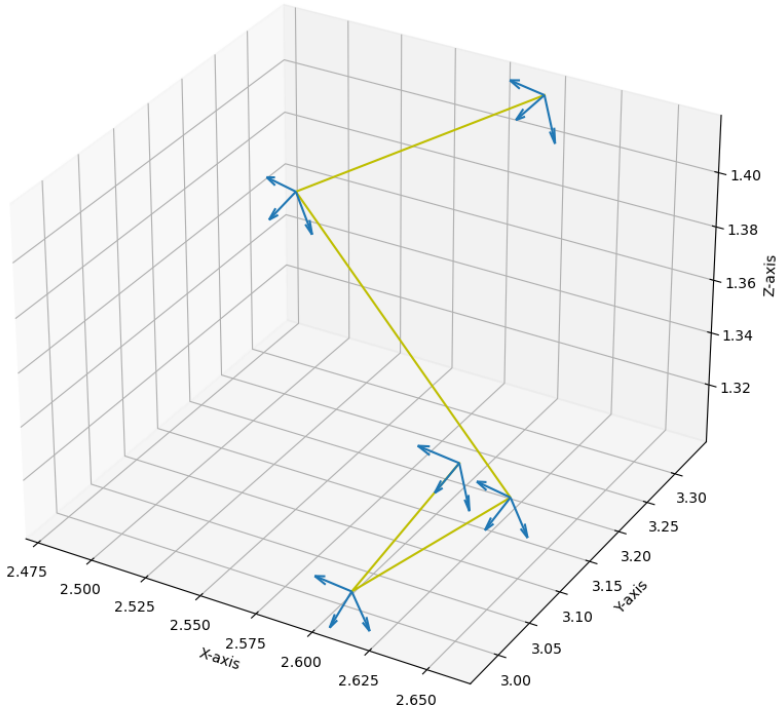


Figure 3.2: Sample of Scannet dataset pose. Each point represent the pose of the camera in 3D frame when a particular frame is captured. The representation shows how the camera is moving in the 3D world.

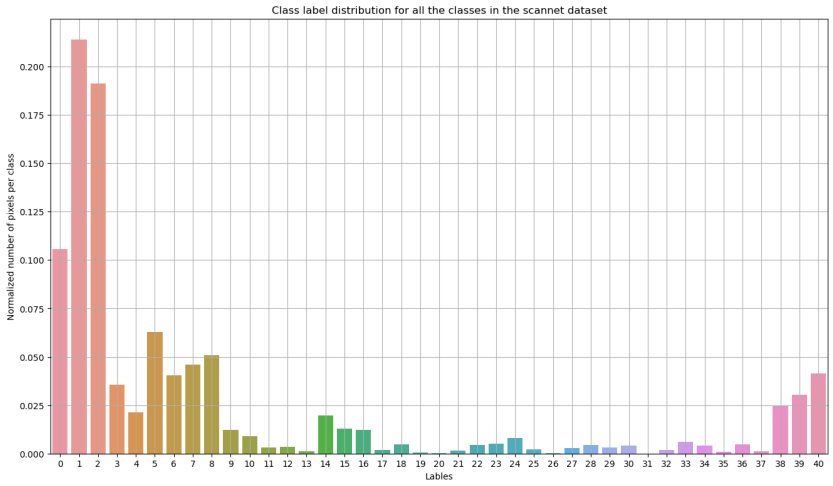


Figure 3.3: Scannet dataset class distribution. Horizontal axis represent the classes in Scannet dataset and vertical axis represent the normalized number of pixels per class. There are high number of pixels in entire data belonging to class 1 and class 2.

is a cost-effective alternative to real-world data. Virtual KITTI 2 is the next-generation dataset with an improved photo-realistic and featured version of the original virtual KITTI dataset. The virtual KITTI 2 contains stereo camera views to expand the application areas. The dataset contains the exact five sequences clones with camera 0 representing the same dataset as virtual KITTI, and camera 1 is 0.5327m to its right. Each camera contains RGB, class segmentation, instance segmentation, depth, forward and backward optical flow, and forward and backward scene flow images. Each sequence contains camera parameters, vehicle color, pose, and bounding boxes. One sample from the sequence and corresponding pose representation in the 3D plane is shown in Figure 3.4 and 3.5 respectively.

3.2 Data Collection and Preprocessing

After submitting the agreement terms, the RGB-D scannet dataset can be downloaded from the python script sent to the user from the dataset development group. The description and format of the dataset can be found on GitHub [94]. There are 1500 scans. Any specific scans, and file types, can be downloaded. There is multiple information regarding a scan in a file. These files are further processed to color, label, pose, and depth data for each scan. More details are documented in the GitHub [95] repository. The raw RGB image is normalized before passing it into the model for quick convergence of learning. The input image is a 3-channel RGB data in .jpg format, the label is a single-channel image in .png format with 40 classes total, and the pose data is in text format. The pose data contains the rotation matrix and transnational vector inhomogeneous matrix format. The vkitti dataset is also in a similar format. However, there is a total of 15 classes in the vkitti dataset. The raw RGB image, label, and pose data are shown in figure 3.8.

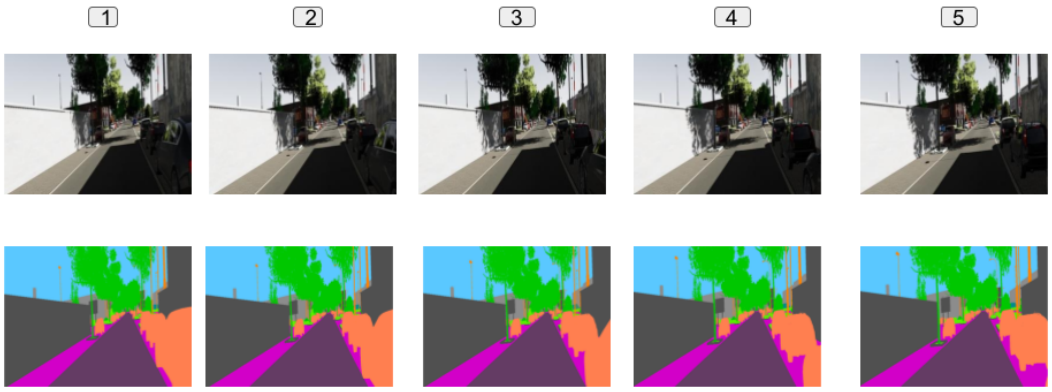


Figure 3.4: Sample of Virtual Kitti 2 dataset. Top row represent the continuous frame number followed by rgb and semantic labels. Courtesy of [93]

3.3 Experimental Design

The experiment aims to incorporate temporal fusion in the latent space to understand the impact of past frame information fusion in future frame segmentation prediction. The Unet model **??** perfectly suits this experiment. The unet model consists of an encoder and a decoder network with latent space encoding in between. Three types of experiments were conducted with the unet model. In the first type, the plain vanilla unet model is taken in the second type of experiment unet model with the gaussian process; the third type is the unet model with Long Short Term Memory (LSTM). In the last two experiments, the latent space encoding is subjected to temporal fusion by the gaussian process and LSTM.

3.3.1 U-Net Vanilla model

U-Net vanilla model is a simple semantic segmentation model with encoder-decoder architecture. The encoder is a contracting path, and the decoder is an expanding path. The input image is fed into the encoder, consisting of a convolutional neural net. The architecture consists of two 3x3 2d convolutional layers followed by a rectified linear unit (ReLU) and batchnorm2d with 2d downsampling Maxpool layer. At every step of the convolutional block, the number of out channels/ feature channels is doubled. From 64 features in the first convolutional block output to 1024 features at the downsampled fifth layer or the latent space encoding. The upsampling layer consists of a 2d convolution with a halved feature map at every step, followed by a concatenation of the corresponding feature map from the encoder path. The convolution2d consists of two 3x3 convolutions, each followed by a ReLU function. At the final layer of the decoder, a 2d convolutional layer was introduced to map the previous convolutional block output to the predicted class labels. There are 23 convolutional blocks in the entire network.

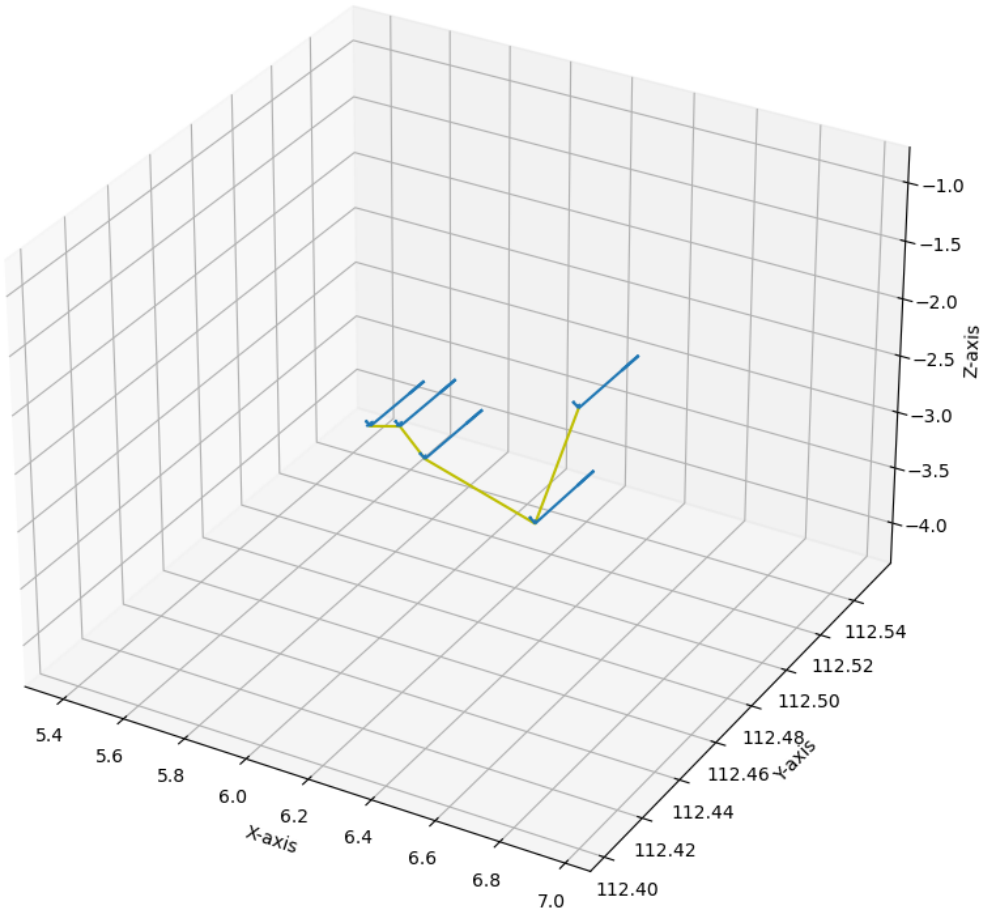


Figure 3.5: Sample of Virtual Kitti 2 dataset pose. Each point represent the pose of the camera in 3D frame when a particular frame is captured. The representation shows how the camera is moving in the 3D world.

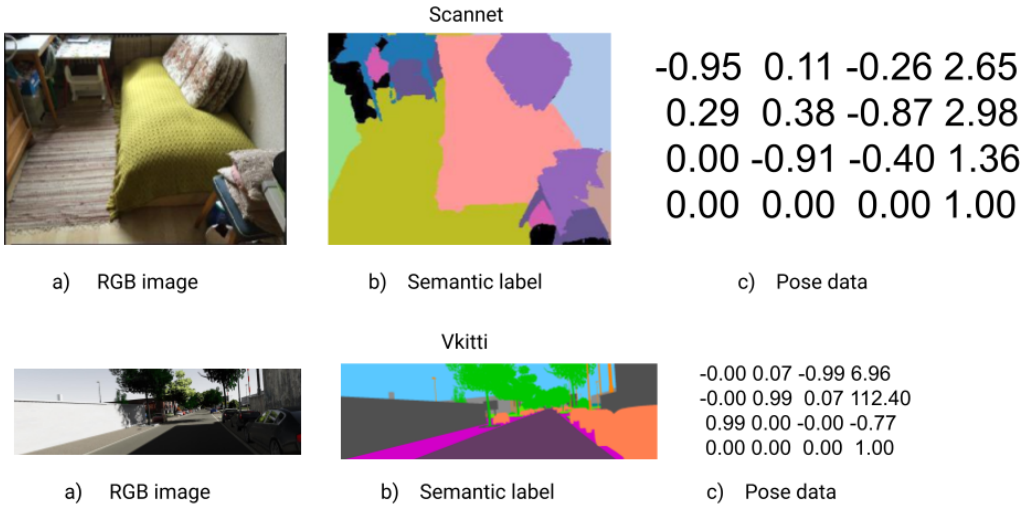


Figure 3.6: RGB, Label and Pose sample of Scannet and Vkitti data

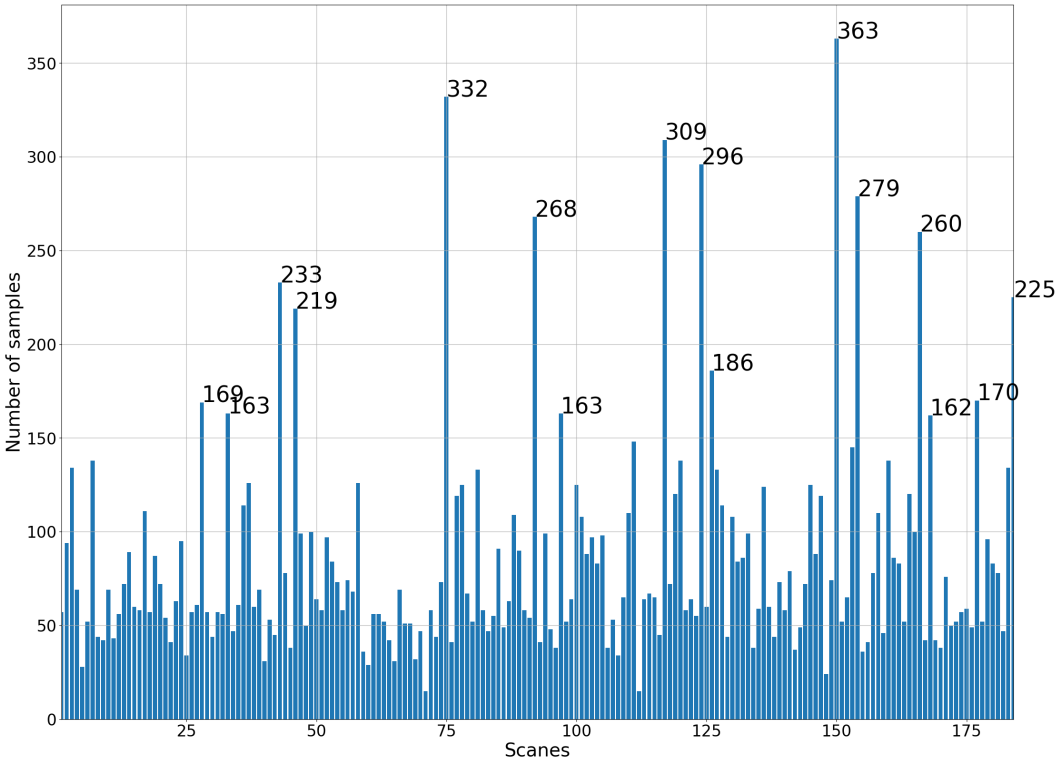


Figure 3.7: Scannet data distribution. Horizontal axis represent the video sequence number and vertical axis represent the number of frames in each video sequence.

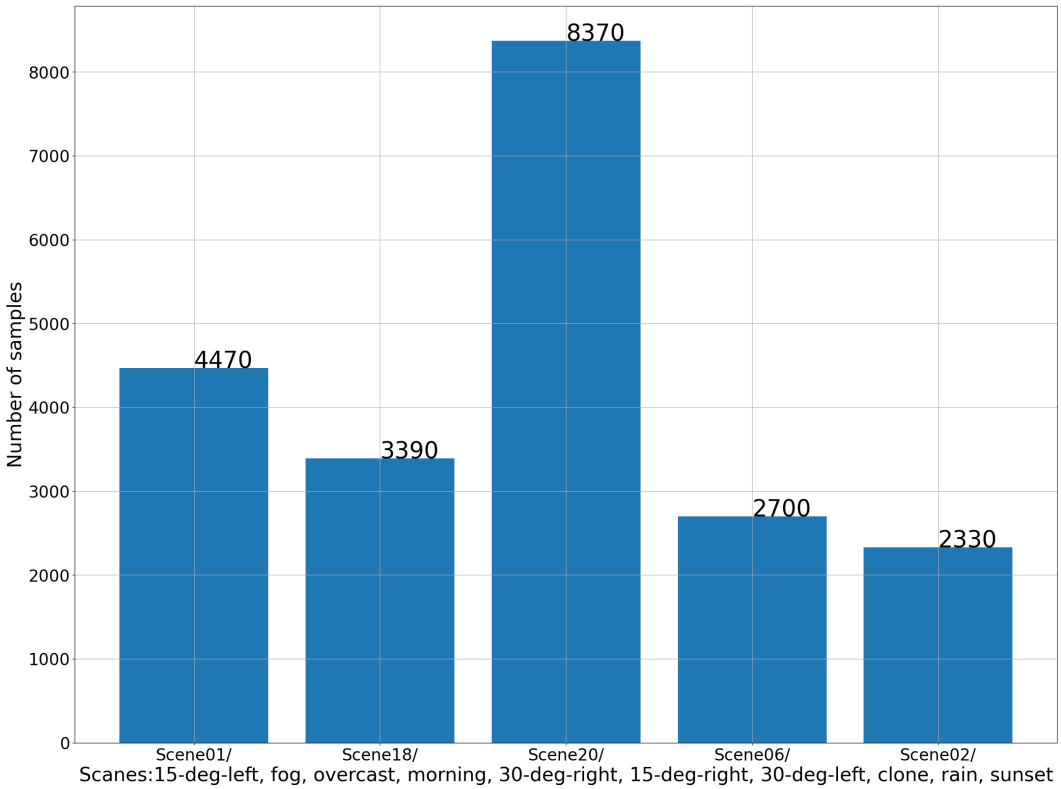


Figure 3.8: Vkitti data distribution. Horizontal axis represent the scenes and under each scene there is 15-deg-left, fog, overcast, morning, 30-deg-right, 15-deg-right, 30-deg-left, clone, rain and sunset categories.

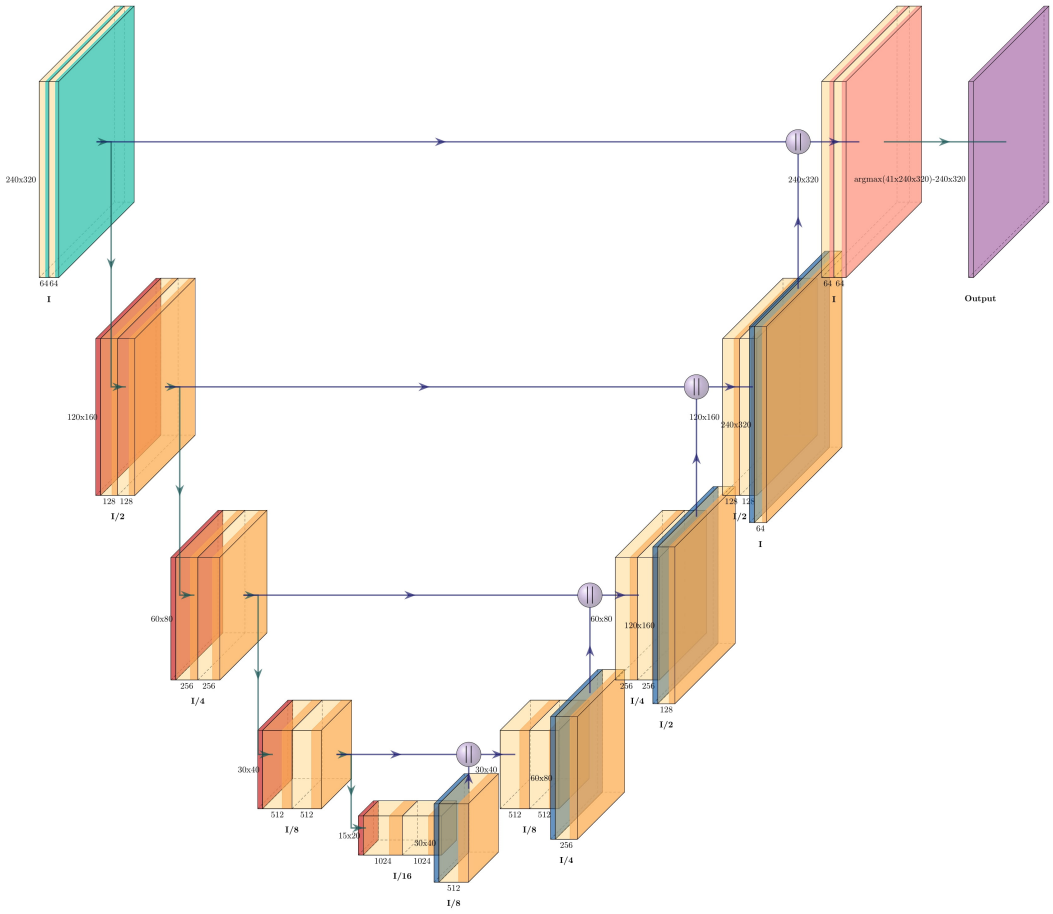


Figure 3.9: Unet model architecture. Generated with modification PlotNeuralNet

3.3.2 U-Net with Gaussian process

In the second type of experiment, the latent space encoding of the U-Net model is taken and subjected to the gaussian process. The gaussian process ensures that the frames close to each other have similar latent space encodings. This is done by building the kernel of the gaussian process with a distance matrix. A probabilistic prior on the latent space accounts for the prior knowledge that poses very close have similar latent space encoding to the poses that are very far from each other. This information is encoded in the covariance matrix. The distance measured between the camera poses is calculated with a metric to define the closeness. The distance between the camera poses is calculated with the help of work by Yuxin Hou et al. [02] and Mazzotti et al. [97]. This measures the distance between the rigid body poses. The pose-distance measure between the two camera poses P_i, and P_j is defined in equation 3.1. Where t is the translation vector, R is the rotational vector, I is the identity matrix, and $tr()$ is the trace of the matrix.

$$D[P_i, P_j] = \sqrt{||t_i - t_j||^2 + \frac{2}{3}tr(I - R_i^T R_j)} \tag{3.1}$$

The covariance function is defined from this distance matrix $D[P_i, P_j]$. The covariance function is chosen from the Matern class [97] [98] in equation 3.2.

$$k(P, P') = \gamma^2(1 + \frac{\sqrt{3}D[P, P']}{l} exp(-\frac{\sqrt{3}D[P, P']}{l})) \tag{3.2}$$

The hyperparameter γ^2 and l define the magnitude and length-scale of the processes. Independent GP priors to all values in Z_i, and the output of the encoder is assumed to be the noise corrupted version of the expected latent space encodings. The GP regression model is defined from this initialization settings and defined in the equation 3.4 3.3 [02]. The architecture of the unet model with gaussian process is depicted in the figure 3.10. The encoder output is assumed to be noise corrupted hence the term $\epsilon_{j,i}$.

$$z_j(t) \sim GP(0, k(P[t], P[t'])) \tag{3.3}$$

$$EncoderOutput = z_j(t_i) + \epsilon_{j,i}, \epsilon_{j,i} \sim N(0, \sigma^2) \tag{3.4}$$

3.3.3 U-Net with Long Short Term Memory (LSTM)

In the third type of experiment, the U-Net model is subjected to temporal fusion with the help of Long Short Term Memory (LSTM) by passing the latent space encoding onto the LSTM convolutional cell. The overlapping information in the consecutive frame data is passed onto the future frame prediction by incorporating the convolutional LSTM cell to improve performance and model the spatiotemporal relations compared to the vanilla model. The partial scene geometry information from the previous frame is used in the current step frame prediction. A hidden state propagation approach from the LSTM method is used to pass the latent space information. The adaption of temporal fusion with LSTM is inspired by

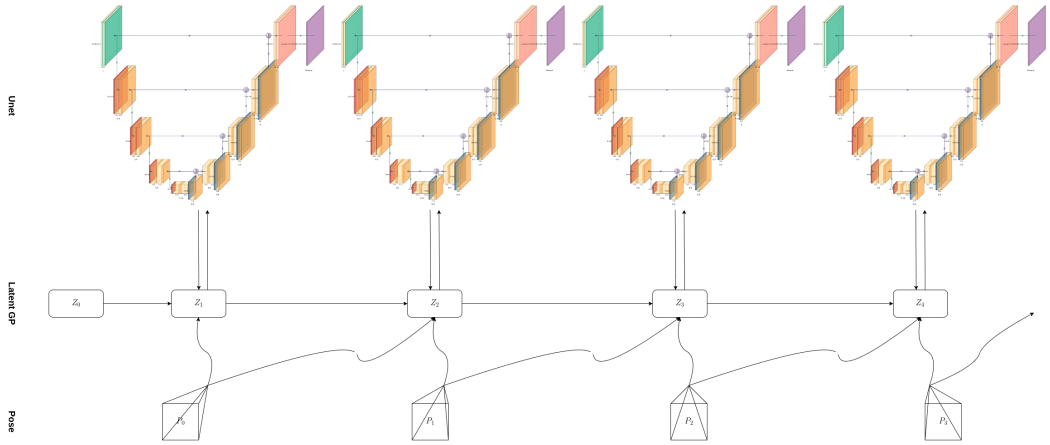


Figure 3.10: Unet model architecture with temporal fusion in latent space using Gaussian Process.The latent space encoding is propagated forward. During every computation the Gaussian Process takes latent space encoding, previous updated latent space and the current and previous pose to update the current latent space encoding. Generated with Diagrams.net

the work of Arda Düzçeker et al. [13]. The author introduced a convolutional LSTM cell in the latent space to learn the shared information between the frames to predict the depth of objects in video sequence data. The convolutional LSTM cell is inspired from [99]. The original LSTM version is explained in the article [100]. The hidden state H and cell state C are initialized to a value. Let X denote the output of the bottleneck encoder network, then the logic to compute the hidden state and current state can be calculated as below (Courtesy of [13])

$$i_t = \sigma(w_{xi} * X_t + w_{hi} * H_{t-1}) \tag{3.5}$$

$$f_t = \sigma(w_{xf} * X_t + w_{hf} * H_{t-1}) \tag{3.6}$$

$$o_t = \sigma(w_{xo} * X_t + w_{ho} * H_{t-1}) \tag{3.7}$$

$$g_t = ELU(layernorm(w_{xg} * X_t + w_{hg} * H_{t-1})) \tag{3.8}$$

$$C_t = layernorm(f_t \bigodot C_{t-1} + i_t \bigodot g_t) \tag{3.9}$$

$$H_t = o_t \bigodot ELU(C_t) \tag{3.10}$$

Where $*$ denotes the convolution, $\bigodot$ is the Hadamard product, σ is the sigmoid activation function, and w is convolution filter weights. Figure 3.11 presents the exact pictorial representation. Each frame of the sequence video is passed onto the bottleneck U-net model in order, and prediction is made for individual frames. The output from the encoder network is passed through the LSTM cell. The output from the first LSTM cell act as the input to the subsequent LSTM cell computation; thereby, it carries the learned information forward from one frame computation to the following frame computation.

