DEEP LEARNING FOR ROAD SEGMENTATION IN INDIAN CONTEXT

DISSERTATION PHASE-2 REPORT

Submitted in partial fulfillment of
the requirement for the award of M.Tech Degree in
Computer Science and Engineering with specialization in Artificial Intelligence
of the APJ Abdul Kalam Technological University

Submitted By

Daya Mariam Alex

TVE22CSAI09

M.Tech Computer Science and Engineering
with specialization in Artificial Intelligence

Under the guidance of

Dr. Jerrin Thomas Panachakel

Assistant Professor

Dept. of ECE
CET, Thiruvananthapuram

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
COLLEGE OF ENGINEERING TRIVANDRUM

JUNE 2024

DECLARATION

I undersigned hereby declare that the dissertation phase II report entitled "**Deep Learning For Road Segmentation In Indian Context**", submitted for partial fulfillment of the requirements for the award of the degree of Master of Technology of the **APJ Abdul Kalam Technological University, Kerala** is a bonafide work done by me under supervision of **Dr. Jerrin Thomas Panachekal**. This submission represents my ideas in my own words, and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and or the university and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not previously formed the basis for the award of any degree, diploma, or similar title from any other university.

Place : Trivandrum

Date : 14-06-2024 Daya Mariam Alex

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COLLEGE OF ENGINEERING TRIVANDRUM

CERTIFICATE

This is to certify that this Dissertation Phase-2 report entitled **" DEEP LEARNING FOR ROAD SEGMENTATION IN INDIAN CONTEXT"** is a bonafide record of Dissertation done by **DAYA MARIAM ALEX**, KTU ID: **TVE22CSAI09** under our guidance towards the partial fulfillment of the requirements for the award of the Degree of Master of Technology in Computer Science and Engineering with specialization in Artificial Intelligence, of the APJ Abdul Kalam Technological University during the year 2022-2024.

Dr. Jerrin Thomas Panachakel	Dr. Ajeesh Ramanujan	Dr. Salim A
Assistant Professor	Associate Professor	Professor
Project Guide	Project Co-ordinator	Head of the Department
Department of ECE	Department of CSE	Department of CSE

Abstract

Road segmentation is a computer vision task that involves identifying and separating the pixels corresponding to drivable roads from an image. It is an important step in various applications like advanced driver-assisted systems, autonomous driving, urban planning, and traffic management. Deep learning techniques, such as convolutional neural networks, are commonly used for this task.

This project focuses on implementing Road Segmentation on the Indian Driving Dataset (IDD). The Weights & Biases (WandB) logger was utilized to track experiments and fine-tune hyperparameters for training various segmentation models with a pretrained MobileNet-v2 backbone on a subset of the IDD dataset (IDD-lite). Additionally, the trained models were deployed on JetsonTX2, and their inference speeds were benchmarked.

Keywords: Road Segmentation, IDD-lite, Segmentation-Models-Pytorch, MobileNet-v2, ENet, Wandb Logger, TensorRT, JetsonTX2

Acknowledgment

I express my gratitude to everyone who directly and indirectly helped me to complete this project.

I appreciate Dr. Savier JS, Principal of CET, Dr. Salim A, Head of the Department, Department of Computer Science and Engineering, CET, and Dr. Ajeesh Ramanujan, Project Coordinate, Department of Computer Science and Engineering, CET for their support in ensuring all facilities for undertaking this project in the campus.

I especially thank my guide Dr. Jerrin Thomas Panachakel, Assistant Professor, Department of Electronics and Communication Engineering, CET, and co-guide Dr. Deepak S, Assistant Professor, Department of Electronics and Communication Engineering, GEC Thrissur for giving me valuable suggestions and prompt expert feedback at all stages of my project.

I also thank my friends and family for their constant encouragement throughout my project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With rapid global progress in autonomous vehicle technology, India is now catching up. Despite numerous R&D, regulatory, and infrastructural challenges, there are some Indian startups making headlines with their indigenous autonomous vehicle technology, Swaayyaatt Robotics claims to have developed Advanced Level-5 autonomy in autonomous navigation and secured $4 Million to advance its development.[3].

Panoptic segmentation of Birds Eye View(BEV) of Road scenes is used by autonomous vehicles for downstream tasks like planning and control[4]. Segmentation of drivable roads is a key primary subtask that must be achieved to achieve the complex task of Panoptic segmentation in BEV. Segmenting road pixels have intrinsic complexity, especially in unstructured driving environments. This project focuses on achieving Road Segmentation on the Indian Driving Dataset.

Indian Driving Dataset curated by IIIT-H is a landmark dataset containing images of roads in unstructured driving environments.[5],[6]. There have been studies on Semantic segmentation on the Dataset, but Road segmentation has not been focused. A lightweight Road Segmentation AI model can be used in ADAS in private and public transportation, autonomous navigation of specialized road maintenance machines, etc. In Dissertation Phase I, it was decided to carry out FCN-8 and ENet on the Indian Driving Dataset and three main challenges were identified to overcome. The three challenges were:

1. Managing the significant loss.

2. Ensuring systematic experiment tracking.

3. Developing an effective training strategy for real-time model ENet.

FCN-8 with 14M encoder parameters was discarded when MobileNet-v2 encoder with 2M encoder parameters was found in the library Segmentation-Models-PyTorch. Instead, different SOTA segmentation models with pretrained MobileNet-v2 backbone were experimented with. To prevent huge losses from occurring during training, a Hyperparameter sweep using Wandb Logger[7] was used to determine the correct combination of learning rates, batch size, and image size. The Logger was also used to track experiments as detailed in 3. After the models were trained, their ONNX files were used to run the models on JetsonTX2 using TensorRT, as elaborated in 8.

The Python implementation of ENet's training was revised on finding the author's source code written in Lua[8], the implementation couldn't be completed as debugging the code for distributed training was left unfinished due to time constraints. The results of implementation in a single GPU are detailed in 9

By overcoming key challenges, this Dissertation Phase II entails the initial groundwork done to develop end-to-end Deep Learning models for Road Segmentation in BEV space for the Indian Driving Dataset and contribute to the domain of advancing autonomous navigation in India.

1 Problem Statement

Perform pixel-level classification to distinguish drivable roads from images taken from monocular cameras and output binary mask.

2 Objectives

- Preprocess the Indian Driving Dataset(IDD) for Road segmentation.

Figure 1.1: Problem Statement: Binary Segmentation of Drivable Roads

- Train and evaluate deep learning models on the IDD to carry out road segmentation.

Chapter 2

Literature Review

Extensive literature was surveyed throughout Mini-project, Dissertation Phase I, and Dissertation Phase II to gather information about the relevant trends, nuances of concepts, persisting problems, and relevant optimization techniques in the selected topic of research. The review encapsulated here aims to coalesce an identified literature gap with the relevant literature to overcome this gap.

Identified Literature Gap: Road Segmentation for Indian contexts is a useful stand-alone task that is not directly studied using the Indian Driving Dataset.

1 Road Segmentation

Vision-Centric BEV Perception: A Survey[1]

This paper discusses the updates in vision-centric Bird's Eye View (BEV) perception, which is critical for autonomous driving. The paper discusses the evolution from traditional geometric methods to modern deep learning techniques for BEV perception, categorizing these approaches into depth-based, MLP-based, and transformer-based methods. The survey highlights the strengths and limitations of each approach, presents detailed analyses and comparisons, and suggests future research directions.

The paper discusses practical implementation details on existing datasets but does not discuss the process involved in creating the ground truth required for the deep-learning techniques.

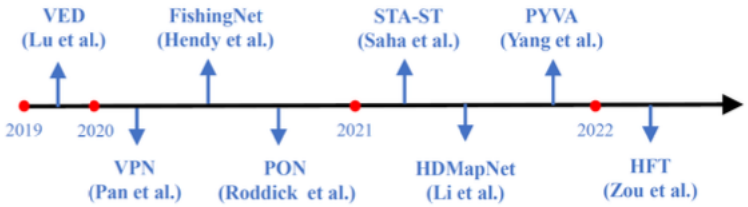


Figure 2.1: Chronological overview of MLP-based PV to BEV methods[1]

Monocular Semantic Occupancy Grid Mapping with Convolutional Variational Encoder-Decoder Networks[9]

This is a seminal paper that uses a Variational Encoder Decoder(VED) to map perspective images to Semantic Occupancy Grids in BEV space. Their goal was to learn metric, semantic, and topological information simultaneously in real-time from photometric data.

VED is composed of a low-level feature extractor and a modified version of the Variational auto-encoder(VAE). They trained the network in a supervised encoder-decoder manner with combined latent loss of KL divergence and mapping loss of Cross-entropy. This paper also details the ground truth creation from the Cityscapes Semantic dataset using its semantic annotations and stereo images via semantic segmentation and semi-global matching disparity.

The method could achieve real-time performance with inference rates of around 35 Hz for images at a resolution of 256×512 pixels and output maps with 64×64 grid cells using a Titan V GPU. The method demonstrates resilience to pitch and roll perturbations and generalizes well to the KITTI dataset, despite differences in domain and camera parameters.

One limitation of the ground truth creation suggested, is that it does not mitigate the distortions that arise in the BEV space caused due to pixels located above the ground.

Driving among Flatmobiles: Bird-Eye-View occupancy grids from a monocular camera for holistic trajectory planning[2]

To mitigate the distortion caused by projecting 3D objects into the BEV, this paper introduces the Flatmobile representation. This technique segments the footprint of vehicles in the camera view, respecting the flat-world hypothesis implied by the ho-

mography(Figure 2.2), thus preventing the stretching effect. They use 3D bounding box information to make the ground truth Occupancy Grid Map(OGM) for the vehicles.

A two-stage network is proposed for end-to-end trajectory planning. The first stage predicts occupancy grids as semantic masks in the camera view, which are then warped into BEV using homography. For the second stage, the network integrates these BEV occupancy grids into an encoder-decoder LSTM for trajectory planning, using past trajectories and the destination position of the ego vehicle.

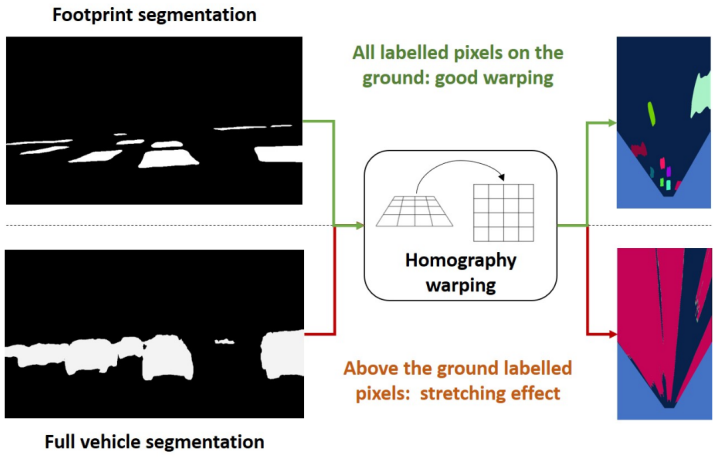


Figure 2.2: Footprint segmentation respects the flat world hypothesis implied by the homography, avoiding the distortion caused by the pixels located above ground [2]

Even though the proposed method outperforms existing models in both accuracy and interpretability, results produced by this method are not reliable enough for deployment.

2 Indian Driving Dataset

This IIIT-H curated dataset is the largest dataset available for Indian Driving conditions. The various sub-datasets are IDD, IDD Detection, Missing Traffic Signs Video Dataset, Fine-grained Vehicle Detection, IDD-AWA(adverse Weather), IDD 3D, IDD Temporal, and IDD Multimodal dataset(a subset of IDD 3D).