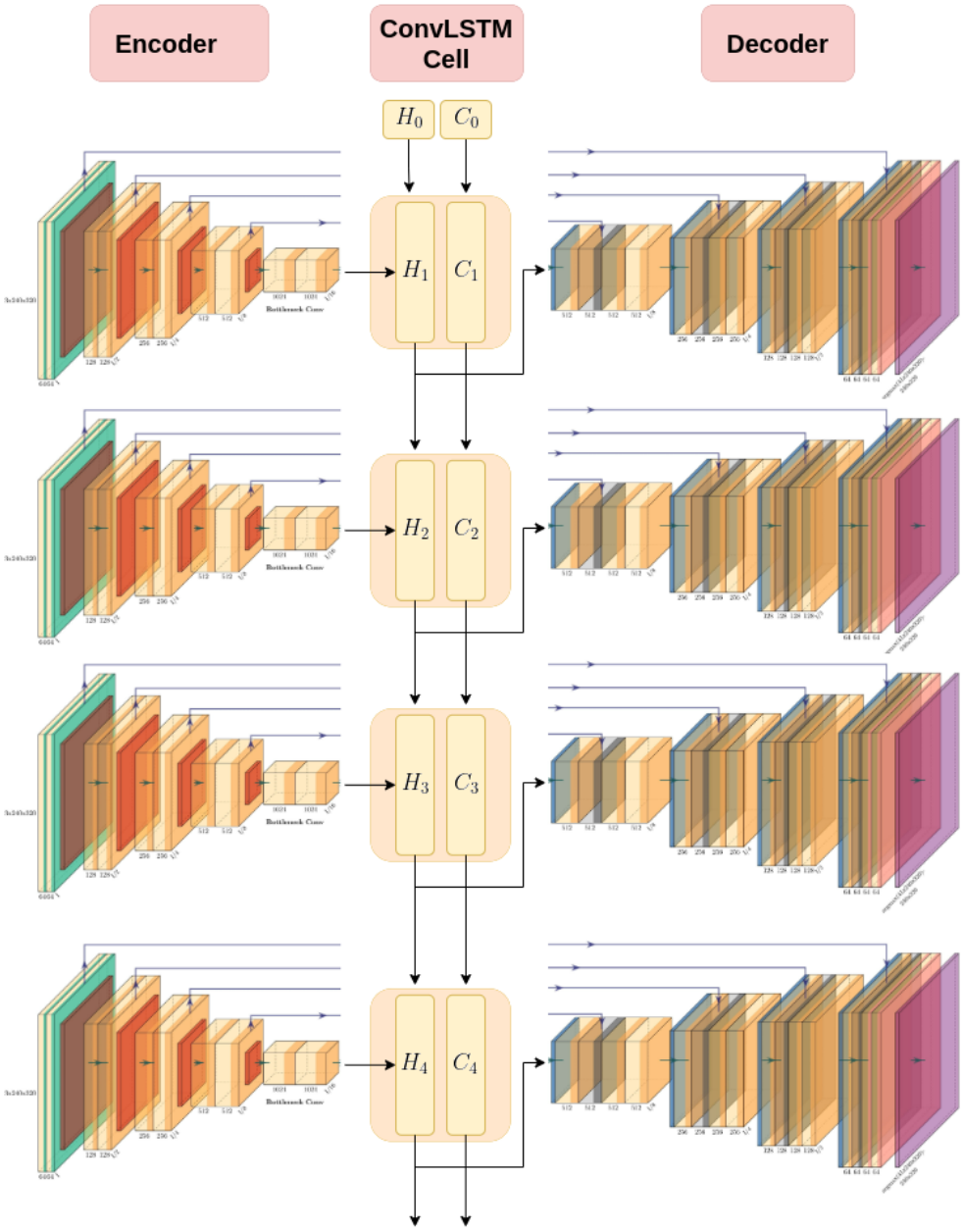


Figure 3.11: Unet model architecture with temporal fusion in latent space using the ConvLSTM cell. At every frames semantic label computation the previous convolutional LSTM cell hidden and cell state are passed to the current cell computation. Thereby modeling the temporal dependency in the consecutive frames. Generated with Diagrams.net

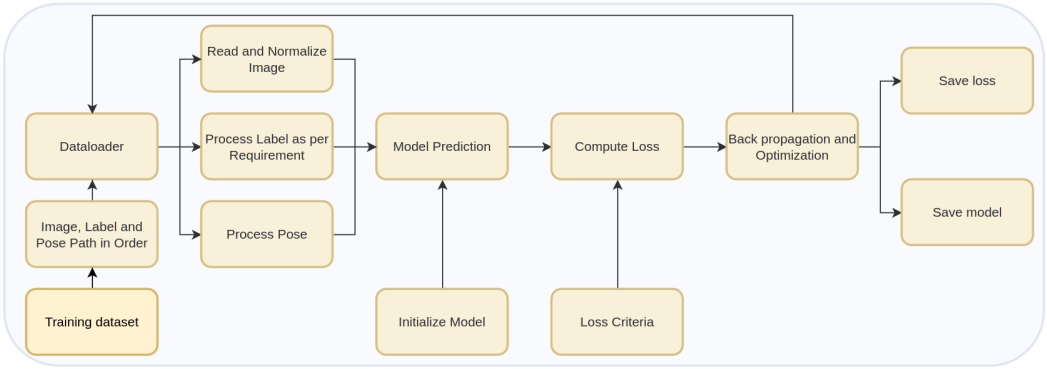


Figure 3.12: Training pipeline. Generated with Diagrams.net

3.4 Training and Evaluation Pipeline

Datasets are split into training and evaluation sets. During the training phase, the image, label, and pose paths are added to a list and then passed onto the data loader. The image data is read and normalized. The experiment is conducted with a different number of classes by merging the classes. This is done in the label processing stage. A model is initialized with different parameter settings. The processed image, label, and pose data are passed onto the initialized model. A loss criterion is defined to compute the losses. Loss is computed from the ground truth value and the model prediction. Backpropagation from the loss is calculated, followed by the optimization step. In the final stage, the loss and model are saved. The data loader loads data in batches. The process is repeated until all the data in the data loader iterate through the entire data once. The entire process is repeated for a different number of epochs. Model performance is monitored during each epoch run and stopped once the loss reaches a small value and the model fits the training data.

The evaluation dataset path is stored in the list during the evaluation process. The individual list contains Image, Label, and Pose path information. Since the dataset is a continuous sequence data, the path of the data stored in the list is in order. In most cases, there is overlapping information between the consecutive frames. In the evaluation, datasets are passed onto the data loader. Images are normalized from this data loader, and labels are processed per the experiment's requirement. Pose information is in text format. The text data is read, passed to a list, and returned by the data loader. The trained model is loaded, and the processed data from the data loader is passed onto the model. The predicted output from the model is evaluated against the evaluation metric, and the results are saved.

3.5 Hardware Configuration

Training of the models is carried out in different sources. The workstation used to train and evaluate the models involve NVIDIA GeForce RTX 3070 Laptop GPU, 8192 MB, Tesla T4, 12680 MB, Google Colab, and Nvidia Tesla V100 PCIe GPU with 5120 Cuda cores, and 640 Tensor cores, 16 GB HBM2 memory University cluster.

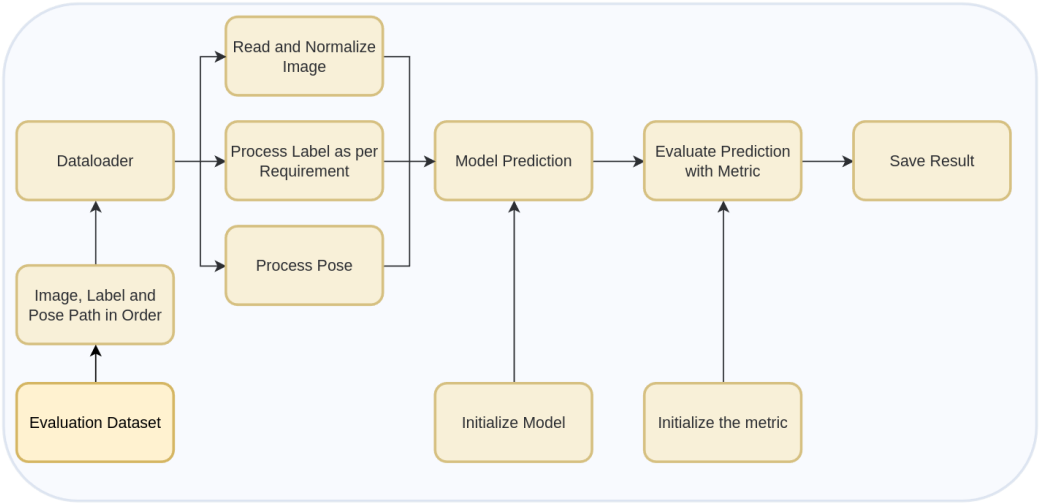


Figure 3.13: Evaluation pipeline. Generated with Diagrams.net

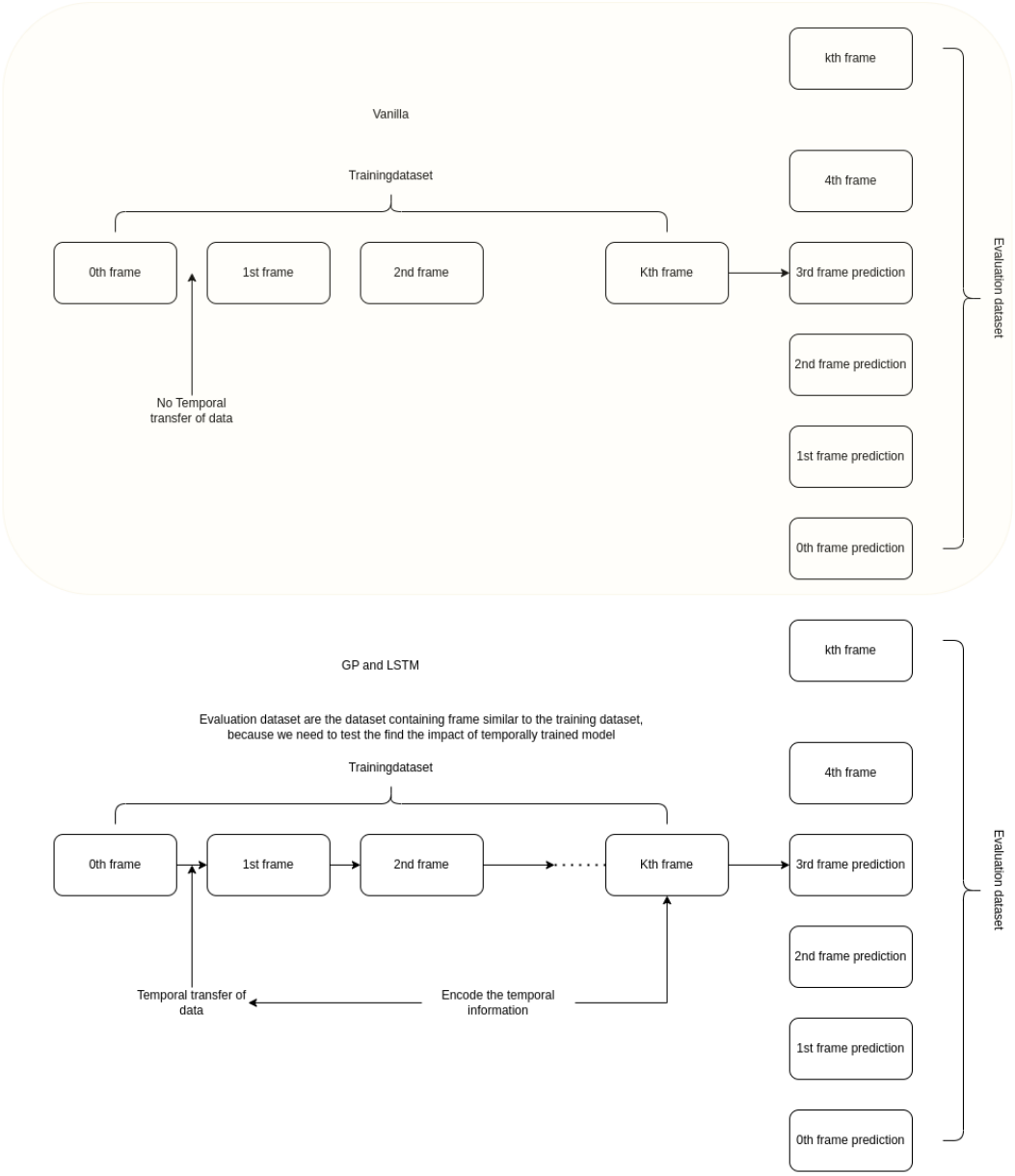


Figure 3.14: Pictorial representation of training and evaluation procedure. The picture represent the how the training and evaluation is conducted with and without the temporal fusion. In the Vanilla network the model is trained on individual frames without temporal fusion and in the GP and LSTM model temporal data is propagated from one frame to another. Generated with Diagrams.net

4

Evaluation and Experimental Result

The evaluation and experimental result chapter contain the metrics used to evaluate the conducted experiment and the discussion on the research questions. The results of the research question are answered in the subsequent sections, listing the result in a table. Three research questions answered in the section are the results of the state-of-the-art temporal fusion techniques, how the results are compared to each other, and finally, the cross-transfer of the temporal fusion techniques from depth estimation to semantic segmentation. The model is trained and evaluated on three datasets Scannet [80], Virtual Kitti [81], and VIODE [82] datasets.

4.1 Evaluation Metric

The proposed model needs to be validated to understand the impact of the newly trained model. To validate the model, different evaluation metrics are proposed. The description of the proposed model is listed below.

4.1.1 Pixel Accuracy

Pixel accuracy is commonly defined as the percent of pixels in an image correctly classified into a particular class. Accuracy is defined below

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

where

TP = True Positive

TN = True Negative

FP = False Positive

FN = False Negative

Per class mean pixel accuracy (mPA)

Per class mean pixel accuracy is the average pixel accuracies of all the classes.

Pixel accuracy (PA) and Mean pixel accuracy (mPA) can also be defined below [83]

$$\textbf{PA} = \frac{\sum_{j=1}^{k} n_{jj}}{\sum_{j=1}^{k} t_j}$$

$$\textbf{mPA} = \frac{1}{k} \frac{\sum_{j=1}^{k} n_{jj}}{\sum_{j=1}^{k} t_j}$$

where n_{jj} is the total number of pixels both classified and labeled as class j.

4.1.2 IoU

Intersection over Union (IoU), also known as the Jaccard index, quantifies the overlapping between the target mask and the predicted output. In other words, the number of standard pixels between the target and prediction masks by the total number of pixels that exist between the masks [84]. A pictorial representation of the same is presented in Fig 4.1

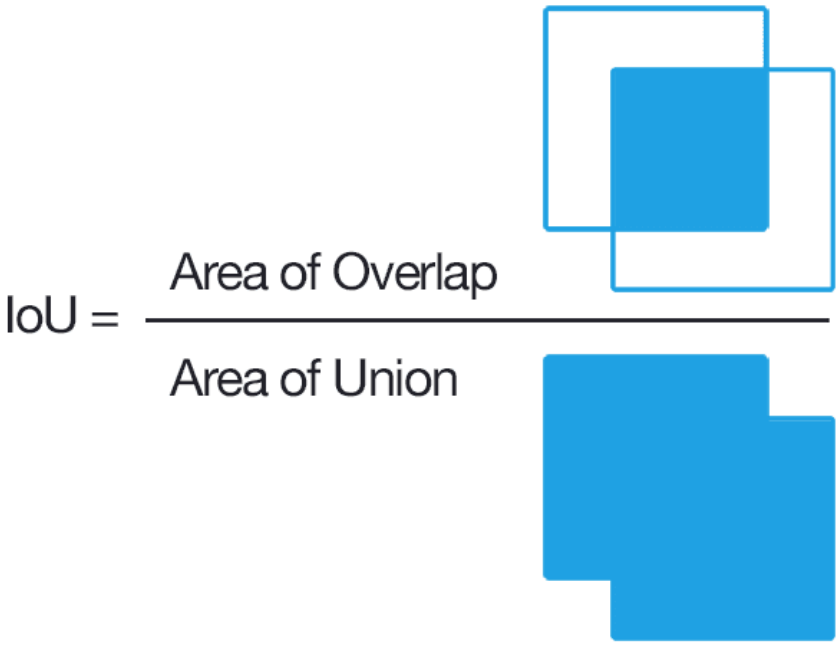


Figure 4.1: Definition of IoU. Courtesy of [9]

IoU is calculated for each class separately and then averaged over all the classes to provide mean IoU.

$$\textbf{IoU} = \frac{\sum_{j=1}^{k} n_{jj}}{\sum_{j=1}^{k}(n_{ij} + n_{ji} + n_{jj})}, i \neq j$$

where n_{ij} is the pixels which are labeled as class i but classified as class j and n_{ji} is the total number of pixels labeled as class j, but classified as class i. [83]

$$\textbf{mIoU} = \frac{1}{k} \sum_{j=1}^{k} \frac{n_{ij}}{n_{ij} + n_{ji} + n_{jj}}, i \neq j$$

Frequency-weighted IoU (FwIoU). It is a metric derived from the mIoU, which weighs each class's importance depending on appearance frequency using t_j [83]

$$\textbf{FwIoU} = \frac{1}{\sum_{j=1}^{k} t_j} \sum_{j=1}^{k} t_j \frac{n_{jj}}{n_{ij} + n_{ji} + n_{jj}}, i \neq j$$

4.2 Hypothesis

Semantic segmentation is a vital task in the computer vision domain that assigns labels to every pixel in an image. Semantic segmentation is applied in 2D images, Video data, and 3D or Volumetric data. The video sequence data contains multiple image frames stacked together. Given the high frame rate of the video, the consecutive frames are related to each other due to the overlapping regions in the successive frames. In most semantic segmentation tasks, the segmentation is performed on each frame or the keyframes. Past information can be fused into current step computation by taking the learned features from previous frames [8].

The work hypothesizes that the performance improves by fusing the past information onto the current frame segmentation computation by the following approaches.

(i) Subjecting the latent space encoding of the Unet model to the Gaussian process by taking the pose of the current frame and the previous frame, the performance improves in comparison to segmentation without temporal fusion since the overlapping information present in the past is used for computation of the current frame segmentation.

(ii) Placing the ConvLSTM cell at the latent space encoding of the Unet model to transfer the geometry information from the past to the current step helps improve semantic segmentation performance.

The approach is cross-transferred from the depth estimation task [2] [13].

4.3 RQ3: How to cross-transfer the temporal fusion technique to semantic segmentation?

The U-net model was developed to categorize biomedical images. Currently, the model is used in other application areas as well. The original network is mainly based on the data augmentation strategy. The

network consists of an encoder and decoder along with the interconnection between different layers of encoder and decoder. The expanding path of the U-net model helps capture the objects' location in the image. The vanilla U-net model code is referenced from the Kaggle competition [85]. The result presented below is for the vanilla U-net model trained on the scannet dataset [80]. The model is trained for 300 epochs. The plain vanilla model is trained with a neural network-based unet model. The experiment is conducted in three ways

- Considering all the classes
- Containing only two classes by combining the low pixel distribution classes together (Wall and Other)
- Containing three classes by combining classes with low pixel distribution (Wall, Furniture, and Other)

The parameter used to train a plain unet model is tabulated in 4.11.

4.3.1 Combining scannet dataset classes for experiment

There are 41 classes [B.1 , B.4 , B.3] present in the entire Scannet dataset.

To experiment, 185 indoor video sequence data are considered. The distribution of the entire scannet data classes is presented in Fig 4.13. Of 185 sequences, 149 sequences are taken for training and the remaining 36 for testing. The distribution of pixels in the training data and testing data represented in Fig 4.3.

The pixel distribution of the scannet dataset classes is not uniform. Experiments were conducted in three ways; in the first approach, all the classes were considered for experimenting, and no combining of the classes. In the second approach, all the classes other than the wall class are combined to assign a class label of "zero," and the "Wall" class is chosen as the second class. The distribution of the pixels is depicted in Fig 4.4. The Wall, Furniture, and Other classes combined have a high pixel distribution; hence all the furniture classes are combined as class 2nd, Wall as the first class, and the remaining classes are combined into 0th class as Other classes, resulting in three unique classes. The distribution of the three classes is presented in Fig 4.5.

4.3.2 Distance and Kernel matrix

A distance matrix is a square matrix that represents the distance between a pair of objects. Distance matrix can be constructed from the pose information of the captured video sequence. The closer the two frames are in a sequence, the distance between them is; the smaller same can be represented with the matrix plot. Two types of the distance matrix are plotted in Fig 4.7 and 4.8. In the first type of plot, an ordered frames pose is represented in the x and y-axis. The distance of the pose with respect to itself is zero, and as the frame number increases, the distance keeps increasing. In the second type of unordered

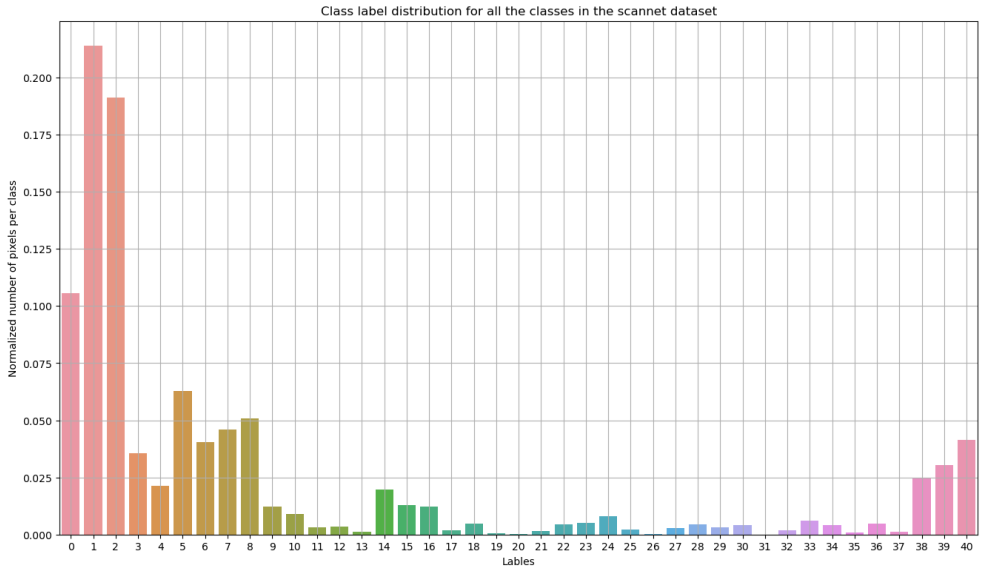
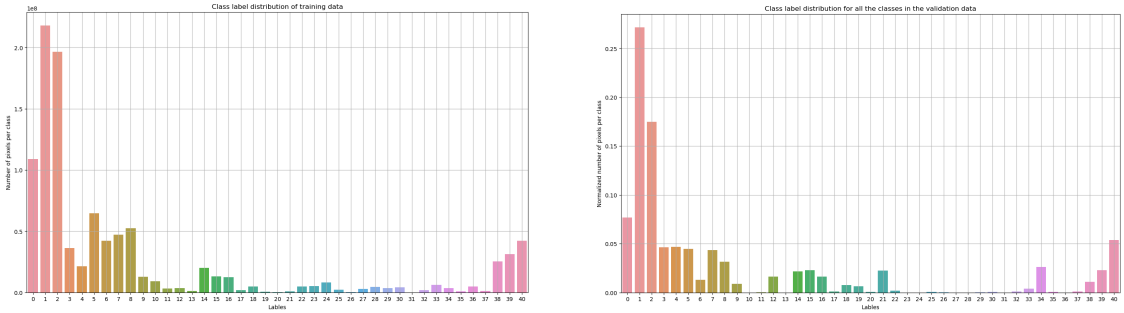


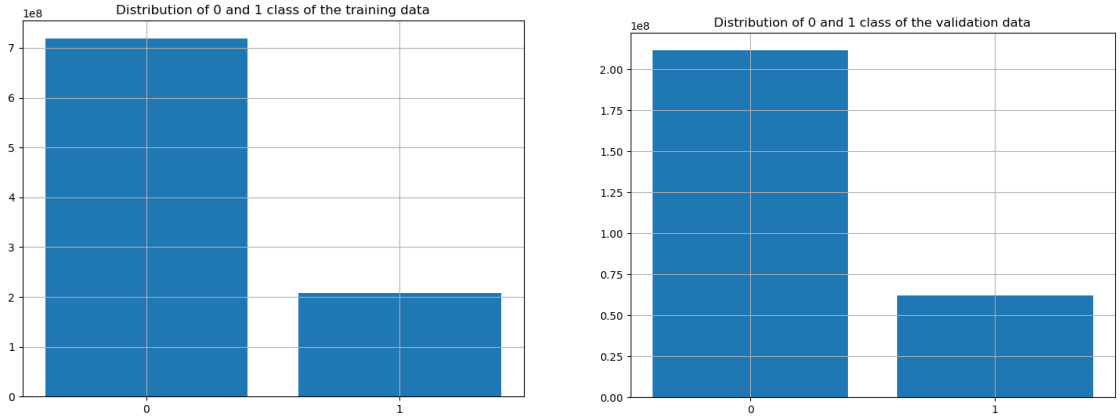
Figure 4.2: Per class pixel distribution of the entire scannet dataset. X-axis represent the labels in the Scannet dataset and Y-axis represent the number of pixels per class. High number of pixels belongs to class 1 and class 2



(a) Per class pixel distribution of the training dataset pixel class label

(b) Per class pixel distribution of the testing dataset pixel class label

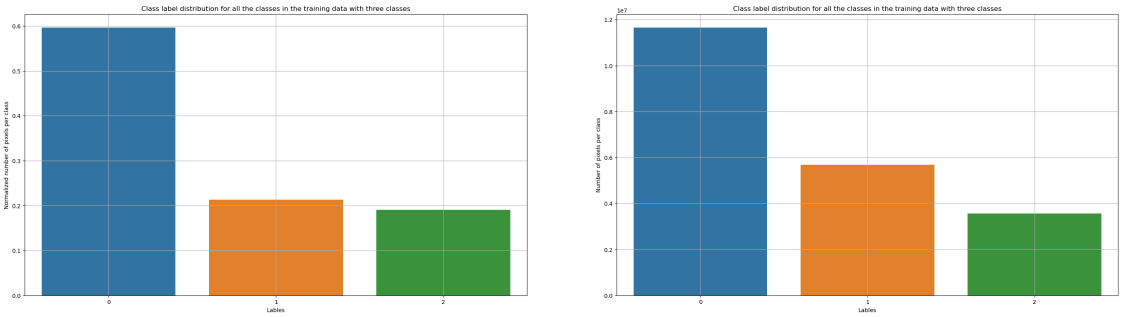
Figure 4.3: Pixel distribution for the scannet data containing all the classes



(a) Per class pixel distribution of the training dataset pixel class label

(b) Per class pixel distribution of the testing dataset pixel class label

Figure 4.4: Pixel distribution for the scannet data for two classes. There are only two classes because all the classes except class 1 are combined with the 0th class. Horizontal axis represent the class and vertical axis represent the number of pixels belonging to that particular class.



(a) Per class pixel distribution of the training dataset pixel class label

(b) Per class pixel distribution of the testing dataset pixel class label

Figure 4.5: Pixel distribution for the scannet data for three classes. There are only three classes because all the classes except class 1 and 2 are combined with the 0th class. Horizontal axis represent the class and vertical axis represent the number of pixels belonging to that particular class.

(a) Ordered set of images

(b) Unordered set of images label

Figure 4.6: Ordered and Unordered set of images. Ordered set of images are the images from the video sequence without shuffling the frames. In unordered set of images the frames are shuffled.

pose plot, the frame's pose is shuffled and presented as the matrix plot. These pictures represent the distance of one frame to the other.

The distance matrix is used to construct the kernel of the Gaussian process. The kernel represents the similarity or correlation between the two data points. Closer data points in the distance space have a high correlation, and further away data points have a low correlation, and the same can be observed in the plotting of the kernel matrix as a heatmap in Fig 4.7 and 4.8. The corresponding ordered and unordered frames images are presented in 4.6.

4.3.3 Experiment1.1: U-Net Vanilla, GP and LSTM model considering all the classes

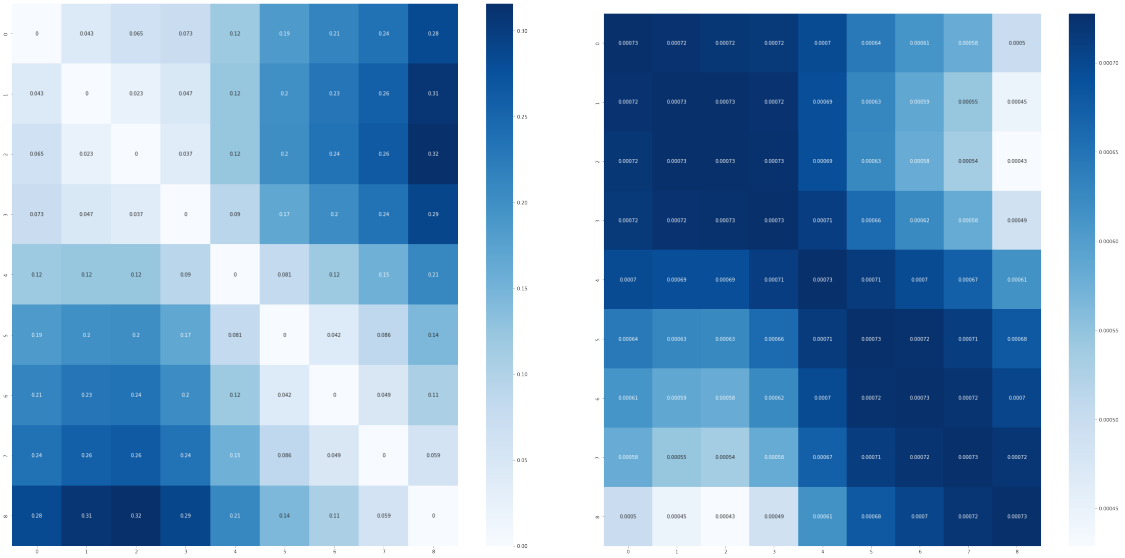
The below sections explain the results of the conducted experiments. The model is trained and tested with the scannet dataset without merging the classes. The model's performance is evaluated with Pixel accuracy, Mean pixel accuracy, meanIoU, IoU, and FwIoU metrics.

Experiment1.1.2: U-Net vanilla model

The pixel accuracy is 50.96% for the validation dataset. This is due to a large number of classes with unequal pixel distribution. However, the IoU for the Wall, Floor, and Mattress is high at 0.56, 0.608, and 0.382 due to the high pixel distribution for these classes. Frequency-weighted IoU considers the unequal pixel distribution, and the values stand at 0.3531. The predicted and the ground truth pixel distribution is presented in Fig 4.10. The bar plot shows that the predicted "Wall" class distribution is higher than the ground truth. The Wall and Floor class distribution is similar to the baseline and predicted class labels.

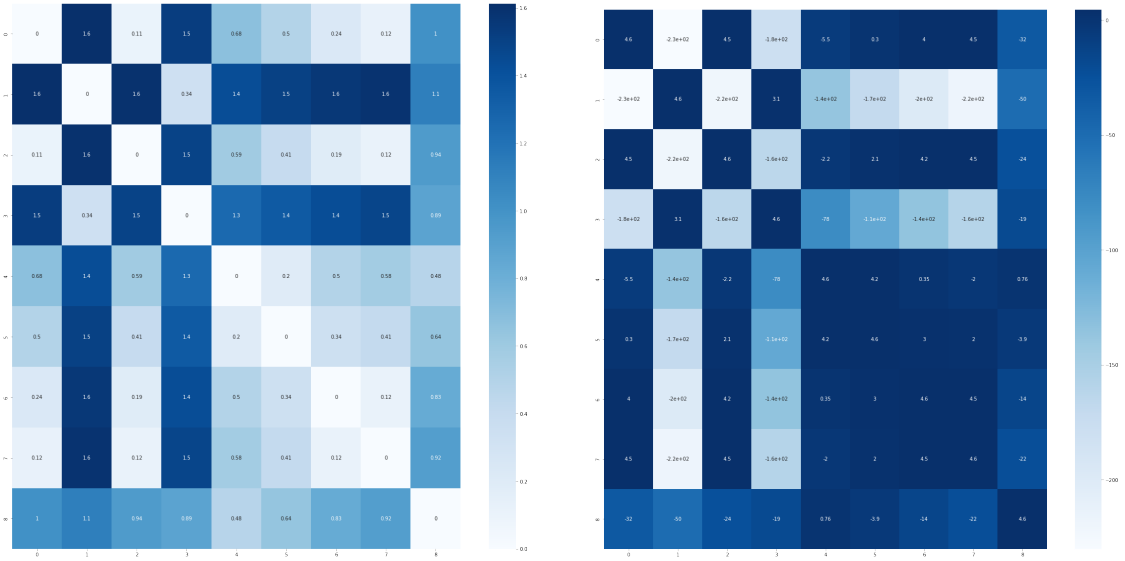
Experiment1.1.2: U-Net GP model

In the GP model experiment, the information from the previous frame is passed onto the current frame segmentation computation. The accuracy is slightly better than the vanilla model, with an increase of 1% with the temporal fusion GP prior to integration. The latent space encoding is subjected to the Gaussian



(a) Distance matrix depicted as heatmap (b) Kernel matrix depicted as heatmap label

Figure 4.7: Distance matrix and Kernel matrix for ordered set of images. Distance matrix represent the distance between the consecutive eight frames poses with respect to each other. Kernel matrix represent the covariance between the consecutive eight frame poses.



(a) Distance matrix depicted as heatmap (b) Kernel matrix depicted as heatmap label

Figure 4.8: Distance matrix and Kernel matrix for unordered set of images. Distance matrix represent the distance between the shuffled eight frames poses with respect to each other. Kernel matrix represent the covariance between the shuffled eight frame poses.

Metric	Value
Pixel Accuracy	0.5096
Pixel Mean accuracy	0.1907
meanIOU	0.1102
FwIoU	0.3531

Table 4.1: Vanilla model performance with respect to all the metrics. Higher values means top performing model.

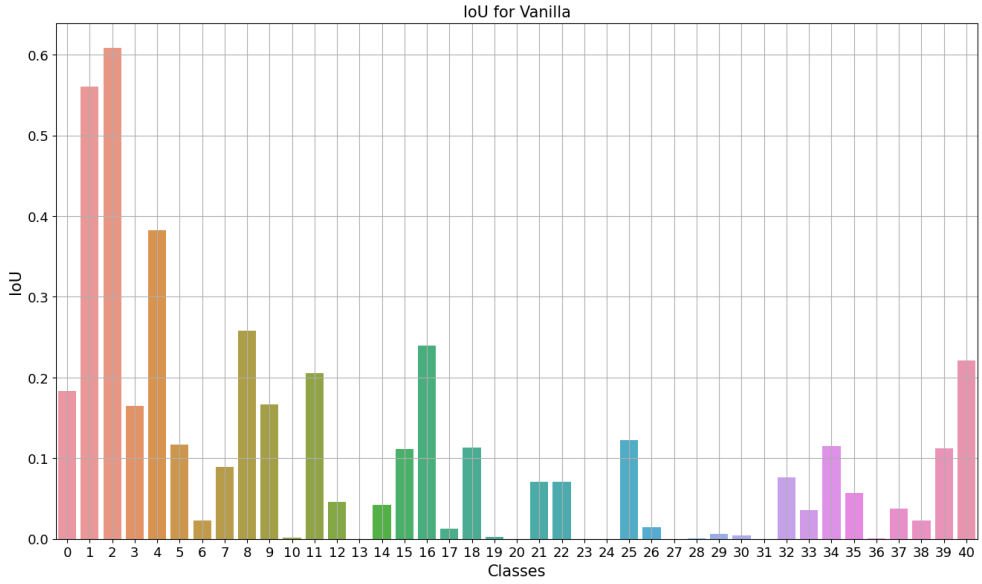
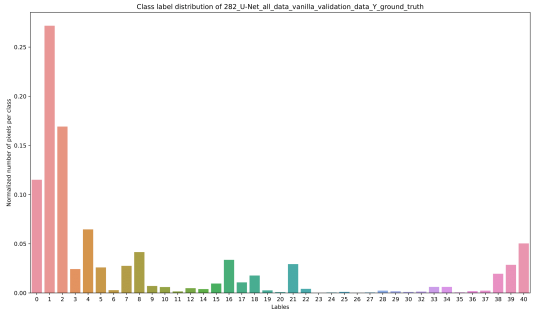
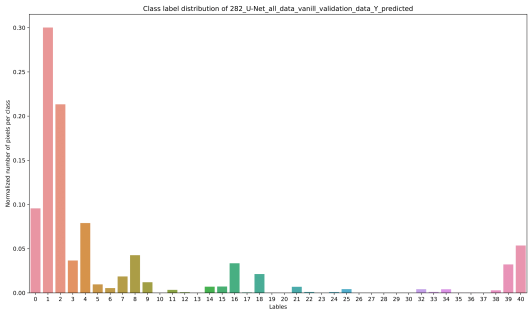


Figure 4.9: IoU for vanilla model considering all classes in dataset



(a) Per class pixel distribution of the ground truth pixel class label



(b) Per class pixel distribution of the predicted pixel class label

Figure 4.10: Pixel distribution for the ground truth and predicted scannet data for vanilla unet model

Metric	Value
Pixel Accuracy	0.5184
Pixel Mean accuracy	0.1679
meanIOU	0.1161
FwIoU	0.3497

Table 4.2: GP model performance with respect to all the metrics. Higher values means top performing model.

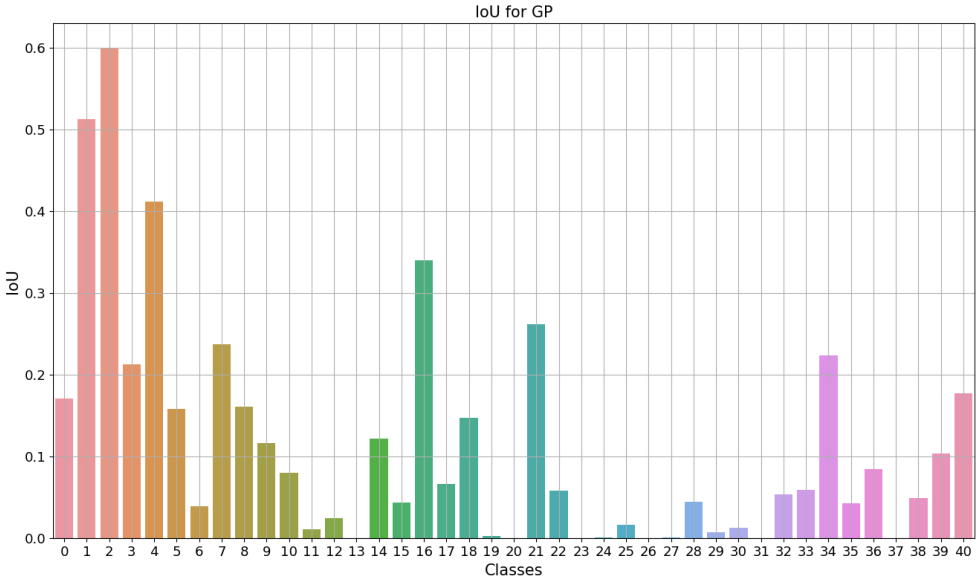
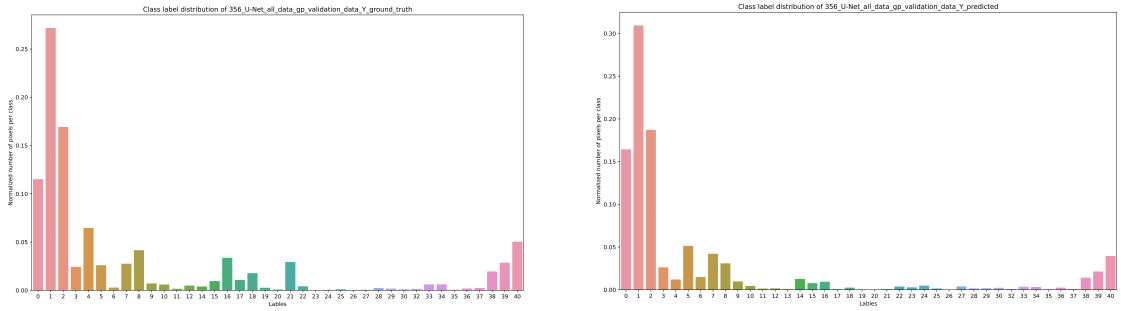


Figure 4.11: IoU for GP model considering all classes in dataset

process and then passed onto the decoder for segmentation computation. The mean pixel accuracy is similar to the vanilla model, with slight improvement. A large number of classes is causing the drop in performance. As expected ”Wall,” ”Floor,” and ”Furniture” classes have high IoU values due to the presence of high pixel distribution.



(a) Per class pixel distribution of the ground truth pixel class label for gp model

(b) Per class pixel distribution of the predicted pixel class for gp model label

Figure 4.12: Per class pixel distribution of the predicted pixel class label for gp model

Metric	Value
Pixel Accuracy	**0.5426**
Pixel Mean accuracy	**0.3509**
meanIOU	**0.1411**
FwIoU	**0.7643**

Table 4.3: LSTM model performance with respect to all the metrics. Higher values means top performing model.

Experiment1.1.2: U-Net LSTM model

The scene geometry from the past frame is passed onto a current frame efficiently with the lstm model. The ConvLSTM cell at the latent space compress and store the past information in its states. The change in the viewpoint is captured by the hidden state and passed onto the next hidden state for the computation of the segmentation map. From the result table **??**, it is evident that the accuracy has improved by 6% and 5% in comparison with the vanilla and GP models, respectively. Mean accuracy has increased by quite a significant margin. A large margin also improves the IoU values. IoU for the "Wall" class is highest with 0.61 in comparison with the vanilla and GP approach. However, for the cabinet class, GP performed well. Table, bookshelf, Counter, and desk class GP outperformed the vanilla and LSTM model.

Accuracy, mean accuracy, meanIoU, and FWIoU comparison is plotted as the bar graph in 4.18. The accuracy is highest for the LSTM model compared to the vanilla and GP. However, the mean accuracy is lowest for the GP model concerning other models. The experiment shows that the performance improves by incorporating temporal fusion in the semantic segmentation for the continuous sequence data. The overlapping and the past information is learned, stored, and passed onto the computation of the future frames' semantic segmentation. The temporal dependency between the frames is exploited to efficiently perform the segmentation by taking pose into account in the Gaussian process and compressed information in the hidden state of the ConVlstm cell. The Gaussian process works because if the two frames are very close to each other, represented by the pose information, there is a high probability of overlapping regions

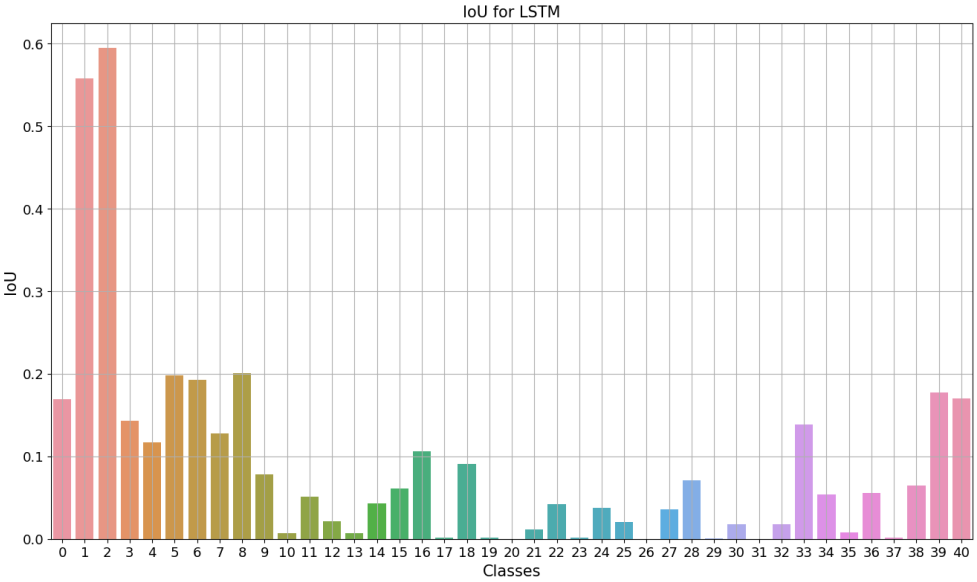
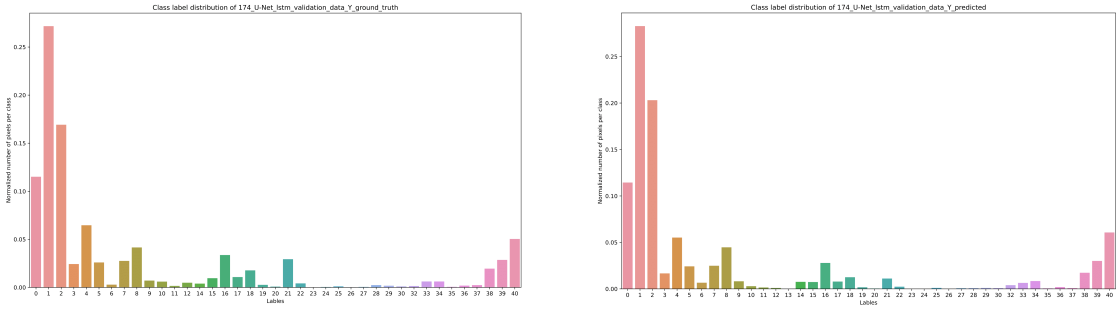


Figure 4.13: IoU for LSTM model considering all classes in dataset



(a) Per class pixel distribution of the ground truth pixel class label for lstm model

(b) Per class pixel distribution of the predicted pixel class for lstm model

Figure 4.14: Per class pixel distribution of the predicted pixel class label for lstm model

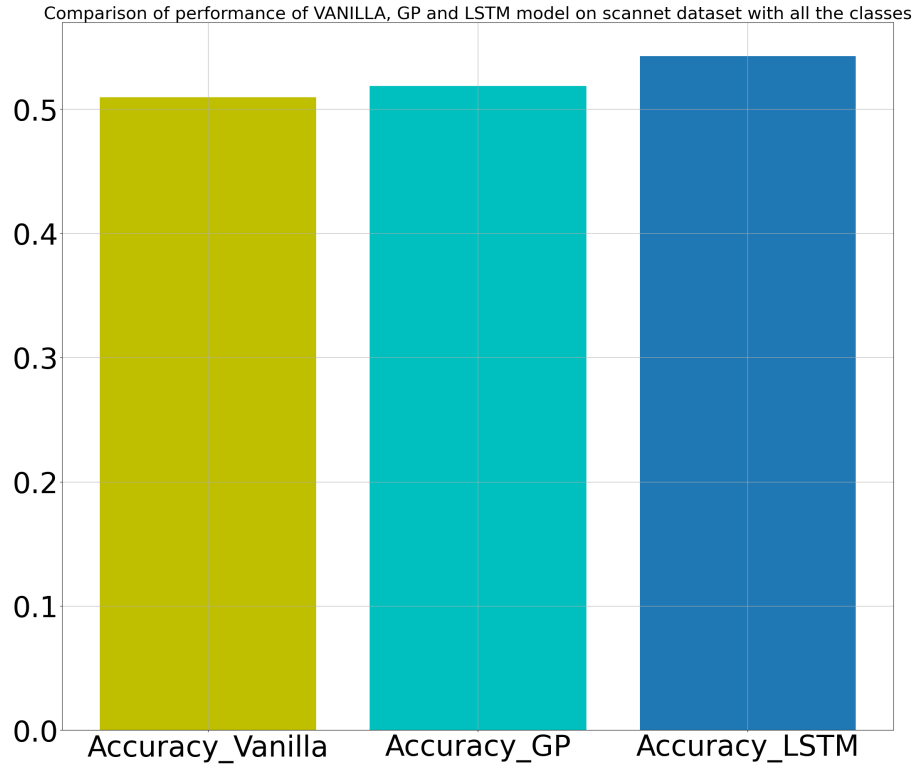


Figure 4.15: Comparison of VANILLA, GP and LSTM model performance based on accuracy metric. Higher the value means top performing model.

in the consecutive frames. The output of the decoder is based on the input to the decoder. The output of the encoder is passed onto the gaussian process; it takes into account the current and previous pose information and outputs a latent encoding based on the distance between the current and the previous frame. Hence, if two frames are close to each other, then the latent output of the Gaussian process is similar, resulting in information transfer from the past learned latent representation to the current latent output resulting in improved performance. The raw RGB ground truth and predicted segmentation map are compared in Fig 4.19. It can be observed from the figure that the Vanilla and LSTM model performed reasonably well in comparison with the GP model. However, overall metric-wise, LSTM outperformed vanilla and GP.

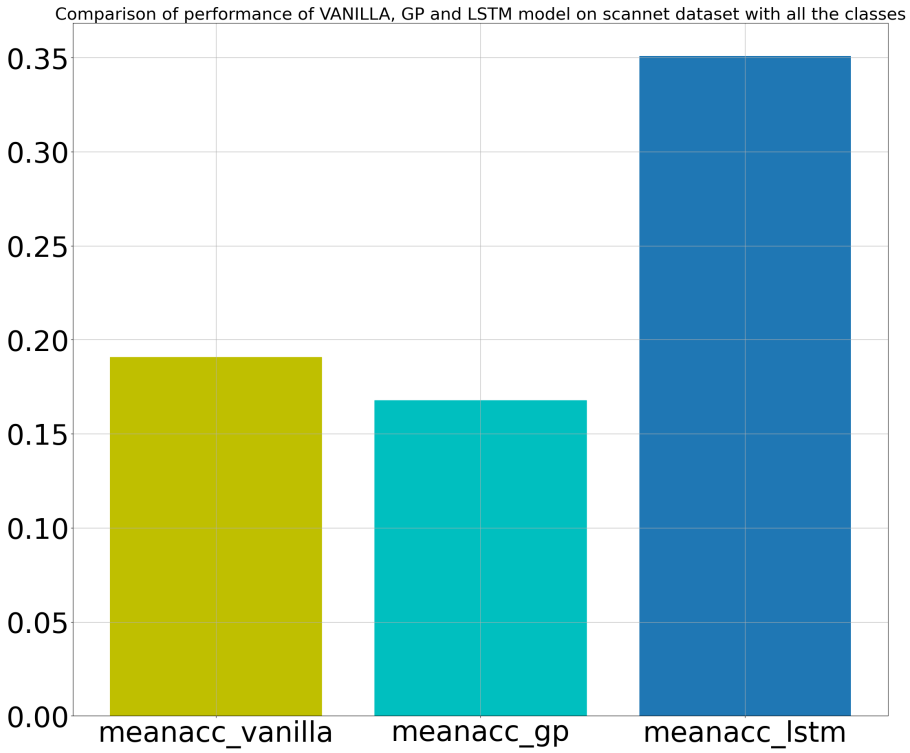


Figure 4.16: Comparison of VANILLA, GP and LSTM model performance based on mean accuracy metric. Higher the value means top performing model.