IDD-3D: Indian Driving Dataset for 3D Unstructured Road Scenes[5]

The IDD-3D comprises of multi-modal data collected from high-quality LiDAR sensors and multiple cameras, resulting in 12,000 annotated LiDAR frames. It includes high-quality annotations for 3D object bounding boxes with 9 degrees of freedom (DoF) and instance IDs to enable tracking.
The IDD Multimodal is a subset of IDD-3D and has Stereo images, LIDAR data, and On-Board-Diagnostics(OBD) data. Both the datasets have the drawback that it does not have semantic annotations to the LIDAR points and can only be used for benchmarking 3D object detection and tracking.

IDD: A Dataset for Exploring Problems of Autonomous Navigation in Unconstrained Environments[6]

IDD consists of 10,004 finely annotated images collected from 182 drive sequences on Indian roads. The dataset includes 34 classes, reflecting a wider range of traffic participants and road conditions compared to other benchmarks like Cityscapes.
This dataset has the drawback that it can only be used for Semantic Segmentation. Work has not been done for Road Segmentation on the Indian Driving Dataset. For this, the 3D data with semantic annotations must be created.

Semantic Segmentation Datasets for Resource Constrained Training [10]

The IDD dataset is huge and training it requires large computational resources even for Binary Segmentation. The authors introduce two variants of the India Driving Dataset (IDD), named IDD-mini and IDD-lite, designed for efficient training on low-resource hardware. Models trained on these datasets were deployed on Raspberry Pi, achieving satisfactory inference times and accuracy, thereby proving their feasibility for real-time applications in constrained environments.

3 Deep-Learning for Real-Time applications

As the usefulness of Road Segmentation requires accurate, reliable systems with low latency, Deep Learning Strategies for real time applications were studied.

A survey of methods for low-power deep learning and computer vision[11]

This paper examines strategies to optimize deep neural networks (DNNs) for deployment on low-power devices. Given the high computational and energy demands of accurate DNNs, these optimizations are crucial for applications on mobile and embedded systems, which are often constrained by limited power and computational resources.

Key Techniques Discussed:

1. **Parameter Quantization and Pruning:**
    - Quantization reduces the bit-width of DNN parameters, thus lowering memory and computation costs without significantly affecting accuracy. Techniques such as fixed-point and binary quantization are highlighted.
    - Pruning eliminates redundant parameters and connections, reducing model size and computational overhead. This involves identifying and removing unimportant weights, often using importance scores or sensitivity analysis.
2. **Compressed Convolutional Filters and Matrix Factorization:**
    - Filter Compression replaces large convolutional filters with smaller, more efficient ones, like 1x1 convolutions, which maintain performance while reducing complexity.
    - Matrix Factorization decomposes large layers into smaller matrices, streamlining operations and minimizing redundant computations.
3. **Network Architecture Search (NAS):** NAS employs automated techniques to explore and identify optimal network architectures tailored for specific performance and efficiency requirements. This includes using reinforcement learning

to iteratively improve the design based on predefined criteria like accuracy and latency.

4. **Knowledge Distillation:** This process trains a smaller, more efficient network (student) to replicate the performance of a larger, more complex model (teacher). The student network learns from the teacher's predictions and activations, achieving similar accuracy with fewer resources.

Advantages and Challenges:

- **Advantages:** These techniques collectively enable significant reductions in energy consumption and computational requirements while maintaining or slightly reducing accuracy. They facilitate the deployment of sophisticated DNNs on low-power devices, enabling applications such as real-time object detection on drones or mobile devices.

- **Challenges:** Implementing these techniques can be complex, often requiring custom hardware or specialized data structures. Additionally, some methods, like NAS, can be computationally intensive and expensive to train.

The paper concludes by suggesting future research directions and evaluation metrics to further advance the development of low-power DNNs. By focusing on metrics beyond accuracy, such as energy consumption and memory usage, researchers can better address the practical constraints of deploying deep learning models in real-world, resource-constrained environments.

The two important efficient networks selected for this Dissertation Phase are MobileNet-v2 and ENet.

MobileNet-v2: Inverted Residuals and Linear Bottlenecks [12]

The architecture is built upon two main innovations: inverted residuals and linear bottlenecks. The architecture can be seen in Figure 2.3(a) and Figure 2.4(a). The distinct architectural features are discussed below:

- **Inverted Residuals:** Traditional residual connections are reversed; shortcuts connect narrow bottleneck layers instead of wider ones. This design allows for more efficient memory usage and better gradient propagation during training.
- **Linear Bottlenecks:** The architecture includes linear bottleneck layers to prevent non-linearities from destroying information in the low-dimensional subspace. This approach maintains the network's representational power while keeping the architecture simple and efficient.
- **Depthwise Separable Convolutions:** MobileNetV2 uses depthwise separable convolutions to reduce computational complexity and improve efficiency. This technique involves splitting the convolution operation into a depthwise convolution followed by a pointwise convolution, significantly reducing the number of operations required.

ENet: A Deep Neural Network Architecture for Real-Time Semantic Segmentation[13]

Architecture details of the model are shown in Figure 2.3(b) and 2.4(b) and 2.4(c). The ENet uses several innovative techniques to be an effective lightweight segmentation model:

- Early downsampling on smaller volumes of feature maps.
- Bias not computed
- Asymmetric filters to increase the receptive area with fewer parameters.
- Dilated filters in contrast to downsampling.
- Unpooling with saved maxpool indices

4 Technology for the deployment of Deep Learning models in production phase

Technology related to deploying Deep Learning models were researched, the technology selected for this dissertation phase is described below:

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

(a) MobileNet-v2 Layers [12]

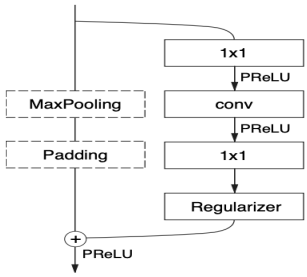
Name	Type	Output size
initial		$16 \times 256 \times 256$
bottleneck1.0	downsampling	$64 \times 128 \times 128$
$4\times$ bottleneck1.x		$64 \times 128 \times 128$
bottleneck2.0	downsampling	$128 \times 64 \times 64$
bottleneck2.1		$128 \times 64 \times 64$
bottleneck2.2	dilated 2	$128 \times 64 \times 64$
bottleneck2.3	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.4	dilated 4	$128 \times 64 \times 64$
bottleneck2.5		$128 \times 64 \times 64$
bottleneck2.6	dilated 8	$128 \times 64 \times 64$
bottleneck2.7	asymmetric 5	$128 \times 64 \times 64$
bottleneck2.8	dilated 16	$128 \times 64 \times 64$
Repeat section 2, without bottleneck2.0		
bottleneck4.0	upsampling	$64 \times 128 \times 128$
bottleneck4.1		$64 \times 128 \times 128$
bottleneck4.2		$64 \times 128 \times 128$
bottleneck5.0	upsampling	$16 \times 256 \times 256$
bottleneck5.1		$16 \times 256 \times 256$
fullconv		$C \times 512 \times 512$

(b) ENet Layers [13]

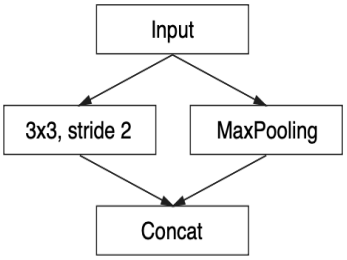
Figure 2.3: MobileNet-v2 and ENet Architecture

Input	Operator	Output
$h \times w \times k$	1x1 conv2d, ReLU6	$h \times w \times (tk$
$h \times w \times tk$	3x3 dwise s=s, ReLU6	$\frac{h}{s} \times \frac{w}{s} \times (tk$
$\frac{h}{s} \times \frac{w}{s} \times tk$	linear 1x1 conv2d	$\frac{h}{s} \times \frac{w}{s} \times k'$

(a) MobileNet-v2 bottleneck module [12]



(b) ENet bottleneck module [13]



(b) ENet initialisation module [13]

Figure 2.4: MobileNet-v2, ENet bottleneck modules

ONNX

ONNX (Open Neural Network Exchange) is a widely adopted open-source format designed for the deployment of deep learning models in production. It provides a standardized framework for representing machine learning models, enabling interoperability between various deep learning frameworks. ONNX facilitates seamless model exchange and optimizes runtime environments, ensuring efficient execution across diverse hardware platforms, including CPUs, GPUs, and specialized accelerators.

TensorRT

TensorRT, developed by NVIDIA, is a high-performance deep learning inference library designed for deploying deep learning models in production. It optimizes trained neural

networks for real-time inference, delivering low-latency and high-throughput performance on NVIDIA GPUs.

Key features of TensorRT include layer fusion, precision calibration (FP16 and INT8), kernel auto-tuning, and dynamic tensor memory management. These optimizations significantly reduce inference time and resource consumption while maintaining model accuracy.

Chapter 3

Experiments and Result Analysis

1 Methodology

All experiments conducted for this dissertation work is aligned with the methodology described in Figure 3.1:

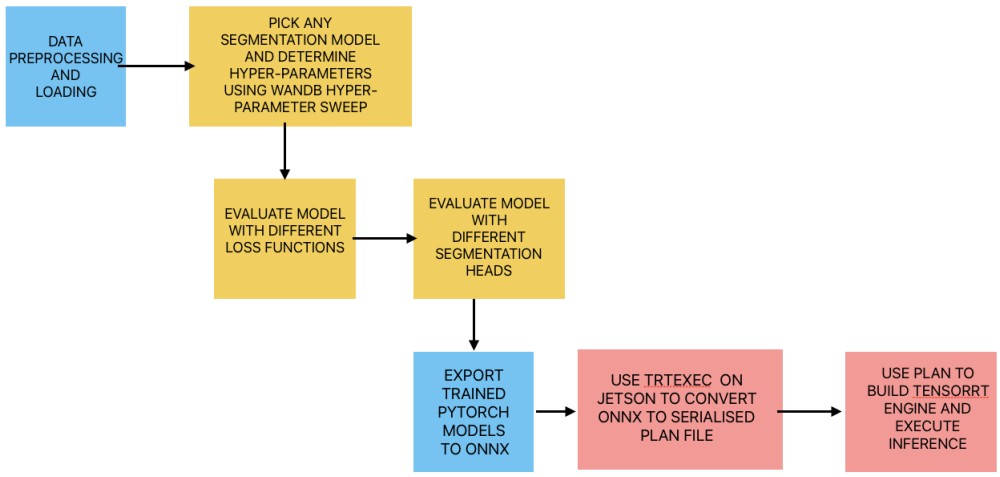


Figure 3.1: Process from data prepocessing to inference

2 Dataset Specification

The IDD-lite was downloaded from Indian Driving Dataset's official page [14].

- Train set: 1403 images, Validation set: 204 images.

- Image size: 320x227
- Classes(7): Drivable, Non-drivable, Living things, Vehicles, Road-side objects, Far-objects, Sky
- The dataset's mean is [0.2707, 0.2828, 0.2797] and the standard deviation is [0.2574, 0.2686, 0.2885].

Figure 3.2: IDD dataset: Image and ground truth (grayscale image with each gray value representing a target class)

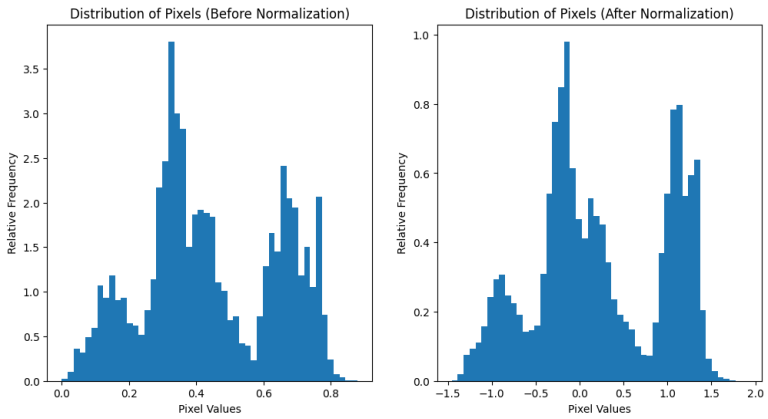


Figure 3.3: Analysing the effect of torchvision.transforms.ToTensor()(left image), torchvision.transforms.ToTensor() and torchvision.transforms.Normalize(mean, standard deviation) (right image)operations on an image from IDD-lite

3 Environment details

For Training Phase

- **Framework: PyTorch Lightning[15]**

  PyTorch lightning framework was selected for its modular design principles and minimalistic syntax, which helps to replace the boilerplate code of PyTorch and avoid errors. It can be compared to the Keras wrapper on Tensorflow.

  The additional advantage of its modular and organized structure is its breakthrough trainer function, akin to the trainer used in distributed training workloads, which allows the code to scale easily from single GPU, multi-GPU, and multi-node workloads.

- **Logger: Weights & Biases(WandB)[7]**

  There are a variety of deep learning experiment tracking APIs available like MLflow, TensorBoard, DVC, CometML, DagsHub, etc. Experiment trackers allow us to store experiment data like training and validation trends, gradient information, logits, result plots, etc.

  WandB logger was selected because of its friendly user interface, real-time log tracking, basic CPU, and GPU profiling plots, and its ability to store up to 100 GB of artifacts(like checkpoints, ONNX files, code, datasets, etc).

- **Development Environment: Jupyter notebook in Kaggle**

  Kaggle provides free 30 GPU hours weekly, All training was conducted using Kaggle's Tesla P100 GPU with 16GB RAM. The notebook uses Python version 3.10

For Inference Phase

- **Framework: PyTorch**

  PyTorch was used only inference is required to be carried out.

- **Development Environment: Jupyter notebook in Jetson TX2 Developer kit**

A virtual environment with PyTorch, Torchvision, and PyCuda was set up. These three important libraries needed to be compatible with the flashed Python version 3.6.9. TensorRT and other NVIDIA drivers(CUDA, CUDNN) are installed during the flashing of the module.

The TX2 has 8GB RAM and a total of 30GB storage of which 13 GB will be available after flashing is complete. It costs approximately 45,000 INR[16] and it has been discontinued from 2020. Flashing was carried out using a host PC with Ubuntu 18.04 and NVIDIA SDK manager.

4 Data Preprocessing and Loading

For Training Phase

Segmentation is a compute-intensive task as the batch of ground truth involved is a volume of the image itself. For the training process to engage smoothly, efficient data processing and data loading pipeline is essential. The libraries used in the data loading pipeline are :

- **Albumentations:** for optimized image transformations.
- **Torchvision:** for optimized dataset creation from path, and optimized dataloader conveying the transformed image and ground truth.
- **DataModule from PyTorch Lightning**: This is an additional wrapper object that consists of methods to ensure that in distributed workloads, data splits, loading, and transforms are consistent along all machines. This is passed to the trainer.fit() method along with the PyTorch-Lightning's Model object 'LightningModule'.
- **Multiprocessing:** This library is crucial to ensure all CPU cores available are engaged in the dataloading pipeline. We must utilize all the CPU resources available to shift the data to GPU for model computations and back to the CPU with results efficiently.

The various image augmentations applied using Albumentations on the training set are:

- Padding(SIZE, SIZE, border_mode=0, value=(0, 255, 0))
- HorizontalFlip()
- RandomCrop(SIZE, SIZE)
- One of these with probability, p =0.3
  - OpticalDistortion(p = 0.3)
  - GridDistortion(p = 0.1)
  - PiecewiseAffine(p = 0.3)
- One of these with p =0.3
  - HueSaturationValue(10,15,10)
  - CLAHE(clip_limit=2)
  - RandomBrightnessContrast()
- Normalize(mean, standard deviation)
- ToTensorV2()

The various image augmentations applied using Albumentations on the validation set are:

- Padding(SIZE, SIZE, border_mode=0, value=(0, 255, 0))
- CenterCrop(SIZE, SIZE)
- Normalize(mean, standard deviation)
- ToTensorV2()

Note that Albumentations is an advanced library, and we can call the same transformation composed on both image and mask even though they have different numbers of channels. Albumentations ensure that only spatial transformations(like resizing, and flipping) and no color transformations(like normalization) will be applied to the mask.

For Inference Phase

For data processing in the inference phase, we cannot use the Albumnetation library as it is only compatible with Python 3.8. In its place, we approximate the transformations with Torchvision's transforms module. There was no need for Pytorch Lightning's DataModule for inference workloads.
The transformations applied to test data are :

- To images:
  - Resize((SIZE,SIZE))
  - ToTensor()
  - normalize(mean, standard deviation)
- To masks:
  - Resize((SIZE,SIZE))
  - ToTensor()

5 Hyperparameter tuning to find learning rate

To select an appropriate combination of batch size, image size, and learning rates, a Hyperparameter sweep was set up using WandB. In their YouTube workshop, it was said that random hyperparameters work well for most workloads, better than Bayesian and grid methods.[17].

For our experiment, a sweep was carried out on a UNet model with pretrained MobileNet-v2 backbone, later it was found that this configuration of hyperparameters was transferable to other models without much deterioration in training statistics. The loss function selected was Dice Loss, which will be compared with other loss functions after hyperparameter tuning

Methodology for Setting Up and Running a Sweep with Weights and Biases (WandB)

First, a sweep_config is selected, next we assign a metric to it. Then we initialize the range of the hyperparameters we wish to tune and the interval of tuning and initialize those in a parameter dictionary. We also update the parameter dictionary with fixed parameters. After all parameters for the model are updated in the parameter dictionary, we pass it to the sweep_config's parameter key. Finally, we can declare a sweep using wandb.sweep by passing in the sweep_config and the project name, we assign it to a 'sweep_id' variable. We can use the 'sweep_id', the train function, and specify the number of runs to call the sweep using a wandb.agent.

Sweep Configuration Selected for UNet experiment

The sweep configuration used for tuning 'image_ip_size', 'lr_encoder', 'lr_decoder', 'batch_size' parameters for a Unet model with pretrained MobileNet-v2 backbone is listed in the A

Analysis Of Sweep Results

All the runs were not equally good, some runs had noisy training and validation trends, their predicted masks were not accurate, and the final layer sigmoid(tracked by WandB from logits) histogram revealed bins with more values close to zero than one. The best training trend and corresponding are shown in Figure 3.4. The predictions of the best model can be seen in Figure 3.8(b)

The model with the best metric, IoU = 0.91 was obtained for the run configuration: 'batch_size'=32, 'lr_encoder'=0.016 ,'lr_decoder'=0.001 and 'image_ip_size'=224.

6 Analysing different loss function's performance on the dataset

As learning rate, batch size, and image input size were found, different loss functions were tried to decide on the best choice for Road Segmentation on IDD-lite.

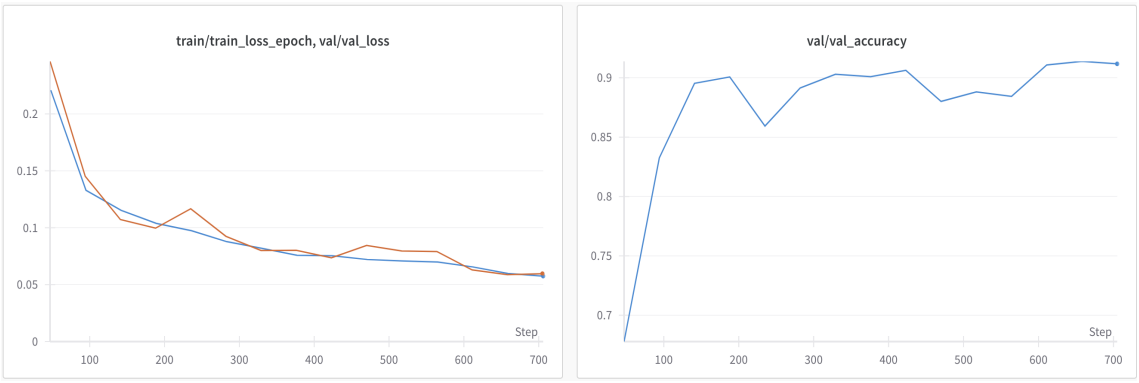


Figure 3.4: The best training trends obtained for the best run from the hyperparameter sweep On UNet with pretrained MobileNet-v2 backbone. Validation loss in red, Training loss in blue(left image)

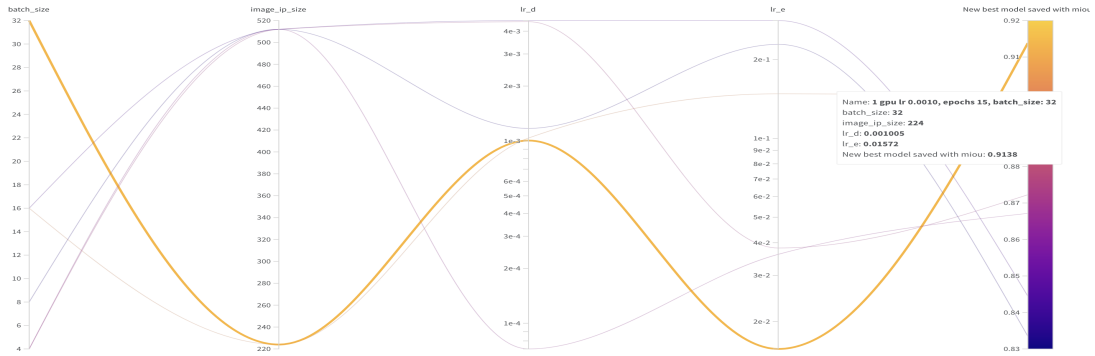


Figure 3.5: Different runs triggered by the Hyperparameter sweep, each spline represents a run, the columns show the different tunable hyperparameters, the final column shows the metric configured to swee_config, The best model is highlighted

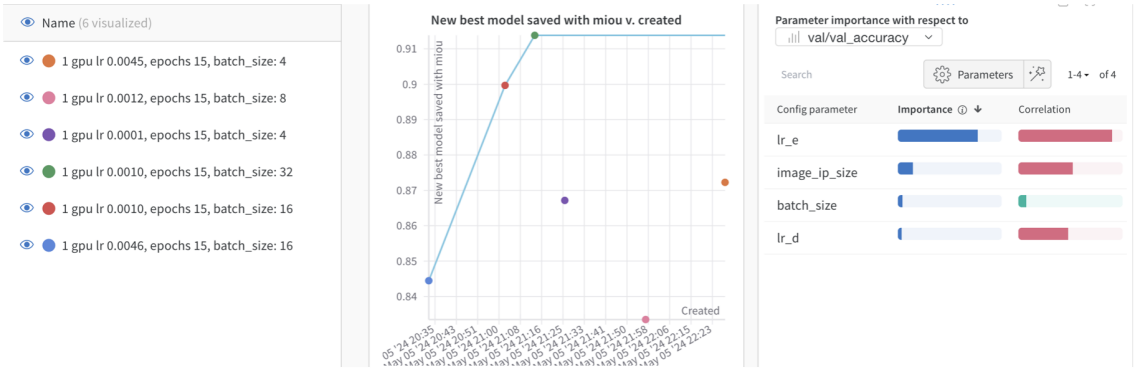


Figure 3.6: The left panel shows the different runs in the sweep, they are color-coded. The middle panel shows a plot corresponding to the run and the best IoU it achieved over 15 epochs. The best IoU model is color-coded with green. The right panel reveals the correlation between hyperparameters and validation accuracy(Iou)