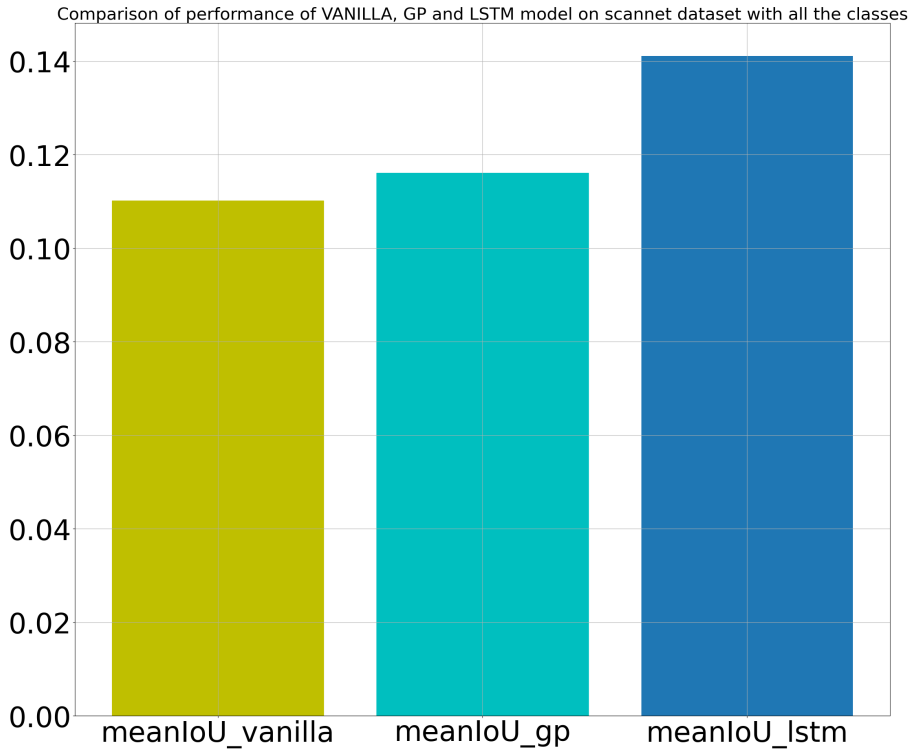


Figure 4.17: Comparison of VANILLA, GP and LSTM model performance based on meanIoU metric. Higher the value means top performing model.

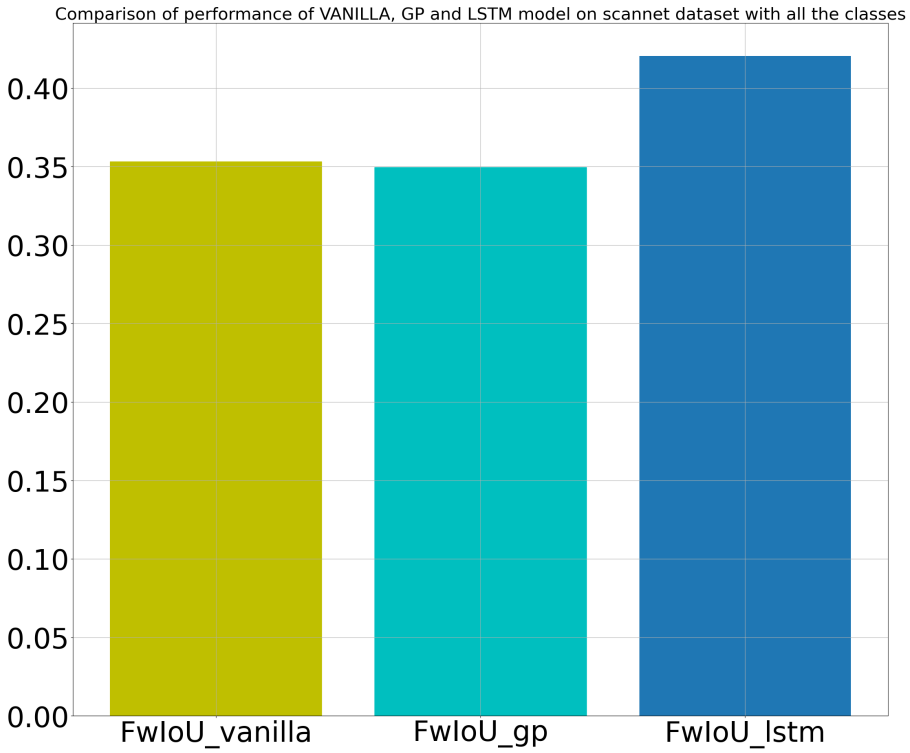


Figure 4.18: Comparison of VANILLA, GP and LSTM model performance based on FwIoU metric. Higher the value means top performing model.

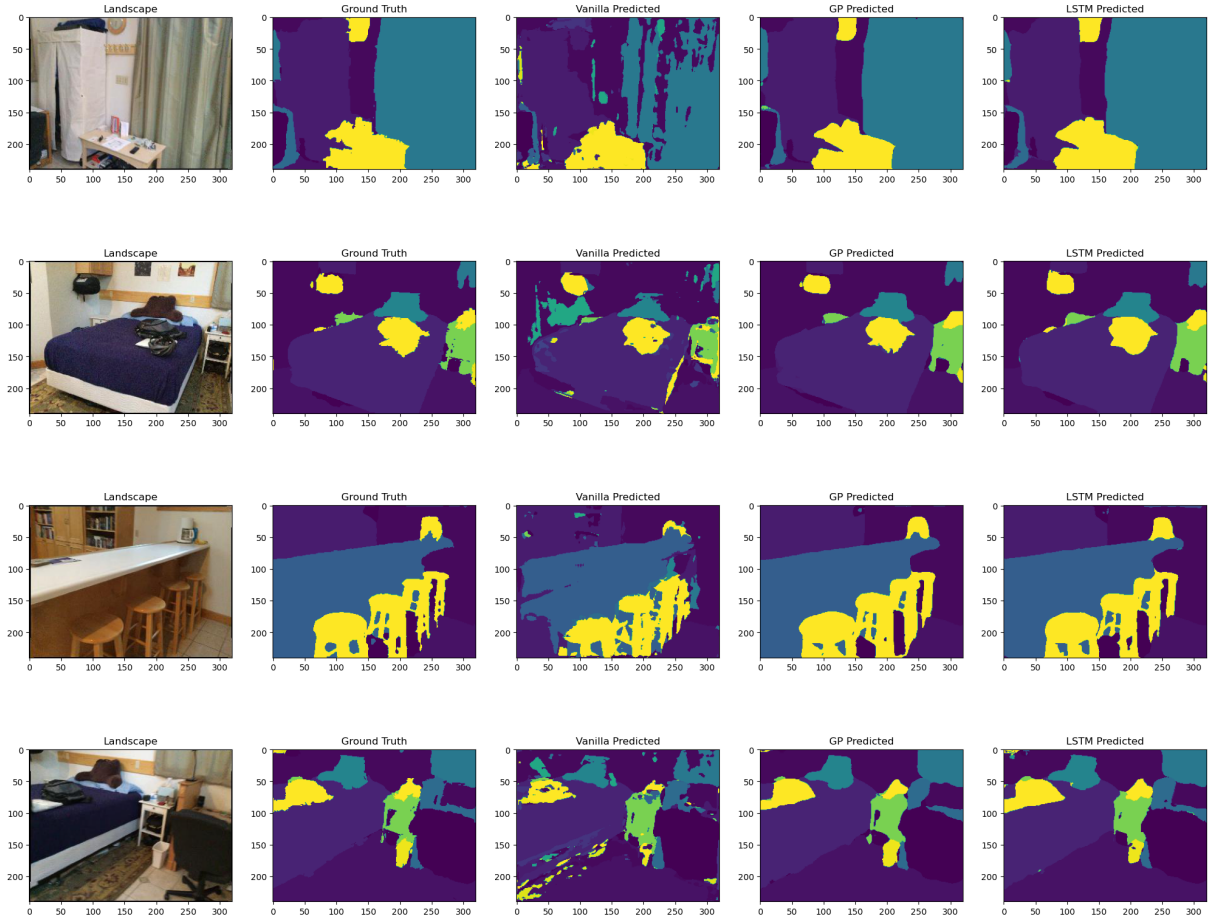


Figure 4.19: Plotting of raw input image, ground truth and predicted output for vanilla, gp and lstm model

4.3.4 Experiment1.2: U-Net Vanilla, Temporally fused GP model and LSTM model considering two class scannet dataset

In the second experiment on the scannet dataset, all the classes are combined except the wall class resulting in the wall and other classes. Table 4.4 compares the results of vanilla, GP, and LSTM model performance. In this case, the pixel accuracy is similar in the vanilla and the lstm model; however, the accuracy dropped for the GP model. A similar observation can be made with the mean accuracy. However, the meanIoU value is high for the lstm model at 0.28, and vanilla and GP produced a result of 0.14 and 0.23. The frequency-weighted intersection over union is high for the lstm model, followed by the vanilla and GP model. A comparison of model output is presented in Fig 4.20. In this case, the wall and other categories in the test image are segmented reasonably well by the LSTM model. The metric bar comparison is shown in Fig 4.25. It is evident from the plot that the mean accuracy is good for the vanilla and GP and lstm performance is below the other models. However, the meanIoU and FWIoU are high for the lstm model.

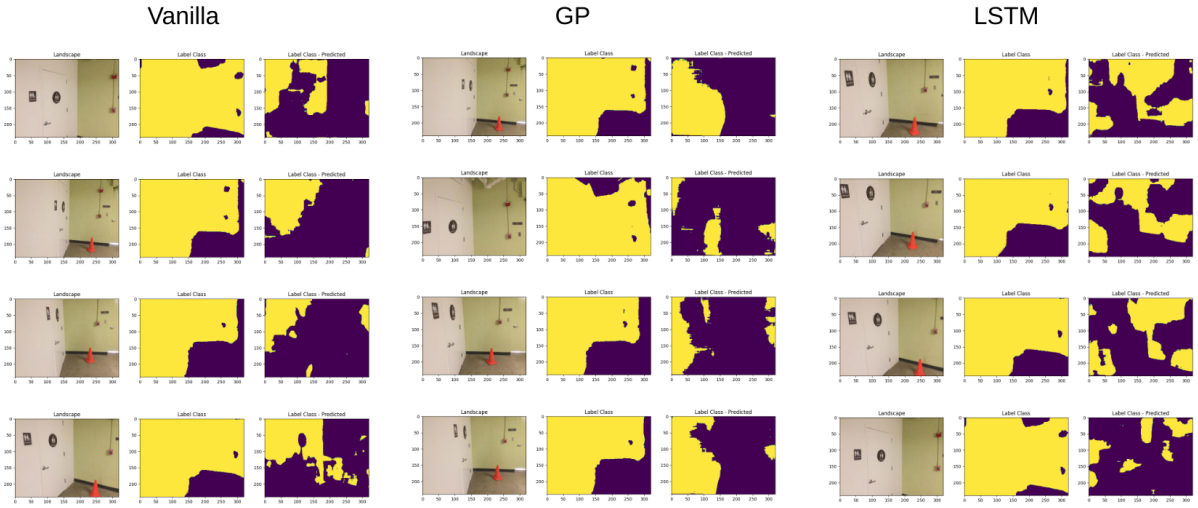


Figure 4.20: Plotting of raw input image, ground truth and predicted output for vanilla, gp and lstm model for two class scannet dataset

Metric	Vanilla	GP	LSTM
Pixel Accuracy	0.4342	0.3764	**0.8549**
Pixel Mean accuracy	0.6232	0.5625	**0.7837**
meanIOU	0.145	0.2318	**0.6593**
FwIoU	0.2793	0.2284	**0.7643**

Table 4.4: Performance of Vanilla model with respect to different metric and two classes. Higher the value means top performing model.

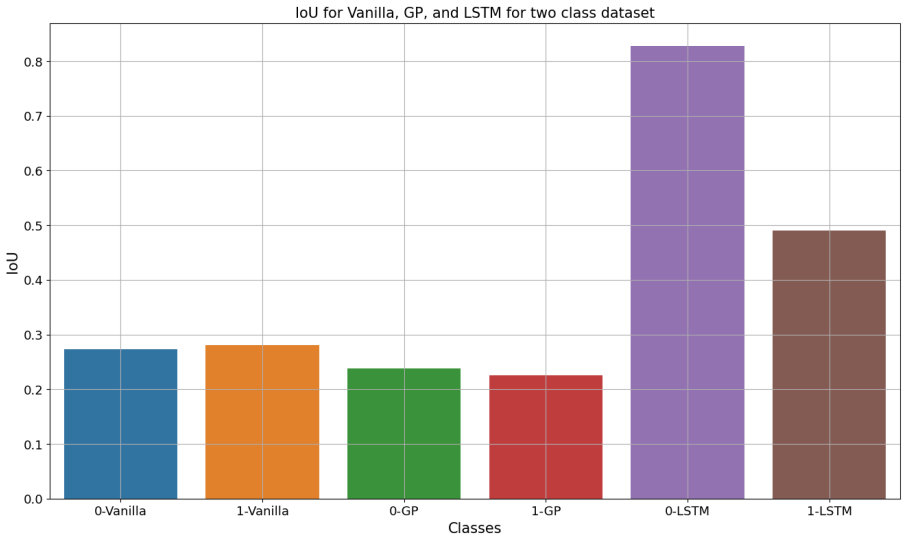


Figure 4.21: IoU for vanilla, GP and LSTM comparison side by side for two classes in scannet dataset

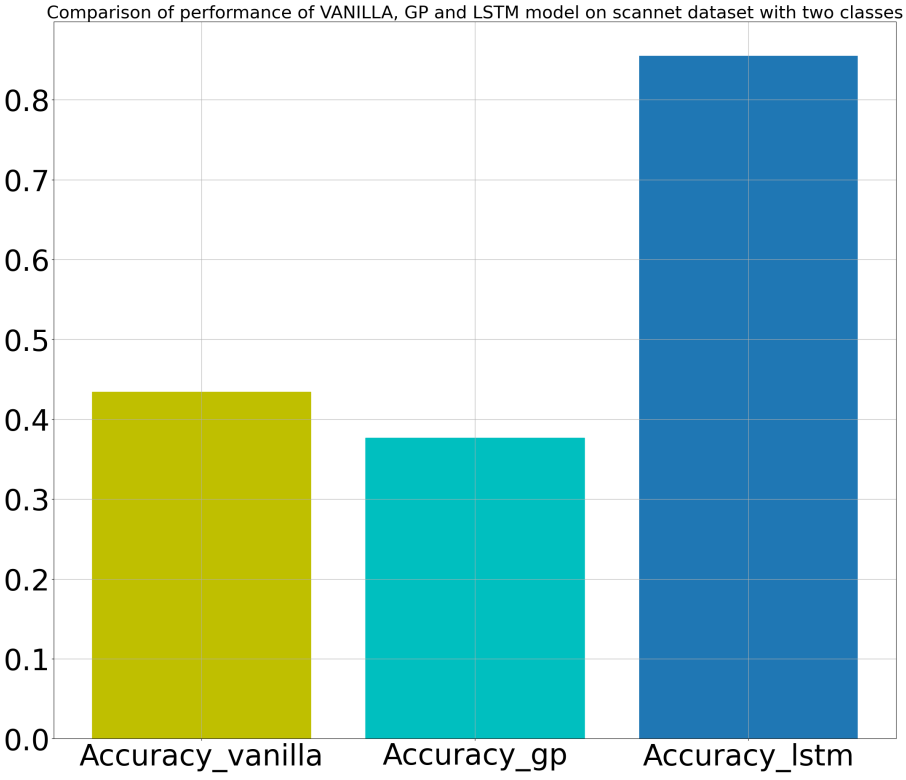


Figure 4.22: Comparison of VANILLA, GP and LSTM model performance based on accuracy metric for scannet two classes

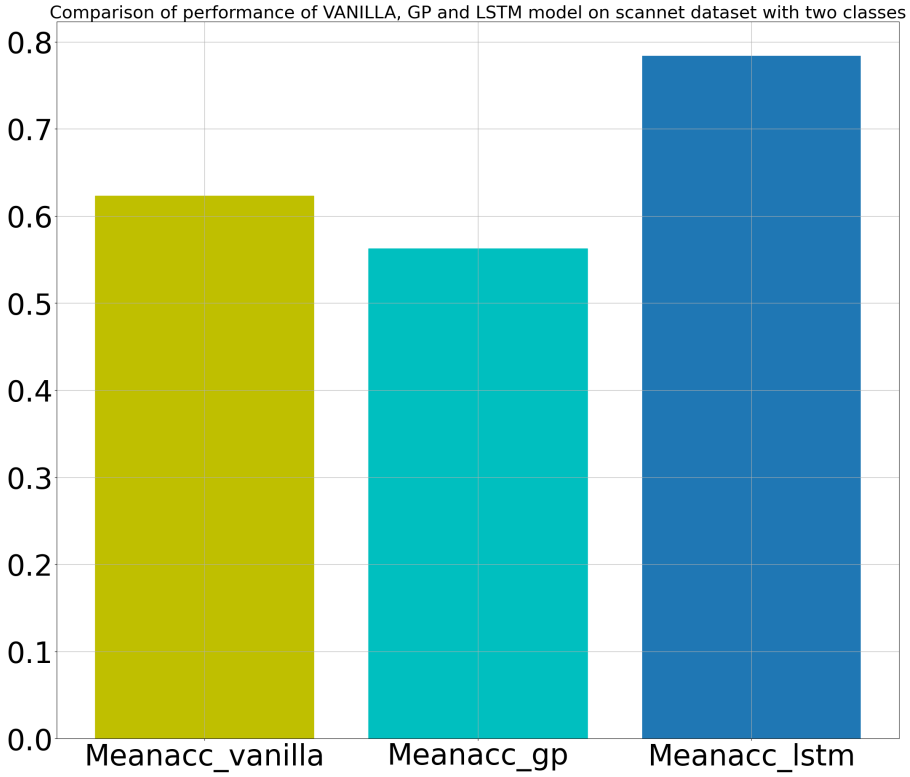


Figure 4.23: Comparison of VANILLA, GP and LSTM model performance based mean accuracy on metric for scannet two classes

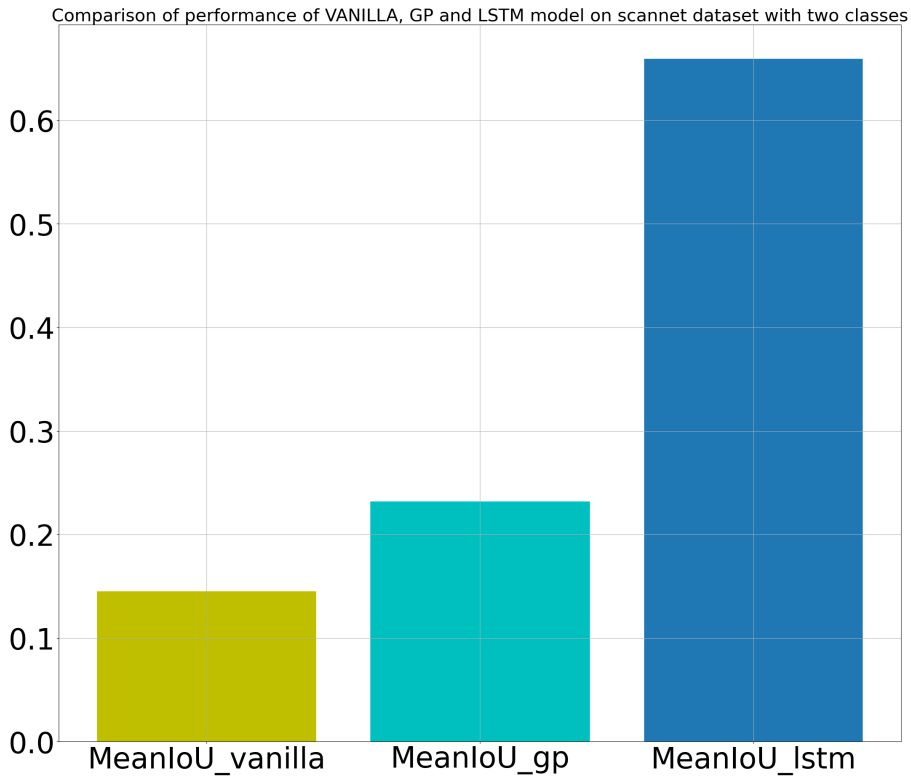


Figure 4.24: Comparison of VANILLA, GP and LSTM model performance based on meanIoU metric for scannet two classes

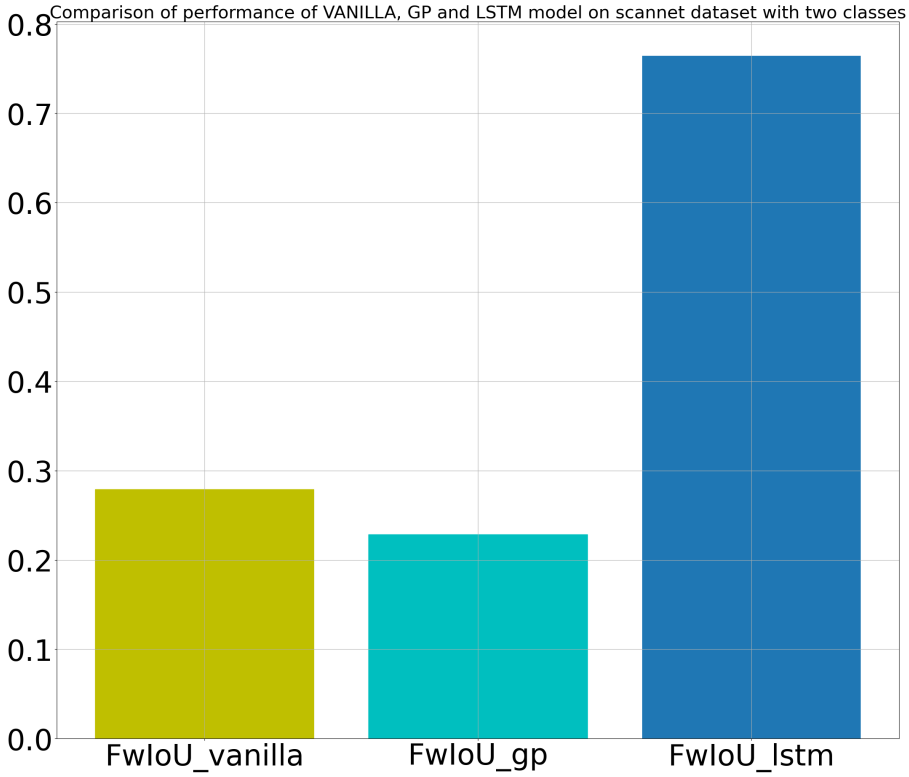


Figure 4.25: Comparison of VANILLA, GP and LSTM model performance based on FwIoU metric for scannet two classes

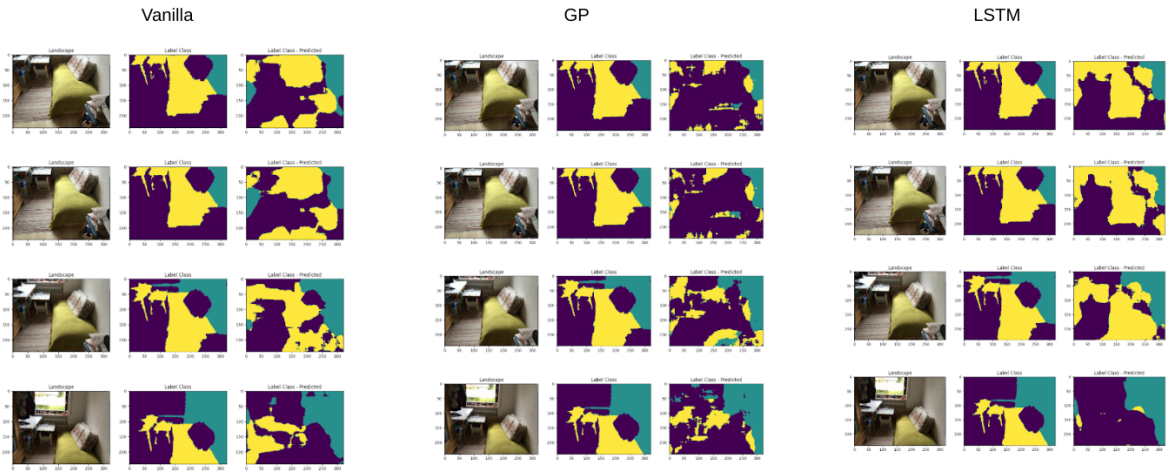


Figure 4.26: Plotting of raw input image, ground truth and predicted output for vanilla, gp and lstm model for three class scannet dataset

4.3.5 Experiment1.3: U-Net Vanilla, temporally fused GP and LSTM model considering three class scannet dataset

This experiment considers wall, furniture, and other combined classes due to high pixel distribution, resulting in three unique classes. There is a massive jump in the pixel accuracy for the lstm and GP models compared to the vanilla model. The mean pixel accuracy is high at 0.55 compared to the vanilla model performance at 0.31. An increase of 24%. The meanIoU value also stands at 0.40 for the lstm model compared to 0.18 for the vanilla model, a difference of 22%. The FwIoU value increased by 25% for the lstm model. The effect of temporal fusion can be clearly seen with these metrics. Temporal fusion improved the performance by a considerable margin. The GP model also performed well in comparison with vanilla and lstm. In this experiment, the pixel classes are well-balanced compared to the two-class approach, where the difference is high. The results are compared in the table Table 4.5. In Fig 4.26, there is temporal fusion in action. The comparison of model performance with a bar plot is depicted in Fig 4.31

Metric	Vanilla	GP	LSTM
Pixel Accuracy	0.3289	0.5731	**0.6266**
Pixel Mean accuracy	0.3135	0.4779	**0.6920**
meanIOU	0.1820	0.3356	**0.4870**
FwIoU	0.2160	0.4033	**0.6245**

Table 4.5: Performance of Vanilla, GP and LSTM model with respect to different metric and three class scannet dataset. Higher the value means top performing model.