For Binary Segmentation, the available losses in the PyTorch-Segmentation-Models are Soft-Binary-Cross-Entropy Loss(SoftBCE loss), Dice loss, Mathew's Correlation loss(MCC Loss), Lovasz Loss and Focal Loss. Because of the problem of class imbal-

ance, loss is less and accuracy high even when visual predictions are not accurate. All the loss functions gave high test accuracy values, Figure 3.7

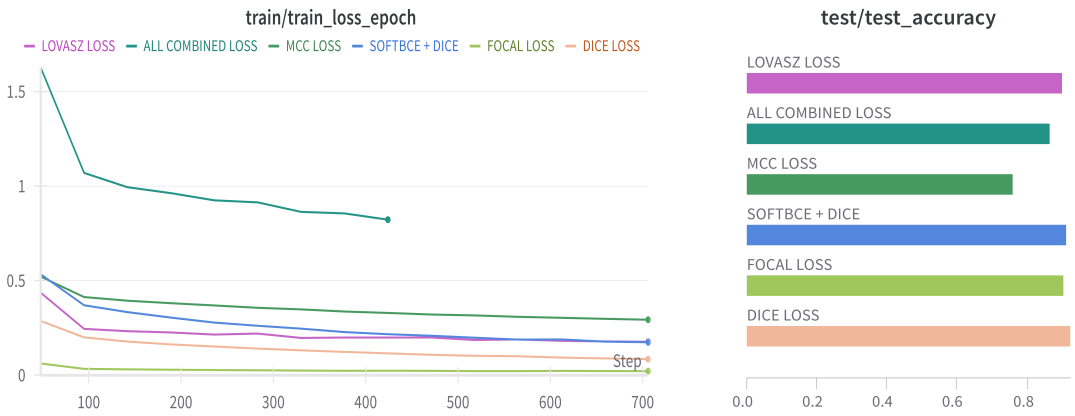
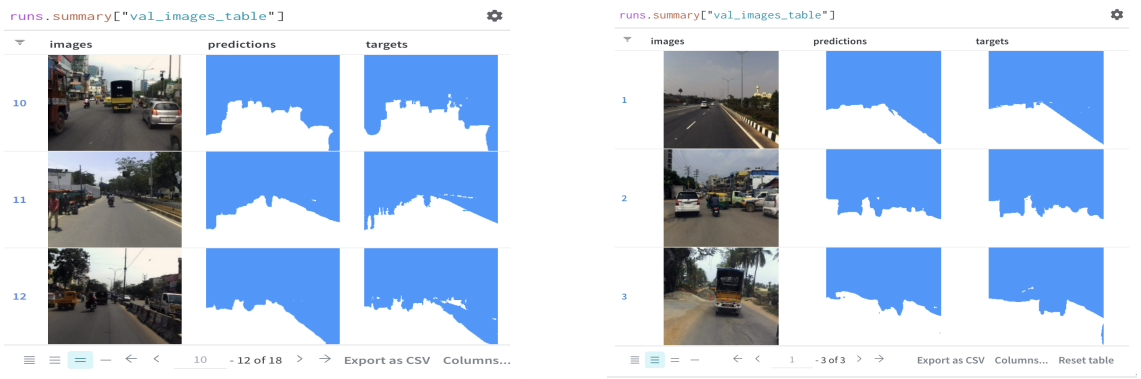


Figure 3.7: Loss function analysis: training trend and test accuracy comparisons

The worst loss for the chosen set of parameters was MCC loss, and the best loss that drops fast and keeps dropping was (Dice + softBCE) loss. It also gave better results than Dice loss alone.Figure 3.8

The combined loss helped to increase training loss, but it didn't do much to improve the accuracy.



(a) BCE+Dice Loss (b) Dice Loss Only

Figure 3.8: Comparison of visual accuracy of Dice loss and combined Dice-SoftBCE loss

7 Trying other Segmentation models with MobileNet-v2 encoder

Predictions of different models

The training trend, test results, predictions, and system statistics were logged using WandB logger to gain exposure to the possible values using the specific configuration of hyperparameters and pretrained MobileNet-v2 encoder.
The other Segmentation heads tried out were:

1. UNet with Skip connections ie. UNet++
2. UNet with Concurrent Spatial Channel Squeeze Excitation ie. UNet_scse
3. Feature Pyramid Network ie. FPN
4. LinkNet
5. Deeplab-v3
6. Path Aggregation Network ie. PANet
7. Pyramid Scene Parsing Network ie.PSPNet
8. Multi-scale Attention Network ie. MANet

The loss trends for different models are shown in Figure 3.9. Some perturbations are present, but they are not very noisy. The validation curve, loss, and accuracy for the Test set are shown in Figure 3.10 and Figure 3.11 respectively. As the test loss and accuracy are similar, we look at the predictions of different models in Figure 3.12 and Figure 3.13. From the predictions, UNet_SCSE and PANet give the most visually accurate results.

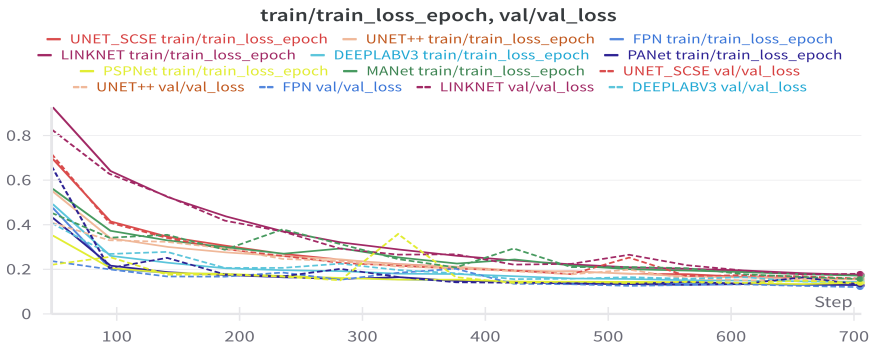


Figure 3.9: Training Trend of MobileNet-v2 with different segmentation heads, Solid: Training curves, Dashed: Validation curves

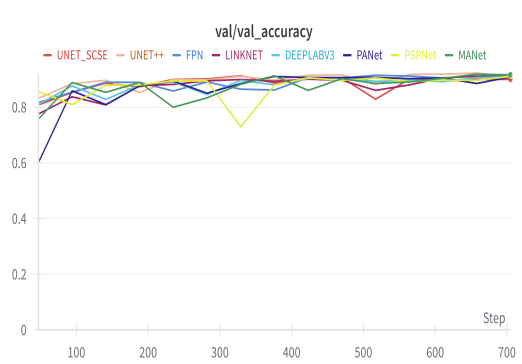


Figure 3.10: Validation accuracy

Figure 3.11: Test loss and Test accuracy

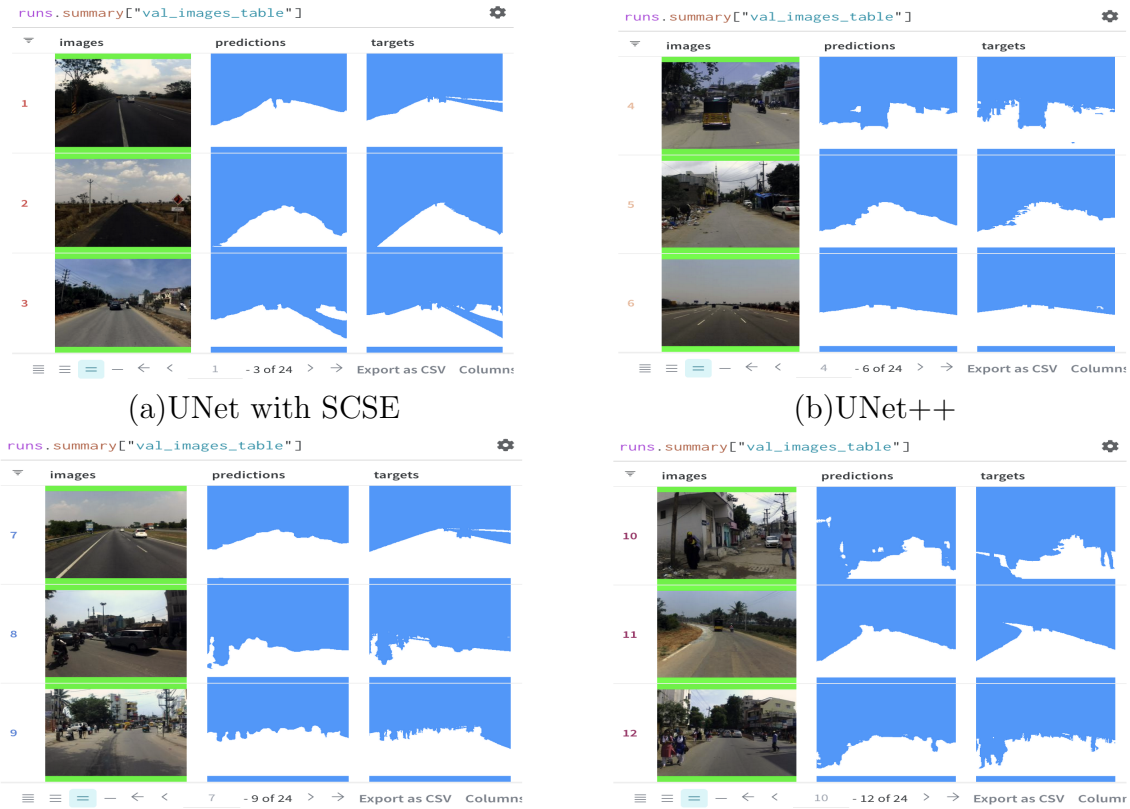
8 Carrying out Inference on Jetson TX2

Details of TensorRT plan creation

TensorRT is the official inference runtime of NVIDIA GPUs. It gets installed on the NVIDIA computing boards when the respective Jetpack for the board is installed. TensorRT allows the implementation of SOTA AI models on embedded systems by using its general-purpose AI compiler to compile the model's serialized plan, the .trt or .engine file.

To create a TensorRT serialized plan we must give TensorRT the weights and computation graph(trace) of our network. Depending on the framework we used for training, the procedure of conversion changes.

For a PyTorch model's trace, we can use the torch.jit directly or export our .pth file to ONNX instead. It is recommended to do ONNX conversion as more types of layers can be traced via ONNX. It must be noted that ONNX conversion of a model is a



(a)UNet with SCSE (b)UNet++

(c) FPN (d) LinkNet

Figure 3.12: Predictions of Different segmentation heads with MobileNet-v2 encoder

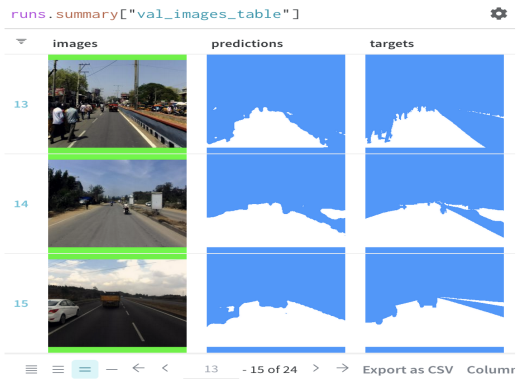
compute-intensive task and it is recommended to be done off-board.

Once the ONNX model is obtained, the easiest way to get the plan is to use the trtexec command on the command line. The trtexec can be found at /usr/src/tensorrt/bin/trtexec.

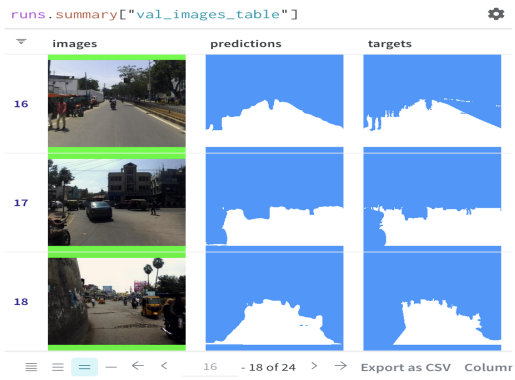
```
1 !/usr/src/tensorrt/bin/trtexec --onnx='pathtoonnx.onnx' --↩
    SaveEngine='pathtoplan.trt' --inputIOFormats=fp16:chw --↩
    outputIOFormats=fp16:chw --fp16 --verbose
2 #use half-precision for input and output, and use channel ↩
    first layout
3 #verbose for creating a log for debugging
```

Note that each plan created is specific to the hardware where the command was executed, so this plan is portable with that specific hardware only.

For our experiment, the ONNX file was not generated for Unet++ and PSPNet, some of their layers did not have ONNX conversion in place. The FPN ONNX model gener-

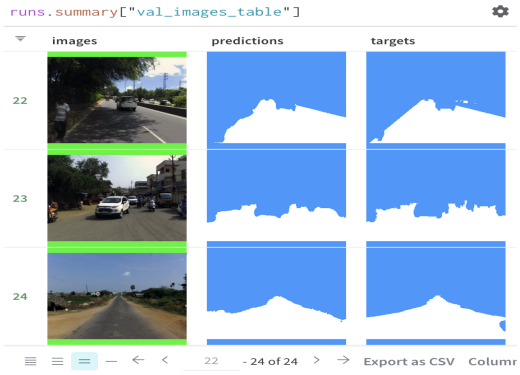


(e) Deeplab-v3



(f) PANet

(e) PSPNet



(f) MANet

Figure 3.13: Predictions of Different segmentation heads with MobileNet-v2 encoder

ated a plan, but it was being read as a null file. Due to time constraints, we proceeded with inference using the remaining five models.

Details of TensorRT inference

A flowchart describing the process from .trt file to inference is shown in Figure 3.14. The code for this procedure was adapted from NVIDIA documentation [18].

When we perform inference using TensorRT it is very important to replicate the exact preprocessing applied to the trained model. For our case, preprocessing approximation from albumentation to torchvision.transforms was not fully successful as can be seen by viewing the predictions in Figure3.16.

However, on applying this prediction for performing inference on video files, it is performing correctly, so the approximations cause only minor visualization errors. We can rectify this error by using torchvision.transforms instead of argumentation during the training phase.

Interestingly, PANet, LinkNet, MANet, and UNet_SCSE are all giving similar latency

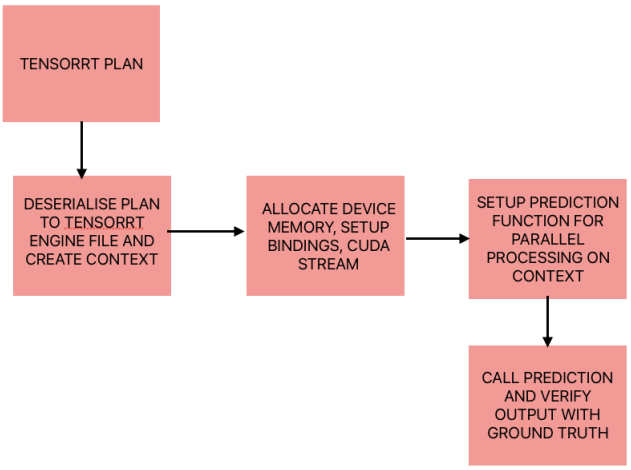
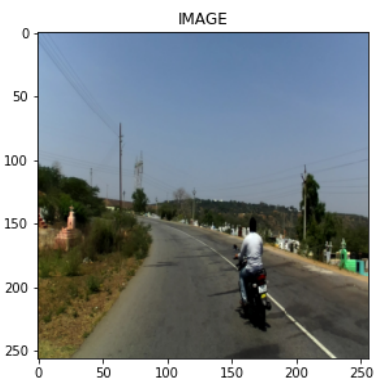
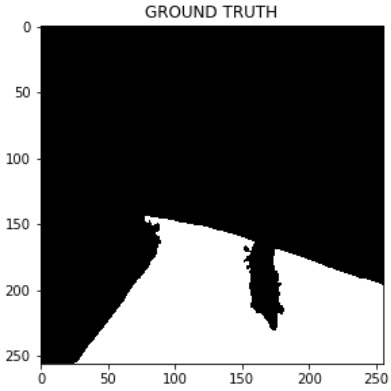


Figure 3.14: TensorRT Inference Procedure



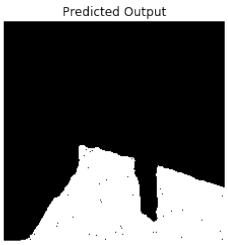
(a) Input Image



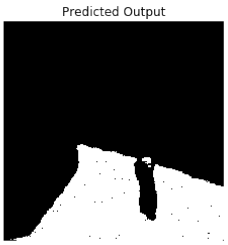
(b) Ground Truth

(c) UNet_SCSE without Thresholding

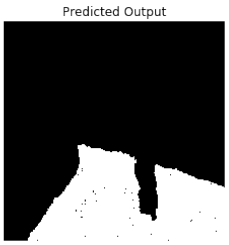
Figure 3.15: Input, Ground Truth, and Unthresholded Prediction



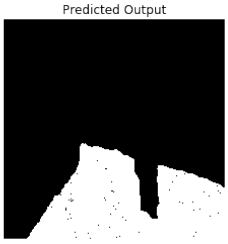
(c) PANet with Thresholding



(d) LinkNet with Thresholding



(e) MANet with Thresholding



(f) Deeplab-v3 with Thresholding

Figure 3.16: Predictions Of Different Models on Jetson TX2

when performing video inference. For a video of one minute in length, it takes approximately three and a quarter minutes to open, process, apply inference, post-process, and write the video file to the disk. And Deeplab-v3 takes four and a half minutes for the same.

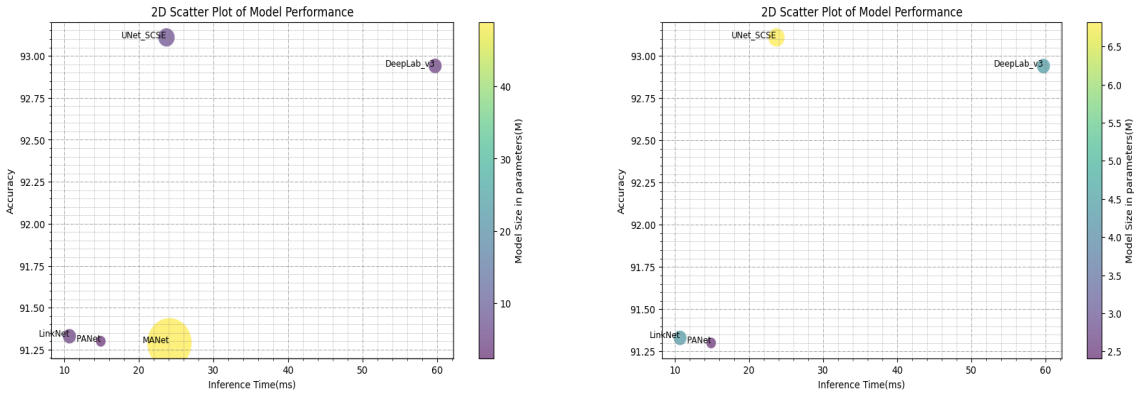


Figure 3.17: Plot showing the accuracy, inference speed, and size of the models

The inference time to predict the mask given an image is visualized with model accuracy and parameter count in Figure 3.17. The time for context creation, warming up of the prediction function, and the inference time after the warm-up is tabulated in 3.1 with the model size parameters.

Table 3.1: Time and Size Summary Of Different Models

Model Name	Context Creation Time (ms)	Warm Up Time (ms)	Image Prediction Time (ms)	Model Size (MB)	Fwd/Bwd Pass size (MB)
UNet_SCSE	38.3	31.6	23.7	28.09	237.6
PANet	45.9	16.8	14.9	10.64	168.33
LinkNet	36.4	12.7	10.7	18.07	190.32
Deeplab-v3	37.7	64.5	59.7	18.30	210.01
MANet	1600	31.5	24.1	196.44	216.80

9 Experimentation of ENet on CamVid lite dataset

It was planned to implement a real-time model, ENet on IDD. However full replication of ENet on baseline dataset CamVid had to be carried out first to test ENet's proper functioning and capabilities.

Environment and Dataset

This was elaborated in detail in Dissertation Phase-1, as a reminder, the CamVid-lite dataset with 12 semantic classes was used, train set, validation, and test set consisting of 367, 101, and 233 images respectively.

The original code was run in 4 TitanX GPU for a maximum of 300 epochs. We tried to replicate the process with Kaggle's P100 GPU with the number of epochs set at 1200.

Hyperparameters

- **Input size:** 360x480
- **Batchsize:** 10
- **Loss:** Cross Entropy loss
- **Activation:** PReLU
- **Dropout:** 0.1 and 0.01
- **Balance weight for each class in loss function:** $w = \frac{1}{ln(c+p_{class})}$, where c = 1.02 to ensure weights lie in the range 0 to 50.
- **Optimizer:** Adam, weight decay: 0.0005

Creating The Models For Phased Training

The original code for ENet from the authors[8] proposes training ENet in two phases:

- **Train the encoder on a downsampled ground truth**: This is implemented by performing point-wise convolution to the encoder output layer so that features get mixed to get volume for the total number of output classes. And train this encoder with Cross-Entropy Loss.
- **Attach the decoder to the encoder and do full model training.** This replication was done by creating the full model definition,(without the encoder convolutional layers), then copying the saved pretrained encoder's weights to the model. The code for this process can be found in B. After the full model is created, train it using Cross-Entropy Loss.

Phased Training

Encoder Training: The WandB log for training trend and validation accuracy is shown in Figure 3.18. We can observe the problem of stagnant validation loss. It is also interesting how, even when the validation loss is hugely stagnant, an upward trend(although limited) in validation accuracy is observed. However, the predictions from this training phase, viewed in Figure 3.20(a) are very poor.