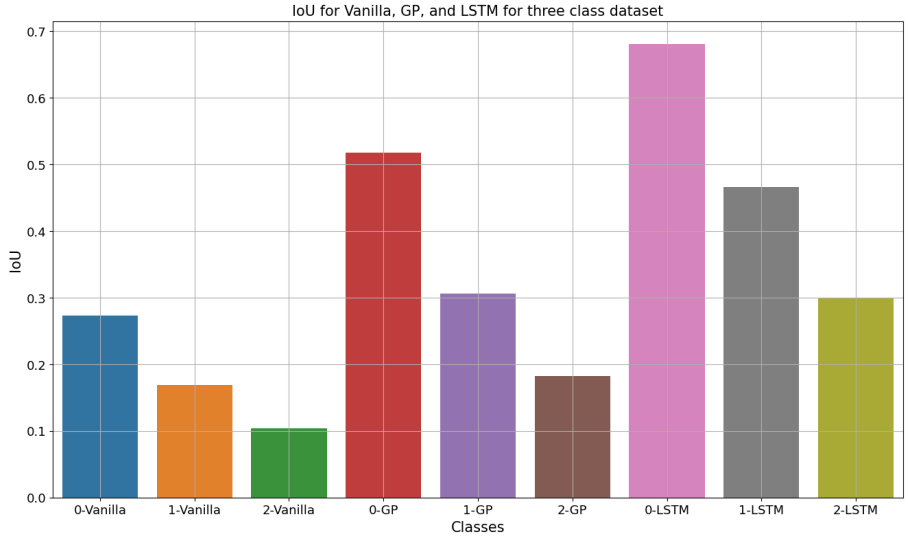


Figure 4.27: IoU for vanilla, GP and LSTM comparison side by side for three classes in scannet dataset

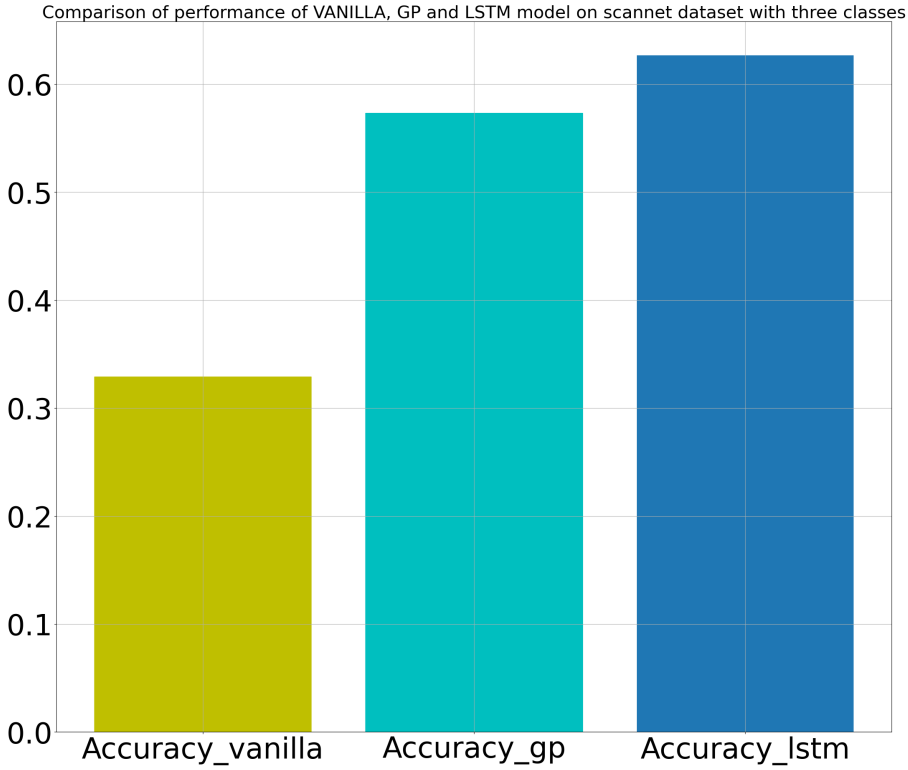


Figure 4.28: Comparison of VANILLA, GP and LSTM model performance based on accuracy metric for scannet three class scannet dataset. Higher the value means top performing model.

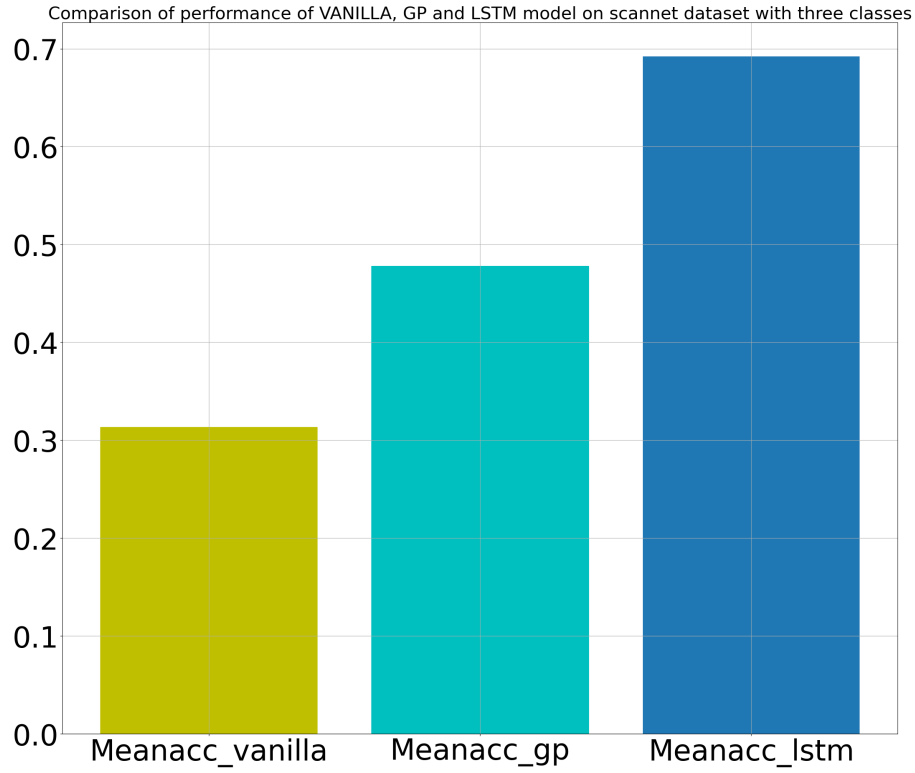


Figure 4.29: Comparison of VANILLA, GP and LSTM model performance based on mean accuracy metric for three class scannet dataset. Higher the value means top performing model.

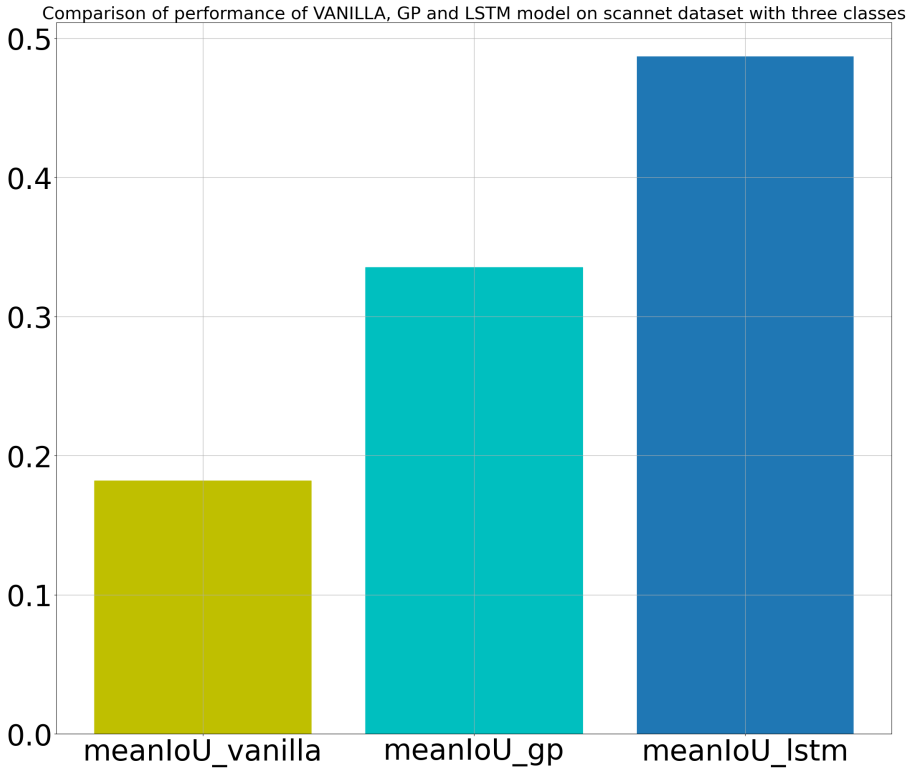


Figure 4.30: Comparison of VANILLA, GP and LSTM model performance based on meanIoU metric for three class scannet dataset. Higher the value means top performing model.

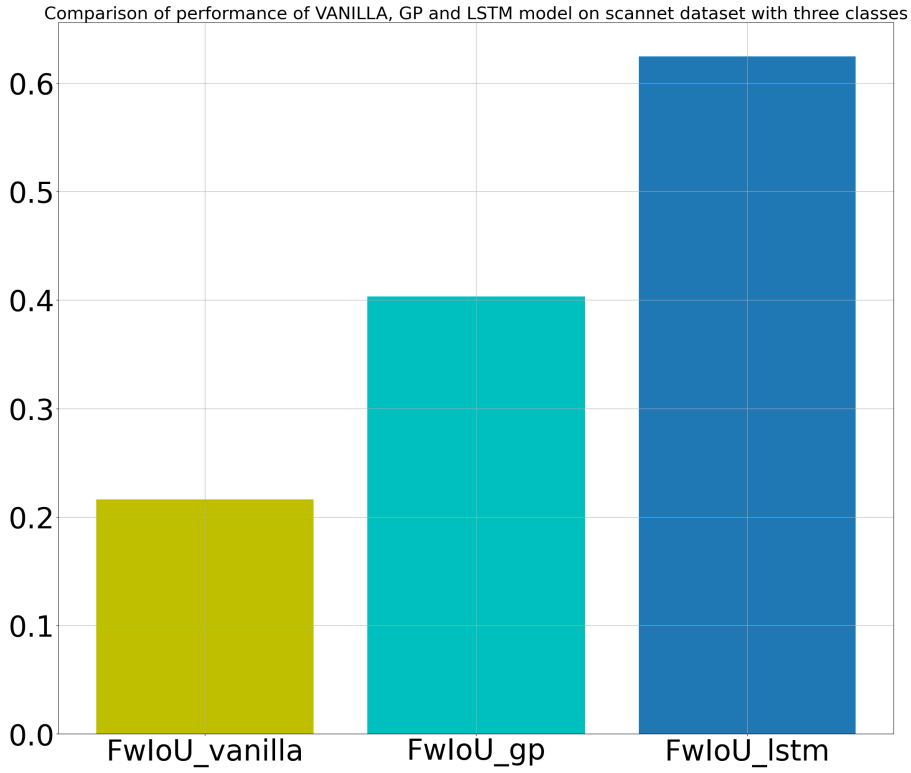


Figure 4.31: Comparison of VANILLA, GP and LSTM model performance based FwIoU on metric for three class scannet dataset. Higher the value means top performing model.

Metric	Vanilla	GP	LSTM
Pixel Accuracy	0.7423	0.7606	**0.8684**
Pixel Mean accuracy	0.6216	0.6314	**0.7449**
meanIOU	0.4854	0.5122	**0.6022**
FwIoU	0.6451	0.6585	**0.7857**

Table 4.6: Performance of Vanilla, GP and LSTM model with respect to different metric. Higher the value means top performing model.

4.4 Experiment 2.1: Experiment with vkitti dataset with fog and morning as the testing data

Similar to the experiments conducted on the scannet dataset, experiments were conducted on the vkitti dataset. Vkitti is a dataset containing a virtual car moving in a cityscape and recorded with a different configuration. There are a totally of 15 classes in the entire dataset. The results were tabulated in Table 4.6. There are a total of ten categories in the entire dataset. Namely 15-deg-left, fog, overcast, morning, 30-deg-right, 15-deg-right, 30-deg-left, clone, rain, sunset. Under each category, there is a total of 14 classes, namely Terrain, Sky, Tree, Vegetation, Building, Road, GuardRail, TrafficSign, TrafficLight, Pole, Misc, Truck, Car, and Van. In this experiment, fog and morning are taken as the testing data and trained on the remaining eight categories of data. The result of the experiments was tabulated in Table 4.6. The Vanilla, GP, and LSTM model performance are compared in the table. Pixel accuracy improved significantly with the LSTM model compared to the Vanilla and Gaussian process. A jump of 11%. Similar characteristics can be observed concerning the other metrics. All the categories are similar; only the environment changes. A variation in an environment affected the model performance. In this experiment, LSTM performed well compared to the Vanilla and GP models. The accuracy of the model is high, 0.86. Temporal fusion can be observed with mean pixel accuracy, FwIoU, and meanIoU metric with improvement in result. The model is evaluated with the testing data, and the results are calculated for every batch and plotted as the box plot in Fig 4.36. For all of the metric, the LSTM produced a better result in comparison to other models. IoU for the building, Misc, and Truck was low with respect to the other classes. The performance of vanilla and the GP model are comparable to each other. Four test images were taken and run through vanilla, GP, and LSTM model, and the ground truth and the predictions were compared side by side. Vanilla and GP models ideally segment the Truck; however, the LSTM model does not correctly segment the Truck. However, the car, pole, road, and sky categories are perfectly categorized by these models. The side-by-side comparison is presented in Figure 4.32. The experiments were conducted in a batch of 8.

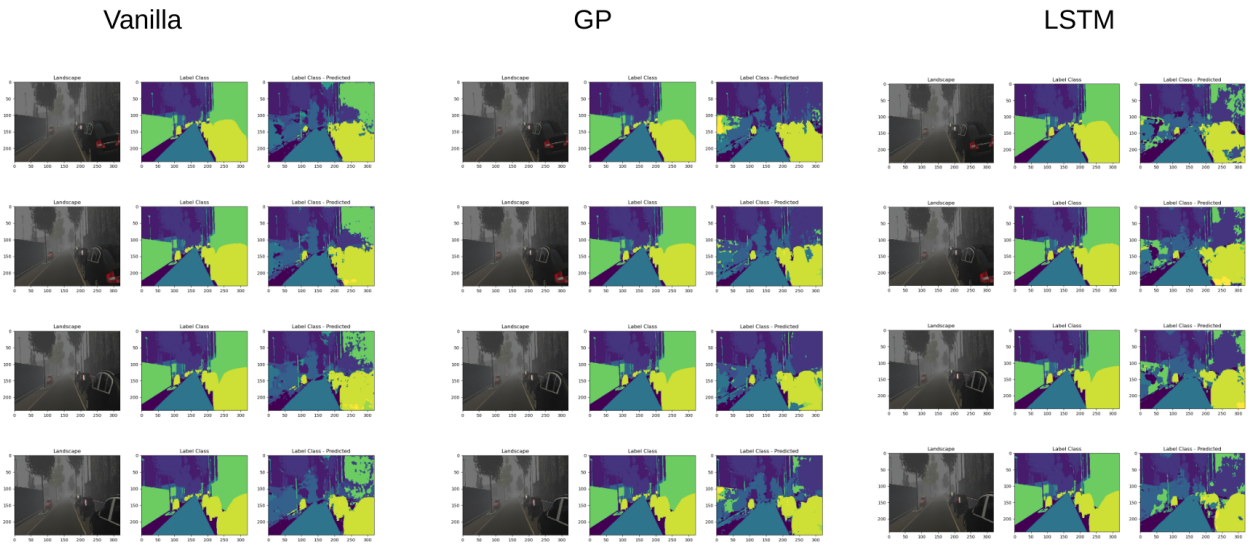


Figure 4.32: Plotting of raw input image, ground truth and predicted output for vanilla, gp and lstm for vkitti dataset with fog and morning as the validation dataset

4.4.1 Experiment 2.1.2: Impact of training data batch size on the evaluation dataset performance

The neural network model can be trained in batch data. The impact of the batch is studied in this experiment. The temporal fusion aims to pass continuous sequence image data and model the temporal dependency resulting in improved performance. In this experiment, a batch of 1, 2, and 8 data are passed onto the temporal fusing neural network model to train and evaluate the performance. In a batch mode of 1, the information from each frame is passed onto the next frame temporally. At a time single image is passed to the network for learning. Two images are passed in the batch mode of 2 at a time, and temporal information from these two images is learned and passed onto the subsequent two frames' prediction. In a batch mode of 8 at a time, a total of 8 image segmentation are learned from the ground truth. The overlapping information present in the set of 8 images is passed onto the following eight sets of continuous sequence data at a time processing eight images. With this experiment, it is found that the performance improves as the number of images in a batch; in other words, as the batch size increases, the performance improves. The same can be observed with the comparison of model performance on a batch size of 1, 2, and 8 in Figure 4.39. However, there is a limitation to the number of images in a batch. In the experiment, a maximum of 8 images per batch was tried; if the batch size increases above eight, the Cuda memory overflow is observed. This is a limitation set by the hardware.

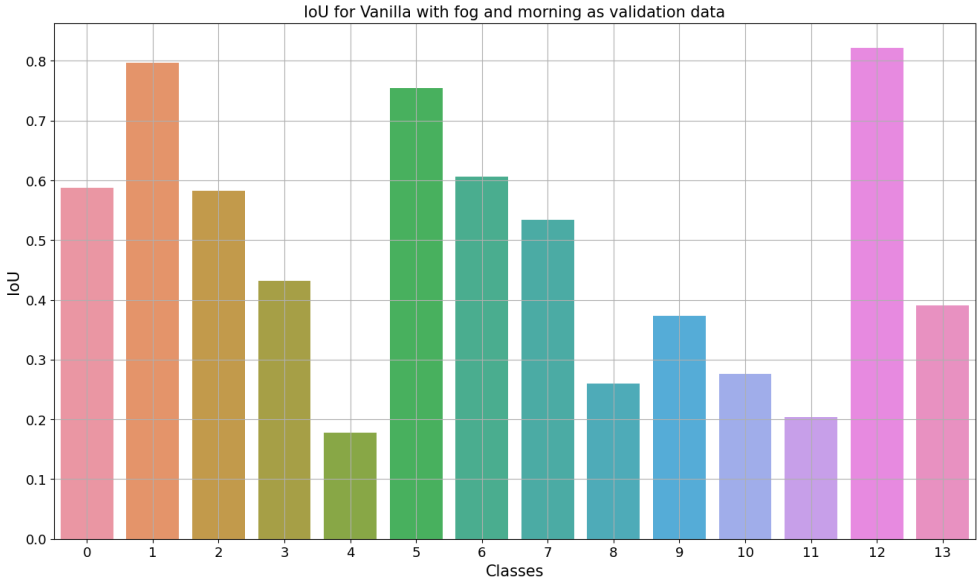


Figure 4.33: IoU for vanilla model tested with vkitti dataset with fog and morning as the validation data

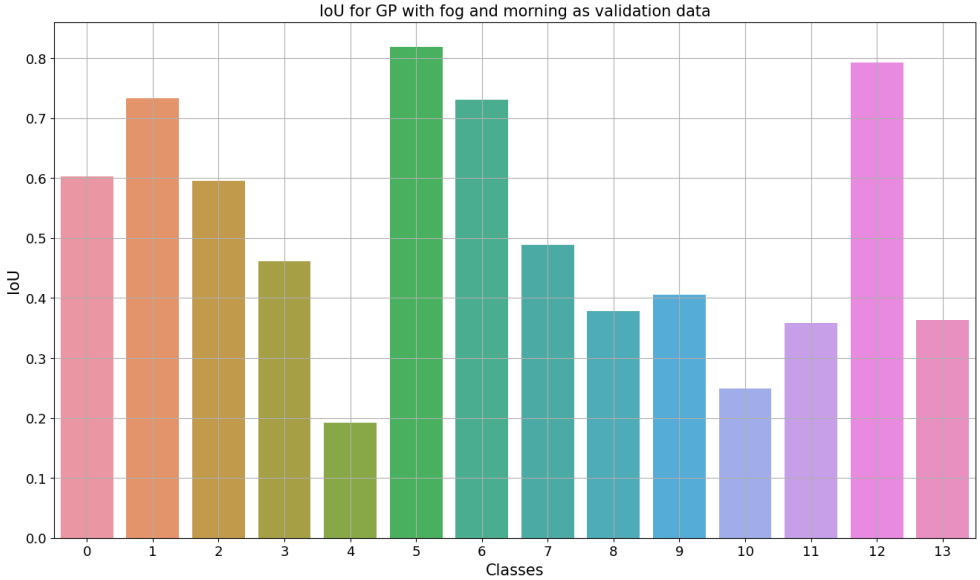


Figure 4.34: IoU for GP model tested with vkitti dataset with fog and morning as the validation data

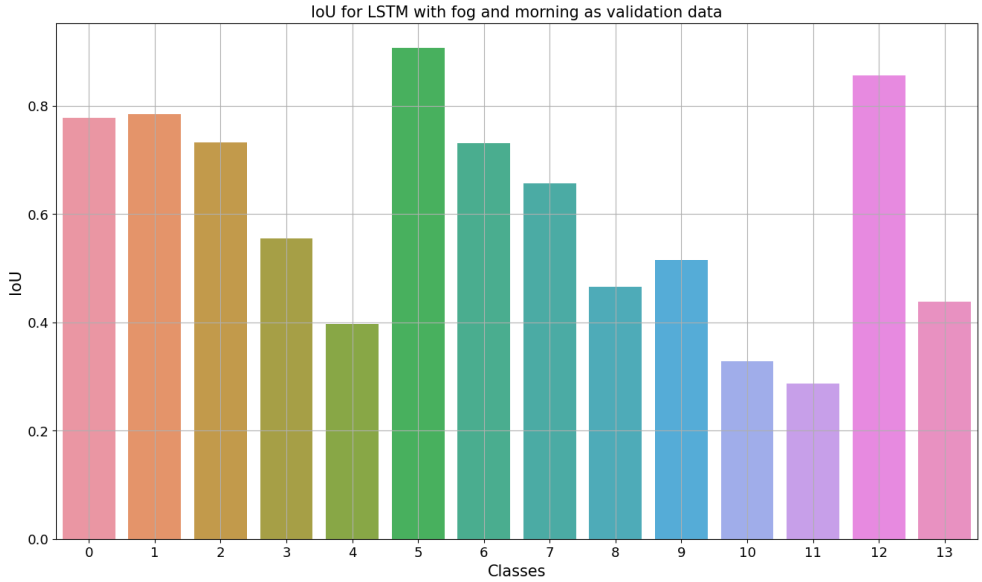


Figure 4.35: IoU for LSTM model tested with vkitti dataset with fog and morning as the validation data

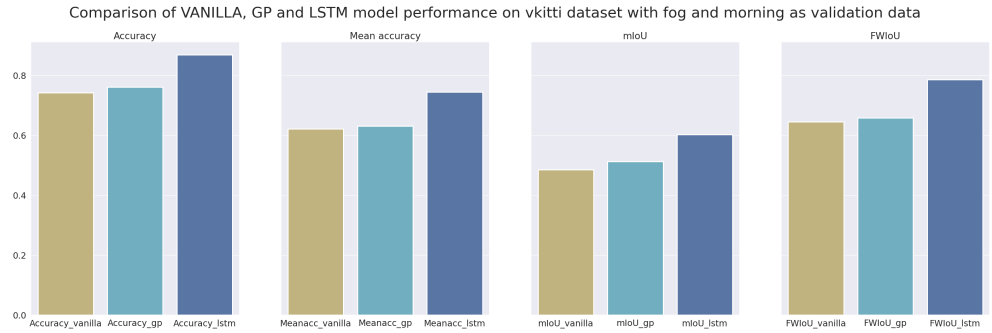


Figure 4.36: Performance comparison of the VANILLA, GP and LSTM model for fog and morning as the validation data. Higher the value means top performing model.