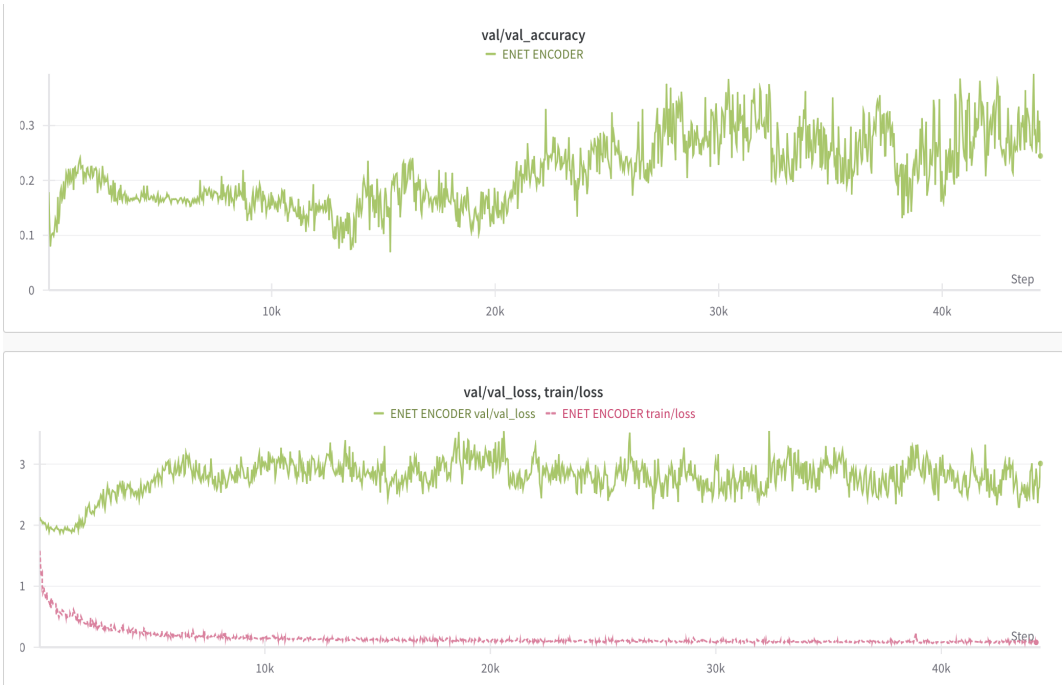
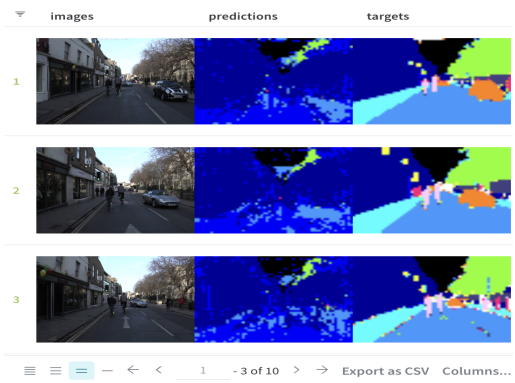


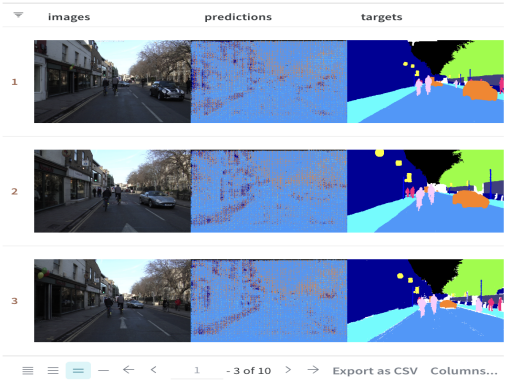
Figure 3.18: ENet Encoder Training Trend For 1200 Epochs

Decoder Attached Full Model Training Training: We continued the training process, the training trend and validation trend for this phase can be seen in Figure 3.19. This time too validation loss is stagnant. The corresponding predictions can be viewed in Figure 3.20(b)

Figure 3.19: ENet-Decoder Attached Training Trend For 1200 Epochs



(a) ENet Encoder Predictions After 1200 Epochs of training



(b) ENet-Decoder Attached Predictions After 1200 Epochs of training

Figure 3.20: ENet Validation Predictions With Regular Batchnorm Layers

Changing Setting of Batchnorm Layers

As the model was not working with the author-defined parameters, a hyperparameter sweep using WandB was tried on the model, but again the same problem of stagnating validation loss was present.

Then we tried out an idea, we put the train image itself into the model, but the model was not segmenting it. Then, it was understood that the problem was not overfitting per se, but that the model was only working in model.train() mode and the source of error came while switching to model.eval() during validation.

The two changes in the model architecture during the model.eval() mode are:

- **Dropout layers:** No dropout is applied, all nodes will be used in the computations.
- **Batchnorm layers:** They will use pre-computed running mean and running standard deviations of the train set for normalization calculation of the validation set.

As the dropout percentage was less, it was unlikely that activation of that percentage of nodes would distort the results. So it was ruled out. Batch normalization on the other hand is computed differently in distributed training.

Difference Between Calculations Of Batch Normalization In a Single GPU and Multiple GPUs

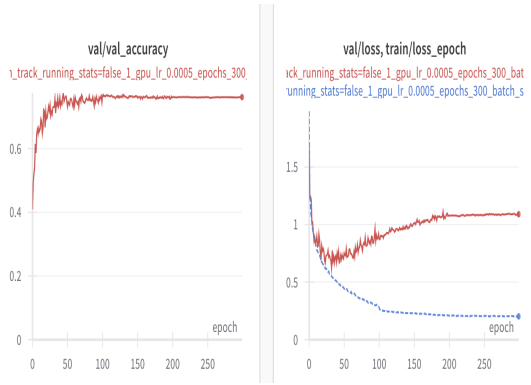
As our batch size is 10, we will explain with that example. In the case of a single GPU,

the mean for that mini-batch will be used for the calculation of the running mean. When we are distributing our batch to 4 GPUs, the split will be 3, 3, 2, 2, in this case, the mean for each split will be calculated and with the best practice of syncing batch normalization, they will be averaged to a global mean, and that global mean will be used in the running mean calculation. The same applies to standard deviation.

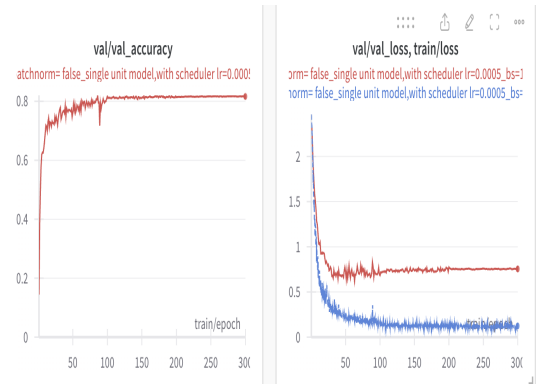
Turning Running Statistics OFF in Batch Normalization layers

We can make our batch normalization layer use the batch statistics of the mini-batch at hand, ie. instead of using the pre-computed running mean and running standard deviation of the train set for all validation mini-batches, use the mean and standard deviation of each validation mini-batch for its normalization calculation. We do this by not calculating the running statistics for the train set. So that during training and validation, batch normalization will be done using the current batch statistics. We just have to pass in the argument 'track_running_stat'=False to nn.Batchnorm(num_channels) to accomplish this.

The training trend after setting this mode to batch norm layers of the network is shown in Figure 3.21. The predictions of the new encoder and decoder attached encoder are shown in Figure 3.22., giving an accuracy of mIoU 66.87 %, surpassing the author's reported mIoU of 51.30 %. The predictions for the test set using this setting are summarised in Figure 3.23.

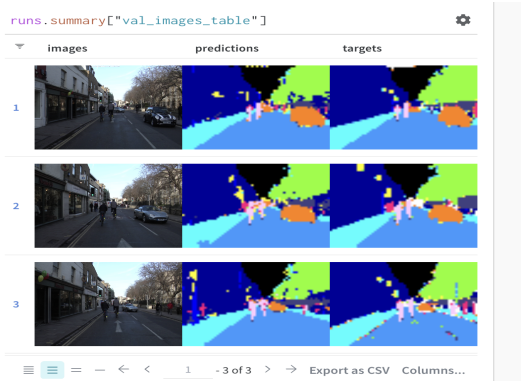


(a) ENet encoder validation loss(Red) decreasing, then overfitting

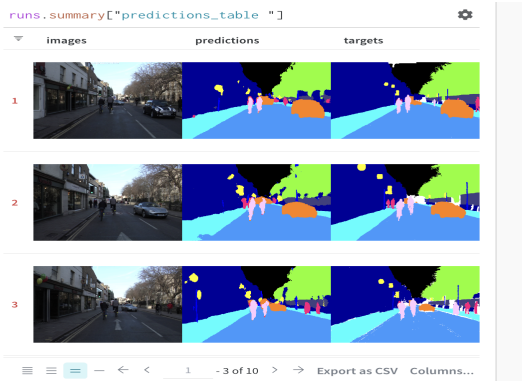


(b) ENet Decoder Attached Full Model, Validation Loss Decreasing With Training Loss

Figure 3.21: Improved Training Trends after switching to 'track_running_stats'=False Batchnorm Layers



(a) ENet Encoder Predictions After 300 epochs



(b) ENet Decoder Attacched Predictions After 300 epochs

Figure 3.22: ENet Validation Predictions With 'track_running_stats'=False Batchnorm Layers

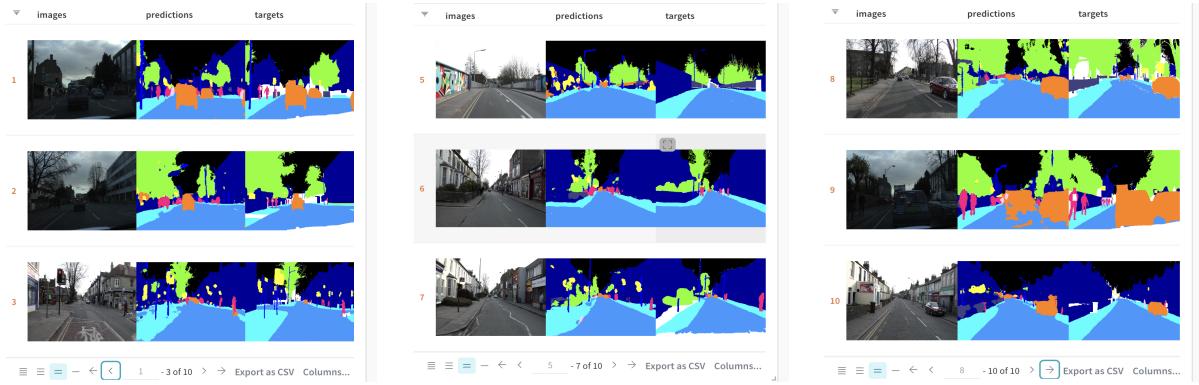


Figure 3.23: ENet Test Predictions With 'track_running_stats'=False Batchnorm Layers

ENet on Jetson TX2

ENet has efficient architecture and it was able to run on Jetson TX2 without any optimization using TensorRT. The result of Inference from the Camvid Dataset is shown in Figure 3.24. The inference time obtained was 61.2 ms. The inference time and model size summary can be found in Table 3.2

Table 3.2: Time and Size Summary Of ENet

Model Name	No: of Classes	Image Prediction Time (ms)	No: of Parameters(K)	Model Size (MB)	Fwd/Bwd Pass size (MB)
ENet	12	61.2	371.81	3.56	468.98

Limitations of The Trained ENet

Very poor results were obtained on using this model to perform inference on a video file. From this, we can observe that by setting 'track_running_stats'=False we get noisy and unreliable predictions for small batch sizes(which for inference is one), especially

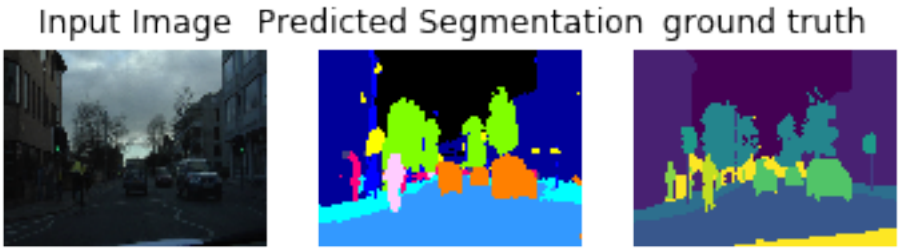
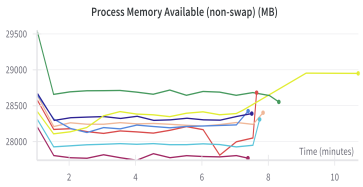


Figure 3.24: Inference of ENet on JetsonTX2

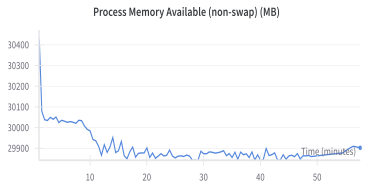
for new data. Also latency of video processing needs to be improved, the total time it took to read the frames from a one-minute length video and write the processed colormap to the disk was four and a half minutes.

10 System metrics interpretation

Eighteen system statistics were logged during the model training phase, but the focus is on metrics that are especially relevant to embedded systems. The key system metrics tracked while trying different segmentation models (denoted with SMP) on P100 GPU and those tracked during ENet-encoder, ENet-full model training are shown in the following figures.

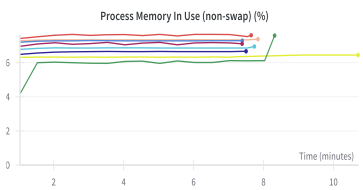


(a) Free Process Memory Available For The SMP Models

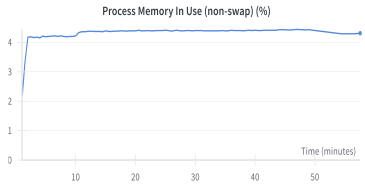


(b) Free Process Memory Available To ENet Encoder Model

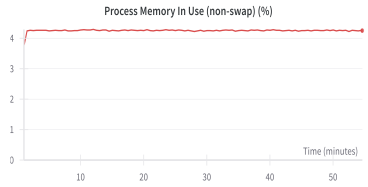
(c) Free Process Memory Available To ENet Full Model



(a) Process Memory in Use by The SMP Models

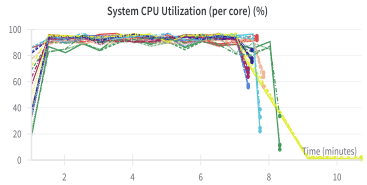


(b) Process Memory in Use by The ENet Encoder Model

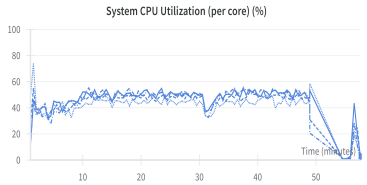


(c) Process Memory in Use by The ENet Full Model

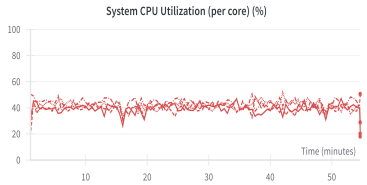
Figure 3.25: Process Memory Available and Process Memory Getting Used by Different models



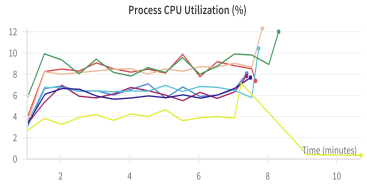
(a) Total System CPU utilization of The SMP Models



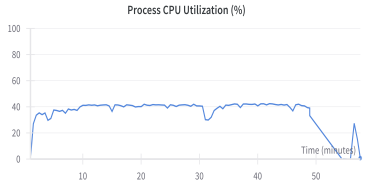
(b) Total System CPU utilization of The ENet Encoder Model



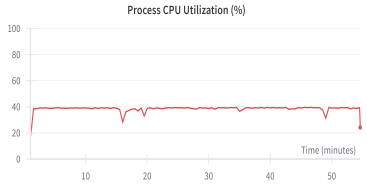
(c) Total System CPU utilization of The ENet Full Model



(A) CPU Utilization by the Processes of The SMP Models

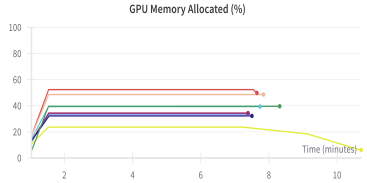


(B) CPU Utilization by the Processes of The ENet Encoder Model

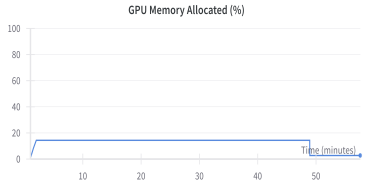


(C) CPU Utilization by the Processes of The ENet Full Model

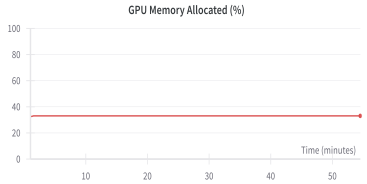
Figure 3.26: System CPU utilization and Process CPU utilization of Different Models



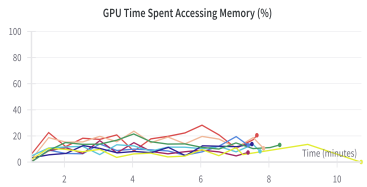
(a) GPU Memory Allocation of The SMP Models



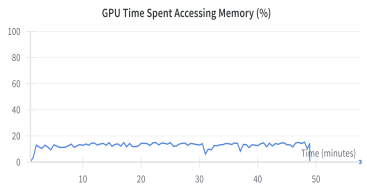
(b) GPU Memory Allocation of The ENet Encoder Model



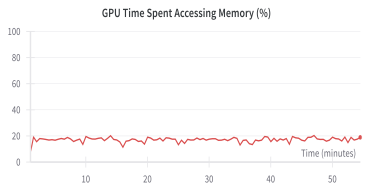
(c) GPU Memory Allocation of The ENet Full Model



(A) GPU Memory Accessing Time of The SMP Models



(B) GPU Memory Accessing Time of The ENet Encoder Model



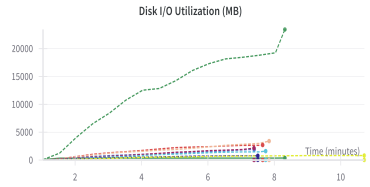
(C) GPU Memory Accessing Time of The ENet Full Model

Figure 3.27: GPU Memory Allocated and GPU Memory Accessing Time of Different Models

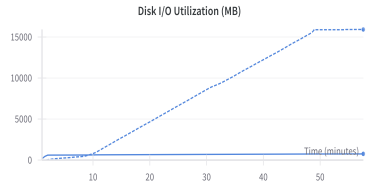
Interpretation for the metrics in the figures are as follows:

1. **GPU Memory Allocation and GPU Utilization(%):** To ensure we don't get out of memory errors and GPU is used to maximum capacity.

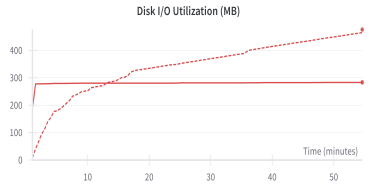
2. **CPU Utilization per core and Process CPU Utilization (%):** When our CPU Utilization of Process is high along with low Total CPU utilization, it shows



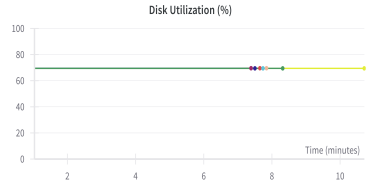
(a) Disk I/O utilization of The SMP Models



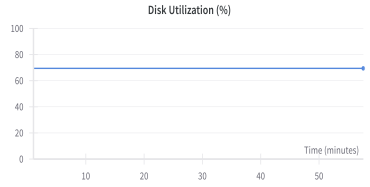
(b) Disk I/O utilization of The ENet Encoder Model



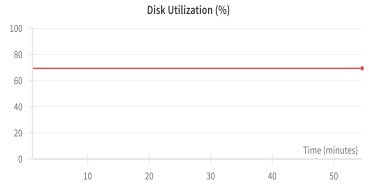
(c) Disk I/O utilization of The ENet Full Model



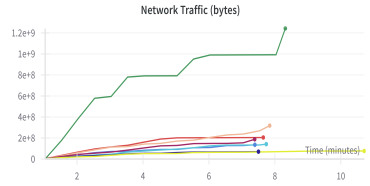
(A) Disk Utilization Percentage of The SMP Models



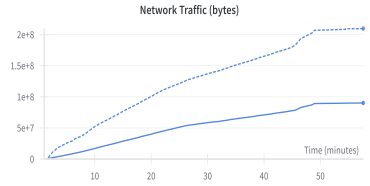
(B) Disk Utilization Percentage of The ENet Encoder Model



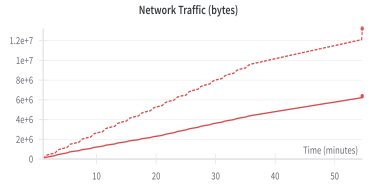
(C) Disk Utilization Percentage of The ENet Full Model



(I) Network Traffic of The SMP Models

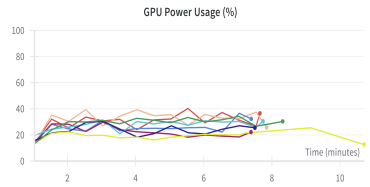


(II) Network Traffic of The ENet Encoder Model

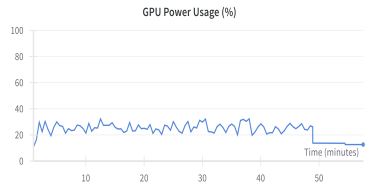


(III) Network Traffic of The ENet Full Model

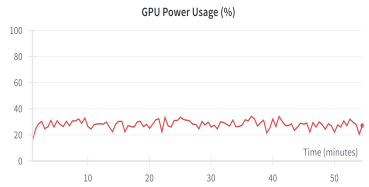
Figure 3.28: Disk I/O Utilization(MB), Disk Utilization Percentage, and Network Traffic of Different Models



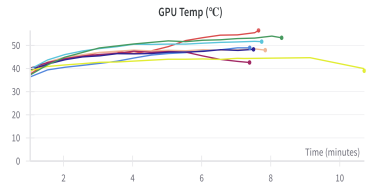
(a) GPU Power Use Percentage of The SMP Models



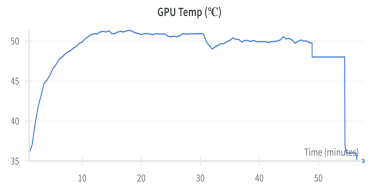
(b) GPU Power Use Percentage of The ENet Encoder Model



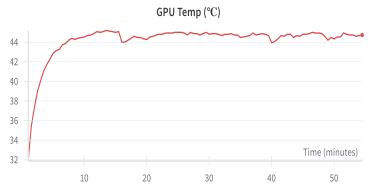
(c) GPU Power Use Percentage of The ENet Full Model



(A) GPU Temperature of The SMP Models



(B) GPU Temperature of The ENet Encoder Model



(C) GPU Temperature of The ENet Full Model

Figure 3.29: Power Use Percentage and GPU Temperature of Different Models

that our program is computationally intensive and is compute bound. We can see how ENet, the real-time model, is compute bound by comparing these metrics with other models in Figure 3.26.