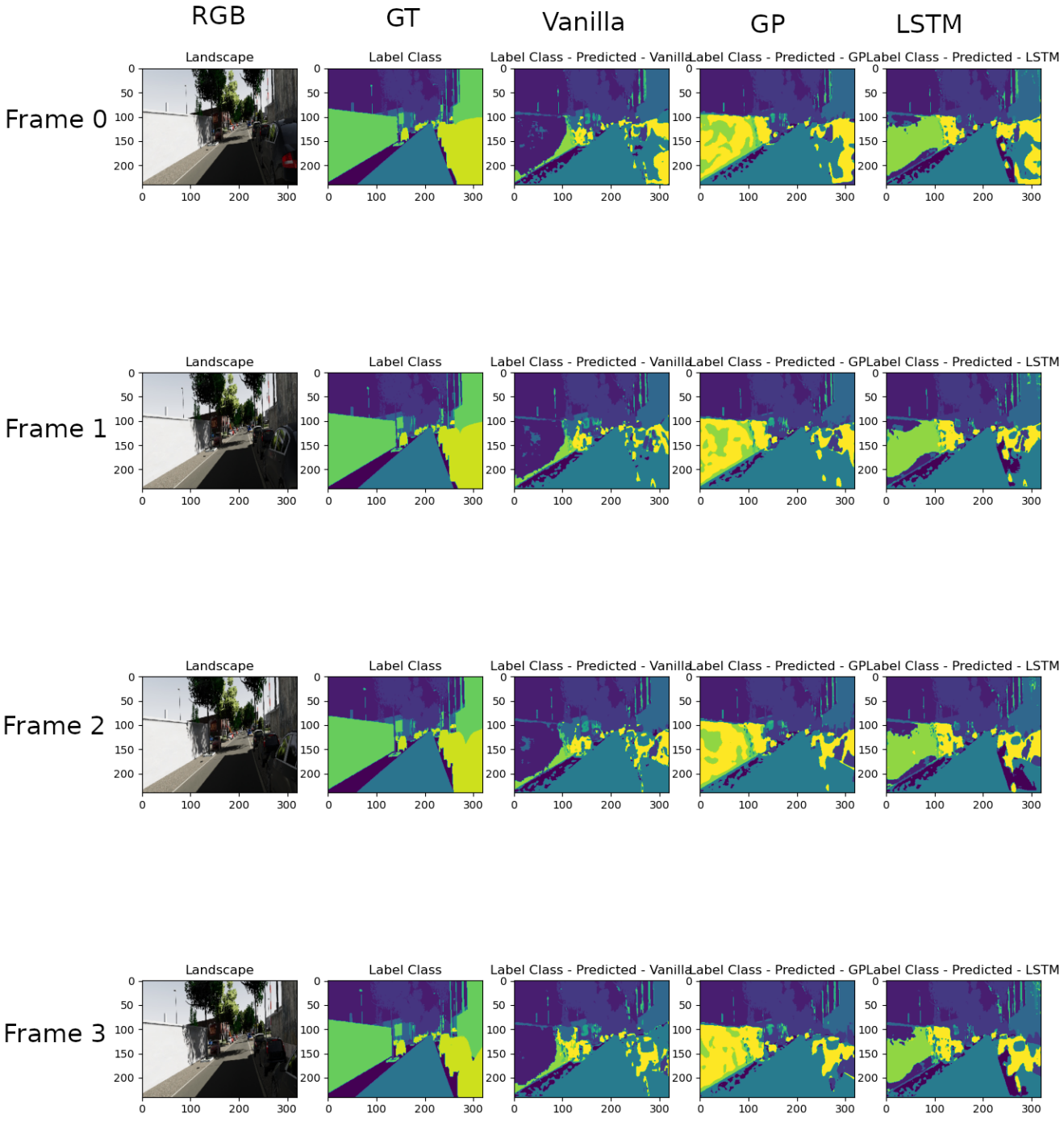


Figure 4.37: Side by side comparison of models predictions for continuous sequence data from frame 0 to 3

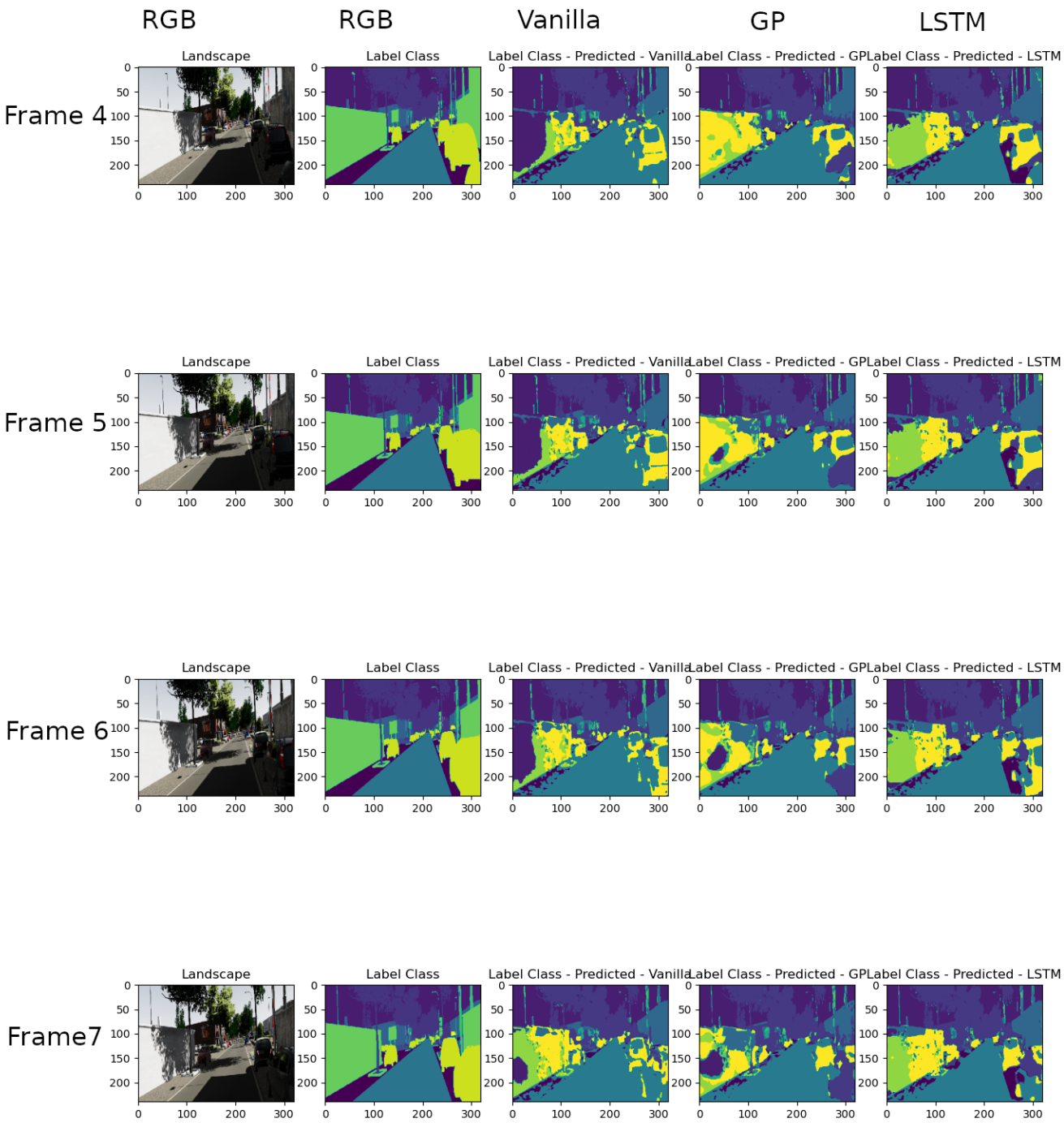


Figure 4.38: Side by side comparison of models predictions for continuous sequence data from frame 4 to 7

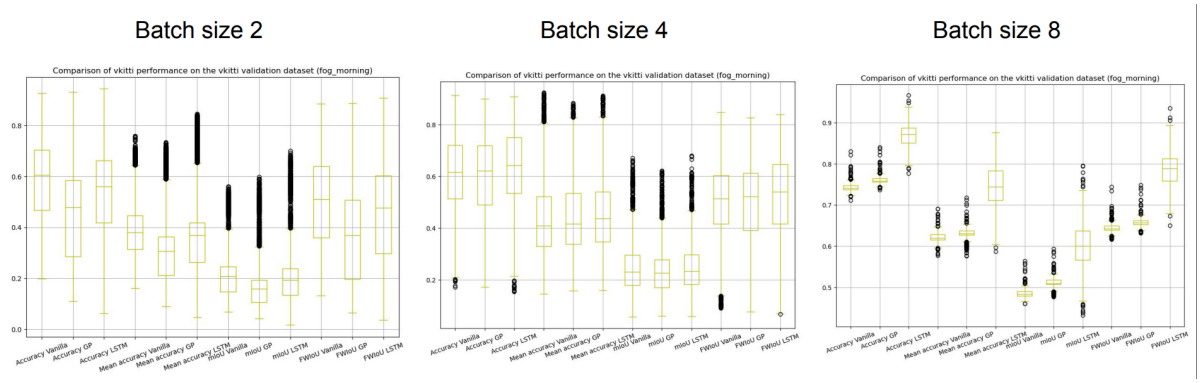


Figure 4.39: Impact of training batch size on the evaluation dataset prediction. The model is trained with batch of 2, 4 and 8. The performance of the model is plotted with different metrics. Higher the value means top performing model.

Metric	Vanilla	GP	LSTM
Pixel Accuracy	0.9419	0.9329	**0.9491**
Pixel Mean accuracy	0.7797	0.7878	**0.8083**
meanIOU	0.6884	0.6977	**0.7206**
FwIoU	0.8959	0.8788	**0.9074**

Table 4.7: Performance of Vanilla model with respect to different metric with clone, sunset, rain, 15 degree right, and 30 degree left as validation data. Higher the value means top performing model.

4.5 Experiment 2.2: Experiment with vkitti dataset with clone, sunset, rain, 15-deg-right, and 30-deg-left as the testing data

In the second type of experiment on vkitti data, five('clone/', 'sunset/', 'rain/', '15-deg-right/', '30-deg-left/') categories out of 10 categories are taken for testing. The model is trained on five (15-deg-left/','fog/', 'overcast/', 'morning/', and '30-deg-right/') categories. The result of the experiments is tabulated in Table 4.7. It is a side-by-side comparison of the model performance based on a different metric. The performance improvement can be observed with the menIoU metric. The meanIoU improved by 1% and 3% for GP and lstm with respect to the vanilla model. Similar characteristics can be observed with the mean pixel accuracy. However, the pixel accuracy remained constant for all three models. The side-by comparison of the raw RGB image, ground truth, and predicted segmentation map is shown in Fig 4.40.

The box plot of the performance of the vanilla, GP, and lstm model is shown in Fig 4.44. The pixel accuracy and FwIoU are high for the lstm model, and the performance is comparable for the GP and vanilla models. Mean pixel accuracy and meanIoU are high for the lstm model. Traffic light, Pole, and Misc class IoU are low in vanilla, GP, and lstm models. The same is shown in Fig **??**. Hence from the result table, we can prove that incorporating the temporal fusion data for the semantic segmentation improves the performance.

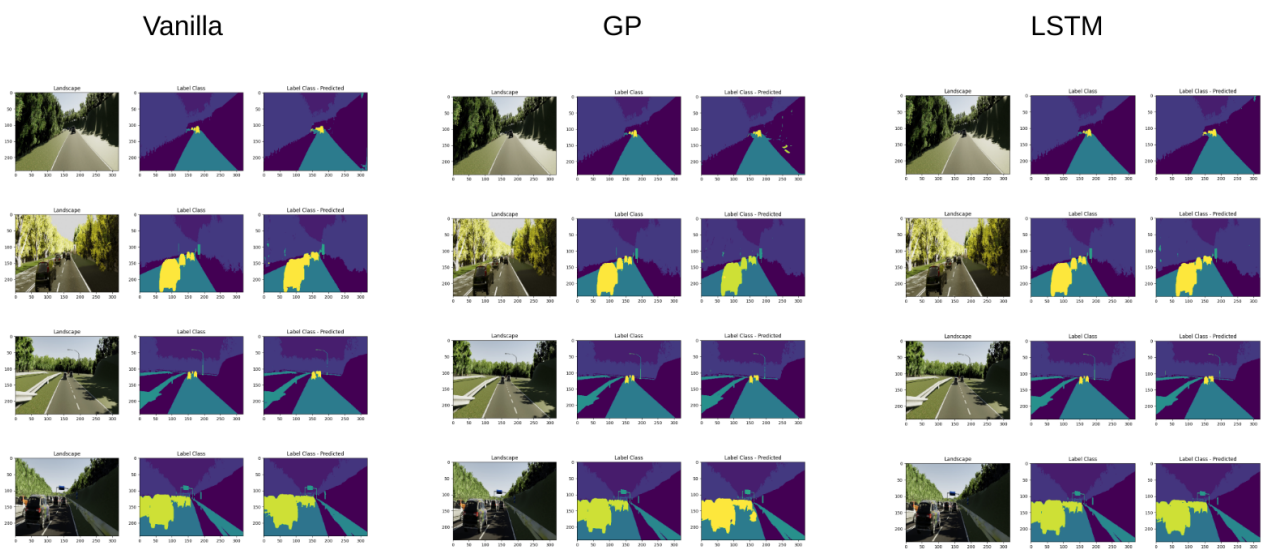


Figure 4.40: Plotting of raw input image, ground truth and predicted output for Vanilla, GP and LSTM for vkitti dataset with five categories of dataset taken for validation.

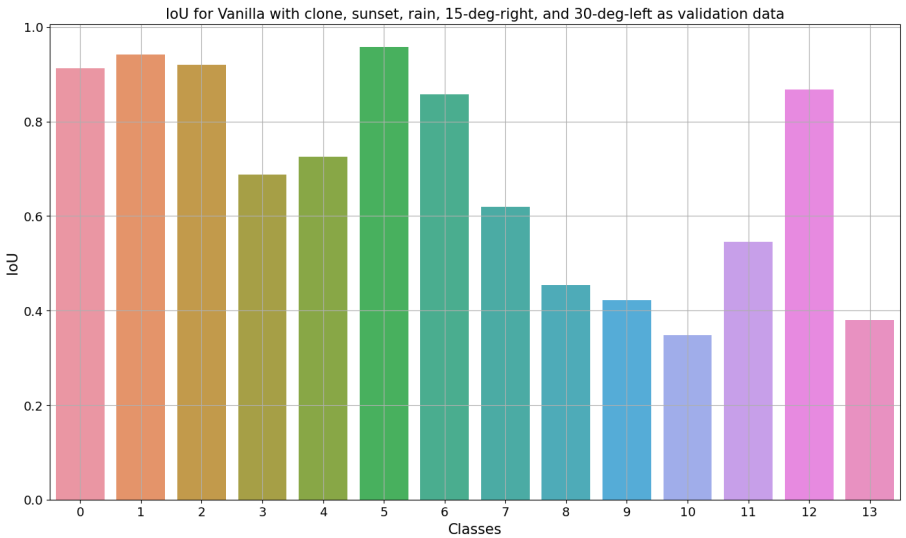


Figure 4.41: IoU for vanilla model with clone, sunset, rain, 15 degree right, and 30 degree left as validation data

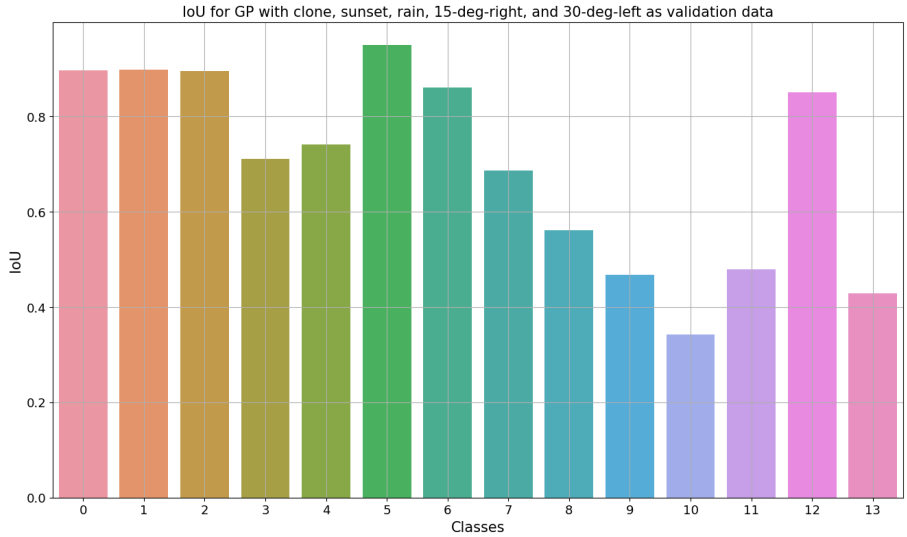


Figure 4.42: IoU for GP model with clone, sunset, rain, 15 degree right, and 30 degree left as validation data

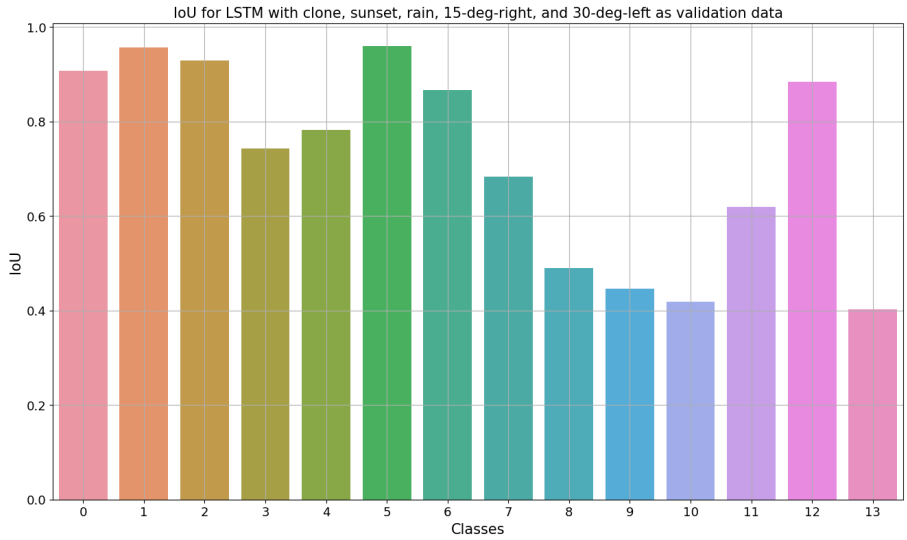


Figure 4.43: IoU for LSTM model with clone, sunset, rain, 15 degree right, and 30 degree left as validation data

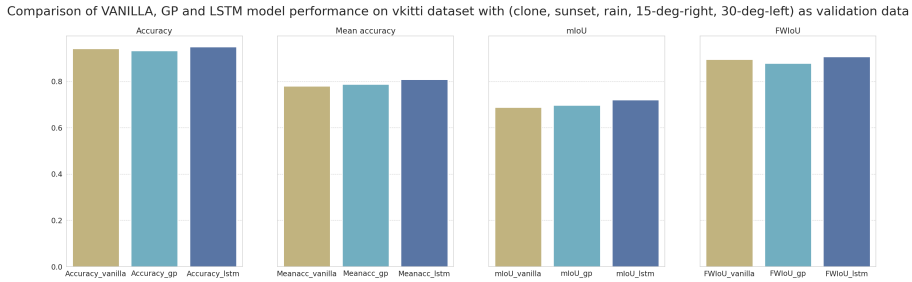


Figure 4.44: Comparison of Vanilla, GP, and LSTM model performance with clone, sunset, rain, 15-deg-right, 30-deg-left as the validation data. Higher the value means top performing model.

Scannet Dataset				
		All classes	Two classes	Three classes
Vanilla	Accuracy	0.5096	0.4342	0.3289
	Mean accuracy	0.1907	0.6232	0.3135
	mIoU	0.1102	0.145	0.182
	FWIoU	0.3531	0.2793	0.216
GP	Accuracy	0.5184	0.3764	0.5731
	Mean accuracy	0.1679	0.5625	0.4779
	mIoU	0.1161	0.2318	0.3356
	FWIoU	0.3497	0.2284	0.4033
LSTM	Accuracy	**0.5426**	**0.8549**	**0.6266**
	Mean accuracy	**0.3509**	**0.7837**	**0.692**
	mIoU	**0.1411**	**0.6593**	**0.487**
	FWIoU	**0.7643**	**0.7643**	**0.6245**

Table 4.8: Comparison of Vanilla, GP and LSTM model predictions based on four metrics and three different conducted experiments for scannet dataset. In all the experiments LSTM model out performed Vanilla and GP.

4.6 RQ3 Conclusion

The Long Short-Term Memory (LSTM) model outperformed the performance of the vanilla/baseline and Gaussian Process (GP) models. The LSTM model can effectively model long temporal data, and the Gaussian Process (GP) uses pose information to model temporal data. The GP method is adopted from the depth estimation approach, where a cost volume is generated by taking consecutive frames. This cost volume captures the overlapping information present in the consecutive frames. However, building a cost volume is optional for semantic segmentation, as it is specifically designed for the depth estimation task. However, a modified GP approach is performed by subjecting the latent space encoding to GP and averaging the previous latent space encoding that was subjected to GP with the current latent space encoding. All the conducted experiments results are tabulated in table 4.8, 4.9, 4.10.

Vkitti Dataset			
		Two categories	Five categories
Vanilla	Accuracy	0.7423	0.9419
	Mean accuracy	0.6216	0.7797
	mIoU	0.4854	0.6884
	FWIoU	0.6451	0.8959
GP	Accuracy	0.7606	0.9329
	Mean accuracy	0.6341	0.7878
	mIoU	0.5122	0.6977
	FWIoU	0.6585	0.8788
LSTM	Accuracy	**0.8684**	**0.9491**
	Mean accuracy	**0.7449**	**0.8083**
	mIoU	**0.6022**	**0.7206**
	FWIoU	**0.7857**	**0.9074**

Table 4.9: Comparison of Vanilla, GP and LSTM model predictions based on four metrics and three different conducted experiments for vkitti dataset. In all the experiments LSTM model out-performed Vanilla and GP.

Impact of batch size on model performance			
		Batch 4	Batch 8
Vanilla	Accuracy	0.5982	**0.7423**
	Mean accuracy	0.5952	**0.6216**
	mIoU	0.6204	**0.4854**
	FWIoU	0.4317	**0.6451**
GP	Accuracy	0.438	**0.7606**
	Mean accuracy	0.4509	**0.6341**
	mIoU	0.2423	**0.5122**
	FWIoU	0.2379	**0.6585**
LSTM	Accuracy	0.2476	**0.8684**
	Mean accuracy	0.4944	**0.7449**
	mIoU	0.4912	**0.6022**
	FWIoU	0.5222	**0.7857**

Table 4.10: The Vanilla, GP and LSTM model are trained with different batch sizes. The model prediction is compared side by side with different metrics. Hidden pattern in the data is learned quickly with increase in the batch size.

4.7 Experiment hyper-parameter and configuration detail

The experiments were conducted with the configuration listed in the table 4.11. Training of Vanilla, Gaussian Process and Long Short Term Memory models are done in parallel and the training time is listed in the table.

Parameter	Value
Learning rate	0.001
Batch size	8
Epoch	200
Weight decay factor	1e-5
Optimizer	Adam
Parallel processing	Cuda
Logger	tensorboard
Training time for Scannet	8 hours
Training time for Vkitti	5 hours
Learning rate Finder	torch lr finder, LRFinder

Table 4.11: Hyperparameter to train the model.

5

Android Deployment

The Android deployment section explains the procedures for integrating the trained and evaluated semantic segmentation model on an embedded device. The original model was trained and evaluated on a highly computational-powered device, but generally, the embedded device is a low computational device with less energy consumption. In this case, the embedded device used to deploy the segmentation model is an Android One Plus 7 mobile phone with 8 GB of RAM and a Snapdragon 855 processor. Before deploying the model on an Android device, the heavily trained model needs to be optimized to be compatible with the device.

5.1 Framework

The process of handling a single input image or a sequence is illustrated in Figure 5.1. The pipeline starts by loading an optimized model and a single image from the media storage in evaluation mode. The image is then passed through the model for prediction. In the final step, the segmented image from the python script is returned to the main Kotlin routine to be displayed. The model is deployed on an Android device with 6 GB of RAM running Android 11. The main routine script is written in Kotlin and integrates with the python script to process the data. The Android device used to deploy the deep learning model is specified in Table 5.1. The trained model was optimized using the PyTorch mobile optimizer with evaluation mode modules. The tested Android device takes 13 seconds to read, segment, and display a single input image.

Device	Oneplus 7
Android version	Oxygen OS 11.0.9.1.GM57AA
Processor	Snapdragon 855
RAM	6 GB
Model	GM1901

Table 5.1: Specification of the android device

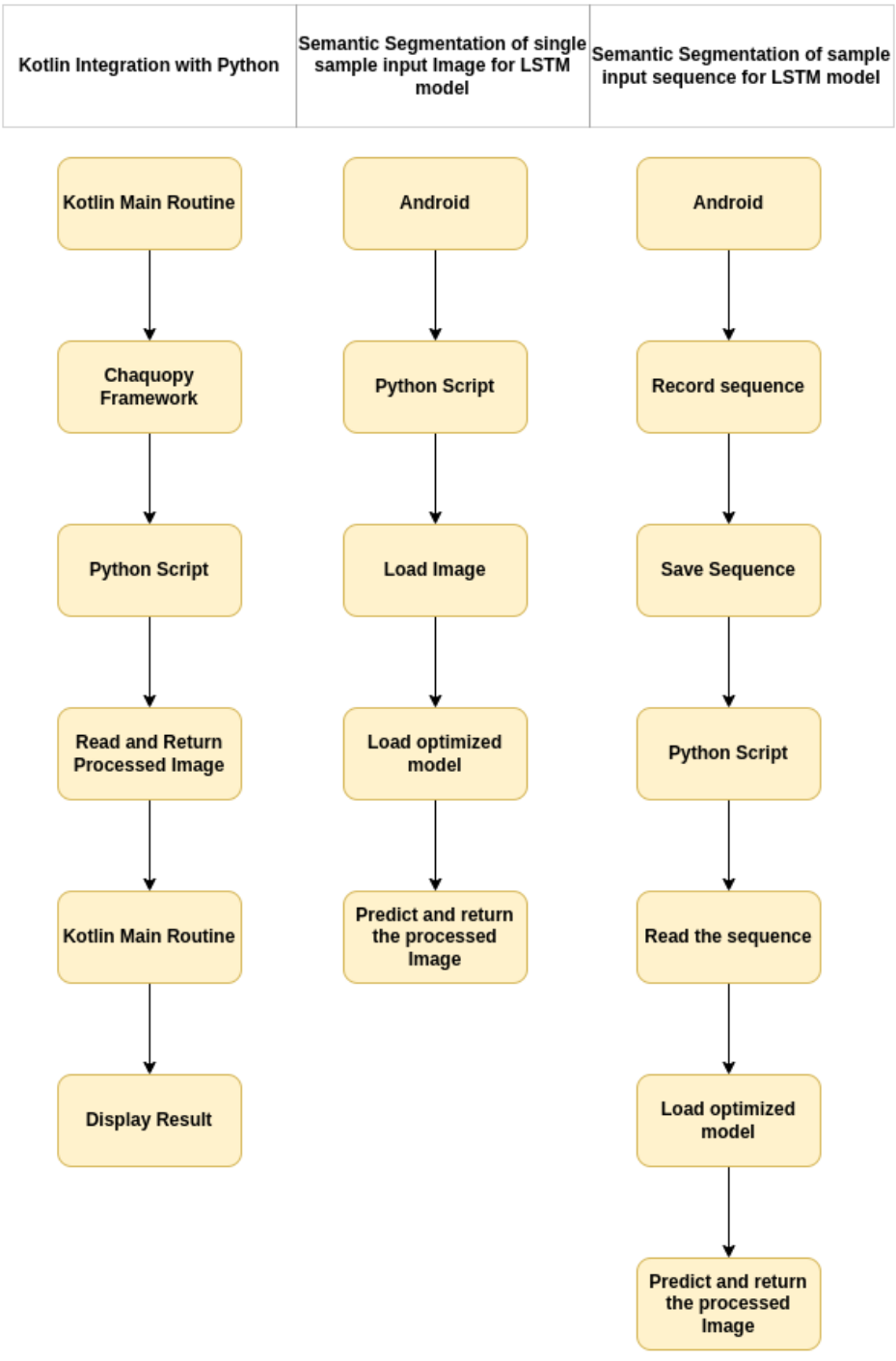


Figure 5.1: Kotlin integration with python and sequence of steps from loading the image to display of predicted semantic map.