3. **Process Memory available and Process Memory in use (MB):** In Figure 3.27 we see that models like PSP-Net and ENet are handling operations such that they continue to have RAM available for computation; their design architecture may be allowing this.

4. **Disk I/O Utilization(MB), Disk Utilization (%), and Network Traffic(bytes):** From Figure 3.28 we see that the data processed by ENet is significantly larger than other models. This is because ENet is working on Semantic Segmentation with higher resolution images (12 class segmentation, 360x480 picture resolution).

5. **GPU Temperature and GPU Power Usage(%) of The Models:** We must ensure the temperature and power requirements of the hardware are acceptable. We see in Figure 3.29 that temperature and power consumption of all models are comparable.

Chapter 4

Conclusion

In this dissertation, we explored the application of deep learning techniques for Road Segmentation in the Indian context using the IDD-lite.

WandB logger was used to systematically track experiments and tune hyperparameters. Different loss functions and semantic segmentation model architectures were implemented using the Segmentation-Models-Pytorch library. Trained models were deployed for inference on JetsonTX2 using TensorRT. Some progress was made in the implementation of a real-time model ENet and its system metrics was studied with other Deep Learning models.

1 Challenges

- **Latency in Video-Processing:** The open-CV pipeline for video processing was showing latency on Jetson TX2.
  NVIDIA has a video analytics toolkit, Deepstream SDK which enables high-throughput video processing without latency. We must use a pipeline creation software, Graph Composer along with Deepstream SDK to efficiently build pipelines on Cloud GPUs and deploy them in embedded systems. We can integrate our trained models in these graphs using DeepStream SDK plugins. But in order to do so, we must write configuration files to create those custom model plugins for ourselves. More research needs to be done to figure out how to write configuration files to produce plugins for custom models.

- **ENet training**: It is expected that ENet will start training properly in a multi-GPU setting. PyTorch lightning allows us to carry out multi-GPU training with just one line of code, but by the time an HPC was available, the errors could not be debugged in time.

2 Future work

- **Creating BEV annotations for IDD-3D :**As highlighted in the literature review, IDD has a shortcoming of Semantic Segmentation and multi-modal dataset being disparate. To make a BEV ground truth for Road Segmentation, we can assign LIDAR points with their semantic label as described in [9]. To do this efficiently, we can use a trained Perspective Road Segmentation Deep Learning model to segment the images in IDD-3D. We can then improve the annotations manually.

- Make Deepstream plugin for model Inference to use in Graph Composer

Appendix A

Hyperparameter Sweep configuration

```
sweep_config = {
    'method': 'random'
    }
metric = {
    'name': 'New best model saved with Iou',
    'goal': 'maximize'
    }
sweep_config['metric'] = metric
parameters_dict = {
    'batch_size':{
        'values':[4,8,16,32]
    },
    'lr_e': {
    'distribution': 'log_uniform_values',
    'min': 5e-3,
    'max': 5e-1
    },
    'lr_d':{
    'distribution': 'log_uniform_values',
    'min': 5e-5,
```

```
21      'max': 5e-3
22      },
23      'image_ip_size':{
24          'values': [224,384,512]
25      }
26  }
27  parameters_dict.update({
28      'epochs':{'value': 15 },
29      'model_name': {'value':'unet'},
30      'encoder_name' :  {'value':'mobilenet_v2'},
31      'encoder_weights' :{'value':'imagenet'},
32  })
33  sweep_config['parameters'] = parameters_dict
34  sweep_id = wandb.sweep(sweep_config, project='multiprocessed ↩
       dataloader,idd_lite_unet_binary_road_segmenation')
35  # wandb.agent(sweep_id, train_using_wandb, count=15)
```

Appendix B

Creating ENet Model From Saved Encoder And Empty Decoder For Combined Training

```
model = ENet(12)# Initialize all layers, encoder and decoder combined model

path_to_checkpoint = '/kaggle/input/enet_camvid_encoder/pytorch/track_running_status_batch_norm_equals_false/1/CNNEncoder_for_ENet_trained_on_Camvid_epoch44_acc0.774.pth'
checkpoint = torch.load(path_to_checkpoint)
state_dict = checkpoint['state_dict']

new_state_dict = { k:v for k, v in state_dict.items() if k in model.state_dict()}
model.load_state_dict(new_state_dict, strict = False)
```

References

[1] Y. Ma, T. Wang, X. Bai, H. Yang, Y. Hou, Y. Wang, Y. Qiao, R. Yang, D. Manocha, and X. Zhu, "Vision-centric bev perception: A survey," *arXiv preprint arXiv:2208.02797*, 2022.

[2] A. Loukkal, Y. Grandvalet, T. Drummond, and Y. Li, "Driving among flatmobiles: Bird-eye-view occupancy grids from a monocular camera for holistic trajectory planning," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 51–60, 2021.

[3] "Indian ai and robotics startup swaayaatt robotics secures $4 million to advance level 5 autonomy." `https://analyticsindiamag.com/indian-ai-and-robotics-startup-swaayatt-robots-secures-4-mn-to-advance-level-5-autonomy/`.

[4] D. Unger, N. Gosala, V. R. Kumar, S. Borse, A. Valada, and S. Yogamani, "Multi-camera bird's eye view perception for autonomous driving," *Computer Vision: Challenges, Trends, and Opportunities*, p. 279, 2024.

[5] S. Dokania, A. Hafez, A. Subramanian, M. Chandraker, and C. Jawahar, "Idd-3d: Indian driving dataset for 3d unstructured road scenes," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 4482–4491, 2023.

[6] G. Varma, A. Subramanian, A. Namboodiri, M. Chandraker, and C. Jawahar, "Idd: A dataset for exploring problems of autonomous navigation in unconstrained environments," in *2019 IEEE winter conference on applications of computer vision (WACV)*, pp. 1743–1751, IEEE, 2019.

[7] "Weights and biases." `https://wandb.ai/site`.

[8] "Enet training, e-lab." https://github.com/e-lab/ENet-training, 2018.

[9] C. Lu, M. J. G. Van De Molengraft, and G. Dubbelman, "Monocular semantic occupancy grid mapping with convolutional variational encoder–decoder networks," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 445–452, 2019.

[10] A. Mishra, S. Kumar, T. Kalluri, G. Varma, A. Subramaian, M. Chandraker, and C. Jawahar, "Semantic segmentation datasets for resource constrained training," in *Computer Vision, Pattern Recognition, Image Processing, and Graphics: 7th National Conference, NCVPRIPG 2019, Hubballi, India, December 22–24, 2019, Revised Selected Papers 7*, pp. 450–459, Springer, 2020.

[11] A. Goel, C. Tung, Y.-H. Lu, and G. K. Thiruvathukal, "A survey of methods for low-power deep learning and computer vision," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pp. 1–6, IEEE, 2020.

[12] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[13] A. Paszke, A. Chaurasia, S. Kim, and E. Culurciello, "Enet: A deep neural network architecture for real-time semantic segmentation," *arXiv preprint arXiv:1606.02147*, 2016.

[14] "India driving dataset." https://idd.insaan.iiit.ac.in/.

[15] "Pytorch lightning." https://lightning.ai/docs/pytorch/stable/.

[16] https://robu.in/product/nvidia-jetson-tx2-development-kit/.

[17] "Tune hyperparameters easily with w and b sweeps." https://www.youtube.com/watch?v=9zrmUIlScdY.

[18] https://github.com/NVIDIA/TensorRT/blob/main/quickstart/IntroNotebooks/4.%20Using%20PyTorch%20through%20ONNX.ipynb.

RE-2022-306237 - Turnitin Plagiarism Report

by Daya Alex

Submission date: 17-Jun-2024 05:07PM (UTC+0700)
Submission ID: 271718638795
File name: RE-2022-306237.docx (3.97M)
Word count: 7290
Character count: 37995

RE-2022-306237-plag-report

ORIGINALITY REPORT

8%	5%	5%	1%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	upcommons.upc.edu Internet Source	1%
2	d-nb.info Internet Source	1%
3	Chenyang Lu, Marinus Jacobus Gerardus van de Molengraft, Gijs Dubbelman. "Monocular Semantic Occupancy Grid Mapping with Convolutional Variational Encoder-Decoder Networks", IEEE Robotics and Automation Letters, 2019 Publication	1%
4	arxiv.org Internet Source	1%
5	export.arxiv.org Internet Source	1%
6	Shubham Dokania, A. H. Abdul Hafez, Anbumani Subramanian, Manmohan Chandraker, C.V. Jawahar. "IDD-3D: Indian Driving Dataset for 3D Unstructured Road	<1%

Scenes", 2023 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV), 2023
Publication

7	Abdelhak Loukkal, Yves Grandvalet, Tom Drummond, You Li. "Driving among Flatmobiles: Bird-Eye-View occupancy grids from a monocular camera for holistic trajectory planning", 2021 IEEE Winter Conference on Applications of Computer Vision (WACV), 2021 Publication	<1 %
8	"Computer Vision, Pattern Recognition, Image Processing, and Graphics", Springer Science and Business Media LLC, 2020 Publication	<1 %
9	Yuxuan Wang, Lanxin Zeng, Wen Yang, Jian Kang, Huai Yu, Mihai Datcu, Gui-Song Xia. "Extracting Building Footprints in SAR Images via Distilling Boundary Information from Optical Images", IEEE Transactions on Geoscience and Remote Sensing, 2024 Publication	<1 %
10	medium.com Internet Source	<1 %
11	Submitted to Indian Institute of Technology IIT Palakkad Student Paper	<1 %

12	Islomjon Shukhratov, Andrey Pimenov, Anton Stepanov, Nadezhda Mikhailova, Anna Baldycheva, Andrey Somov. "Optical Detection of Plastic Waste Through Computer Vision", Intelligent Systems with Applications, 2024 Publication	<1 %
13	hal.utc.fr Internet Source	<1 %
14	www.catalyzex.com Internet Source	<1 %
15	yangkailun.com Internet Source	<1 %
16	Submitted to University of Sydney Student Paper	<1 %
17	ichi.pro Internet Source	<1 %
18	deepai.org Internet Source	<1 %
19	readthedocs.org Internet Source	<1 %
20	dblp.org Internet Source	<1 %
21	David Restrepo, Chenwei Wu, Sebastián Andrés Cajas, Luis Filipe Nakayama, Leo	<1 %

Anthony Celi, Diego M López. "Multimodal Deep Learning for Low-Resource Settings: A Vector Embedding Alignment Approach for Healthcare Applications", Cold Spring Harbor Laboratory, 2024
Publication

---

22 Shangwei Guo, Jin Lu, Zhengchao Lai, Jun Li, Shaokun Han. "Depth-Assisted Camera-Based Bird's Eye View Perception for Autonomous Driving", 2023 IEEE 11th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), 2023 <1 %
Publication

---

23 repo.iain-tulungagung.ac.id <1 %
Internet Source

---

24 Guto Leoni Santos, Patricia Takako Endo, Theo Lynn, Djamel Sadok, Judith Kelner. "A reinforcement learning-based approach for availability-aware service function chain placement in large-scale networks", Future Generation Computer Systems, 2022 <1 %
Publication

---

25 Jiayuan Du, Shuai Su, Rui Fan, Qijun Chen. "Chapter 10 Bird’s Eye View Perception forAutonomous Driving", Springer Science and Business Media LLC, 2023 <1 %
Publication

26	Thomas Roddick, Roberto Cipolla. "Predicting Semantic Map Representations From Images Using Pyramid Occupancy Networks", 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020 Publication	<1%
27	Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition", Neural Computation, 1989 Publication	<1%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		