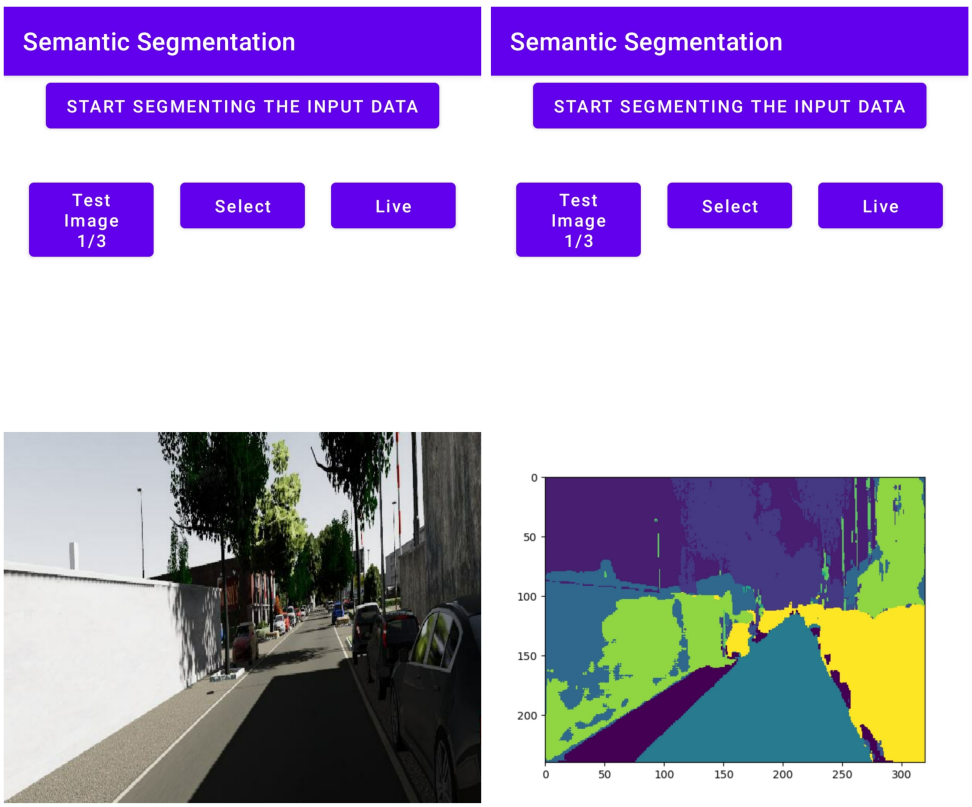


Figure 5.2: Display of loading the image and predicted semantic map

5.2 Display of results

The Android application is developed in Kotlin programming language. The layout of the application is shown in Figure C.10. The layout consists of six main parts: Heading, Start segmenting the input data, Test Image, Select, Live, and Results display. The Heading is the name of the application, Test Image is the image to be segmented, and any sample image can be selected for segmentation. The Select option allows you to choose images from the media folders and transfer them for processing. The Live option enables live video segmentation and display in real time. Once the target image has been selected, the Start segmenting the input data button initiates the processing of the input data and displays the result on the Android screen. A full description can be found in the image C.10.

5.3 Runtime

The android device is subjected with a sample input and the response time is tabulated in the table 5.2. The sample is a single frame from the video sequence data. The image goes through multiple stages starting from reading the image to processing and display of the results.

Android version	Oxygen OS 11.0.9.1.GM57AA
Image Loading time	2 seconds
Image processing	9 seconds
Display of result	2 seconds
Total	13 seconds

Table 5.2: Runtime detail for processing of single image

6

Conclusions

In general settings, semantic segmentation of the video sequence data is done by performing segmentation on the keyframes or the entire frames without fusing the rich information from the previous frame. The overlapping data in the consecutive frames can be propagated and fused forward to improve the segmentation performance. This project studies the latent space encoding temporal fusion in the context of continuous sequence video data with the help of the Gaussian Process (GP) and Long Short-Term Memory (LSTM). Comparative evaluation of the baseline, GP, and LSTM is conducted, and the best-performing model is deployed on an android device. In order to conduct the experiment, Scannet and Vkitti data are considered. It is a continuous video sequence data with overlapping information in consecutive frames. The overlapping data is leveraged for temporal fusion with the help of GP and LSTM. The overlapping information is modeled with the help of camera pose information for the GP, and a convolution LSTM cell is introduced in the latent space encoding for temporal fusion from the previous frames. The impact of the training batch size on the model performance is also studied. The main objective of the project is to

- Literature review on the temporal fusion models
- Comparison of the state of the art temporal fusion architecture performance
- Implementation of the baseline encoder-decoder type Unet model
- Model the overlapping information with pose information and fuse the data in the latent space encoding with the Gaussian Process
- Temporal fusion in the latent space encoding with LSTM
- Comparative evaluation of the baseline, GP and LSTM model performance
- Implementation of best performing model on the android device

Challenges encountered during the projects

- Finding the dataset containing the pose information
- How to fuse the information?

- Lightweight encoder-decoder architecture

The project objectives were implemented after thoroughly investigating the temporal fusion approach. The challenges are addressed technically. Temporal fusion in the context of video sequence data is challenging due to the problem's complexity. Suitable tools and standard approaches are followed to tackle the fusion of highly related consecutive frames information, thereby solving the fusion problem. Standard research approaches are followed in a technically sound manner, starting with a literature review to implementing the cross-transferred theorized approach from the depth estimation problem. A detailed introduction to the general temporal fusion approach, motivation, challenges and difficulties, use cases, and research questions are discussed in the introduction section 1. There is a significant amount of work focused on the temporal fusion approach, which involves combining features and using different architectures for fusion. Best performing temporal fusion approaches are discussed in the state-of-the-art section 2. After a detailed analysis of the state-of-the-art temporal fusion approaches, an appropriate dataset meeting the requirements set by the experiments need to be collected. A sample of the collected dataset is described. The architecture of the baseline Unet model, temporal fusing Gaussian process, and Long Short Term Memory Unet model is discussed in the methodology section 3. The training pipeline, evaluation pipeline, and hardware configuration are also discussed in this section. Multiple metrics are used to evaluate the performance of the baseline and temporally fused semantic segmentation models, such as Pixel accuracy, mean pixel accuracy, IoU, mIoU, and FwIoU. A hypothesis is initially developed before conducting the experiments. All the research questions starting with state-of-the-art temporal fusion architecture, comparison of model performance, cross-transfer of temporal fusion technique to semantic segmentation, Gaussian Process, and LSTM approaches, are discussed. The model performance results are tabulated and compared side by side to reject or accept the developed hypothesis. Finally, the best-performing models are described with different metrics in the evaluation and experimental result 4 section. The best-performing models are finally deployed in an android device using the Kotlin and Chaquopy framework. The implementation details are described in the android deployment 5 section. The report is summarized in the conclusion section 6.

6.1 Contributions

The contribution of the project are listed below

- Literature review on state of the art temporal fusion in semantic segmentation
- Detailed analysis of the video sequence data
- Implementation of the baseline encoder-decoder based Unet model
- Incorporation of Gaussian Process temporal fusion in latent space encoding of Unet model
- Incorporation of Long Short Term Memory temporal fusion in latent space encoding of Unet model
- Study of training batch size on the model performance
- Implementation of best performing model on a android device

6.2 Limitation

The temporal fusion using the Gaussian Process involve solving of a matrix equation to propagate the overlapping information forward, and the process involve matrix inversion. As the matrix size increases the inversion becomes costly and takes long time to process, thereby increasing the processing power requirements. The temporal fusion works if there is overlapping information in the consecutive frames. Hence for a video sequence data the frames rate needs to be high so that the probability of overlapping information is quite high. The effect of temporal fusion is not immediately seen rather observed after few frames prediction. Gaussian Process model the temporal data with the help of covariance matrix. The covariance matrix depend on the distance matrix, which in turn depend on the pose information of the captured frames.Thereby the Gaussian Process works only if there is pose information. Extra sensor is needed to extract the pose data. However, the Long Short Term Memory (LSTM) works without the need of pose information, but need high computation in comparison with the Gaussian Process. The android and ios are in early stage of deploying the deep learning models with these low computational device. There are lot of modules needed to deploy the developed model. Hence, a third party software needs to be incorporated to tackle the setback.

6.3 Future work

The transformers can be employed in the temporal fusion of the latent space encoding, thereby propagating the overlapping information forward. The kernel of the Gaussian Process can be changed, thereby experimenting with different kernel and tabulating the results is the future direction for the temporal fusion approach. The impact of temporal fusion on video sequence data containing non overlapping information can be studied in the future. Lighting condition has a affect on the model prediction and learning process. Noise can be injected into the video sequence data and uncertainty of model prediction can be studied.

A

Design Details

The code related to the project can be found at the following GITHUB repositories `https://github.com/RnDProjectsDeebul/ManojKolpeThesis`.

Listing A.1: Unet Vanilla model architecture description

```
UNet_vanilla(
(contracting_11): Sequential(
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
        track_running_stats=True)
(3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(contracting_12): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
     ceil_mode=False)
(contracting_21): Sequential(
(0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
(3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(contracting_22): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
     ceil_mode=False)
```

```
(contracting_31): Sequential(
(0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
(3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(contracting_32): MaxPool2d(kernel_size=2, stride=2, padding=0,
 dilation=1, ceil_mode=False)
(contracting_41): Sequential(
(0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
(3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(contracting_42): MaxPool2d(kernel_size=2, stride=2, padding=0,
     dilation=1, ceil_mode=False)
(middle): Sequential(
(0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
(3): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(expansive_11): ConvTranspose2d(1024, 512, kernel_size=(3, 3),
     stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_12): Sequential(
(0): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
```

```
(2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
(3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(expansive_21): ConvTranspose2d(512, 256, kernel_size=(3, 3),
     stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_22): Sequential(
(0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
(3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(expansive_31): ConvTranspose2d(256, 128, kernel_size=(3, 3),
     stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_32): Sequential(
(0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
(3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(expansive_41): ConvTranspose2d(128, 64, kernel_size=(3, 3),
     stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_42): Sequential(
(0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
(3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
(4): ReLU()
(5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
     track_running_stats=True)
)
(output): Conv2d(64, 41, kernel_size=(3, 3), stride=(1, 1),
     padding=(1, 1))
)
```

Listing A.2: Unet GP model architecture description

```
UNet_gp(
(contracting_11): Sequential(
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(contracting_12): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(contracting_21): Sequential(
(0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(contracting_22): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(contracting_31): Sequential(
(0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
(3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(contracting_32): MaxPool2d(kernel_size=2, stride=2, padding=0,
 dilation=1, ceil_mode=False)
(contracting_41): Sequential(
(0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(contracting_42): MaxPool2d(kernel_size=2, stride=2, padding=0,
 dilation=1, ceil_mode=False)
(middle): Sequential(
(0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(expansive_11): ConvTranspose2d(1024, 512, kernel_size=(3, 3),
 stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_12): Sequential(
(0): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
```

```
(5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True)
)
(expansive_21): ConvTranspose2d(512, 256, kernel_size=(3, 3),
stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_22): Sequential(
(0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(expansive_31): ConvTranspose2d(256, 128, kernel_size=(3, 3),
stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_32): Sequential(
(0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(expansive_41): ConvTranspose2d(128, 64, kernel_size=(3, 3),
 stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_42): Sequential(
(0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True)
(3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
 track_running_stats=True)
)
```

```
(output): Conv2d(64, 41, kernel_size=(3, 3), stride=(1, 1),
 padding=(1, 1))
)
```

Listing A.3: Unet LSTM model architecture description

```
UNet_lstm(
(contracting_11): Sequential(
(0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(contracting_12): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(contracting_21): Sequential(
(0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(contracting_22): MaxPool2d(kernel_size=2, stride=2, padding=0,
 dilation=1, ceil_mode=False)
(contracting_31): Sequential(
(0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
```

```
track_running_stats=True)
)
(contracting_32): MaxPool2d(kernel_size=2, stride=2, padding=0,
dilation=1, ceil_mode=False)
(contracting_41): Sequential(
(0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(contracting_42): MaxPool2d(kernel_size=2, stride=2, padding=0,
 dilation=1, ceil_mode=False)
(middle): Sequential(
(0): Conv2d(512, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(1024, 1024, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(1024, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(expansive_11): ConvTranspose2d(1024, 512, kernel_size=(3, 3),
stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_12): Sequential(
(0): Conv2d(1024, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(expansive_21): ConvTranspose2d(512, 256, kernel_size=(3, 3),
```

```
stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_22): Sequential(
(0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(expansive_31): ConvTranspose2d(256, 128, kernel_size=(3, 3),
stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_32): Sequential(
(0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(expansive_41): ConvTranspose2d(128, 64, kernel_size=(3, 3),
stride=(2, 2), padding=(1, 1), output_padding=(1, 1))
(expansive_42): Sequential(
(0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(1): ReLU()
(2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
(3): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(4): ReLU()
(5): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
)
(output): Conv2d(64, 41, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
(lstm_cell): MVSLayernormConvLSTMCell(
(conv): Conv2d(2048, 4096, kernel_size=(3, 3), stride=(1, 1),
```

```
padding=(1, 1), bias=False)
)
)
```

B

Parameters

Id	Classes
1	Wall, Shower Walls, Closet Wall, Shower Wall, Pantry Wall, Closet Walls, Bath Walls, Pantry Walls, Door Wall,
2	Floor, Shower Floor, Closet Floor,
3	Cabinet, Kitchen Cabinet, Kitchen Cabinets, File Cabinet, Bathroom Vanity, Cabinets, Bathroom Cabinet, Cabinet Doors, Open Kitchen Cabinet, File Cabinets, Trash Cabinet, Media Center,
4	Bed, Mattress, Loft Bed, Sofa Bed, Air Mattress,
5	Chair, Office Chair, Armchair, Sofa Chair, Stack Of Chairs, Folded Chair, Folded Chairs, Massage Chair, Recliner Chair, Rocking Chair, Stack Of Folded Chairs,
6	Couch, Sofa,
7	Table, Coffee Table, End Table, Dining Table, Folded Table, Round Table, Side Table, Air Hockey Table,
8	Door, Doorframe, Doors, Bathroom Stall Door, Closet Doors, Closet Door, Shower Door, Mirror Doors, Cabinet Door, Glass Doors, Sliding Door, Closet Doorframe,
9	Window,
10	Bookshelf, Bookshelves,
11	Picture, Poster, Painting, Pictures,
12	Kitchen Counter, Counter, Bathroom Counter,
13	Blinds,
14	Desk,
15	Shelf, Organizer Shelf, Pantry Shelf, Closet Shelf,
16	Curtain, Curtains,
17	Dresser,
18	Pillow, Pillows, Couch Cushions, Cushion,
19	Mirror,
20	Mat, Yoga Mat,
21	Clothes, Clothing, Cloth, Sock, Kitchen Apron, Costume, Socks,
22	Ceiling, Closet Ceiling,
23	Books, Book, Music Book,
24	Refrigerator, Mini Fridge, Cooler,
25	Tv,
26	Paper, Papers,
27	Towel, Towels, Hand Towel,
28	Shower Curtain,
29	Box, Boxes, Mailboxes, Mailbox, Storage Box, Pizza Box, Boxes Of Paper, Jewelry Box, Cat Litter Box, Covered Box, Pizza Boxes,
30	Whiteboard,
31	Person, Legs,
32	Nightstand,
33	Toilet, Urinal,
34	Sink,
35	Lamp, Lamp Base, Desk Lamp, Wall Lamp, Table Lamp, Ceiling Lamp, Night Lamp,
36	Bathtub,
37	Bag, Paper Bag, Messenger Bag, Ikea Bag, Duffel Bag, Bag Of Coffee Beans, Grocery Bag, Golf Bag, Garbage Bag, Coffee Bean Bag, Trash Bag, Cosmetic Bag, Shopping Bag, Food Bag,

Table B.1: Classes and ids of the Scannet dataset

Id	Classes
38	Board, Stove, Light, Bathroom Stall, Bar, Light Switch, Ceiling Light, Range Hood, Blackboard, Rail, Bulletin Board, Ledge, Shower, Windowsill, Dishwasher, Stair Rail, Stairs, Handicap Bar, Column, Oven, Pillar, Structure, Shower Head, Projector Screen, Staircase, Fireplace, Breakfast Bar, Hand Rail, Water Fountain, Kitchen Island, Pipes, Shower Control Valve, Handrail, Step, Dart Board, Grab Bar, Railing, Stair, Soap Bar, Studio Light, Shower Doors, Boards, Frame, Garage Door, Platform, Elevator, Wood Beam, Banister, Curtain Rod, Chandelier, Stovetop, Glass,
39	Trash Can, Radiator, Recycling Bin, Ottoman, Bench, Tv Stand, Wardrobe Closet, Trash Bin, Seat, Closet, Ladder, Piano, Water Cooler, Stand, Washing Machine, Rack, Washing Machines, Wardrobe Cabinet, Clothes Dryer, Ironing Board, Keyboard Piano, Music Stand, Furniture, Crate, Clothes Dryers, Drawer, Footrest, Piano Bench, Foosball Table, Footstool, Compost Bin, Tripod, Treadmill, Chest, Folded Ladder, Drying Rack, Pool Table, Heater, Toolbox, Beanbag Chair, Dollhouse, Ping Pong Table, Clothing Rack, Podium, Luggage Stand, Rack Stand, Futon, Book Rack, Seating, Workbench, Easel, Luggage Rack, Headboard, Display Rack, Crib, Bedframe, Closet Wardrobe, Wardrobe, Bunk Bed, Magazine Rack, Furnace, Stepladder, Baby Changing Station, Flower Stand, Display,

Table B.2: Classes and ids of the Scannet dataset

Id	Classes
40	Object, Monitor, Backpack, Plant, Toilet Paper, Shoes, Keyboard, Bottle, Stool, Computer Tower, Telephone, Cup, Jacket, Microwave, Paper Towel Dispenser, Suitcase, Laptop, Printer, Soap Dispenser, Fan, Tissue Box, Blanket, Copier, Soap Dish, Laundry Hamper, Storage Bin, Coffee Maker, Decoration, Clock, Mouse, Basket, Dumbbell, Bucket, Sign, Speaker, Container, Shower Curtain Rod, Tube, Storage Container, Paper Towel Roll, Ball, Laundry Basket, Cart, Dish Rack, Purse, Bicycle, Tray, Plunger, Paper Cutter, Toilet Paper Dispenser, Bin, Toilet Seat Cover Dispenser, Guitar, Fire Extinguisher, Pipe, Vacuum Cleaner, Plate, Cd Case, Bowl, Closet Rod, Scale, Broom, Hat, Guitar Case, Water Pitcher, Laundry Detergent, Hair Dryer, Divider, Power Outlet, Coffee Kettle, Toaster, Shoe, Alarm Clock, Water Bottle, Case Of Water Bottles, Toaster Oven, Coat Rack, Storage Organizer, Machine, Fire Alarm, Vent, Power Strip, Calendar, Toilet Paper Holder, Potted Plant, Stuffed Animal, Luggage, Headphones, Candle, Projector, Dustpan, Rod, Globe, Step Stool, Vending Machine, Ceiling Fan, Swiffer, Jar, Hamper, Poster Tube, Case, Carpet, Thermostat, Coat, Smoke Detector, Flip Flops, Banner, Clothes Hanger, Whiteboard Eraser, Iron, Instrument Case, Toilet Paper Rolls, Soap, Block, Wall Hanging, Toothbrush, Shirt, Cutting Board, Vase, Exercise Machine, Shorts, Tire, Teddy Bear, Bathrobe, Faucet, Thermos, Rug, Tupperware, Shoe Rack, Beer Bottles, Salt, Dispenser, Remote, Carton, Slippers, Soda Stream, Toilet Brush, Cooking Pot, Stapler, Scanner, Elliptical Machine, Kettle, Metronome, Dumbell, Rice Cooker, Sewing Machine, Flowerpot, Nerf Gun, Binders, Quadcopter, Pitcher, Hanging, Mail, Hoverboard, Water Heater, Spray Bottle, Rope, Plastic Container, Soap Bottle, Sleeping Bag, Frying Pan, Oven Mitt, Pot, Hand Dryer, Shampoo Bottle, Hair Brush, Tennis Racket, Display Case, Boiler, Bananas, Carseat, Helmet, Umbrella, Coffee Box, Envelope, Wet Floor Sign, Controller, Dolly, Shampoo, Paper Tray, Changing Station, Poster Printer, Screen, Crutches, Stack Of Cups, Toilet Flush Button, Trunk, Plastic Bin, Car, Shaving Cream, Shredder, Statue, Hose, Bike Pump, Coatrack, Bear, Humidifier, Toothpaste, Mouthwash Bottle, Poster Cutter, Food Container, Camera, Card, Mug, Cardboard, Flag, Magazine, Exit Sign, Rolled Poster, Wheel, Blackboard Eraser, Organizer, Doll, Laundry Bag, Sponge, Lotion Bottle, Can, Lunch Box, Food Display, Storage Shelf, Sliding Wood Door, Pants, Wood, Bottles, Washcloth, Cups, Exercise Ball, Roomba, Bike Lock, Briefcase, Bath Products, Star, Map, Ipad, Traffic Cone, Toiletry, Canopy, Paper Organizer, Barricade, Cap, Dumbbell Plates, Cooking Pan, Santa, Boat, Kinect, Plastic Storage Bin, Dishwashing Soap Bottle, Xbox Controller, Banana Holder, Ping Pong Paddle, Airplane, Conditioner Bottle, Tea Kettle, Toilet Paper Package, Wall Mounted Coat Rack, Film Light, Chain, Sweater, Kitchen Mixer, Water Softener, Trolley, Loofa, Shower Faucet Handle, Toy Piano, Fish, Electric Panel, Suitcases, Tape, Plates, Alarm, Fire Hose, Toy Dinosaur, Cone, Hatrack, Subwoofer, Fire Sprinkler, Photo, Barrier, Stacks Of Cups, Beachball, Folded Boxes, Contact Lens Solution Bottle, Folder, Mail Trays, Slipper, Sticker, Lotion, Buddha, File Organizer, Paper Towel Rolls, Fuse Box, Knife Block, Cd Cases, Stools, Hand Sanitzer Dispenser, Teapot, Pen Holder, Tray Rack, Wig, Switch, Plastic Containers, Night Light, Notepad, Mail Bin, Elevator Button, Gaming Wheel, Drum Set, Coffee Mug, Baby Mobile, Diaper Bin, Stepstool, Paper Shredder, Dress Rack, Cover, Exercise Bike, Kitchenaid Mixer, Soda Can, Tap, Cable, Binder, Towel Rack, Medal, Telescope, Baseball Cap, Battery Disposal Jar, Mop, Tank, Mail Tray, Centerpiece, Stick, Dryer Sheets, Bycicle, Clip, Postcard, Display Sign, Paper Towel, Boots, Tennis Racket Bag, Clothes Hangers, Starbucks Cup,
0	Other

Table B.3: Classes and ids of the Scannet dataset

Id	Classes
0	Other
1	Terrain
2	Sky
3	Tree
4	Vegetation
5	Building
6	Road
7	Guardrail
8	Traffic Sign
9	Traffic light
10	Pole
11	Misc
12	Truck
13	Car
14	Van

Table B.4: Vkitti dataset classes

C

Training details

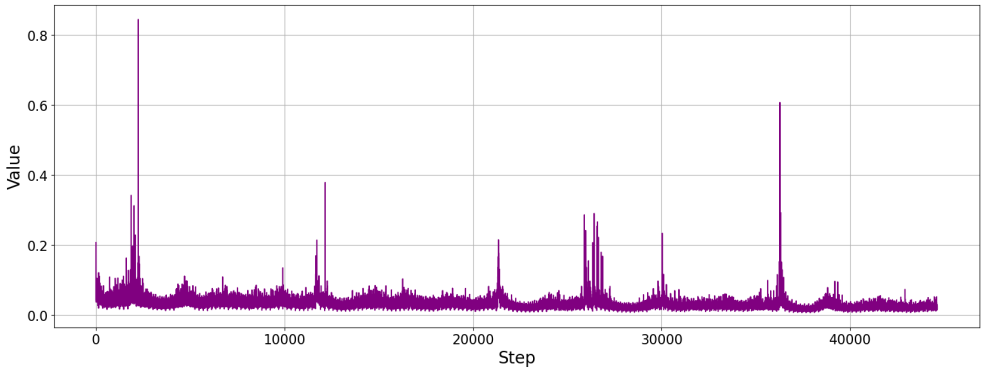


Figure C.1: Scannet step losses for vanilla

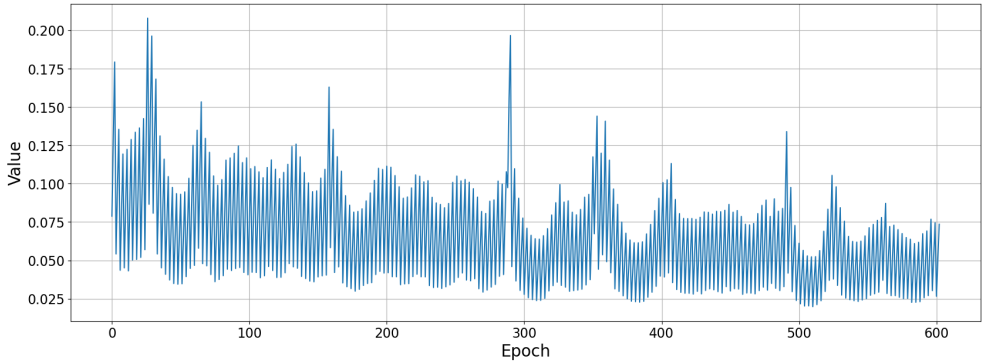


Figure C.2: Scannet epoch losses for vanilla

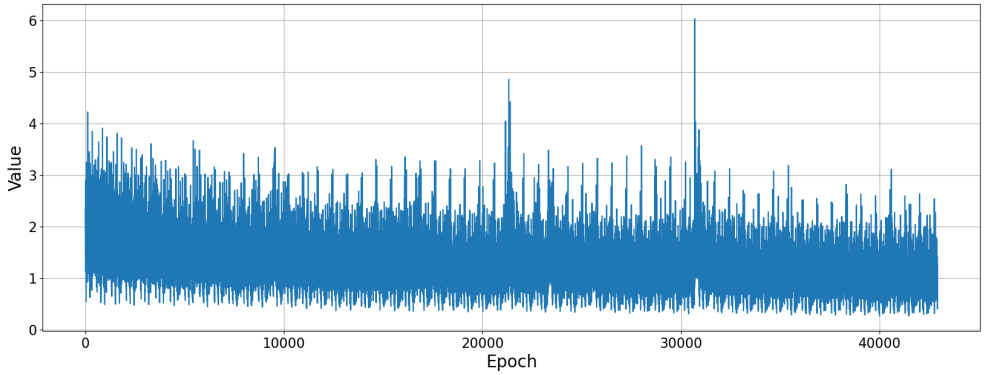


Figure C.3: Scannet step losses for GP

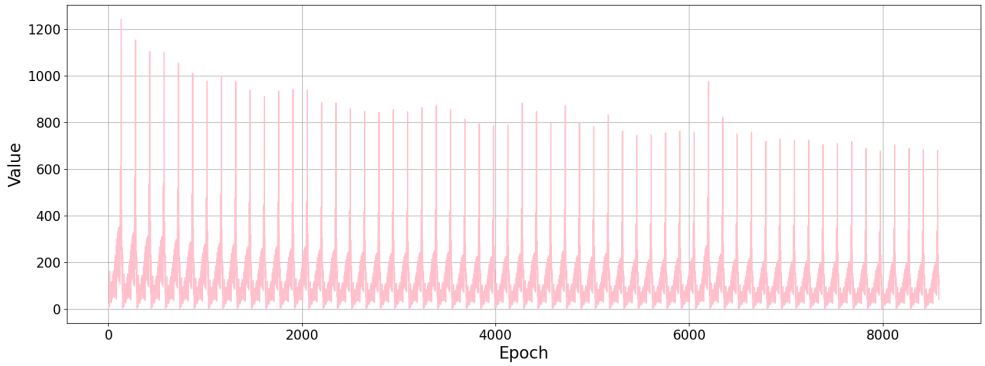


Figure C.4: Scannet epoch losses for GP

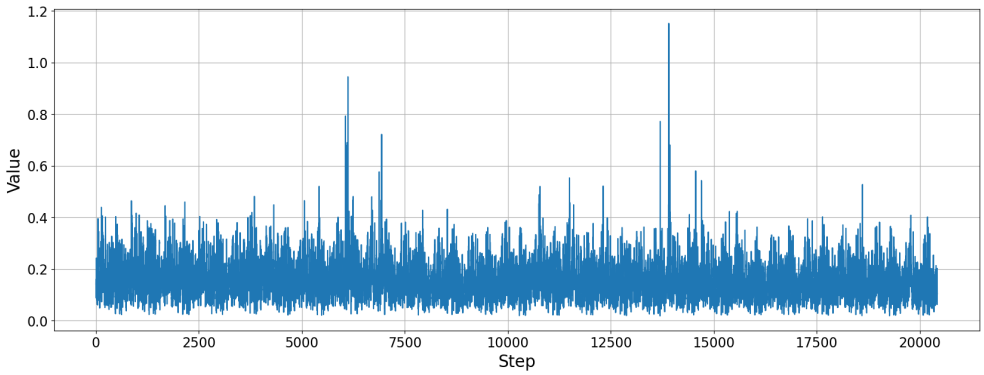


Figure C.5: Scannet step losses for lstm

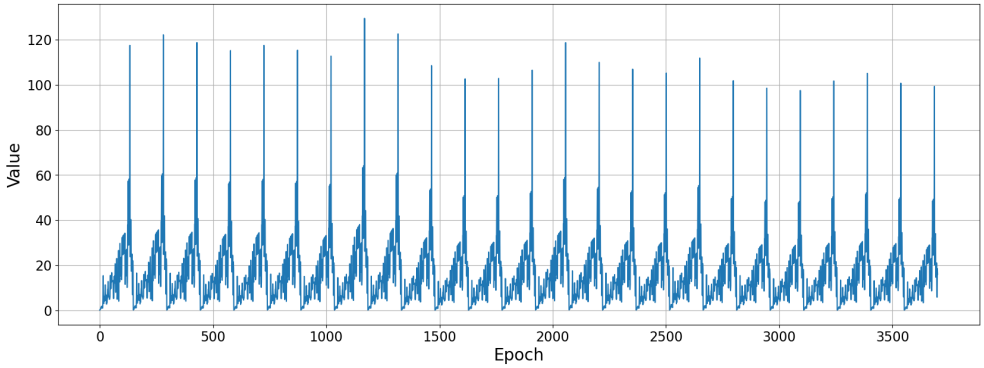


Figure C.6: Scannet epoch losses for lstm

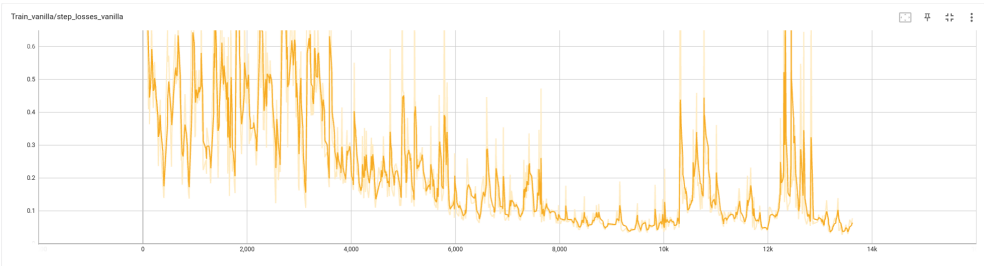


Figure C.7: Vkitti step loss vanilla

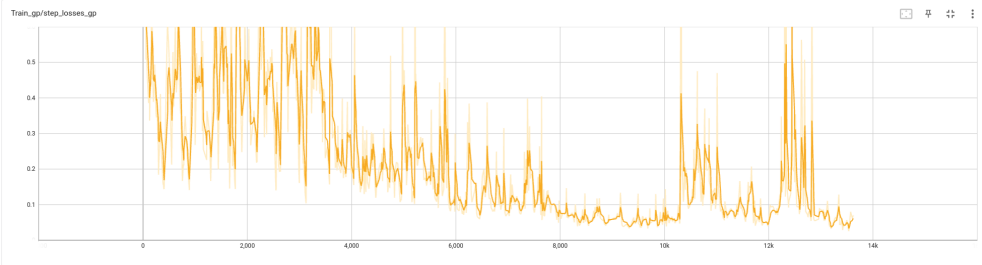


Figure C.8: Vkitti step loss GP

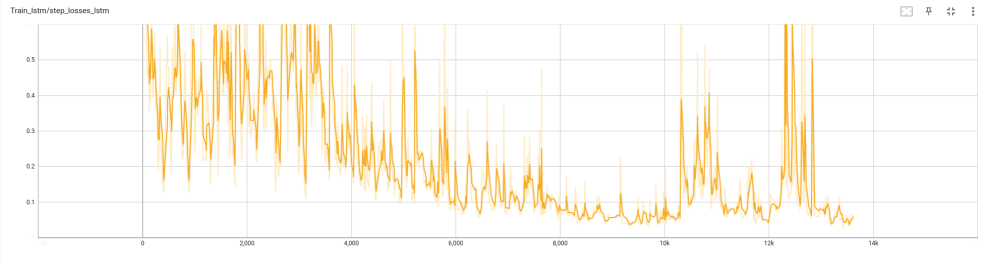


Figure C.9: Vkitti step loss lstm

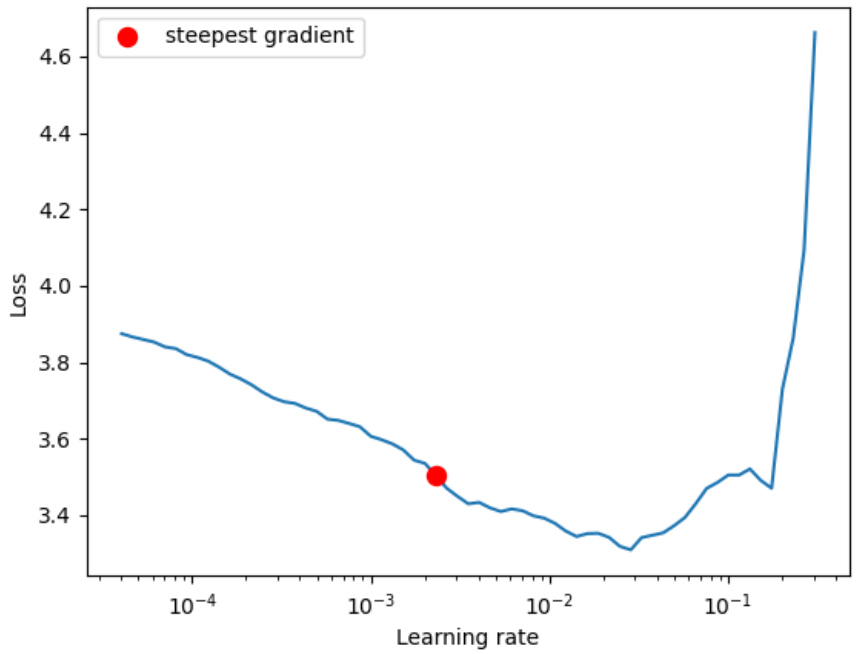


Figure C.10: Learning rate finder

References

[1] Michael Middleton. Deep learning vs. machine learning — what's the difference?, 2021.

[2] Yuxin Hou, Juho Kannala, and Arno Solin. Multi-view stereo by temporal nonparametric fusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2651–2660, 2019.

[3] Your complete guide to image segmentation. `https://www.telusinternational.com/articles/guide-to-image-segmentation#:~:text=Different%20types%20of%20image%20segmentation%20tasks,types%20of%20image%20segmentation%20tasks.` Accessed: 2022-08-22.

[4] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[5] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[6] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[7] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[8] Ping Hu, Fabian Caba, Oliver Wang, Zhe Lin, Stan Sclaroff, and Federico Perazzi. Temporally distributed networks for fast video semantic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8818–8827, 2020.

[9] Adrian Rosebrock. Intersection over union (iou) for object detection, 2016.

[10] Danilo P Mandic, Dragan Obradovic, Anthony Kuh, Tülay Adali, Udo Trutschell, Martin Golz, Philippe De Wilde, Javier Barria, Anthony Constantinides, and Jonathon Chambers. Data fusion for modern engineering applications: An overview. In *International Conference on Artificial Neural Networks*, pages 715–721. Springer, 2005.

[11] Federico Castanedo. A review of data fusion techniques. *The scientific world journal*, 2013, 2013.

[12] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[13] Arda Duzceker, Silvano Galliani, Christoph Vogel, Pablo Speciale, Mihai Dusmanu, and Marc Pollefeys. Deepvideomvs: Multi-view stereo on video with recurrent spatio-temporal fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15324–15333, 2021.

[14] Minghan Li, Shuai Li, Lida Li, and Lei Zhang. Spatial feature calibration and temporal fusion for effective one-stage video instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11215–11224, 2021.

[15] Ko Ming Hsiao, Geoff West, Svetha Venkatesh, and Mohan Kumar. Temporal data fusion in multisensor systems using dynamic time warping. In *SENSORFUSION 2005: Workshop on Information Fusion and Dissemination in Wireless Sensor Networks*, pages 1–9. IEEE, 2005.

[16] Ko Ming Hsiao, Geoff West, Svetha Venkatesh, and Mohan Kumar. Temporal data fusion in multisensor systems using dynamic time warping. In *SENSORFUSION 2005: Workshop on Information Fusion and Dissemination in Wireless Sensor Networks*, pages 1–9. IEEE, 2005.

[17] Andreas Krause, Daniel P Siewiorek, Asim Smailagic, and Jonny Farringdon. Unsupervised, dynamic identification of physiological and activity context in wearable computing. In *ISWC*, volume 3, page 88, 2003.

[18] Jong-Min Lee, ChangKyoo Yoo, and In-Beum Lee. On-line batch process monitoring using a consecutively updated multiway principal component analysis model. *Computers & Chemical Engineering*, 27(12):1903–1912, 2003.

[19] Wei Han, Pooya Khorrami, Tom Le Paine, Prajit Ramachandran, Mohammad Babaeizadeh, Honghui Shi, Jianan Li, Shuicheng Yan, and Thomas S Huang. Seq-nms for video object detection. *arXiv preprint arXiv:1602.08465*, 2016.

[20] Kai Kang, Hongsheng Li, Junjie Yan, Xingyu Zeng, Bin Yang, Tong Xiao, Cong Zhang, Zhe Wang, Ruohui Wang, Xiaogang Wang, et al. T-cnn: Tubelets with convolutional neural networks for object detection from videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 28(10):2896–2907, 2017.

[21] Guanghan Ning, Zhi Zhang, Chen Huang, Xiaobo Ren, Haohong Wang, Canhui Cai, and Zhihai He. Spatially supervised recurrent convolutional neural networks for visual object tracking. In *2017 IEEE international symposium on circuits and systems (ISCAS)*, pages 1–4. IEEE, 2017.

[22] Zhichao Lu, Vivek Rathod, Ronny Votel, and Jonathan Huang. Retinatrack: Online single stage joint detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14668–14678, 2020.

[23] Okan Köpüklü, Xiangyu Wei, and Gerhard Rigoll. You only watch once: A unified cnn architecture for real-time spatiotemporal action localization. *arXiv preprint arXiv:1911.06644*, 2019.

[24] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1933–1941, 2016.

[25] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

[26] Emeç Erçelik, Ekim Yurtsever, and Alois Knoll. Temp-frustum net: 3d object detection with temporal fusion. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 1095–1101. IEEE, 2021.

[27] Jiejie Zhu, Liang Wang, Jizhou Gao, and Ruigang Yang. Spatial-temporal fusion for high accuracy depth maps using dynamic mrfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(5):899–909, 2009.

[28] Gang Wu, Yi Wu, Long Jiao, Yuan-Fang Wang, and Edward Y Chang. Multi-camera spatio-temporal fusion and biased sequence-data learning for security surveillance. In *Proceedings of the eleventh ACM international conference on Multimedia*, pages 528–538, 2003.

[29] Michael Teutsch and Wolfgang Krüger. Spatio-temporal fusion of object segmentation approaches for moving distant targets. In *2012 15th International Conference on Information Fusion*, pages 1988–1995. IEEE, 2012.

[30] David Forsyth and Jean Ponce. *Computer vision: A modern approach.* Prentice hall, 2011.

[31] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3150–3158, 2016.

[32] King-Sun Fu and JK Mui. A survey on image segmentation. *Pattern recognition*, 13(1):3–16, 1981.

[33] W Ladys law Skarbek and Andreas Koschan. Colour image segmentation a survey. *IEEE Transactions on circuits and systems for Video Technol-ogy*, 14(7):1–80, 1994.

[34] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[35] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[36] 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.

[37] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, and Yoshua Bengio. The one hundred layers tiramisu: Fully convolutional densenets for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 11–19, 2017.

[38] Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

[39] Maoke Yang, Kun Yu, Chi Zhang, Zhiwei Li, and Kuiyuan Yang. Denseaspp for semantic segmentation in street scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3684–3692, 2018.

[40] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[41] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnet for real-time semantic segmentation on high-resolution images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 405–420, 2018.

[42] Jizhong Deng, Zhaoji Zhong, Huasheng Huang, Yubin Lan, Yuxing Han, and Yali Zhang. Lightweight semantic segmentation network for real-time weed mapping using unmanned aerial vehicles. *Applied Sciences*, 10(20):7132, 2020.

[43] Chen-Chiung Hsieh, Dung-Hua Liou, and David Lee. A real time hand gesture recognition system using motion history image. In *2010 2nd international conference on signal processing systems*, volume 2, pages V2–394. IEEE, 2010.

[44] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[45] Clement Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1915–1929, 2012.

[46] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.

[47] Lauv Patel, Tripti Shukla, Xiuzhen Huang, David W Ussery, and Shanzhi Wang. Machine learning methods in drug discovery. *Molecules*, 25(22):5277, 2020.

[48] T Ciodaro, D Deva, JM De Seixas, and D Damazio. Online particle detection with neural networks based on topological calorimetry information. In *Journal of physics: conference series*, volume 368, page 012030. IOP Publishing, 2012.

[49] Jinny X Zhang, Boyan Yordanov, Alexander Gaunt, Michael X Wang, Peng Dai, Yuan-Jyue Chen, Kerou Zhang, John Z Fang, Neil Dalchau, Jiaming Li, et al. A deep learning model for predicting next-generation sequencing depth from dna sequence. *Nature communications*, 12(1):1–10, 2021.

[50] Julia Hirschberg and Christopher D Manning. Advances in natural language processing. *Science*, 349(6245):261–266, 2015.

[51] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. Deep learning vs. traditional computer vision. In *Science and information conference*, pages 128–144. Springer, 2019.

[52] Sharada P Mohanty, David P Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. *Frontiers in plant science*, 7:1419, 2016.

[53] Zhongchun Han and Anfeng Xu. Ecological evolution path of smart education platform based on deep learning and image detection. *Microprocessors and Microsystems*, 80:103343, 2021.

[54] Shrey Srivastava, Amit Vishvas Divekar, Chandu Anilkumar, Ishika Naik, Ved Kulkarni, and V Pattabiraman. Comparative analysis of deep learning image detection algorithms. *Journal of Big Data*, 8(1):1–27, 2021.

[55] Shervin Minaee, Yuri Y Boykov, Fatih Porikli, Antonio J Plaza, Nasser Kehtarnavaz, and Demetri Terzopoulos. Image segmentation using deep learning: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 2021.

[56] Christian S Jensen and Richard T Snodgrass. Temporal data management. *IEEE Transactions on knowledge and data engineering*, 11(1):36–44, 1999.

[57] Gowtham Atluri, Anuj Karpatne, and Vipin Kumar. Spatio-temporal data mining: A survey of problems and methods. *ACM Computing Surveys (CSUR)*, 51(4):1–41, 2018.

[58] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International Journal of Forecasting*, 37(4):1748–1764, 2021.

[59] Yue Meng, Rameswar Panda, Chung-Ching Lin, Prasanna Sattigeri, Leonid Karlinsky, Kate Saenko, Aude Oliva, and Rogerio Feris. Adafuse: Adaptive temporal fusion network for efficient action recognition. *arXiv preprint arXiv:2102.05775*, 2021.

[60] Arda Duzceker, Silvano Galliani, Christoph Vogel, Pablo Speciale, Mihai Dusmanu, and Marc Pollefeys. Deepvideomvs: Multi-view stereo on video with recurrent spatio-temporal fusion. In

Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 15324–15333, 2021.

[61] Kai Zhang, Yifan Sun, Rui Wang, Haichang Li, and Xiaohui Hu. Multiple fusion adaptation: A strong framework for unsupervised semantic segmentation adaptation. *arXiv preprint arXiv:2112.00295*, 2021.

[62] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[63] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[64] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on pattern analysis and machine intelligence*, 23(11):1222–1239, 2001.

[65] J-L Starck, Michael Elad, and David L Donoho. Image decomposition via the combination of sparse representations and a variational approach. *IEEE transactions on image processing*, 14(10):1570–1582, 2005.

[66] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[67] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[68] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

[69] Ke Sun, Yang Zhao, Borui Jiang, Tianheng Cheng, Bin Xiao, Dong Liu, Yadong Mu, Xinggang Wang, Wenyu Liu, and Jingdong Wang. High-resolution representations for labeling pixels and regions. *arXiv preprint arXiv:1904.04514*, 2019.

[70] Jun Fu, Jing Liu, Yuhang Wang, Jin Zhou, Changyong Wang, and Hanqing Lu. Stacked deconvolutional network for semantic segmentation. *IEEE Transactions on Image Processing*, 2019.

[71] Ronghang Hu, Piotr Dollár, Kaiming He, Trevor Darrell, and Ross Girshick. Learning to segment every thing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4233–4241, 2018.

[72] Xide Xia and Brian Kulis. W-net: A deep model for fully unsupervised image segmentation. *arXiv preprint arXiv:1711.08506*, 2017.

[73] Fausto Milletari, Nassir Navab, and Seyed-Ahmad Ahmadi. V-net: Fully convolutional neural networks for volumetric medical image segmentation. In *2016 fourth international conference on 3D vision (3DV)*, pages 565–571. IEEE, 2016.

[74] Xiaojie Jin, Xin Li, Huaxin Xiao, Xiaohui Shen, Zhe Lin, Jimei Yang, Yunpeng Chen, Jian Dong, Luoqi Liu, Zequn Jie, et al. Video scene parsing with predictive feature learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5580–5588, 2017.

[75] Raghudeep Gadde, Varun Jampani, and Peter V Gehler. Semantic video cnns through representation warping. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4453–4462, 2017.

[76] Xiaojie Jin, Xin Li, Huaxin Xiao, Xiaohui Shen, Zhe Lin, Jimei Yang, Yunpeng Chen, Jian Dong, Luoqi Liu, Zequn Jie, et al. Video scene parsing with predictive feature learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5580–5588, 2017.

[77] David Nilsson and Cristian Sminchisescu. Semantic video segmentation by gated recurrent flow propagation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6819–6828, 2018.

[78] Samvit Jain, Xin Wang, and Joseph E Gonzalez. Accel: A corrective fusion network for efficient semantic segmentation on video. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8866–8875, 2019.

[79] Behrooz Mahasseni, Sinisa Todorovic, and Alan Fern. Budget-aware deep semantic video segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1029–1038, 2017.

[80] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 2017.

[81] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2, 2020.

[82] Koji Minoda, Fabian Schilling, Valentin Wüest, Dario Floreano, and Takehisa Yairi. VIODE: A simulated dataset to address the challenges of visual-inertial odometry in dynamic environments. *IEEE Robotics and Automation Letters*, 6(2):1343–1350, 2021.

[83] Irem Ulku and Erdem Akagündüz. A survey on deep learning-based architectures for semantic segmentation on 2d images. *Applied Artificial Intelligence*, pages 1–45, 2022.

[84] JEREMY JORDAN. Evaluating image segmentation models, 2018.

[85] Kaggle. Kaggle competition.

[86] Ko Ming Hsiao, Geoff West, Svetha Venkatesh, and Mohan Kumar. Temporal data fusion in multisensor systems using dynamic time warping. In *SENSORFUSION 2005: Workshop on Information Fusion and Dissemination in Wireless Sensor Networks*, pages 1–9. IEEE, 2005.

[87] Xiang Li, Jinglu Wang, Xiao Li, and Yan Lu. Hybrid instance-aware temporal fusion for online video instance segmentation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 1429–1437, 2022.

[88] Miran Heo, Sukjun Hwang, Seoung Wug Oh, Joon-Young Lee, and Seon Joo Kim. Vita: Video instance segmentation via object token association. *arXiv preprint arXiv:2206.04403*, 2022.

[89] De-An Huang, Zhiding Yu, and Anima Anandkumar. Minvis: A minimal video instance segmentation framework without video-based training. *arXiv preprint arXiv:2208.02245*, 2022.

[90] Bowen Cheng, Anwesa Choudhuri, Ishan Misra, Alexander Kirillov, Rohit Girdhar, and Alexander G Schwing. Mask2former for video instance segmentation. *arXiv preprint arXiv:2112.10764*, 2021.

[91] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5828–5839, 2017.

[92] Manolis Savva Maciej Halber Thomas Funkhouser Matthias Nießner Angela Dai, Angel X. Chang. Scannet: Richly-annotated 3d reconstructions of indoor scenes, 2018.

[93] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual kitti 2. *arXiv preprint arXiv:2001.10773*, 2020.

[94] Manolis Savva Maciej Halber Thomas Funkhouser Matthias Nießner Angela Dai, Angel X. Chang. Scannet: Richly-annotated 3d reconstructions of indoor scenes, 2018.

[95] Manoj Kolpe Lingappa. Scannet data processing, 2022.

[96] Claudio Mazzotti, Nicola Sancisi, and Vincenzo Parenti-Castelli. A measure of the distance between two rigid-body poses based on the use of platonic solids. In *Symposium on Robot Design, Dynamics and Control*, pages 81–89. Springer, 2016.

[97] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 11 2005.

[98] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[99] Andrea Palazzi. Convlstm pytorch, 2022.

[100] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. *Advances in neural information processing systems*, 28, 2015.